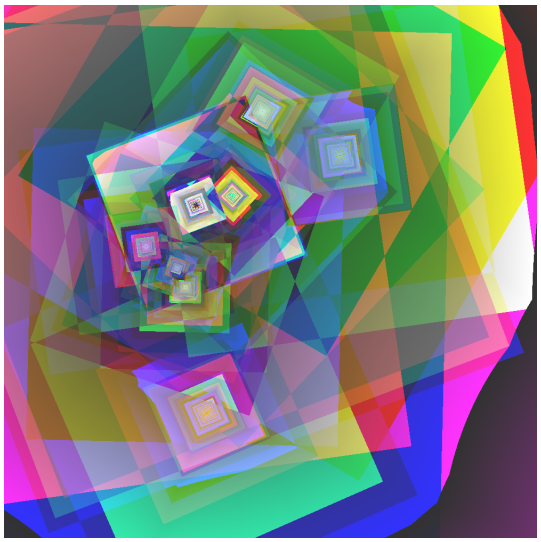
Olimpiada Națională de Informatică 2022

Organizată de către
Societatea pentru Excelență și Performanță în Informatică
în parteneriat cu Ministerul Educației



Tehnoredactare

Tamio-Vesa Nakajima

Copertă

Tamio-Vesa Nakajima

ISBN

978-606-9030-51-6

Materialul acesta este distribuit gratuit! Spor la studiu!

Date de contact

Societatea pentru Excelența si Performanta în Informatică
www.sepi.ro
contact@sepi.ro

Infobits Academi
www.sepi.ro
contact@sepi.ro

Autori

Funcția	Nume și prenume	Instituția
Vicepreședinte Olimpiada Națională	Lica Daniela	Centrul Județean de Excelență Prahova, Ploiești
Cordonator loturi naționale și juniori	Nicoli Marius	Colegiul Național *„Frații Buzești”*, Craiova
Coordonator lot seniori	Panaete Adrian	Colegiul Național *„A.T.Laurian”*, Botoşani
Vicepreședinte secțiunea gimnaziu	Cerchez Emanuela	Colegiul Național *„Emil Racoviță”*, Iaşi
Vicepreședinte secțiunea liceu	Pracsiu Dan	Liceul Teoretic *„Emil Racoviță”*, Vaslui
Coordonator comisia tehnică	Oprița Petru	Liceul Regina Maria, Dorohoi
Coordonator clasa V-a	Pintea Adrian Doru	Inspectoratul Școlar Județean Cluj
Clasa a V-a	Banu Denis Andrei	Facultatea de Informatică, Universitatea *„Alexandru Ioan Cuza”*, Iași
Clasa a V-a	Dabelea Delia	Colegiul Național *„Spiru Haret”*, Târgu Jiu
Clasa a V-a	Iordaiche Eugenia-Cristiana	Liceul Teoretic *„Grigore Moisil”*, Timișoara
Clasa a V-a	Manolache Gheorghe	Colegiul Național de Informatică, Piatra Neamț
Clasa a V-a	Nicoli Marius	Colegiul Național *„Frații Buzești”*, Craiova
Clasa a V-a	Simulescu Adriana	Liceul Teoretic *„Grigore Moisil”*, Timișoara
Clasa a V-a	Șandor Nicoleta Lenuța	Colegiul Național *„Mihai Eminescu”*, Satu Mare
Clasa a V-a	Tîmplaru Roxana Gabriela	Liceul Tehnologic *„Ștefan Odobleja”*, Craiova
Coordonator clasa a VI-a	Pracsiu Dan	Liceul Teoretic *„Emil Racoviță”*, Vaslui
Clasa a VI-a	Balacea Georgeta	Colegiul Național Vasile Alecsandri, Galați
Clasa a VI-a	Boian Flavius	Colegiul Național *„Spiru Haret”*, Târgu Jiu
Clasa a VI-a	Cardaș Cerasela Daniela	Colegiul Național *„A.T.Laurian”*, Botoșani
Clasa a VI-a	Costineanu Raluca	Colegiul Național *„Ștefan cel Mare”*, Suceava

Funcția	Nume și prenume	Instituția
Clasa a VI-a	Dumitrașcu Dan Octavian	Colegiul Național *„Dinicu Golescu”*, Câmpulung Muscel, Argeş
Clasa a VI-a	Frâncu Cristian	Clubul Nerdvana, București
Clasa a VI-a	Grecea Violeta	Colegiul Național de Informatică *„Matei Basarab”*, Râmnicu Vâlcea
Clasa a VI-a	Pintea Rodica	Liceul Teoretic *„Radu Vlădescu”*, Pătârlagele, Buzău
Clasa a VI-a	Pintescu Alina	Colegiul Național *„Gheorghe Șincai”*, Baia Mar
Clasa a VI-a	Rotar Dorin-Mircea	Colegiul Național *„Samuil Vulcan”*, Beiuș
Coordonator clasa a VII-a	Lica Daniela	Centrul Județean de Excelență Prahova
Clasa a VII-a	Ungureanu Florentina	Colegiul Național de Informatică, Piatra-Neamț
Clasa a VII-a	Anton Cristina Elena	Colegiul Național *„Gheorghe Munteanu Murgoci”* / CCD, Brăila
Clasa a VII-a	Burța Alin	Colegiul Național *„B.P. Hașdeu”*, Buzău
Clasa a VII-a	Gorea-Zamfir Claudiu-Cristian	Școala Gimnazială *„Alexandru cel Bun”*, Iași
Clasa a VII-a	Ioniță Alexandru	Facultatea de Informatică, Universitatea *„Alexandru Ioan Cuza”*, Iași
Clasa a VII-a	Nicu Vlad Laurențiu	Liceul Teoretic *„Mihail Kogălniceanu”*, Vaslui
Clasa a VII-a	Popa Bogdan-Ioan	Universitatea București, Facultatea de Matematică și Informatică, București
Clasa a VII-a	Șerban Marinel	Colegiul Național *„Emil Racoviță”*, Iași
Coordonator clasa a VIII-a	Cerchez Emanuela	Colegiul Național *„Emil Racoviță”*, Iaşi
Clasa a VIII-a	Bălașa Filonela	Colegiul Național *„Grigore Moisil”*, București
Clasa a VIII-a	Diac Paul	Facultatea de Informatică, Universitatea *„Alexandru Ioan Cuza”*, Iași
Clasa a VIII-a	Miron Lucia	Colegiul Național *„Costache Negruzzi”*, Iași
Clasa a VIII-a	Moț Nistor	Școala gimnazială *„Dr. Luca”*, Brăila
Clasa a VIII-a	Muntean Radu	ETH, Zurich, Elveţia
Clasa a VIII-a	Pit-Rada Ionel-Vasile	Colegiul Național *„Traian”*, Drobeta Turnu Severin
Clasa a VIII-a	Popa Daniel	Liceul Teoretic *„Aurel Vlaicu”*, Orăștie, Hunedoara
Clasa a VIII-a	Pop Ioan Cristian	Universitatea Politehnica București
Coordonator clasa a IX-a	Eugen Nodea	Colegiul Național *„Tudor Vladimirescu”*, Târgu Jiu
Clasa a IX-a	Arhire Andrei	Universitatea *„Al. I. Cuza”*, Iași
Clasa a IX-a	Candale Silviu	Colegiul Național Liviu Rebreanu, Bistrița
Clasa a IX-a	Chește Ciprian	Liceul Tehnologic *„Grigore C Moisil”*, Buzău
Clasa a IX-a	Cismaru Mihaela	NetRom Software Craiova, Craiova
Clasa a IX-a	Gheorghe Liviu Armand	Universitatea București, Facultatea de Matematică și Informatică

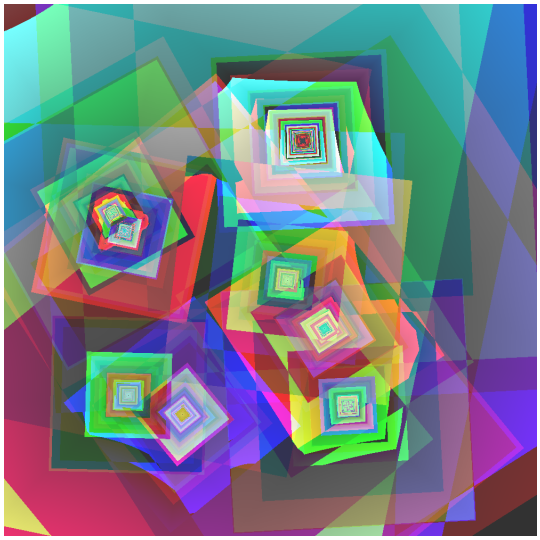
Funcția	Nume și prenume	Instituția
Clasa a IX-a	Iordache Ioan-Bogdan	Universitatea din București, București
Clasa a IX-a	Petrescu Alexandru	University of Oxford
Clasa a IX-a	Popescu Ioan	Universitatea Politehnica București, Facultatea de Automatică și Calculatoare
Clasa a IX-a	Popescu Mihai Cristian	Universitatea Babeș-Bolyai, Cluj-Napoca
Clasa a IX-a	Ciurea Stelian	Universitatea *„Lucian Blaga”*, Sibiu
Coordonator clasa a X-a	Zoltan Szabo	Inspectoratul Școlar Județean Mureș/Colegiul *„Petru Maior”*, Reghin
Clasa a X-a	Boca Alina Gabriela	Colegiul Național de Informatică *„Tudor Vianu”*, București
Clasa a X-a	Coroian David Nicolae	Delft University of Technology, Delft
Clasa a X-a	Cotor Andrei	Universitatea Babeș-Bolyai, Cluj-Napoca
Clasa a X-a	Dăscălescu Ștefan Cosmin	Universitatea București, Piatra Neamț
Clasa a X-a	Popescu Carmen	Colegiul Național *„Gheorghe Lazăr”*, Sibiu
Clasa a X-a	Râpeanu George - Alexandru	Universitatea Babeș-Bolyai, Cluj-Napoca
Clasa a X-a	Tulbă Lecu Theodor Gabriel	Universitatea Politehnica Bucuresti
Coordonator clasele a XI-a şi a XII-a	Panaete Adrian	Colegiul Național *„A.T.Laurian”*, Botoşani
Clasele XI-XII	Bicsi Lucian	Universitatea din Bucuresti, București
Clasele XI-XII	Bunget Mihai	Colegiul Național *„Tudor Vladimirescu”*, Târgu-Jiu
Clasele XI-XII	Chichirim Stelian	Universitatea București, București
Clasele XI-XII	Constantinescu Andrei-Costin	ETH Zurich, Zurich
Clasele XI-XII	George Chichirim	University of Oxford
Clasele XI-XII	Nanu Ruxandra Laura	University of Oxford
Clasele XI-XII	Oncescu Costin-Andrei	University of Oxford
Clasele XI-XII	Popescu Doru Anastasiu	Universitatea din Pitești, Departamentul de Matematică-Informatică
Clasele XI-XII	Sitaru Bogdan	University of Oxford
Clasele XI-XII	Tamio-Vesa Nakajima	University of Oxford
Clasele XI-XII	Tinca Matei	Vrije Universiteit Amsterdam
Clasele XI-XII	Udriștoiu Alexandra Maria	Universitatea București
Membu comisie baraj juniori	Manz Victor	Colegiul Național de Informatică *„Tudor Vianu”*, București
Membu comisie baraj juniori	Voroneanu Radu	Google, Zurich
Membru comisia tehnică	Bolohan Mihai	Liceul Regina Maria, Dorohoi
Membru comisia tehnică	Birta Andrei	Școala Gimnazială *„Mihail Kogălniceanu”*, Dorohoi

Funcția	**Nume și prenume**	**Instituția**
Membru comisia tehnică	Genoiu Nicolae Claudiu	Liceul Teoretic *„Ion Barbu”*, Pitești
Membru comisia tehnică	Iosub Neculai	Colegiul Naţional *„Gheorghe Roșca Codreanu”*, Bârlad
Membru comisia tehnică	Luca Adrian	Colegiul Tehnic *„Ioan C. Ștefănescu”*, Iași
Membru comisia tehnică	Miclea Adrian	Liceul Teoretic *„Eugen Pora”*, Cluj-Napoca
Membru comisia tehnică	Neamțu Daniel Viorel	Colegiul Tehnic Costin D. Neniţescu, Piteşti
Membru comisia tehnică	Nichifor Ștefan	Colegiul Naţional Iaşi, Iaşi
Membru comisia tehnică	Nunu Liviu	Colegiul Național *„Vasile Alecsandri”*, Galați
Membru comisia tehnică	Oprea Cosmin	Colegiul Naţional Mircea cel Bătrân, Râmnicu Vâlcea
Membru comisia tehnică	Ștefănucă Alin	Liceul *„Dimitrie Cantemir”*, Darabani

Tehnoredactarea volumului acesta a fost făcută de Tamio-Vesa Nakajima.

Cuprins

Partea I
Probleme



Capitolul 1 Olimpiada Județeană de Informatică

1.1 Clasa a V-a

1.1.A Problema Ceas

Un atelier de fabricat ceasuri cu cuc are nevoie de plăcuțe cu numerele pentru orele pe care trebuie să le așeze pe discul ceasurilor. Aceste numere sunt realizate la o imprimantă.

Din cauza unei erori imprimanta tipărește plăcuțe cu numere naturale, unele mai mari ca 12. Atelierul poate utiliza doar plăcuțe cu numere cuprinse între 0 și 12. Pentru a utiliza aceste numere este nevoie ca ele să fie tăiate începând din partea dreaptă în grupuri de maxim 2 cifre, fiecare grup reprezentând valoarea de pe o plăcuță, care să fie o cifră la 0 la 9 sau unul dintre numerele 10, 11, 12. Dacă pe o plăcuță se găsește un număr mai mare ca 12 atunci plăcuța trebuie tăiată, astfel încât în urma tăierii să se obțină numere de cel mult 2 cifre. Dacă în numărul de pe o plăcuță cifra zecilor este 0, atunci la prima tăiere se ia doar cifra unităților, altfel dacă numărul format cu cifra zecilor și unităților este mai mare ca 12, atunci se taie prima dată cifra unităților, iar dacă numărul format cu cifra zecilor și unităților este 10, 11 sau 12 se taie prima dată numărul format din ultimele 2 cifre, apoi procedeul se repetă până la tăierea completă a plăcuței. Imprimanta a realizat N plăcuțe. De exemplu dacă plăcuța este 12030, după tăiere se obțin $0, 3, 0, 12$.

Cerința 1

Determinați numărul total de apariții ale cifrei X pe plăcuțe înainte de tăiere.

Cerința 2

Determinați numărul de tăieturi realizate conform enunțului.

Date de intrare

Pe prima linie a fișierului `ceas.in` se află valorile C, X și N separate prin câte un singur spațiu. Pe linia a doua se află N numere naturale separate prin câte un singur spațiu, având semnificația din enunț. Pentru $C = 1$ se rezolvă doar cerința 1, iar pentru $C = 2$ se rezolvă doar cerința 2.

Date de ieșire

Fișierul `ceas.out` conține pe prima linie un singur număr natural care reprezintă valoarea calculată conform cerinței.

Restricții și precizări

- $1 \leq N \leq 100\,000$.
- $0 \leq X \leq 9$.
- Valorile din șir sunt numere naturale ≤ 50.000.
- Pentru testele în care avem $C = 2$ valoarea X este prezentă în fișierul de intrare chiar dacă nu este folosită în rezolvare.
- Pentru teste în valoare de 39 de puncte avem $C = 1$.
- Pentru teste în valoare de 61 de puncte avem $C = 2$.

Exemple

ceas.in	ceas.out
1 0 6 1010 40 201 5123 31 6	4
2 0 6 120 40 201 5123 31 6	7

Explicații

Pentru primul exemplu, pe plăcuțe cifra 0 apare de patru ori.
Pentru al doilea exemplu, în ordinea tăierilor se obțin: 0,12; 0,4; 1,0,2; 3,12,5; 1,3; 6. Numărul de tăieturi este 7.

1.1.B Problema Sss

Se dă un număr N, și un șir de N numere naturale nenule.

Cerința 1

Determinați suma valorilor aflate pe ultimele K poziții în șir, unde K reprezintă valoarea celei mai din dreapta cifre nenule a primei valori din șir.

Cerința 2

Ne imaginăm împărțirea șirului în secvențe în următorul mod: prima secvență este formată din primele L elemente, a doua este formată din următoarele $L-1$ elemente, a treia este formată din următoarele $L-2$ elemente și așa mai departe; ultima secvență este formată dintr-un singur element și acesta *coincide cu ultimul element din șir*. Considerând suma valorilor fiecărei secvențe, să se determine cea mai mare dintre aceste sume.

Date intrare

Pe prima linie a fișierului `sss.in` se află două valori C și N separate printr-un spațiu. Pe linia a doua se află N numere naturale separate prin câte un spațiu. Pentru $C = 1$ se rezolvă doar cerința 1 iar pentru $C = 2$ se rezolvă doar cerința 2.

Date ieșire

Fișierul `sss.out` conține un singur număr care reprezintă valoarea calculată conform cerinței.

Restricții și precizări

- $1 \leq N \leq 100\,000$.
- Valorile din șir sunt numere naturale nenule $\leq 100\,000$.
- Se garantează că pentru testele în care $C = 1$ șirul are cel puțin K elemente.
- Se garantează că valoarea lui N permite descompunerea conform descrierii, pentru testele care au $C = 2$.
- Pentru teste în valoare de 51 de puncte avem C=1.
- Pentru 27 de puncte dintre testele în care C=1 primul număr din șir are o cifră.
- Pentru teste în valoare de 49 de puncte avem C=2.
- Pentru teste în valoare de 22 de puncte dintre cele care au C=2, valoarea lui N este mai mică sau egală cu 10.
- Denumirea problemei este o prescurtare de la „sume și secvențe”.

Exemple

sss.in	sss.out
1 6 120 4 21 5 31 6	37
2 10 1 4 2 1 3 6 1 6 5 3	11

Explicații

În primul exemplu ultima cifră nenulă a primului element din șir este 2. Suma ultimelor două valori din șir este 37.

În al doilea exemplu descompunerea se poate realiza în secvențe de lungimile 4, 3, 2 și 1. Sumele obținute pentru fiecare sunt: 8, 10, 11, 3.

1.2 Clasa a VI-a

1.2.A Problema Cmmdc

Se dă un șir $a_1, a_2, \ldots, a_n$ de numere naturale nenule.

Cerință

Să se determine răspunsul pentru una din următoarele cerințe:
1. Cel mai mare divizor comun al celor n numere.
2. Cel mai mare divizor comun care se poate obține alegând exact $n-1$ elemente din șir.
3. Cel mai mare divizor comun care se poate obține alegând exact $n-2$ elemente din șir.

Date de intrare

Fișierul de intrare `cmmdc.in` conține pe prima linie un număr natural T reprezentând cerința cerută (1, 2, sau 3), pe a doua linie se află numărul natural nenul n, iar de pe următoarele n linii se găsesc, câte un număr pe fiecare linie, cele n elemente ale șirului.

Date de ieșire

În fișierul `cmmdc.out` se va afișa răspunsul pentru cerința cerută.

Restricții și precizări

- $2 \leq a_i \leq 2^{63} - 1$ pentru $1 \leq i \leq n$ (numerele sunt de tip `long long`).

#	Punctaj	Restricții
1	16	$T = 1$, $3 \leq n \leq 100\,000$ și $a_i \leq 50\,000\,000$, pentru $1 \leq i \leq n$
2	20	$T = 1$ și $3 \leq n \leq 100\,000$
3	21	$T = 2$ și $3 \leq n \leq 3\,000$
4	21	$T = 2$ și $3 \leq n \leq 100\,000$
5	12	$T = 3$ și $3 \leq n \leq 300$
6	10	$T = 3$ și $3 \leq n \leq 2\,000$

Exemple

cmmdc.in	cmmdc.out	Explicații
1 5 48 40 20 16 80	4	T = 1, deci se cere determinarea celui mai mare divizor comun al celor cinci numere: 48, 40, 20, 16 și 80. Răspunsul este 4.
2 5 48 40 20 16 80	8	T = 2, deci se rezolvă cerința 2. Eliminând numărul 20, rămân n - 1 = 4 numere, iar cmmdc(16, 48, 40, 80) = 8, care este și maximul posibil.
3 5 48 40 20 16 80	20	T = 3, deci se rezolvă cerința 3. Eliminând numerele 16 și 48 rămân n - 2 = 3 numere, iar cmmdc(40, 20, 80) = 20, care este și maximul posibil.

1.2.B Problema Vecine

Se dă un șir de n cifre $c_1, c_2, \ldots, c_n$, adică $0 \leq c_i \leq 9$. Dintr-un șir de cifre se poate obține un șir de $1 \leq m \leq n$ numere $a_1, a_2, \ldots, a_m$ astfel:

- Inițial considerăm fiecare cifră un număr și obținem șirul de n numere $a_i = c_i$.
- Un număr nou poate fi obținut prin lipirea unei secvențe de două sau mai multe numere vecine din șirul original. Două elemente dintr-un șir se numesc vecine dacă acestea se regăsesc în șir pe poziții alăturate.
- Operația de lipire de două sau mai multe numere se poate realiza de oricâte ori atât timp cât numărul obținut este mai mic sau egal cu 2 000 000 000, nu începe cu cifra 0 și există cel puțin două numere în șir.
- De exemplu șirul $[3, 5, 0, 2, 7, 3]$ poate deveni $[35, 0, 2, 73]$ prin lipirea numerelor $3, 5 \rightarrow 35$ și $7, 3 \rightarrow 73$, care ulterior poate deveni $[3502, 73]$ prin lipirea numerelor $35, 0, 2 \rightarrow 3502$. Dar nu putem crea șirul $[35, 02, 73]$, deoarece am avea un număr care începe cu 0.

Două numere vecine sunt consecutive dacă primul este cu 1 mai mic decât al doilea.

Cerință

Cunoscându-se șirul de cifre inițial, să se obțină următoarele rezultate:

1. Presupunând că nu se face nici o lipire de cifre, fiecare cifră devenind un număr în șir, adică $a_i = c_i$, să se determine câte perechi de numere vecine consecutive există în șir.
2. Să se determine o modalitate de lipire a cifrelor astfel încât să se obțină cele mai mari două numere vecine consecutive și să se afișeze primul dintre aceste numere.

Date de intrare

Fișierul de intrare `vecine.in` conține pe prima linie două numere p și n, p reprezentând cerința 1 sau 2, iar pe linia următoare cele n cifre, despărțite prin câte un spațiu.

Date de ieșire

În fișierul de ieșire `vecine.out` se va afla un singur număr natural. Dacă $p = 1$, acesta va reprezenta răspunsul pentru cerința 1. Dacă $p = 2$, acesta va reprezenta răspunsul pentru cerința 2.

Restricții și precizări

- Pentru cerința 2 se garantează că numerele ce se pot obține nu vor depăși 2 000 000 000.
- Tot pentru cerința 2 se garantează existența a cel puțin o pereche de numere vecine consecutive.
- Cifra 0 poate forma singură doar numărul 0.
- Două numere vecine sunt consecutive dacă primul este cu 1 mai mic decât al doilea.

#	Punctaj	Restricții
1	20	$p = 1$ și $3 \leq n \leq 100\,000$
2	80	$p = 2$ și $3 \leq n \leq 100\,000$

Exemple

vecine.in	vecine.out
1 18 3 2 1 2 1 0 6 3 0 5 6 3 0 6 9 2 9 3	2
2 18 3 2 1 2 1 0 6 3 0 5 6 3 0 6 9 2 9 3	6305

Explicații

Pentru primul exemplu:

$$[3, 2, \underline{1, 2}, 1, 0, 6, 3, 0, \underline{5, 6}, 3, 0, 6, 9, 2, 9, 3]$$

Există două perechi de numere vecine consecutive formate dintr-o singură cifră: $1, 2$ și $5, 6$.

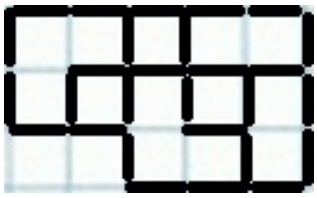
Pentru cel de-al doilea exemplu putem lipi următoarele secvențe:

$$[3, 2, 1, 2, 1, 0, \underline{6, 3, 0, 5}, \underline{6, 3, 0, 6}, 9, 2, 9, 3] \rightarrow [3, 2, 1, 2, 1, 0, 6305, 6306, 9, 2, 9, 3]$$

Perechea cu cele mai mari două numere vecine consecutive este 6305 și 6306. Conform cerinței s-a scris în fișier doar primul număr din pereche.

1.3 Clasa a VII-a

1.3.A Problema Pătrățele



Gigel are în fața sa pe o foaie de matematică un desen obținut prin trasarea mai multor linii orizontale și verticale de lungime 1 de-a lungul modelului foii de matematică.

Privind desenul de pe foaie el se întreabă: *„Oare câte pătrate s-au format din liniile trasate?"*

În desenul alăturat se vede foaia formată din 3 linii şi 5 coloane, precum şi liniile trasate până la un moment dat. Se pot distinge trei pătrate de latură 1, două pătrate de latură 2 şi un pătrat de latură 3.

În problema noastră vom codifica fiecare pătrat de latură 1 de pe foaie cu un număr natural cuprins între 0 şi 15 obținut prin însumarea codificării fiecărei laturi astfel:

1. dacă latura de sus este trasată,
2. dacă latura din dreapta este trasată,
3. dacă latura de jos este trasată,
4. dacă latura din stânga este trasată.

În acest fel desenul alăturat poate fi codificat printr-un tablou bidimensional de dimensiuni 3×5 cu valorile:

9	7	15	13	7
14	15	11	15	11
1	3	12	7	14

Cerință

Fiind date dimensiunile n şi m ale foii de matematică, precum şi tabloul bidimensional de dimensiune $n \times m$ care conține codificarea foii, să se determine:

1. numărul total de pătrate existente pe foaia de matematică în desenul realizat conform codificării,
2. distribuția numărului de pătrate în ordinea strict crescătoare a lungimii laturilor,
3. unde poate fi trasată încă o linie astfel încât numărul total de pătrate să crească și să devină maxim posibil.

Date de intrare

Fişierul de intrare `patratele.in` conţine pe prima linie trei numere naturale n m t, separate prin câte un spaţiu, indicând dimensiunile foii de matematică $n \times m$, respectiv cerinţa care trebuie rezolvată (1, 2 sau 3).

Fiecare dintre următoarele n linii conţine câte m numere naturale, fiecare dintre acestea reprezentând codificarea foii de matematică.

Date de ieșire

Fișierul de ieșire `patratele.out` va conține următoarele în funcție de cerința cerută:

1. Dacă $t = 1$, pe prima linie numărul total de pătrate determinat;
2. Dacă $t = 2$, pe fiecare linie vor fi afișate câte două numere naturale nenule a și b, separate printr-un spaţiu, indicând lungimea laturii pătratelor — a, respectiv numărul de pătrate cu latura de lungimea respectivă — b, în ordinea strict crescătoare a valorilor lui a;
3. Dacă $t = 3$, prima linie va conține numărul maxim de pătrate, iar linia a doua va conține 2 valori naturale *lin*, *col* și un cuvânt *pozitie* separate printr-un spațiu, unde:
   - *lin*, *col* reprezintă coordonatele pătratului de latură 1 unde se trasează linia suplimentară;
   - *pozitie* $\in$ {`SUS`, `DREAPTA`, `JOS`, `STANGA`, `NU`} (se va afișa `NU` în cazul în care nu se poate trasa nicio linie — în acest caz cele 3 valori numerice afișate vor fi de asemenea 0).

Restricții și precizări

- Numerotarea liniilor și coloanelor foii de matematică începe de la 1.
- Dacă la cerința $t = 3$ se obțin mai multe poziții de trasare a liniei, se va afișa soluția cu indicele liniei minim, iar în caz de egalitate după linii, se va afișa soluția cu indicele coloanei minim. În cazul în care există mai multe posibilități de trasare a unei linii în același pătrat, pozițiile vor fi luate în ordinea `SUS`, `DREAPTA`, `JOS`, `STANGA`.
- $1 \leq n, m \leq 60$.

#	Punctaj	Restricții
1	30	$t = 1$
2	30	$t = 2$
3	10	$t = 3$ și $1 \leq n, m \leq 20$
4	30	$t = 3$

Exemple

pătrățele.in	**pătrățele.out**	**Explicații**
3 5 1 9 7 15 13 7 14 15 11 15 11 1 3 12 7 14	6	Se rezolvă cerința 1. În total au fost găsite 6 pătrate
3 5 2 9 7 15 13 7 14 15 11 15 11 1 3 12 7 14	1 3 2 2 3 1	Se rezolvă cerința 2. 3 pătrate de latură 1 2 pătrate de latură 2 1 pătrat de latură 3

pătrățele.in	**pătrățele.out**	**Explicații**
3 5 3 9 7 15 13 7 14 15 11 15 11 1 3 12 7 14	9 2 5 JOS	Se rezolvă cerința 3. Dacă se trasează încă o linie la pătratul din linia 2 coloana 5 jos, se mai obțin încă 3 pătrate
3 3 3 9 1 3 8 0 2 12 0 0	0 0 0 NU	Se rezolvă cerința 3. Nu se poate adăuga niciun pătrat nou prin trasarea unei singure linii

1.3.B Problema Pseudocomp

Àles a primit ca temă următoarea problemă: *„Fiind dat un șir A cu N numere naturale distincte, să se calculeze suma cifrelor fiecărui element al șirului"*.

După ce și-a terminat tema, acesta observă că sunt mai multe perechi de indici (i, j) pentru care dacă $A_i < A_j$ atunci $S_i > S_j$, unde S_i reprezintă suma cifrelor lui A_i. El le va numi pe acestea perechi speciale de indici.

Cerință

Terminând prea repede tema, Àles primește o temă suplimentară cu două cerințe:

1. Determină două numere aflate în șirul A, pentru care indicii corespunzători formează o pereche specială.
2. Câte perechi speciale de indici (i, j) se găsesc în şirul A?

Ajutați-l pe Àles să rezolve tema suplimentară.

Date de intrare

Pe prima linie a fișierului `pseudocmp.in` se găsesc două numere naturale: T și N. Pe următoarea linie se găsesc N numere naturale, separate printr-un spațiu, reprezentând valorile din șirul A. Numărul T reprezintă numărul cerinței.

Date de ieșire

Pe prima linie a fișierului `pseudocmp.out`:

1. Dacă $T = 1$, se găsesc două numere naturale x, y, cu $x < y$, separate printr-un spațiu, reprezentând răspunsul pentru cerința 1 dacă există soluție sau -1, dacă nu există soluție. Dacă există mai multe soluții, se acceptă oricare dintre acestea.
2. Dacă $T = 2$, se găsește un singur număr natural, reprezentând răspunsul la cerința 2.

Restricții și precizări

- $1 \leq N \leq 100\,000$.
- $1 \leq A_i \leq 1\,000\,000$, pentru $1 \leq i \leq N$.

#	Punctaj	Restricții
1	15	$T = 1, N \leq 1000$
2	25	$T = 1, 1000 < N$
3	25	$T = 2, N \leq 1000$
4	35	$T = 2, 1000 < N$

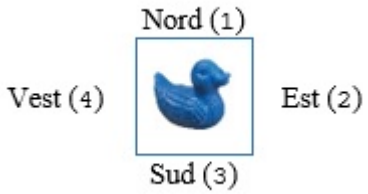
Exemple

pseudocomp.in	pseudocomp.out	Explicații
1 6 213 123 523 51 99 92	99 123	99 este mai mic decât 123 iar suma cifrelor lui 99 este 18, suma cifrelor lui 123 este 6, 18 > 6
2 6 213 123 523 51 99 92	6	Cele 6 perechi de indici sunt: (5, 1), (5, 2), (5, 3), (6, 1), (6, 2), (6, 3).
1 5 6 5 2 1 3	-1	

1.4 Clasa a VIII-a

1.4.A Problema Pelican

Într-o minunată zi de primăvară, P răţuşte au ieşit la plimbare pe lac. Un pelican milităros, care stătea pe mal, a decis să facă instrucţie cu nevinovatele raţe. Pentru aceasta, a cartografiat imediat lacul şi l-a reprezentat ca o matrice cu N linii (numerotate de la 0 la $N-1$ de sus în jos) şi N coloane (numerotate de la 0 la $N-1$ de la stânga la dreapta). Astfel, poziţia oricărei raţe pe lac poate fi identificată prin linia şi coloana pe care se află raţa. Raţele sunt orientate cu faţa spre una dintre direcţiile Nord, Sud, Est, Vest. Pelicanul a codificat direcţiile 1, 2, 3, 4 ca în figură.



Când pelicanul strigă: „Comanda la mine!" raţele trebuie să execute simultan cele K comenzi pe care le dă pelicanul. Comenzile pelicanului sunt codificate astfel:

Comandă	**Efect**	**Exemplu**
A nr	Raţa avansează cu *nr* poziţii în direcţia spre care este orientată. Dacă avansând depăşeşte marginea tablei de joc va intra pe latura opusă.	De exemplu, pe un lac 5×5, o raţă plasată în poziţia $(1,3)$ cu orientare 1 (Nord), executând comanda A 3 se va deplasa astfel: $(1,3) \rightarrow (0,3) \rightarrow (4,3) \rightarrow (3,3)$.
R nr	Raţa se roteşte cu $nr \times 90^\circ$ în sens orar, unde $nr \in \{1,2,3,4\}$.	De exemplu, dacă orientarea iniţială a raţei este 1 (Nord), comanda R 2 va schimba orientarea spre 3 (Sud).
Z nr	Raţa zboară pe linia nr/N şi coloana $nr\%N$, păstrând orientarea. Se garantează că $nr/N \in \{0,1,\ldots,N-1\}$.	De exemplu, pe un lac 5×5, după executarea comenzii Z 7, raţa va ajunge pe linia 1 şi coloana 2.

Cerinţă

Scrieţi un program care, cunoscând poziţia iniţială pe lac a celor P raţe şi succesiunea comenzilor pelicanului, determină poziţia finală a fiecărei raţe.

Date de intrare

Fişierul de intrare `pelican.in` conţine pe prima linie trei numere naturale N P K, cu semnificaţia din enunţ. Pe următoarele P linii sunt date câte 3 numere naturale i j d ($0 \leq i, j < N$ şi $1 \leq d \leq 4$) cu semnificaţia că pe linia i şi coloana j se găseşte o raţă orientată în direcţia d. Ultimele K linii conţin cele K comenzi, câte o comandă pe o linie, în formatul specificat în enunţ (un caracter din mulţimea $\{'A', 'R', 'Z'\}$ şi un număr natural). Valorile scrise pe aceeaşi linie sunt separate de câte un spaţiu.

Date de ieșire

Fişierul de ieşire `pelican.out` va conţine P linii. Pe linia i va fi scrisă poziţia celei de a i-a raţe din fişierul de intrare (linia şi coloana separate printr-un singur spaţiu) după executarea în ordine a celor K comenzi.

Restricții și precizări

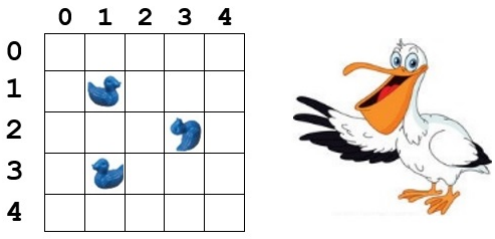
- $1 \leq N \leq 1\,000$.
- $1 \leq P \leq 10\,000$.
- $1 \leq K \leq 100\,000$.
- Mai multe raţe pot ocupa aceeaşi poziţie.
- Se garantează că datele din fişierul de intrare sunt corecte.
- Pentru teste valorând 76 de puncte fişierul de intrare nu conţine comanda Z.
- Pentru teste valorând 20 de puncte $N \leq 100$, $P \leq 100$ şi $K \leq 1000$.
- Pentru teste valorând 36 de puncte $N > 100$, $1000 \leq P \leq 5000$ şi $K \leq 50000$.

Exemple

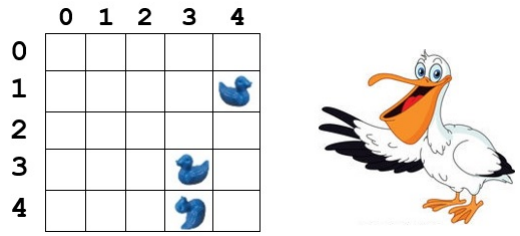
pelican.in	pelican.out
5 3 4 1 1 2 2 3 1 3 1 4 A 3 R 3 A 1 A 3	2 4 4 4 2 3

Explicație

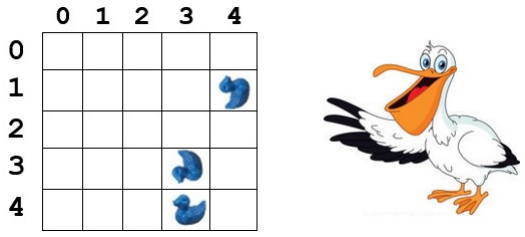
Lacul are 5 linii şi 5 coloane. Pe lac există 3 raţe poziţionate ca în figură.



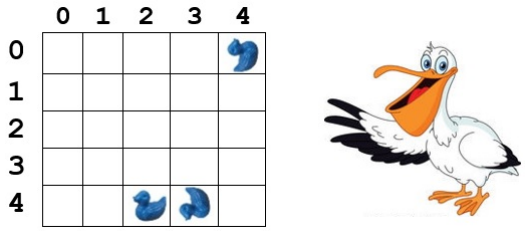
Pelicanul dă 4 comenzi pe care toate cele 3 raţe le execută în ordine.
Raţele execută comanda A 3



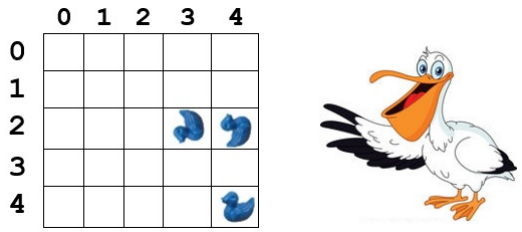
Raţele execută comanda R 3 (se rotesc cu 270° în sens orar)



Raţele execută comanda A 1



Raţele execută comanda A 3



1.4.B Problema Strips

Ana şi Bogdan au inventat un nou joc, pe care l-au denumit Strips. Este un joc de strategie, dar şi de antrenare a memoriei, deoarece se joacă pe o tablă care nu este vizibilă pentru cei doi jucători în timpul jocului. Tabla de joc este o bandă albă de lungime N cm, pe care sunt marcate poziţii de lungime 1 cm. Poziţiile sunt numerotate pe tablă de la 0 la $N-1$, poziţia 0 fiind marcată la începutul tablei (capătul din stânga), iar poziţia $N-1$ fiind marcată la sfârşitul tablei (capătul din dreapta). La începutul jocului fiecare jucător are Nr benzi colorate, toate de aceeaşi lungime L cm. Benzile Anei sunt de culoare roşie, iar benzile lui Bogdan sunt de culoare verde. Jucătorii mută alternativ, prima la mutare fiind Ana. La o mutare, jucătorul care este la rând alege o poziţie de pe tabla de joc şi dacă poziţia este validă, pe tabla de joc va fi plasată o bandă a jucătorului respectiv, cu capătul din stânga în poziţia aleasă. Dacă poziţia nu este validă, mutarea nu va fi executată, iar jucătorul respectiv va primi 1 punct de penalizare şi pierde banda care ar fi trebuit plasată pe tablă la poziţia respectivă (aceasta este eliminată din joc). O poziţie este considerată validă, dacă pe tabla de joc poate fi plasată o bandă de lungime L cu capătul din stânga al benzii fixat la poziţia specificată, astfel încât banda să fie integral pe tabla de joc, fără a se suprapune sau a se atinge cu o zonă de pe bandă colorată în culoarea adversarului. Jocul se termină când jucătorii nu mai au benzi. Fiecare jucător are ca scop să obţină o zonă pe bandă de lungime cât mai mare colorată în culoarea sa. O zonă de pe bandă este constituită din poziţii consecutive, colorate cu aceeaşi culoare.

Cerinţă

Scrieţi un program care citeşte lungimea tablei de joc, numărul de benzi colorate pe care le are fiecare jucător la începutul jocului, lungimea benzilor, precum şi poziţiile specificate de jucători pe parcursul jocului şi rezolvă următoarele două cerinţe:

1. determină numărul de puncte de penalizare pentru fiecare dintre cei doi jucători;
2. determină pentru fiecare jucător care este lungimea maximă a unei zone de pe tabla de joc colorată în culoarea sa la sfârşitul jocului.

Date de intrare

Fişierul de intrare `strips.in` conţine pe prima linie un număr natural C care reprezintă cerinţa care urmează a fi rezolvată (1 sau 2). Pe cea de-a doua linie se află trei numere naturale separate prin câte un spaţiu N Nr L, cu semnificaţia din enunţ. Celelalte linii ale fişierului de intrare conţin în ordine poziţiile specificate de jucători pe parcursul jocului, câte o poziţie pe o linie.

Date de ieșire

Fişierul de ieşire `strips.out` va conţine o singură linie pe care vor fi scrise două numere naturale *rezA rezB*, separate printr-un singur spaţiu.

Dacă $C = 1$ atunci *rezA* este numărul de puncte de penalizare acumulate de Ana, iar *rezB* numărul de puncte de penalizare acumulate de Bogdan.

Dacă $C = 2$ atunci *rezA* este lungimea maximă a unei zone de culoare roşie la sfârşitul jocului, iar *rezB* este lungimea maximă a unei zone de culoare verde la sfârşitul jocului.

Restricții și precizări

- $1 \leq N \leq 10^9$.
- $1 \leq Nr \leq 50\,000$.
- $1 \leq L \leq 20\,000$.
- Se garantează că pentru datele de test, la finalul jocului, pentru fiecare dintre cei doi jucători numărul de zone disjuncte de pe tabla de joc colorate în culoarea jucătorului respectiv este $\leq 5\,000$.
- Poziţiile sunt numere naturale mai mici decât N.
- Fiindcă sunt începători, Ana şi Bogdan încă nu joacă optim.
- Pentru teste valorând 50 de puncte cerinţa este 1.
- Pentru teste valorând 40 de puncte $1 \leq N \leq 10^6$, $1 \leq L \leq 1\,000$ şi $1 \leq Nr \leq 1\,000$.

Exemple

strips.in	strips.out
1 20 4 3 9 15 2 13 5 17 0 12	0 1
2 20 4 3 9 15 2 13 5 17 0 12	8 7

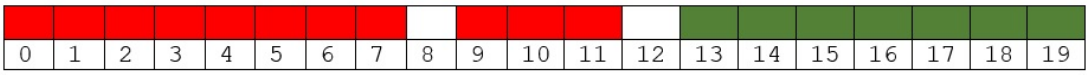
Explicație

Tabla de joc are 20 de cm, poziţiile fiind numerotate de la 0 la 19. Jucătorii au fiecare câte 4 benzi de lungime 3

- La prima mutare Ana aplică o bandă roşie peste poziţiile 9, 10, 11.
- La a doua mutare Bogdan aplică o bandă verde peste poziţiile 15, 16, 17.
- La a treia mutare Ana aplică o bandă roşie peste poziţiile 2, 3, 4.

- La a patra mutare Bogdan aplică o bandă verde peste poziţiile 13, 14, 15.
- La a cincea mutare Ana aplică o bandă roşie peste poziţiile 5, 6, 7.
- La a şasea mutare Bogdan aplică o bandă verde peste poziţiile 17, 18, 19.
- La a şaptea mutare Ana aplică o bandă roşie peste poziţiile 0, 1, 2.
- La a opta mutare Bogdan a ales o poziţie invalidă (pentru că atinge o zonă de culoare roşie), ca urmare mutarea nu va fi executată (va avea 1 punct de penalizare)

Tabla de joc la finalul jocului va arăta astfel



1.5 Clasa a IX-a

1.5.A Problema Bâlbă

Regele George al VI-lea al Regatului Unit s-a confruntat cu o problemă neobișnuită pentru o persoană care trebuia să țină discursuri: era bâlbâit. Acesta se bâlbâia chiar și când spunea numere. Interesant este faptul că, atunci când spunea un număr, el repeta doar una dintre cifrele acelui număr, imediat după ce pronunța cifra respectivă. Spre exemplu, numărul 70 243 putea fi rostit atunci când se bâlbâia ca 770 243 sau ca 700 243 sau ca 702 243 sau ca 702 443 sau ca 702 433.

Un *palilindrom* este un număr natural pentru care există o bâlbâială a regelui care îl transformă într-un palindrom. Spre exemplu, 25 373 552 este un palilindrom, pentru că după o bâlbâială poate deveni 255 373 552, acesta fiind un număr palindrom.

Cerințe

Fiind dat un număr natural nenul X să se determine:

1. Câte numere diferite poate genera X după o bâlbâială și câte numere diferite pot deveni X după o bâlbâială.
2. Cel mai mare număr palilindrom care se poate forma cu cifrele lui X. Nu este obligatoriu să se folosească toate cifrele lui X.

Date de intrare

Pe prima linie a fișierului de intrare `balba.in` se află numărul C, număr care poate fi 1 sau 2 și reprezintă cerința ce trebuie rezolvată.

Pe cea de-a doua linie se află numărul N, reprezentând numărul de cifre al numărului X.

Pe următoarea linie se află, în ordine, cifrele lui X, separate prin câte un spațiu.

Date de ieșire

Dacă C este 1, fișierul de ieșire `balba.out` va avea obligatoriu două linii, fiecare linie conținând exact un număr.

Pe prima linie se va scrie un număr natural ce reprezintă câte numere diferite poate genera X după o bâlbâială.

Pe cea de-a doua linie se va scrie un număr natural ce reprezintă câte numere diferite pot deveni X după o bâlbâială.

Dacă C este 2, pe prima linie a fișierului de ieșire `balba.out` se va scrie cel mai mare număr palilindrom ce se poate crea cu cifrele lui X.

Restricții și precizări

- $1 \leq N \leq 10^5$.
- Numărul X este un număr natural nenul cu maxim 100 000 de cifre.
- Un număr palindrom este un număr care are aceeași valoare dacă este citit de la stânga la dreapta sau de la dreapta la stânga.
- Pentru rezolvarea corectă a cerinței 1 se vor acorda 40 de puncte. Pentru fiecare număr corect afișat se va acorda jumătate din punctajul asociat testului.
- Pentru rezolvarea corectă a cerinței 2 se vor acorda 60 de puncte.

Exemple

bâlbă.in	**bâlbă.out**
1 8 7 0 2 2 4 3 3 3	5 2
1 25 1 2 3 1 2 3 4 1 2 3 4 5 1 2 3 4 5 6 1 2 3 4 5 6 7	25 0
2 11 2 4 7 8 1 4 8 7 4 2 1	87442112478
2 7 1 2 3 4 0 0 0	4

Explicații

Pentru exemplul 1, numerele diferite care pot fi generate din 70 224 333 printr-o bâlbâială sunt: 770 224 333, 700 224 333, 702 224 333, 702 244 333, 702 243 333. Numerele diferite din care 70 224 333 poate fi generat printr-o bâlbâială sunt: 7 024 333, 7 022 433.

Pentru exemplul 2, sunt 25 de numere diferite care pot fi generate din X printr-o bâlbâială, însă X nu poate fi generat de niciun număr printr-o bâlbâială.

Pentru exemplul 3, mai există și alte palilindroame care se pot forma cu cifrele lui 24 781 487 421, însă 87 442 112 478 este cel mai mare dintre ele. Numărul 87 442 112 478 este palilindrom, pentru ca acesta se poate transforma după o bâlbâială într-un număr palindrom, și anume 874 421 124 478.

Pentru exemplul 4, nu se poate forma un palilindrom care să aibă toate cifrele lui X. Astfel, cel mai mare palilindrom care se poate crea folosind cifrele lui X este 4. Numărul 4 este palilindrom, pentru ca acesta se poate transforma după o bâlbâială într-un număr palindrom, și anume 44.

1.5.B Problema Oneout

Definim un număr *liber de pătrate* ca fiind un număr natural care nu are ca divizor niciun pătrat perfect mai mare ca 1. Prin convenție, 1 este considerat *liber de pătrate*.

Așadar, șirul numerelor libere de pătrate este: $1, 2, 3, 5, 6, 7, 10, 11, 13, 14, 15, 17, \ldots$

Se consideră un șir de N numere naturale X_i, $1 \leq i \leq N$, unde N este un număr natural.

O secvență este un subșir format din numere aflate pe poziții consecutive în șirul dat.

Definim o *bisecvență* ca un subșir nevid obținut prin eliminarea dintr-o secvență a unui număr care nu este la începutul sau la sfârșitul secvenței.

Cerințe

1. Să se determine câte numere libere de pătrate conține șirul dat.
2. Să se determine cea mai lungă bisecvență din șir formată din numere libere de pătrate, obținută prin eliminarea unui număr care *nu* este liber de pătrate

Date de intrare

Fișierul de intrare `oneout.in` conține pe primul rând un număr natural C, care poate fi doar 1 sau 2, reprezentând cerința, pe a doua linie numărul natural N iar pe a treia linie N numere naturale, separate prin câte un spațiu, cu semnificația de mai sus.

Date de ieșire

Dacă C este egal cu 1, în fișierul de ieșire `oneout.out` se va scrie numărul de numere libere de pătrate din șir.

Dacă C este egal cu 2:

- pe prima linie a fișierului de ieșire `oneout.out` se vor scrie două numere L și K despărțite printr-un spațiu, unde L reprezintă lungimea maximă a unei bisecvențe cu proprietățile cerute, iar K reprezintă numărul de bisecvențe de lungime maximă existente în șir.
- pe următoarele K linii se vor scrie indicii de început și de sfârșit ai fiecărei bisecvențe de lungime maximă găsite, în ordinea crescătoare a indicelui de start, despărțite printr-un spațiu.
- dacă șirul nu conține nicio bisecvență cu proprietățile cerute, în fișierul de ieșire se va scrie -1.

Restricții și precizări

- $3 \leq N \leq 1\,000\,000$.
- $2 \leq X_i \leq 1\,000\,000, 1 \leq i \leq N$.
- Lungimea unei bisecvențe reprezintă numărul de numere din aceasta.

Subtaskuri

- Pentru teste în valoare de 37 puncte $C = 1$, din care pentru teste în valoare de 24 puncte $3 \leq N \leq 25$.
- Pentru teste în valoare de 63 puncte $C = 2$, din care pentru teste în valoare de 23 puncte $3 \leq N \leq 101$.

Exemple

oneout.in	oneout.out
1 6 10 2 12 7 8 15	4
2 6 10 2 12 7 8 15	3 1 1 4
2 7 5 28 17 24 15 20 18	2 2 1 3 3 5
2 9 3 10 5 8 9 11 4 15 21	3 1 6 9

Explicații

Pentru primul exemplu, $C = 1, N = 6, X_{1-6} = \{10, 2, 12, 7, 8, 15\}$. Se rezolvă prima cerință.

Sunt 4 numere libere de pătrate în șirul X_{1-6} și anume $10, 2, 7, 15$.

Pentru al doilea exemplu, $C = 2, N = 6, X_{1-6} = \{10, 2, 12, 7, 8, 15\}$. Se rezolvă a doua cerință.

Dacă se elimină 12 se obține bisecvența $10, 2, 7$ de lungime 3. Dacă se elimină 8 se obține bisecvența $7, 15$ de lungime 2. Deci există o singură bisecvență de lungime maximă $= 3$, care începe în poziția 1 și se termină în poziția 4.

Pentru al treilea exemplu, $C = 2, N = 7, X_{1-7} = \{5, 28, 17, 24, 15, 20, 18\}$. Se rezolvă a doua cerință.

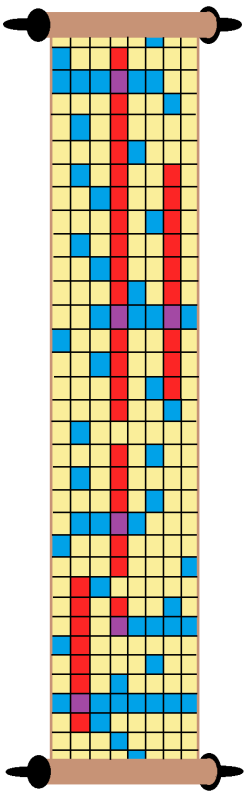
Dacă se elimină 28 se obține bisecvența $5, 17$ de lungime 2. Dacă se elimină 24 se obține bisecvența $17, 15$ tot de lungime 2. Deci există două bisecvențe de lungime maximă $= 2$. Prima începe în poziția 1 și se termină în poziția 3. A doua începe în poziția 3 și se termină în poziția 5.

Pentru al patrulea exemplu, $C = 2, N = 9, X_{1-9} = \{3, 10, 5, 8, 9, 11, 4, 15, 21\}$. Se rezolvă a doua cerință.

8 nu poate fi eliminat deoarece este situat la sfârșitul unei bisecvențe iar 9 nu poate fi eliminat pentru că ar fi începutul unei bisecvențe.

Singurul număr care nu este liber de pătrate ce poate fi eliminat este 4 și se va obține bisecvența $11, 15, 21$ de lungime 3 care începe în poziția 6 și se termină în poziția 9.

1.5.C Problema Pergament



Deși nu obișnuiește să deseneze, Adrian are o pasiune inedită: îi place să schițeze pe hârtie orașe imaginare ... mai exact cum ar arăta acestea văzute de sus. În acest an, de ziua lui a primit cadou un pergament! Normal că menirea acestuia va fi ca Adrian să deseneze pe el schița celui mai mare oraș pe care și l-a imaginat până acum.

Pergamentul are lățimea unei coli de hârtie, însă lungimea sa este neașteptat de mare. De asemenea, pergamentul este împărțit în pătrate astfel încât pe lungime se află exact N pătrate iar pe lățime se află exact K pătrate. Astfel, Adrian are la dispoziție exact NK pătrate pe care le poate colora.

El decide să coloreze doar străzile orașului, deoarece nu are timp de mai mult și plănuiește să folosească două tipuri de străzi:

1. Străzi orizontale
   - Vor fi desenate ca o secvență continuă de pătrate albastre.
   - Pe fiecare rând de la 1 la N se va afla *exact* o stradă orizontală. Deci, la final vor fi *exact* N străzi orizontale.
   - Fiecare stradă se desfășoară pe un singur rând.
   - Lungimea fiecărei străzi va fi de minim un pătrat și de maxim K pătrate și este egală cu numărul de pătrate ce o compun.
   - Strada poate începe pe oricare pătrat de pe rând și poate avea orice lungime cât timp nu depășește limitele pergamentului.
2. Străzi verticale
   - Vor fi desenate ca o secvență continuă de pătrate roșii.
   - Adrian va desena exact Q străzi verticale, desfășurate pe una dintre coloanele de la 1 la K.
   - Pe o coloană pot exista mai multe străzi verticale cu condiția să nu se suprapună. Nu este obligatoriu să existe străzi verticale pe toate coloanele.
   - Lungimea fiecărei străzi va fi de minim un pătrat și de maxim N pătrate și este egală cu numărul de pătrate ce o compun.
   - Strada poate începe pe oricare pătrat de pe coloană și poate avea orice lungime cât timp nu depășește limitele pergamentului.

La final, Adrian observă că anumite pătrate au devenit mov, deoarece fac parte atât dintr-o stradă verticală cât și din una orizontală, deci au fost colorate atât cu roșu cât și cu albastru. Adrian este fascinat de apariția acestora și vrea să știe câte pătrate mov sunt în desenul său. Fiind prea obosit să le numere, vă roagă pe voi să-l ajutați.

Cerință

Cunoscând numerele N, K, Q, precum și poziționarea celor N străzi orizontale și a celor Q străzi verticale, să se determine numărul de pătrate mov din pergament.

Date de intrare

Pe prima linie a fișierului de intrare `pergament.in` se află trei numere naturale separate prin cate un spațiu: N, K, Q, cu semnificația din enunț.

Pe a doua linie se află patru numere naturale separate prin câte un spațiu: A, B, C, D.

Pe a treia linie se află două numere naturale X_1, Y_1, unde X_1 reprezintă coloana pătratului de început al străzii orizontale de pe rândul 1, iar Y_1 reprezintă lungimea acesteia.

Datele următoarelor $N-1$ străzi se vor calcula prin formulele de mai jos, unde X_i reprezintă coloana pătratului de început al străzii orizontale de pe rândul i ($2 \leq i \leq N$), iar Y_i reprezintă lungimea acesteia:

$$X_i = 1 + (X_{i-1}A + B) \bmod K,$$
$$Y_i = 1 + (Y_{i-1}C + D) \bmod (K - X_i + 1).$$

Pe următoarele Q linii se află câte trei numere naturale J, R, L, unde J reprezintă coloana pe care se află strada verticală, R reprezintă rândul pe care se află pătratul de început al străzii, iar L reprezintă lungimea străzii.

Date de ieșire

În fișierul de ieșire `pergament.out` se va afla un singur număr natural ce reprezintă numărul de pătrate mov din desenul lui Adrian.

Restricții și precizări

- $1 \leq N \leq 10\,000\,000$.
- $1 \leq K \leq 50$.
- $1 \leq Q \leq 100\,000$.
- $1 \leq A, B, C, D \leq 10\,000\,000$.
- $1 \leq X_i \leq K$.
- $1 \leq Y_i \leq K - X_i + 1$.
- $1 \leq J \leq K$.
- $1 \leq R \leq N$.
- $1 \leq L \leq N - R + 1$.
- Rândurile sunt numerotate de la 1 la N, iar coloanele sunt numerotate de la 1 la K.

Subtaskuri

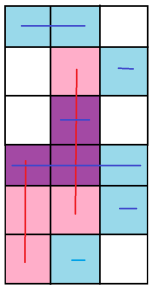
- Pentru 40 puncte $N \leq 20\,000$.
- Pentru 70 de puncte $N \leq 500\,000$.
- Pentru 100 de puncte nu există condiții adiționale.

Exemple

pergament.in	pergament.out
6 3 2 1 1 1 1 1 2 2 2 4 1 4 3	3

Explicație

Imaginea alăturată reprezintă pergamentul desenat de Adrian din exemplu.



Conform formulelor, vom avea următoarele străzi orizontale:

- Linia 1: $X_1 = 1$ și $Y_1 = 2$.
- Linia 2: $X_2 = 3$ și $Y_2 = 1$.
- Linia 3: $X_3 = 2$ și $Y_3 = 1$.
- Linia 4: $X_4 = 1$ și $Y_4 = 3$.
- Linia 5: $X_5 = 3$ și $Y_5 = 1$.
- Linia 6: $X_6 = 2$ și $Y_6 = 1$.

Se observă că există exact 3 pătrate mov.

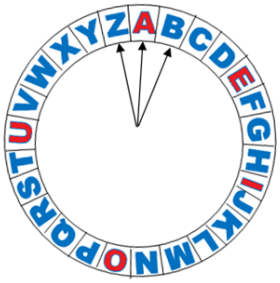
1.6 Clasa a X-a

1.6.A Problema Circular

O imprimantă circulară are litere mari ale alfabetului englezesc dispuse circular de la `A` la `Z`. Imprimanta are un indicator care inițial este plasat la litera `A`.

Pentru a tipări o literă indicatorul imprimantei se mișcă la stânga sau dreapta. Mișcarea indicatorului către o literă alăturată aflată la stânga sau la dreapta literei curente se realizează într-o secundă. De exemplu: pentru a tipări șirul `BCY` mișcarea indicatorului se va face către dreapta de la `A` la `B` într-o secundă, apoi de la `B` la `C` într-o secundă, apoi către stânga de la `C` la `Y` în 4 secunde. În total pentru a tipări șirul `BCY` sunt necesare 6 secunde. Imprimanta va alege întotdeauna sensul cel mai avantajos de deplasare, astfel încât timpul de deplasare să fie minim.

Imprimanta tipărește literele în două culori roșu sau albastru. Unele litere se tipăresc cu cerneală roșie, restul cu cerneală albastră. Pentru simplitate le vom numi litere roșii și litere albastre.



Fiind date un șir de litere albastre nu neapărat distincte și mulțimea literelor roșii ale imprimantei, să se calculeze:

1. Care este timpul pentru tipărirea la imprimantă circulară a șirului de litere albastre.
2. Să se insereze între oricare două litere albastre aflate pe poziții consecutive câte o literă roșie astfel încât să se obțină timpul minim pentru tipărire și să se afișeze:
   - timpul minim,
   - numărul de șiruri distincte care sunt tipărite cu timp minim,
   - șirul minim lexicografic dintre toate șirurile ce sunt tipărite în acest timp.

Date de intrare

Fișierul `circular.in` conține:

1. pe prima linie un număr natural c cu valori posibile 1 sau 2 reprezentând cerința problemei,
2. pe a doua linie un șir de litere albastre, nu neapărat distincte,
3. pe a treia linie mulțimea literelor roșii distincte în ordine alfabetică.

Date de ieșire

În fișierul `circular.out` se va afișa în funcție de cerință:

1. Dacă $c = 1$, un singur număr natural reprezentând timpul necesar pentru tipărirea la imprimantă a șirului de litere albastre,
2. Dacă $c = 2$ se vor tipări trei rezultate, fiecare pe câte o linie:
   - timpul minim pentru tipărire conform cerinței a doua,
   - numărul de șiruri distincte care sunt tipărite cu timp minim $\bmod 666\,013$,
   - șirul minim lexicografic ce obține acest timp.

Restricții și precizări

- Cele două șiruri conțin doar litere mari ale alfabetului englez.
- Lungimea șirului de litere albastre nu depășește 50 000 de litere.
- Mulțimea literelor roșii nu depășește 25 de litere, care sunt distincte și afișate în ordine alfabetică.
- Toate celelalte litere care nu se regăsesc în mulțimea literelor roșii, sunt albastre.
- Pentru cazul $c = 2$ se acordă punctaj parțial astfel:
  - 25% din punctaj, pentru afișarea timpului minim,
  - 25% din punctaj, pentru afișarea numărului de șiruri ce obțin timpul minim,
  - 50% din punctaj, pentru afișarea șirului minim lexicografic.
- **Atenție!** Pentru obținerea punctajului la cerința a doua, pentru orice test, în fișierul de ieșire trebuie să existe exact trei linii care respectă formatul cerut.

#	Punctaj	Restricții
1	24	$c = 1$
2	76	$c = 2$

Exemple

circular.in	circular.out	Explicații
1 BBTH AEIOU	21	Timpul de tipărire al șirului BBTH este 21 și se obține astfel: de la A la B = 1 secundă de la B la B = 0 secundă de la B la T = 8 secunde de la T la H = 12 secunde
2 AMYMAMAMY BCDEFGHIJKLNOPQRSTUVWX	96 568708 ABMNYNMBABMBABMNY	Timpul minim de tipărire este 96. Avem 214358881 șiruri distincte, iar 214358881 mod 666013=568708. Prima soluție în ordine lexicografică este ABMNYNMBABMBABMNY.

circular.in	circular.out	Explicații
2 BBTH AEIOU	23 4 BABATIH	Timpul minim pentru tipărirea la imprimantă este 23 și se obține pentru șirul BABATIH astfel: de la A la B = 1 secundă de la B la A = 1 secundă de la A la B = 1 secundă de la B la A = 1 secundă de la A la T = 7 secunde de la T la I = 11 secunde de la I la H = 1 secundă în total 23 de secunde. Avem 4 șiruri pentru care se obține timp minim la tipărire: BABATIH, BABATOH, BABUTIH, BABUTOH Prima soluție în ordine lexicografică este BABATIH.

1.6.B Problema Pulsar

Data stelară 3210:

Căpitanul navei USS Entrerprise, Jean-Luc Picard se află într-o misiune importantă în cuadrantul Beta al galaxiei.

Acesta trebuie să ajungă cât mai rapid de la planeta Vulcan până la planeta Qo'noS, dar din păcate pentru această misiune Jean-Luc Picard nu va putea să ajungă instantaneu la destinație folosind warp drive-ul navei, ci va trebui să se deplaseze în mod normal, din sector în sector.

Harta galaxiei este reprezentată sub forma unei tabele bidimensionale de dimensiune $N \times N$, în care fiecare celulă reprezintă un sector al galaxiei. Coordonatele sectorului în care se află planeta Vulcan sunt (x_s, y_s), iar coordonatele sectorului în care se află planeta Qo'noS sunt (x_f, y_f).

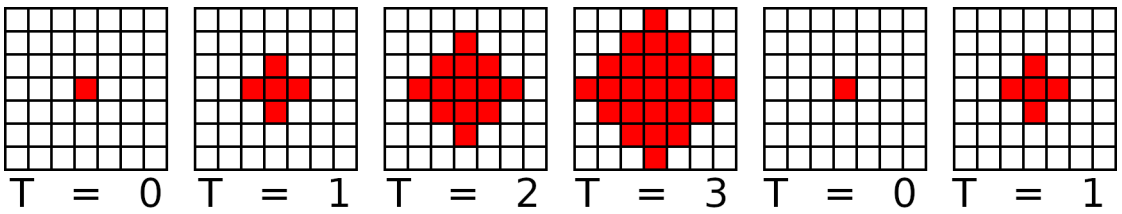
USS Enterprise se poate deplasa într-o unitate de timp dintr-un sector în oricare dintre sectorele adiacente, fie pe aceeasi linie, fie pe aceeasi coloană. În plus, nava poate staționa o perioadă nedeterminată de timp în orice sector. Nava se poate afla doar pe un sector care la momentul actual de timp nu o pune în pericol.

Pentru că nicio aventură nu este lipsită de pericole, drumul lui Jean-Luc Picard este presărat de *pulsari*, obiecte cosmice foarte periculoase care lansează în vecinătatea lor, la intervale fixe de timp, unde gravitaționale care ar putea distruge USS Enterprise.

Un pulsar P_i este caracterizat prin patru variabile (x_i, y_i, r_i, t_i), unde (x_i, y_i) reprezintă coordonatele sectorului în care se regăsește pulsarul, r_i reprezintă raza de acțiune a pulsarului, iar t_i reprezintă starea în care se află pulsarul la momentul de început al deplasării navei.

Un pulsar P_i trece periodic printr-un număr de r_i stări de la 0 la $r_i - 1$. Când acesta se află în starea t, acesta afectează toate sectoarele aflate la o distanță Manhattan mai mică sau egală cu t față de sectorul în care se află acesta. Dacă pulsarul la un moment de timp se află în starea t, la momentul următor se va afla în starea $(t + 1) \bmod r_i$.

Un exemplu de funcționare al unui pulsar cu rază de acțiune $r = 4$, timp de 6 unități de timp, începând cu $t = 0$ este următorul:



Vouă vă revine rolul de a îl ajuta pe Jean-Luc Picard și să îi răspundeți la una din următoarele întrebări știind harta galaxiei:

1. Care este numărul maxim de sectoare ale galaxiei S_{max} afectate la orice moment de timp de către cel puțin un pulsar.
2. Care este timpul minim T_{min} de care are nevoie Jean-Luc Picard pentru a ajunge pe planeta Qo'noS.

Date de intrare

Din fișierul `pulsar.in` se vor citi următoarele:

- Pe prima linie se vor afla trei numere C, N și P separate prin câte un spațiu, reprezentând cerința ce trebuie rezolvată, dimensiunea galaxiei și numărul de pulsari din galaxie.
- Pe următoarele P linii se vor afla câte patru numere separate prin spațiu, x_i, y_i, r_i, t_i, reprezentând descrierea pulsarului P_i.
- Pe penultima linie se vor afla două numere separate printr-un spațiu reprezentând coordonatele sectorului planetei Vulcan x_s și y_s.
- Pe ultima linie se vor afla două numere separate printr-un spațiu reprezentând coordonatele sectorului planetei Qo'noS x_f și y_f.

Date de ieșire

În fișierul `pulsar.out` se va afișa un singur număr în funcție de cerință:

- Dacă $C = 1$, atunci se va afișa numărul S_{max}.
- Dacă $C = 2$, atunci se va afișa numărul T_{min}.

Restricții și precizări

- Distanța Manhattan dintre două coordonate (x_1, y_1) și (x_2, y_2) este definită ca: $|x_1 - x_2| + |y_1 - y_2|$.
- Nava nu va putea părăsi la niciun moment de timp harta galaxiei.
- Undele pulsarilor pot părăsi harta galaxiei, dar acele sectoare nu reprezintă interes pentru problema noastră.
- Se garantează că la momentul plecării, nava nu este aflată în pericol.
- Se garantează că există soluție.
- Pot exista mai mulți pulsari în același sector.
- $C \in \{1, 2\}$.
- $3 \leq N \leq 500$.
- $1 \leq P \leq 15\,000$.
- $0 \leq t_i < r_i \leq 6$, pentru $1 \leq i \leq P$.
- $1 \leq x_s, y_s, x_f, y_f \leq N$.
- $1 \leq x_i, y_i \leq N$, pentru $1 \leq i \leq P$.

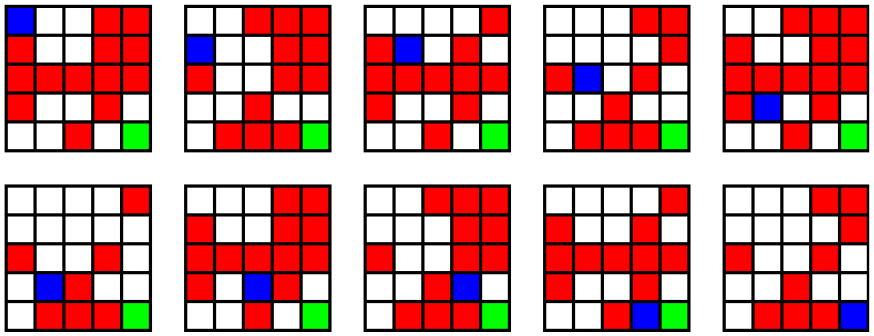
#	Punctaj	Restricții
1	19	$C = 1$
2	22	$C = 2$ și $r_i = 1$, pentru $1 \leq i \leq P$
3	9	$C = 2$, $N \leq 7$ și $r_i <= 3$ pentru $1 \leq i \leq P$
4	13	$C = 2$, $t_i = 0$, pentru $1 \leq i \leq P$
5	37	$C = 2$

Exemple

pulsar.in	pulsar.out
1 5 4 3 1 2 1 1 5 3 1 5 3 2 0 3 4 2 1 1 1 5 5	14
2 5 4 3 1 2 1 1 5 3 1 5 3 2 0 3 4 2 1 1 1 5 5	9

Explicații

Mai jos se poate observa dumul realizat de USS Enterprise. Cu albastru s-a ilustrat nava, cu rosu, zonele afectate de pulsari, iar cu verde planeta Qo'nos:



Pentru primul exemplu, se observă că nu va exista niciodată un moment de timp în care pulsarii să ocupe mai mult de 14 sectoare.

În figura de mai sus, este prezentat un posibil drum de durată 9. Acest timp este și minim pentru exemplul dat.

1.6.C Problema Transport

Anul 1905:

Un stat din America de Sud și-a propus investiții majore în infrastructura feroviară. Brazilianul Badinho este managerul unei companii de transport feroviar pe o magistrală importantă. De-a lungul magistralei se află N stații, numerotate de la 1 la N. Fiecărei stații îi corespunde un număr X_i care reprezintă numărul de kilometri de la începutul magistralei până la stația i ($X_1 = 0$). Pentru simplitate Badinho reprezintă magistrala ca o dreaptă, iar stațiile ca puncte pe dreapta respectivă, stația i aflându-se la coordonata X_i.

O rută reprezintă o submulțime de cel puțin 2 stații dintre cele N, cu semnificația că în aceste stații se vor face opriri. Orice rută operată de Badinho are 2 stații numite capete, definite ca fiind cea mai apropiată stație, inclusă în rută, de începutul magistralei respectiv cea mai îndepărtată stație, inclusă în rută, de începutul magistralei.

Compania lui Badinho va primi o subvenție pentru deschiderea unei noi rute, care va fi proporțională cu lungimea rutei deschise. Mai exact, Badinho va primi C reali (realul este monenda națională a Braziliei) pentru fiecare kilometru din noua rută. Lungimea rutei se definește ca fiind distanța dintre capete.

Badinho poate deschide două tipuri de rute:

- Regio — se fac opriri în toate stațiile dintre cele două capete.
- Expres — unele stații dintre cele două capete pot fi traversate fără a opri în ele.

Pentru a deschide o rută Badinho trebuie să construiască câte un depou în capetele rutei respective. Costul pentru a construi un depou în stația i este D_i reali.

Știind că Badinho trebuie să cheltuiască întreaga sumă pe care ar primi-o dintr-o subvenție, să se determine:

1. Numărul de moduri de a deschide o rută de tip Regio, **modulo** $10^9 + 7$.
2. Numărul de moduri de a deschide o rută de tip Expres, **modulo** $10^9 + 7$.

Date de intrare

În fișierul `transport.in` se află:

- Pe prima linie tipul cerinței T, care poate avea valoarea 1 sau 2.
- Pe a doua linie N și C, separate printr-un spațiu, reprezentând numărul de stații, respectiv suma primită per kilometru ca subvenție.
- Pe următoarele N linii, pe linia $i + 2$ se află câte o pereche X_i și D_i, separate printr-un spațiu, reprezentând distanța la care se află stația i față de începutul magistralei, respectiv costul de a construi un depou în stația i.

Date de ieșire

În fișierul `transport.out` se va afișa:

- Dacă $T = 1$, mumărul de moduri de a deschide o rută de tip Regio, **modulo** $10^9 + 7$.
- Dacă $T = 2$, mumărul de moduri de a deschide o rută de tip Expres, **modulo** $10^9 + 7$.

Restricții și precizări

- Două rute se consideră distincte dacă diferă prin cel puțin o stație.
- $2 \leq N \leq 200\,000, 1 \leq C \leq 10^9$.
- $0 \leq X_i, D_i \leq 10^9$, pentru $1 \leq i \leq N$.
- $X_1 = 0$.
- sirul X este sortat strict crescător: $X_i < X_j$, pentru $1 \leq i < j \leq N$.
- toate liniile de cale ferată ale magistralei sunt deja construite, singurele costuri pe care le va suporta Badinho sunt cele de construire a depourilor.

#	Punctaj	Restricții
1	12	$T = 1, N \leq 1\,000$
2	26	$T = 1, N \leq 200\,000$
3	6	$T = 2, N \leq 15$
4	15	$T = 2, N \leq 1\,000$
5	41	$T = 2, N \leq 200\,000$

Exemple

transport.in	transport.out
1 5 1 0 2 1 1 3 10 4 15 6 4	2
2 5 1 0 2 1 1 3 10 4 15 6 4	12

Explicații

Pentru primul exemplu Rutele posibile în condițiile cerinței 1 sunt: $\{1, 2, 3, 4, 5\}$, $\{2, 3, 4, 5\}$

Ruta $\{1, 2, 3, 4, 5\}$ conține opriri în stațiile $1, 2, 3, 4, 5$. Stațiile 1 și 5 sunt cele 2 capete. Suma primită din subvenție este: $1 \times (6 - 0) = 6$ reali ($6 - 0$ reprezintă distanța dintre stația 1 și 5), iar costul de construire a celor 2 depouri e: $2 + 4 = 6$ reali.

Pentru exemplul al doilea Rutele posibile în condițiile cerinței 2 sunt: $\{1,5\}$, $\{1,2,5\}$, $\{1,3,5\}$, $\{1,4,5\}$, $\{1,2,3,5\}$, $\{1,2,4,5\}$, $\{1,3,4,5\}$, $\{1,2,3,4,5\}$, $\{2,5\}$, $\{2,3,5\}$, $\{2,4,5\}$, $\{2,3,4,5\}$

Ruta $\{1,2,5\}$ conține opriri în stațiile $1,2,5$. Stațiile 1 și 5 sunt cele 2 extreme. Suma primită din subvenție e: $1 \times (6-0) = 6$ reali, iar costul de construire a celor 2 depouri e: $2+4 = 6$ reali.

1.7 Clasele XI–XII

1.7.A Problema Dulciuri

Tsubasa-chan adoră dulciurile! De curând a apărut un nou tip de desert. Astfel decide să înfăptuiască o nouă fabrică care să producă acest produs delicios.

Fabrica conține un container imens pătratic, plin de aluat, de $10^6 \times 10^6$ unități. Fiecare punct din container are drept coordonate o pereche de numere reale (x, y), unde $0 \leq x, y \leq 10^6$, iar fiecare punct are o dulceață. Dulceața unui punct este un număr real, inițial 0. Pentru fabricarea desertului este nevoie de Q operații, care pot fi de următoarele tipuri:

- O *îndulcire verticală*, determinată de o coordonată x_u întreagă și o valoare întreagă v. După această operație, toate punctele din container (x, y) unde $x_u \leq x < x_u + 1$ devin mai dulci cu v.
- O *îndulcire orizontală*, determinată de o coordonată y_u întreagă și o valoare întreagă v. După această operație, toate punctele din container (x, y) unde $y_u \leq y < y_u + 1$ devin mai dulci cu v.
- O *degustare*, determinată de 4 coordonate întregi x_q, y_q, x'_q, y'_q. Pentru aceeastă operație, Tsubasa ia o lingură, o pune în aluat la punctul (x_q, y_q), și apoi o duce in linie dreaptă la punctul (x'_q, y'_q). Mișcarea se efectuează într-o secundă, cu viteză constantă. După aceea, Tsubasa gustă desertul, vrând să afle *dulceața totală* a aluatului din lingură. Această valoare se calculează în felul următor: dacă lingura trece prin zone de dulceața d_1 pentru t_1 secunde, de dulceață d_2 pentru t_2 secunde, ..., și de dulceață d_k pentru t_k secunde, atunci dulceața totală din lingură este $t_1 d_1 + t_2 d_2 \ldots + t_k d_k$. Nu se modifică dulceața din container.

Cerință

Dându-se toate operațiile întreprinse în producerea desertului, să se găsească dulcețile totale ce sunt găsite la toate operațiile de degustare.

Date de intrare

Pe prima linie a fișierului de intrare `dulciuri.in` se va găsi numărul Q de operații.

Pe următoarele Q linii urmează descrieri a tuturor operațiilor, câte una pe linie, în ordine. O operație este codificată în felul următor:

- O îndulcire verticală este codificată prin 1 x_u v.
- O îndulcire orizontală este codificată prin 2 y_u v.
- O degustare este codificată prin 3 x_q y_q x'_q y'_q.

Date de ieșire

În fișierul de ieșire `dulciuri.out`, să se afișeze toate rezultatele degustărilor, în ordine, câte una pe linie. Rezultatul unei degustări se consideră a fi corect daca eroarea absolută sau relativă față de soluția comisiei este cel mult 10^{-7}.[1]

[1] Dacă soluțiile comisiei și a concurentului sunt s^*, s atunci eroarea absolută este $|s^* - s|$ și eroarea relativă este $|s^* - s|/|s^*|$.

Restricții și precizări

- Toate coordonatele din datele de intrare sunt întregi în intervalul $[0, 10^6]$.
- $0 \leq v \leq 1\,000$.
- v este întreg.
- $1 \leq Q \leq 100\,000$.

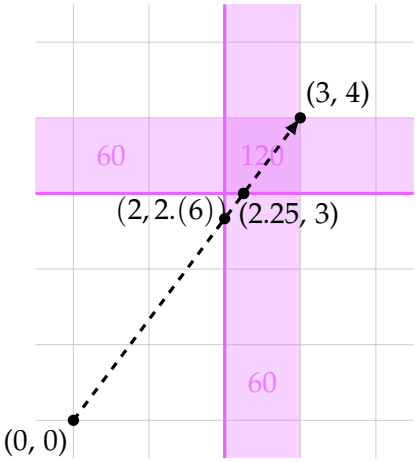
#	Punctaj	Restricții
1	20	Nu se fac îndulciri orizontale. $Q \leq 2\,000$
2	20	Pentru fiecare degustare, fie $x_q = x'_q$ sau $y_q = y'_q$. $Q \leq 2\,000$
3	10	Se face cel mult o degustare
4	20	Toate degustările se fac după toate îndulcirile
5	10	$Q \leq 2\,000$
6	20	Fără restricții suplimentare

Exemple

dulciuri.in	dulciuri.out
3 1 2 60 2 3 60 3 0 0 3 4	35
4 1 2 10 3 2 0 2 1 3 3 0 3 1 3 2 0 2 0	10 0 10
6 1 4 413 1 3 234 2 5 244 2 3 777 3 1 2 14 15 3 31 4 2 40	128.3076923077 29.0881226054

Explicație

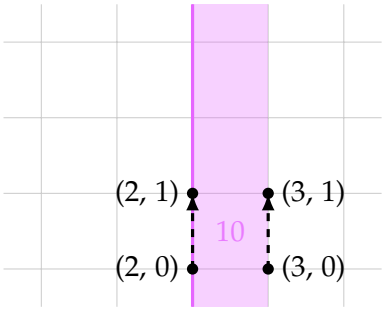
Situația pentru degustarea din primul exemplu este explicată în diagrama de mai jos.



Zonele roz sunt zonele în care s-a aplicat o îndulcire, și numerele reprezintă cu cât s-a îndulcit. Zona din intersecția îndulcirilor are dulceața 120. Linia diagonală punctată reprezintă traseul.

Traseul are lungimea $\sqrt{3^2 + 4^2} = 5$, și este completat în o secunda — astfel are viteza de 5 unități pe secundă. Segmentul de la $(2, 2.(6))$ la $(2.25, 3)$ are lungimea $\sqrt{(2.25 - 2)^2 + (2.(6) - 3)^2} = \sqrt{(1/4)^2 + (1/3)^2} = 5/12$, și are dulceața 60 — astfel el este traversat în $(5/12) \times (1/5) = 1/12$ secunde, și contribuie cu $(1/12) \times 60 = 5$ la dulceața totală. Segmentul de la $(2.25, 3)$ la $(3, 4)$ are lungimea $\sqrt{(3 - 2.25)^2 + (4 - 3)^2} = 5/4$, și are dulceața 120 — astfel el este traversat în $(5/4) \times (1/5) = 1/4$ secunde, și contribuie cu $(1/4) \times 120 = 30$ la dulceața totală. Astfel, cum segmentul de la $(0, 0)$ la $(2, 2.(6))$ contribuie cu 0, dulceața totală este 35.

Situația pentru degustările din al doilea exemplu este explicată în diagrama de mai jos.



În primul traseu (cel din stânga) trecem mereu printr-o zonă cu dulceața 10, deci rezultatul degustării este 10. În al doilea traseu (cel din dreapta) trecem mereu printr-o zonă cu dulceața 0, deci rezultatul degustării este 0. În al treilea traseu, stăm pe loc pentru o secundă într-o zona de dulceață 10, deci răspunsul este 10.

1.7.B Problema Investiție

După o lungă activitate în domeniul instalaţiilor sanitare, Dorel s-a hotărât să investească averea acumulată în acţiuni ale mai multor companii. Astfel, el dispune de o listă cu N companii la care vrea să cumpere acţiuni, în M zile consecutive.

În prima zi, suma de bani investită în compania i este $s[1][i] = a[i]$, pentru orice $i = \overline{1, N}$, unde valorile $a[i]$ sunt date.

Numerele $a[1], a[2], \ldots, a[N]$ reprezintă o permutare a numerelor $1, 2, \ldots, N$.

În ziua a j-a el va investi în compania i o sumă de bani egală cu $s[j][i] = s[j-1][a[i]]$, pentru orice zi $j = \overline{2, M}$ şi orice companie $i = \overline{1, N}$.

Cerință

După finalizarea planului de investiţii, Dorel vrea să realizeze Q statistici referitoare la sumele investite.

Fiind date Q seturi de valori z_i, z_f, c_l, c_r, el doreşte să afle ce sumă a investit în perioada cuprinsă între zilele z_i şi z_f (inclusiv acestea), la companiile cu numere de ordine cuprinse între c_l şi c_r (inclusiv acestea).

Date intrare

Pe prima linie a fişierului de intrare `investitie.in` se află numerele N şi M, separate prin spaţiu.

Pe a doua linie se află valorile $a[1], a[2], \ldots, a[N]$, separate prin spaţiu.

Pe a treia linie se află valoarea lui Q.

Pe următoarele Q linii se află câte patru valori, z_i, z_f, c_l, c_r, separate prin spaţiu.

Date ieșire

În fişierul de ieşire `investitie.out` se vor afişa, pe linii diferite, sumele investite corespunzătoare fiecăreia din cele Q statistici din fişierul de intrare.

Restricții și precizări

- $1 \leq N, Q \leq 100\,000$.
- $1 \leq M \leq 1\,000\,000\,000$.
- $1 \leq z_i \leq z_f \leq M$.
- $1 \leq c_l \leq c_r \leq N$.
- $0 \leq c_r - c_l \leq 100$.

#	Punctaj	Restricții
1	10	$M = 1$
2	20	$1 \leq N, M \leq 100, 1 \leq Q \leq 1\,000$
3	12	$101 \leq N, M \leq 3\,000$
4	24	$1 \leq N \leq 50$
5	34	Fără alte restricţii

Exemple

investiție.in	investiție.out
8 3 3 1 7 2 6 4 5 8 5 1 1 3 7 1 2 1 4 1 3 2 8 2 3 3 6 3 3 3 3	24 29 93 24 6

Explicație

Sumele investite în cele trei zile, în acţiuni ale celor opt companii, vor fi:

$$\begin{matrix} 3 & 1 & 7 & 2 & 6 & 4 & 5 & 8 \\ 7 & 3 & 5 & 1 & 4 & 2 & 6 & 8 \\ 5 & 7 & 6 & 3 & 2 & 1 & 4 & 8 \end{matrix}$$

Prima statistică cere suma investită în prima zi în companiile cu numere de ordine de la 3 la 7. Suma este $7+2+6+4+5=24$.

A doua statistică cere suma investită în primele două zile în companiile cu numerele de ordine de la 1 la 4. Suma este $(3+1+7+2)+(7+3+5+1)=29$.

Similar se calculează celelalte statistici.

1.7.C Problema Superhedgy

Ariciul Gălușcă este un arici obișnuit pe timp de zi. Noaptea, însă, el este de fapt eroul misterios al orașului Hedgytown — un oraș mai special, deoarece are clădiri atât deasupra solului, cât și sub pamânt, unde gravitația este inversată.

Orașul poate fi văzut ca o dreaptă (ce reprezintă solul), cu un șir de clădiri dreptunghiulare lipite deasupra solului, și un șir de clădiri dreptunghiulare lipite dedesubtul solului. Sunt N clădiri peste pământ și M sub pământ. Cele două șiruri încep și se termină la aceleași poziții. Fiecare clădire este caracterizată de trei valori: L, H și E. L reprezintă lățimea clădirii, H reprezintă înălțimea clădirii și E reprezintă efortul necesar pentru a folosi liftul din acea clădire.

Inițial, Gălușcă se afla pe partea din stânga a orașului, la nivelul solului (în dreptul primei clădiri) și trebuie să ajungă pe partea din dreapta a orașului, tot la nivelul solului (în dreptul ultimei clădiri). În acest scop, el se poate deplasa pe contururile clădirilor, consumând o unitate de efort pentru a se deplasa o unitate pe conturul unei clădiri — sau folosind lifturile.

O deplasare cu liftul este mereu de pe conturul unei clădiri *până pe conturul clădirii aflate pe cealaltă parte a solului* — cu alte cuvinte, când folosește liftul, ariciul nu are voie să se deplaseze doar până la sol sau să se oprească în interiorul clădirii. În cazul în care clădirea pe care se află ariciul necesită efortul E pentru a folosi liftul și clădirea de pe partea opusă a solului necesită efortul E', atunci efortul total necesar pentru a folosi liftul este $E + E'$.

Deplasarea cu liftul se efectuează vertical. Astfel, liftul îl va lăsa pe Gălușcă la aceeași distanță orizontală față de începutul orașului, doar că pe acoperișul clădirii opuse. Liftul nu poate fi folosit unde se întâlnesc două clădiri adiacente orizontal — nici dacă se întâlnesc pe partea solului de unde începe deplasarea, nici dacă se întâlnesc pe partea solului unde se termină deplasarea. *Atenție, cum Gălușcă este un erou, acesta nu va merge niciodată înapoi — doar va înainta sau va folosi lifturile.*

Cerință

Se cere să se afle numărul minim de unități de efort pe care trebuie să le depună ariciul Gălușcă pentru a ajunge la destinația sa.

Date intrare

Pe prima linie a fișierului de intrare `superhedgy.in` se dă N, reprezentând numărul de clădiri de deasupra solului.

Pe următoarele N linii ale fișierului de intrare se dau L, H, și E, reprezentând datele clădirilor de deasupra solului, în ordinea în care sunt plasate în oraș începând cu cea mai apropiată clădire de Gălușcă.

Pe linia $N + 2$ a fișierului de intrare se află M, reprezentând numărul de clădiri dedesubtul solului.

Pe următoarele M linii ale fișierului de intrare se dau L, H, și E, reprezentând datele clădirilor de dedesubtul solului, în ordinea în care sunt plasate în oraș începând cu cea mai apropiată clădire de Gălușcă.

Date ieșire

În fișierul de ieșire `superhedgy.out` se va afișa un singur număr natural, reprezentând numarul minim de unități de efort pe care trebuie să le depună Gălușcă pentru a ajunge la destinație.

Restricții și precizări

- $1 \leq N, M \leq 100\,000$.
- $1 \leq L, H \leq 1\,000\,000\,000$.
- $0 \leq E \leq 1\,000\,000\,000$.
- Suma lungimilor clădirilor de deasupra și de dedesubtul pământului coincid. Fie L_{Total} această lungime totală.

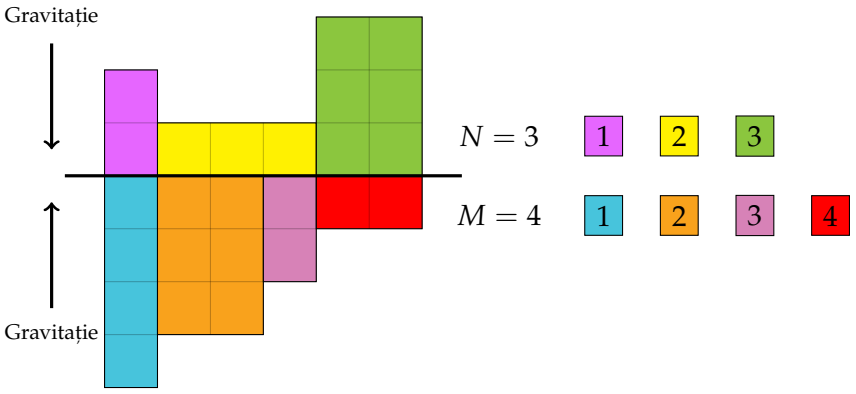
#	Punctaj	Restricții
1	20	$1 \leq L_{Total} \leq 10$
2	20	$E = 0$ pentru toate clădirile din oraș, $1 \leq L_{Total} \leq 100\,000$
3	40	$1 \leq L_{Total} \leq 100\,000$
4	20	Fără restricții suplimentare

Exemple

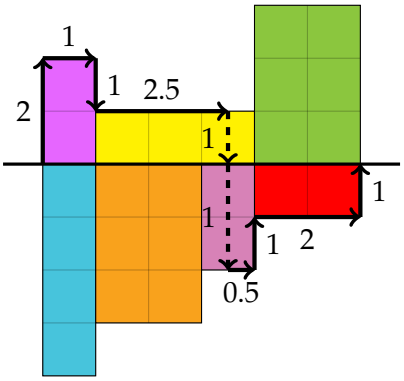
superhedgy.in	superhedgy.out
3 1 2 5 3 1 1 2 3 1 4 1 4 10 2 3 1 1 2 1 2 1 1	13

Explicație

Orașul va arăta ca în figura de mai jos:



Un traseu posibil al ariciului va fi următorul:



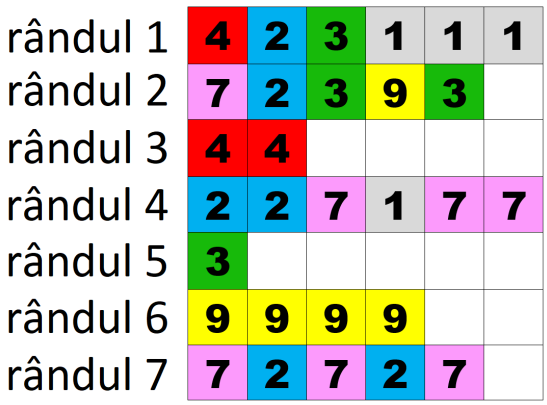
Atenție! Deplasarea cu liftul nu s-ar fi putut efectua mai la stânga sau mai la dreapta cu 0.5 unități.

Capitolul 2 Olimpiada Națională de Informatică

2.1 Clasa a V-a

2.1.A Problema Culori

Pe o foaie a unui caiet de matematică sunt N rânduri de pătrățele pe care Andrei le-a numerotat de sus în jos cu valori de la 1 la N. Pe fiecare rând, Andrei colorează unul sau mai multe pătrățele având la dispoziție un set de 9 creioane de culori diferite, culori ce sunt codificate cu valori distincte de la 1 la 9. Pentru fiecare rând al caietului, Andrei stabilește un număr de pătrățele alăturate ce le va colora și procedează astfel: alege un creion cu care colorează primul pătrățel (cel din stânga foii sale), apoi procedează la fel pentru al doilea pătrățel și așa mai departe până termină de colorat numărul de pătrățele stabilit de el pentru rândul respectiv (pot exista două sau mai multe pătrățele colorate la fel). Lungimea unui rând este astfel determinată de numărul tuturor pătrățelelor colorate de pe acel rând.



Cerințe

Cunoscând numărul N de rânduri cu pătrățele, numărul de pătrățele colorate de pe fiecare rând și culoarea fiecărui pătrățel, scrieți un program care să determine:

1. *Lmax* si *Kmax*, două numere naturale, unde *Lmax* reprezintă lungimea maximă unui rând ce are proprietatea că oricare două pătrățele alăturate au culori diferite, iar *Kmax* reprezintă câte astfel de rânduri sunt pe foaie.
2. Cel mai mare număr natural ce se poate forma prin lipirea tuturor cifrelor corespunzătoare culorilor de pe același rând, parcurse de la stânga la dreapta.

Date intrare

Fişierul `culori.in` conţine:

- pe prima linie două numere naturale C și N, unde C reprezintă numărul cerinţei şi poate avea valorile 1 sau 2, iar N reprezintă numărul rândurilor din caiet colorate de Andrei;
- pe fiecare din următoarele N linii, numere naturale despărțite prin câte un spațiu. Fiecare linie corespunde unui rând al foii de caiet, în ordinea numerotării rândurilor. Primul număr de pe fiecare linie reprezintă numărul pătrățelelor colorate de Andrei pe rândul respectiv, iar apoi următoarele valori reprezintă codurile culorilor folosite pentru colorarea pătrățelelor de pe rândul respectiv, fiecare corespunzând câte unui pătrățel, în ordine, începând cu primul de pe acel rând (cel din stânga), până la ultimul de pe acel rând (cel din dreapta).

Date ieșire

Fişierul `culori.out` va conţine pe prima linie:

1. pentru cerința 1, două numere naturale *Lmax* și *Kmax*, în această ordine și despărțite printr-un spațiu;
2. pentru cerința 2, un singur număr natural determinat conform cerinței.

Restricții și precizări

- $1 \leq N \leq 10000$.
- $1 \leq$ numărul pătrățelelor colorate de pe fiecare rând ≤ 500.
- Pentru rezolvarea corectă a primei cerințe se acordă 27 de puncte, iar pentru rezolvarea corectă a celei de-a doua cerințe se acordă 73 de puncte.
- Pentru teste în valoare de 10 puncte și $C = 2$, caietul nu conține două rânduri cu același număr de pătrățele colorate.
- Pentru teste în valoare de 23 puncte și $C = 2$, numărul pătrățelelor de pe fiecare rând este mai mic sau egal cu 19.

Exemple

culori.in	**culori.out**
1 7 6 4 2 3 1 1 1 5 7 2 3 9 3 2 4 4 6 2 2 7 1 7 7 1 3 4 9 9 9 9 5 7 2 7 2 7	5 2

culori.in	culori.out
2 7 6 4 2 3 1 1 1 5 7 2 3 9 3 2 4 4 6 2 2 7 1 7 7 1 3 4 9 9 9 9 5 7 2 7 2 7	423111

Explicații

Exemplul 1 Primul exemplu corespunde desenului din imaginea de mai sus.

Se va rezolva cerința 1.

Rândurile 2, 5 și 7 au proprietatea din cerință. Lungimea rândului 2 este 5, a rândului 5 este 1, iar a rândului 7 este 5, deci lungimea maximă a unui rând este 5 și sunt 2 rânduri de această lungime.

$Lmax = 5$ și $Kmax = 2$.

Exemplul 2 In al doilea exemplu se va rezolva cerința 2.

Numerele naturale construite din cifrele fiecărui rând sunt: 423111, 72393, 44, 227177, 3, 9999 și 72727.

Cel mai mare dintre ele este 423111.

2.1.B Problema Joc

Doi copii vor să joace un joc cu doi pioni și o tablă formată din N căsuțe numerotate de la 1 la N, așezate una după cealaltă, pe aceeași linie. Jocul are următoarele reguli:

- se așază pionii pe prima căsuță de pe tablă (fiecare copil are propriul pion),
- primul copil este cel care începe jocul,
- copiii vin la tabla de joc alternativ,
- cel care este la rând face, după regula de mai jos, una sau mai multe mutări înainte să cedeze locul celuilalt:
  - calculează o valoare X în modul descris mai jos,
  - își mută pionul înainte cu X poziții iar, dacă valoarea X calculată este 6, are dreptul la calcularea unei alte valori X, deci la încă o mutare, necedând încă locul celuilalt copil, iar dacă valoarea X este diferită de 6 cedează locul la tablă;
- X se calculează după regula:
  - dacă numărul mutării este impar atunci:

$$X = ((\textit{numărulMutării} + ((\textit{numărulCăsuțeiPionului} + N) \bmod 10)) \bmod 6) + 1,$$

  - dacă numărul mutării este par atunci:

$$X = ((((\textit{numărulMutării} + 1) \bmod 5) + ((\textit{numărulCăsuțeiPionului} + N) \bmod 10)) \bmod 6) + 1.$$

  unde N este numărul căsuțelor tablei de joc, *numărulMutării* semnifică a câta mutare este, mod este operația prin care se obține restul împărțirii întregi a două numere, iar valoarea rezultată, X, este una dintre cifrele 1, 2, 3, 4, 5 sau 6, cum de altfel se deduce din formulele de mai sus.
- în urma înaintării, dacă pionul ajunge pe o căsuță ocupată în acel moment de celălalt pion, îi ia locul acestuia, iar pionul care ocupa căsuţa este trimis la căsuța cu numărul 1 (întoarcerea acestui pion la poziția 1 nu se contorizează ca mutare);
- dacă un pion, după înaintare, ar ajunge în afara tablei de joc, este așezat pe căsuța N (ultima);
- este câștigător copilul care ajunge primul cu pionul la căsuţa N de pe tabla de joc, și atunci jocul se încheie.

Cerințe

Dându-se numărul N, determinați:

1. Numărul divizorilor lui N;
2. Numărul maxim de apariții ale unei valori calculate în timpul jocului prin formulele descrise;
3. Numerele căsuțelor ocupate, în timpul jocului, de pionul câștigătorului în ordinea în care acestea sunt vizitate.

Date intrare

Pe prima linie a fișierului `joc.in` se află două numere naturale, C și N separate printr-un spațiu.

Dacă $C = 1$, atunci se rezolvă doar prima cerință, dacă $C = 2$, atunci se rezolvă doar a doua cerință iar dacă $C = 3$, atunci se rezolvă doar cea de-a treia cerință.

Date ieșire

Fișierul de ieșire este `joc.out`.

- Dacă $C = 1$ sau $C = 2$, acesta conține un număr natural ce reprezintă răspunsul pentru cerința respectivă.
- Dacă $C = 3$, acesta conține un șir de numere naturale, separate prin câte un spațiu, care reprezintă răspunsul pentru a treia cerință.

Restricții și precizări

- $2 \leq N \leq 10.000$.
- Pentru teste în valoare de 23 de puncte, $C = 1$.
- Pentru alte teste în valoare de 33 de puncte, $C = 2$.
- Pentru alte teste în valoare de 44 de puncte, $C = 3$.
- Se garantează că există un câștigător.
- Pe parcursul jocului, copii pot ajunge pe căsuțe pe care le-au mai vizitat.
- Se garantează că numărul căsuțelor ocupate de copii este mai mic decât 100.000.
- Problema nu urmăreşte găsirea vreunei proprietăţi speciale pentru șirurile de valori calculate prin formulele date.

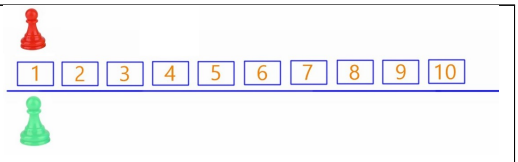
Exemple

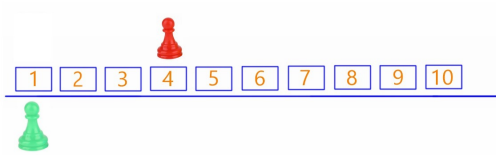
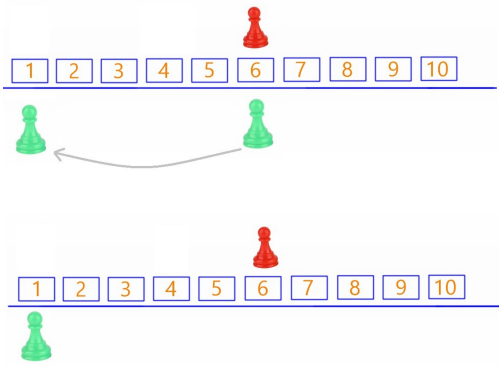
joc.in	joc.out
1 10	4
2 10	2
3 10	1 4 6 10

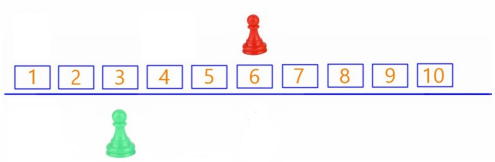
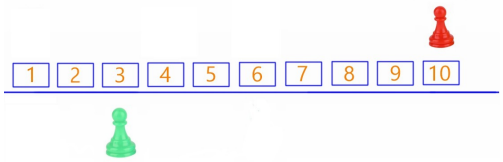
Explicații

În primul exemplu $C = 1$ deci se rezolvă prima cerință. $N = 10$ are 4 divizori.

În al doilea exemplu $C = 2$ deci se rezolvă a doua cerință.

Ambii pioni se află la căsuța 1. În imaginile alăturate, primul copil are pionul roșu și al doilea copil are pionul verde.	

<table>
<tr>
<td>Mută primul copil (pionul acestuia se află la căsuța 1)
• suntem la prima mutare (număr impar),
• se calculează cifra pentru înaintarea pionului:
$$X = (1 + (1 + 10) \bmod 10) \bmod 6 + 1 = 3,$$
• primul jucător înaintează pionul de la căsuța 1 la căsuța 4.</td>
<td> 
</td>
</tr>
<tr>
<td>Mută al doilea copil (pionul acestuia se află la căsuța 1)
• suntem la a doua mutare (număr par),
• se calculează cifra pentru înaintarea pionului:
$$X = ((((2 + 1) \bmod 5) + ((1 + 10) \bmod 10)) \bmod 6) + 1 = 5,$$
• al doilea copil înaintează pionul de la căsuța 1 la căsuța 6.</td>
<td> 
</td>
</tr>
<tr>
<td>Mută primul copil (pionul acestuia se află la căsuța 4)
• suntem la a treia mutare (număr impar),
• se calculează cifra pentru înaintarea pionului
$$X = (3 + (4 + 10) \bmod 10) \bmod 6 + 1 = 2,$$
• primul copil înaintează pionul de la căsuța 4 la căsuța 6,
• cum pionul celui de-al doilea copil se află la căsuța 6 el este întors la prima căsuță, deci pionul celui de-al doilea copil ocupă acum căsuța 1.</td>
<td> 
</td>
</tr>
</table>

Mută al doilea copil (pionul acestuia se află la căsuța 1) • suntem la a patra mutare (număr par), • se calculează cifra pentru înaintarea pionului $X = ((4+1) \bmod 5 + (1+10) \bmod 10) \bmod 6 + 1 = 2,$ • al doilea copil deplasează pionul de la căsuța 1 la căsuța 3.	
Mută primul copil (pionul acestuia se află la căsuța 6) • suntem la a cincea mutare (număr impar), • se calculează cifra pentru înaintarea pionului $X = (5 + (6+10) \bmod 10) \bmod 6 + 1 = 6,$ • primul copil deplasează pionul de la căsuța 6 la căsuța 10 (ar trebui să se deplaseze la căsuța 12 care este în afara tablei de joc), • cifra este 6 copilul are dreptul la încă o mutare dar a ajuns deja cu pionul la căsuța de final și se termină jocul.	

Primul copil este câștigător. Cifrele calculate au fost, în ordine, 3, 5, 2, 2, 6, cifra 2 a apărut de cele mai multe ori adică de 2 ori.

În al treilea exemplu C=3 deci se rezolvă a treia cerință. Primul copil este câștigător, el a ocupat în această ordine căsuțele: 1, 4, 6, 10.

2.1.C Problema Rotire25

George a primit ca temă la matematică următoarea problemă. Se dă un număr X, asupra acestui număr se pot face următoarele transformări:

1. În această ordine (toți acești 3 pași reprezintă o transformare):
   - se înmulțește numărul cu 5 (de exemplu: $X = 416$ devine $416 \cdot 5 = 2080$),
   - se elimină toate zerourile din număr (2080 devine 28),
   - se oglindește numărul (28 devine 82),
2. În această ordine (toți acești 3 pași reprezintă o transformare):
   - se înmulțește numărul cu 2 (de exemplu: $X = 32$ devine $32 \cdot 2 = 64$),
   - se elimină toate zerourile din număr (64 rămâne 64),
   - se oglindește numărul (64 devine 46),

George trebuie să aplice alternativ cele două transformări asupra numărului X. Prima dată aplică transformarea 1, apoi pe rezultatul obținut se aplică transformarea 2, apoi pe rezultat se aplică iar transformarea 1, apoi iar transformarea 2 și așa mai departe. George trebuie să aplice asupra numărului X exact K transformări, în ordinea descrisă mai sus.

Cerințe

Dându-se numerele X și K determinați:

1. Produsul dintre ultima cifră a numărului $\underbrace{X \cdot X \cdot X \cdot \ldots \cdot X}_{\text{de } K \text{ ori}}$ și prima cifră a lui X.
2. Numărul rezultat după aplicarea celor K transformări.

Date intrare

Pe prima linie a fișierului de intrare `rotire25.in` se află trei numere separate prin câte un spațiu: C, X și K.

Dacă $C = 1$ se va rezolva doar prima cerință, iar dacă $C = 2$ se va rezolva doar a doua cerință.

Date ieșire

Fișierul `rotire25.out` va conține un singur număr. Dacă $C = 1$, acest număr reprezintă rezultatul pentru prima cerință, iar dacă $C = 2$, acest număr reprezintă rezultatul pentru a doua cerință.

Restricții și precizări

- $1 \leq X \leq 999$.
- $1 \leq K \leq 1\,000\,000\,000$.
- Pentru teste în valoare de 29 de puncte, $C = 1$.
- Pentru teste în valoare de 71 de puncte, $C = 2$.
- Pentru teste în valoare de 7 puncte, $C = 1$ și se garantează că numărul obținut în urma înmulțirilor este $\leq 10^{18}$ ($1\,000\,000\,000\,000\,000\,000$).
- Pentru alte teste în valoare de 11 puncte: $C = 1$ și $K \leq 100\,000$.
- Pentru teste în valoare de 39 de puncte: $C = 2$ și $K \leq 100\,000$.
- Pentru alte teste în valoare de 9 puncte: $C = 2$ și $X \leq 9$.

Exemple

rotire25.in	rotire25.out
1 27 3	6
2 13 3	551
2 42 1782321	12

Explicații

Primul exemplu Se rezolvă cerinţa 1: $X = 27$, $K = 3$, $27 \cdot 27 \cdot 27 = 19683 \rightarrow$ ultima cifră este 3. Prima cifră a lui 27 este 2, deci rezultatul este $2 \cdot 3 = 6$.

Al doilea exemplu Se rezolvă cerinţa 2: $X = 13$, $K = 3$. Se fac urmatoărele transformări:

- $13 \cdot 5 = 65$, scoatem zerourile si rotim $\rightarrow 56$,
- $56 \cdot 2 = 112$, scoatem zerourile și rotim $\rightarrow 211$,
- $211 \cdot 5 = 1055$, scoatem zerourile $\rightarrow 155$, rotim $\rightarrow 551$.

Al treilea exemplu Se rezolvă cerinţa 2: $X = 42$, $K = 1782321$. După ce se fac cele K transformări se ajunge la numărul 12.

2.2 Clasa a VI-a

2.2.A Problema Iluminat

Primarul orașului X dorește să aibă un iluminat public modern. Pentru aceasta, realizează o schiță sub forma unui pătrat cu n linii și n coloane în care fiecare element situat la intersecția unei linii cu o coloană reprezintă un cartier.

Primarul a calculat pentru fiecare cartier care este numărul de stâlpi de iluminat public din acel cartier. Fiecare stâlp are un singur bec care inițial este aprins. Acesta a observat un lucru interesant: toate cartierele au un număr diferit de stâlpi de iluminare, iar valoarea maximă a numărului de stâlpi dintr-un cartier este n^2.

Pentru a fi realizată într-un mod cât mai eficient, stingerea becurilor se realizează în următorul mod:

- în prima etapă se sting becurile din cartierul cu număr maxim de stâlpi de iluminat, ceea ce duce la stingerea becurilor din cartierele de pe aceeași linie precum și din cele pe aceeași coloană cu cartierul cu număr maxim de stâlpi.
- procedeul se reia la fiecare etapă pentru toate cartierele în care nu au fost stinse becurile, până când rămâne un singur cartier iluminat.

Cerință

Cunoscând numerele naturale nenule n și k, precum și numărul de stâlpi de iluminat din fiecare cartier, să se determine:

1. Câți stâlpi de iluminat se află în cartierul cu număr maxim de stâlpi de iluminat la etapa cu numărul k din procedeul de stingere a becurilor?
2. Câte becuri se sting, în total, la etapa cu numărul k?
3. Care este numărul maxim de becuri aprinse într-o zonă pătratică a orașului de dimensiune $k \times k$, înainte de a începe stingerea becurilor?

Date de intrare

Fişierul de intrare `iluminat.in` conţine pe prima linie o cifră c (1, 2 sau 3), reprezentând cerința cerută.

Pe linia următoare se găsesc două numere naturale nenule n și k, separate printr-un spațiu.

Pe următoarele n linii se află n^2 numere naturale distincte, câte n pe fiecare linie, separate prin câte un spațiu, cu semnificația din enunț.

Date de ieșire

În fișierul `iluminat.out` se va afișa răspunsul în funcție de cerință:

- dacă $c = 1$ se va afișa pe prima linie un singur număr reprezentând numărul de stâlpi de iluminat din cartierul cu număr maxim de stâlpi de iluminat la etapa k,
- dacă $c = 2$ se va afișa pe prima linie un singur număr reprezentând câte becuri se sting, în total, la etapa cu numărul k,
- dacă $c = 3$ se va afișa numărul maxim de becuri aprinse într-o zonă pătratică de dimensiune $k \times k$ înainte de stingerea becurilor.

Restricții și precizări

- $c \in \{1, 2, 3\}$.
- $1 \leq k < n \leq 1000$.
- Numărul de becuri din fiecare cartier este mai mic sau egal cu n^2.

#	Punctaj	Restricții
1	28	$c = 1$
2	36	$c = 2$
3	36	$c = 3$

Exemple

iluminat.in	iluminat.out
1 4 2 1 2 3 4 16 13 5 6 12 9 7 14 10 11 8 15	15
2 4 2 1 2 3 4 16 13 5 6 12 9 7 14 10 11 8 15	52
3 4 2 1 2 3 4 16 13 5 6 12 9 7 14 10 11 8 15	50

Explicații

Primul exemplu Cerința este 1. Se sting becurile din cartierul având 16 stâlpi de iluminat, ceea ce duce la stingerea becurilor de pe stâlpii din linia 2 și din coloana 1. Tabloul devine:

0	2	3	4
0	**0**	**0**	**0**
0	9	7	14
0	11	8	15

La etapa a doua, primul cartier în care se sting becurile are 15 stâlpi de iluminat.

Al doilea exemplu Cerința este 2. Se sting becurile din cartierul având 16 stâlpi de iluminat, ceea ce duce la stingerea becurilor de pe stâlpii din linia 2 și din coloana 1. Tabloul devine:

0	2	3	4
0	**0**	**0**	**0**
0	9	7	14
0	11	8	15

La etapa a doua, primul cartier în care se sting becurile are 15 stâlpi de iluminat, ceea ce duce la stingerea becurilor din linia 4 și din coloana 4 din noul tablou. Acesta devine:

0	2	3	**0**
0	0	0	**0**
0	9	7	**0**
0	**0**	**0**	**0**

Numărul total de becuri stinse la etapa cu numărul $k = 2$ este: $15 + 11 + 8 + 14 + 4 = 52$.

Al treilea exemplu Cerința este 3. Numărul maxim de becuri aprinse într-o zonă pătratică a orașului de dimensiune 2×2 este 50, în zona cu colțul stânga-sus pe linia 2 și coloana 2

16	13
12	9

2.2.B Problema Inundație

Fie un șir de N coloane de ciment (pozițiile lor fiind numerotate de la 1 la N) de aceeași lățime și diverse înălțimi. Ele sunt încadrate la stânga (poziția 0) și la dreapta (poziția $N+1$) de ziduri foarte înalte. Apa începe să curgă de deasupra primei coloane, câte o pătrățică de apă pe secundă. Apa se acumulează dacă are pereți în stânga și în dreapta, altfel curge mai jos către dreapta. Deasupra fiecărei coloane de ciment se poate forma astfel o coloană de apă, cu înălțimea egală cu numărul de pătrățele de la nivelul apei până la zona de contact cu coloana de ciment.

Cerință

1. Care este înălțimea H a celei mai înalte coloane de apă după ce apa a ajuns peste tot la înălțimea celei mai înalte coloane de ciment?
2. Care este numărul T de secunde în care apa ajunge să acopere coloana numărul P?
3. Care este poziția D a celei mai din dreapta coloane acoperită de apă după S secunde?
4. Care este poziția R a celei mai din stânga coloane pe care o putem reduce cu o unitate astfel încât apa să ajungă cât mai repede la coloana P?

Date de intrare

Fișierul de intrare `inundatie.in` conține pe prima linie un număr natural C, reprezentând cerința ce trebuie rezolvată (1, 2, 3, sau 4).

Pe a doua linie numerele N, P și S despărțite prin câte un spațiu cu semnificația din enunț.

Pe a treia linie se găsesc N numere naturale separate prin câte un spațiu ce reprezintă înălțimile coloanelor.

Date de ieșire

Fișierul de ieșire `inundatie.out` va conține un singur număr, astfel:

1. Dacă $C = 1$: înălțimea H cu semnificația de mai sus.
2. Dacă $C = 2$: timpul T cu semnificația de mai sus.
3. Dacă $C = 3$: poziția D cu semnificația de mai sus.
4. Dacă $C = 4$: poziția R cu semnificația de mai sus.

Restricții și precizări

- $C \in \{1, 2, 3, 4\}$.
- $3 \leq N \leq 100\,000$.
- $1 \leq$ înălțimea oricărei coloane din șir $\leq 20\,000$.
- $1 \leq P \leq N$.
- $1 \leq S \leq 100\,000$.
- O coloană de înălțime h este acoperită de apă dacă apa a ajuns la înălțimea h.

#	Punctaj	Restricții
1	11	$C = 1$
2	25	$C = 2$
3	33	$C = 3$
4	31	$C = 4$

Exemple

inundatie.in	inundatie.out
1 32 15 45 8 5 5 4 3 3 7 5 4 3 3 5 4 3 4 5 6 5 4 4 3 4 5 4 3 2 1 2 3 4 5 9	8
2 32 15 45 8 5 5 4 3 3 7 5 4 3 3 5 4 3 4 5 6 5 4 4 3 4 5 4 3 2 1 2 3 4 5 9	21
3 32 15 45 8 5 5 4 3 3 7 5 4 3 3 5 4 3 4 5 6 5 4 4 3 4 5 4 3 2 1 2 3 4 5 9	29
4 32 15 45 8 5 5 4 3 3 7 5 4 3 3 5 4 3 4 5 6 5 4 4 3 4 5 4 3 2 1 2 3 4 5 9	7

Explicații

Toate exemplele se referă la figura de mai sus, diferă doar numărul cerinței. În toate $N = 32$, $P = 15$ și $S = 45$.

Primul exemplu Linia portocalie de înălțime 9 este nivelul apei la momentul când toate coloanele devin acoperite de apă. Cea mai înaltă coloană de apă este cea cu numărul 27, având 8 pătrățele de apă.

Al doilea exemplu În imaginea de mai sus, liniile roșii arată nivelurile apei la momentul când apa acoperă coloana de la poziția $P = 15$. Observăm că sunt 21 de pătrățele de apă sub linie, deci este nevoie de 21 de secunde pentru a acoperi coloana 15.

Al treilea exemplu Linia verde arată nivelul apei după 42 de secunde. Ea acoperă coloana numărul 29. Lăsând apa să curgă încă 3 secunde (până la cele $S = 45$ secunde) nivelul nu se ridică suficient pentru a ajunge la coloana 30 deoarece ar mai fi nevoie de 5 pătrățele de apă, adică încă 5 secunde.

Al patrulea exemplu Cea mai din stânga coloană pe care o vom reduce cu unu este coloana numărul 7. Astfel apa va ajunge cu 5 secunde mai devreme la coloana $P = 15$. Linia maro (linia de apă de înălțime 6 care se întinde de la coloana 2 la coloana 6) arată cele 5 pătrățele cu care reducem timpul. Orice altă coloană am reduce nu va ajunge așa repede.

2.2.C Problema Șiruri

Se citește un număr natural N și un șir de N numere naturale $a_1, a_2, \ldots, a_N$. Numerele din șir nu conțin cifra 0. Începând de la primul număr din șir către ultimul se vor efectua următoarele modificări:

- dacă ultima cifră a unui număr este egală cu prima cifră a următorului număr din șir cele două numere se unesc, cel de-al doilea lipindu-se de primul. Acest număr nou format se transformă, oprindu-se doar o dată fiecare cifră care apare în număr: cea mai din stânga apariție a cifrei se păstrează, următoarele apariții fiind eliminate. De exemplu, putem uni numerele 21245 și 51278 rezultând numărul 2124551278. Se iau cifrele o singură dată rezultând 214578. Numărul nou format se poate uni la rândul lui cu următorul și așa mai departe.
- dacă ultima cifră a unui număr nu este egală cu prima cifră a următorului număr din șir cele două numere nu se unesc, dar primul număr din cele două se va transforma, păstrându-se doar o dată fiecare cifră care apare în număr: cea mai din stânga apariție a cifrei se păstrează, următoarele apariții fiind eliminate.

Cerință

Dându-se cele N numere din șir să se determine:

1. Câte numere din *șirul inițial* nu au nevoie de transformare (conțin doar cifre distincte)?
2. Câte numere va conține șirul după realizarea tuturor operațiilor de unire?
3. Care este numărul maxim de cifre ale unui număr din noul șir și câte numere au acest număr maxim de cifre?

Date de intrare

Fişierul de intrare `siruri.in` conţine pe prima linie un număr natural c (1, 2 sau 3).

Pe a doua linie se găsește un număr natural nenul N.

Pe a treia linie se află N numere naturale separate de câte un spațiu reprezentând șirul inițial.

Date de ieșire

În fișierul de ieșire `siruri.out` se va afla în funcție de cerința dată:

- dacă $c = 1$, se va afișa pe prima linie numărul de numere ce nu au nevoie de transformare;
- dacă $c = 2$, se va afișa pe prima linie numărul de numere din șir după realizarea tuturor operațiilor de unire;
- dacă $c = 3$, se vor afișa pe prima linie două numere separate printr-un singur spatiu, reprezentând numărul maxim de cifre ale unui număr după efectuarea operațiilor de unire, respectiv numărul de astfel de numere cu număr maxim de cifre.

Restricții și precizări

- $c \in \{1, 2, 3\}$.
- $1 \leq N \leq 100\,000$.
- $1 \leq a_i \leq 1\,000\,000\,000$.
- a_i conține doar cifre nenule, pentru oricare $1 \leq i \leq n$.

#	Punctaj	Restricții
1	18	$c = 1$
2	40	$c = 2$
3	42	$c = 3$

Exemple

siruri.in	siruri.out
1 8 21245 51278 869 33447 723 397897 545786 6783221	3
2 8 21245 51278 869 33447 723 397897 545786 6783221	4
3 9 21245 51278 869 33447 723 397897 545786 6783221 235788946	8 3

Explicații

În primul exemplu cerința este 1. Sunt 8 numere în șir, dintre care doar 3 au cifre distincte: $[51278, 869, 723]$.

În al doilea exemplu cerința este 2. Sunt 8 numere în șir, după transformări șirul va arăta astfel: $[21457869, 3472, 3978, 54786321]$.

În al treilea exemplu cerința este 3. Sunt 9 numere în șir, după transformări șirul va arăta astfel: $[21457869, 3472, 3978, 54786321, 23578946]$. Numărul maxim de cifre ale unui număr din noul șir este 8 și sunt 3 numere în noul șir care au 8 cifre: $[21457869, 54786321, 23578946]$.

2.3 Clasa a VII-a

2.3.A Problema Microbuz

O companie de transport cu microbuze din județul Iași a adoptat o strategie proprie pentru rutele din județ:

- niciun traseu nu poate avea mai mult de 165 kilometri,
- distanța între două stații consecutive este de un kilometru,
- un pasager poate pleca din orice stație şi poate să își cumpere bilete pentru parcurgerea a $1, 2, \ldots, 10$ kilometri,
- fiecare dintre cele zece distanţe posibile au bilete cu preţuri distincte.

De exemplu:

1 km	2 km	3 km	4 km	5 km	6 km	7 km	8 km	9 km	10 km
11 lei	14 lei	18 lei	23 lei	29 lei	36 lei	44 lei	45 lei	53 lei	64 lei

Gigel, care călătoreşte cu microbuzul exact N kilometri, poate să aleagă una sau mai multe distanţe dintre cele zece stabilite și să cumpere biletele corespunzătoare. Compania impune ca un pasager să poată cumpăra maxim 3 bilete de același tip.

Gigel observă că pentru aceeași sumă poate achiziționa seturi diferite de bilete *cu prețuri distincte*. Pentru exemplu de mai sus, cu 98 de lei el poate cumpăra câte un bilet cu prețurile 11, 14, 29, 44 lei sau câte un bilet cu prețurile 45, 53 lei ($11 + 14 + 29 + 44 = 45 + 53$).

Cerințe

Scrieţi un program care să rezolve următoarele cerințe:

1. determină preţul minim al unui set de bilete ce poate fi achiziționat pentru parcurgerea a exact N kilometri;
2. determină distanţele alese de Gigel, astfel încât preţul total al călătoriei să fie minim;
3. determină două seturi distincte de bilete pentru care prețul total al biletelor este același și este cel mai mare posibil. Pentru fiecare set nu este permisă alegerea mai multor bilete cu același preț și nu există două bilete cu același preț în ambele seturi.

Date de intrare

Fişierul de intrare `microbuz.in` are trei linii. Pe prima linie se află un număr natural C ce reprezintă cerința (1, 2 sau 3). Linia a doua conține zece valori naturale ordonate *strict crescător*, separate prin câte un spaţiu, reprezentând preţurile pentru parcurgerea a $1, 2, \ldots, 10$ kilometri. Linia a treia conţine valoarea N reprezentând distanţa pe care doreşte să o parcurgă Gigel.

Date de ieșire

Fişierul de ieşire `microbuz.out` are următoarea structură:

1. dacă $C = 1$, pe prima linie se va afișa un întreg reprezentând costul minim al călătoriei;
2. dacă $C = 2$, pe fiecare linie se vor afișa câte două numere întregi, separate printr-un spaţiu, reprezentând distanța parcursă şi preţul biletului corespunzător;
3. dacă $C = 3$, pe prima linie se va afișa prețul comun al celor două seturi de bilete, iar pe următoarele două linii câte un set de bilete dat prin numărul de km pentru biletele din set, în ordine strict crescătoare, separate prin câte un spațiu.

Restricții și precizări

- $C \in \{1, 2, 3\}$.
- $1 \leq N \leq 165$.
- cele 10 prețuri sunt numere naturale din intervalul $[10, 99]$.
- răspunsul la cerința 3 nu depinde de valoarea lui N.

#	Punctaj	Restricții
1	28	$C = 1$
2	38	$C = 2$
3	34	$C = 3$

Exemple

microbuz.in	microbuz.out	Explicații
1 11 14 18 23 29 36 44 45 53 64 15	86	Cerința este 1. Costul minim total este 86
2 11 14 18 23 29 36 44 45 53 64 15	3 18 4 23 8 45	Cerința este 2. Biletele cumpărate și prețurile lor sunt: 3 km parcurși, preț 18 4 km parcurși, preț 23 8 km parcurși, preț 45

microbuz.in	**microbuz.out**	**Explicații**
2 13 17 18 19 21 22 25 28 31 37 39	7 25 7 25 8 28 8 28 9 31	Cerința este 2. Biletele cumpărate și prețurile lor sunt: 7 km parcurși, preț 25 7 km parcurși, preț 25 8 km parcurși, preț 28 8 km parcurși, preț 28 9 km parcurși, preț 31
3 11 14 18 23 29 36 44 45 53 64 15	163 2 3 4 7 10 5 6 8 9	Cerința este 3. Prețul maxim comun este 163 Setul 1: câte un bilet cu prețurile 14 18 23 44 64 Setul 2: câte un bilet cu prețurile 29 36 45 53

2.3.B Problema Raza

Pe planeta Quadratia, locuitorii folosesc forme pătrate în tot ceea ce construiesc. Ei au trimis pe solul planetei N mașini speciale de apărare, numite *rovere*, care se deplasează pe traiectorii ce descriu pătrate. Harta planetei poate fi privită ca un tablou bidimensional cu număr infinit de linii și coloane, numerotarea acestora începând cu 1. Fiecare element al acestui tablou se identifică prin numărul liniei, respectiv numărul coloanei pe care se găsește.

Fiecare rover este descris prin 3 numere naturale nenule L, C și R, reprezentând poziția (linia L și coloana C) elementului din colțul din stânga-sus al pătratului ce descrie traiectoria, respectiv lungimea R a laturii acestuia. Toate roverele pornesc inițial din elementul din colțul stânga-sus al pătratului ce descrie traiectoria, pe care o parcurg în sensul acelor de ceasornic, traversând câte un element pe secundă. Ele sunt programate să parcurgă această traiectorie fără a se opri. Este posibil ca mai multe rovere să se afle, la un moment dat, la aceeași poziție de pe hartă.

Rectilinienii sunt singurii dușmani ai quadratienilor, ei deosebindu-se de aceștia prin folosirea consecventă a liniilor drepte în tot ceea ce construiesc. Rectilinienii au construit o armă laser de mare putere, pe care vor să o folosească la distrugerea roverelor quadratiene. Arma a fost adusă în poziția $(1,1)$ de pe harta planetei, adică elementul de pe prima linie și prima coloană. Arma poate emite *o singură* rază distructivă, timp de o secundă, dezafectând în acest timp toate roverele aflate în secunda respectivă pe diagonala principală a tabloului care descrie harta planetei. Arma nu poate fi detectată în primele S secunde. Traiectoria roverelor poate trece prin poziția în care a fost amplasată arma, fără a o putea detecta în primele S secunde.

În imaginea alăturată avem două rovere, acestea fiind identificate prin tripletele de numere $4,2,6$, respectiv $6,9,4$.

Traiectoria primului rover începe din poziția $(4,2)$ și are latura 6. Numerele de pe traiectorie reprezintă secunda la care se parcurge respectivul element al tabloului. Roverul va ajunge din nou în poziția inițială la secundele 21, 41, 61, etc. Primul rover intersectează, la prima sa trecere, diagonala principală în două puncte: $(4,4)$ la secunda 3 și $(7,7)$ la secunda 9. Al doilea rover intersectează diagonala principală într-un singur punct, $(9,9)$ la secundele 10, 22, 34, etc.

Cerințe

Scrieţi un program care să rezolve următoarele cerințe:

1. Determină numărul roverelor a căror traiectorie se intersectează cu diagonala principală a tabloului ce descrie harta.
2. Determină numărul maxim de rovere, care pot fi distruse simultan, într-o singură secundă, înainte de trecerea celor S secunde, precum și secunda minimă în care se poate realiza acest lucru.

Date de intrare

Datele de intrare se citesc din fișierul `raza.in`, care are următoarea structură:

- pe prima linie se află numărul natural $T \in \{1,2\}$, semnificând numărul cerinței de rezolvat.
- pe a doua linie se află numerele naturale nenule N și S, separate printr-un spațiu, reprezentând numărul roverelor, respectiv numărul de secunde în care arma nu este detectată.

- pe următoarele N linii se află câte 3 numere naturale nenule, L, C, R, separate prin câte un spațiu, reprezentând coordonatele poziției inițiale, respectiv lungimea laturii traiectoriei fiecărui rover.

Date de ieșire

Datele de ieșire se vor afișa în fișierul `raza.out`, care are următoarea structură în funcție de cerință:

- dacă $T = 1$, pe prima linie se va afișa numărul roverelor a căror traiectorie se intersectează cu diagonala principală;
- dacă $T = 2$, pe prima linie se vor afișa două numere naturale, separate printr-un spațiu, reprezentând numărul maxim de rovere dezactivate și secunda minimă în care se poate realiza acest lucru.

Restricții și precizări

- $1 \leq N \leq 10\,000$.
- $1 \leq S \leq 100\,000$.
- $1 \leq L, C \leq 50\,000$.
- $3 \leq R \leq 1\,000$.
- $1 \leq$ Numărul maxim de rovere care se intersectează cu raza armei $\leq 1\,000$.

#	Punctaj	Restricții
1	28	$T = 1$
2	72	$T = 2$

Exemple

raza.in	raza.out	Explicații
1 2 100 4 2 6 6 9 4	2	Cerința este 1. Exemplul reprezintă desenul din enunț.
2 2 5 4 2 6 6 9 4	1 3	Cerința este 2. Este distrus doar primul rover, momentul în care se distruge acest rover este 3.

raza.in	raza.out	Explicații
1 5 10000 3 3 3 4 7 4 6 4 3 9 2 3 9 7 5	4	Cerința este 1. Sunt patru rovere care intersectează diagonala.
2 5 10000 3 3 3 4 7 4 6 4 3 9 2 3 9 7 5	2 3	Cerința este 2. Numărul maxim de rovere, care pot fi distruse este 2. Acest lucru se poate realiza cel mai devreme la secunda 3.

2.3.C Problema Text

Alexandra citește un text format din litere mici și mari ale alfabetului englez și spații. Fiind interesată de criptografie, ea elimină toate spațiile și apoi încadrează literele, în ordinea în care acestea apar în text, într-un tablou bidimensional, în care numărul de linii este mai mic sau egal decât numărul de coloane.

Întrucât pot exista mai multe moduri de încadrare a textului, Alexandra îl alege pe cel pentru care diferența absolută dintre numărul liniilor și al coloanelor tabloului este minimă. Completarea cu literele textului a tabloului bidimensional se face în ordinea crescătoare a liniilor și în cadrul fiecărei linii, în ordinea crescătoare a coloanelor.

Cerințe

1. Notând cu N numărul de linii și cu M numărul de coloane ale tabloul bidimensional obținut, afișați elementele acestuia pe primele N linii ale fișierului de ieșire, fiecare linie conținând exact M litere fără spații între ele. Afișarea se va face în ordinea crescătoare a indicilor liniilor și pe fiecare linie în ordinea crescătoare a indicilor coloanelor.
2. Determinați cel mai lung palindrom de pe o linie sau de pe o coloană a tabloului obținut. În cazul în care există mai multe palindroame de aceeași lungime, se va afișa cel care este cel mai mare din punct de vedere lexicografic conform codului ASCII.
3. Determinați care este numărul maxim de elemente dintr-un subtablou dreptunghiular, ce conține doar vocale.

Date de intrare

Datele de intrare se citesc din fișierul `text.in`, care are următoarea structură:

- pe prima linie se află numărul natural C care poate avea doar valorile 1, 2 sau 3, semnificând numărul cerinței de rezolvat;
- pe a doua linie se află textul pe care îl va prelucra Alexandra.

Date de ieșire

Datele de ieșire se vor afișa în fișierul `text.out`, care are următoarea structură:

- dacă $C = 1$, N linii, fiecare conținând câte M litere, repezentând elementele tabloului obținut;
- dacă $C = 2$, o singură linie conținând cel mai lung palindrom determinat;
- dacă $C = 3$, o singură linie conținând un număr natural reprezentând numărul de elemente din subtabloul determinat.

Restricții și precizări

- $2 \leq$ numărul total de caractere din textul inițial al Alexandrei $\leq 200\,000$
- Vocalele sunt: `A`, `E`, `I`, `O`, `U`, `a`, `e`, `i`, `o`, `u`.
- Se garantează că N, numărul de linii ale tabloului obținut este > 1.

#	Punctaj	Restricții
1	16	$C = 1$ și numărul de caractere din șirul inițial $\leq 10\,000$
2	36	$C = 2$ și numărul de caractere din șirul inițial $\leq 1\,000$
3	16	$C = 3$ și numărul de caractere din șirul inițial $\leq 1\,000$
4	16	$C = 3$ și numărul de caractere din șirul inițial $\leq 50\,000$
5	16	$C = 3$, fără restricții suplimentare

Exemple

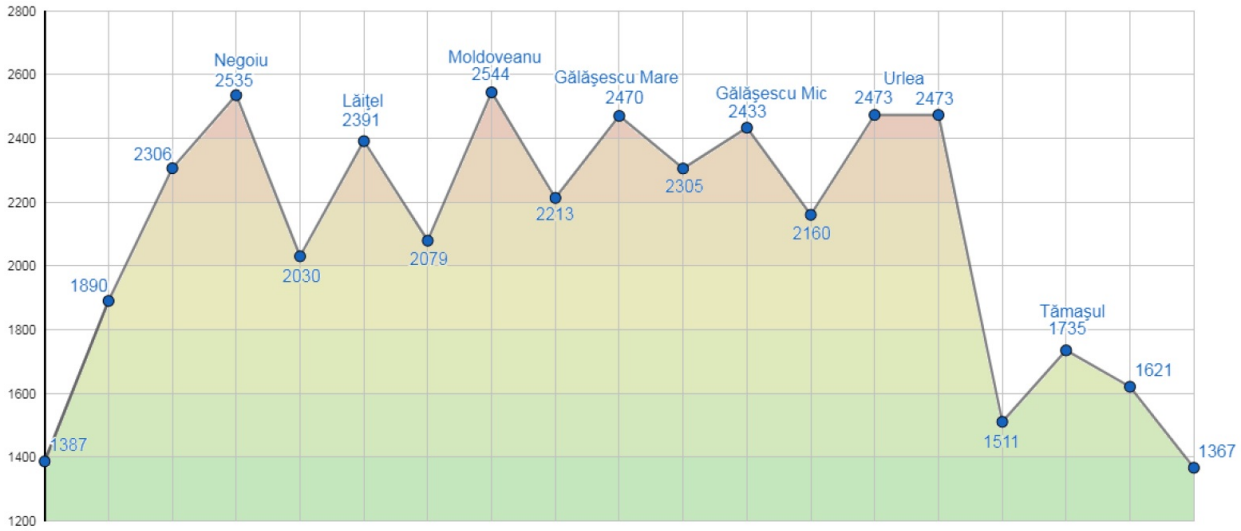
text.in	text.out	Explicații
1 Ana nu are mere	Anan uare mere	Numărul de caractere după eliminare spațiilor este 12. Textul plasat în tablou conține N = 3 linii și M = 4 coloane.
1 A fost odata ca niciodata	Afostod atacani ciodata	Numărul de caractere după eliminare spațiilor este 21. Textul plasat în tablou conține N = 3 linii și M = 7 coloane.
2 A fost odata ca niciodata	oao	Afostod atacani ciodata Tabloul are pe coloana 3 palindromul oao iar pe liniile 2 și 3 palindromul ata.
3 Aceasta oaie e alba	6	Acea stao aiee alba

2.4 Clasa a VIII-a

2.4.A Problema Proeminența

Cea mai cunoscută măsură a unui vârf de munte este altitudinea sa. Vârfuri secundare ale unui munte înalt pot avea altitudini considerabile, dar în general acestea nu sunt atât de relevante. Din acest motiv, geografii au introdus o nouă măsură pentru un vârf: *proeminenţa*.

Un *profil* este o reprezentare a altitudinilor în puncte succesive, între care ne putem deplasa la stânga sau dreapta.



În desen avem profilul munţilor Făgăraş, pe care exemplificăm următoarele definiţii:

1. Un *vârf* este un punct (sau o succesiune de puncte consecutive cu aceeaşi altitudine), pentru care altitudinile punctelor alăturate în stânga, respectiv în dreapta sunt strict mai mici decât altitudinea vârfului. De exemplu, Negoiu este un vârf (altitudinea sa este 2535m, punctul de pe profil situat în stânga sa are altitudinea 2306m, iar punctul de pe profil situat în dreapta sa are altitudinea 2030m). Un alt exemplu de vârf este Urlea, fiind reprezentat pe profil prin două puncte consecutive cu altitudinea 2473m, având în stânga un punct cu altitudinea 2160m, iar în dreapta un punct cu altitudinea 1511m.
2. *Cea mai adâncă vale* dintre două vârfuri este altitudinea minimă a unui punct care se află între cele două vârfuri. De exemplu, în desen între Negoiu şi Moldoveanu cea mai adâncă vale este la 2030m.
3. *Proeminenţa* unui vârf este diferenţa *minimă* dintre altitudinea lui şi *cea mai adâncă* vale dintre el şi un vârf strict mai înalt decât el. Dacă nu există un vârf strict mai înalt, proeminenţa este altitudinea vârfului. De exemplu, pentru a determina proeminenţa vârfului Gălăşescu Mare putem considera vârfurile strict mai înalte (în ordine de la stânga la dreapta: Negoiu, Moldoveanu şi Urlea). Dacă plecăm de pe Negoiu, diferenţa dintre cea mai adâncă vale şi Gălăşescu Mare este $2470 - 2030 = 440$m. Dacă plecăm de pe Moldoveanu, diferenţa

dintre cea mai adâncă vale şi Gălăşescu Mare este $2470 - 2213 = 257$m. Dacă plecăm de pe Urlea, diferenţa dintre cea mai adâncă vale şi Gălăşescu Mare este $2470 - 2160 = 310$m. Diferenţa minimă este 257m, deci proeminenţa vârfului Gălăşescu Mare este dată de vârful Moldoveanu.

Cerinţe

Scrieţi un program care, cunoscând configuraţia unui profil, rezolvă următoarele două cerinţe:
1. determină numărul de vârfuri existente pe profilul respectiv;
2. determină proeminenţa fiecărui vârf de pe profil.

Date de intrare

Fişierul de intrare `proeminenta.in` conţine pe prima linie numărul natural C, reprezentând cerinţa care trebuie să fie rezolvată: 1 sau 2. Pe cea de a doua linie se află numărul natural N, reprezentând numărul de puncte din profil. Pe a treia linie se află N numere naturale separate prin câte un spaţiu $a_1, a_2 \dots a_N$, reprezentând altitudinile celor N puncte de pe profil, în ordine de la stânga la dreapta.

Date de ieșire

Fişierul de ieşire `proeminenta.out` va conţine o singură linie pe care va fi scris răspunsul la cerinţa C.

Dacă $C = 1$ răspunsul va fi numărul natural Nr, reprezentând numărul de vârfuri.

Dacă $C = 2$ răspunsul va fi o succesiune de Nr valori naturale separate prin câte un singur spaţiu, reprezentând proeminenţele celor Nr vârfuri, în ordinea din fişierul de intrare.

Restricții și precizări

- $3 \leq N \leq 10^6$.
- $0 \leq a_i < 2^{31}$, pentru $1 \leq i \leq N$.
- $a_1 < a_2$ și $a_{N-1} > a_N$.

#	Punctaj	Restricții
1	10	Pentru teste valorând 10 de puncte: $C = 1$.
2	30	Pentru teste valorând 30 de puncte: $C = 2$ și $N \leq 1000$.
3	20	Pentru teste valorând 20 de puncte: $C = 2$ și $1000 < N$, iar altitudinile a_i sunt distincte două câte două.

Exemple

proeminenta.in	**proeminenta.out**
1 19 1387 1890 2306 2535 2030 2391 2079 2544 2213 2470 2305 2433 2160 2473 2473 1511 1735 1621 1367	7
2 19 1387 1890 2306 2535 2030 2391 2079 2544 2213 2470 2305 2433 2160 2473 2473 1511 1735 1621 1367	505 312 2544 257 128 313 224

Explicație

Profilul din fişierele de intrare este cel reprezentat în desen. Cele 7 vârfuri sunt denumite pe desen.

Deși altitudinile apar în tabel pe mai multe linii, ele vor fi scrise pe o singură linie în fișierul de intrare.

2.4.B Problema RGB

Ionuţ, tânăr programator, se lansează pe piaţa producătorilor de jocuri pe calculator. Jocul pe care l-a proiectat se numeşte RGB. În joc există N personaje extraterestre. Fiindcă Ionuţ nu este de acord cu teoria omuleţilor verzi, personajele lui sunt de 3 culori:

- R extratereştri de culoare roşie;
- G extratereştri de culoare verde;
- B extratereştri de culoare albastră.

Fiecare extraterestru are o anumită putere, exprimată printr-un număr natural impar, puterile oricăror doi extratereştri fiind diferite.

Pe parcursul jocului fiecare extraterestru va lupta cu fiecare dintre ceilalţi extratereştri. Rezultatul unei lupte între doi extratereştri depinde de puterea acestora, dar şi de culoarea lor.

Într-o luptă dintre doi extratereştri de aceeaşi culoare, va câştiga cel cu puterea cea mai mare.

Într-o luptă între doi extratereştri de culori diferite, puterile lor se modifică după cum urmează, iar după modificare lupta o va câştiga extraterestrul cu puterea mai mare:

- Puterea unui extraterestru roșu este dublată dacă adversarul este un extraterestru verde.
- Puterea unui extraterestru verde este dublată dacă adversarul este un extraterestru albastru.
- Puterea unui extraterestru albastru este dublată dacă adversarul este un extraterestru roșu.

După fiecare luptă, puterile extratereștrilor revin la valorile inițiale, în caz că s-au modificat.

Cerința

Scrieţi un program care, cunoscând culorile şi puterile extratereştrilor, rezolvă următoarele două cerinţe:

1. determină puterea extraterestrului care câştigă cele mai multe lupte; dacă există mai mulţi astfel de extratereştri, se va afişa puterea minimă;
2. determină pentru fiecare extraterestru numărul de lupte câştigate de acesta.

Date de intrare

Fişierul de intrare `rgb.in` va conţine pe prima linie numerele naturale C, R, G și B, unde C este cerinţa care trebuie să fie rezolvată (1 sau 2), R reprezintă numărul de extratereștri roșii, G numărul de extratereștri verzi, iar B numărul de extratereștri albaştri.

Pe cea de a doua linie se află R numere naturale impare în ordine strict crescătoare, reprezentând puterile celor R extratereștri roșii.

Pe cea de a treia linie se află G numere naturale impare în ordine strict crescătoare, reprezentând puterile celor G extratereștri verzi.

Pe cea de a patra linie se află B numere naturale impare în ordine strict crescătoare, reprezentând puterile celor B extratereștri albaştri.

Valorile scrise pe aceeaşi linie sunt separate prin câte un spaţiu.

Date de ieșire

Pentru $C = 1$, fişierul de ieşire `rgb.out` va conţine o singură linie pe care va fi scrisă puterea extraterestrului care câştigă cele mai multe lupte; dacă există mai mulţi extratereştri care câştigă un număr maxim de lupte, se va afişa puterea minimă.

Pentru $C = 2$, fişierul de ieşire `rgb.out` va conţine trei linii. Pe prima linie se va scrie numărul de lupte câştigate de fiecare extraterestru roșu. Pe a doua linie, se va scrie numărul de lupte câştigate de fiecare extraterestru verde. Pe a treia linie, se va scrie numărul de lupte câştigate de fiecare extraterestru albastru. Pentru fiecare culoare, valorile vor fi afişate considerând ordinea extratereştrilor din fişierul de intrare.

Valorile scrise pe aceeaşi linie vor fi separate prin câte un spaţiu.

Restricții și precizări

- $N = R + G + B$.
- $1 \leq R, G, B \leq N - 2$.
- $1 \leq$ puterea oricărui extraterestru $\leq 2 \cdot N - 1$.

#	Punctaj	Restricții
1	21	$C = 1$, iar $1 \leq N \leq 500\,000$
2	18	$C = 2$, iar $1 \leq N \leq 1\,000$
3	25	$C = 2$, iar $1\,000 < N \leq 100\,000$
4	36	$C = 2$, iar $100\,000 < N \leq 500\,000$

Exemple

rgb.in	rgb.out
1 1 2 2 3 1 7 5 9	7
2 1 2 2 3 1 7 5 9	1 0 4 2 3

Explicații

Pentru primul exemplu, $C = 1$, deci se va rezolva prima cerinţă. Există un extraterestru roşu, care are puterea 3, doi extratereştri verzi, având puterile 1, respectiv 7 şi doi extratereştri albaștri cu puterile 5, respectiv 9.

Extraterestrul cu puterea 7 este singurul care va câştiga cele mai multe lupte (în cazul acesta, chiar toate):

- când luptă cu extraterestrul verde cu puterea 1 câştigă, pentru că are puterea mai mare;
- când luptă cu extraterestrul roşu cu puterea 3, acesta îşi va dubla puterea (va avea puterea 6), dar va fi insuficient pentru a câștiga lupta;
- când luptă contra extraterestrului albastru cu puterea 9, va avea puterea dublată (14), prin urmare va câştiga şi această luptă.

Pentru al doilea exemplu, $C = 2$, deci se va rezolva a doua cerinţă.

- Extraterestrul cu puterea 3 poate câştiga doar o luptă (contra extraterestrului cu puterea 1).
- Extraterestrul cu puterea 1 nu poate câştiga nicio luptă.
- Extraterestrul cu puterea 7 poate câștiga toate luptele (vezi explicaţia de la exemplul precedent).
- Extraterestrul cu puterea 5 poate câştiga două lupte (contra extratereştrilor cu puterile 1 şi 3).
- Extraterestrul cu puterea 9 poate câştiga 3 lupte (contra extratereştrilor cu puterile 1, 3 și 5).

2.4.C Problema Subșir

Fie $S = (S_1, S_2, \ldots, S_N)$ un şir format din N numere naturale mai mici decât 10 şi x un număr natural cu p cifre, reprezentat în baza 10 prin $x = x_1 x_2 \ldots x_p$. Observaţi că poziţiile din şirul S sunt numerotate de la stânga la dreapta de la 1 la N, iar cifrele numărului x sunt numerotate de la stânga la dreapta de la 1 la p (cifra p fiind cifra unităţilor).
Vom spune că *x apare ca subșir* în șirul S dacă există pozițiile $1 \leq j_1 < j_2 < \ldots < j_p \leq N$ astfel încât să avem $S_{j_i} = x_i$ pentru orice $1 \leq i \leq p$.
De exemplu, pentru $S = (2, 3, 7, 9)$, 27 apare ca subșir în S, dar 93 nu apare ca subșir în S.
Numim *prefix de lungime j al lui S* secvența $S_1, S_2, \ldots, S_j$ (secvenţa care începe la poziţia 1 şi se termină la poziţia j, $1 \leq j \leq N$).

Cerinţe

Scrieţi un program care, cunoscând şirul S, rezolvă următoarele două cerinţe:

1. fiind date Nr perechi de forma x j, determină pentru fiecare pereche dacă numărul x apare ca subșir în prefixul de lungime j al lui S şi afişează valoarea 1 în caz afirmativ, respectiv valoarea 0 în caz contrar;
2. fiind date Nr perechi de forma a b, determină şi afişează pentru fiecare pereche numărul de numere naturale din intervalul închis $[a, b]$ care apar ca subșir în șirul S.

Date de intrare

Fişierul de intrare `subsir.in` conţine pe prima linie un număr natural C care reprezintă cerinţa care urmează a fi rezolvată (1 sau 2). Pe a doua linie se află numărul natural N. Pe a treia linie se află N numere naturale mai mici decât 10, $S_1, S_2, \ldots, S_N$, reprezentând elementele şirului S. Pe linia a patra se află un număr natural Nr care reprezintă numărul de perechi. Pe următoarele Nr linii se află câte o pereche de numere naturale. Numerele scrise pe aceeaşi linie în fişierul de intrare sunt separate prin câte un spaţiu.

Date de ieșire

Fişierul de ieşire `subsir.out` va conţine Nr linii, pe cea de a i-a linie fiind scris rezultatul pentru cea de a i-a pereche dintre cele Nr perechi din fişierul de intrare, conform cerinţei C.

Restricții și precizări

- $1 \leq N \leq 10^4$.
- $1 \leq Nr \leq 10^4$.
- $0 \leq x < 10^7$.
- $1 \leq j \leq N$.
- $0 \leq a \leq b < 10^7$.

#	Punctaj	Restricții
1	30	Pentru teste valorând 30 de puncte cerinţa este 1.
2	70	Pentru teste valorând 70 de puncte cerinţa este 2.

Exemple

subsir.in	subsir.out
1 10 5 6 1 0 3 2 5 2 3 1 5 1 4 1 2 153 6 153 9 72 8	1 0 0 1 0
2 10 5 6 1 0 3 2 5 2 3 1 3 0 20 40 105 81 99	11 15 0

Explicații

Pentru primul exemplu, cerinţa este 1, deci trebuie să verificăm pentru fiecare dintre cele $Nr = 5$ perechi x j specificate dacă x apare ca subşir în prefixul de lungime j al lui S.

Prima pereche este $(1, 4)$. Numărul 1 apare ca subşir în prefixul 5, 6, 1, 0, deci se afişează 1.

A doua pereche este $(1, 2)$. Numărul 1 nu apare ca subşir în prefixul 5, 6, deci se afişează 0.

A treia pereche este $(153, 6)$. Numărul 153 nu apare în prefixul 5, 6, 1, 0, 3, 2, deci se va afişa 0.

A patra pereche este $(153, 9)$. De data aceasta numărul 153 apare ca subşir în prefixul 5, 6, 1, 0, 3, 2, 5, 2, 3, deci se afişează 1.

A cincea pereche este $(72, 8)$. Numărul 72 nu apare ca subşir în niciun prefix al lui S, deci nici în cel de lungime 8 și se va afişa 0.

Pentru al doilea exemplu cerinţa este 2, deci trebuie să determinăm pentru fiecare dintre cele $Nr = 3$ perechi a b specificate câte numere naturale din intervalul $[a, b]$ apar ca subşir în S.

Pentru intervalul $[0, 20]$ există 11 numere, acestea fiind 0, 1, 2, 3, 5, 6, 10, 11, 12, 13, 15.

Pentru intervalul $[40, 105]$ există 15 numere, acestea fiind 50, 51, 52, 53, 55, 56, 60, 61, 62, 63, 65, 101, 102, 103, 105.

Pentru intervalul $[81, 99]$ răspunsul este 0, deoarece nu există niciun număr în acest interval care să fie subşir al lui S.

2.5 Clasa a IX-a

2.5.A Problema Colibri

Se dau N triplete de numere naturale (a_i, b_i, c_i), unde $c_i \neq 0$ și $1 \leq i \leq N$, fiecare reprezentând câte un număr rațional q_i egal cu

$$\frac{(-1)^{a_i} b_i}{c_i}.$$

Cerință

Găsiți un subșir nevid al șirului $q_1, q_2, \ldots, q_N$ al cărui produs al valorilor să fie maxim posibil.

Date de intrare

Fișierul de intrare `colibri.in` conține pe prima linie numărul N. Următoarele N linii descriu cele N triplete: pe linia i se află numerele naturale a_i, b_i, c_i, separate prin spații.

Date de ieșire

Pe prima linie a fișierului de ieșire `colibri.out` se află un șir de N cifre. Cifra i, unde $1 \leq i \leq N$, este 1 dacă și numai dacă q_i este selectat în subșirul soluție, altfel este 0. Cifrele șirului *nu* se vor separa prin spații.

Restricții și precizări

- $1 \leq N \leq 50\,000$.
- $0 \leq a_i, b_i \leq 1\,000\,000$, oricare ar fi $1 \leq i \leq N$.
- $1 \leq c_i \leq 1\,000\,000$, oricare ar fi $1 \leq i \leq N$.
- Dacă există mai multe soluții, atunci se acceptă orice soluție corectă.
- Spunem că un șir x este subșir al unui șir y dacă și numai dacă x se poate obține din y eliminând o parte din elementele lui y (inclusiv nici unul) fără a schimba ordinea relativă a elementelor rămase.

#	Punctaj	Restricții
1	30	$N \leq 19$ și $a_i, b_i, c_i \leq 9$
2	20	$N \leq 19$
3	20	$a_i, b_i, c_i \leq 9$
4	30	Fără restricții suplimentare

Exemple

colibri.in	colibri.out
5 0 0 1 2 4 2 4 7 7 1 2 3 0 3 2	01001

Explicație

În exemplu $N = 5$, $q_1 = \frac{0}{1}$, $q_2 = \frac{4}{2}$, $q_3 = \frac{7}{7}$, $q_4 = -\frac{2}{3}$ și $q_5 = \frac{3}{2}$.

Produsul maxim posibil este egal cu 3. Acesta se poate obține luând fie subșirul constând din numerele q_2 și q_5, fie luând subșirul format din numerele q_2, q_3 și q_5. Cu alte cuvinte, și răspunsul `01101` este corect.

2.5.B Problema Geogra

Pasionați de geografie, Alex și Răzvan joacă online Geoguessr.

Harta lumii este alcătuită din N locații numerotate de la 1 la N, fiecare desemnând un punct în plan de coordonate (X_i, Y_i). Alex a studiat atent toate cele N locații și a determinat o listă de L caracteristici de interes pentru locațiile date, numerotate de la 1 la L. De exemplu, caracteristica 1 ar putea fi „se află respectiva locație în Europa?", iar caracteristica 2 ar putea fi „se vorbește limba spaniolă în locația respectivă?", și așa mai departe. Pentru fiecare locație i și fiecare caracteristică j se consideră cunoscută valoarea $Z_{i,j} \in \{0, 1\}$ cu proprietatea că $Z_{i,j} = 1$ dacă și numai dacă locația i are caracteristica j. De exemplu, dacă locația 1 se află în Asia și acolo se vorbește limba spaniolă, atunci am avea $Z_{1,1} = 0$ și $Z_{1,2} = 1$.

Un joc de Geoguessr este format din Q runde. La o rundă, calculatorul alege o locație i din cele N, fără a o dezvălui lui Alex. În schimb, Alex primește câteva ilustrații cu locul respectiv, scopul lui Alex fiind acela de a descoperi locația i. Analizând atent doar ilustrațiile, Alex poate determina pentru orice caracteristică j dacă locația i o are sau nu. Totuși, timpul pentru fiecare rundă fiind limitat, Alex nu va reuși mereu să afle în timp util valorile $Z_{i,j}$ pentru toate caracteristicile j, ci doar pentru unele. Așadar, dorind să fie cât mai eficient, Alex și-a format următoarea strategie de joc, moștenită de la Bunicul lui: prima dată Alex va afla valoarea Z_{i,T_1}, apoi, dacă a mai rămas timp, Alex va afla valoarea Z_{i,T_2}, apoi valoarea Z_{i,T_3}, și așa mai departe până când expiră timpul alocat rundei. În funcție de limita de timp a rundei (care poate varia de la o rundă la alta), cu această strategie este posibil ca Alex să afle mai multe sau mai puține informații despre locație, inclusiv niciuna (adică nicio valoare $Z_{i,j}$ descoperită). Vom nota cu $R_1, R_2, \ldots, R_L$ informațiile descoperite de Alex până la finalul rundei. Mai exact, $R_j = Z_{i,j}$ dacă Alex a apucat să afle dacă locația i are caracteristica j în timp util, sau $R_j = -1$ altfel.

Atenție! Strategia lui Alex, reprezentată de șirul $T_1, T_2, \ldots, T_N$, este *aceeași* pentru toate cele Q runde. Șirul $T_1, T_2, \ldots, T_N$ este format din valori distincte și $T_1, T_2, \ldots, T_N \in \{1, 2, \ldots, N\}$.

Cerințe

La finalul unei runde este posibil ca mai multe locații din cele N să corespundă cu șirul R de informații aflate de Alex, așa că acesta își pune două întrebări existențiale:

1. Care este numărul K de locații dintre cele N care respectă șirul R de informații aflate pe parcursul rundei? Spunem că o locație i respectă șirul R dacă pentru orice caracteristică j avem ca $R_j = Z_{i,j}$ sau $R_j = -1$.
2. Se dă câte o valoare W_i pentru fiecare locație i din cele N, $1 \leq i \leq N$. Considerând locațiile care respectă șirul R, fie acestea $1 \leq i_1, i_2, \ldots, i_K \leq N$, pentru un punct P de coordonate întregi (A, B) din plan, nu neapărat unul corespunzător unei locații dintre cele N, definim

$$d_P = W_{i_1}\left(|A - X_{i_1}| + |B - Y_{i_1}|\right) + W_{i_2}\left(|A - X_{i_2}| + |B - Y_{i_2}|\right) + \ldots + W_{i_K}\left(|A - X_{i_K}| + |B - Y_{i_K}|\right).$$

   Care este punctul P din plan pentru care d_P este minim? Dacă sunt mai multe astfel de puncte P, se cere cel cu A minim. Dacă încă este egalitate, atunci se cere cel cu B minim.

Se dă un număr $C \in \{1, 2\}$. Pentru $C = 1$ să se afișeze răspunsul la prima întrebare a lui Alex pentru fiecare din cele Q runde. Pentru $C = 2$ să se afișeze răspunsul la a doua întrebare a lui Alex pentru fiecare din cele Q runde.

Date de intrare

Pe prima linie se află numărul natural C, reprezentând cerința care trebuie rezolvată.

Pe cea de-a doua linie se află numerele naturale N și L, reprezentând numărul locațiilor și, respectiv, numărul caracteristicilor studiate de Alex.

Urmează descrierile celor N locații, în ordine de la 1 la N, fiecare pe câte doua linii. Descrierea unei locații i se face după cum urmează:

- Pe prima linie se afla numerele naturale X_i și Y_i. Dacă $C = 2$, în continuarea acestor doua valori se află numărul natural W_i. Valorile de pe linie sunt separate prin spații.
- Pe a doua linie se afla $Z_{i,1}, Z_{i,2}, \ldots, Z_{i,L}$, separate prin spații.

Pe următoarea linie se află numărul natural Q, reprezentând numărul de runde din cadrul jocului.

Urmează descrierile celor Q runde, în ordine de la 1 la Q, fiecare pe câte o linie. Descrierea unei runde i se face prin șirul de valori $R_1, R_2, \ldots, R_L$ găsit de Alex în runda respectivă, valorile fiind separate prin spații.

Date de ieșire

Se vor afișa Q linii. Dacă $C = 1$, linia i va conține numărul locațiilor care respectă șirul R de informații aflate de Alex în runda i. Dacă $C = 2$, linia i va conține doua numere naturale A și B, reprezentând coordonatele punctului P căutat pentru care d_P este minim în runda i, unde $1 \leq i \leq N$.

Restricții și precizări

- $1 \leq N, Q \leq 100\,000$.
- $1 \leq L \leq 30$.
- $1 \leq X_i, Y_i \leq 10^9$, oricare ar fi $1 \leq i \leq N$.
- $1 \leq W_i \leq 10\,000$, oricare ar fi $1 \leq i \leq N$.
- $Z_{i,j} \in \{0, 1\}$, oricare ar fi $1 \leq i \leq N$, $1 \leq j \leq L$.
- $R_j \in \{-1, 0, 1\}$, oricare ar fi $1 \leq j \leq L$.
- $1 \leq K \leq N$, pentru fiecare rundă din cele Q.
- Nu există două locații care desemnează același punct din plan.
- Se garantează că Alex respectă aceeași strategie, reprezentată de șirul $T_1, T_2, \ldots, T_N$, în fiecare din cele Q runde.

#	Punctaj	Restricții
1	9	$C = 1$ și $1 \leq N, Q, L, X_i, Y_i \leq 10$
2	11	$C = 1$ și $T_i = i$ pentru $1 \leq i \leq N$
3	17	$C = 1$
4	9	$C = 2$ și $1 \leq N, Q, L, X_i, Y_i, W_i \leq 10$
5	7	$C = 2$ și $1 \leq N, Q \leq 100$ și $1 \leq X_i, Y_i \leq 10\,000$
6	7	$C = 2$ și $1 \leq N, Q \leq 400$ și $1 \leq X_i, Y_i \leq 250\,000$
7	7	$C = 2$ și $1 \leq N, Q \leq 1\,000$
8	12	$C = 2$ și $W_i = 1$
9	21	$C = 2$

Exemple

geogra.in	geogra.out
1 7 4 1 4 1 1 0 1 3 1 1 1 1 0 7 2 0 1 0 0 9 7 1 0 1 0 3 9 0 0 0 0 5 6 0 0 1 0 4 4 1 1 0 0 5 1 0 1 0 -1 0 -1 0 -1 -1 -1 -1 -1 1 -1 -1 1 1 -1 0	1 3 7 4 2

geogra.in	geogra.out
2 7 4 1 4 1 1 1 0 1 3 1 1 1 1 1 0 7 2 5 0 1 0 0 9 7 1 1 0 1 0 3 9 2 0 0 0 0 5 6 1 0 0 1 0 4 4 1 1 1 0 0 5 1 0 1 0 -1 0 -1 0 -1 -1 -1 -1 -1 1 -1 -1 1 1 -1 0	9 7 3 7 5 2 7 2 3 1

Explicații

În cazul acestor exemple, strategia lui Alex este $T_1 = 2, T_2 = 4, T_3 = 1, T_4 = 3$.

Spre exemplu, în runda 2 Alex află mai întâi valoarea R_2, apoi valoarea R_4, dar, din cauza faptului că rămâne fără timp, nu mai reușește sa afle nici valoarea R_1 și nici valoarea R_3.

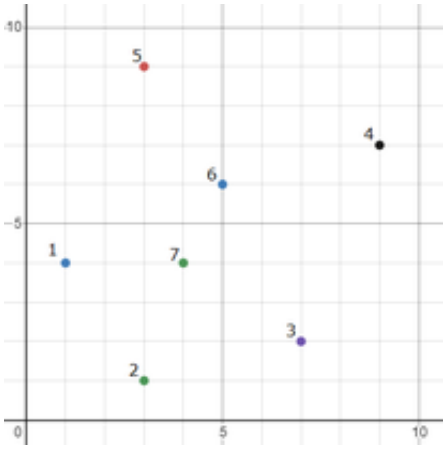
Pentru runda 1, locația care respectă șirul R găsit de Alex este 4. Pentru $C = 2$, punctul cerut este $P(9,7)$, iar $d_P = 0$.

Pentru runda 2, locațiile care respectă șirul R găsit de Alex sunt $4, 5, 6$. Pentru $C = 2$, punctul cerut este $P(3,7)$, iar $d_P = 13$.

Pentru runda 3, locațiile care respectă șirul R găsit de Alex sunt $1, 2, 3, 4, 5, 6, 7$. Pentru $C = 2$, punctul cerut este $P(5,2)$, iar $d_P = 53$.

Pentru runda 4, locațiile care respectă șirul R găsit de Alex sunt $1, 2, 3, 7$. Pentru $C = 2$, punctul cerut este $P(7,2)$, iar $d_P = 18$.

Pentru runda 5, locațiile care respectă șirul R găsit de Alex sunt $2, 7$. Pentru $C = 2$, punctul cerut este $P(3,1)$, iar $d_P = 4$.


10
5
0
5
10
1
2
3
4
5
6
7

2.5.C Problema Schi

Ultimul sezon rece a fost unul cu multa zăpadă căzută în zona montană, prin urmare magazinul de articole sportive din stațiunea de schi Semenic a avut vânzări mari, mai ales de clăpari (bocanci speciali pentru schiat). Acum fiind sfârșit de sezon, angajații magazinului constată că le-au rămas N cutii goale în care au fost ambalate perechile de clăpari vândute. Aceste cutii au forma unui paralelipiped dreptunghic la care fața superioară constituie capacul. Capacul și (evident) fața opusă capacului sunt de formă pătrată, iar înălțimea este una oarecare, ea depinzând de modelul de clăpari. La unele cutii capacul este prezent, la altele capacul a dispărut.

Aceste cutii trebuie predate unei firme de reciclare, dar până ajunge mașina pentru reciclat în Semenic, cutiile trebuie depozitate. Prin urmare unul dintre angajații magazinului primește sarcina să depoziteze cutiile punându-le una peste alta sub forma unui turn. Angajatul ia cutiile în ordinea în care le găsește în magazin și le pune una peste alta. Pentru a asigura o oarecare stabilitate a turnului, el le așează astfel încât axele care unesc centrul capacului cu centrul feței opuse capacului de la fiecare cutie să fie coliniare. Cutiile sunt astfel lăsate pe rând să cadă de la o înălțime suficient de mare, cu fețele laterale perpendiculare pe podea și paralele cu cele ale cutiilor deja plasate, până când întâlnesc una dintre aceste cutii sau podeaua. În ceea ce privește așezarea cutiilor care au capacul lipsă, angajatul le așează fie cu golul format prin lipsa capacului în sus, fie în jos.

Cerințe

Cunoscând N (numărul de cutii din magazin), ordinea în care cutiile sunt așezate în turn, iar pentru fiecare cutie latura capacului, înălțimea cutiei și dacă are capac sau, în cazul în care capacul lipsește, dacă cutia este așezată cu golul în sus sau cu golul în jos determinați:

1. înălțimea turnului astfel format;
2. numărul de cutii ale căror fețe laterale sunt vizibile dacă se privește turnul din lateral.

Date de intrare

Pe prima linie a fișierului de intrare `schi.in` se află numărul C, număr care poate fi 1 sau 2 și reprezintă cerința care trebuie rezolvată.

Pe ce de-a doua linie se află numărul natural N, reprezentând numărul de cutii care sunt așezate în turn.

Pe fiecare dintre următoarele N linii se află câte trei valori L, H și M, separate prin câte un spațiu. Valorile de pe linia $i+2$ ($1 \leq i \leq N$) reprezintă informații referitoare la cea de-a i-a cutie adăugată în turn, și anume:

- L — latura capacului cutiei;
- H — înălțimea cutiei;
- M — o valoare egală cu 0, 1 sau 2, având următoarea semnificație:
  - $M = 0$ — cutia are capacul lipsă și este așezată cu golul în jos;
  - $M = 1$ — cutia are capac;
  - $M = 2$ — cutia are capacul lipsă și este așezată cu golul în sus.

Date de ieșire

Dacă C este 1, fișierul de ieșire `schi.out` va conține pe prima linie răspunsul pentru cerința 1 (înălțimea turnului format).

Dacă C este 2, fișierul de ieșire `schi.out` va conține pe prima linie răspunsul pentru cerința 2 (numărul de cutii ale căror fețe laterale sunt vizibile dacă se privește turnul din lateral).

Restricții și precizări

- $1 \leq N \leq 200\,000$.
- Dimensiunile cutiilor sunt numere naturale din intervalul $[1, 10^9]$.
- Laturile capacelor cutiilor sunt distincte două câte două.
- Grosimea pereților care formează fețele cutiilor (inclusiv capacul) este neglijabilă.
- Construcția turnului se realizează pe o podea plană de dimensiuni infinite.

#	Punctaj	Restricții
1	12	$C = 1$ și $N \leq 2\,000$
2	12	$C = 2$ și $N \leq 2\,000$
3	8	$C = 1$ și nu avem nicio cutie așezată cu golul în sus
4	8	$C = 2$ și nu avem nicio cutie așezată cu golul în sus
5	8	$C = 1$ și nu avem nicio cutie așezată cu golul în jos
6	8	$C = 2$ și nu avem nicio cutie așezată cu golul în jos
7	22	$C = 1$
8	22	$C = 2$

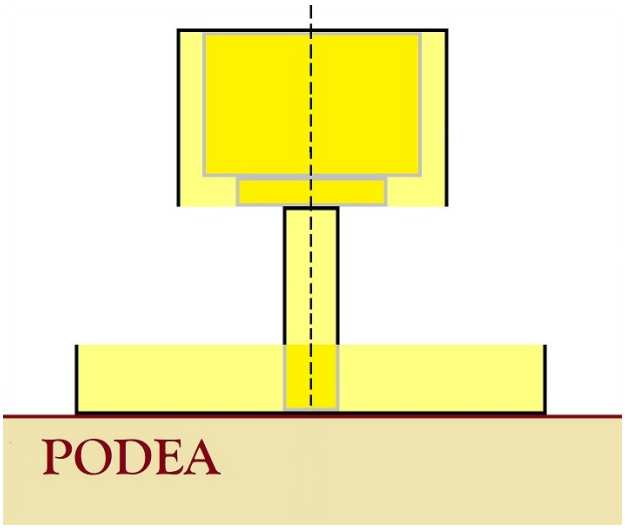
Exemple

schi.in	schi.out
1 5 20 3 2 3 7 1 9 1 1 11 5 1 12 6 0	13

schi.in	schi.out
2 5 20 3 2 3 7 1 9 1 1 11 5 1 12 6 0	3

Explicații

Înălțimea turnului de cutii este 13. Privind din lateral sunt vizibile 3 cutii, și anume cele care au capacele cu laturile 20, 3 și 12.



2.6 Clasa a X-a

2.6.A Problema ChangeMin

Dat fiind un șir de N numere naturale $A_1, A_2, \ldots, A_N$, considerăm următorul algoritm, prezentat în pseudocod:

```
1: cnt ← 0
2: score ← 0
3: for i de la 1 la N do
4:     min ← ∞
5:     for j de la i la N do
6:         if A[j] < min then
7:             min ← A[j]
8:             cnt ← cnt + 1
9:             score ← score + cnt · A[j]
10:        end if
11:    end for
12: end for
```

Cerințe

1. Care este valoarea lui *cnt* la sfârșitul algoritmului?
2. Care este valoarea lui *score* la sfârșitul algoritmului, modulo 666 013?

Date de intrare

Pe prima linie a fișierului de intrare `changemin.in` se află numerele naturale T și N, separate printr-un spațiu, reprezentând cerința care trebuie rezolvată, respectiv numărul de numere ce urmează a fi citite.

Pe următoarea linie se află N numere naturale separate printr-un spațiu, reprezentând numerele $A_1, A_2, \ldots, A_N$.

Date de ieșire

Fișierul de ieșire `changemin.out` va conține:

- Pentru $T = 1$: un singur număr natural, reprezentând valoarea lui *cnt* la finalul execuției algoritmului;
- Pentru $T = 2$: un singur număr natural, reprezentând valoarea lui *score* la finalul execuției algoritmului, modulo 666 013.

Restricții și precizări

- $T \in \{1, 2\}$.
- $1 \leq N \leq 1\,000\,000$.
- $1 \leq A_i \leq 1\,000\,000\,000$ pentru oricare $1 \leq i \leq N$.

#	Punctaj	Restricții
1	6	$T = 1$ și $N \leq 1\,000$
2	9	$T = 1$ și $A_i \leq 2$
3	21	$T = 1$, $N \leq 100\,000$ și $A_i \leq 200$
4	24	$T = 1$
5	4	$T = 2$ și $N \leq 1\,000$
6	6	$T = 2$ și $A_i \leq 2$
7	11	$T = 2$, $N \leq 100\,000$ și $A_i \leq 200$
8	19	$T = 2$

Exemple

changemin.in	changemin.out	Explicații
1 5 3 4 2 2 1	11	cnt este incrementată de 11 ori pe parcursul execuției algoritmului
2 5 3 4 2 2 1	103	score este obținută astfel: score = 3 * 1 + 2 * 2 + 1 * 3 + 4 * 4 + 2 * 5 + 1 * 6 + 2 * 7 + 1 * 8 + 2 * 9 + 1 * 10 + 1 * 11

2.6.B Problema Dragonfruit

„Anul 1905, toamna"

Cu ajutorul tău, Badinho a primit subvenția de la stat, iar construcția rutei a fost finalizată în timp record. Datorită succesului, acesta a decis sa își deschidă o nouă afacere în Ciudad de México. Pe plaiurile deținute de primăria orașului crește o specie rară de cactus, care la maturitate va da roade fructe *pitahaya*, cunoscute și sub denumirea de „dragon fruits".

Câmpul deținut de primărie are forma unui rând de N cactuși, dispuși de la stânga la dreapta, numerotați de la 1 la N. Notăm cu A_i pentru $1 \leq i \leq N$ numărul de fructe coapte din cactusul i. El Alcalde (primarul) dorește să recolteze exact K fructe de pe câmp, așa că a contractat firma lui Badinho.

Din motive logistice, recoltarea cactușilor se va realiza în cel mult S etape. Într-o etapă, drona care recoltează fructele va survola o singură parcelă de cactuși și va culege toate fructele coapte din cactușii survolați. O parcelă este definită ca o subsecvență de cactuși. De exemplu, parcela $[6, 9]$ este formată din cactușii $6, 7, 8$ și 9. Pentru a nu deranja populația de cactuși, fiecare cactus poate fi survolat cel mult o dată pe parcursul întregii operațiuni de recoltare. A survola o parcelă înseamnă a survola toți cactușii din parcela respectivă.

Pentru a planifica cât mai bine recoltarea fructelor, Badinho trebuie să-și facă un *plan de recoltare*. Mai exact, un astfel de plan constă dintr-un număr $S' \leq S$ de etape de recoltare care vor fi efectuate și din parcelele care vor fi survolate la fiecare din cele S' etape. Două planuri de recoltare se consideră diferite dacă există cel puțin o parcelă survolată într-o etapă a unuia dintre ele care să nu fie survolată și în vreo etapă a celuilalt. De exemplu, dacă într-un plan format din două etape se survolează parcela $[1, 2]$, urmată de parcela $[3, 4]$, iar în alt plan format din două etape se survolează parcela $[3, 4]$, urmată de parcela $[1, 2]$, atunci planurile se consideră identice. În schimb, dacă într-un plan cu o singură etapă se survolează parcela $[1, 2]$, iar în alt plan format din două etape se survolează parcelele $[1, 1]$ și $[2, 2]$, atunci planurile se consideră distincte. Un alt exeeplu este că dacă într-un plan format din două etape se survolează parcela $[1, 3]$, urmată de parcela $[4, 4]$, iar în alt plan format din două etape se survolează parcela $[1, 2]$, urmată de parcela $[3, 4]$, atunci planurile se considera a fi distincte.

Cerințe

Pentru a-și stabili planul de recoltare, Badinho este interesat de răspunsurile la următoarele întrebări:

1. Care este numărul minim total de cactuși care ar trebui survolați într-un plan de recoltare?
2. Câte planuri de recoltare în care se survolează un număr minim de cactuși există? Deoarece acest număr poate fi foarte mare, se cere doar valoarea sa modulo $1\,000\,000\,007$.

Badinho își dorește să afle raspunsurile la cele două întrebări într-un număr T de scenarii. Un scenariu este determinat de valorile N, K, S, $A_1, A_2, \ldots, A_N$.

Date de intrare

Pe prima linie a fișierului `dragonfruit.in` se află numărul T, urmat de cele T scenarii, fiecare în următorul format:

- pe prima linie numerele naturale N, K, S, separate prin spații;
- pe a doua linie numerele $A_1, A_2, \ldots, A_N$ separate prin spații.

Date de ieșire

Pentru fiecare scenariu dintre cele T se va afișa câte o linie în fișierul de ieșire `dragonfruit.out` care va conține două numere naturale separate printr-un spațiu reprezentând, în ordine, răspunsurile la cerințele 1 și 2 din enunț pentru scenariul curent. În cazul în care, pentru un anumit scenariu, nu există niciun plan de recoltare care să satisfacă cerințele date, se va afișa 0 0 pe linia respectivă.

Restricții și precizări

- $1 \leq T \leq 5$.
- $1 \leq K \leq 1\,000$.
- $1 \leq S \leq 10$.
- $0 \leq A_i \leq 1\,000$ pentru oricare $1 \leq i \leq N$.

#	Punctaj	Restricții
1	4	$S = 1$ și $1 \leq N \leq 2\,000$
2	7	$S = 2$ și $1 \leq N \leq 100$
3	9	$S = 3$ și $1 \leq N \leq 50$
4	10	$S = 1$ și $1 \leq N \leq 100\,000$
5	13	$S = 2$ și $1 \leq N \leq 2\,000$
6	23	$1 \leq N \leq 70$
7	34	$1 \leq N \leq 2\,000$

Exemple

dragonfruit.in	dragonfruit.out
4 8 7 4 2 1 0 1 0 2 1 99 2 99 10 50 50 3 8 1 1 8 1 5 3 2 2 1 2 1 1	5 3 0 0 1 1 2 9

Explicații

Scenariul 1 Următoarele planuri de recoltare recoltează 7 dragon fruits survolând un număr minim de cactuși:

- Planul 1: $[1, 2]$, $[4, 4]$, $[6, 7]$;

- Planul 2: $[1,1]$, $[2,2]$, $[4,4]$, $[6,7]$;
- Planul 3: $[1,2]$, $[4,4]$, $[6,6]$, $[7,7]$.

Observați că planul de recoltare $[1,3]$, $[4,4]$, $[6,7]$ nu se consideră valid, deoarece se survolează un număr de 6 cactuși, care nu este minim, chiar dacă s-au cules $K = 7$ dragon fruits.

Scenariul 2 Nu există niciun plan de recoltare în care să fie recoltate 99 dragon fruits.

Scenariul 3 Există un singur plan de recoltare în care să fie recoltate 8 dragon fruits: $[2,2]$.

Scenariul 4 Următoarele planuri de recoltare recoltează 3 dragon fruits survolând un număr minim de 2 cactuși:

- Planul 1: $[1,2]$;
- Planul 2: $[2,3]$;
- Planul 3: $[3,4]$;
- Planul 4: $[1,1]$, $[2,2]$;
- Planul 5: $[1,1]$, $[4,4]$;
- Planul 6: $[1,1]$, $[5,5]$;
- Planul 7: $[2,2]$, $[3,3]$;
- Planul 8: $[3,3]$, $[4,4]$;
- Planul 9: $[3,3]$, $[5,5]$.

Observați că planul de recoltare $[2,2]$, $[4,4]$, $[5,5]$ nu se consideră valid, deoarece se survolează un număr de 3 cactuși, care nu este minim, chiar dacă s-au cules $K = 3$ dragon fruits.

2.6.C Problema Munte

O *permutare* cu N elemente este un șir $a_1, a_2, \ldots, a_N$ care conține toate numerele naturale $1, 2, \ldots, N$, fiecare număr exact o dată.

Un exemplu de permutare cu 6 elemente poate fi următorul șir: $[3, 1, 6, 4, 5, 2]$.

Despre o permutare vom spune că este de tip *munte*, dacă printre cele N elemente ale sale există un element de indice M astfel încât $1 < M < N$, secvența formată din primele M elemente este strict crescătoare, iar secvența formată din ultimele $N - M + 1$ elemente este strict descrescătoare. Adică, folosind notația matematică, avem $a_1 < a_2 < \ldots < a_{M-1} < a_M > a_{M+1} > \ldots > a_N$.

Un exemplu de munte cu 6 elemente poate fi următorul șir: $[1, 2, 4, 5, 6, 3]$.

În problema noastră vom considera că avem o permutare cu N elemente indexate de la 1 la N. Asupra elementelor permutării putem aplica două tipuri de operații de interschimbare simetrică:

1. `swap`$(P, N - P + 1)$ — în permutarea a se interschimbă elementele de pe pozițiile P si $N - P + 1$, egal distanțate de cele două margini ale permutării. Operația necesită $1 \leq P \leq N$ și $P \neq N - P + 1$.
2. `dswap`$(P, Q, N - P + 1, N - Q + 1)$ — în permutarea a se interschimbă simultan două perechi de elemente. Se vor interschimba simultan elementele de pe pozițiile P și Q, respectiv elementele de pe pozițiile $N - P + 1$ și $N - Q + 1$. Operația se poate aplica dacă și numai dacă toate cele patru poziții sunt distincte două câte două. Cu alte cuvinte, operația necesită $1 \leq P, Q \leq N$ și mulțimea $\{P, Q, N - P + 1, N - Q + 1\}$ să aibă cardinal patru.

Plecând de la permutarea noastră și aplicând succesiv operații de ambele tipuri, unde fiecare tip poate fi folosit *de zero sau mai multe ori*, se poate obține o gamă variată de permutări.

Cerințe

1. Să se precizeze dacă pentru permutarea dată aplicând oricâte operații de `swap` sau `dswap` se poate obține o permutare de tip munte.
2. Plecând de la permutarea dată, câte permutări de tip munte distincte se pot obține aplicând oricâte operații de `swap` sau `dswap`?

Date de intrare

Fișierul `munte.in` conține pe prima linie două numere naturale $C \in \{1, 2\}$ și N, reprezentând cerința care trebuie rezolvată, respectiv numărul de elemente ale permutării a.

Pe a doua linie se găsesc N numere naturale separate prin câte un singur spațiu, reprezentând permutarea a.

Date de ieșire

Fișierul `munte.out` va conține o singură linie, în funcție de cerință:

- pentru $C = 1$, în cazul în care se poate obține cel puțin o permutare de tip munte plecând de la șirul a se va afișa cuvântul `DA`, altfel cuvântul `NU`;
- pentru $C = 2$, se va afișa un singur număr natural, reprezentând numărul permutărilor de tip munte distincte ce se pot obține aplicând doar operații de `swap` și `dswap` asupra permutării citite. Întrucât această valoare poate fi foarte mare, se va tipări rezultatul modulo 111 181 111.

Restricții și precizări

- $C \in \{1, 2\}$.
- $4 \leq N \leq 200\,000$.

#	Punctaj	Restricții
1	3	$C = 1$ și $N \leq 8$
2	3	$C = 2$ și $N \leq 8$
3	9	$C = 1$ și $N \leq 15$
4	9	$C = 2$ și $N \leq 15$
5	5	$C = 1$ și $N \leq 30$
6	6	$C = 2$ și $N \leq 30$
7	6	$C = 1$ și $N \leq 2\,000$, N impar și $a_{\frac{N+1}{2}} = N$
8	7	$C = 2$ și $N \leq 2\,000$, N impar și $a_{\frac{N+1}{2}} = N$
9	7	$C = 1$ și $N \leq 2\,000$, N par și $a_{\frac{N}{2}} = N$ și $a_{\frac{N}{2}+1} = N - 1$
10	8	$C = 2$ și $N \leq 2\,000$, N par și $a_{\frac{N}{2}} = N$ și $a_{\frac{N}{2}+1} = N - 1$
11	8	$C = 1$ și $N \leq 2\,000$
12	9	$C = 2$ și $N \leq 2\,000$
13	10	$C = 1$
14	10	$C = 2$

Exemple

munte.in	munte.out	Explicații
2 7 3 4 2 7 1 6 5	4	Se pot obține următoarele patru permutări de tip munte: 1 3 4 7 6 5 2 2 3 4 7 6 5 1 2 5 6 7 4 3 1 1 5 6 7 4 3 2
1 6 6 3 4 5 1 2	DA	Se pot obține următoarele permutări de tip munte: 1 2 4 5 6 3 3 6 5 4 2 1

munte.in	munte.out	Explicații
1 6 1 2 3 5 4 6	NU	Nu există niciun mod de a transforma permutarea în munte.
1 7 2 3 4 1 7 6 5	NU	Nu există niciun mod de a transforma permutarea în munte.

Explicație detaliată pentru exemplul 1

În primul exemplu avem $N = 7$ și $a = [3,4,2,7,1,6,5]$. Putem obține 4 permutări de tip munte distincte:

1. dswap$(1,5,7,3)$ — $[3,4,2,7,1,6,5]$, se obține $[1,4,5,7,3,6,2]$.
   dswap$(2,5,6,3)$ — $[1,4,5,7,3,6,2]$, se obține $[1,3,6,7,4,5,2]$.
   swap$(3,5)$ — $[1,3,6,7,4,5,2]$, se obține permutarea de tip munte $[1,3,4,7,6,5,2]$.
2. dswap$(1,3,7,5)$ — $[3,4,2,7,1,6,5]$, se obține $[2,4,3,7,5,6,1]$.
   dswap$(2,3,6,5)$ — $[2,4,3,7,5,6,1]$, se obține permutarea de tip munte $[2,3,4,7,6,5,1]$.
3. dswap$(7,2,1,6)$ — $[3,4,2,7,1,6,5]$, se obține $[6,5,2,7,1,3,4]$.
   dswap$(5,7,3,1)$ — $[6,5,2,7,1,3,4]$, se obține permutarea de tip munte $[2,5,6,7,4,3,1]$.
4. dswap$(2,5,6,3)$ — $[3,4,2,7,1,6,5]$, se obține $[3,1,6,7,4,2,5]$.
   dswap$(2,1,6,7)$ — $[3,1,6,7,4,2,5]$, se obține $[1,3,6,7,4,5,2]$.
   swap$(2,6)$ — $[1,3,6,7,4,5,2]$, se obține permutarea de tip munte $[1,5,6,7,4,3,2]$.

2.7 Clasele XI–XII

2.7.A Problema Lupușor și Mielu

Nicu Constantin și Stefan Mihăilescu-Brăila în „Lupușor și Mielu” (1975)

Mielu de la bucătărie urmează să plece definitiv din întreprinderea unde lucrează. Tovarășul Lupușor de la Personal îi oferă o ultimă șansă: să îl învingă la un joc de cărți. Mielu acceptă, iar Lupușor îi prezintă regulile jocului, după cum urmează. Pe masă sunt așezate N cărți de joc numerotate de la 1 la N. Fiecare carte i, unde $1 \leq i \leq N$, are înscrisă pe ea două numere întregi pozitive a_i şi b_i. Toate numerele înscrise pe cărți sunt distincte două câte două. La prima mutare a jocului Mielu alege o parte dintre cărți și le elimină din joc. El poate alege să nu elimine nici o carte, însă nu este permisă eliminarea tuturor cărților. La a doua mutare, Lupușor alege două cărţi dintre cele rămase în joc, cu indicii inițiali i și j. Lupușor are voie inclusiv să aleagă aceeași carte de două ori (adică $i = j$). Dacă $a_i > b_j$ atunci Lupușor câștigă jocul, altfel câştigă Mielu. Lupușor este deosebit de viclean, așa că el întotdeauna va alege la a doua mutare valorile i și j astfel încât să câştige, dacă acest lucru este posibil.

Cerință

Mielu este acum pus în impas: pentru a-şi păstra poziția, el va trebui să răspundă la două întrebări grele, date fiind N și valorile $a_1, \ldots a_N$ și $b_1, \ldots, b_N$:

1. Care este numărul minim de cărţi pe care le poate Mielu elimina la prima mutare astfel încât Lupușor să piardă jocul. Dacă acest lucru este imposibil, atunci răspunsul se consideră -1.
2. În câte feluri poate Mielu elimina cărţi la prima mutare astfel încât Lupușor să piardă jocul. Răspunsul se cere modulo $10^9 + 7$. Atenție!, în acest caz nu este necesară eliminarea unui număr minim de cărți!

Se dă un număr $C \in \{1, 2\}$. Dacă $C = 1$ atunci Mielu va trebui să raspundă numai la întrebarea 1, altfel numai la întrebarea 2.

Mai grav! De M ori tovarășul director vine și schimbă valorile înscrise pe câte o carte din cele N. Mai exact, la a i-a modificare, pentru $1 \leq i \leq M$, directorul schimbă valorile a_{id_i} și b_{id_i} înscrise pe cartea id_i în c_i și, respectiv, d_i. După fiecare astfel de modificare Mielu va trebui să raspundă din nou la întrebarea indicată de numărul C. Se garantează că după fiecare modificare valorile înscrise pe cărţi rămân distincte două câte două.

Date de intrare

Pe prima linie a fișierului de intrare `lupusor.in` se află C. Pe a doua linie se află N. Pe a treia linie se găsesc numerele $a_1, \ldots, a_N$, separate prin spații. Pe a patra linie se găsesc numerele $b_1, \ldots, b_N$, separate prin spații. Pe a cincea linie se află M. Următoarele M linii descriu modificările dispuse de director: linia i conține numerele intregi pozitive id_i, c_i, d_i, separate prin spatii, unde $1 \leq i \leq M$.

Date de ieșire

În fișerul de ieșire `lupusor.out` se vor tipări $M + 1$ linii. Fiecare linie va conține un singur număr întreg, reprezentând răspunsul la prima, sau, respectiv, a doua cerinţă, în funcție de valoarea lui C. Prima linie va reprezenta răspunsul înainte de modificări, a doua linie răspunsul după prima modificare efectuată, și asa mai departe, linia i va reprezenta răspunsul după primele $i - 1$ modificări.

Restricții și precizări

- $1 \leq N \leq 100\,000$.
- $0 \leq M \leq 100\,000$.
- $1 \leq a_i, b_i \leq 2N + 2M$, oricare ar fi $1 \leq i \leq N$.
- $1 \leq c_i, d_i \leq 2N + 2M$, oricare ar fi $1 \leq i \leq M$.
- $1 \leq id_i \leq N$, oricare ar fi $1 \leq i \leq M$.
- Fiecare număr $1 \leq k \leq 2N + 2M$ apare în exact unul dintre șirurile a, b, c sau d.

#	Punctaj	Restricții
1	5	$C = 1$ și $N, M \leq 10$
2	6	$C = 1$ și $N, M \leq 100$
3	6	$C = 1$ și $N, M \leq 400$
4	10	$C = 1$ și $N, M \leq 2\,000$
5	15	$C = 1$ și $N, M \leq 50\,000$
6	7	$C = 1$
7	5	$C = 2$ și $N, M \leq 10$
8	7	$C = 2$ și $N, M \leq 100$
9	7	$C = 2$ și $N, M \leq 400$
10	13	$C = 2$ și $N, M \leq 2\,000$
11	13	$C = 2$ și $N, M \leq 50\,000$
12	6	$C = 2$

Exemple

lupusor.in	lupusor.out
1 3 1 7 6 5 10 8 2 2 9 3 3 2 4	1 2 1
2 3 1 7 6 5 10 8 2 2 9 3 3 2 4	4 2 3
1 1 1 2 1 1 4 3	0 -1
2 1 1 2 1 1 4 3	1 0

Explicații

În primul exemplu, cartea 1 are inițial înscrise pe ea numerele $a_1 = 1$ și $b_1 = 5$, pe care le vom scrie simplificat $(1,5)$, cartea 2 numerele $(7,10)$, iar cartea 3 numerele $(6,8)$. Avem $C = 1$, deci se răspunde numai la întrebarea 1. Observăm că dacă Mielu nu ar elimina nicio carte, atunci Lupușor ar câștiga alegând $i = 2$ și $j = 1$ deoarece $a_2 = 7 > 5 = b_1$. În schimb, dacă Mielu alege să elimine cartea 1, atunci în joc rămân doar cărțile 2 si 3, cu valori $(7,10)$ și $(6,8)$. În acest caz Lupușor nu are cum să mai aleaga i și j dintre cărțile rămase astfel incat $a_i > b_j$, deci câștigă Mielu.

După prima modificare, cartea 1 are înscrise pe ea numerele $(1,5)$, cartea 2 numerele $(9,3)$, iar cartea 3 numerele $(6,8)$. De acestă data, este necesară eliminarea a două cărți: fie cărțile 1 și 2, fie cărțile 2 și 3.

După a doua modificare, cartea 1 are înscrise pe ea numerele $(1,5)$, cartea 2 numerele $(9,3)$, iar cartea 3 numerele $(2,4)$. De acestă data, este suficientă doar eliminarea cărții 2.

Al doilea exemplu este identic cu primul, doar că se răspunde la întrebarea 2 (deoarece $C = 2$). Înainte de modificări există 4 moduri de a elimina o parte din cărți astfel încât Mielu să câștige:

1. Se elimină doar cartea 1;
2. Se elimină cărțile 1 și 2;
3. Se elimină cărțile 2 și 3;
4. Se elimină cărțile 1 și 3.

Observăm că eliminarea tuturor cărților se interzice prin regulile jocului.

După prima modificare există 2 moduri de a elimina o parte din cărți astfel încât Mielu să câștige:

1. Se elimină cărțile 1 și 2;
2. Se elimină cărțile 2 și 3.

După a doua modificare există 3 moduri de a elimina o parte din cărți astfel încât Mielu să câștige:

1. Se elimină doar cartea 2;
2. Se elimină cărțile 1 și 2;
3. Se elimină cărțile 2 și 3.

Observăm că numai primul dintre aceste 3 moduri elimină un număr minim de cărți, însă pe noi ne interesează numărul total de moduri, indiferent dacă acestea elimină un număr minim de cărți sau nu.

2.7.B Problema Regate și Alianțe

În tărâmul ONI se află N regate legate între ele prin M muchii bidirecționale. Se garantează că se poate ajunge de la orice regat la oricare alt regat folosind aceste muchii. Aceste regate vor să facă alianțe între ele și se vor folosi de puncte de frontieră pentru a realiza acest lucru.

Fiecare muchie i, unde $1 \leq i \leq M$, are asociat un număr natural c_i reprezentând costul construcției unui punct de frontieră pe aceasta. Mai mult decât atât, fiecare regat i, unde $1 \leq i \leq N$, are asociat un număr natural r_i reprezentând costul construcției unui punct de frontieră la intrarea acestuia.

Pentru ca regatul X să intre într-o alianță cu regatul Y, unde $1 \leq X, Y \leq N, X \neq Y$, acesta are două opțiuni:

- construiește un punct de frontieră pe *o singură* muchie i cu cost c_i, astfel încât *orice drum* de la X la Y trece prin acest punct de frontieră;
- construiește un punct de frontieră la intrarea regatului său (regatului X) cu cost r_X.

Evident, regatul X va alege costul *minim* pentru a intra într-o alianță cu regatul Y. Vom nota acest cost minim cu $Cost(X, Y)$. Atenție, costul ca regatul X să intre într-o alianță cu regatul Y poate fi diferit de costul ca regatul Y să intre într-o alianță cu regatul X!

Pentru ca regatul X să fie într-o *alianță perfectă* acesta trebuie să facă alianță cu toate celalate regate.

Atenție, în formarea unei alianțe nu se iau în considerare punctele de frontieră construite în formarea altor alianțe. Cu alte cuvinte, $Cost(X, Y)$ se calculează independent pentru fiecare pereche (X, Y)!

Cerință

Pentru fiecare regat trebuie să calculați costul pe care acesta trebuie să-l plătească pentru a fi într-o alianță perfectă. Cu alte cuvinte, pentru fiecare regat i, unde $1 \leq i \leq N$, trebuie să calculați $\sum_{j=1, j \neq i}^{N} Cost(i, j)$.

Date de intrare

Pe prima linie a fișierului de intrare `regate.in` se află N și M, numărul de regate, respectiv, numărul de muchii bidirecționale. Pe a doua linie se găsesc numerele $r_1, \ldots, r_N$, separate prin spații, unde r_i reprezintă costul construcției unui punct de frontieră la intrarea regatului i. Pe următoarele M linii se află câte trei numere naturale a_i, b_i, c_i, separate prin spații, $1 \leq i \leq M$, semnificând că există o muchie bidirecțională între regatul a_i și regatul b_i, iar c_i reprezintă costul construcției unui punct de frontieră pe muchia i.

Date de ieșire

În fișierul de ieșire `regate.out` se vor afișa N linii. Fiecare linie i va conține un singur număr întreg, reprezentând costul ce trebuie plătit pentru ca regatul i să fie într-o alianță perfectă, cu alte cuvinte, $\sum_{j=1, j \neq i}^{N} Cost(i, j)$.

Restricții și precizări

- $1 \leq N \leq 200\,000$.
- $1 \leq M \leq 200\,000$.
- $1 \leq c_i \leq 10^9$.
- $1 \leq r_i \leq 10^9$
- $1 \leq a_i, b_i \leq N$, $a_i \neq b_i$ (adică nu există muchie de la un regat la el insuși).
- Între două regate poate exista cel mult o muchie.

#	Punctaj	Restricții
1	5	$N \leq 2000$ si $M = N - 1$, iar regatele și muchiile vor forma un lanț în ordinea $1, 2, \ldots, N$
2	10	$N \leq 200$ si $M \leq 400$
3	20	$N \leq 2000$ si $M \leq 2000$
4	10	Toate numerele din șirul c sunt egale
5	35	$M = N - 1$
6	20	Fără restricții suplimentare

Exemple

regate.in	regate.out
5 5 8 13 9 7 12 1 2 10 2 3 10 3 4 10 4 5 10 1 3 10	32 46 36 28 40

Explicație

De exemplu, pentru regatul 2 avem:

$$Cost(2,1) = 13$$
$$Cost(2,3) = 13$$
$$Cost(2,4) = 10$$
$$Cost(2,5) = 10$$

Suma acestora este 46.

Atenție că, de exemplu, în calculul lui $Cost(2,1)$, nu putem pune un punct de frontieră pe muchia $(1,2)$ de cost 10, deoarece există drumul $2 \to 3 \to 1$ care nu trece prin muchia $(1,2)$.

2.7.C Problema Schema și Investițiile

După o lungă activitate în domeniul instalaţiilor sanitare, Dorel s-a hotărât să-și investească averea acumulată în acţiuni ale mai multor companii. După prima sa încercare care a avut succes, el a reușit să strângă și mai mulți bani decât avea înainte. Astfel, acesta s-a hotărât să continue această activitate și să investească în mai multe proiecte ale companiilor în care a investit anterior.

Dorel are o sumă de bani G și are la dispoziție N proiecte numerotate de la 1 la N în care poate să investească bani. Pentru fiecare proiect se cunoaște valoarea a_i reprezentând câți bani dorește el să investească în proiectul i.

Schema de investiție a lui Dorel funcționează în felul următor: el analizează proiectele pe rând, iar când se află la un proiect i, dacă are suficienți bani (adică dacă suma de bani de care dispune este mai mare sau egală decât a_i), atunci el va investi bani în acel proiect (iar din suma de bani de care dispune se scade a_i). Altfel, el nu va investi niciun ban și va analiza următorul proiect.

Întrucât Dorel dorește să nu își consume toți banii și să învețe să fie econom, el dorește să reordoneze proiectele astfel încât după ce analizează fiecare proiect și își aplică algoritmul descris să rămână cu cât mai mulți bani.

Cerință

Să se afle care este suma maximă de bani cu care poate să rămână Dorel după ce își reordonează strategic proiectele și aplică algoritmul său.

Date de intrare

Pe prima linie a fișierului de intrare `schema.in` se află două numere N și G, separate printr-un spațiu. Pe a doua linie se află N numere separate prin spații, al i-ulea număr fiind a_i.

Date de ieșire

Pe singura linie a fișierului de ieșire `schema.out` se va tipări suma maximă de bani cu care poate să rămână Dorel.

Restricții și precizări

- $1 \leq N \leq 2\,000$.
- $0 \leq G \leq 5\,000$.
- $0 \leq a_i \leq 5\,000$ pentru i de la 1 la N.

#	Punctaj	Restricții
1	11	Ordinea proiectelor dată în fișierul de intrare este cea optimă.
2	19	$N \leq 7$
3	14	Șirul a este format din două valori distincte care se repetă, în orice ordine.
4	12	$N \leq 80$
5	25	$N \leq 2\,000$ și $0 \leq a_1 + a_2 + \cdots + a_N \leq 150$
6	19	Fără restricții suplimentare

Exemple

schema.in	schema.out
3 10 7 4 5	3

Explicație

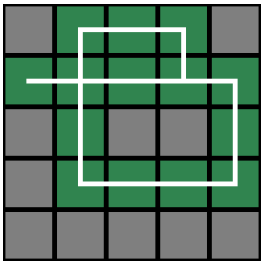
Dorel păstrează ordinea inițială a proiectelor. Pentru primul proiect, acesta are suficienți bani pentru a investi în el, așa că o să facă asta. Va rămâne cu 3 unități monetare. Pentru celălalte doua proiecte, acesta nu va avea suficient cât să plătească pentru ele, așa că va rămâne în final cu suma de 3. Acesta este răspunsul optim.

Dacă acesta și-ar fi reordonat obiectele în $[4, 7, 5]$, la primul proiect va avea suficienți bani, deci o să investească. La al doilea proiect are 6 unități monetare și are nevoie de 7 ca să investească, așa că o să ignore proiectul. Pentru ultimul proiect, acesta are suficienți bani pentru a investi. În final rămâne cu o unitate monetară. Astfel, această reordonare nu este optimă.

Capitolul 3 Proba de Baraj

3.1 Juniori

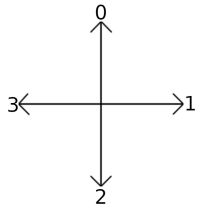
3.1.A Problema Autostradă



Gimi tocmai a câștigat o licitație pentru asfaltarea unei noi autostrăzi. Firma lui este responsabilă de prelucrarea zonelor dintr-o suprafață bidimensională de dimensiuni $N \times N$. Știm că dacă un drum va trece prin a i-a linie, respectiv a j-a coloană, atunci acea zonă de la poziția (i, j) va trebui asfaltată. Liniile și coloanele suprafeței bidimensionale sunt numerotate de la 1 la N.

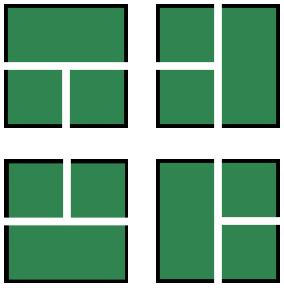
Din nefericire pentru el, a aflat abia după semnarea contractului că proiectul nu este așa de simplu precum a crezut. Drumul va fi determinat de traseul urmat de o dronă. Mai mult, Gimi nici nu știe acest traseu, dar a primit de la proiectantul autostrăzii poziția inițială a dronei (l_s, c_s), reprezentând linia și coloana zonei în care se află inițial drona, și o listă de K instrucțiuni pe care acesta le-a aplicat dronei.

Fiecare instrucțiune este o pereche de tipul (dir, p), cu semnificația că drona s-a deplasat pe direcția dir cu p unități, unde dir poate avea următoarele valori:

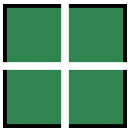


- 0 — direcția Nord,
- 1 — direcția Est,
- 2 — direcția Sud,
- 3 — direcția Vest.

Spre exemplu, dacă drona se află în poziția $(1, 3)$ și a primit instrucțiunea $(2, 3)$, atunci drona se va deplasa spre sud (direcția 2) cu 3 unități și va parcurge zonele $(2, 3), (3, 3)$ și $(4, 3)$ unde se va opri. Toate aceste zone vor trebui asfaltate de firma lui Gimi.



Cele patru tipuri de intersecții de tip $\top$.



Intersecție de tip $+$.

Costul de asfaltare al unei zone prin care trece un drum simplu este C_z. Din fericire pentru Gimi, o zonă parcursă de mai multe ori de dronă trebuie asfaltată o singură dată. Însă, el a observat că pot apărea niște cazuri particulare:

- Dacă într-o zonă se produce o intersecție de tip $\top$, înseamnă că este nevoie de benzi de accelerare/decelerare în intersecție. Atunci costul de asfaltare al zonei devine C_t.
- Dacă intr-o zonă se produce o intersecție de tip $+$, înseamnă că este nevoie de construirea unui pod, caz în care costul de asfaltare al zonei devine C_p.

Cerință

Având aceste informații, Gimi vrea să verifice dacă traseul dronei este valid, adică drona nu va părăsi niciodată suprafața de care este responsabilă firma lui Gimi. În cazul în care traseul este invalid Gimi vrea să știe a câta instrucțiune dintre cele K a determinat mutarea dronei în afara suprafeței. Dacă traseul este valid, el vrea să determine costul total de asfaltare al autostrăzii.

Date de intrare

Prima linie din fișierul de intrare `autostrada.in` va conține patru numere naturale N, K, l_s și c_s, unde N este dimensiunea suprafeței, K este numarul de instrucțiuni aplicate dronei, iar (l_s, c_s) reprezintă linia și coloana zonei în care se află inițial drona.

A doua linie va conține trei numere naturale C_z C_t și C_p, reprezentând costurile de asfaltare ale unei zone simple, unei intersecții de tip $\top$, respectiv unei intersecții de tip $+$.

Următoarele K linii vor conține câte două numere naturale dir_i și p_i, reprezentând valorile specifice celei de-a i-a instrucțiuni primite de dronă.

Numerele scrise pe aceeași linie sunt separate printr-un singur spațiu.

Date de ieșire

Dacă traseul dronei este invalid, pe prima linie din fișierul de ieșire `autostrada.out` se va afișa textul `TRASEU INVALID`, iar apoi, pe a doua linie, un singur număr natural, reprezentând a câta instrucțiune dintre cele K a determinat mutarea dronei în afara suprafeței.

Altfel, pe prima linie din fișierul de ieșire se va afișa textul `TRASEU VALID`, iar apoi, pe a doua linie, un singur număr natural reprezentând costul total necesar pentru asfaltarea autostrăzii.

Restricții și precizări

- $2 \leq N \leq 2\,000$.
- $1 \leq K \leq 1\,000\,000$.
- $1 \leq l_s, c_s \leq N$.
- $1 \leq C_z, C_t, C_p \leq 100$.
- $dir_i \in \{0, 1, 2, 3\}$ pentru orice $1 \leq i \leq K$.
- $1 \leq p_i \leq N$ pentru orice $1 \leq i \leq K$.
- Poziția inițială și poziția finală a dronei trebuie asfaltate.

#	Punctaj	Restricții
1	14	Traseul este invalid
2	6	$1 \leq K \leq 2\,000$ traseul este valid și nu conține niciun tip de intersecție
3	8	$1 \leq K \leq 2\,000$ și traseul este valid și nu conține intersecții de tip $+$
4	9	$1 \leq K \leq 2\,000$ și traseul este valid și nu conține intersecții de tip $\top$
5	24	$1 \leq K \leq 2\,000$ și traseul este valid
6	39	Traseul este valid

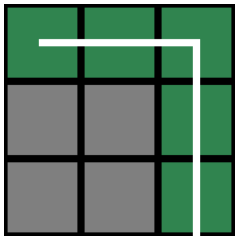
Exemple

autostrada.in	autostrada.out
3 3 1 1 1 2 3 1 2 2 3 3 1	TRASEU INVALID 2
5 7 2 1 1 2 3 1 4 2 2 3 3 0 3 1 2 2 1 3 3	TRASEU VALID 17

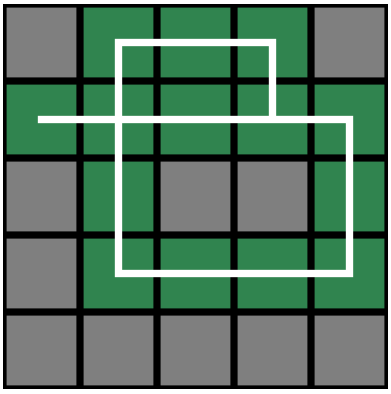
Explicații

În exemplele de mai jos, cu verde s-au marcat zonele ce trebuiesc asfaltate, iar cu linii albe s-a indicat traseul dronei. Zonele gri sunt zone nevizitate de traseul dronei.

Primul exemplu A doua instrucțiune este invalidă, drona ieșind din suprafață întrucât ar ajunge in coordonatele $(4, 3)$.



Al doilea exemplu Traseul determinat de dronă este:



Se observă că toate zonele, mai puțin $(2, 2)$ și $(2, 4)$ sunt simple, adică nu sunt intersecții, și fiecare au costul de asfaltare $C_z = 1$.

În $(2, 2)$ avem o intersecție de tip $+$, deci costul de asfaltare va fi $C_p = 3$.

În $(2, 4)$ avem o intersecție de tip $\top$, deci costul de asfaltare va fi $C_t = 2$.

În total, costul este $12 \times 1 + 1 \times 2 + 1 \times 3 = 17$.

3.1.B Problema Bug

Dacă vrei să-ţi schimbi buletinul trebuie să mergi la Serviciul de Evidenţă a Populaţiei. Acolo trebuie să iei un număr de ordine şi să aştepţi să-ţi vină rândul. Numerele de ordine sunt emise de un robot, în ordinea 1, 2, 3, ... Programatorul Vasile, care a elaborat soft-ul pentru robot şi care asigură (contra cost) întreţinerea sistemului, a creat intenţionat un bug în sistem. Vasile are un număr natural preferat N. Un număr de ordine x va fi emis dacă şi numai dacă x este un subşir al lui N (adică toate cifrele lui x apar în N în ordinea din x, nu neapărat pe poziţii consecutive). Dacă numărul de ordine curent x nu îndeplineşte această condiţie, robotul se blochează şi nu mai emite numere de ordine.

Cerinţă

Scrieţi un program care, cunoscând valoarea lui N, numărul natural preferat de Vasile, rezolvă următoarele două cerințe:

1. determină câte cifre are numărul de ordine x care conduce la blocarea robotului;
2. determină numărul de ordine x care conduce la blocarea robotului.

Date de intrare

Fișierul de intrare `bug.in` conţine pe prima linie cerința C care trebuie să fie rezolvată (1 sau 2). Pe cea de a doua linie se află numărul natural N.

Date de ieșire

Fişierul de ieşire `bug.out` va conţine o singură linie pe care va fi scris răspunsul la cerința C.

Restricții și precizări

- N este un număr natural nenul având cel mult 100 000 de cifre.

#	Punctaj	Restricții
1	23	$C = 1$
2	10	$C = 2$ și N are cel mult 18 cifre.
3	10	$C = 2$, N are cel puţin 20 de cifre și rezultatul are cel mult 7 cifre.
4	30	$C = 2$, N are cel puţin 101 şi cel mult 10 000 de cifre.
5	27	$C = 2$ și nu există alte restricții.

Exemple

bug.in	bug.out	Explicații
1 1032	1	Cel mai mic număr natural nenul care nu este subşir al lui 1032 are o singură cifră.
2 1032	4	Cel mai mic număr natural nenul care nu este subşir al lui 1032 este 4.
1 2563201231245876156078 9	2	Cel mai mic număr natural nenul care nu este subşir al lui 2563201231245876156078 9 are două cifre.
2 2563201231245876156078 9	42	Cel mai mic număr natural nenul care nu este subşir al lui 2563201231245876156078 9 este 42.

3.1.C Problema Triprime

Un număr se numește *triprim* dacă este produsul a trei numere prime distincte. Exemple de numere triprime: $30 = 2 \times 3 \times 5$, $42 = 2 \times 3 \times 7$, $231 = 3 \times 7 \times 11$. Exemple de numere care nu sunt triprime: $77 = 7 \times 11$ (prea puține numere prime în produs), $3003 = 3 \times 7 \times 11 \times 13$ (prea multe numere prime în produs), $18 = 2 \times 3 \times 3$ (numerele prime nu sunt distincte), $10241 = 7 \times 7 \times 11 \times 19$ (prea multe numere prime în produs).

Cerinţă

Date fiind numerele A și B să se afișeze numărul de numere triprime din intervalul $[A, B]$ (inclusiv A și B).

Date de intrare

Fișierul de intrare `triprime.in` conține pe prima linie două numere naturale A și B, despărțite printr-un singur spațiu.

Date de ieșire

Fișierul de ieșire `triprime.out` va conține numărul de numere triprime din intervalul $[A, B]$.

Restricții și precizări

- $1 \leq A \leq B \leq 390\,000\,000$.

#	Punctaj	Restricții
1	18	$1 \leq B \leq 1\,500\,000$
2	6	$1\,500\,000 < B \leq 2\,500\,000$
3	20	$2\,500\,000 < B \leq 4\,500\,000$
4	31	$4\,500\,000 < B \leq 35\,000\,000$
5	25	Nu există alte restricții.

Exemple

triprime.in	triprime.out	Explicații
`1 50`	`2`	`Sunt două numere triprime de la 1 la 50: 30 = 2 * 3 * 5 și 42 = 2 * 3 * 7.`

triprime.in	**triprime.out**	**Explicații**
50 105	5	Sunt cinci numere triprime de la 50 la 105: 66 = 2 * 3 * 11, 70 = 2 * 5 * 7, 78 = 2 * 3 * 13, 102 = 2 * 3 * 17 și 105 = 3 * 5 * 7.
1000 3000	348	Sunt 348 de numere triprime în intervalul [1000, 3000].

3.2 Seniori

3.2.A Problema 3dist

Noul cartier din Buguré face mari furori în toată țara. Acesta este alcătuit din N locuințe care pot fi cumpărate, a i-a locuință aflându-se la coordonatele (X_i, Y_i). Notăm cu $dist(i, j) = |X_i - X_j| + |Y_i - Y_j|$ și $d(i) = \min_{j \neq i}(dist(i, j))$.

Fiind prieteni din copilărie și nedespărțiți, Àles, RANDy și Țeba doresc să fie și ei în rând cu lumea bună și decid să achiziționeze și ei locuințe, câte una pentru fiecare. Cei trei țin foarte mult la relația lor și fiecare dorește să aibă ceva de spus în alegerea locuințelor, așa că ei convin următoarele:

- $\grave{A} < R < \c{T}$,
- $dist(\grave{A}, R) = dist(R, \c{T}) = dist(\grave{A}, \c{T})$,
- $d(\grave{A}) = d(R) = d(\c{T}) = dist(\grave{A}, R)$,

unde notăm cu inițiala numelui fiecăruia numărul de ordine al casei pe care o cumpără fiecare.

La o analiză mai detaliată, Țeba, creierul echipei, observă că au foarte multe alegeri posibile și își pune întrebarea: *„În câte moduri ne putem alege locuințele astfel încât cerințele de mai sus să fie respectate?”*.

Date de intrare

De pe prima linie se va citi numărul natural N. Pe următoarele N linii se află câte două numere X_i, Y_i, reprezentând coordonatele locuințelor.

Date de ieșire

Pe prima linie se va afișa un singur număr S reprezentând răspunsul întrebării lui Țeba.

Restricții și precizări

- $1 \leq N \leq 250\,000$.
- $0 \leq X_i, Y_i \leq 1\,000\,000\,000$ pentru oricare $1 \leq i \leq N$.
- Nu vor exista două locuințe aflate la aceleași coordonate.

#	Punctaj	Restricții
1	7	$N \leq 100$
2	14	$N \leq 2\,000$
3	28	$X_i, Y_i \leq 2\,000$ pentru oricare $1 \leq i \leq N$
4	51	Fără restricții suplimentare

Exemple

stdin	stdout	Explicații
5 1 1 3 1 2 2 2 6 4 4	1	Singurul triplet care respectă cerințele este: (1, 2, 3). Tripletul (3, 4, 5) respectă dist(3, 4) = dist(4, 5) = dist(3, 5) însă d(3) = 2, d(5) = 4 și d(4) = 4.

3.2.B Problema Piezișă

Cunoscut pentru multe lucruri importante, cum ar fi metroul orașului și festivalul Nespus, orașul Jluc găzduiește încă un obiectiv turistic, totuși mai puțin cunoscut decât cele menționate anterior — *Piezișă*.

La prima vedere, Piezișă este doar o stradă, cu mai multe magazine de-a lungul ei. Mai exact, aceasta are n magazine poziționate de-a lungul ei, numerotate de la 0 la $n-1$. Totuși, Piezișă este mai mult decât ceea ce pare: este un loc unde se creează amintiri. Magazinul i are un număr asociat v_i, care reprezintă calitatea amintirilor create în acel magazin.

Auzind de această stradă, Alex își dorește să viziteze un interval continuu de magazine în seara aceasta. El are q planuri de asemenea intervale, al i-lea fiind de forma $[l_i, r_i]$. Pentru a nu pierde timp, el dorește să meargă pe Piezișă cu noua sa *trotinetă electrică*. Totuși, Alex este superstițios, și este convins că dacă suma `xor` a valorilor v_i dintr-un interval vizitat nu ar fi 0, atunci asta i-ar aduce ghinion. Așadar, pentru fiecare plan, el dorește să afle lungimea intervalului de lungime minima care conține magazinele din plan, și care are suma `xor` 0.

Formal, se dă un șir de n valori intregi, și q intervale de forma $[l_i, r_i]$. Trebuie să calculați, pentru fiecare astfel de interval, lungimea intervalului de lungime minimă $[x, y]$, cu proprietatea că $x \leq l_i \leq r_i \leq y$ și pentru care v_x `xor` v_{x+1} `xor` ... `xor` $v_y = 0$.

Date de intrare

Prima linie conține un singur număr natural, reprezentând valoarea lui n. A doua linie conține n numere întregi, reprezentând valorile $v_0, v_1, \ldots, v_{n-1}$. A treia linie conține un singur număr natural, reprezentând valoarea lui q. Pe următoarele q linii se află valorile corespunzătoare planurilor. Linia $i+3$ conține 2 valori întregi, reprezentând l_i și r_i.

Date de ieșire

Se vor afișa q linii. Pe linia i, se va afla răspunsul la al i-lea plan.

Restricții și precizări

- Operația `xor` reprezintă operația de sau exclusiv pe biți. În C/C++, acest operator este `^`.
- Dacă pentru un plan, nu există niciun segment cu suma `xor` egala cu 0, atunci răspunsul este -1.
- $1 \leq n \leq 500\,000$.
- $1 \leq q \leq 500\,000$.
- $0 \leq v_i < 2^{30}$, pentru oricare $0 \leq i < n$.
- $0 \leq l_i \leq r_i < n$, pentru oricare $0 \leq i < n$.

#	Punctaj	Restricții
1	10	$1 \leq n, q \leq 500$
2	15	$1 \leq n, q \leq 3\,000$
3	5	$v_i < 4$, pentru oricare $0 \leq i < n$
4	40	$1 \leq n \leq 100\,000$
5	30	Nu există alte restricții

Exemple

stdin	stdout	Explicații
6 2 0 3 3 2 2 4 5 5 1 1 0 1 2 2	2 1 5 2	Primul plan dorește să viziteze magazinul de pe poziția 5, care are valoarea 2. Observăm că subsegmentul [4, 5] (care conține valorile 2, 2) are suma xor 0, așadar el este cel mai mic subsegment cu xor 0 care conține [5, 5]. Răspunsul va fi, așadar, 2. Al doilea plan conține doar magazinul de pe poziția 1, care are valoarea 0. Segmentul optim pentru acest plan este [1, 1]. Al treilea plan conține magazinele 0 si 1, care au valorile 2, 0. Segmentul optim pentru acest plan este [0, 4]. Al doilea plan conține doar magazinul de pe poziția 2, care are valoarea 3. Segmentul optim pentru acest plan este [2, 3].
10 5 7 3 6 1 2 5 2 5 4 3 7 9 5 8 1 5	8 4 -1	Pentru primul plan, unul dintre segmentele optime este [2, 9], de lungime 8. Pentru al doilea plan, unul dintre segmentele optime este [5, 8], de lungime 4. Pentru al treilea plan, nu există niciun subsegment valid, așadar este afișat -1.

3.2.C Problema Portocal

Ana a primit un arbore cu N noduri cu rădăcina în nodul 1, unde fiecare nod are asociată o valoare naturală de la 1 la M. Pentru fiecare nod, va scrie câte un șir format din valorile nodurilor de pe lanțul de la rădăcină la acel nod, în ordine. Apoi, va sorta cele N șiruri lexicografic.

După ce a obținut șirurile sortate, Ana se gândește la un șir S format din K numere naturale cuprinse, de asemenea, între 1 și M, și îl întreabă pe Portocal dacă există cel puțin un șir egal cu S. Cum Portocal este un motan leneș, el va verifica în fiecare zi un singur șir, începând cu primul în ordinea sortării, până când va găsi unul egal cu S.

Portocal a observat că unele noduri din arbore nu au încă asociată o valoare, așa că și-a propus să dea el valori acestor noduri (tot între 1 și M) înainte ca Ana să se apuce de scris șirurile. Scopul lui este să găsească șirul S cât mai repede. Pentru a decide cum să completeze valorile din arbore, are nevoie de răspunsul la două întrebări:

1. Numărul minim de zile în care Portocal poate găsi șirul S.
2. Numărul de moduri în care poate completa valorile lipsă din arbore astfel încât să obțină acel număr minim de zile. Cum acest număr poate fi foarte mare, se mulțumește cu restul împărțirii la 1 000 000 009.

Evident, Portocal își dorește să găsească cel puțin un șir egal cu S. Se garantează că există cel puțin o modalitate de a completa valorile arborelui pentru a realiza acest lucru.

Date de intrare

Prima linie va conține un număr C. Dacă $C = 1$, trebuie răspuns la întrebarea 1, iar dacă $C = 2$, trebuie răspuns la întrebarea 2. Pe a doua linie se vor afla numerele N, M și K. Pe a treia linie se găsesc N numere naturale $val_1, \ldots, val_N$, reprezentând valorile asociate nodurilor din arbore. Dacă val_i este -1, înseamnă că nodul i nu are încă asociată o valoare. Pe a patra linie se vor afla K numere naturale reprezentând șirul S. Următoarele $N - 1$ linii vor conține câte două numere u și v, reprezentând muchiile arborelui.

Date de ieșire

Se va afișa un singur număr reprezentând răspunsul la prima, respectiv a doua întrebare, în funcție de valoarea lui C.

Restricții și precizări

- $C \in \{1, 2\}$.
- $1 \leq K \leq N \leq 500\,000$.
- $1 \leq M \leq 500\,000$.
- $1 \leq val_i \leq M$ sau $val_i = -1$, oricare $1 \leq i \leq N$.
- $1 \leq S_i \leq M$, oricare $1 \leq i \leq K$.

#	Punctaj	Restricții
1	8	$C = 1$, $N \leq 13$, $M \leq 3$
2	19	$C = 1$, $N \leq 5\,000$
3	22	$C = 1$
4	11	$C = 2$, $N \leq 13$, $M \leq 3$
5	40	$C = 2$

Exemple

stdin	stdout	Explicații
1 8 3 3 -1 -1 2 -1 -1 -1 1 2 1 2 2 1 2 2 3 2 4 4 5 1 6 6 7 1 8	4	Portocal poate completa valorile nodurilor astfel: 1 2 2 2 1 3 1 2. Primele șiruri în ordine lexicografică vor fi: 1 - pentru nodul 1 1 2 - pentru nodul 2 1 2 - pentru nodul 8 1 2 2 - pentru nodul 3, care este egal cu S. Astfel, răspunsul este 4. Observați că, deși mai există un șir egal cu S, pentru nodul 4, Portocal se oprește când îl va găsi pe primul.
2 8 3 3 -1 -1 2 -1 -1 -1 1 2 1 2 2 1 2 2 3 2 4 4 5 1 6 6 7 1 8	6	Există 6 moduri de a completa valorile din arbore astfel încât Portocal să găsească șirul S în ziua 4. Acestea sunt: 1 2 2 2 1 3 1 2 1 2 2 3 1 3 1 2 1 2 2 2 2 3 1 2 1 2 2 3 2 3 1 2 1 2 2 2 3 3 1 2 1 2 2 3 3 3 1 2

3.2.D Problema „Miyuki vrea să împăturească arborigami”

Kaguya, vicepreședintele consiliului de studenți, este răcită. Miyuki vrea, cum este tradițional, să îi ofere cadou o mie de cocori de hârtie. Poate nu o mie... (*prea exagerat pentru o simplă răceală*), și poate nu cocor... (*prea tradițional și oricum nu am hârtie de origami...*). Un arbore stea ar trebui să fie ideal!

Miyuki are un arbore (graf conex aciclic) format din N noduri numerotate de la 1 la N. El dorește să îl transforme într-un arbore stea de dimensiune $N - K$, adică un graf conex aciclic care are cel puțin $N - K - 1$ frunze (noduri cu exact 1 vecin).

Pentru a transforma arborele său într-un arbore stea, Miyuki va efectua K operații de împăturire a câte două noduri. Pentru a i-a operație de împăturire, Miyuki:

- Alege două noduri distincte a_i și b_i existente în acel moment în arbore.
- Notează cu V mulțimea vecinilor nodurilor a_i și b_i (nodurile care au o muchie directă către cel puțin unul dintre a_i sau b_i).
- Șterge din V nodurile a_i și b_i, dacă acestea erau prezente.
- Șterge din arbore nodurile a_i și b_i, cât și muchiile care aveau cel puțin un capăt într-unul din nodurile a_i și b_i.
- Adaugă în arbore un nod cu numărul $N + i$.
- Adaugă muchii între nodul $N + i$ și fiecare din nodurile din mulțimea V.

După o astfel de operație, graful rezultat trebuie să fie în continuare arbore; mai precis, operația efectuată nu trebuie să introducă vreun ciclu. Altfel, operația este invalidă și nu poate fi efectuată.

Deoarece nu vrea să pară că s-a străduit prea mult, Miyuki vrea să facă un număr K minim de operații. Pentru că secretara Chika a promis că nu îl mai învață nimic, trebuie să-l ajutați pe Miyuki să determine:

1. Care este numărul minim K de operații pentru a transforma arborele inițial în arbore stea.
2. Care sunt cele K operații prin care arborele inițial este transformat într-un arbore stea.

Date de intrare

De pe prima linie se va citi un singur număr natural N, reprezentând dimensiunea arborelui inițial. Pe următoarele $N - 1$ linii vor fi descrise muchiile arborelui inițial, pe linia $i + 1$ aflându-se două numere naturale u_i și v_i, reprezentând nodurile unite de a i-a muchie din arbore.

Date de ieșire

Pe prima linie se va afișa un singur număr natural K, reprezentând numărul minim de operații ce trebuie efectuate. Pe următoarele K linii se vor afișa K perechi de numere naturale a_i și b_i ($1 \leq i \leq K$), separate prin câte un spațiu, reprezentând, în ordine, operațiile de împăturire ce se efectuează asupra arborelui.

Restricții și precizări

- $1 \leq N \leq 500\,000$.
- Dacă există mai multe soluții posibile, se poate afișa oricare.

- Dacă se determină corect numărul minim de operații K, dar operațiile afișate nu transformă corect arborele într-unul stea, sau una dintre operații este invalidă conform cu definiția din enunț, se va acorda 30% din punctaj.

#	Punctaj	Restricții
1	10	$1 \leq N \leq 15$
2	20	$1 \leq N \leq 200$
3	10	$u_i = i, v_i = i + 1$, pentru $1 \leq i < N$
4	60	Nu există alte restricții

Exemple

stdin	stdout	Explicații
5 1 2 2 3 3 4 4 5	1 2 4	Se efectuează o singură operație de împăturire, între nodurile 2 și 4. Operația decurge în felul următor - vecinii nodurilor 2 și 4 sunt V = 1, 3, 5 - ștergem nodurile 2 și 4 din arbore - adăugăm nodul N+1 = 6 și muchiile (1, 6), (3, 6), (5, 6) Arborele final este cel compus din nodurile 1, 3, 5 și 6 și muchiile (1, 6), (3, 6), (5, 6). Observăm că nodurile 1, 3, și 5 au un singur vecin și doar 6 are 3 vecini, deci arborele rezultat este stea.

stdin	stdout	Explicații
6 1 2 2 3 3 4 4 5 5 6	2 2 4 5 7	Se efectuează două operații de împăturire: - (2, 4), adăugând nodul N+1 = 7; - (5, 7), adăugând nodul N+2 = 8. Arborele final este cel compus din nodurile 1, 3, 6 și 8 și muchiile (1, 8), (3, 8), (6, 8).

3.2.E Problema Guguștiuc

Gimi Guguștiucul tocmai a ajuns intr-o situație destul de complicată. El are de participat la N ședințe, a i-a ședință desfașurându-se în intervalul de timp deschis la capete (x_i, y_i). El poate participa la mai multe ședințe simultan, fiind online.

Pentru a-și simplica programul, Gimi a decis să ia niște pauze și să elimine cateva ședințe (să nu mai participe deloc la ele). El a aplicat o listă de Q operații, nu neapărat foarte inspirate:

- `split t`: Gimi va lua o pauză la momentul de timp t. Deci, pentru fiecare ședința din intervalul de timp (x_i, y_i), dacă se respectă condiția $x_i < t < y_i$, atunci ședința respectivă este eliminată și înlocuită cu două ședințe noi în intervalele de timp deschise la capete (x_i, t) și (t, y_i),
- `skip t`: Gimi nu va mai participa deloc la toate ședințele care sunt în plină desfașurare la momentul de timp t. Cu alte cuvinte, pentru fiecare fiecare ședință din intervalul de timp (x_i, y_i), dacă se respectă condiția $x_i < t < y_i$, atunci Gimi va elimina ședința.

Gimi vrea să știe dupa cele Q operații care este suma duratelor tuturor ședințelor ramase. Durata unei ședințe din intervalul de timp (x, y) se definește ca fiind $y - x$. Duratele ședințelor se adună în întregime, chiar dacă există intervale de timp pe care acestea se suprapun.

Date de intrare

Pe prima linie se găsesc două numere N și Q. Pe următoarele N linii se găsesc câte două numere, x_i, y_i pe fiecare linie, acestea reprezentând câte un interval în care se desfășoară o ședință. Pe următoarele Q linii se găsesc câte două numere a_i și t_i. Dacă a_i este 1, atunci este descrisă o operație de tip `split` folosind valoarea t_i. Dacă a_i este 2, atunci este descrisă o operație de tip `skip` unde este folosită valoarea t_i.

Date de ieșire

Se va afișa un singur număr reprezentând suma duratelor tuturor ședințelor ramase, după aplicarea celor Q operații.

Restricții și precizări

- $1 \leq N, Q \leq 500\,000$.
- $1 \leq x_i, y_i, t_i \leq 1\,000\,000$, oricare ar fi $1 \leq i \leq Q$.
- $1 \leq a_i \leq 2$, oricare ar fi $1 \leq i \leq Q$.

#	Punctaj	Restricții
1	9	$1 \leq N, Q \leq 200$
2	12	$N = 1$
3	13	$N, Q \leq 1\,000$
4	12	$x_i \leq x_{i+1}, y_i \leq y_{i+1}$ pentru $1 \leq i < N$
5	11	$1 \leq N \leq 50\,000$ și $x_i, y_i, t_i \leq 50\,000$
6	19	$1 \leq N \leq 100\,000$
7	24	Nu există alte restricții

Exemple

stdin	stdout
2 3 1 10 4 10 1 3 1 6 2 5	10

Explicații

Gimi are două ședințe în intervalele de timp $(1, 10)$ și $(4, 10)$.

După prima operație, el elimină ședința din intervalul $(1, 10)$ și adăugă două ședințe noi în intervalele de timp $(1, 3)$ și $(3, 10)$. Acum are trei ședințe la intervalele de timp $(1, 3)$, $(3, 10)$ și $(4, 10)$.

După a doua operație, Gimi elimină ședința din intervalul $(3, 10)$ și adăugă două ședințe noi în intervalele de timp $(3, 6)$ și $(6, 10)$. De asemenea, Gimi elimină ședința din intervalul $(4, 10)$ și se adaugă ședințele $(4, 6)$ și $(6, 10)$. Acum are cinci ședințe la intervalele de timp $(1, 3)$, $(3, 6)$ și $(6, 10)$, $(4, 6)$ și $(6, 10)$.

După ultima operație, Gimi elimină ședința de la intervalul de timp $(3, 6)$ și ședința de la intervalul de timp (4.6). Rămân ședințele de la intervalele de timp $(1, 3)$, $(6, 10)$ și încă una tot la intervalul de timp $(6, 10)$.

Suma duratelor tuturor ședințelor ramase va fi $(3 - 1) + (10 - 6) + (10 - 6) = 10$.

3.2.F Problema Hoață

Într-un muzeu se află un coridor liniar format din N camere, numerotate de la 1 la N. În camera $1 \leq i \leq N$ se găsește o rezerva infinită de lingouri de aur de același tip de valoare v_i și greutate g_i. În prima cameră intră K hoți, fiecare având în spinare câte un rucsac de capacitate G, inițial gol. Când un hoț se află în camera i, acesta poate sustrage oricâte lingouri din camera curentă și să le adauge în rucsacul său, cu condiția ca suma greutăților lingourilor din rucsac să nu depășească G. Un lingou o dată furat, acesta va rămâne în rucsacul hoțului până la ieșirea din muzeu.

Hoții acționează în grup, așa că ei vor face turul muzeului în N pași, după cum urmează: la pasul $1 \leq i \leq N$ toți hoții avansează din camera i în camera $i+1$, unde camera $N+1$ se consideră exteriorul muzeului. Observăm că după primii i pași toți hoții se vor afla în camera $i+1$. Conducerea muzeului a instalat alarme în dreptul ușilor dintre oricare două camere consecutive. Mai exact, alarma $1 \leq i \leq N$ este instalată între camerele i și $i+1$ și este caracterizată de o valoare x_i. Aceasta se declanșează dacă și numai dacă în momentul când hoții trec pe ușa dintre camerele i și $i+1$ există cel puțin $x_i + 1$ hoți ale căror rucsacuri au aceeași greutate totală la acel moment, deoarece în acest caz s-ar efectua un control de rutină și hoții ar fi prinși (acest lucru se întâmplă chiar și dacă hoții nu au furat nimic până la acel moment). Bineînțeles, alarma N este instalată între camera N și exteriorul muzeului.

Odată ieșiți din muzeu, hoții calculează captura totală ca fiind suma valorilor v corespunzătoare lingourilor din cele K rucsacuri. Se dau T scenarii, și pentru fiecare se cere captura maximă posibilă în condițiile date, sau -1 dacă orice s-ar întâmpla hoții ar fi prinși.

Date de intrare

Prima linie conține un singur număr natural T, reprezentând numărul de scenarii. Urmează descrierile celor T scenarii. Descrierea unui scenariu se face după cum urmează: pe prima linie trei numere naturale N, K, G, separate prin spații; pe următoarele N linii câte trei numere naturale, unde pe linia $1 \leq i \leq N$ se află numerele v_i, g_i, x_i, separate prin spații.

Date de ieșire

Se vor afișa T linii, reprezentând, în ordinea dată, răspunsurile pentru cele T scenarii date la intrare.

Restricții și precizări

- $1 \leq T \leq 900$.
- $1 \leq N \leq 300$.
- $1 \leq K \leq 50$.
- $1 \leq G \leq 300$.
- $1 \leq v_i \leq 300$, oricare ar fi $1 \leq i \leq N$.
- $1 \leq g_i \leq 300$, oricare ar fi $1 \leq i \leq N$.
- $1 \leq x_i \leq 50$, oricare ar fi $1 \leq i \leq N$.
- $1 \leq S_N \leq 900$, unde cu S_N am notat suma valorilor N corespunzătoare celor T scenarii.

#	Punctaj	Restricții
1	11	$N \leq 4$, $K \leq 3$, $G \leq 7$, $S_N \leq 12$, $v_i \leq 20$, $2 \leq g_i \leq 7$, $x_i \leq 3$, oricare ar fi $1 \leq i \leq N$.
2	18	Există $1 \leq j \leq N$ astfel încât $x_i = K$ oricare ar fi $1 \leq i \leq N$, $i \neq j$.
3	40	$N \leq 40$, $G \leq 40$, $S_N \leq 120$, $v_i \leq 40$, $g_i \leq 40$, oricare ar fi $1 \leq i \leq N$.
4	31	Fără restricții suplimentare.

Exemple

stdin	stdout
3 2 1 3 10 2 1 9 1 2 2 2 3 10 2 1 9 1 2 2 3 3 10 2 1 9 1 2	27 46 -1

Explicații

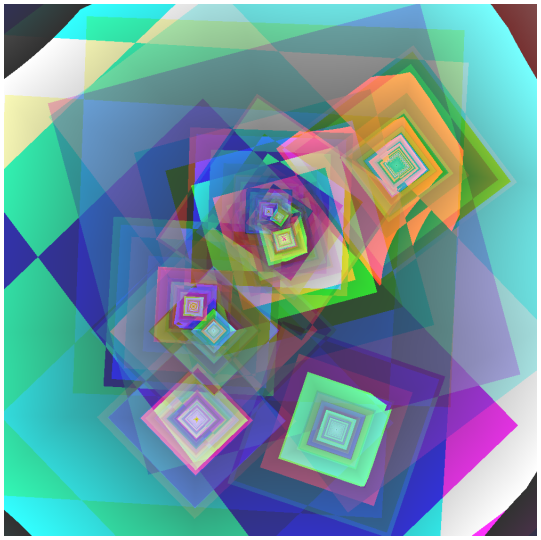
Sunt $T = 3$ scenarii.

Primul scenariu Avem $N = 2$ camere și $K = 1$ hoț înzestrat cu un rucsac de capacitate $G = 3$. În camera 1 se afla o rezervă infinită de lingouri de aur de valoare 10 și greutate 2, iar în camera 2 se află o rezervă infinită de lingouri de aur de valoare 9 și greutate 1. Alarma dintre camera 1 și camera 2 are $x_1 = 1$, iar alarma dintre camera 2 și ieșire are $x_2 = 2$. În condițiile date alarmele nu vor suna indiferent ce alege să facă hoțul, așa că acesta poate obține o captura maximă de $27 = 9 + 9 + 9$ furând trei lingouri din camera 2.

Al doilea scenariu Acest scenariu este identic cu primul, doar că avem $K = 2$ hoți, fiecare având cate un rucsac de capacitate 3. Dacă ambii hoți iau câte 3 lingouri din camera 2, atunci aceștia ar avea o captură totală de $54 = 6 \times 9$. Din păcate, dacă ar face acest lucru, ei ar fi prinși de alarma dintre camerele 1 și 2. Observăm că ei ar fi prinși de aceasta alarmă chiar și dacă aleg sa nu fure nimic din nicio cameră! Captura maximă, de fapt, se obține, de exemplu, dacă primul hoț alege să fure câte un lingou din fiecare cameră (total $19 = 10 + 9$), iar al doilea hoț alege să fure trei lingouri din camera 2 (total $27 = 9 + 9 + 9$). În total $46 = 19 + 27$.

Al treilea scenariu Acest scenariu este identic cu primele două, doar că avem $K = 3$ hoți. În acest caz cei trei hoți nu vor putea trece de camera 1 fără sa declanșeze alarma, deci răspunsul este -1.

Partea II
Descriere Soluțiilor



Capitolul 4 Olimpiada Județeană de Informatică

4.1 Clasa a V-a

4.1.A Problema Ceas

Propusă de: prof. Pintea Adrian Doru — Colegiul Național „Andrei Mureșanu" Dej

Pentru cerința 1, se citesc pe rând cele N numere și pentru fiecare număr se calculează de câte ori conține cifra X.

Pentru cerința 2, se citesc pe rând cele N numere și pentru fiecare număr se fac tăieturile a uneia(cifra unităților) sau a două cifre(cifra zecilor și cifra unităților) conform cerinței, numărând fiecare taietură realizată.

4.1.B Problema Sss

Propusă de: prof. Marius Nicoli — Colegiul Național „Frații Buzești" Craiova; Programator la Syncro Soft, Craiova

Pentru cerința 1, aflăm mai întâi valoarea lui K imediat după citirea primului termen al șirului. Pentru acest lucru folosim algoritmul clasic de parcurgere a cifrelor unui număr (împărțiri repetate la 10). Ulterior citim și celelalte numere și acumulăm la suma cerută valorile celor aflate pe poziții mai mari sau egale cu $N - K + 1$ (considerăm numerotarea de la poziția 1).

Pentru cerința 2, imediat după citirea lui N și înainte să citim elementele șirului, determinăm valoarea L. Vom folosi o repetiție în care construim suma $1 + 2 + 3 + \ldots$ până când aceasta devine egală cu N (lucru garantat de restricțiile din enunț). Numărul de termeni ai acestei sume este chiar L. Acum putem începe citirea valorilor șirului, de exemplu putem folosi o variabilă sumaCurentă în care acumulăm inițial valorile primelor L elemente, apoi resetăm sumaCurentă și acumulăm la ea valorile următoarelor $L - 1$ și așa mai departe până terminăm de citit valorile din șir.

4.2 Clasa a VI-a

Mulțumim studentului Gabriel Tulbă-Lecu pentu munca depusă în aceste editoriale.

4.2.A Problema Cmmdc

Propusă de: prof. Dan Pracsiu — Liceul Teoretic „Emil Racoviță"

Deoarece sunt folositori în rezolvarea celor trei cerințe, construim de la început doi vectori de lungime n, st și dr cu semnificația următoare:

- st_i va reține cel mai mare divizor comun al numerelor $a_1, a_2, \ldots, a_i$,
- dr_i va reține cel mai mare divizor comun al numerelor $a_i, a_{i+1}, \ldots a_n$.

Vectorul st se va construi de la stânga la dreapta, iar vectorul dr de la dreapta la stânga, după relațiile:

$$\begin{aligned} st_1 &= a_1 \\ st_i &= \text{cmmdc}(st_{i-1}, a_i), && 2 \leq i \leq n \\ dr_n &= a_n \\ dr_i &= \text{cmmdc}(dr_{i+1}, a_i), && 1 \leq i \leq n-1 \end{aligned}$$

Cerința 1

Soluție brută Se calculează cmmdc-ul prin factorizare, sau se caută manual cmmdc-ul și se verifică dacă acesta este divizor pentru fiecare element din șir și se păstrează maximul găsit dintre toți candidații.

Pentru această rezolvare brută a primei cerințe, se pot obține 16 puncte.

Soluție optimă Dacă se folosește algoritmul lui Euclid pentru calcularea soluției, răspunsul este st_n (sau dr_1).

Complexitatea temporală este $O(n \log_2(max))$. Pentru rezolvarea corectă a primei cerințe, se pot obține 36 de puncte.

Cerința 2

Soluție brută Se calculează pentru fiecare $1 \leq i \leq n$, care este cmmdc-ul dacă eliminăm elementul a_i și se păstrează rezultatul maxim.

Complexitatea temporală este $O(n^2 \log_2(max))$. Pentru rezolvarea brută a celei de-a doua cerințe, se pot obține 20 de puncte.

Soluție optimă Trebuie să aflăm cmmdc-ul maxim dacă se elimină exact un număr din șir.

Pentru a realiza acest lucru, parcurgem șirul și pentru fiecare poziție i:

- dacă se elimină a_1, atunci cel mai mare divizor comun este dr_2,
- dacă se elimină a_n, atunci cel mai mare divizor comun este st_{n-1},
- dacă se elimină a_i, pentru $2 \leq i \leq n-1$, atunci cel mai mare divizor comun al numerelor rămase este $\text{cmmdc}(st_{i-1}, dr_{i+1})$.

Dintre toate aceste valori determinate aflăm maximul.

Complexitatea temporală este $O(n \log_2(max))$. Pentru rezolvarea corectă a celei de-a doua cerințe, se pot obține 42 de puncte.

Cerința 3

Soluție brută Se calculează pentru fiecare pereche ordonată $1 \leq i < j \leq n$, care este cmmdc-ul dacă eliminăm elementul a_i și a_j și se păstrează rezultatul maxim.

Complexitatea temporală este $O(n^3 \log_2(max))$. Pentru rezolvarea brută a celei de-a doua cerințe, se pot obține 10 puncte.

Soluție optimă Pentru a putea face calculele mai ușor, e bine să știm faptul că $\text{cmmdc}(x, 0) = x$, pentru orice număr natural x.

Trebuie să aflăm cel mai mic divizor comun maxim dacă se elimină exact două elemente ale șirului. Fie a_i și a_j elementele pe care le eliminăm, unde $i < j$. Ne vor trebui atunci valorile lui $\text{cmmdc}(a_1, a_2, \ldots, a_{i-1})$, lui $\text{cmmdc}(a_{i+1}, \ldots, a_{j-1})$ și lui $\text{cmmdc}(a_{j+1}, \ldots, a_n)$. Știm însă că $\text{cmmdc}(a_1, a_2, \ldots, a_{i-1}) = st_{i-1}$, iar $\text{cmmdc}(a_{j+1}, \ldots, a_n) = dr_{j+1}$. Mai trebuie să aflăm $c = \text{cmmdc}(a_{i+1}, \ldots, a_{j-1})$ în mod eficient.

Parcurgem șirul de la primul până la penultimul element și pentru fiecare a_i, $1 \leq i \leq n-1$:

- Plecăm cu $c = 0$.
- Parcurgem cu j secvența $a_{i+1}, a_{i+2}, \ldots, a_n$.
- Când $j = i + 1$, atunci $c = 0$.
- Când $j = i + 2$, atunci $c = \text{cmmdc}(c, a_{i+1})$.
- Când $j = i + 2$, atunci $c = \text{cmmdc}(c, a_{i+1})$.
- ...
- La un pas oarecare j, $c = \text{cmmdc}(c, a_{j-1})$.

La fiecare pas j se actualizează valoarea maximă obținută din eliminarea lui a_i și a_j:

$$M = \max(\text{cmmdc}(a_1, a_2, \ldots, a_{i-1}), \text{cmmdc}(a_{i+1}, \ldots, a_{j-1}), \text{cmmdc}(a_{j+1}, \ldots, a_n))$$

Complexitatea temporală este $O(n^2 \log_2(max))$. Pentru rezolvarea corectă a primei cerințe, se pot obține 22 de puncte.

4.2.B Problema Vecine

Propusă de: prof. Rodica Pintea — Liceul Teoretic „Radu Vlădescu"

Cerința 1

Două cifre alăturate din șir, c_i și c_{i+1}, sunt consecutive dacă $c_i + 1 = c_{i+1}$, pentru orice $1 \leq i \leq n-1$. Se parcurge șirul de cifre și se contorizează aceste perechi.

Complexitatea temporală este $O(n)$. Pentru rezolvarea corectă a primei cerințe, se pot obține 20 de puncte.

Cerința 2

Două numere sunt vecine dacă sunt consecutive și dacă primul număr este cu exact 1 mai mare decât cel de-al doilea.

O observație ce trebuie făcută este că două numere consecutive au fie același număr de cifre, fie diferă prin 1. Cazurile când numărul cifrelor diferă sunt când primul număr este de forma

$$\underbrace{99\ldots9}_{\text{de } k \text{ ori}},$$

iar cel de-al doilea este de forma

$$1\underbrace{00\ldots0}_{\text{de } k \text{ ori}}.$$

Astfel, problema se poate rezolva cu următorul algoritm:

1. Pentru fiecare $1 \leq k \leq 10$ fixăm lungimea primului număr la k.
2. Pentru fiecare poziție posibilă $1 \leq i \leq n - 2k + 1$, se verifică dacă $X + 1 = Y$, unde $X = \overline{c_i c_{i+1} c_{i+2} \ldots c_{i+k-1}}$ și $Y = \overline{c_{i+k} c_{i+k+1} c_{i+k+2} \ldots c_{i+2k-1}}$, dacă X și Y au aceeași lungime
3. Pentru fiecare poziție posibilă $1 \leq i \leq n - 2k$, se verifică dacă $X + 1 = Y$, unde $X = \overline{c_i c_{i+1} c_{i+2} \ldots c_{i+k-1}}$ și $Y = \overline{c_{i+k} c_{i+k+1} c_{i+k+2} \ldots c_{i+2k}}$, dacă Y are cu o cifră mai mult decât X.
4. dacă unul dintre cazurile de mai sus sunt adevărate se actualizează maximul găsit cu X, dacă este cazul.

Complexitatea temporală a acestui algoritm este $O(nk_{max})$, unde $k_{max} = 10$, reprezintă lungimea maximă posibilă a unui număr.

4.3 Clasa a VII-a

Mulțumim studentului Gabriel Tulbă-Lecu pentu munca depusă în aceste editoriale.

4.3.A Problema Pătrățele

Propusă de: prof. Marinel Șerban — Colegiul Național „Emil Racoviță", Iași

Observații

Problema testează cunoștințe despre:
- lucrul cu tablouri unidimensionale și bidimensionale;
- reprezentarea numerelor în baza 2;
- operații pe biți sau operații echivalente.

Soluția 1 — prof. Marinel Șerban

Această soluție obține între 70 și 84 de puncte în funcție de implementare.

Verificarea faptului că un pătrățel are una dintre laturi desenată se face simplu utilizând reprezentarea în baza 2 a codului pătrățelului astfel: fie x valoarea codului pătrățelului:
- pentru latura de sus, dacă această latură există, atunci $x\&1 = 1$ sau $x\%2 = 1$;
- pentru latura din dreapta, dacă această latură există, atunci $x\&2 = 1$ sau $x/2\%2 = 1$;
- pentru latura de jos, dacă această latură există, atunci $x\&2 = 4$ sau $x/4\%2 = 1$;
- pentru latura din stânga, dacă această latură există, atunci $x\&2 = 8$ sau $x/8\%2 = 1$.

Cerințele 1 și 2 La aceste cerințe:
1. determinăm latura maximă $l_{max} = \min(n, m)$ pe care o poate avea un pătrat;
2. pentru toate laturile posibile $1 \leq l \leq l_{max}$ verificăm toate posibilitățile de a exista un pătrat de latura l, cu colțul stânga-sus în poziția (x_s, y_s) (evident x_s parcurge valorile de la 1 la $n - l + 1$ iar y_s toate valorile de la 1 la $m - l + 1$);
3. pentru fiecare pătrat de latură l și colțul stânga-sus în (x_s, y_s) verificăm dacă cele 4 laturi sunt complet desenate, iar dacă da incrementam un contor.

Pentru cerința 1 afișăm numărul total de pătrate valide găsite.

Pentru cerință 2 trebuie să contorizăm fiecare latură de pătrat separat și să le afișăm în ordine crescătoare.

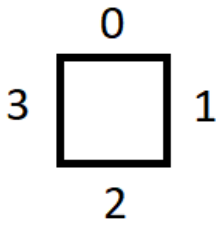
Complexitatea temporală este $O(nm \min(n, m)^2)$.

Cerința 3 Pentru această cerință:
1. rezolvăm cerința 1 pentru tabloul nemodificat, reținând numărul de pătrate P_{init};
2. parcurgem toate elementele din matrice și plasăm orice latură lipsă, apoi numărăm din nou pătratele obținute și reținem maximul și locul care a produs acest maxim P_{max};
3. dacă $P_{max} = P_{init}$, înseamnă că nu există niciun mod de a desena o singură linie pentru a mării numărul de pătrate, altfel afișăm rezultatul.

Complexitatea temporală este $O(n^2 m^2 \min(n, m)^2)$

Soluția 2 — stud. Bogdan-Ioan Popa, Facultatea de Matematică și Informatică, Universitatea din București

Această soluție obține 100 de puncte. În cele ce urmează vom numerota cei 4 pereți ai unei celule conform figurii de mai jos.



Cerințele 1 și 2 Se precalculează următoarele tablouri bidimensionale:

- $up_{i,j}$, lungimea secvenței maximale de pereți de tip 1 care are ca punct de oprire în celula (i, j) și se extinde în sus pe coloană;
- $left_{i,j}$, lungimea secvenței maximale de pereți de tip 2 care are ca punct de oprire în celula (i, j) și se extinde în stânga pe linie;
- $right_{i,j}$, lungimea secvenței maximale de pereți de tip 0 care are ca punct de oprire în celula (i, j) și se extinde în dreapta pe linie;
- $down_{i,j}$, lungimea secvenței maximale de pereți de tip 3 care are ca punct de oprire în celula (i, j) și se extinde în jos pe coloană.

Pentru fiecare celulă (i, j) și fiecare dimensiune de latură k se verifică dacă pătratul care are lungimea laturii k și colțul stânga sus în celula (i, j) are laturile desenate. Condițiile sunt următoarele.

$$down_{i,j} \geq k$$
$$right_{i,j} \geq k$$
$$up_{i+k-1,j+k-1} \geq k$$
$$left_{i+k-1,j+k-1} \geq k$$

Pentru prima cerință doar numărăm pătratele care respectă conditia de mai sus.

Pentru cea de-a doua cerință, vom cunstrui un vector de frecvență cnt_k = numărul de pătrate de latură k ce respectă condiția. Se afisează toate perechile (k, cnt_k), pentru care $cnt_k > 0$.

Complexitatea temporală este $O(nm \min(n, m))$.

Cerința 3 Pentru fiecare celulă (i, j) și pentru fiecare perete $0 \leq d \leq 3$ se verifică dacă peretele d al celulei (i, j) este desenat sau nu. Dacă peretele nu este desenat:

- se desenează (a nu se uita să se marcheze această desenare și celulei cu care se învecinează celula (i, j) prin peretele d);
- se rezolvă prima cerință considerând modificarea făcută;
- se actualizează maximul;
- se șterge peretele tocmai desenat.

Complexitatea temporală este $O(n^2 m^2 \min(n, m))$

Soluția 3 — Asist. Drd. Alexandru Ioniță, Universitatea „Alexandru Ioan Cuza", Iași

Pentru rezolvarea cerinței 3 există o soluție mai eficientă. Această soluție obține de asemenea 100 de puncte.

Soluția presupune următoarea precalculare. Fiecare linie orizontală din partea de sus a unei celule (i, j) din foaie va fi marcată într-o matrice: $sp^0_{i,j} = 1$. Peste matricea sp^0 vom face apoi sume parțiale pentru a putea afla in $O(1)$ câte linii din partea de sus sunt setate pe un anumit interval dintr-un rând. Vom repeta aceste operații pentru toate tipurile de linii rămase, generând cate o astfel de matrice pentru fiecare direcție rămasă: sp^1 — dreapta, sp^2 — jos, sp^3 — stânga.

Vom itera astfel prin fiecare patrat (determinat unic de o celula (x, y) si o lungime l) ce poate fi încadrat în foaia de dimensiune $n \times m$, având 3 cazuri:

1. pătratul are toate laturile complet acoperite de linii deja trasate;
2. pătratul mai are nevoie ca exact o singura linie sa fie trasată pentru a fi complet acoperit;
3. pătratul are nevoie de mai mult de o linie sa fie trasată pentru a fi complet acoperit.

Cazurile 1 si 3 nu ne interesează în mod special (trebuiesc doar numărate câte pătrate sunt in cazul 1, pentru a fi adăugate la răspuns).

În schimb, cazul 2 este de interes: pentru aceste pătrate trebuie să vedem care este linia lipsă. Daca notăm cu (x, y) celula liniei respective si cu z tipul liniei ($z = 0$ daca este `SUS`, $z = 1$, daca este `DREAPTA`, etc.) incrementând cu o unitate la coordonatele liniei într-un tablou tridimensional $used_{x,y,z}$.

Această marcare trebuie făcută și pentru celula vecina acestei linii (daca tocmai am analizat o linie de tipul `SUS` din celula (x, y), va trebui sa incrementăm valoarea in *used* și pentru celula $(x - 1, y)$ linia de `JOS`).

Observați că, făcând aceste marcări pentru fiecare pătrat, la final, în matricea *used* vom avea la fiecare poziție (x, y, z) numărul de pătrate care vor fi completate în cazul in care trasăm linia corespunzătoare acelei poziții.

Astfel, nu ne rămâne decât să căutăm care este celula cu cea mai mare valoare din vectorul *used* si să îi afișăm coordonatele.

Complexitatea temporală este $O(n^2m^2)$.

4.3.B Problema Pseudocmp

Propusă de: stud. Bogdan-Ioan Popa — Facultatea de Matematică și Informatică, Universitatea din București

Soluție — stud. Bogdan-Ioan Popa

Cerința 1 Această soluție la prima cerință obține 40 de puncte.

Dacă nu există nicio pereche specială, înseamnă că pentru oricare doi indici (i, j), dacă $A_i < A_j$ atunci $S_i \leq S_j$.

Această observație ne duce cu gândul la a sorta acest șir A. Pentru că sunt multe numere ($N \leq 100\,000$) și valorile din șir sunt mici ($A_i \leq 1\,000\,000$) vom folosi o sortare prin numărare. Vom introduce numerele într-un vector caracteristic, apoi îl vom parcurge de la 1 la 1 000 000 pentru a obține șirul sortat.

După sortare facem următoarea observație: nu va exista nicio pereche specială dacă și numai dacă $S_i \leq S_{i+1}$ pentru orice $1 \leq i < N$. Astfel, dacă șirul sortat respectă condiția enunțată mai sus, răspunsul va fi -1, altfel răspunsul va fi o pereche de numere A_k, A_{k+1} (în ordinea aceasta) pentru care $S_k > S_{k+1}$ și $1 \leq k < N$.

Complexitatea temporală este de $O(NC_{max} + Val_{max})$, unde Val_{max} reprezintă valoarea maximă a unui număr din șir, iar C_{max} reprezintă numărul maxim de cifre ale unui număr din șir.

Cerința 2 Această soluție la a doua cerință obține 60 de puncte.

Cum șirul A este sortat crescător, perechile speciale vor respect condiția $i < j$. Rămâne să calculăm pentru fiecare $1 \leq j \leq N$ câți indici i există astfel încât $1 \leq i < j$ și $S_i > S_j$.

La fiecare pas vom ține vectorul de frecvență $freq_s$ = câți indici i aflați la stânga lui j există astfel încât $S_i = s$. Apoi însumăm valorile $freq_s$, pentru fiecare s de la $S_j + 1$ la S_{max}, unde S_{max} reprezintă suma maximă a cifrelor unui număr. Odată cu calcularea acestei sume, putem actualiza vectorul *freq*, crescând $freq_{S_j}$ cu 1.

Complexitatea temporală este $O(NS_{max} + Val_{max})$.

4.4 Clasa a VIII-a

4.4.A Problema Pelican

Propusă de: prof. Nistor Moţ — Şcoala Gimnazială „Dr. Luca", Brăila

Problema admite mai multe abordări. O posibilă abordare este de a citi succesiv comenzile pelicanului şi de a executa fiecare comandă pentru fiecare dintre cele P raţe. Este evident o abordare corectă, dar care va depăşi timpul de execuţie. Totuşi această abordare poate fi îmbunătăţită bazându-ne pe următoarele observaţii:

1. Dacă în fişierul de intrare există o comandă Z, atunci vom executa această comandă pentru toate raţele. Dacă există mai multe comenzi Z, doar ultima contează, deci doar aceasta va fi executată. Toate comenzile A până la comanda Z executată pot fi ignorate, doar comenzile R contează. În plus, observăm că nu trebuie să executăm fiecare comandă R separat, putem cumula numărul de grade şi putem executa o singură rotaţie finală pentru toate raţele.
2. Ideea de cumulare poate fi utilizată mai departe. Pentru a reduce numărul de comenzi executate de toate raţele, putem lucra pe secvenţe de comenzi A, respectiv secvenţe de comenzi R. Pentru fiecare secvenţă de comenzi identice vom cumula nr şi la finalul secvenţei executăm o deplasare/rotaţie cumulată pentru toate raţele. Desigur, pentru deplasări lucrăm modulo N, iar pentru rotaţii lucrăm modulo 4.

Aceste observaţii pot conduce la un punctaj bun, dar pentru 100 de puncte trebuie observat că două raţe care au iniţial aceeaşi orientare se vor deplasa „solidar", adică păstrează aceeaşi poziţie relativă. Dacă avem două raţe cu aceeaşi orientare în poziţiile iniţiale (i_1, j_1) şi (i_2, j_2), comanda A nr va duce ambele raţe în poziţii care păstrează aceeaşi distanţă între ele. De exemplu, pentru orientarea Sud cele două raţe vor ajunge în poziţiile $(i_1 + nr, j_1)$ şi respectiv $(i_2 + nr, j_2)$. Această observaţie ne permite să efectuăm comenzile doar cu 4 raţe „virtuale" ce pornesc din poziţia (0,0) având orientarile N, E, S, respectiv V.

La final, pentru fiecare raţă vom determina poziţia sa finală în funcţie de poziţia raţei virtuale care avea iniţial aceeaşi orientare cu raţa respectivă. Dacă raţa virtuală ajunge în poziţia finală (x, y), atunci o raţă care iniţial avea aceeaşi orientare cu cea virtuală şi era plasată în poziţia (i_1, j_1) va ajunge în poziţia finală $(i_1 + x, j_1 + y)$. Evident, toate deplasările se fac modulo N. La fel, rotaţiile se execută modulo 4.

Mai mult, dacă dorim, putem renunţa la a calcula poziţia finală pentru toate cele 4 direcţii şi să efectuăm mutările doar pentru o singură direcţie, celelalte putând fi deduse prin rotaţii cu câte 90 de grade. De exemplu, considerând că o raţă ce porneşte din poziţia 0,0 având orientarea iniţială N ajunge în poziţia finală (x, y), atunci raţa din pozitia (i_1, j_1) ajunge:

- dacă are orientarea iniţială N, în poziţia finală $(i_1 + x, j_1 + y)$
- dacă are orientarea iniţială S, în poziţia finală $(i_1 - x, j_1 - y)$
- dacă are orientarea iniţială E, în poziţia finală $(i_1 + y, j_1 - x)$
- dacă are orientarea iniţială V, în poziţia finală $(i_1 - y, j_1 + x)$.

4.4.B Problema Strips

Propusă de: prof. Emanuela Cerchez — Colegiul Naţional „Emil Racoviţă" Iaşi

Soluţia 1 (40 de puncte)

Pentru $N \leq 10^6$, putem utiliza un vector pentru a reprezenta culorile benzilor plasate pe tabla de joc (0 pentru poziţie neocupată, 1 pentru o poziţie ocupată de o bandă roşie, respectiv 2 pentru o poziţie ocupată de o bandă verde). Citim succesiv mutările. Pentru fiecare mutare verificăm dacă este validă. Dacă da, plasăm pe tabla de joc o bandă în poziţia respectivă. Dacă nu, contorizăm un punct penalizare pentru jucătorul care este la rând. Pentru cerinţa 2 este suficient să parcurgem tabla de joc şi să determinăm lungimea maximă a unei secvenţe formată numai din 1, respectiv lungimea maximă a unei secvenţe formată numai din 2.

Soluţia 2 (100 de puncte)

Pe măsură ce plasăm benzi pe tabla de joc formăm practic zone continue de aceeaşi culoare, care nu se suprapun. În loc să reţinem tabla de joc, vom reţine pentru fiecare jucător zonele obţinute din benzile plasate de acesta pe tabla de joc. O zonă va fi reprezentată prin cele două extremităţi (poziția de început şi poziția de sfârșit a zonei).

```
struct zona { int inc, sf; };
```

Considerăm că jucătorul 0 este Ana şi jucătorul 1 este Bogdan. Structurile de date utilizate sunt:

```
zona Z[2][NRMAX]; //zonele celor doi jucatori
int nrz[2];       //numarul de zone pentru fiecare jucator
int p[2];         //penalizarea fiecarui jucator
int lgmax[2];     //lungimea maxima a unei zone pentru fiecare jucator
```

NRMAX este numărul maxim de zone ale unui jucător (se garantează că la finalul jocului *NRMAX* nu depășește 5 000, dar pe parcursul jocului poate fi mai mare).

Vom reţine zonele în ordinea crescătoare a pozițiilor de început.

Vom citi succesiv mutările celor doi jucători. La o mutare verificăm mai întâi validitatea acesteia. Să notăm *poz* poziţia corespunzătoare mutării curente şi cu *cine* jucătorul care este la mutare. Evident, adversarul va fi $1 - cine$.

Pentru a verifica validitatea vom căuta pe tabla adversarului cea mai mică poziţie (*unde*) pentru care $poz \leq Z[adversar][unde].inc$. Zonele fiind sortate, vom face o căutare binară.

Dacă o bandă plasată în poziţia *poz* se intersectează cu sau de lipește de cea de pe poziţia *unde* (la dreapta) sau cu cea de pe poziţia $unde - 1$ (la stânga) atunci mutarea nu este validă.

În cazul în care mutarea este validă, plasăm o bandă pe tabla jucătorului care este la mutare. Pentru aceasta facem din nou o căutare binară, de data aceasta în zonele jucătorului care este la mutare şi determinăm cea mai mică poziţie (*unde*) pentru care $poz \leq Z[cine][unde].inc$.

Pot apărea următoarele situaţii:

1. banda poate fi alipită zonei $Z[cine][unde]$, dacă această zonă există şi

$$poz + L - 1 \geq Z[cine][unde].inc - 1;$$

2. banda poate fi alipită zonei $Z[cine][unde-1]$, dacă această zonă există şi

$$poz \leq Z[cine][unde-1].sf + 1;$$

3. în urma alipirii zonele $Z[cine][unde-1]$ şi $Z[cine][unde]$ pot fi şi ele alipite, formând astfel o singură zonă (le alipim şi eliminăm una dintre ele);
4. dacă nicio alipire nu este posibilă, inserăm o nouă zonă cu începutul *poz* şi sfârşitul $poz + L - 1$ în poziţia *unde* (astfel zonele rămânând sortate).

Eficienţa timp a acestui algoritm este afectată de situaţiile 3 sau 4 (eliminarea, respectiv inserarea fiind liniară).

Utilizarea unor structuri de date avansate (de exemplu, structura de date set din STL) poate optimiza acest pas, dar o astfel de implementare depăşeşte nivelul clasei a VIII-a și nu este necesară pentru a obține 100 de puncte.

Este posibilă o implementare bazată pe aceeaşi idee, asociind fiecărei zone culoarea acesteia şi reţinând zonele sortate după extremitatea iniţială într-un singur vector. Dar în acest caz dimensiunea vectorului este mai mare şi timpul necesar pentru eliminare/inserare creşte.

4.5 Clasa a IX-a

4.5.A Problema Bâlbă

Propusă de: stud. Gheorghe Liviu Armand

Cerința 1 (40 de puncte)

Pentru a rezolva această cerință, trebuie să facem anumite observații asupra modului în care un număr se schimbă după o bâlbâială.

Observăm că dacă în numărul X se află 2 cifre egale alăturate, se va obține același număr dacă cifra pe care o va repeta regele, adică cifra bâlbâită, este prima dintre acestea sau a 2-a dintre acestea. De asemenea, dacă în numărul X există 2 cifre egale care nu sunt alăturate, se vor obține numere diferite dacă cifra pe care o va repeta regele este prima dintre acestea sau a 2-a dintre acestea. Evident, numărul obținut prin bâlbâirea unei cifre va fi diferit de numărul obținut prin bâlbâirea unei cifre diferite ca valoare.

Astfel, deducem că prin bâlbâirea oricărei cifre dintr-o secvență de cifre egale din numărul X se va obține exact același număr, iar prin bâlbâirea unei cifre care nu este în acea secvență se va obține un număr diferit de cel obținut anterior.

Așadar, numărul de numere diferite pe care X le poate genera printr-o bâlbâiala este, de fapt, numărul de secvențe de cifre egale din numărul X.

Pentru a afla numărul de numere diferite care devin X după o bâlbâială, ne bazăm tot pe observațiile trecute și pe faptul că dacă o cifră a fost bâlbâită, atunci în numărul X aceasta se va afla într-o secvență de cifre egale formată din minim 2 cifre (cu alte cuvinte, cifra care a fost bâlbâită în numărul inițial nu se poate regăsi în X într-o secvență de cifre egale formată doar dintr-o cifra).

Deci, numărul de numere diferite care devin X după o bâlbâială este, de fapt, numărul de secvente de cifre egale de lungime minim 2 din numărul X.

O soluție a acestei cerințe are complexitatea timp $O(n)$.

Cerința 2 (60 de puncte)

Pentru rezolvarea acestei cerințe există mai multe abordări, iar în continuare vom prezenta 3 dintre aceste soluții care obțin punctajul maxim. Am notat cu Σ numărul total de cifre diferite care ar putea apărea în numărul X, întrucât problema s-ar putea generaliza, lucrând în acest fel cu un Σ mult mai mare. În cazul nostru $\Sigma = 10$.

Mai întâi vom pune cifrele numărului X într-un vector de frecvență pentru a putea lucra mai ușor și mai rapid cu ele.

Un prim aspect important este că un număr palilindrom este un palindrom fără o cifră (și anume cifra care se va adăuga după bâlbâială). Observăm că într-un palindrom, fiecare cifră apare de un număr par de ori, mai puțin eventuala cifră care se poate afla în mijloc dacă acesta are lungime impară. Așadar, într-un palindrom avem maxim o cifră de frecventă impară, iar restul cifrelor au frecvență pară. Astfel, deducem ca un palilindrom poate avea maxim 2 cifre

de frecventă impară și restul de frecventă pară, întrucât bâlbâirea unei cifre va avea ca efect schimbarea parității frecvenței cifrei respective.

Deci, atenția noastră va pica asupra cifrei bâlbâite și a cifrei din mijloc. (Aveți grijă la cazurile când acestea au aceeași valoare sau coincid!.) Dacă găsim un mod bun de a afla aceste cifre, atunci va trebui să vedem cum putem insera doar câte o apariție a fiecărei dintre aceste 2 cifre în cel mai mare palindrom care se poate forma cu restul cifrelor din număr. Cifra care se va bâlbâi și pe care noi o vom insera nu este fix cifra care se va bâlbâi, dar este cifra simetrica a cifrei bâlbâite în palindromul creat după bâlbâială. Pentru simplitate, ne vom referi la aceasta ca „cifra bâlbâită". (Nu este obligatoriu să avem mereu o cifră bâlbâită și o cifră din mijloc, întrucât sunt multe cazuri în care în cel mai mare palilindrom care se poate forma nu trebuie sa aibă o singură cifră în mijloc sau o cifră bâlbâită care nu e în mijloc.)

Soluția 1 Fixam valoarea cifrei care se va bâlbâi și valoarea cifrei care va fi în mijloc. (Vom analiza și cazurile în care aceste cifre sunt inexistente.)

Întrucât am fixat, în complexitate $O(\Sigma^2)$, cele 2 cifre, tot ce mai trebuie sa facem este să vedem care este cel mai mare palindrom (de lungime pară) care se poate forma folosind restul cifrelor, lucru care se poate face în complexitate $O(N)$.

Cifra bâlbâită (care nu e în mijloc) va trebui adăugată în palindromul calculat fie în jumătatea stângă, fie în jumătatea dreaptă. O observație importantă este că se obține un palindrom mai mare dacă este adăugată în partea stângă. Din toate aceste palilindroame obținute se va afișa maximul.

Complexitate temporală a acestei soluții este $O(N\Sigma^2)$.

Soluția 2 Observăm că putem fixa doar valoarea cifrei care se va bâlbâi (și nu e în mijloc) și că nu mai e nevoie să fixăm și valoarea cifrei din mijloc, întrucât aceasta poate fi determinată: este cea mai mare cifră cu frecvența impară!

Așadar, ne fixăm în complexitate $O(\Sigma)$ valoarea cifrei care se va bâlbâi și apoi vom calcula cel mai mare palindrom (de orice lungime de data aceasta) care se poate forma folosind restul cifrelor, lucru care se poate face în complexitate $O(N)$. Din toate aceste palilindroame obținute se va afișa maximul.

Complexitate temporală a acestei soluții este $O(N\Sigma)$.

Soluția 3 Observăm că nu mai este nevoie sa fixăm valoarea cifrei care se va bâlbâi, întrucât aceasta poate fi determinată: este cea mai mare cifră cu frecvența impară și minim 3.

La această soluție sunt mai multe cazuri de analizat decât la celelalte, întrucât acum trebuie sa determinăm și cifra bâlbâită și cea mai mare cifră din mijloc, iar una dintre determinări o poate influența pe cealaltă.

Astfel, această soluție are cea mai bună complexitate temporală: $O(N)$.

4.5.B Problema Oneout

Propusă de: prof. Chește Ciprian

Soluția 1 — prof. Chește Ciprian

Se determină un nou șir w pentru care $w[i] = 0$ dacă $v[i]$ este liber de pătrate și $w[i] = 1$ dacă $v[i]$ nu este liber de pătrate, folosind descompunerea în factori primi.

Utilizând acest șir se identifică pozițiile elementelor de 1 care au „vecini" elemente de 0. Din aceste poziții se extinde secvența către stânga și către dreapta. Se actualizează lungimea maximă dacă este cazul.

Pentru a determina câte secvențe de lungime maximă sunt se parcurge încă odată șirul și apoi încă odată pentru a afișa indicii de start și de final ai secvențelor de lungime maximă. Soluția are ordinul de complexitate $O(n\sqrt{n})$.

Soluția 2 — prof. Chește Ciprian Se determină șirul numerelor libere de pătrate printr-o metoda de tip ciur asemănătoare cu ciurul lui Eratostene.

Se poate folosi următorul algoritm:

```
// Fie max cea mai mare valoare din sirul v.

for(i = 1; i <= max; i++)
    w[i] = 0;
for(i = 2; i*i <= max; i++)
    for(j = i*i; j <= max; j += i*i)
        w[j] = 1;
```

Sensul algoritmului este următorul:

1. se inițializează șirul w cu 0 (presupunem că toate numerele sunt libere de pătrate),
2. am eliminat toate pătratele perfecte și multiplii acestora.

Se urmează apoi etapele de la soluția anterioara folosind șirul numerelor libere de pătrate generat anterior.

Deoarece metoda de determinare a șirului numerelor libere de pătrate are ordin de complexitate $O(n \log n)$ soluția obține 100 puncte.

Soluția 3 — stud. Gheorghe Liviu Armand După ce determinăm dacă un număr e liber de pătrate, atunci putem proceda la fel ca la soluțiile descrise anterior pentru a afla numărul de numere libere de pătrate și lungimea celei mai lungi bisecvențe care respectă condițiile din enunt.

Dar cum aflam dacă un număr x este liber de pătrate? Putem lua în ordine crescătoare fiecare număr prim p și să determinăm dacă x este divizibil cu pătratul lui. Dacă este, atunci x nu este liber de pătrate și ne putem opri. Dacă nu este, atunci înseamnă că p se afla fie la puterea 0, fie la puterea 1 în descompunerea lui x în factori primi. Dacă p se află la puterea 1, îl vom împărți pe x la p, întrucât, în acest fel, va mai trebui să parcurgem mai puține numere prime decât dacă nu am face aceasta împărțire.

Principala idee a acestei soluții este următoarea: dacă am realiza procedeul acesta pentru toate numerele prime care *ar putea* apărea la puterea a-3-a în descompunerea în factori primi

a lui x și dacă toate verificările cu aceste numere prime nu au condus încă la un verdict exact, atunci numărul meu x rămas după eventualele împărțiri poate fi:

- fie un număr prim q la puterea 1, adică $x = q$;
- fie un număr prim q la puterea 2, adică $x = q^2$;
- fie un produs de 2 numere prime q si r, adică $x = qr$.

Observăm că x nu poate să fie un produs de 2 numere prime q și r astfel încât $x = qr^2$ sau $x = q^2r$ sau $x = q^2r^2$. Presupunem, prin absurd, că x poate să fie sub una dintre aceste forme. Fără a restrânge generalitatea problemei, presupunem că $q \leq r$. Atunci $q^3 \leq x$. Dar conform procedeului descris mai sus, am eliminat din descompunerea în factori primi a lui x orice număr prim care *ar fi putut* fi la o putere mai mare sau egală cu 3. Așadar, după terminarea procedeului, q nu ar mai fi apărut în descompunerea în factori primi a lui x. Deci, avem o contradicție cu enunțul. Presupunerea făcută este falsă. Așadar, x poate fi numai sub una din cele 3 forme bune enumerate mai sus.

Din cele 3 cazuri bune, observăm că x nu este liber de pătrate doar atunci când este un pătrat perfect. Astfel, putem întreba după fiecare pas dacă nu cumva x a devenit pătrat perfect și ne putem opri. Dacă s-a terminat tot procedeul, înseamnă că x este liber de pătrate.

O altă optimizare pe care o putem face este ca pentru fiecare număr x întâlnit pe tot parcursul procedeului (deci și cel inițial și cei obținuți după împărțirile care au loc în cadrul procedeului) să ținem minte rezultatul obținut la sfârșitul procedeului (adică facem memoizare). Acest lucru este corect, întrucât observăm că toate numerele x întâlnite pe parcursul unui procedeu sunt de același tip: fie libere de pătrate, fie nu sunt libere de pătrate.

Astfel, cand vom întâlni pe parcursul procedeului un x deja calculat, vom putea opri procedeul, întrucât știm ce rezultat va da, deoarece ni l-am salvat când l-am întâlnit prima oară.

Așadar, acest procedeu va itera doar prin primele 25 de numere prime (în cazul limitelor din această problemă) în cazul cel mai nefavorabil. În practică se comporta mult mai bine și de multe ori va încheia procedeul înainte să parcurgă cele 25 de numere prime. Această soluție este un pic mai lenta decât soluția anterioară, însa se descurcă foarte bine cu numere mai mari.

În concluzie, această soluție obține 100 de puncte. Are complexitate timp $O(25N)$, însă se comporta mai bine de atât în practică.

4.5.C Problema Pergament

Propusă de: student Mihaela Cismaru

Soluția 1 (40 de puncte)

Pentru obținerea punctajului se poate implementa o soluție brută în care păstram structura pergamentului într-o matrice de $N \times K$ poziții. Notăm în matrice toate pozițiile prin care trece o stradă verticală sau orizontală. După adăugarea tuturor străzilor, prin parcurgerea matricei poziție cu poziție se pot număra toate intersecțiile dintre cele două tipuri de străzi. Complexitatea acestei soluții este $O(NK)$.

Soluția 2 (70 de puncte)

Pentru obținerea a încă 30 de puncte este necesară stocarea pe rând în memorie a câte o coloana din pergament, respectiv un vector de N poziții. Pentru fiecare poziție a unei coloane putem verifica dacă prin aceasta trece strada orizontală a rândului de care aparține. Facem transformarea tuturor străzilor verticale ce aparțin de fiecare coloană în poziții ce determină începerea sau terminarea unei străzi verticale. Sortând crescător aceste poziții pentru o coloana și parcurgând de sus în jos, putem ști în timp real dacă ne aflăm pe o stradă verticală sau nu. Adunând toate intersecțiile de pe fiecare coloană obținem astfel numărul final cerut. Complexitatea acestei soluții este tot $O(NK + Q \log Q)$ dar memoria folosită va fi de doar $O(N + Q)$.

Soluția 3 (100 de puncte)

Pentru obținerea punctajului maxim putem păstra în memorie doar străzile verticale și vectori de dimensiune a doar K poziții. Transformăm datele străzilor verticale în poziții de început și de final a unei străzi verticale și le sortăm după rând (complexitate $Q \log Q$). Parcurgem fiecare rând de sus în jos, ținând minte într-un vector pozițiile în care se afla o stradă verticală și un alt vector în care vom calcula sumele parțiale ale vectorului anterior. La fiecare poziție unde se termină sau începe o stradă verticală vom recalcula cei doi vectori. Numărul acestor evenimente va fi de exact $2Q$ iar recalcularea vectorului de sume parțiale va avea complexitate $O(K)$ în total având o complexitate de doar $O(QK)$. Pentru fiecare rând putem afla câte intersecții există interogând intervalul străzi orizontale de pe acel rând în vectorul din sume parțiale, interogare cu o complexitate de doar $O(1)$. Se vor face exact N astfel de interogări, astfel soluția finală va avea complexitatea optimă de $O(N + KQ + Q \log Q)$

4.6 Clasa a VII-a

Mulțumim studentului Gabriel Tulbă-Lecu pentu munca depusă în aceste editoriale.

4.6.A Problema Circular

Propusă de: prof. Boca Alina Gabriela — Colegiul Național de Informatică „Tudor Vianu" București

Cerința 1

Pentru rezolvarea primei cerințe se parcurge șirul de litere albastre și pentru oricare două litere alăturate se calculează cea mai mică distanță dintre litera curentă și litera următoare din șir.

Pentru a calcula distanta minimă dintre două litere A_i și A_{i+1} va trebui sa calculăm minimul dintre cele două cazuri:

- de la A_i la A_{i+1} în sensul acelor de ceasornic
- de la A_i la A_{i+1} în sens opus acelor de ceasornic

Pentru a lua în considerare faptul că imprimanta începe de pe poziția A, putem considera $A_0 = \texttt{A}$.

Complexitatea temporală este $O(N)$. Rezolvarea primei cerințe obține 24 de puncte.

Cerința 2

Pentru a rezolva cerința 2, vom precalcula un tablou bidimensional de 26×26:

$$cost_{i,j} = \text{numărul minim de pasi pentru a ajunge de la a } i\text{-a litera din alfabet, la a } j\text{-a.}$$

Ulterior, se parcurge șirul literelor albastre și între fiecare două litere albastre A_i, A_{i+1}, se caută în șirul literelor roșii acele litere R pentru care distanța $cost_{A_i,R} + cost_{R,A_{i+1}}$ este minimă.

Costul minim reprezintă suma distantelor minime obținute, la care trebuie adăugat costul de a aduce capul de printare de la $L_0 = \texttt{A}$ la L_1.

Pentru a construi șirul minim lexicografic, între oricare două litere albastre, vom insera dintre toate literele roșii ce obțin un cost minim pe cea mai mică lexicografic.

Numărul de soluții reprezintă produsul dintre numărul de posibilități de a insera o literă roșie între două litere albastre care generează distanța minimă.

Complexitatea temporală este: $O(N\Sigma)$, unde $\Sigma = 26$ reprezintă dimensiunea alfabetului. Rezolvarea celei de-a doua cerințe obține 76 de puncte.

4.6.B Problema Pulsar

Propusă de: stud. Tulbă-Lecu Theodor-Gabriel — Universitatea Politehnica din București

Observații

Pentru a rezolva problema, inițial trebuie făcută următoarea observație: După un anumit timp minim T, pulsarele vor reveni înapoi în starea inițială de la momentul de timp $t = 0$. Vom numi acest timp T, *perioada pulsarelor*.

În continuare, trebuie făcută observația că dacă toate pulsarele revin în starea inițială după T unități de timp, cum un pulsar P_i de perioadă r_i, se află în starea inițială doar la momente de timp care sunt multiplii de r_i, atunci T este și el multiplu al lui r_i.

Astfel, T este cel mai mic multiplu comun al perioadelor pulsarilor: $T = \text{cmmmc}(r_1, r_2, \ldots, r_P)$. Cum pentru restricțiile problemei: $1 \leq r_i \leq 6$, oricare ar fi $1 \leq i \leq P$, rezultă că $T \leq 60$.

Cerința 1

Subtask 1 În urma observațiilor făcute, cerința 1, poate fi exprimată astfel: pentru fiecare moment de timp de la 0 la $T - 1$ câte sectoare ale galaxiei din cele $N \times N$ sunt afectate de cel puțin un pulsar?

Acest lucru poate fi rezolvat calculând pentru fiecare moment de timp, un tablou bidimensional:

$$afectat_{i,j} = \begin{cases} 1, & \text{dacă există cel puțin un pulsar care afectează sectorul } (i,j), \\ 0, & \text{altfel.} \end{cases}$$

Tabloul bidimensional *afectat* poate fi calculat prin marcarea pentru fiecare pulsar în parte a tuturor sectoarelor afectate de acesta la momentul de timp actual cu 1, toate valorile din *afectat* fiind inițial inițializate cu 0.

Răspunsul pentru cerința 1 este maximul dintre numărul de valori de 1 din tabloul *afectat*, pentru fiecare moment de timp de la 0 la $T - 1$.

Complexitatea temporală este $O(T(N^2 + P{R_{max}}^2))$, unde R_{max} este perioada maximă a unui pulsar.

Pentru rezolvarea corectă a cerinței 1 se pot obține 19 puncte.

Cerința 2

Subtask 2 Primul subtask al cerinței a doua reprezintă un caz particular al problemei. Dacă $r_i = 1$, oricare ar fi $1 \leq i \leq P$, atunci toti pulsarii vor afecta doar sectoarele în care aceștia se află.

Astfel, problema devine găsirea un drum de lungime minimă de la sectorul (x_s, y_s) la sectorul (x_f, y_f) într-o hartă ce conține obstacole.

Acest lucru se poate rezolva utilizând algoritmul lui Lee[1].

Complexitatea temporală este: $O(N^2 + P)$.

Pentru rezolvarea corectă a subtaskului 2 se pot obține 22 de puncte.

[1] https://www.pbinfo.ro/articole/18589/algoritmul-lui-lee, articol pbInfo — Algoritmul lui Lee, Prof. Silviu Candale

Subtask 3 Pentru acest subtask, cum $N \leq 10$, harta galaxiei are dimensiuni suficient de mici pentru ca problema să fie rezolvată utilizând metoda backtracking. Se vor genera toate drumurile posibile de la sectorul (x_s, y_s) la sectorul (x_f, y_f), iar la fiecare pas se verifică dacă celula adăugată la drum, este afectată de vreun pulsar.

Subtask-urile 4 și 5 Pentru a rezolva complet cerința a doua, trebuie să ne folosim de observațiile făcute anterior.

Știind că harta galaxiei este periodică cu o perioadă T putem să reprezentăm starea navei sub forma unui triplet (x, y, t) unde x și y reprezintă coordonatele navei, iar t reprezintă starea hărții galaxiei.

Dacă nava se află la într-o stare (x, y, t), aceasta la următorul moment de timp se va afla la:

1. $(x, y, (t+1) \bmod T)$, dacă nava stă pe loc,
2. $(x, y-1, (t+1) \bmod T)$, dacă nava se deplasează la stânga,
3. $(x, y+1, (t+1) \bmod T)$, dacă nava se deplasează la dreapta,
4. $(x+1, y, (t+1) \bmod T)$, dacă nava se deplasează în jos,
5. $(x-1, y, (t+1) \bmod T)$, dacă nava se deplasează în sus.

Astfel, nava se va deplasa intr-un tabel tridimensional, a treia dimensiune fiind starea hărții, iar obstacolele vor fi create de zonele afectate de pulsari. Această problemă se poate rezolva tot cu ajutorul algoritmului lui Lee, care va trebui modificat pentru a funcționa pe trei dimensiuni.

Răspunsul va fi timpul minim cu care se poate ajunge din $(x_s, y_s, 0)$ în (x_f, y_f, t) pentru oricare $0 \leq t < T$.

Complexitatea temporală este: $O(T(N^2 + PR_{max}{}^2))$.

4.6.C Problema Transport

Propusă de: stud. Cotor Andrei — Universitatea „Babeș-Bolyai" din Cluj-Napoca

Observații

1. O rută Regio reprezintă o subsecvență din șirul de stații de forma: $[st, st+1, st+2 \ldots, dr]$, $1 \leq st < dr \leq N$, unde st și dr reprezintă capetele rutei.
2. O rută Expres reprezintă un subșir din șirul de stații de forma: $[st, i_1, i_2, \ldots, i_k, dr]$, $1 \leq st < i_1 < i_2 < \cdots < i_k < dr \leq N$, $k \geq 0$, unde st și dr reprezintă capetele rutei.
3. Relația care corespunde restricțiilor din enunț e: $D_{st} + D_{dr} = C(X_{dr} - X_{st})$. Aceasta poate fi rescrisă în felul următor: $D_{st} + D_{dr} = C(X_{dr} - X_{st}) \implies D_{st} + D_{dr} = CX_{dr} - CX_{st} \implies D_{st} + CX_{st} = CX_{dr} - D_{dr}$
4. În relația obținută la observația 3 expresia din stânga egalului depinde doar de st (X_{st}, D_{st}), iar cea din dreapta egalului doar de dr (X_{dr}, D_{dr})

Cerința 1

Subtask 1 (12 puncte) Se parcurge fiecare subsecvență, fixând capătul din stânga și cel din dreapta, și se verifică condiția $D_{st} + D_{dr} = C(X_{dr} - X_{st})$. Dacă este respectată condiția se incrementează rezultatul.

Complexitate temporală: $O(N^2)$.

Optimizare Soluția pentru subtask-ul 1 poate fi optimizată parcurgând, pentru un dr fixat, doar indicii st pentru care (st, dr) pot forma o pereche de capete validă. Mai exact, după ce a fost fixat dr, se calculează valoare expresiei $CX_{dr} - D_{dr}$ și se parcurg doar indicii st pentru care $D_{st} + CX_{st} = CX_{dr} - D_{dr}$ (observația 3).

Cu ajutorul acestei optimizări se pot obține 12 puncte din cele 26 acordate pentru subtask-ul 2 (în plus față de cele acordate pentru subtask-urile precedente).

Subtask 2 (26 puncte)

Se fixează capătul dreapta dr. Trebuie numărate câte capete stânga st există astfel încât perechea de capete (st, dr) este una validă, mai exact $D_{st} + CX_{st} = CX_{dr} - D_{dr}$ (observația 3).

Pentru a obține aceste valori va fi nevoie de o normalizare, valorile expresiilor $D_{st} + CX_{st}$ sau $CX_{dr} - D_{dr}$ putând fi foarte mari. Astfel pentru fiecare stație i se reține într-un vector de lungime $2N$, care urmează să fie utilizat pentru normalizare, valorile $D_i + CX_i$ și $CX_i - D_i$.

Complexitate temporală: $O(N \log N)$.

Cerința 2

Subtask 3 (6 puncte) N fiind mic se poate utiliza bactracking pentru a genera toate subșirurile din șirul de stații. Pentru fiecare subșir generat se verifică dacă respectă condițiile din enunț.

Subtask 4 (15 puncte) Se fixează fiecare combinație de două capete ale unei rute (notate cu *st*, respectiv *dr*), care respectă condițiile din enunț. Între cele două capete fixate există $dr - st - 1$ stații. Astfel pentru perechea de capete (st, dr) numărul de rute Expres este egal cu numărul de subșiruri care se pot forma din subsecvența $[st + 1, st + 2, \ldots, dr - 1]$, adică $2^{dr-st-1}$.

Complexitate temporală: $O(N^2)$ sau $O(N^2 \log N)$ în funcție de implementare.

Optimizare Optimizarea prezentată pentru cerința 1 poate fi utilizată și în acest caz.

Cu ajutorul acestei optimizări se pot obține 14 puncte din cele 41 acordate pentru subtask-ul 5 (în plus față de cele acordate pentru subtask-urile precedente).

Subtask 5 (41 puncte) În mod asemănător cu subtask-ul 2, se face normalizarea și se fixează capătul dreapta *dr*. Fie $\{st_1, st_2, st_3 \ldots st_k\}$ mulțimea de capete stanga cu care *dr* formează o pereche de capete validă. Astfel numărul de rute valide care îl au capăt dreapta pe *dr* este

$$2^{dr-st_1-1} + 2^{dr-st_2-1} + 2^{dr-st_3-1} + \cdots + 2^{dr-st_k-1}.$$

Relația se prelucrează și se obține

$$2^{dr} \left(\frac{1}{2^{st_1+1}} + \frac{1}{2^{st_2+1}} + \frac{1}{2^{st_3+1}} + \cdots + \frac{1}{2^{st_k+1}} \right).$$

Relația se înmulțește și se împarte cu 2^N și se obține

$$2^{dr-N}(2^{N-st_1-1} + 2^{N-st_2-1} + 2^{N-st_3-1} + \cdots + 2^{N-st_k-1}).$$

Suma $2^{N-st_1-1} + 2^{N-st_2-1} + 2^{N-st_3-1} + \cdots + 2^{N-st_k-1}$ se calculează în timpul parcurgerii pentru fixarea capătului dreapta.

Astfel complexitatea temporală este $O(N \log N)$.

4.7 Clasele XI–XII

4.7.A Problema Dulciuri

Propusă de: stud. doctorand Tamio-Vesa Nakajima — Facultatea de Informatică, Universitatea Oxford

Subtaskul 1 (20 puncte)

Pentru primul subtask, observăm că soluția pentru o degustare este dată de raportul dintre suma îndulcirilor care au afectat acea degustare, și distanța orizontală pe care o parcurge degustarea. Prima valoare se poate calcula efectiv parcurgând îndulcirile relevante.

Subtaskul 2 (20 puncte)

Pentru al doilea subtask, observăm ca soluția pentru o interogare verticală este dată de suma îndulcirilor verticale care afectează acea interogare, plus rezultatul de la subtaskul 1. Acestea se pot calcula asemănător cu subtaskul 1 — parcurgând efectiv toate îndulcirile de până acum. Totodată o interogare orizontală se rezolvă analog cu o interogare verticală.

Subtaskul 3 (10 puncte)

Vom face acum câteva observații suplimentare.

Observatia 1. *Dulceața totală este egală cu dulceața datorată îndulcirilor verticale plus dulceața datorată îndulcirilor orizontale.*

Astfel, la fiecare pas ne interesează doar dulceața datorată îndulcirilor orizontale sau verticale *separat* — în alte cuvinte, îndulcirile verticale și orizontale sunt *independente*. Vom descrie cum se calculează dulceață datorată îndulcirilor verticale, cele orizontale tratându-se analog. Așadar, care este această dulceață?

Observatia 2. *Considerăm o interogare de la (x, y) la (x', y'). Fie v_i suma tuturor îndulcirilor verticale pentru fâșia $i \leq x < i + 1$. Presupunem fără pierdere de generalitate că $x \leq x'$. Dacă $x = x'$ atunci dulceața datorată îndulcirilor verticale este v_x; altfel este:*

$$\frac{\sum_{i=x}^{x'-1} v_i}{x' - x}.$$

Pentru al treilea subtask, valoarea din observația 2 se poate calcula efectiv, iterând prin toate îndulcirile.

Subtaskul 4 (20 de puncte)

Pentru al patrulea subtask, valoarea din observația 2 se poate calcula folosind sume parțiale pe șirul v. Această tehnică se mai numește șmenul lui Mars.

Subtaskul 5 (10 puncte)

Pentru acest subtask observăm că șirul valorile din observația 2 se pot calcula parcurgând toate îndulcirile precedente.

Soluție completă

În acest subtask, pentru a calcula valorile din observația 2, menținem o structură de date ce va reprezenta secvența v. Această structură de date trebuie să poată simula incrementarea unui element v_i, și trebuie să poată calcula suma unei subsecvențe din șirul v_i. Mai multe structuri de date pot realiza acest lucru suficient de eficient pentru rezolvarea problemei: arbori indexați binari, arbori de intervale sau împărțirea în bucăți de $\sqrt{n}$.

4.7.B Problema Investiţie

Propusă de: prof. Mihai Bunget — Colegiul Naţional „Tudor Vladimirescu", Târgu Jiu

Rezolvarea problemei presupune cunoştinţe despre sume parţiale pe vector, respectiv matrice, puterile unei permutări, ciclurile unei permutări, ordinul unei permutări.

Ideea principală pentru rezolvarea acestei probleme, este să observăm că liniile matricei s sunt de fapt puterile permutării a.

Într-adevăr, avem $s[1][i] = a[i], s[2][i] = s[1][a[i]] = a[a[i]] = a^2[i]$, pentru $i = \overline{1, N}$.

Inductiv, presupunând că $s[k][i] = a^k[i]$, deducem că $s[k+1][i] = s[k][a[i]] = a^k[a[i]] = a^{k+1}[i]$, pentru $i = \overline{1, N}$.

Cum liniile matricei s sunt puterile permutării a, puterile unei permutări fiind în număr infinit însă numărul permutărilor de ordin N este finit ($N!$), rezultă că deducem că va exista o putere a^{k+1} care coincide cu permutarea identică a^1, urmând ca mai apoi liniile matricei să se repete (să cicleze) cu o perioadă k. În cazul în care k e minim cu această proprietate, el se numeşte ordinul lui a.

O altă observaţie este faptul că orice permutare se poate descompune într-un produs de cicluri disjuncte. Un ciclu de lungime k este o secvenţă $i_1, i_2, \ldots, i_k$ astfel încât $a[i_1] = i_2, a[i_2] = i_3, \ldots, a[i_{k-1}] = i_k, a[i_k] = i_1$.

Această descompunere în cicluri se poate face astfel: pentru un indice i care nu a fost procesat aflăm $a[i], a[a[i]], a[a[a[i]]], \ldots$ până când obţinem valoarea i. Valorile astfel obţinute formează un ciclu, numărul valorilor reprezentând lungimea ciclului. Se poate vedea uşor, prin calculul puterilor sale, că ordinul unui ciclu este egal cu lungimea ciclului. De asemenea, ordinul unei permutări este egal cu cel mai mic multiplu comun al lungimilor ciclurilor disjuncte din descompunerea permutării.

De exemplu, pentru permutarea $a = [3, 4, 5, 2, 1]$, dscompunerea în cicluri o obţinem astfel: pentru $i = 1$ avem $a[i] = 3, a[a[i]] = 5, a[a[a[i]]] = 1$ şi astfel obţinem primul ciclu: $(1, 3, 5)$. Primul indice neprocesat este 2, pentru care avem: $i = 2, a[i] = 4, a[a[i]] = 2$ şi obţinem ciclul: $(2, 4)$.Deoarece lungimile celor două cicluri sunt 3, respectiv 2, deducem că ordinul permutării este 6.

Subtaskul 1 (10 puncte)

Pentru a putea calcula în $O(1)$ cele Q statistici, se precalculează sumele parţiale pentru vectorul a.

Se formează un nou vector b, astfel încât $b[i] = \sum_{j=1}^{i} a[j]$, calculul unei statistici pe intervalul de indici $[c_l, c_r]$ presupunând calculul $b[c_r] - b[c_{l-1}]$.

Complexitate temporală: $O(N + Q)$.

Notă. Pentru a rezolva acest subtask, putem să calculăm statistica cerută iterativ, prin parcurgerea secvenţei $[c_l, c_r]$, deoarece $c_r - c_l \leq 100$.

Subtaskul 2 (20 de puncte)

Se generează matricea s a sumelor investite în cele M zile, apoi pentru a afla o statistică se calculează suma elementelor submatricei determinată de setul de valori z_i, z_f, c_l,c_r. Aici se aplică metoda „Brutus".

Complexitatea temporală: $O(MN + 100QM)$.

Subtaskul 3 (12 puncte)

Se generează matricea s a sumelor investite în cele M zile şi se calculează sumele parţiale 2D ale acestei matrice. Fiecare statistică se va calcula în $O(1)$.

Dacă notăm cu *sp* matricea sumelor partiale, atunci ea se poate calcula folosind recurenţa

$$sp[j][i] = sp[j][i-1] + col[i],$$
$$col[i] = col[i] + s[j][i],$$

pentru $i = \overline{1,N}, j = \overline{1,M}$, unde $sp[j][i] = \sum_{l=1}^{j} \sum_{c=1}^{i} s[l][c]$.

Calculul unei statistici cu parametrii z_i,z_f, c_l, c_r se face cu formula

$$sp[z_f][c_r] - sp[z_f][c_l - 1] - sp[z_i - 1][c_r] + sp[z_i - 1][c_l - 1].$$

Complexitate $O(MN + Q)$.

Notă. Pentru a rezolva acest subtask, se pot calcula sumele parţiale doar pe coloane, iar pentru a calcula o statistică se află suma pentru fiecare coloană în parte.

Subtaskul 4 (24 de puncte)

Deoarece N are valoare mică, iar liniile matricei s reprezintă puterile permutării a, sirul a se va repeta în matricea s după un număr relativ mic de zile. Intuitiv, dacă $N = 50 = 2 + 3 + 5 + 7 + 11 + 13 + 9$, avem $\text{cmmmc}(2, 3, 5, 7, 11, 13, 9) = 90090$, valoarea maximă a ordinului unei permutări de lungime 50 fiind 180180.

Ordinul maxim al unei permutări de lungime N este notat $g(N)$ şi se numeşte funcţia lui Landau (A000793, OEIS). Pentru $N = 50$ avem $g(50) = 180180$.

Pentru matricea generată până la repetarea şirului a se vor calcula sumele parţiale 2D. Cum M este mult mai mare, se va folosi periodicitatea acestei matrice, adică se va afla de câte ori se repetă în intervalul $[z_i, z_f]$ matricea generată, la care se adaugă un rest ce nu completează o matrice întreagă.

Complexitate $O(N \operatorname{ord}(a) + Q)$, unde $\operatorname{ord}(a)$ este ordinul permutării a.

Notă. Pentru a rezolva acest subtask, nu este necesar să știm care este ordinul maxim posibil, ci doar să observăm că liniile matricei ciclează.

Subtaskul 5 (34 de puncte)

Pe lângă ideea principală enunţată anterior, mai trebuie observat că pe fiecare coloană a matricei s se repetă periodic aceleaşi elemente, mai exact elementele permutării a care formează un ciclu. Într-adevăr, elementele matricei s situate pe coloana i sunt, în ordine, $a[i], a[a[i]], a[a[a[i]]], \ldots, i$, care apoi se repetă.

Astfel, vom descompune mai întâi permutarea a în cicluri , pentru fiecare element al ciclului reţinând poziţia sa în ciclu. De asemenea, pentru fiecare ciclu vom face sumele parţiale ale elementelor sale.

Pentru a calcula o statistică, vom afla pentru fiecare coloană cuprinsă între c_l şi c_r suma elementelor cuprinse între zilele z_i şi z_f. Cum aceste elemente sunt elementele unui ciclu care

se repetă de mai multe ori, vom afla de câte ori se repetă şi vom multiplica acest număr cu suma elementelor ciclului, rămânând şi un rest care se va calcula prin aflarea poziţie în ciclu a elementelor rămase. Se vor însuma rezultatele obţinute pentru fiecare coloană în parte, dintre coloanele cu indicii de ordine de la c_l la c_r.

Complexitate $O(N + 100Q)$.

4.7.C Problema Superhedgy

Propusă de: stud. Mihaela Cismaru — NetRom Software, Universitatea din Craiova

Subtaskul 1 (20 de puncte)

Pentru 20 de puncte este suficientă calcularea tuturor traseelor posibile și a efortului depus pentru acestea prin metoda backtracking. Se afișează minimul dintre acestea.

Subtaskul 2 (20 de puncte)

Pentru alte 20 de puncte este necesară o implementare de complexitate $O(L_{Total})$. Dat fiind că efortul de a folosi liftul va fi mereu 0 putem alege fără ezitare să schimbăm partea orașului în care ne aflăm pentru ca următoarea miscare să aibă valore minimă. Pentru aceast subtask functionează o abordare de tip greedy.

Subtaskul 3 (40 de puncte)

Pentru alte 40 de puncte este necesară o implementare tot de complexitate $O(L_{Total})$ dar de această dată costul lifturilor poate fi nenul, astfel folosirea lifturilor nu este mereu optima. Soluția va fi implementarea unei dinamici. Calculm, pentru $1 \leq i \leq N$ și $1 \leq j \leq M$:

$$D_{sus}[i] = \text{efortul minim de a ajunge pe pozitia } i \text{ in partea de sus a orasului,}$$
$$D_{jos}[i] = \text{efortul minim de a ajunge pe pozitia } i \text{ in partea de jos a orasului.}$$

Pentru a calcula aceste valori vom avea, pentru $1 \leq i \leq N$ și $1 \leq j \leq M$, formulele:

$$D_{sus}[i] = \min(D_{sus}[i-1] + 1, D_{jos}[i] + E_i + E'_i),$$
$$D_{jos}[i] = \min(D_{jos}[i-1] + 1, D_{sus}[i] + E'_i + E_i).$$

Subtaskul 4 (20 de puncte)

Pentru alte 20 de puncte este necesar calcularea efortului in complexitate de doar $O(N + M)$. Putem realiza acest lucru prin observatia ca putem pastra in memorie doar portiunile relevante din oras, anume portiunile unde se termina si incepe o noua cladire.

Capitolul 5 Olimpiada Națională de Informatică

5.1 Clasa a V-a

5.1.A Problema Culori

Propusă de: Prof. Iordaiche Eugenia-Cristiana — Liceul Teoretic „Grigore Moisil" Timișoara

Cerința 1

O soluție posibilă pentru determinarea lungimii maxime a unui rând ce are proprietatea că oricare două pătrățele alăturate au culori diferite, constă în parcurgerea liniară a tuturor pătrățelelor fiecărui rând și compararea succesivă a culorilor pentru oricare două pătrățele alăturate. Actualizăm la fiecare pas *Lmax* cu lungimea rândului corect identificat, conform cerinței.

KMax este o variabilă de tip contor în care vom număra toate rândurile de pătrățele ale caietului, ce au lungimea egală cu Lmax.

```
if(nr_culori > Lmax) {
    Lmax = nr_culori;
    Kmax = 1;
} else if (Lmax == nr_culori)
  Kmax++;
```

Cerința 2

Pentru a determina cel mai mare număr format prin lipirea tuturor cifrelor unui rând de pătrățele, putem utiliza un algoritm de compararea lexicografică a două șiruri de cifre, parcurgându-le element cu element de la dreapta la stânga. Identificăm astfel, cel mai mare număr natural ce se poate construi cu cifrele unui rând de pătrățele. Acest număr va fi memorat cifră cu cifră într-un tablou unidimensional.

Parcurgem fiecare rând de pătrățele de la dreapta la stânga, cifră cu cifră și comparăm din punct de vedere lexicografic două șiruri de cifre. În momentul în care găsim un șir de cifre mai mare din punct de vedere lexicografic, actualizăm datele memorate în tabloul ce va memora rezultatul final și continuăm până în momentul în care vom compara toate șirurile de cifre de pe rândurile caietului.

5.1.B Problema Joc

Propusă de: Prof. Dabelea Delia — Colegiul Național „Spiru Haret" Târgu Jiu

Cerința 1

O soluție posibilă ar fi descompunerea în factori primi ai numărului natural N, numărul divizorilor acestuia fiind egal cu produsul puterilor factorilori primi crescuți cu o unitate.

Cerința 2

Ne putem folosi de un vector de frecvență cu doar 7 elemente, indexate de la 0 la 6, în care, contorizăm fiecare cifră X calculată. După construirea acestuia, valoarea maximă din el reprezintă numărul maxim de apariții căutat. Un rezultat corect depinde, evident, de simularea corectă a jocului.

Cerința 3

Pentru gestionarea corectă a mutărilor alternative a pionilor pe tablă, vom folosi două variabile (să le notăm a pentru primul copil și b pentru al doilea copil), variabile care nu vor avea niciodată valoare identică. Spre exemplu $a = 1$ și $b = 0$ sau $a = 0$ și $b = 1$ (1 – mută, 0 – nu mută) în funcție de copilul care deplasează pionul.

Numerele căsuțelor vizitate de pioni în timpul jocului se păstrează în doi vectori (un vector pentru pionul primului copil și altul pentru pionul celui de-al doilea copil).

Simulăm jocul printr-o structură repetitivă care se încheie în momentul în care un pion ajunge pe ultima căsuță. La fiecare pas, înaintăm pionul copilului care este la mutare X căsuțe, păstrăm numărul noii căsuțe în vectorul corespunzător și respectăm celelalte reguli din enunț.

5.1.C Problema Rotire25

Propusă de: stud. Banu Denis Andrei — Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza", Iași

Cerința 1

Subtask 1 (7 puncte) Când numărul obținut în urma înmulțirilor este $\leq 10^{18}$. Se poate calcula $\underbrace{X \cdot X \cdot X \cdot \ldots \cdot X}_{\text{de } K \text{ ori}}$ folosind structura repetitivă for, iar rezultatul poate să fie reținut într-o variabilă de tipul `long long`. Se ia ultima cifră a acestui rezultat și se înmulțește cu prima cifră a lui X.

Subtask 2 (11 puncte) Când $K \leq 100\,000$. Trebuie să observăm că atunci când înmulțim două numere, ultima cifră a rezultatului este dată de înmulțirea ultimei cifre din primul număr cu ultima cifră din al doilea număr. Este suficient să reținem ultima cifră a rezultatului după fiecare înmulțire. Astfel rezultatul va rămâne mic și va putea fi stocat într-o variabilă de tip `int`. Putem folosi din nou structura repetitivă for, iar la final înmulțim rezultutatul cu prima cifră a lui X.

Toată cerința 1 (29 de puncte) Când $K \leq 10^9$. De data aceasta nu mai putem sa iterăm de K ori deoarece ar dura prea mult. Trebuie sa observăm că înmulțirea unei cifre cu ea însăși de K ori are o anumită periodicitate astfel:

- 0, 1, 5 și 6 - au perioada 1. De oricâte ori am înmulți una dintre aceste cifre cu ea, rezultatul va avea ultima cifră egală cu aceasta.
- 4 și 9 - au perioada 2.
- 2, 3, 7 și 8 - au perioada 4.

Ca urmare, dacă X are ultima cifră 0, 1, 5, 6 rezultatul va fi tot acea cifră indiferent de valoarea lui K. Dacă ultima cifră este 4 sau 9, dacă K este impar rezultatul este acea cifră adică 4 respectiv 9, dacă este par rezultatul este ultima cifră înmulțită o dată cu ea, adică $4 \cdot 4 = 16 \rightarrow 6$ respectiv $9 \cdot 9 = 81 \rightarrow 1$.

Cerința 2

Subtask 1 (39 de puncte) Când $K \leq 100\,000$ putem simula transformările și la final afișăm rezultatul.

Toată cerința 2 (71 de puncte) Când $K \leq 100.000$ trebuie să observăm că după ce am făcut primele 2, 3 transformări, șirul rezultat în urma aplicării transformărilor este periodic.

De exemplu, pentru $X = 13$:

Numărul de transformări aplicate	Numărul rezultat
0	13
1	65
2	211
3	551
4	211
5	511
...	...
K par	211
K impar	511

Întotdeauna perioada va fi 2. Putem să facem simularea intr-un mod asemănător cu subtaskul 1 și să reținem numărul de la transformarea curenta și numărul din urmă cu două transformări. În momentul în care numărul curent ajunge egal cu numărul din urmă cu două transformări, atunci am găsit numerele din perioada. Dacă K are aceeași paritate cu pasul curent, afișăm numărul curent, dacă nu, mai facem o transformare și afișăm numărul obținut în urma acestei transformări.

Observatia 3. *Pentru a scăpa de cazurile particulare, atunci când K este mic, se poate face simulare exact ca la subtaskul 1.*

Demonstrație pentru periodicitate

Pentru simplitate vom nota cu $\xrightarrow{\cdot 5}$ pasul din transformare unde înmulțim numărul cu 5, cu $\xrightarrow{\cdot 2}$ pasul din transformare unde înmulțim numărul cu 2, cu $\xrightarrow{e}$ eliminarea zerourilor din număr, și cu $\xrightarrow{o}$ oglindirea numărului.

Pentru $X = \overline{a}$ Aici avem două cazuri, în funcție de paritatea lui a.

Dacă a este par:

$$a \xrightarrow{\cdot 5} a \cdot 5 = 10 \cdot \frac{a}{2} \xrightarrow{e} \frac{a}{2} \xrightarrow{o} \frac{a}{2} \xrightarrow{\cdot 2} a \xrightarrow{e} a \xrightarrow{o} a$$

Am ajuns la numărul de la care am plecat și urmează să facem aceeași transformare, deci secvența de transformări se va repeta.

Dacă a este impar:

$$\begin{aligned} a \xrightarrow{\cdot 5} a \cdot 5 = 10 \cdot \frac{a-1}{2} + 5 \xrightarrow{e} 10 \cdot \frac{a-1}{2} + 5 \xrightarrow{o} 10 \cdot 5 + \frac{a-1}{2} \\ \xrightarrow{\cdot 2} 100 + (a-1) \xrightarrow{e} 10 + (a-1) \xrightarrow{o} 10 * (a-1) + 1 \\ \xrightarrow{\cdot 5} 100 \cdot \frac{a-1}{2} + 5 \xrightarrow{e} 10 \cdot \frac{a-1}{2} + 5 \xrightarrow{o} 10 \cdot 5 + \frac{a-1}{2} \end{aligned}$$

Pentru $X = \overline{ab}$ Aici avem mai multe cazuri, în funcție de paritatea lui a și b.

Dacă a este par, b par:

$$10 \cdot a + b \xrightarrow{\cdot 5} 100 \cdot \frac{a}{2} + 10 \cdot \frac{b}{2} \xrightarrow{e} 10 \cdot \frac{a}{2} + \frac{b}{2} \xrightarrow{o} 10 \cdot \frac{b}{2} + \frac{a}{2}$$
$$\xrightarrow{\cdot 2} 10 \cdot b + a \xrightarrow{e} 10 \cdot b + a \xrightarrow{o} 10 \cdot a + b$$

Dacă a este par, b impar:

$$10 \cdot a + b \xrightarrow{\cdot 5} 100 \cdot \frac{a}{2} + 10 \cdot \frac{b-1}{2} + 5 \xrightarrow{e} 100 \cdot \frac{a}{2} + 10 \cdot \frac{b-1}{2} + 5 \xrightarrow{o} 100 \cdot 5 + 10 \cdot \frac{b-1}{2} + \frac{a}{2}$$
$$\xrightarrow{\cdot 2} 1000 + 10 \cdot (b-1) + a \xrightarrow{e} 100 + 10 \cdot (b-1) + a \xrightarrow{o} 100 \cdot a + 10 \cdot (b-1) + 1$$
$$\xrightarrow{\cdot 5} 1000 \cdot \frac{a}{2} + 100 \cdot \frac{b-1}{2} + 5 \xrightarrow{e} 100 \cdot \frac{a}{2} + 10 \cdot \frac{b-1}{2} + 5 \xrightarrow{o} 100 \cdot 5 + 10 \cdot \frac{b-1}{2} + \frac{a}{2}$$
$$\xrightarrow{\cdot 2} 1000 + 10 \cdot (b-1) + a \xrightarrow{e} 100 + 10 \cdot (b-1) + a \xrightarrow{o} 100 \cdot a + 10 \cdot (b-1) + 1$$

Daca a este impar, b par:

$$10 \cdot a + b \xrightarrow{\cdot 5} 100 \cdot \frac{a-1}{2} + 10 \cdot \left(\frac{b}{2} + 5\right) \xrightarrow{e} 10 \cdot \frac{a-1}{2} + \frac{b}{2} + 5 \xrightarrow{o} 10 \cdot \left(\frac{b}{2} + 5\right) + \frac{a-1}{2}$$
$$\xrightarrow{\cdot 2} 100 + 10 \cdot b + (a-1) \xrightarrow{e} 100 + 10 \cdot b + (a-1) \xrightarrow{o} 100 \cdot (a-1) + 10 \cdot b + 1$$
$$\xrightarrow{\cdot 5} 1000 \cdot \frac{a-1}{2} + 100 \cdot \frac{b}{2} + 5 \xrightarrow{e} 100 \cdot \frac{a-1}{2} + 10 \cdot \frac{b}{2} + 5 \xrightarrow{o} 100 \cdot 5 + 10 \cdot \frac{b}{2} + \frac{a-1}{2}$$
$$\xrightarrow{\cdot 2} 1000 + 10 \cdot b + (a-1) \xrightarrow{e} 100 + 10 \cdot b + (a-1) \xrightarrow{o} 100 \cdot (a-1) + 10 \cdot b + 1$$

(Cum b este o cifra, $\frac{b-1}{2} + 5 \leq 9$.)

Dacă a este impar, b impar:

$$10 \cdot a + b \xrightarrow{\cdot 5} 100 \cdot \frac{a-1}{2} + 10 \cdot \left(\frac{b-1}{2} + 5\right) + 5 \xrightarrow{e} 100 \cdot \frac{a-1}{2} + 10 \cdot \left(\frac{b-1}{2} + 5\right) + 5$$
$$\xrightarrow{o} 100 \cdot 5 + 10 \cdot \left(\frac{b-1}{2} + 5\right) + \frac{a-1}{2}$$
$$\xrightarrow{\cdot 2} 1000 + 100 + 10 \cdot (b-1) + (a-1) \xrightarrow{e} 1000 + 100 + 10 \cdot (b-1) + (a-1)$$
$$\xrightarrow{o} 1000 \cdot (a-1) + 100 \cdot (b-1) + 10 + 1$$
$$\xrightarrow{\cdot 5} 10000 \cdot \frac{a-1}{2} + 1000 \cdot \frac{b-1}{2} + 10 \cdot 5 + 5 \xrightarrow{e} 1000 \cdot \frac{a-1}{2} + 100 \cdot \frac{b-1}{2} + 10 \cdot 5 +$$
$$\xrightarrow{o} 1000 \cdot 5 + 100 \cdot 5 + 10 \cdot \frac{b-1}{2} + \frac{a-1}{2}$$
$$\xrightarrow{\cdot 2} 10000 + 1000 + 10 \cdot (b-1) + (a-1) \xrightarrow{e} 1000 + 100 + 10 \cdot (b-1) + (a-1)$$
$$\xrightarrow{o} 1000 \cdot (a-1) + 100 \cdot (b-1) + 10 + 1$$

Într-un mod asemănător se poate demonstra și pentru X de forma $\overline{abc}$.

5.2 Clasa a VI-a

5.2.A Problema Iluminat

Propusă de: prof. Prof. Violeta Grecea — Colegiul Național de Informatică „Matei Basarab" Râmnicu Vâlcea

Cerințele 1 și 2

Rezolvarea acestor două cerințe presupune simularea stingerii becurilor: pe linia și coloana pe care se află numărul maxim se sting toate becurile.

Soluție brută Constă în calcularea efectivă a maximului prin parcurgerea tuturor elementelor din matrice și apoi, parcurgerea liniei și a coloanei pe care se află acesta și atribuirea valorii 0 pentru toate elementelor întâlnite pe această linie și această coloană. Procedeul se repetă pentru fiecare etapă, până la etapa k de stingere a becurilor.

Apoi:

- Pentru cerința 1 se afișează maximul găsit la etapa k.
- Pentru cerința 2 se calculează suma elementelor de pe linia și coloana maximului, prin parcurgerea liniară a acestora. Maximul nu trebuie adăugat de 2 ori (o dată la parcugerea liniei și a doua oară la parcurgerea coloanei).

Această soluție nu va lua punctaj maxim, deoarece nu se încadrează în timp (complexitate temporală $O(kn^2)$, punctajul maxim posibil fiind de 36 de puncte pentru ambele cerințe.

Soluție optimă Presupune folosirea unor vectori astfel:

- doi vectori, *xLinie* și *xColoană*, fiecare având n^2 elemente, în care $xLinie_i$ = linia în care se află valoarea i în tablou, iar $xColoană_i$ = coloana în care se află valoarea i în tablou. Odată cu citirea datelor de intrare se completează și valorile corespunzătoare acestor doi vectori.
- doi vectori linie și coloana, cu maximum n elemente fiecare, în care:
  - $linie_i = 0$ dacă becurile de pe linia i nu au fost stinse (maximul nu s-a găsit încă pe linia i)
  - $linie_i = 1$ dacă becurile de pe linia i au fost stinse (maximul s-a găsit pe linia i)

  Același rol îl are vectorul coloană, referitor la coloanele pe care se găsește maximul.

Simularea stingerii becurilor presupune parcurgerea simultană, în ordinea descrescătoare a indicilor, a vectorilor *xLinie* și *xColoană*, până când se găsește o valoare i pentru care $linie_i = 0$ și $coloană[i] = 0$. Acesta este, pe rând maximul căutat la o etapă. Procedeul se repetă pentru fiecare etapă, până la etapa k de stingere a becurilor. Apoi:

- Pentru cerința 1 se afișează maximul maximul găsit la etapa k.
- Pentru cerința 2 se calculează suma elementelor de pe linia și coloana maximului, prin verificarea condiției ca linia și coloana corespunzătoare elementului să fie egale cu 0 (becurile nu sunt încă stinse).

Această abordare va lua punctaj maxim la aceste două cerințe. complexitatea temporală este $O(n^2 + nk)$.

Cerința 3

Presupune calcule care să pornească de la tabloul inițial.

Soluție brută Constă în parcurgerea fiecărei submatrice cu k linii și k coloane și calcularea sumei elementelor sale.

Soluție optimă Constă în precalcularea unor sume parțiale, utilizând formula pentru sume partiale 2D:

$$cartier_{i,j} = cartier_{i,j-1} + cartier_{i-1,j} - cartier_{i-1,j-1} + cartier_{i,j}$$

Pe baza acestor sume parțiale se calculează maximul dintr-o submatrice de dimensiune kxk a tabloului. Această abordare a cerinței, de complexitate $O(n^2)$, va lua punctaj maxim.

5.2.B Problema Inundatie

Propusă de: prof. Cristian Frâncu — Nerdvana

Obervații

Următoarele observații sunt importante pentru toate cerințele:

- Apa va acoperi coloanele în ordinea de la stânga la dreapta. Este ordinea în care vârfurile coloanelor sunt atinse de apă;
- Odată ce apa atinge vârful unei coloane, deci acoperă acea coloană, nu este obligatoriu ca apa să se acumuleze deasupra acelei coloane. Dacă are unde se duce spre dreapta, înălțimile coloanelor fiind mai mici sau egale cu coloana curentă, apa se va deplasa.

Cerința 1

Se cere înălțimea maximă a unei coloane de apă când apa ajunge peste tot la nivelul coloanei maxime.

La acest moment nivelul apei este uniform și are exact înălțimea celei mai înalte coloane. Deci cea mai înaltă coloană de apă va fi deasupra înălțimii minime. Răspunsul este $hmin - h_{min}$, diferența dintre înălțimea maximă și cea minimă de la intrare.

Complexitate timp: $O(N)$; complexitate memorie: $O(N)$

Cerința 2

Se cere timpul în care apa ajunge la înălțimea de pe poziția P.

Presupunem că apa tocmai a ajuns la coloana P. În acel moment există apă acumulată în diverse coloane spre stânga. Dacă însumăm numărul de pătrătele al fiecărei coloane de apă vom obține timpul necesar. Așadar pornim de la coloana P, de înălțime H, deplasându-ne către stânga. Deoarece apa a ajuns la coloana P înseamnă că spre stânga apa are cel puțin acea înălțime. Avem, deci, două posibilități:

1. Coloana din stânga, h_i, este mai mică sau egală cu cea curentă. În acest caz nivelul apei este la înălțime H, iar coloana de apă are înălțime $H - h_i$. Vom însuma această înălțime la totalul de timp necesar.
2. Coloana din stânga, h_i, este mai mare decât cea curentă. În acest caz vom actualiza H la acea înălțime h_i.

Continuăm, astfel, deplasarea către stânga însumând coloanele de apă și actualizând înălțimea H. Suma acestor coloane este timpul minim în care apa ajunge la înălțimea P.

Complexitate timp: $O(N)$; complexitate memorie: $O(N)$

Cerința 3

Se cere numărul înălțimii la care ajunge apa după S secunde.

Soluție 1: căutare binară O soluție este să folosim rezolvarea cerinței 1: vom căuta binar coloana la care ajunge apa. Vom porni cu un interval de căutare $[1, N]$ și la fiecare pas ne vom întreba, pentru coloana de la jumatea acelui interval, în cât timp ajunge apa la acea coloană, folosind algoritmul de la cerința 2. Dacă timpul depășește S vom deplasa limita din dreapta a intervalului, în caz contrar pe cea din stânga.

Complexitate timp: $O(N \log_2 N)$, complexitate memorie: $O(N)$

Soluție 2: parcurgere de la stânga la dreapta Pornim de la coloana 1, de înălțime H. Avem două variante:

1. Coloana din dreapta, h_i, este mai mică sau egală cu cea curentă, H. În acest caz nu consumăm apă pentru a ne deplasa spre dreapta, deci nu avem nimic de făcut.
2. Coloana din dreapta, h_i, este mai mare decât cea curentă, H. În acest caz apa se acumulează. Ne *înălțăm* cu o unitate față de H și ne deplasăm către stânga până găsim prima coloană la această înălțime (sau până depășim prima coloană). La fiecare deplasare adunăm unu la necesarul de apă. Dacă nu am depășit S, ne înălțăm încă o unitate față de H și repetăm procedeul de deplasare la stânga. Ne vom înălța până ce ajungem la înălțimea h_i, sau până ce depășim *bugetul* de S unități de apă. Dacă ajungem la înălțimea h_i, vom relua deplasarea către stânga.

Ultima coloană la care am reușit să ajungem este răspunsul la cerință.

Complexitate timp: $O(N)$; complexitate memorie: $O(N)$

Cerința 4

Se cere numărul celei mai din stânga înălțimi pe care o vom reduce cu o unitate pentru ca apa să ajungă cât mai repede la coloana cu numărul P.

Mergând de la P către stânga vom număra distanțele dintre înălțimi despărțite de apă. Distanța maximă, D, este cea unde vom reduce coloana din dreapta, deoarece reducând acea coloană cu unu timpul se scurtează cu D. Procedăm astfel:

- Pornim de la coloana numărul P de înălțime H, parcurgând înălțimile către stânga. Ne oprim la prima înălțime h_i mai mare sau egală cu H și reținem distanța dintre cele două înălțimi, $D = P - i + 1$, precum și poziția înălțimii din dreapta acelei distanțe, P. Distanța D este chiar diferența de timp cu care se scurtează timpul în care apa ajunge la coloana P, dacă reducem coloana P cu unu.
- Actualizăm înălțimea curentă H ca fiind h_i și P ca fiind i și continuăm deplasarea către următoarea înălțime mai mare sau egală cu H, obținând o nouă distanță D. La fiecare pas reținem D dacă este mai mare sau egal cu cel de până acum, împreună cu numărul coloanei din dreapta distanței D, adică P. Este necesar mai mare sau egal deoarece dorim cea mai din stânga coloană pe care o putem reduce.

Trebuie să dăm, însă, atenție specială cazului când avem mai multe coloane de aceeași înălțime despărțite de înălțimi mai mici, de exemplu cazul 6, 3, 3, 3, 3, 3, 6, 3, 3, 6. Reducând cel mai din dreapta 6 la 5 vom reduce timpul cu 2 secunde, distanța până la următorul 6. Dar dacă reducem următorul 6 la 5 nu vom câștiga nimic, de fapt vom pierde o secundă.

Pentru a trata acest caz, după fiecare deplasare și calcul de distanță D, dacă h_i are aceeași înălțime cu H ne vom deplasa la stânga suplimentar până ce găsim o coloană strict mai înaltă decât H.

Complexitate timp: $O(N)$; complexitate memorie: $O(N)$

5.2.C Problema Șiruri

Propusă de: prof. Prof. Flavius Boian — Colegiul Național „Spiru Haret" Târgu-Jiu

Cerința 1

O soluție eficientă este marcarea într-un vector de frecvență a cifrelor existente pentru fiecare număr și oprirea în cazul în care o cifră este deja marcată.

Cerința 2

Se verifică dacă ultima cifră a numărului curent este egală cu prima cifră a numărului următor. Dacă acest lucru este adevărat se lipesc cele două numere, se elimină cifrele care apar de mai multe ori, păstrându-se doar prima apariție și se repetă procedeul atât timp cât este posibil.

Verificarea dacă numărul nou format se va uni cu următorul se face după ce în număr cifrele rămân o singură dată. Dacă un număr nu are ultima cifră egală cu a următorului, acesta se transformă prin ștergerea cifrelor care apar de mai multe ori și păstrarea celei mai din stânga apariții a acestora.

Cerința 3

Problema se reduce la a determina, în șirul nou format, câte numere cu număr maxim de cifre există. Se parcurge șirul, se reține numărul cu număr maxim de cifre și numărul de apariții al unui astfel de număr. Rezolvarea cerinței se poate face printr-o singură parcurgere a șirului. Complexitate timp: $O(NL_{max})$, complexitate memorie: $O(N)$, unde L_{max} reprezintă numărul maxim de cifre ale unui număr.

5.3 Clasa a VII-a

5.3.A Problema Microbuz

Propusă de: Prof. Marinel Șerban

Soluție prof. Marinel Șerban

Observații Problema testează cunoștințe despre:

1. lucrul cu vectori
2. utilizarea unui algoritm de tip succesor
3. lucru cu baze de numerație
4. determinarea minimului și maximului unei mulțimi de valori

Cerințele 1 și 2 Se vor genera toate combinațiile posibile de bilete în mod succesiv.

Pentru început, vom reține în tabloul $max\_cnt_i = \min(3, \frac{n}{i})$, pentru i de la 1 la 10, numărul maxim de bilete de tip i pe care le putem cumpăra fără a depăși n kilometri.

În continuare, vom genera fiecare combinație cu ajutorul tabloului cnt_i = numărul de bilete de tip i din combinație, în mod succesiv utilizând un *algoritm de tip succesor*. Pentru fiecare combinație vom calcula distanța parcursă de combinația:

$$distanță = 1 \cdot cnt_1 + 2 \cdot cnt_2 + \cdots + 10 \cdot cnt_10$$

și suma costurilor biletelor din acea combinație

$$cost = km_1 \cdot cnt_1 + km_2 \cdot cnt_2 + \cdots + km_10 \cdot cnt_10$$

Dacă distanța este egală cu n, atunci se actualizează răspunsul optim.

Pseudocodul arată astfel:

for i de la 1 la 10 **do**
  $max\_cnt_i \leftarrow \min(3, n/i)$
end for
$cost\_minim \leftarrow \infty$; $ok \leftarrow 1$; $cost \leftarrow 0$; $distanță \leftarrow 0$
while $ok = 1$ **do**
  $i \leftarrow 10$; $cnt_i \leftarrow cnt_i + 1$; $distanță \leftarrow distanță + i$; $cost \leftarrow cost + km_i$
  while $i > 0$ și $cnt_i > max\_cnt_i$ **do**
    $sumă \leftarrow sumă - cnt_i \cdot km_i$; $cost \leftarrow cost - cnt_i \cdot i$; $cnt_i \leftarrow 0$; $i \leftarrow i - 1$
  end while
  if $i = 0$ **then**
    $ok = 0$
    sfârșit
  end if
  if $distanță$ **then**

```
15:         if cost_minim > cost then
16:             cost_minim ← cost
17:             for i de la 1 la 10 do
18:                 răsp_i ← cnt_i
19:             end for
20:         end if
21:     end if
22: end while
```

Existența soluției pentru cerința 3 În condițiile problemei:
Suma minimă a unei submulțimi este 10 — mulțimea cu un bilet de cost 10.
Suma maximă a unei submulțimi este 855 — mulțimea de bilete $\{91, 92, 93, 94, 95, 96, 97, 98, 99\}$.
Deci există $855 - 10 + 1 = 846$ sume posibile.
În același timp există $C_{10}^1 + C_{10}^2 + \cdots + C_{10}^9 = 1022$ submulțimi.
Conform principiului cutiei lui Dirichlet există cel puțin două submulțimi de sumă egală.

Cerința 3 Se vor genera toate combinațiile posibile de submulțimi de două seturi bilete disjuncte în mod consecutiv.

Se va proceda similar ca la primele două cerințe, dar acum vom encoda starea unui bilet cu una din cele 3 valori:

0. biletul nu face parte din nici un set
1. biletul face parte din setul 1
2. biletul face parte din setul 2

Pentru fiecare combinație se determină cele 2 seturi de bilete. Se reține maximul și combinația care produce acest maxim.

Soluție prof. Claudiu-Cristian Gorea-Zamfir

Cerințele 1 și 2 Soluția urmează pașii următori

- Se generează toate numerele de 10 cifre, în baza 4 și se rențin pentru un număr cifrele sale într-un tablou *cif*. Acest tablou va reține numărul de bilete cumpărate pentru fiecare tip de bilet i de la 1 la 10;
- Se calculează prețul astfel:

$$preț = cif_1 \cdot p_1 + cif_2 \cdot p_2 + \cdots + cif_{10} \cdot p_{10};$$

- Se calculează distanța astfel:

$$dist = cif_1 \cdot 1 + cif_2 \cdot 2 + \cdots + cif_{10} \cdot 10;$$

- Dacă $dist = n$ și $preț < preț\_min$ atunci reținem configurația actuală a tabloului *cif* și actualizăm $preț\_min = preț$.

Cerința 3 Soluția urmează pașii următori:

- Fiecărui bilet îi asociem 3 stări:
  0. biletul nu este ales
  1. biletul este ales în mulțimea 1
  2. biletul este ales în mulțimea 2
- Se generează toate numerele de 10 cifre în baza 3 în mod similar cu cerințele anterioare;
- În funcție de mulțimea asociată, vom calcula sumele sum_1 și sum_2, iar dacă acestea sunt egale și maxime, vom reține suma maximă și configurația.

Soluție prof. Daniela-Elena Lica

Cerințele 1 și 2 Problema acceptă și o abordare de tip *programare dinamică*, asemănătoare cu cea folosită pentru rezolvarea Problemei Rucsacului.

Astfel, considerăm tabloul unidimensional:

$$cost_i = \text{costul minim necesar pentru a călători i kilometri.}$$

Fie:

$$bilet_i = \text{prețul unui bilet ce asigură parcurgerea unei distanțe de i km.}$$

Construirea tabloului *cost* Să presupunem că la pasul curent considerăm achiziționarea unui bilet ce asigură parcurgerea unei distanțe de d km, cu prețul egal cu $bilet_d$;

Obținem astfel următoarea relație de recurență:

$$cost_{v+d} = \min(cost_{v+d}, cost_v + bilet_d)$$

Pentru orice valoare v, parcursă descrescător (pentru a evita suprascrieri incorecte) de la $(165 - d)$ până la 0.

De remarcat este faptul că fiecare tip de bilet (dintre cele 10) poate fi cumpărat de cel mult trei ori, astfel că se va efectua adăugarea distanței de d km în *rucsac* de exact trei ori.

Pentru prima cerință este suficient să calculăm doar tabloul unidimensional *cost*, cu o complexitate de timp egală cu: $O(n \cdot S_{max})$, unde $n = 10$ și $S_{max} = 165$ pentru problema noastră.

Pentru a rezolva și cea de-a doua cerință, pe lângă acest tablou, putem reține, pentru fiecare valoare v de la 0 la 165, și modul în care $cost_v$ a fost obținut. Astfel, în cadrul egalității, dacă:

$$cost_v + bilet_d < cost_{v+d}$$

în adiție cu actualizarea valorii lui $cost_{v+d}$, putem considera că reconstrucția necesară pentru $v + d$ km este identică cu cea necesară pentru v km, la care mai trebuie să adăugăm achiziționarea unui bilet de d km. Aceste reconstrucții pot fi efectuate, de exemplu, cu ajutorul unor `struct`-uri, în $O(nS_{max})$ memorie – pentru fiecare posibilitate de distanță, pot exista cel mult 30 de tichete utilizate în atingerea acesteia.

Cerința 3 Folosind scrierea în baza 2 a numerelor naturale de la 0 la $2^e - 1$, putem genera toate submulțimile unei mulțimi ce are e elemente. Astfel, putem genera toate submulțimile mulțimii cu 10 elemente din problema de față:

$$\{bilet_1, bilet_2, bilet_3, \ldots, bilet_9, bilet_{10}\}$$

De exemplu, să considerăm, în contextul acestei scrieri, numărul binar 0000010111:

- bitul cel mai semnificativ se află în partea stângă;
- Aici, cel mai semnificativ bit are valoarea 0, este *nesetat*;
- Au fost selectate numerele:

$$bilet_1, bilet_2, bilet_3 \text{ și } bilet_5$$

- Fie $S = bilet_1 + bilet_2 + bilet_3 + bilet_5$;
- La acest pas, ne-ar mai interesa să vedem dacă există o modalitate de a alege niște numere din mulțimea rămasă, adică:

$$\{bilet_4, bilet_6, bilet_7, bilet_8, bilet_9, bilet_{10}\}$$

astfel încât însumate să dea S. Pentru aceasta, folosim un algoritm identic cu cel folosit în cadrul cerinței 2.

Soluție stud. Bogdan-Ioan Popa

În cele ce urmează vom reprezenta o submulțime de bilete ca fiind un număr în baza 2 scris cu K biți. Dacă în configurația (submulțimea) *conf* bitul i este 1 înseamnă că biletul cu $i + 1$ km este luat în submulțimea *conf*.

Cerințele 1 și 2 Se calculează vectorul de perechi $D_i = (cost, conf)$, unde *cost* reprezintă costul minim să obținem un drum de distanță exact i, *conf* reprezintă submulțimea de bilete cu care se obține acest cost minim. Ce rămâne să găsim acum este un triplet $(dist_1, dist_2, dist_3)$ pentru care $dist_1 + dist_2 + dist_3 = N$ și $D_{dist_1.cost} + D_{dist_2.cost} + D_{dist_3.cost}$ este minim.

Pentru reconstrucția soluției ne vom folosi de biletele cumpărate în configurațiile $D_{dist_1.conf}$, $D_{dist_2.conf}$ respectiv $D_{dist_3.conf}$.

Complexitatea aceste soluții este $O(2K)$, unde K este numărul de tipuri de bilete posibile pe care le putem cumpăra. Pentru această problemă $K = 10$.

Cerința 3 Se calculează tabloul bidimensional:

$$A_{conf,sum\_cost} = \begin{cases} 1, & \text{dacă există o submulțime } x \text{ a submulțimii } conf \text{ astfel încât suma costurilor} \\ & \text{biletelor din submulțimea } x \text{ să fie } sum\_cost \\ 0, & \text{altfel} \end{cases}$$

Ce rămâne de făcut este să iterăm prin toate submulțimile $conf$ posibile ale celor K bilete.

Fie *comp_conf* submulțimea complementară submulțimii *conf*.

Fie s suma costurilor biletelor din submulțimea *conf*.

Dacă $A_{comp\_conf,s} = 1$ înseamnă că am găsit două submulțimi disjuncte de bilete care însumează același cost. Nu uităm însă că $A_{comp\_conf,s} = 1$ indică faptul că există o submulțime a lui

comp_conf, s pentru care suma costurilor biletelor din ea să fie s și nu înseamnă neapărat că suma costurilor biletelor din *comp_conf* este egală cu s.

Complexitatea soluției este $O(2^K S_{max})$, unde S_{max} reprezintă suma totală a costurilor biletelor.

5.3.B Problema Raza

Propusă de: prof. Alin Burța

Soluție prof. Alin Burța

Cerința 1 Încă de la citirea datelor de intrare, se verifică dacă traiectoria roverului curent se intersectează cu diagonala principală și contorizăm numărul roverelor care îndeplinesc condiția.

Avem 3 cazuri:

- Roverul curent nu intersectează diagonala, caz care nu ne interesează;
- Roverul curent intersectează diagonala într-o singură pozitie;
- Roverul curent intersectează diagonala în două poziții.

Fie (x, y) linia respectiv coloana colțului din stânga-sus al traiectoriei roverului curent, iar r lungimea laturii traiectoriei. Condiția de intersecție a traiectoriei cu diagonala este:

$$x \leq y + r - 1 \text{ și } x + r - 1 \geq y$$

Cerința 2 Concomitent cu determinarea roverelor, care intersectează diagonala principală vom construi 3 vectori, cu următoarele semnificații:

- unu_i = secunda în care roverul i intersectează prima dată diagonala în cadrul primei parcurgeri a traiectoriei;
- doi_i = secunda în care roverul i intersectează a doua oară diagonala în cadrul primei parcurgeri a traiectoriei sau 0, dacă roverul NU intersectează diagonala în două poziții;
- $prd_i = 4r - 4$, perioada roverului i (numărul de secunde necesare parcurgerii întregii traiectorii;

Numărul de secunde S este relativ mic, deci putem calcula, pentru fiecare secundă între 1 și S, câte rovere se găsesc pe diagonala principală în acel moment. Acest calcul presupune marcarea, pentru fiecare rover, a tuturor momentelor de timp în care roverul atinge diagonala principală, folosind cei trei vectori construiți anterior. Reținem la fiecare pas numărul maxim curent și momentul de timp corespunzător.

Soluție prof. Gorea-Zamfir Claudiu-Cristian

La citirea roverelor, identificăm 5 situații bazate pe valorile de plecare, raza atinge traiectoria astfel:

- 2 colțuri ($l = c$) — momentele de timp sunt $\{0, 2r, 4r, 6r, \dots\}$;
- 1 colț, dreapta-sus ($c + r = l$) — momentele de timp sunt $\{1r, 5r, 9r, \dots\}$;
- 1 colț, dreapta-sus ($l + r = c$) — momentele de timp sunt $\{3r, 7r, 11r, \dots\}$;
- 2 laturi nord și est ($c < l$ si $l < c + r$) — momentele de timp sunt $\{1 + l - c, 1 + l - c + 4r, 1 + l - c + 8r, \dots\}$ și $\{2r + 1 - l + c, 6r + 1 - l + c, 10r + 1 - l + c, \dots\}$;
- 2 laturi sud și vest ($l < c$ si $c < l + r$) — momentele de timp sunt $\{1 + c - l + 2r, 1 + c - l + 6r, 1 + c - l + 10r, \dots\}$ și $\{1 - c + l + 4r, 1 - c + l + 8r, 1 - c + l + 12r, \dots\}$.

În fiecare situație contorizăm faptul ca raza intersectează traiectoria acestui rover. Numărul de rovere intersectate, fiind răspunsul cerinței 1.

Pentru cerința 2, în fiecare dintre cele 5 situații, pe baza relațiilor de tip progresie aritmetică obținute din periodicitatea întâlnirii roverului cu raza, vom folosi un vector de frecvență pentru a marca la fiecare moment care rovere ar fi afectate. La final determinăm maximul din acest vector și momentul în care raza va emite.

5.3.C Problema Text

Propusă de: prof. Vlad-Laurențiu Nicu

Soluție prof. Daniela-Elena Lica

Cerința 1 Presupunând că șirul de caractere citit are o lungime de L caractere după eliminarea spațiilor, se poate determina numărul de linii, respectiv de coloane, într-o complexitate egală cu $O(\sqrt{L})$, prin parcurgerea tuturor divizorilor lui L.

Cerința 2 Pentru ușurință, se poate implementa o funcție ce primește ca parametru un șir de caractere să zicem, de lungime v, și determină într-o complexitate pătratică $O(v^2)$ care, în practică, se comportă foarte bine, care este cea mai lungă secvență palindromică a acestuia.

Pentru determinarea unei secvențe maximale palindromice, se poate alege care să fie caracterul din mijloc pentru palindroame de lungime impară, respectiv care să fie cele două caractere identice din mijloc pentru palindroame de lungime pară, și apoi se încearcă extinderea simultană a capetelor palindromului, spre stânga cât și spre dreapta.

Astfel, putem apela această funcție de $N + M$ ori, unde N este numărul de linii ale matricei obținute, și M este numărul de coloane ale matricei obținute, transmițând, de fiecare dată, ca parametru al funcției, șirul obținut prin parcurgerea unei linii, respectiv a unei coloane.

Cerința 3 Problema se poate rezolva într-o complexitate de timp $O(NM) = O(L)$.

Astfel, încercăm să găsim răspunsul parcurgând linie cu linie matricea obținută.

Presupunem că, la pasul curent, ne aflăm pe linia i. Introducem notația:

$$H_j = \begin{cases} \text{lungimea maximă a unei subsecvențe de vocale ce se termină în celula } (i, j). \\ \text{Astfel, } H_j \text{ poate lua doar valori naturale, cuprinse între 1 și } M. \end{cases}$$

În mod particular, dacă celula (i, j) conține o consoană, atunci $H_j = 0$.

Încercăm, pe rând, să fixăm fiecare coloană j ca fiind coloana cu *înălțimea* minimă dintr-un subtablou dreptunghiular ce se termină pe linia i.

Definim:

$$left_j = \text{cea mai din stânga poziție pentru care } H_{left_j+1} \geq H_j, H_{left_j+2} \geq H_j, \ldots, H_{j-1} \geq H_j$$

$$right_j = \text{cea mai din dreapta poziție pentru care } H_{j+1} \geq H_j, \ldots, H_{right_j-2} \geq H_j, H_{right_j-1} \geq H_j$$

Putem observa că pentru orice j subtabloul dreptunghiular în cauză are: $l = H_j$ linii și $c = right_j - 1 - (left_j + 1) + 1$ coloane, deci o arie egală cu: lc elemente.

Tablourile unidimensionale *left* și *right* pot fi calculate în complexitate de timp $O(M)$, cu ajutorul a două parcurgeri, una de la stânga la dreapta și una de la dreapta la stânga. Pentru a afla răspunsul, respectiv aria maximă a unui dreptunghi ce conține doar vocale, se calculează maximul tuturor numerelor lc descrise anterior.

Soluție prof. Vlad-Laurențiu Nicu

Cerința 1 Se citește textul și se elimină spațiile formându-se o matrice de caractere de dimensiune divizori ai lungimii și e afișează matricea obținută.

Cerința 2 Pentru determinarea palindromului de lungime maximă se poate folosi algoritmul lui Manacher.[1]

Acest algoritm constă în modificarea șirului inițial prin inserarea unui caracter oarecare (care sigur nu apare în sir exemplu caracterul `*`) între oricare două caractere. Nu e nevoie de transformarea efectivă a șirului, se poate lucra cu indicii șirului astfel încât să considerăm ca acest caracter a fost inserat. De exemplu șirul `abccba` devine `*a*b*c*c*b*c*`.

Pentru a găsi cea mai lungă secvență palindrom va trebui să extindem în jurul fiecărui caracter t_i, astfel încât secvența dintre t_{i-d} și t_{i+d} să fie palindrom.

Vom păstra în vectorul L_i – cel mai lung palindrom cu centrul în i. Valoarea maximă din vectorul L va reprezenta lungimea celui mai lung palindrom.

Cerința 3 Pentru determinarea dreptunghiului de arie maximă folosim un algoritm cu histograme (o funcție in trepte).

Se fixează două linii i_1 și i_2. Pentru fiecare linie se determină numărul minim de zerouri spre dreapta pe liniile $i_1, i_1 + 1, \ldots, i_2$.

Se determină astfel aria maximă.

Soluție stud. Bogdan-Ioan Popa, stud. Tulbă-Lecu Theodor-Gabriel

Cerința 3 Se iterează prin fiecare pereche de linii $(i_1, i_2), i_1 \leq i_2$. Pentru o pereche (i_1, i_2) fixată vom dori să calculăm care este dreptunghiul de arie maximă ce conține doar vocale care are linia de sus pe linia i_1 și linia de jos pe linia i_2. Pentru a reuși acest lucru vom considera vectorul $C_j = 1$, dacă pe coloana j între liniile i_1 și i_2 sunt doar vocale sau 0 dacă nu. Calcularea acestui vector se poate face cu sume parțiale sau folosindu-ne de vectorul C calculat anterior pentru perechea $(i_1, i_2 - 1)$.

Ce rămâne de făcut este să găsim cea mai lungă secvență de valori consecutive, fie ea de lungime K. Atunci dreptunghiul de arie maximă care conține doar vocale cu linia sus pe i_1 și linia de jos pe i_2 are aria egală cu $K(i_2 - i_1 + 1)$. Complexitatea timp a acestei soluții este $O(MN^2) = O(L\sqrt{L})$.

Soluție prof. Șerban Marinel

Cerința 1 Citirea se face caracter cu caracter, reținându-se într-un vector caracterele diferite de spațiu. Evident, concomitent se determină numărul de caractere, apoi se calculează numărul de linii L și numărul de coloane C al matricei pentru cerința 1.

[1] https://cp-algorithms.com/string/manacher.html

Cerința 2 Pentru a determina palindromul maxim se parcurge matricea linie cu linie, căutându-se palindroame de lungime *lung* din mulțimea $\{C, C-1, C-2, \dots\}$.

Dacă *lung* devine mai mic decât lungimea palindromului detectat până în acel moment se trece la linia următoare. Dacă se determină un palindrom de lungime egală cu lungimea maximă detectată până în acel moment, se reține palindromul cel mai mare lexicografic. Același procedeu se repetă apoi coloană cu coloană, plecând de acestă dată de la *lung* = L, apoi $L-1$, etc.

Cerința 3 Pentru acestă cerință construim o matrice în care am înlocuit vocalele cu 1 și consoanele cu 0 - acestă matrice poate fi calculată încă de la cerința 1. Pornind de la acestă matrice construim o *matrice de sume parțiale*. În acestă matrice am căutat dreptunghiuri de sumă maximă. Pentru datele de test foarte mari acestă metodă depășește timpul de execuție dat.

5.4 Clasa a VIII-a

5.4.A Problema Proeminența

Propusă de: Lector dr. Paul Diac — Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iaşi

Pentru ambele cerințe trebuie eliminate altitudinile consecutive egale. Acest lucru se poate face prin citirea altitudinilor într-o variabilă temporară x și adăugarea în vector doar a valorilor diferite de ultima adăugată:

```
if (x != a[n-1]) { a[n++] = x; }
```

Cerintă 1

Se parcurg altitudinile $0 < i < n-1$ și se numără vârfurile (pe poziția i este un vârf dacă $a[i-1] < a[i] > a[i+1]$).

Cerința 2

Soluție $O(N^3)$ O primă soluție pentru a determina proeminența unui vârf de pe poziția i este să aplicăm definiția în mod direct. Parcurgem orice alt vârf j și le procesăm pe cele cu $a[j] > a[i]$. Pentru a determina cea mai adâncă vale dintre i și j, se parcurg pozițiile k dintre $\min(i,j)$ și $\max(i,j)$ și se reține cea mică valoare $a[k]$. Proeminența vârfului i va fi diferența minimă dintre $a[i]$ și valorile $a[k]$ obținute astfel. Soluția se încadrează în timp pe testele cu $N \leq 1000$.

Soluție $O(N^2)$ După ce fixăm un vârf i pentru care calculăm proeminența, putem parcurge vârfurile j începând de la pozitia i prima dată în stânga, apoi în dreapta. Prin aceste parcurgeri putem determina vârfuri $a[j] > a[i]$ concomitent cu a determina cea mai adâncă vale dintre ele. Reținem acestă altitudine minimă într-o variabilă pe care o folosim și pentru următoarele poziții j.

Optimizare Analizând mai atent definiția proeminenței, deducem că pe un profil proeminența unui vârf i va fi dată de *primul* vârf mai mare decât $a[i]$ situat în stânga sa $a[s] > a[i]$ ($s < i$) sau de *primul* vârf mai mare decât $a[i]$ situat în dreapta sa $a[d] > a[i]$ ($d > i$). Justificarea constă în faptul că, dacă ar exista alte vârfuri $a[k] > a[i]$ cu $k < s$ sau $k > d$ până la care văile ar fi și mai adânci, ele nu vor fi relevante, deoarece atunci când se alege minimul dintre diferența $a[i]$ și cea mai adâncă vale, ele dau o diferență mai mare, care nu modifică minimul. Soluțiile de complexitate $O(N^2)$ se încadrează în timp pe un număr semnificativ mai mare de teste, în funcție de optimizările efectuate.

Soluție $O(N)$ Pentru a proiecta o soluție de complexitate liniară determinăm mai întâi *„proeminența la stânga"*, parcurgând vârfurile de la stânga către dreapta. Procedăm apoi similar pentru a determina *„proeminența la dreapta"*, parcurgând vârfurile de la dreapta către stânga și la final afișăm minimul dintre cele două valori obținute pentru proeminență.

Pentru a calcula *proeminența la stânga*, parcurgem vârfurile în ordinea crescătoare a indicilor i. Pentru orice vârf i doar cel mai apropiat vârf j din stânga lui i ($j < i$), strict mai înalt decât vârful i este relevant, prin urmare toate vârfurile dintre acestea $j < k < i$ care au altitudini mai mici $a[k] < a[i]$, pot fi ignorate mai departe. Este deci suficient să reținem o *stivă* cu vârfuri montane în ordinea strict descrescătoare a înălțimilor. Pentru fiecare dintre aceste vârfuri, vom reține și cea mai adâncă vale situată în dreapta vârfului, până la următorul vârf adăugat în stivă, vale pe care o determinăm „din mers".

Rezumând, parcurgem profilul de la stânga la dreapta şi identificăm vârfurile montane (fie $a[i]$ altitudinea vârfului montan curent).

Cât timp în stivă există elemente şi $a[i] \geq$ altitudinea vârfului montan plasat la vârful stivei, extragem din stivă un element (evident, doar elementul situat la vârful stivei poate fi extras). Atunci când extragem un element din stivă, actualizăm, dacă este cazul, şi înălţimea celei mai adânci văi până la vârful montan curent.

Dacă stiva a devenit vidă, inserăm vârful montan curent în stivă şi iniţializăm înălţimea celei mai adânci văi până la momentul curent cu $a[i]$.

Dacă stiva este nevidă, atunci $a[i] <$ altitudinea vârfului plasat la vârful stivei. În acest caz:

1. pentru vârful montan situat în vârful stivei memorăm cea mai mică altitudine a unei văi identificate până la vârful montan curent;
2. determinăm proeminenţa la stânga a vârfului montan curent ca fiind diferenţa dintre $a[i]$ şi altitudinea minimă a unei văi până la vârful montan curent;
3. reiniţializăm înălţimea minimă a văii curente cu $a[i]$;
4. inserăm vârful montan curent în stivă.

Pentru a evita duplicarea codului pentru *„proeminența la dreapta"*, putem oglindi vectorul și repeta aceeași procedură.

5.4.B Problema RGB

Propusă de: stud. Ioan Cristian Pop — Universitatea Politehnica București

O observație cheie care simplifică implementarea problemei este aceea că este imposibil ca doi extratereștri să aibă puteri egale în cadrul unei lupte, toți având puteri impare distincte. În cadrul unei lupte în care extratereștrii au culori diferite, puterea unuia dintre extratereștri va deveni pară (iar comparația va fi par cu impar, deci tot între valori distincte). În plus, puterile extratereștrilor de aceeași culoare sunt ordonate crescător în fișierul de intrare.

O altă observație cheie este că problema se rezolvă similar pentru fiecare culoare, deci rezolvarea pentru o anumită culoare se poate aplica și pentru celelalte două culori.

Cerința 1 — Soluție $O(N)$

Pentru $C = 1$, este suficient să calculăm câte lupte va câștiga cel mai puternic extraterestru de fiecare culoare, apoi să afișăm puterea extraterestrului care câștigă un număr maxim de lupte. Pentru a calcula numărul de lupte câștigate de un extraterestru, este necesar să calculăm doar câte lupte va câștiga luptând cu extratereștrii de celelalte două culori (el fiind cel mai puternic de culoarea sa va câștiga toate luptele cu extratereștrii având aceeași culoare cu el). O atenție specială trebuie acordată cazului în care există mai mulți extratereștri care câștigă același număr maxim de lupte (se va afișa puterea cea mai mică).

Cerința 2 — Soluție $O(N^2)$

Pentru $C = 2$, simulăm toate luptele dintre extratereștri și reținem câte lupte câștigă fiecare.

Cerința 2 — Soluție $O(N \log N)$

Fie X și Y doi extratereștri de culori diferite. Observăm că, dacă X va câștiga lupta contra lui Y, atunci X va câștiga luptele contra tuturor extratereștrilor de aceeași culoare ca Y, însă cu putere mai mică. Prin urmare, pentru fiecare extraterestru, putem căuta binar care este cel mai puternic extraterestru de fiecare culoare diferită de a sa pe care îl va învinge. O astfel de soluție poate obține mai multe puncte în funcție de cât de eficientă este implementarea.

Cerința 2 — Soluție $O(N)$

Putem optimiza soluția anterioară. Fie Z următorul extraterestru de aceeași culoare ca X (cu putere mai mare). Atunci, Z va câștiga din start toate luptele pe care X le va câștiga. Deci, ce trebuie verificat este dacă mai câștigă și alte lupte în plus. Să presupunem că A este cel mai puternic extraterestru de o culoare diferită pe care X îl învinge. Atunci, Z îi va învinge pe toți până la A inclusiv și vom verifica dacă va mai învinge și pe alții de aceeași culoare, căutând secvențial, începând după A. Să presupunem că cel mai puternic extraterestru pe care îl va învinge va fi B. Atunci, pentru următorul extraterestru de aceeași culoare ca X și Z, căutarea va începe după B, și așa mai departe. Analog pentru cealaltă culoare diferită. Tehnica de rezolvare mai este cunoscută drept „Two pointers".

De precizat este că citirea poate fi parsată (datele citite ca șir de caractere și apoi transformate în numere). Nu este însă necesară pentru a obține 100 de puncte.

5.4.C Problema Subșir

Propusă de: prof. Ionel-Vasile Piţ-Rada — Colegiul Naţional „Traian", Drobeta-Turnu Severin

Cerința 1 — Soluție „naivă", $O(NNr)$

Pentru fiecare pereche (x, j), se determină șirul cifrelor lui x și apoi, începând cu cifra cea mai semnificativă (cea din stânga) și continuând spre cifra cea mai puțin semnificativă (ultima, cea din dreapta), se caută succesiv prima apariție a cifrei respective și când se găsește această poziție se continuă căutarea de acolo pentru cifra următoare. Dacă se depășește poziția j sau una dintre cifre nu se mai găsește, atunci se oprește căutarea.

Cerința 2 — Soluție „„naivă", $O(N\alpha)$

Am notat cu α suma lungimilor intervalelor de interogare, care este cel mult egală cu $10000000Nr$. Pentru fiecare interval $[a, b]$ și pentru fiecare număr x din interval se aplică ideea de la cerința 1, cu limita de căutare $j = N$.

Optimizări

Pentru obținerea unei performanțe mai bune este util să facem următoarele precalculări:

1. Pentru fiecare poziție $1 \leq i \leq N$ și pentru fiecare cifră zecimală $0 \leq cif \leq 9$, determinăm $poz[i][cif]$= cea mai mică (prima) poziție din șirul S pe care apare cifra cif, după poziția i sau $N + 1$ în caz că cifra cif nu mai apare în S după poziția i.
   Pentru a verifica dacă x apare ca subșir în prefixul de lungime j al lui S, parcurgem cifrele lui x de la stânga la dreapta și construim subșirul preluând pozițiile din poz (dacă cifra curentă din x este cif, iar poziția pe care apare precedenta cifra din x este i, atunci poziția în subșir a cifrei cif va fi $poz[i][cif]$. Astfel, numărul de pași efectuați pentru verificarea unei valori x este cel mult egal cu numărul de cifre din x.
   Asemănător, se poate rezolva cerința 2, verificând succesiv fiecare valoare din fiecare interval $[a, b]$.
2. O a doua precalculare foarte utilă este aceea prin care pentru fiecare număr x din intervalul $[0, 10^7]$ se determină $minpoz[x]$ = cea mai mică poziție din S cu proprietatea că numărul x apare ca subșir în prefixul de lungime $minpoz[x]$ al lui S, respectiv $minpoz[x] = N + 1$ în cazul în care x nu apare ca subșir în S.
   Inițializarea se face prin $minpoz[x] = poz[1][x]$, pentru $0 \leq x \leq 9$, apoi, în ordine crescătoare, pentru fiecare număr x cu $10 \leq x \leq 10^7$, vom calcula

   ```
   minpoz[x]=poz[minpoz[x/10]][x%10];
   ```

   Valorile pentru $minpoz$ se determină astfel cu complexitate liniară.
3. O a treia precalculare se poate realiza utilizând tehnica sumelor parțiale. Vom determina $nr[x]$ = numărul de numere din intervalul $[0, x]$ care apar ca subșir în S. Utilizând vectorul nr, putem determina numărul de valori x din intervalul $[a, b]$ care apar ca subșir în S ca fiind $nr[b] - nr[a - 1]$ (dacă $a > 0$) sau $nr[b]$ (dacă $a = 0$).

După aceste precalculări, la cerința 1, pentru fiecare pereche (x, j) verificarea are complexitatea $O(1)$, deci complexitatea totală (pentru cele Nr perechi) este $O(Nr)$. Analog, la cerinţa 2 pentru fiecare interval $[a, b]$ numărarea se va realiza tot în $O(1)$, deci complexitatea totală (pentru cele Nr intervale) va fi $O(Nr)$.

Problema admite multiple alte abordări, care obţin diferite punctaje.

5.5 Clasa a IX-a

5.5.A Problema Colibri

Propusă de: stud. Alexandru Petrescu, Universitatea din Oxford
Pregătită de: stud. Andrei Arhire, Universitatea A.I. Cuza, Iași
Editorial redactat de: stud. Andrei Arhire, Universitatea A.I. Cuza, Iași

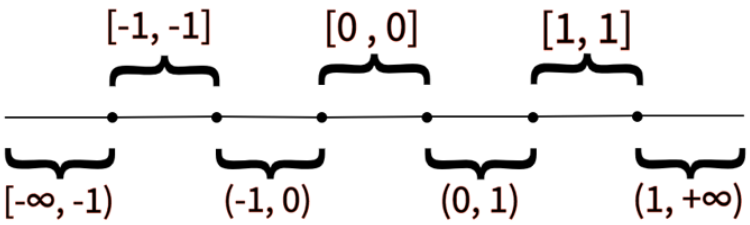
Soluția 1

Vom încerca să găsim subșirul de termeni cu cel mai mare produs presupunând că produsul aparține intervalului $[1, +\infty)$. În cazul în care nu există un astfel de subșir vom căuta în mod similar pentru intervalele $(0, 1)$, $[0, 0]$ și $[-\infty, 0)$.

- O variantă de subșir care este soluție și are produsul în $[1, +\infty)$ conține toate fracțiile pozitive supraunitare și pozitive echiunitare. De asemenea conține și cele mai mici $2K$ fracții negative cu mențiunea că cele mai mari două dintre acestea formează un produs mai mare sau egal cu 1.
- În continuare știm că nu exista soluție în $[1, +\infty)$ și încercăm să găsim una în $(0, 1)$. Dacă există atunci conține cel mult 2 fracții. Fie este formată din cea mai mare fracție pozitiva, fie este formată din cele mai mici 2 fracții negative.
- Cel mai mare produs al unui subșir este 0 atunci când nu exista nicio fracție pozitivă, există cel mult o fracție negativă și cel puțin o fracție cu rezultatul 0.
- Produsul aparține intervalului $[-\infty, -1)$ când există o singură fracție cu valoare negativă.

Plecând de la aceste observații problema se poate rezolva în $O(N \log N)$. Mai întâi se verifică dacă produsul este cel mult 0. Se sortează fracțiile după rezultat și se mențin 2 pointeri, fiecare la un capăt. Cel din capătul stâng se va deplasa cu câte 2 poziții spre dreapta cât timp produsul fracției indicate cu cel al fracției următoare este cel puțin 1 și ambele fracții sunt negative, iar cel din capătul drept se va deplasa cu câte o poziție spre stânga cât timp fracția indicată este cel puțin 1. Dacă în urma operațiilor ambii pointeri nu se află în pozițiile inițiale atunci nu există soluție cu produs în $[1, +\infty)$. Altfel subșirul este format din primele două sau ultima fracție.

Soluția 2



Pentru a rezolva problema vom împarți fracțiile în 7 categorii în funcție de rezultat.
Mai departe ne vom referi prin:

Categoria I la mulțimea fracțiilor cu rezultatul în $[-\infty, -1)$,

Categoria II la mulțimea fracțiilor cu rezultatul în $[-1, -1]$,

Categoria III la mulțimea fracțiilor cu rezultatul în $(-1, 0)$,

Categoria IV la mulțimea fracțiilor cu rezultatul în $[0, 0]$,

Categoria V la mulțimea fracțiilor cu rezultatul în $(0, 1)$,

Categoria VI la mulțimea fracțiilor cu rezultatul în $[1, 1]$,

Categoria VII la mulțimea fracțiilor cu rezultatul în $(1, +\infty)$.

O idee de rezolvare este să considerăm fiecare din cele 7 intervale în ordine descrescătoare și să verificăm dacă există un subșir de fracții care înmulțite dau o valoare în intervalul la care ne referim.

$(1, +\infty)$. Pentru a obține un produs mai mare ca 1 e nevoie ca cel puțin una din următoarele propoziții sa fie adevărate:

- există o fracție în VII;
- există două fracții în I;
- există o fracție în I și una în II;
- există o fracție în I și una în III care înmulțite dau mai mult decât 1.

Pentru a obține valoarea maxima se aleg în această ordine:

1. toate fracțiile din VII.
2. cele mai mici fracții din mulțimea I două câte două. Astfel, după alegere, în I fie nu rămâne nimic (dacă $|I|$ este număr par), fie rămâne cea mai mare fracție ($|I|$ este impar).
3. dacă a rămas o fracție în I și există o fracție în II atunci ambele sunt alese.
4. dacă a rămas o fracție în I și există o fracție în III al căror produs este > 1 atunci ambele sunt alese.

$[+1, +1]$. Din moment ce suntem aici înseamnă că nu există un subșir ale cărui fracții să dea un produs > 1. Pentru a obține un produs egal cu 1 e nevoie ca cel puțin una din următoarele sa fie adevărată:

- exista o fracție în VI;
- exista 2 fracții în II;
- exista o fracție în I și una în III care înmulțite dau 1.

$(0, +1)$. Din moment ce suntem aici înseamnă că nu există un subșir ale cărui fracții să dea un produs ≥ 1. Pentru a obține un produs din $(0, +1)$ e nevoie ca cel puțin una din următoarele sa fie adevărată:

- exista o fracție în V;
- exista o fracție în II și una în III;
- exista o fracție în I și una în III care înmulțite aparțin $(0, +1)$;

- exista 2 fracții în III.

Se alege subșirul care obține cel mai mare produs raportat la cele 4 cazuri de mai sus.

$[0, 0]$. Dacă există cel puțin o fracție cu numărătorul 0 atunci răspunsul este 0.

$(-1, 0)$ sau $[-1, -1]$ sau $[-\infty, -1)$. Dacă suntem în acestă situație atunci cu siguranță testul conține o singură fracție și aceasta este negativă. Prin urmare aceasta reprezintă și soluția problemei.

5.5.B Problema Geogra

Propusă de: stud. Gheorghe Liviu Armand, Universitatea din București
Pregătită de: stud. Gheorghe Liviu Armand, Universitatea din București
Editorial redactat de: stud. Gheorghe Liviu Armand, Universitatea din București

Pentru a rezolva problema, trebuie ca mai întâi sa calculăm șirul T.

Ordonăm șirurile R găsite de Alex în fiecare runda crescător după numărul de valori diferite de -1 din cadrul fiecărui șir. Întrucât știm că Alex respectă șirul T în fiecare rundă, putem să deducem un șir T posibil doar uitându-ne la șirurile R în ordinea stabilită anterior. Astfel, când vedem că numărul de valori diferite de -1 diferă de la un șir R la următorul șir R din ordinea stabilită, ne putem da seama că pozițiile pe care în primul sir sunt valori de -1 și în al doilea șir sunt valori diferite de -1 vor urma în șirul T după pozițiile pe care în primul sir sunt valori diferite de -1. Pentru a determina un șir T posibil este îndeajuns ca de fiecare dată când găsim astfel de poziții să le adaugăm în șirul T în ordinea în care le găsim. Șirul T obținut nu este mereu unic și este posibil să difere de cel al lui Alex, însă soluția nu este afectată de acest fapt, după cum vom vedea în continuare.

Pe baza acestui sir T obținut, putem rearanja valorile din fiecare sir R și Z_i. Astfel, valorile din fiecare sir vor fi în ordinea în care Alex le va afla.

Pentru a putea răspunde eficient la întrebări, vom construi $L+1$ vectori în care vom reține cele N locații. Al j-lea vector dintre aceștia, $0 \leq j \leq L$, va conține cele N locații ordonate după numărul format din primii j biți din cadrul șirului Z_i asociat fiecărei locații.

Pentru fiecare întrebare, ne vom uita în al nr-lea vector dintre cei $L+1$ formați anterior, unde nr reprezintă numărul valorilor diferite de -1 din șirul R asociat rundei respective. În acest vector, putem căuta binar cea mai din stânga și cea mai din dreapta apariție a numărului format din biții ce reprezinta primele nr valori din cadrul șirului R din runda actuala. Astfel, vom obține un interval nevid ce reprezinta toate locațiile care respectă șirul R aflat de Alex în runda actuala.

Cerința 1 (37 de puncte) Pentru a rezolva aceasta cerința, trebuie doar să afișăm lungimea intervalului găsit anterior.

Cerința 2 (63 de puncte) Pentru a rezolva această cerința, mai întâi trebuie să observăm că putem calcula A independent de B.

Pentru a calcula A, trebuie sa ne uitam la coordonatele X ale locațiilor din intervalul găsit anterior. Dacă valorile W ale locațiilor din interval ar fi 1, atunci A ar fi valoarea mediană a coordonatelor X ale locațiilor din interval. În cazul în care șirul are 2 mediane, atunci se alege valoarea mai mica, întrucât A trebuie sa fie minim. Dacă A ar fi o valoare mai mică decât cea mai mică mediană sau mai mare decât cea mai mare mediană, atunci d_p ar crește. Pe cazul general, unde valorile W ale locațiilor din interval nu sunt toate egale cu 1, ne putem imagina că fiecare locație apare de W-ul ei ori. Astfel, răspunsul ar fi tot mediana acestui nou (și mult mai lung) sir, adică a $sumW/2 + sumW\%2$ valoare X in ordine crescătoare, unde $sumW$ reprezinta suma valorilor W ale locațiilor din interval. Deci, dacă ne-am ordona după X locațiile din intervalul găsit cu cele doua căutări binare, atunci A va fi coordonata X a primei locații în care

suma valorilor W a locațiilor de dinaintea ei în ordinea stabilita devine mai mare sau egala cu $sumW/2 + sumW\%2$. Pentru a reuși să calculăm acest lucru, vom ordona locațiile din interval după coordonata X, vom face sume parțiale după W și vom parcurge locațiile din interval până găsim valoarea căutată.

Pentru a calcula B se procedează asemănător, singura diferența fiind că B trebuie calculat în funcție de coordonatele Y ale locațiilor din intervalul găsit în runda actuală.

Pentru a obține 100 de puncte, trebuie să calculăm o singură dată rezultatul din runde care au același sir R. Astfel, fiecare interval găsit pentru un sir R va fi parcurs o singură data. Întrucât un sir R cu un anumit număr de valori diferite de -1 determină un interval disjunct de intervalul determinat de un sir R diferit cu același număr de valori diferite de -1, fiecare element din cei $L + 1$ vectori făcuți anterior va fi parcurs o singura data în total.

5.5.C Problema Schi

Propusă de: prof. Stelian Ciurea, Universitatea „Lucian Blaga", Sibiu
Pregătită de: stud. Ioan-Bogdan Iordache, Universitatea din București
Editorial redactat de: stud. Ioan-Bogdan Iordache, Universitatea din București

Soluție în complexitate pătratică (24 de puncte)

Cutiile sunt procesate în ordinea în care sunt adăugate în turn. Notăm cutiile cu $c_1, c_2, \ldots, c_N$.

Fie două cutii c_i, c_j cu $i < j$. Considerăm că c_i a fost deja plasată în turn și cunoaștem top_i înălțimea la care se află fața superioară a acestei cutii (eventual înălțimea la care se află fața corespunzătoare capacului lipsă dacă c_i este o cutie de tipul 2, „cu golul în sus"). Știind top_i și implicit poziția cutiei c_i în spațiu, putem determina o restricție pentru top_j, considerând că nu mai există alte cutii în afară de c_i care să întrerupă căderea lui c_j.

Se disting mai multe cazuri în funcție de tipul și orientarea cutiilor (M_i, M_j) și diferența dintre dimensiunile laturilor capacelor (L_i, L_j). Vom arăta mai jos câteva cazuri mai interesante:

- dacă $M_i = 2$, $M_j = 1$ și $L_i < L_j$, obținem restricția $top_j \geq top_i + H_j$
- dacă $M_i = 2$, $M_j = 1$ și $L_i > L_j$, obținem restricția $top_j \geq top_i - H_i + H_j$
- dacă $M_i = 1$, $M_j = 0$ și $L_i < L_j$, obținem restricția $top_j \geq top_i$

De asemenea, cutiei c_j i se mai impune o restricție și de către podea: $top_j \geq H_j$.

Calculând pentru cutia c_j restricțiile impuse de toate cutiile $c_1, c_2, \ldots, c_{j-1}$ și de către podea, putem afla top_j ca fiind maximul dintre acestea.

Înălțimea turnului va fi în final $\max\{top_1, top_2, \ldots, top_N\}$.

Pentru a determina care cutii sunt vizibile din lateral, vom defini „intervalul de vizibilitate" al unei cutii intervalul de înălțimi la care acea cutie este vizibilă (evident ne va interesa pentru câte cutii acest interval are lungime nenulă).

La început, pentru toate cutiile de forma c_i inițializăm intervalul de vizibilitate cu $[top_i - H_i, top_i]$ (intervalul de înălțimi pe care îl ocupă cutia c_i). Cutia c_i poate fi ascunsă/acoperită parțial sau total doar de cutii c_j, pentru care $L_j > L_i$. Așadar, ne vom uita la astfel de cutii c_j, care fie nu acoperă în niciun fel cutia c_i, fie o acoperă total, deci deja putem stabili că c_i nu este vizibilă din lateral, fie parțial, caz în care c_j acoperă fie un prefix, fie un sufix al intervalului de vizibilitate al lui c_i.

Matematic, dacă notăm viz_i intervalul de vizibilitate al lui c_i calculat până la un pas intermediar și dorim să actualizăm acest interval pe baza unei cutii c_j cu $L_j > L_i$, realizăm diferența de intervale/mulțimi: $viz_i \leftarrow viz_i \setminus [top_j - H_j, top_j]$. Este ușor de observat din procesul de plasare al cutiilor, că această diferență va rezulta mereu într-un interval.

Rezolvând astfel ambele cerințe ale problemei, obținem complexitatea $O(N^2)$.

Soluție în complexitate liniară (100 de puncte)

Pentru a obține 100 de puncte, va fi nevoie de o rafinare a soluției de mai sus. Pentru a rezolva prima cerință vom încerca să calculam din nou top_i pentru fiecare cutie c_i. Observația care va reduce complexitatea este că nu avem nevoie să calculăm restricțiile impuse de $c_1, \ldots, c_{i-1}$, cutiei c_i, ci doar de către o parte dintre ele.

Notăm $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ ($1 \leq i_1 < i_2 < \ldots < i_k < i$) cutiile care ar putea întrerupe căderea lui c_i. Observăm ca acest șir de cutii este descrescător după L: dacă am avea c_{i_j} și $c_{i_{j+1}}$ cu $L_{i_j} < L_{i_{j+1}}$, cutia c_{i_j} ar fi inutilă (oricum ar cădea, c_i se va lovi mai întâi de $c_{i_{j+1}}$, nu de c_{i_j}).

O altă observație utilă este că pentru două cutii c_{i_j} și $c_{i_{j+1}}$ cu $L_{i_j} > L_{i_{j+1}} > L_i$, restricția impusă de $c_{i_{j+1}}$ cutiei c_i va fi cel puțin la fel de mare ca restricția impusă de c_{i_j}. Cu alte cuvinte, în subșirul descrescător de cutii $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$, pentru a determina top_i vom calcula restricțiile doar cu un sufix al acestui subșir (cutiile cu L mai mic decât L_i, și cutia cu L minim mai mare decât L_i).

După ce am calculat top_i, eliminăm din subșir sufixul format din cutii cu L mai mic decât L_i și adăugăm cutia c_i la finalul subșirului. Astfel per total numărul de restricții calculat pentru toate cutiile este direct proporțional cu numărul de ștergeri din acest subșir, iar din moment ce o cutie este adăugată și ștearsă din subșir cel mult o dată, complexitatea va fi liniară.

Pentru a determina vizibilitatea cutiilor, vom folosi noțiunea de interval de vizibilitate descrisă mai sus. Vom actualiza aceste intervale pentru toate cutiile mai întâi considerând doar acoperirile realizate de cutiile de tipul 2 (cu golul în sus) și apoi separat cele realizate de cutii de tipul 0 (cu golul în jos).

Vom explica în continuare procesul pentru acoperirile produse de cutii de tipul 2. Vom procesa cutiile în ordinea în care au fost adăugate în turn. În momentul de față suntem la cutia c_i de un tip oarecare și vrem să vedem ce cutii de tipul 2 o acoperă parțial/total. Evident aceste cutii, ca să poată acoperi c_i, trebuiau adăugate înainte și să aibă L mai mare decât L_i. Dintre cutiile de tip 2 care respectă aceste condiții, ne interesează *top*-ul maxim (extremitatea superioară maximă) a acestor cutii care poate acoperi un prefix al intervalului de vizibilitate pentru c_i.

O observație care ne va duce la soluția dorită este că dacă după o cutie de tip 2 se adaugă la un pas ulterior o cutie cu L mai mare (de orice tip), cutia de tip 2 nu va mai acoperi pe nimeni începând cu acest moment.

De aceea, la pasul i putem menține subșirul de cutii de tip 2: $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ ($1 \leq i_1 < i_2 < \ldots < i_k < i$) cu proprietatea că $L_{i_1} > L_{i_2} > \ldots > L_{i_k}$. Când procesăm cutia c_i eliminăm din subșir toate cutiile cu L mai mic decât L_i (acestea nu pot acoperi c_i și nicio altă cutie adăugată în viitor). Presupunem că rămânem acum cu k' cutii în subșir, pentru a determina ce prefix din intervalul de vizibilitate al lui c_i ar fi acoperit, trebuie să determinăm $\max\{top_{i_1}, top_{i_2} \ldots, top_{i_{k'}}\}$. Acesta nu este altceva decât un maxim pe prefix pentru subșirul nostru, pe care îl putem menține în complexitate constantă atunci când modificăm subșirul. După acest pas, în cazul în care c_i este de tipul 2 o adăugăm la finalul subșirului și continuăm cu cutia $i + 1$.

Pentru determinarea acoperirilor produse de cutii de tipul 0 (cu golul în jos) strategia este similară, însă cutiile vor trebui parcurse în ordinea inversă a adăugării lor în turn („de sus în jos").

În final, complexitatea obținută pentru ambele cerințe este $O(N)$.

5.6 Clasa a X-a

5.6.A Problema Munte

Propusă de: Prof. Szabó Zoltan — Inspectoratul Școlar Județean Mureș
Stud. Theodor-Gabriel Tulbă-Lecu — Universitatea Politehnica București

Toate demonstrațiile lemelor folosite pentru demonstrarea corectitudinii acestei probleme se pot găsi la finalul acestei secțiuni.

Observații

În primul rând, pentru a rezolva problema trebuie să înțelegem ce efect au transformările `swap` și `dswap` asupra unei permutari:

- Operația `swap` schimbă între ele două elemente egal distanțate de capetele permutării
- Operația `dswap` schimbă între ele două elemente arbitrare, dar odată cu acestea schimbă si elementele corespunzătoare egal depărtate de capetele permutării ale acestor elemente.

Astfel, putem observa următorul invariant: Două elemente egal distanțate de capetele permutării a_i și a_{n-i+1} vor rămâne egal distanțate indiferent de ce operații efectuăm asupra permutării. Vom numi astfel de numere *numere perechi*.

De asemenea, în cazul în care permutarea are lungime impară, atunci elementul $a_{\frac{n+1}{2}}$ este un punct fix, adică nu putem să îi modificăm poziția cu niciuna din cele două tipuri de operații. Acest caz este identic cu cel în care permutarea are lungime pară și nu îl vom folosi pentru demonstrația soluției.

În continuare, pentru a oferi un mod vizual de întelegere a demonstrațiilor si pentru a vizualiza care sunt numerele perechi vom reprezenta o permutare astfel:

În cazul în care $n = 2k$ este par:

a_1	a_2	$\ldots$	a_{k-1}	a_k
a_n	a_{n-1}	$\ldots$	a_{n-k+2}	a_{n-k+1}

Permutare munte

O permutare munte este o permutare care până la un moment dat este strict crescătoare, iar mai apoi devine strict descrescătoare. Vom nota acesată poziție în care se face tranziția între cele două monotonii vf, unde $1 < vf \leq k$. Dacă $vf > k$, atunci putem aplica o transformare `swap` pe toate valorile de la 1 la k, acest proces are ca efect inversarea primei linii cu cea de-a doua.

Deci, fără pierderea generalității, o permutare munte arată astfel:

a_1	$<$	a_2	$<$	$\ldots$	$<$	a_{n-vf+1}	$<$	$\ldots$	$<$	a_{k-1}	$<$	a_k
												$\wedge$
a_n	$<$	a_{n-1}	$<$	$\ldots$	$<$	a_{vf}	$>$	$\ldots$	$>$	a_{n-k+2}	$>$	a_{n-k+1}

Permutarea munte minim lexicografică (PMML)

Dintre toate permutările munte care se pot obține dintr-o anumită permutare, există o permutare unică care este minim lexicografică.

O permutare a este mai mică lexicografică decat o altă permutare b dacă si numai dacă $\exists k \in \{1, 2, \dots n\}$ astfel încât $a_i < b_i$ și $a_j = b_j$, $\forall\ i \in \{1, 2, \dots, i-1\}$.

Lemă 1. *PMML respectă propriatatea că* $a_i < a_{n-i+1}\ \forall\ i \in \{1, 2, \dots, k\}$ *și poate fi reprezentată grafic astfel:*

$$\begin{array}{ccccccccccccccc} a_1 & < & a_2 & < & \dots & < & a_{n-vf+1} & < & \dots & < & a_{k-1} & < & a_k \\ \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge \\ a_n & < & a_{n-1} & < & \dots & < & a_{vf} & > & \dots & > & a_{n-k+2} & > & a_{n-k+1} \end{array}$$

Corolar 1. Orice permutare pentru care se poate ajunge folosind doar operații de swap și dswap la o permutare munte poate fi transformată în complexitate $O(n)$ sau $O(n \log_2 n)$ în PMML.

Calcularea numărului de permutări munte

Datorită Corolarului 1, știm că putem obține din permutarea inițială PMML. Totodată, cum în Lema 1 am demonstrat că din orice permutare munte se poate ajunge la PMML doar prin operații de swap și putem observa că operația de swap este propria sa inversă (dacă aplicăm swap de 2 ori pe aceeasi coloană ne întoarcem la starea inițială), și reciproca este adevărată: putem ajunge din PMML la orice permutare munte ce poate fi obținută folosind doar operații de swap.

O altă observație este că două operații de swap pe două poziții distincte $i \neq j$, sunt independente și putem alege pentru fiecare dacă o aplicăm sau nu. Astfel, numărul de permutări munte ce poti fi obținute va fi 2^C, unde C reprezintă numărul maxim de operații de swap distincte pe care le putem aplica asupra PMML astfel încât rezultatul să rămână o permutare munte.

$$\begin{array}{ccccccccccccccccccc} a_1 & < & a_2 & < & \dots & < & a_{i-1} & < & a_i & < & \dots & < & a_{n-vf+1} & < & \dots & < & a_{k-1} & < & a_k \\ \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge & & \wedge \\ a_n & < & a_{n-1} & < & \dots & < & a_{n-i} & < & a_{n-i+1} & < & \dots & < & a_{vf} & > & \dots & > & a_{n-k+2} & > & a_{n-k+1} \end{array}$$

Din desenul anterior putem observa că pentru a putea aplica swap pe o anumită poziție $1 \leq i \leq k$, trebuie fie ca $i = 1$ sau $a_i > a_{n-i}$.

Complexitatea algoritmului este dată de complexitatea aducerii permutării inițiale la PMML și se arată in Corolarul 1 că poate fi implementată în $O(n)$ sau $O(n \log_2 n)$. Această soluție obține 100 de puncte.

Demonstrații

Demonstrația lemei 1. Vom presupune că există PMML cu un $i \in \{1, 2, \dots, k\}$ astfel încât $a_i > a_{n-i+1}$ și $a_j < a_{n-j+1}\ \forall\ 1 \leq j < i$.

Pentru $i > n - vf + 1$, intrucât subșirul $a_{n-vf+1}, \dots, a_{vf+1}$ este sortat crescător nu poate exista un i, astfel încât $a_i > a_{n-i+1}$ și să fie respectată proprietatea de munte.

Pentru $1 \leq i < n - vf + 1$ premutarea arată astfel:

a_1	$<$	a_2	$<$	$\ldots$	$<$	a_{i-1}	$<$	a_i	$<$	$\ldots$	$<$	a_{n-vf+1}	$<$	$\ldots$	$<$	a_{k-1}	$<$	a_k
$\wedge$		$\wedge$		$\wedge$		$\wedge$		$\vee$		?		?		$\wedge$		$\wedge$		$\wedge$
a_n	$<$	a_{n-1}	$<$	$\ldots$	$<$	a_{n-i}	$<$	a_{n-i+1}	$<$	$\ldots$	$<$	a_{vf}	$>$	$\ldots$	$>$	a_{n-k+2}	$>$	a_{n-k+1}

Din figura anterioară putem observa că $a_i > a_{n-i+1} > a_{n-i}$ și $a_{n-i+1} > a_{n-i} > a_{i-1}$, deci putem inversa a_i cu a_{n-i+1} folosind o operație de swap si se va păstra proprietatea de munte, iar pentru a pătsra proprietatea de munte și pentru restul sirului, vom aplica swap si pe toate perechile de la $i+1$ la k asfel:

a_1	$<$	a_2	$<$	$\ldots$	$<$	a_{i-1}	$<$	a_{n-i+1}	$<$	$\ldots$	$<$	a_{vf}	$>$	$\ldots$	$>$	a_{n-k+2}	$>$	a_{n-k+1}
$\wedge$		$\wedge$		$\wedge$		$\wedge$		$\wedge$		?		?		$\vee$		$\vee$		$\vee$
a_n	$<$	a_{n-1}	$<$	$\ldots$	$<$	a_{n-i}	$<$	a_i	$<$	$\ldots$	$<$	a_{n-vf+1}	$<$	$\ldots$	$<$	a_{k-1}	$<$	a_k

Noua permutare obținută este tot o permutare munte și este mai mică lexicografic decât permutarea inițială pentru că $a_i > a_{n-i+1}$, deci permutarea inițială nu putea fi PMML. □

Demonstrația Corolarului 1. Utilizând algoritmul de la Lema 1 de mai multe ori, putem demonstra prin inducție matematică că orice permutare munte, poate fi transformată într-o PMML utilizând doar operații de tip swap, deoarece la fiecare pas al algoritmului, prefixul de perechi $a_i > a_{n-i+1}$ crește în lungime cu cel puțin 1.

Mai mult, întrucât știm că pentru orice permutare posibilă a datelor de intrare a problemei există cel putin o permutare munte ce poate fi obținută, atunci putem ajunge la PMML în următorul mod:

- realizăm operații de swap pentru toate perechile cu $a_i > a_{n-i+1}$, unde $i \in \{1, 2, \ldots, k\}$ – acest lucru ne garantează proprietatea de minimalitate lexicografică
- sortăm crescător folosind operații de dswap elementele de la a_1 la a_k. Datorită faptului că se garantează existența a cel puțin unei permutări munte, celelalte elemente vor fi sortate sortate crescător de la a_{n-k+1} la a_{vf} și descrescător de la a_{vf} la a_n. Acest lucru ne garantează proprietatea de munte.

Complexitatea acestui algoritm este dată de sortarea numerelor pereche. Luând în considerare faptul că sirul este o permutare, numerele sunt cuprinse între 1 și n, astfel se pot obține complexitătile:

- $O(n^2)$, dacă se folosesc algoritmi suboptimi precum Bubble Sort.
- $O(n \log n)$, dacă se folosesc algoritmi optimi de sortare prin comparare precum Merge Sort.
- $O(n)$ dacă se folosește Count Sort. □

5.6.B Problema ChangeMin

Propusă de: Stud. Popa Bogdan-Ioan — Universitatea din București

Cerința 1

Se parcurge șirul de la final la început și se introduc elementele rând pe rând într-o stivă. Înainte de a introduce un element A_i în stivă vom scoate din vârful stivei acele valori care sunt mai mici sau egale cu A_i. Se observă că parcurgând stiva din vârful ei în jos vom obține șirul de valori pe care variabila *min* (din algoritmul în pseudocod prezentat) le va lua. Astfel, după inserarea lui A_i în stivă este suficient să adunăm la *cnt* lungimea stivei. Complexitate $O(N)$.

Cerința 2

Pentru a rezolva cerința 2 va trebui mai întâi să obținem valoarea lui *cnt* în urma rezolvării cerinței 1. Mai departe vom face o parcurgere asemănătoare cu cea de la cerința 1. Aflându-ne la indicele *i* după ce am făcut inserarea lui A_i in stivă vom scadea din *cnt* pe L, lungimea stivei S. Pentru a putea calcula variabila *score* va trebui să ținem încă două informații:

$sumCoef = 1 \cdot A_{S_L} + 2 \cdot A_{S_{L-1}} + \cdots + L \cdot A_{S_1}$, care ne va ajuta la calcularea lui *score* și $sumAll = A_{S_L} + A_{S_{L-1}} + \cdots + A_{S_1}$ care ne va ajuta să calculăm atât *score* cât și *sumCoef*. Având aceste valori calculate vom putea aduna la *score* valoarea $cnt \cdot sumAll + sumCoef$

5.6.C Problema Dragonfruit

Propusă de: Stud. Coroian David-Nicolae — Delft University of Technology
Asist. Drd. Andrei-Costin Constantinescu — ETH Zürich
Stud. Cotor Andrei — Universitatea „Babeș-Bolyai" Cluj-Napoca

Subtask-urile 1, 2 și 3

Pentru date de intrare suficient de mici, putem număra toate cazurile posibile, le păstrăm pe cele cu suma lungimilor intervalelor minimă ce au suma K și numărăm câte astfel de intervale exsită.

Pentru fiecare interval este nevoie de fixarea celor două capete, deci se va obține un algoritm cu complexitate $O(N^{2S})$.

Această soluție obține 20 de puncte.

Subtask-ul 4

Pentru cazul în care avem un singur interval, putem să calculăm pentru fiecare capăt stânga al intervalului unde se va afla capătul dreapta astfel încât suma elementelor să fie K.

Pentru a realiza acest lucru se poate utiliza un algoritm de tip *two pointers* în complexitate temporală $O(N)$.

Acest subtask obține 10 puncte.

Subtask-ul 5

Pentru cazul în care avem maxim două intervale, putem să procedăm astfel:

- setăm capetele celui de-al doilea interval pe care il notăm cu $(i, j), i \leq j$
- dacă notăm suma elementelor de pe acest interval cu s, atunci tot ce mai trebuie să facem este să numărăm câte intervale $(k, l), k \leq l < i$ de lungime minimă au suma $K - s$.
- pentru a putea calcula acest lucru vom crea un tablou unidimensional fr_x = numărul de intervale de sumă x de lungime minimă. Acum răspunsul pentru întrebarea de la pasul anterior se va afla în fr_{K-s}.
- când trecem de la i la $i + 1$ cu capătul dreapta al intervalului al doilea, va trebui să adăugăm în fr toate intervalele nou apărute, adică toate intervalele de forma $(k, i), k \leq i$ astfel încât suma pe interval să fie $\leq K$.
- când calculăm răspunsul pentru un pas vom actualiza soluția cea mai bună (de lungime minimă si câte astfel de soluții există).

Această soluție are complexitate temporală de $O(N^2)$.

Soluție oficială

Pentru a rezolva problema vom folosi tehnica *programării dinamice*.

Pentru a găsi o astfel de soluție va trebui să observăm care sunt parametrii de care depinde o posibilă soluție:

- care sunt elementele care s-au luat în considerare pentru soluția parțială;
- câte intervale am folosit pentru a crea soluția si care este starea ultimului interval (daca este încă deschis sau dacă îl considerăm terminat);

- care este suma elementelor soluției.

Astfel, vom definii următorul tablou în care vom stoca atât lungimea minimă a unei soluții (*stare.lungime*) cât si numărul de astfel de soluții (*stare.cnt*) sub forma unei perechi de numere $dp_{i,j,k,l}$ ce reprezintă lungimea minimă a unei soluții și numărul de de astfel de soluții considerând doar elementele de pe pozițiile de la 1 la i ce au suma k și sunt împărțite în j intervale. Variabila l poate lua valorile 0 sau 1 și va reține dacă ultimul interval din cele j s-a terminat (0) sau încă il putem extinde (1).

Starea inițială a dinamicii este $dp_{0,0,0,0} = (0, 0)$, adică dacă nu considerăm niciun element, niciun interval, suma fiind 0, există 0 soluții, lungimea celei mai bune astfel de soluții fiind 0. Toate celelalte stări din matrice vor fi setate cu $(\infty, 0)$ pentru a semnala că nu există nicio astfel de soluție găsită încă.

Pentru a calcula răspunsul reuniunii a două stări din dinamică într-un mod simplu vom definii următoarea funcție:

```
1: procedure COMBINE(stare_1, stare_2)          ▷ returnează rezultatul combinării stărilor
2:     if stare_1.lungime < stare_2.lugime then
3:         return stare_1
4:     else if stare_1.lungime > stare_2.lugime then
5:         return stare_2
6:     else
7:         return (stare_1.lungime, (stare_1.cnt + stare_2.cnt) mod 10^9 + 7)
8:     end if
9: end procedure
```

În continuare vom analiza recurența acestei dinamici:

- dacă se închide intervalul curent:

$$dp_{i,j,k,0} = \text{COMBINE}(dp_{i,j,k,0}, dp_{i-1,j,k,1})$$

- dacă se păstrează închis intervalul curent:

$$dp_{i,j,k,0} = \text{COMBINE}(dp_{i,j,k,0}, dp_{i-1,j,k,0}).$$

- dacă se continuă cu intervalul curent (se adaugă 1 element la lungime):

$$dp_{i,j,k,1} = \text{COMBINE}(dp_{i,j,k,1}, (dp_{i-1,j,k-v_i,0}.lungime + 1, dp_{i-1,j,k-v_i,0}.cnt)).$$

- dacă se deschide un nou interval (se adaugă 1 element la lungime):

$$dp_{i,j,k,1} = \text{COMBINE}(dp_{i,j,k,1}, (dp_{i-1,j-1,k-v_i,0}.lungime + 1, dp_{i-1,j-1,k-v_i,0}.cnt))$$
$$dp_{i,j,k,1} = \text{COMBINE}(dp_{i,j,k,1}, (dp_{i-1,j,k-v_i,0}.lungime + 1, dp_{i-1,j,k-v_i,0}.cnt)).$$

Răspunsul va fi reprezentat de reuniunea soluțiilor din stările de forma $dp_{N,j,K,l}$ pentru orice $1 \leq j \leq S$ și $l \in \{0, 1\}$, întrucât vrem să luăm în considerare toate cele N elemente, iar suma soluției să fie exact K.

Complexitatea temporală a acestei soluții este $O(NSK)$ și obține scorul maxim.

Ne mai rămâne o singură problemă de rezolvat. Complexitatea spațială a acestei soluții este $O(NSK)$, care pentru testele maximale va depăși limita de memorie pentru problemă. Însă putem reduce memoria făcând următoarea observație:

Toate stările $dp_{i,j,k,l}$, pot fi calculate utilizând doar stări de forma $dp_{i-1,j',k',l'}$, deci putem să reținem doar ultimele două *rânduri* din matrice pentru a calcula răspunsul. Astfel, complexitatea spațială este redusă la $O(SK)$.

5.7 Clasele XI–XII

5.7.A Problema Lupușor si Mielu

Propusă de: asist. doctorand Andrei-Costin Constantinescu — ETH Zűrich

Primul pas în rezolvarea problemei constă în înțelegerea mai îndeaproape a jocului jucat de către Lupușor și Mielu. Notăm cu $\mathcal{M} \neq \emptyset$ mulțimea indicilor cărților rămase în joc după prima mutare a lui Mielu. De asemenea, notăm cu $a_{\mathcal{M}} = \{a_i \mid i \in M\}$ și $b_{\mathcal{M}} = \{b_i \mid i \in M\}$ mulțimile valorilor a și, respectiv, b, ale cărților rămase în joc. Pentru o mulțime nevidă A, notăm cu $\min A$ și $\max A$ valoarea minimă și, respectiv, maximă, din mulțimea A. În acest caz, avem că:

Observatia 4. *Mielu câștigă jocul dacă și numai dacă oricare ar fi $i, j \in \mathcal{M}$ avem $a_i < b_j$. Cu alte cuvinte, dacă și numai daca* $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$.

Demonstrație. Cum Lupușor este viclean, acesta va face orice îi stă în putință să câștige odată ce Mielu a fixat mulțimea $\mathcal{M}$. Dacă există doi indici $i, j \in \mathcal{M}$ astfel încât $a_i > b_j$, atunci Lupușor îi poate alege și să câștige, Mielu pierzând. Observați că în acest caz avem $\max a_{\mathcal{M}} > \min b_{\mathcal{M}}$. Altfel, dacă $a_i < b_j$ oricare ar fi $i, j \in \mathcal{M}$, Lupușor va pierde, indiferent ce valori $i, j \in \mathcal{M}$ alege, deci Mielu câștigă. Observați că acest lucru se va întampla dacă și numai dacă $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$. □

Știind aceasta, putem face o a doua observație, destul de naturală la acest moment:

Observatia 5. *Dacă Mielu alege $\mathcal{M}$ astfel încât în joc să rămână o carte $i \in \mathcal{M}$ cu $a_i > b_i$, atunci Mielu pierde.*

Demonstrație. În acest caz Lupușor poate alege $i = j$ și să obțină $a_i > b_j$. □

Prin urmare, cărțile cu $a_i > b_i$ sunt în mare parte imateriale pentru rezolvarea problemei, ele trebuind mereu să fie eliminate pentru ca Mielu să aibă o șansă reală de a câștiga. Pentru simplitate, vom presupune în cele ce urmează $a_i < b_i$ pentru toate cărțile, atât la început, cât și după fiecare modificare dispusă de director. Tratarea cărților care au $a_i > b_i$ necesită numai modificări minore.

Date fiind acestea, cerința 1 a problemei cere să aflăm cardinalul maxim al unei mulțimi $\mathcal{M} \neq \emptyset$ cu proprietatea că $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$ (sau -1 dacă nu există), iar cerința 2 cere numărul de astfel de mulțimi. Pentru ambele cerințe poate fi util să privim perechile $(a_i, b_i)_{1 \leq i \leq N}$ drept intervale pe axa $\mathbb{R}$, însă acest lucru nu este neapărat necesar pentru rezolvarea problemei. Totuși, pe aceste considerente, vom numi numerele $(a_i)_{1 \leq i \leq N}$ *capete stânga*, iar numerele $(b_i)_{1 \leq i \leq N}$ *capete dreapta*, deoarece ele sunt capetele stânga și, respectiv, dreapta ale intervalelor închise $[a_i, b_i]_{1 \leq i \leq N}$. Amintim că $a_i < b_i$, deci intervalele sunt bine definite.

Cerința 1

Vom face următoarea observație, argumentabil mai puțin directă de această dată:

Observatia 6. *Pentru o mulțime M avem că $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$ dacă și numai dacă intervalele închise $[a_i, b_i]_{i \in \mathcal{M}}$ au intersecția nevidă.*

Demonstrație. Observația este o consecință directă a algoritmului de calcul a intersecției unor intervale pe axa $\mathbb{R}$. Mai exact, algoritmul calculează capătul stâng al intersecției ca fiind $\max a_{\mathcal{M}}$, iar capătul drept ca fiind $\min b_{\mathcal{M}}$. Excepție face cazul când $\max a_{\mathcal{M}} > \min b_{\mathcal{M}}$, caz în care intersecția este vidă. Amintim că în cele de mai sus capete de intervale sunt disjuncte două câte două. □

O altă observație utilă este următoarea:

Observatia 7. *O mulțime de intervale $[a_i, b_i]_{i \in \mathcal{M}}$ se intersectează dacă și numai dacă există un număr întreg x astfel încat $a_i \leq x \leq b_i$ oricare ar fi $i \in \mathcal{M}$.*

Demonstrație. Definitia intersecției este echivalentă cu existența unui număr real x care respecta proprietatea. Deoarece capetele intervalelor sunt numere intregi, dacă numarul x respectă proprietatea, atunci și numărul $\lfloor x \rfloor$ respectă proprietatea, de unde rezultă concluzia. □

Definim șirul $s_x = |\{i : 1 \leq i \leq N \text{ si } a_i \leq x \leq b_i\}|$, reprezentând pentru fiecare numar x câte intervale din cele N conțin x. Observațiile 6 și 5.7.A ne dau acum următoarea observație:

Observatia 8. *Mărimea celei mai mari mulțimi $\mathcal{M}$ astfel încat Mielu să câștige este dată de maximul din șirul s. Mai exact, dacă maximul s_x se atinge pentru o valoare x, atunci mulțimea $\mathcal{M}$ este determinată de intervalele ce conțin valoarea x.*

Astfel, problema se poate rezolva în următorul mod:

1. Se calculează șirul s înainte de orice modificari. Acest lucru se poate face în timp liniar folosind Smenul lui Mars.
2. Se calculează maximul din șirul s pentru a obține răspunsul înainte de modificări.
3. Pentru fiecare modificare care elimină un interval $[a, b]$ și adaugă în loc un interval $[c, d]$:
   (a) Se scade 1 din valorile $s_a, s_{a+1}, \ldots, s_b$.
   (b) Se adună 1 la valorile $s_c, s_{c+1}, \ldots, s_d$.
   (c) Se recalculează maximul din șirul s pentru a răspunde din nou la întrebare.

Aceste operații se pot implementa eficient folosind arbori de intervale,[2] folosind tehnica „lazy update”.[3] Arborii de intervale se pot implementa atăt recursiv cât și iterativ. Complexitatea soluției este $O((N + M) \log (N + M))$ și este suficientă pentru a rezolva toate testele de evaluare cu $C = 1$.

[2] www.infoarena.ro/problema/arbint, infoarena.ro/arbori-de-intervale
[3] https://codeforces.com/blog/entry/18051

Cerința 2

Și pentru această cerință vom privi cărțile drept intervale $[a_i, b_i]_{1 \le i \le N}$. Observațiile făcute anterior conduc la următoarea:

Observatia 9. *Numărul de mulțimi $\mathcal{M}$ pentru care Mielu castigă este determinat de numărul de submulțimi nevide ale lui $\{1, 2, \ldots, N\}$ care au intersecția intervalelor asociate nevidă.*

Din păcate, nu este suficient să numărăm pentru fiecare număr întreg x câte submulțimi de intervale îl conțin pe x, deoarece pentru o submulțime pot exista mai multe valori x conținute în intersecția intervalelor. Din acest motiv, este nevoie de un truc: dorim ca pentru fiecare valoare x să numărăm câte submulțimi de intervale îl conțin pe x și au intersecția un interval de forma $[x, x']$, unde $x < x'$ si x' poate fi arbitrar. Cu alte cuvinte, dorim să numărăm pentru fiecare x câte submulțimi de intervale respectă $x = \max a_{\mathcal{M}} < \min b_{\mathcal{M}}$. Remarcăm următoarele:

- Dacă nu există nici un interval cu capatul stânga egal cu x, atunci răspunsul pentru acest x este 0.
- Dacă un astfel de interval există, fie indicele său $\mathcal{I}$, atunci el este unic, și trebuie obligatoriu să facă parte din submulțimea $\mathcal{M}$ pentru ca să avem $x = \max a_{\mathcal{M}} < \min b_{\mathcal{M}}$. Acest lucru rezultă deoarece capetele intervalelor sunt distincte două cate două.
- Odată presupus că $\mathcal{I} \in \mathcal{M}$, condiția $x = \max a_{\mathcal{M}} < \min b_{\mathcal{M}}$ se poate rescrie astfel: oricare ar fi $i \in \mathcal{M} \setminus \{\mathcal{I}\}$ trebuie să avem $a_i < a_{\mathcal{I}} < b_i$, unde amintim că $a_{\mathcal{I}} = x$. Cu alte cuvinte, în afară de intervalul $\mathcal{I}$, celelalte intervale din $\mathcal{M}$ trebuie să conțină valoarea x.
- Notând cu S_x mulțimea intervalelor ce respectă $a_i < x < b_i$ observăm că fiecare astfel de interval poate fi luat în multimea $\mathcal{M}$ independent de celelalte, deci în total sunt $2^{|S_x|-1} = 2^{s_x-1}$ modalități de a alege submulțimea $\mathcal{M}$ astfel încat $\max a_{\mathcal{M}} = x$. Valoarea -1 din exponent corespunde cu tratementul special aplicat intervalului $\mathcal{I}$.

Date fiind acestea, o primă soluție, în complexitate pătratică, este prezentată în cele ce urmează. Soluția, implementată corect, ar trebui să obțină toate mai puțin ultimele 20 de puncte asociate cerinței. După fiecare modificare (și înainte de modificări) se procedează astfel:

- Se calculează valorile s_x folosind Șmenul lui Mars.
- Pentru fiecare valoare x se calculează g_x ca fiind 1 dacă există un (unic) interval $[a_i, b_i]$ respectând $a_i = x$, și 0 altfel.
- Răspunsul la întrebare se calculează ca fiind $\sum_x g_x 2^{s_x-1} = \frac{1}{2} \sum_x g_x 2^{s_x}$, în timp liniar.

Pentru a optimiza soluția, în primul rând observăm că valorile g_x se schimbă în maxim două locuri după fiecare modificare, deci ele pot fi menținute cu efort constant la fiecare modificare. De asemenea, este suficient să lucrăm cu valorile $p_x = 2^{s_x}$, după cum se va vedea în cele ce urmează, lucru care duce la formula mai simplă de calcul a răspunsului $\frac{1}{2} \sum_x g_x p_x$.

Date fiind acestea, putem reformula soluția anterioară în felul următor:

1. Se calculează șirul p_x înainte de orice modificări, în timp liniar. Acest lucru se poate face tot cu Șmenul lui Mars dacă schimbăm operațiile de $+1$ și -1 cu $\times 2$ și $/2$. Amintim că operația de împărțire necesită cunoașterea conceptului de invers modular.[4]
2. Se calculează răspunsul înainte de orice modificări ca $\frac{1}{2} \sum_x g_x p_x$.
3. Pentru fiecare modificare care elimină un interval $[a, b]$ și adaugă în loc un interval $[c, d]$:

[4]Inversul modular al lui 2 este 500 000 004.

(a) Se setează pe 0 valoarea g_a și pe 1 valoarea g_c.
(b) Se împart la 2 valorile $p_a, p_{a+1}, \ldots, p_b$.
(c) Se înmulțesc cu 2 valorile $p_c, p_{c+1}, \ldots, p_d$.
(d) Se recalculează răspunsul ca fiind $\frac{1}{2} \sum_x g_x p_x$.

Și în acest caz operațiile de mai sus se pretează la un atac cu arbori de intervale folosind tehnica lazy update. Complexitatea timp este,și de această dată, $O((N+M)\log(N+M))$, suficientă pentru punctaj maxim pe testele de evaluare cu $C = 2$. Un detaliu de implementare inedit necesar funcționării acestei abordări este menținerea unei variabile reprezentând suma $\sum_{x=\ell}^{r} g_x p_x$ pentru fiecare nod reprezentând un interval $[\ell, r]$ al arborelui de intervale.

5.7.B Problema Schema și Investițiile

Propusă de: student Matei Tinca — Vrije Universiteit Amsterdam

Subtask 1 — Ordinea proiectelor dată în fișierul de intrare este cea optimă

Putem simula efectiv investițiile pe care le face Dorel. Le parcurgem pe rând, iar când a_i este mai mic sau egal decât G, putem să scădem din G pe a_i.

Subtask 2 — $N \leq 7$

Putem aplica un algoritm de generat toate permutările posibile. Astfel, pentru fiecare permutare aplicăm algoritmul descris în subtask-ul 1 și luăm soluția maximă. Pentru această soluție este important în implementare cum se generează permutările. Acestea trebuie generate în $O(N!)$, nu în $O(N^N)$

Subtask 3 — Șirul a este format din două valori distincte care se repetă, în orice ordine

Putem să ne fixăm în câte proiecte investește Dorel din prima valoare și în câte proiecte investește Dorel din a doua valoare. După ce scădem din G suma investită, trebuie să verificăm să nu investim cu banii rămași în proiectele rămase. Practic, trebuie să vedem dacă $G_{rămas} < x$ dacă nu a investit în toate proiectele cu valoare x și dacă $G_{rămas} < y$ dacă nu a investit în toate proiectele cu valoare y.

Subtask 4 — $N \leq 80$

Din subtaskul anterior, putem face următoarea observație: fie m minimul dintre a_i specifice pentru proiectele în care nu investim. Atunci răspunsul va fi mai mic decât m. Presupunând că această condiție nu se respectă, atunci noi vom investi automat în unul din proiectele în care am ales să nu investim și ajungem la o contradicție.

Astfel, putem să folosim următorul raționament pentru a afla răspunsul: ne fixăm indicele proiectului cu costul cel mai mic în care nu investim. Astfel, știm că vom investi în toate proiectele cu cost mai mic decât cel fixat, iar din proiectele cu cost mai mare, noi trebuie să alegem pentru fiecare dacă investim în el sau nu, astfel încât la sfârșit să rămânem cu o sumă de bani mai mare sau egală cu 0 și mai mică strict decât costul proiectului fixat. Suma rămasă la final va fi un posibil răspuns. Astfel, noi vom alege maximul dintre toate răspunsurile posibile.

Astfel, o soluție pe care o putem face este să fixăm fiecare proiect ca fiind minim. Din G scădem toate proiectele cu cost mai mic. Pentru cele mai mari, putem folosi tehnica programării dinamice pentru problema rucsacului. Dinamica ne spune pentru fiecare sumă S_{mari} dacă poate fi obținută ca suma unei submulțimi de elemente. Astfel, putem itera prin toate valorile lui S_{mari} și să verificăm dacă se respectă condiția $G - S_{mici} - S_{mari} < minim$.

În soluția acestei probleme, fixăm fiecare proiect ca fiind minim, și după aplicăm rucsac pe proiectele cu cost mai mare. Complexitatea acestei soluții va fi $O(NG)$.

Un caz particular la care trebuie să avem grijă este faptul că uneori nu putem fixa minimul acela. Asta înseamnă că investim automat în toate proiectele, și trebuie să afișăm $G - a_1 - a_2 - \ldots - a_N$.

Subtask 5 — $N \leq 2000$ și $0 \leq a_1 + a_2 + \ldots + a_N \leq 150$

Putem folosi un algoritm de backtracking similar cu soluția de la subtask-ul 3. Pentru fiecare valoare distinctă, alegem în câte proiecte investim, iar la sfârșit verificăm dacă suma rămasă este mai mică decât proiectele în care nu investim.

Complexitatea acestei soluții va fi $O((f_1+1)(f_2+1)(f_3+1)\ldots(f_N+1))$ unde f_i reprezintă câte proiecte există care au costul i. Putem observa că acest produs este suficient de mic pentru a intra în limita de timp.

Subtask 6 — Fără restricții suplimentare

Observăm faptul că la soluția de la subtask-ul 4 este ineficient faptul că aplicăm rucsac de N ori. Putem sorta elementele descrescător și să aplicăm rucsacul o singură dată. Fixăm indicele i ca fiind minimul în care nu investim. Astfel, toate elementele din stânga lui i sunt mai mari, deoarece am sortat vectorul descrescător. Așa că dacă ne facem dinamica pe prefixe de elemente, noi am calculat pentru fiecare i toate aranjamentele posibile ale elementelor mai mari decât el.

Complexitatea acestei soluții va fi $O(NG + N \log N)$.

Bonus

Aflați profitul maxim pe care îl poate obține Dorel dacă poate să își aleagă o submulțime de proiecte pe care le poate ordona cum vrea. Restricțiile problemei sunt cele din problema originală.

5.7.C Problema Regate si Alianțe

Propusă de: student Stelian Chichirim — Universitatea din București

În această problemă se dă un graf cu N noduri și M muchii bidirecționale, unde fiecare muchie i, $1 \leq i \leq M$, are un cost c_i, iar fiecare nod j, $1 \leq j \leq N$, are un cost r_j. Costul ca un nod X să intre într-o alianță cu un nod Y, este minimul dintre r_X și, costul c_i al muchiei i, unde $1 \leq i \leq M$, astfel încât dacă muchia i este șterssș din graf, nodul X nu mai este în aceeași componentă conexă cu nodul Y.

În alte cuvinte, $Cost(X, Y)$, este minimul dintre r_X și costul muchiilor critice care ne despart nodul X de nodul Y când muchia respectivă este ștearsă din graf.

Primul pas în rezolvarea problemei este să o reducem de la a o rezolva pe graf în a o rezolva pe arbore. Vom identifica fiecare muchie critică din graf (i.e. o muchie care, dacă este ștearsă din graf, numărul de componente conexe ale grafului rezultat crește cu unu). După ce am identificat aceste muchii critice, le ștergem din graf, și calculăm componentele conexe din acest graf rezultat. Apoi, compresăm fiecare componentă conexă într-un nod și fixăm ponderea p al acestui nod ca fiind egală cu numărul de noduri din componenta conexă respectivă. Vom trage o muchie i de cost c_i între componenta compresată A si componenta compresată B, dacă exista un nod a din componenta conexă reprezentată de A și un nod b din componenta conexă reprezentată de B, astfel încât muchia i unește nodurile a și b.

Graful indus de aceste componente conexe compresate este *arbore*. Acest arbore se mai numește și *bridge tree* și poate fi construit în complexitate timp $O(N + M)$.

Am redus problema la: Se dă un arbore cu N noduri, unde fiecare muchie i, $1 \leq i \leq N - 1$, are un cost c_i și fiecare nod j, $1 \leq j \leq N$, are o pondere p_j. Notând cu $m_{x,y}$ valoarea minimă a oricărei muchii pe lanțul simplu din arbore care unește x cu y, pentru fiecare nod k din graful inițial al problemei, astfel încât x_k este nodul compresat din arbore din care k face parte, costul ca acesta să fie într-o alianță perfectă este:

$$\left(\sum_{nod=1}^{N} p_{nod} \min(r_k, m_{nod,x_k}) \right) - r_k.$$

Următorul pas al problemei este să scăpăm de valorile de pe noduri, respectiv șirul r. Pentru aceasta, în graful inițial, pentru fiecare nod i, adăugăm nodul i' cu ponderea p egală cu 0, și muchia (i, i') de cost r_i. Vom face arborele compresat pe acest nou graf, iar componentele conexe compresate formate din nodurile adăugate, i', vor avea ponderea p egală cu 0. Utilizând acest arbore, pentru fiecare nod k din graful inițial al problemei, astfel încât k' este nodul compresat din arbore ce conține nodul k' adăugat anterior, costul ca acesta să fie într-o alianță perfectă este:

$$\left(\sum_{nod=1}^{N} p_{nod} \min(m_{nod,k'}) \right) - r_k.$$

Pentru a calcula această sumă pentru fiecare nod din arbore, vom sorta muchiile arborelui în ordine descrescătoare. Apoi, vom menține o pădure de mulțimi disjuncte (DSU) pentru nodurile arborelui și vom face sume parțiale („șmenul lui Mars") pe arborele pădurilor.

Deci, parcurgem muchiile în ordine descrescătoare, fie c costul muchiei curente, și când trebuie să unim nodul x de nodul y, dacă acestea nu sunt în aceeași componenta în DSU, luăm rădăcina fiecărei componente. Vom nota cu X rădăcina componentei din care face parte x și cu Y rădăcina componentei din care face parte y.

Putem observa că, pentru toate nodurile din componenta lui X trebuie să adăugăm valoarea $c \cdot cnt_Y$, iar, pentru toate nodurile din componenta lui Y trebuie să adăugăm valoarea $c \cdot cnt_X$; unde cnt_X este suma ponderilor nodurilor din componenta lui X. Pentru a face asta, fără pierderea generalității, presupunem că Y va deveni tatăl lui X în DSU. Adăugăm valoarea $c \cdot cnt_X$ în Y, iar în X, adăugăm $c \cdot cnt_Y$ și scădem valoarea curentă din Y. La final, propagăm fiecare valoare din arborele DSU în subarborele acestuia. Răspunsul pentru nodul i, va fi valoarea din arborele DSU a nodului i' minus r_i.

Complexitatea de timp finală este $O((N+M)\log(N+M))$, dată de sortarea valorilor muchiilor. În afară de sortare, complexitatea timp este $O((N+M)\log^*(N+M))$.

Subtask 1 — $N \leq 2000$ și $M = N - 1$, iar regatele și muchiile vor forma un lanț, în ordinea $1, 2, \ldots, N$

Putem observa că în acest caz avem un arbore, mai exact, doar un lanț (în formula de mai sus $p_{nod} = 1$, pentru orice nod).

Pentru a calcula suma descrisă mai sus, ne putem fixa un nod și iterăm prin toate subsecvențele care încep in acest nod și toate subsecvențele care se termină în acest nod, menținând în același timp minimul valorilor muchiilor din subsecvență. Complexitatea de timp finală este $O(NM)$.

Subtask 2 — $N \leq 200$ și $M \leq 400$

Pentru acest subtask, ne putem fixa fiecare nod și fiecare muchie. Ștergem muchia fixată și actualizăm răspunsul pentru nodul fixat și orice alt nod care nu mai este în aceeași componentă conexă cu nodul fixat în urma ștergerii muchiei. Complexitatea de timp finală este $O(NM(N+M))$.

Subtask 3 — $N \leq 2000$ și $M \leq 2000$

Pentru acest subtask, putem calcula arborele compresat în complexitate timp $O(N(N+M))$. Apoi, ne putem fixa fiecare nod ca fiind rădăcina arborelui si calculăm suma de mai sus. Complexitatea de timp finală este $O(N(N+M))$.

Subtask 4 — Toate numerele din șirul c sunt egale

Deoarece costul muchiilor este egal, este de ajuns să calculăm arborele compresat. Pentru un nod i, suntem obligați să luăm costul r_i pentru nodurile care fac parte din aceeași componentă în arborele compresat, iar pentru celelalte noduri j, $Cost(i,j) = min(r_i, c)$, unde c este costul muchiilor. Complexitate de timp finală este $O(N+M)$.

Subtask 5 — $M = N - 1$

Pentru acest subtask, putem face soluția generală a problemei, doar că nu mai este nevoie să calculăm *bridge tree-ul* grafului.

Subtask 6: Fără restricții suplimentare

Descrierea solutiei pe cazul general este prezentată mai sus.

Capitolul 6 Proba de Baraj

6.1 Juniori

6.1.A Problema Autostradă

Propusă de: Stud. Theodor-Gabriel Tulbă-Lecu — Universitatea Politehnica București
Stud. Ioan-Cristian Pop — Universitatea Politehnica București

Problema poate fi împărțită în două subprobleme: generarea tabloului bidimensional și numărarea tipurilor de intersecții.

Generarea tabloului bidimensional

Prin generarea tabloului bidimensional ne referim la identificarea suprafețelor ce trebuiesc asfaltate.

Fie a tabloul bidimensional care va indica suprafețele ce trebuiesc asfaltate. Scopul nostru este ca generarea să fie realizată într-un mod ce facilitează numărarea tipurilor de intersecții.

Codificarea tipurilor de celule Pentru a reține pentru fiecare celulă de ce tip este, putem asocia o anumită valoare pentru fiecare astfel de celulă.

O modalitate de codificare a tipului este ca pentru fiecare direcție față de centrul unei celule să reținem dacă traseul dronei trece prin celulă pe acea direcție utilizând puteri de 2.

Astfel, vom reține pentru celula (i, j) valorile:

- 1 — dacă traseul dronei trece prin nord,
- 2 — dacă traseul dronei trece prin est,
- 4 — dacă traseul dronei trece prin sud,
- 8 — dacă traseul dronei trece prin vest.

La final, vom putea număra câte intersecții $+$ există numărând câte valori din a au exact 4 biți setați cu valoarea 1, câte intersecții $\top$ există numârând câte intersecții au 3 biți setati. Celelalte valori nenule vor reprezenta fie cotituri, fie linii drepte și vor fi numărate ca celule simple.

Dublarea dimensiunilor tabloului Observăm că în soluția anterioară procesul de codificare este destul de complicat deoarece dacă două celule se învecinează, atunci traseul nu trece neapărat de la o celulă la cealaltă.

Putem totuși să evităm acest caz particular prin următoarea operație: o celulă de coordonate (i, j), se translatează în coordonatele $(2i-1, 2j-1)$. Această operație de *dublare* a tabloului, va crea spații goale intre oricare două celule din tabloul original, aceste spații goale sunt vizitate de dronă doar dacă drona inițial a trecut de la o celulă la cealaltă. Observăm ca celulele din tabloul original se vor afla după trasformare pe coordonate cu indicele liniei, respectiv al coloanei, numere impare.

Astfel, după această trasformare, pentru a determina tipul de celulă din tabloul original este suficient, să numărăm câți vecini cu valori nenule are.

Reunirea intervalelor traseului Soluțiile precedente pot fi rezolvate în complexități diferite, cea mai simplă fiind parcurgerea iterativă a tuturor pozițiilor. Această metodă de parcurgere ar avea o complexitate $O(KN)$ și obține aproximativ 47 de puncte.

Putem obține o complexitate mai bună făcând următoarea observație, o mutare a dronei crează un interval pe linia, respectiv coloana pe care aceasta se deplasează. Astfel, putem reține pentru fiecare linie și coloană toate intervalele de pe aceasta, le sortăm, iar mai apoi le reunim.

Această soluție poate fi implementată in $O(N^2 + K \log K)$ și obține aproximativ 90 de puncte.

Șmenul lui Mars (Difference Array) O îmbunătățire a soluției precedente este folosirea șmenului lui Mars[1] pentru a face reunirea intervalelor.

Astfel fiecare dintre cele K operații poate fi efectuată în $O(1)$, iar complexitatea finală a soluției va deveni $O(N^2 + K)$.

Această soluție obține 100 de puncte, indiferent de modul de codificare folosit.

[1] https://infoarena.ro/multe-smenuri-de-programare-in-cc-si-nu-numai

6.1.B Problema Bug

Propusă de: Prof. Emanuela Cerchez — Colegiul Național „Emil Racoviță" Iași
Radu Voroneanu — Google Zürich

Descrierea algoritmului

Vom descrie un algoritm constructiv de determinare a rezultatului.

Vom citi cifrele numărului N şi le vom plasa într-un vector a cu lg elemente.

Vom denumi *bloc* o secvenţă de lungime minimă de cifre din a care conţine toate cifrele de la $\{0, 1, \ldots, 9\}$.

Vom parcurge a de la final către început şi vom identifica blocurile.

Pentru aceasta vom utiliza un vector caracteristic uz_{10}: $uz_i = 1$, dacă cifra i apare în secvenţa curentă, respectiv 0 în caz contrar.

Când vectorul uz conține 10 valori egale cu 1, am identificat un bloc, îl numărăm şi reţinem poziţia sa de început în vectorul *inc*.

Apoi resetăm uz (adică îl reiniţializăm cu 0) pentru a ne pregăti pentru construcţia următorului bloc.

După această parcurgere, am partiționat vectorul a în *nrbloc* blocuri. La începutul lui a este posibil să rămână câteva cifre care nu formează un bloc (notăm acest prefix cu P).

$$a = PB_{nrbloc}B_{nrbloc-1} \ldots B_1$$

Să determinăm c, cea mai mică cifră nenulă care nu apare în prefixul P (1 dacă P este vid).

Toate numerele mai mici sau egale cu

$$\overline{(c-1)\ \underbrace{99\ldots9}_{nrbloc \text{ de } 9}}$$

pot fi obţinute. Cel mai mic număr care nu poate fi obţinut (*rez*) va începe cu c. Pentru a determina celelalte *nrbloc* cifre din *rez*, parcurgem blocurile de la stânga la dreapta şi determinăm pentru fiecare bloc poziţia pe care apare ultima cifră plasată în *rez* în blocul respectiv (să notăm această poziţie i). Cifrele din bloc situate după poziţia i evident nu mai formează un bloc. Determinăm cea mai mică cifră care nu apare în blocul curent de la poziţia $i + 1$ la finalul blocului. Această cifră va fi plasată în *rez*.

De exemplu:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	1	3	6	7	8	9	0	1	2	3	4	5	8	6

	15	16	17	18	19	20	21	22	23	24	25	26
a	2	7	4	9	4	5	2	3	0	7	1	0

$B_1 = 86274945230710$ (poziţiile de la 13 la 26 din a)
$B_2 = 6789012345$ (poziţiile de la 3 la 12 din a)
$P = 13$ (poziţiile 1 şi 2 din a)
$nrbloc = 2$

Cea mai mică cifră nenulă care nu apare în P este 2. Cel mai mic număr care nu se poate obţine începe cu 2. În blocul B_2 cifra 2 apare pe poziţia 9. Cea mai mică cifră care nu apare în B_2 de la poziţiile 10 la 12 este 0.

Cifra 0 apare în B_1 pe poziţia 23. Cea mai mică cifră care nu apare în B_1 de la poziţia 24 până la 26 este 2. Rezultatul va fi 202.

Caz special: P nu este bloc, dar conţine toate cifrele nenule. În acest caz orice număr de $nrbloc + 1$ cifre se poate obţine. Ca urmare vom plasa la începutul rezultatului 10 şi continuăm parcurgerea blocurilor în acelaşi mod.

Corectitudinea algoritmului

Vom demonstra în primul rând că numărul *rez* nu se poate obţine.

Fie c prima cifră a lui *rez*. Aceasta nu apare în P, deci convenabil ar fi să utilizăm prima ei apariţie din B_{nrbloc}. Sufixul lui B_{nrbloc} care începe după prima apariţie a lui c nu este bloc, deci nu conţine toate cifrele. Fie acum c cea mai mică cifră care nu apare în acest sufix (următoarea din *rez*). Aceasta trebuie luată obligatoriu din următorul bloc ş.a.m.d. Când ajungem la ultima cifră a lui *rez*, aceasta nu mai există în ultimul sufix (cel al lui B_1) deci clar *rez* nu se poate obţine.

Să demonstrăm că *rez* este cel mai mic număr care nu se poate obţine.

Să presupunem prin reducere la absurd că există $X < rez$ care nu se poate obţine. Dacă X are mai puţine cifre decât *rez*, sigur se poate obţine (iau cifra i din X din blocul B_i). Deci musai ca X să aibă aceeaşi lungime cu *rez*. Mai mult decât atât X trebuie să înceapă cu aceeaşi cifră ca *rez* (în caz că luăm o cifră mai mică numărul se poate obţine pentru că acea cifră mai mică apare în P, apoi celelalte cifre le luăm din blocurile următoare; în caz că luăm o cifră mai mare X nu ar mai fi mai mic decât *rez*). Prin acelaşi raţionament, a doua cifră a lui X trebuie să coincidă cu a doua cifră din *rez* ş.a.m.d.

6.1.C Problema Triprime

Propusă de: Cristian Frâncu — Clubul Nerdvana România

Să enumerăm câteva soluții în ordinea eficienței lor.

Soluția 1 (18 puncte)

Parcurgem numerele X de la A la B și le descompunem în factori primi. Oprim descompunerea cel târziu când divizorul depășește rădăcina pătrată a numărului, sau imediat ce numărul nu mai poate fi triprim, de exemplu dacă găsim un divizor prim la putere mai mare ca unu.

Soluția va depăși timpul pentru $B > 1.5$ milioane. Timpul folosit este între $O\left(\frac{(B-A)\sqrt{B}}{\log B}\right)$ și $O((B-A)\sqrt{B})$, iar memoria folosită este $O(1)$.

Soluția 2 (24 puncte)

Aceeași soluție ca cea anterioară, testând doar divizorii impari.

Soluția va depăși timpul pentru $B > 2.5$ milioane Timpul folosit este între $O\left(\frac{(B-A)\sqrt{B}}{\log B}\right)$ și $O((B-A)\sqrt{B})$, iar memoria folosită este $O(1)$.

Soluția 3 (44 puncte)

Folosim o precalculare a numerelor prime pentru a descompune mai rapid numerele X în factori primi. Numerele triprime au exact trei factori primi. Cel mai mic factor prim fiind 2, rezultă că al doilea factor prim nu poate fi mai mare decât $\sqrt{390\,000\,000/2}$ care este aproximativ 14 000. Vom precalcula numerele prime până la 14 000 folosind ciurul lui Eratostene. La descompunerea unui număr X vom căuta primii doi factori primi printre numerele prime precalculate. Al treilea factor prim poate fi printre numerele precalculate sau nu. Dacă nu îl găsim printre ele îl vom căuta ca la soluția anterioară, printre numerele impare.

Soluția va depăși timpul pentru $B > 4.5$ milioane. Timpul folosit este între $O\left(\frac{(B-A)\sqrt{B}}{\log B}\right)$ și $O((B-A)\sqrt{B})$. Memoria folosită este $O\left(\sqrt{B} + \frac{\sqrt{B}}{\log B}\right)$ adică un vector ciur de circa 14 000 de bytes și un vector de sub 2 000 de numere prime ce ocupă circa 8KB, în total aproximativ 22KB.

Soluția 4 (75 puncte)

Calculăm numărul de divizori primi al tuturor numerelor până la B folosind ciurul lui Eratostene. Memoria ne permite un vector ciur de caractere până la aproximativ 64 milioane. Soluția va depăși, însă, timpul, înainte de această limită.

Odată calculat ciurul numărului de divizori primi reparcurgem acest ciur. În această a doua parcurgere pentru fiecare număr prim P vom anula toate numerele divizibile cu P^2, setând numărul de divizori la 0.

La o a treia parcurgere a ciurului, între A și B vom număra câte numere cu trei divizori primi avem.

Soluția va depăși timpul pentru $B > 35$ milioane. Timpul folosit este $O(B \log B)$. Memoria folosită este $O(B)$, adică aproximativ 35MB la momentul când apare depășirea de timp.

Soluția 5 (85–100 puncte)

Care este numărul prim maxim care poate apărea într-un număr triprim? Alegând 2 și 3 ca cele mai mici două numere prime, al treilea poate fi maxim $\frac{B}{6}$, adică 65 de milioane. Un vector ciur de caractere va încăpea în memoria disponibilă de 64MB, adică 67 108 864 bytes.

Vom calcula numerele prime până la $\frac{B}{6}$. Apoi vom număra în ciur tripleții de numere prime al căror produs este mai mic sau egal cu X, să denumim acest număr N_X. Răspunsul cerut va fi $N_B - N_{A-1}$.

Cum calculăm numărul de numere triprime până la X? Prin metoda cunoscută unora drept 'metoda celor doi pointeri'. Vom folosi trei indici i, j, k în ciur, astfel încât i, j, k să fie prime și produsul lor să nu depășească X. La fiecare avans al lui i la următorul număr prim, vom calcula j ca fiind următorul număr prim după i și apoi vom porni de la următorul număr prim k și vom avansa k până ce produsul $i \times j \times k$ depășește X, numărând câte numere prime T se află între j și k. La acest moment putem adăuga T numere triprime la rezultat. Apoi avansăm j la următorul număr prim și reducem k către primul număr prim care aduce produsul $i \times j \times k$ sub X. Avem grijă să actualizăm T, numărul de numere prime dintre j și k. Apoi adunăm din nou numărul T la rezultat.

Continuăm în acest fel până ce j îl ajunge pe k. Moment la care avansăm i și reluăm calculul de mai sus.

Această metodă poate lua 100p, dar, în funcție de implementare, este posibil să depășească timpul pe unele teste mari.

Timpul utilizat este $O(B \log B)$ datorat numărării numerelor triprime. Memoria utilizată este $O(B)$, adică 61MB.

Soluția 6 (100 puncte)

Procedăm ca la soluția 5, dar calculăm doar numerele prime impare. Putem folosi un ciur al lui Eratostene modificat, care va ocupa jumătate din memorie, adică 30.5MB. Astfel avem loc pentru un vector separat în care vom stoca toate numerele prime din acest ciur. Sunt aproape 4 milioane de numere prime mai mici decât 65 milioane, care vor ocupa circa 16MB.

Pe vectorul de numere prime putem aplica numărarea de numere triprime mai eficient, folosind aceeași metodă de mai sus, a celor doi pointeri.

Timpul utilizat este $O(B \log \log B)$, deoarece numărarea de numere triprime este liniară în B. Memoria utilizată este $O\left(B + \frac{B}{\log B}\right)$, adică circa 30.5MB pentru ciur și încă circa 15.5MB pentru stocarea numerelor prime, aproximativ 46MB

Soluția 7 (100 puncte)

Tot pentru a folosi mai puțină memorie putem proceda ca la soluția 5, dar folosind un vector ciur pe biți. Apoi procedăm ca la soluția 6, colectând separat numerele prime în vederea numărării eficiente a numerelor triprime.

Timp: $O(B \log \log B)$

Memorie: $O(B + \frac{B}{\log B})$ adică circa 7.5MB pentru ciur și 15.5MB pentru numerele prime, total aproximativ 21MB.

Soluția 8 (100 puncte)

Pentru a folosi și mai puțină memorie (ceea ce duce și la scăderea timpului de executare) putem combina soluțiile 6 și 7, calculând un ciur al lui Eratostene doar pentru numere impare și folosind un vector de biți.

Timpul utilizat este $O(B \log\log B)$. Memoria utilizată este $O\left(B + \frac{B}{\log B}\right)$ adică circa 3.5MB pentru ciur și 15.5MB pentru numerele prime, total aproximativ 19MB.

Soluția 9 (100 puncte)

O soluție cu o cu totul altă idee este următoarea: folosim soluția numărul 3 pentru a descompune în factori primi toate numerele de la 1 la 390 milioane. La fiecare 400 000 de numere scriem într-un fișier numărul de numere triprime găsite până acum. La final vom avea sume parțiale ale numerelor triprime până la X, cu X variind din 400 000 în 400 000. Vom obține 975 de astfel de numere. Timpul de executare al acestui program va fi de circa 10 minute.

Apoi, în programul soluție, vom include aceste 975 de numere ca vector preinițializat. Putem calcula acum

$$N_X = \text{numărul de numere triprime mai mici sau egale cu } X$$

astfel: calculând $X/400\,000$ aflăm rapid numărul de numere triprime până la ultimul interval întreg de 400 000. Pentru restul de numere, până la X, vom folosi algoritmul de la soluția 3. Răspunsul la problemă va fi $N_B - N_{A-1}$.

Aceasta este una din cele mai rapide soluții. Timpul folosit este între $O\left(\frac{K\sqrt{B}}{\log B}\right)$ și $O(K\sqrt{B})$, unde am notat cu K lungimea unui interval de precalculare, adică 400 000. Memoria folosită este $O(B/K + \sqrt{B} + \frac{\sqrt{B}}{\log B})$, formată din memoria ocupată de ciurul până la 14 000, vectorul de numere prime până la 14 000, circa 2 000 de elemente întregi și vectorul de sume parțiale de 975 de elemente; în total aproximativ 26KB.

6.2 Seniori

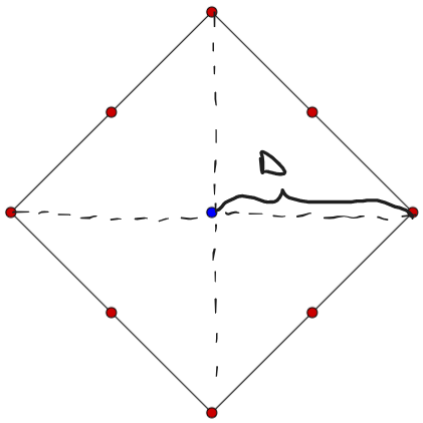
6.2.A Problema 3dist

Propusă de: stud. Bogdan-Ioan Popa — Universitatea din București
asist. doctorand Andrei-Costin Constantinescu — ETH Zürich

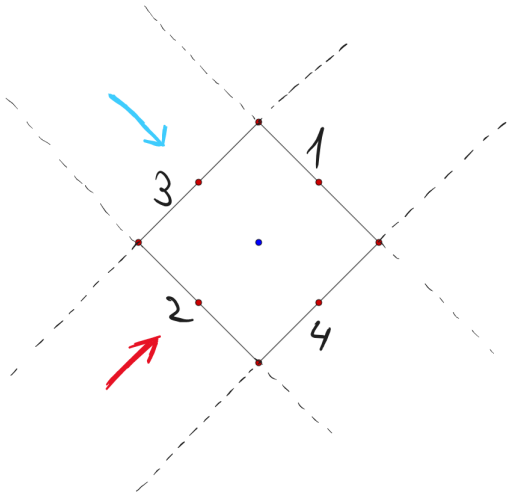
Vom clasifica fiecare punct i în funcție de valoarea $d(i)$. Un triplet (i, j, k) este valid doar dacă $d(i) = d(j) = d(k)$ (i.e. sunt în aceeași clasă), deci are sens să rezolvăm problema independent pentru fiecare clasă de puncte (o mică observație care nu era neapărat necesară pentru rezolvarea problemei este aceea că ne interesează doar punctele care au $d(i)$ par). Pentru a putea calcula $d(i)$ pentru fiecare i de la 1 la N se propune următoarea idee de rezolvare:

- Se sortează punctele în ordine lexicografică;
- Pentru fiecare punct i de la 1 la N vom vrea să determinăm care este distanța minimă către un punct $j < i$. Un astfel de punct va contribui la distanță cu $-X_j$ și $\pm Y_j$ în funcție de cum se compară cu Y_i. Astfel, distanța minimă către un astfel de punct va fi egală cu cea mai mică valoare dintre $X_i + Y_i + \min(-X_j - Y_j \mid j < i, Y_j <= Y_i)$ și $X_i - Y_i + \min(-X_j + Y_j \mid j < i, Y_j > Y_i)$. Pentru a afla acele minime se pot folosi doi arbori de intervale sau doi arbori indexați binar, după o normalizare a valorilor Y;
- Se procedează asemănător pentru a determina distanța minimă către puncte cu $j > i$, de data aceasta parcurgând punctele în ordine inversă.

Acum că avem calculate distanțele d, putem trece la a număra tripletele. După cum am spus și mai sus, vom rezolva independent problema pentru fiecare clasă de puncte. Să presupunem că suntem în clasa D de puncte. Pentru fiecare punct i din această clasă vom determina din câte triplete face parte. Se observă că punctele cu care acesta poate să facă triplet se află pe perimetrul rombului centrat în i care are diagonale de lungime egală cu $2D$ și laturile paralele cu diagonalele reperului cartezian, așa cum se poate vedea mai jos.



În plus, vor exista maxim 8 puncte care se vor afla pe perimetru. Pentru a le putea determina optim, vom *„glisa"* rombul în direcția indicată de săgeata albastră pentru a putea determina punctele de pe laturile 1 și 2, apoi în direcția indicată de săgeata roșie pentru a determina punctele de pe laturile 3 și 4 așa cum se vede în figura de mai jos.



Având determinate cele maxim 8 puncte candidate pentru a face triplet cu punctul i, putem încerca toate combinațiile posibile și să creștem un contor. Răspunsul va fi valoarea contorului împărțită la 3. Complexitatea temporală a soluției este $O(N \log N)$.

6.2.B Problema Piezișă

Propusă de: stud. Ioan Popescu — Universitatea Politehnica din București
stud. George-Alexandru Râpeanu — Universitatea Babeș-Bolyai

O primă observație pe care o putem face este aceea că dacă ne uităm la șirul de xoruri pe prefixe, un query (x, y) se reduce la un query de tipul: găsiți distanța dintre cele cele mai apropriate valori egale din șirul prefixe de sume xor, unde un element din pereche se află la stânga lui x iar alt element se află la dreapta lui y. Formal, șirul de xor pe prefixe este definit prin: $s_i = a_0 \oplus a_1 \oplus a_2 \ldots \oplus a_i$. Cu $\oplus$ s-a notat operația de xor pe biți.

Soluție $O(\textit{distincte} \cdot (N + Q))$ (30 de puncte)

O soluție care rezolvă primele 3 subtaskuri este urmatoarea: Încercăm pentru fiecare valoare *distinctă* din șirul de xor pe prefixe să vedem care ar fi răspunsul optim folosind aceasta valoare pentru fiecare query. Să zicem ca am fixat o astfel de valoare *val*. Se pot precalcula pentru fiecare poziție i din șir, valorile $prev_i$ si $next_i$, unde $prev_i$ reprezintă cea mai mare poziție j din șirul s, cu proprietatea că $j \leq i$ și $s_j = val$. Similar, $next_i$ reprezintă cea mai mica poziție j din șirul s, cu proprietatea că $j \geq i$ și $s_j = val$. Cu ajutorul acestora, segmentul de lungime minimă care incepe și se termină in *val*, care conține un query (x, y) este $[prev_x, next_y]$ (dacă ambele există). Astfel, răspunsul pentru un query va fi lungimea minima a tuturor segmentelor de acest fel.

Soluție $O\left(\frac{QN}{C} + Q \log N + NC \log N\right)$ (70 de puncte)

Ce putem face este să împărțim valorile numerelor în două: cele care au frecvența mai mare decât o valoare aleasă de noi, C, și cele care au frecvența mai mică decât C. Pentru a rezolva un query, putem să ne uităm la valorile care au frecvența mai mare și cele care au frecvența mai mică separat.

Cele mai mari sunt în număr de maxim $\frac{N}{C}$, deci putem afla răspunsul pentru fiecare query în $O\left(\frac{N}{C}\right)$, folosind un algoritm similar cu cel descris in soluția anterioara. Astfel, complexitatea finală în acest caz este $O\left(\frac{QN}{C}\right)$.

Pentru valorile mici, putem să sortăm queryurile crescător după capătul dreapta. Să zicem ca ne aflăm la un query (x, y). Facem notația $next_i = j$ unde $i < y$, și $j \geq y$, iar $s_j = s_i$. Pentru a răspunde la un query, trebuie să aflăm $\min_{i<x}(next_i - i)$. Atunci când trecem la alt query, pentru că avem queryurile sortate după capătul dreapta, y poate doar să crească. Incrementăm capătul dreapta cu 1 până ajunge la y curent. Fiecare iterație ne modifică $next_i$ pentru maxim C poziții. Cum capătul dreapta se modifică de maxim N ori, numărul de modificări este maxim NC. Pentru a face rapid operațiile de update și query, putem folosi un arbore de intervale, complexitatea pentru updateuri și queryuri fiind $O(Q \log N + NC \log N)$.

Alegerea lui C optim este lăsată la latitudinea cititorului.

Soluție $O\left(\frac{QN}{C} + Q\left(B + \frac{N}{B}\right) + NC\right)$ (100 de puncte)

Soluția pentru valorile cu o frecvență mare nu trebuie îmbunătățită, ce ne deranjează pe noi este acel $NC \log N$. Ce putem face mai bine? De C nu prea putem scăpa, de N nici atât, deci $\log N$ este ce ne încurcă pe noi. Avem o structură care răspunde la queryuri și updateuri în $O(\log(N))$, dar numărul de queryuri este mult mai mic decât numărul de updateuri. Ce structură putem folosi care să ne răspundă rapid la queryuri, și mai important, să facă updateurile în $O(1)$? O descompunere în radical!

Pentru simplitate o să avem bucketuri de mărime B. Queryurile sunt simple, au complexitate $O\left(B + \frac{N}{B}\right)$, dar cum facem updateuri în $O(1)$? Pentru a putea face updateuri rapid, trebuie să avem următoarea proprietate adevărată: valorile $next_i$ pot doar să scadă. Pentru a respecta această proprietate, sortăm queryurile descrescător după capătul dreapta.

Alegerea lui B și C optimi este lăsată la latitudinea cititorului.

Soluție $O\left(Q \log Q + QB + \frac{N^2}{B}\right)$ (100 de puncte)

O soluție alternativă poate fi găsită utilizând o modificare a algoritmului lui Mo.[2] Similar ca și în algoritm, vom sorta queryurile prima data după $\left\lfloor \frac{x}{B} \right\rfloor$. În caz de egalitate, vom sorta *descrescător* după y. Este important să facem asta, deoarece, astfel, capătul drept doar va scoate elemente din Mo, o operație care e mai ușoara decât introducerea lor (din motive similare cu cele din soluția anterioară). Acum, pentru capătul stâng, ca să evitam adăugarea elementelor cu capătul stâng, la început de fiecare query el va fi setat la începutul bucketului curent. În momentul în care acesta trebuie ajustat, el va șterge din Mo toate elementele de la începutul bucketului până la $x - 1$, și va ține minte schimbările într-o stivă. Pentru a fi pregătit pentru următorul query, vom da undo folosindu-ne de această stivă.

Soluție $O\left(QB + \left(\frac{N}{B}\right)^2\right)$ (100 de puncte)

O altă soluție care poate să obțină punctaj maxim este următoarea: împărțim șirul s în bucketuri de B elemente. Acum ne propunem să precalculăm $p(i, j)$ = lungimea subsegmentului de lungime minimă care are suma xor egală cu 0, care are capătul stâng într-unul dintre bucketurile $0, 1, \ldots, i$, iar capătul drept într-unul dintre bucketurile $j, j + 1, \ldots$. Asta poate fi realizat relativ simplu. Acum, pentru un query (x, y) avem următoarele cazuri: segmentul optim are unul dintre capete în bucketul lui x sau în bucketul lui y, sau în niciunul. Răspunsul pentru cazul în care segmentul optim nu are capătul nici în bucketul lui x, nici în bucketul lui y este deja acoperit, noi putând să ne uităm direct în $p(bucket(x) - 1, bucket(y) + 1)$. Vom mai analiza doar cazul în care segmentul optim are capătul stâng în bucketul lui x, cazul în care are capătul drept în bucketul lui y fiind analog. Pentru acest caz, vom sorta queryurile descrescător după y, și vom ține pentru pozițiile mai mari decât y-ul curent șirul $first_i$, care ține cea mai mică poziție dintre cele procesate pe care se află valoarea i. Acum, putem doar să iterăm prin pozițiile $bucket(x) \cdot B, \ldots, x$ și să ne folosim de $first_i$ pentru a găsi noile segmente.

[2] https://cp-algorithms.com/data_structures/sqrt_decomposition.html#mos-algorithm

6.2.C Problema Portocal

Propusă de: stud. Alexandra Maria Udriștoiu — Universitatea din București

Vom calcula următoarea programare dinamică pe arbore:

$$d(i,0) = \begin{cases} \text{numărul minim de șiruri formate din valorile de pe lanțuri de} \\ \text{la nodul } i \text{ la nodurile din subarborele lui } i \text{ care sunt mai mici} \\ \text{lexicografic decât șirul } S_{niv_i} \dots S_k \text{ și nici unul dintre șirurile formate} \\ \text{nu este egal cu } S_{niv_i} \dots S_k. \end{cases}$$

$$d(i,1) = \begin{cases} \text{numărul minim de șiruri formate din valorile de pe lanțuri de} \\ \text{la nodul } i \text{ la nodurile din subarborele lui } i \text{ care sunt mai mici} \\ \text{lexicografic decât șirul } S_{niv_i} \dots S_k \text{ și cel puțin unul dintre șirurile} \\ \text{formate nu este egal cu } S_{niv_i} \dots S_k. \end{cases}$$

$$num(i,0) = \begin{cases} \text{numărul de moduri de a completa valorile lipsă din subarborele} \\ \text{nodului } i \text{ pentru care se obține } d(i,0). \end{cases}$$

$$num(i,1) = \begin{cases} \text{numărul de moduri de a completa valorile lipsă din subarborele} \\ \text{nodului } i \text{ pentru care se obține } d(i,1). \end{cases}$$

Cu niv_i s-a notat nivelul nodului i în arbore. Fie w_i numărul de noduri din subarborele lui i și $nval_i$ numărul de noduri din subarbore ce nu au asociate o valoare, excluzând nodul i.

Considerăm cazul general, când $niv_i < K$. Considerăm $fiu_1 \dots fiu_G$ fii nodului i în arbore. Pentru a calcula $d(i,0)$ și $num(i,0)$, vom considera următoarele 3 cazuri:

1. $val_i > S_{niv_i}$. Toate șirurile formate vor fi mai mari lexicografic decât $S_{niv_i}, \dots, S_k$. Așadar, $d(i,0) = 0$ și $num(i,0) = M^{nval_i} \cdot$ *numărul de moduri de a avea* $val_i > S_{niv_i}$.
2. $val_i < S_{niv_i}$. Toate șirurile formate vor fi mai mici lexicografic decât $S_{niv_i}, \dots, S_k$. Așadar, $d(i,0) = w_i$ și $num(i,0) = M^{nval_i} \cdot$ *numărul de moduri de a avea* $val_i < S_{niv_i}$.
3. $val_i = S_{niv_i}$, dar nu vrem să avem niciun șir egal cu $S_{niv_i}, \dots, S_k$. Atunci, $d(i,0) = 1 + d(fiu_1,0) + \dots + d(fiu_G,0)$ și $num(i,0) = num(fiu_1,0) \cdot \dots \cdot num(fiu_G,0)$.

Dintre aceste 3 cazuri, le vom considera doar pe cele pe care le putem obține pentru nodul i (dacă val_i este completată, avem unul singur, iar dacă $val_i = -1$, trebuie să vedem dacă există valori mai mici, respectiv mai mari, decât S_{niv_i}). Dintre valorile $d(i,0)$ obținute, cea finală va fi cea mai mică dintre ele. În cazul în care se obține $d(i,0)$ minim pentru mai multe cazuri, $num(i,0)$ va fi suma numărului de moduri pentru acele cazuri.

Fie $d(i,0/1)$ minim dintre $d(i,0)$ și $d(i,1)$ (nu ne interesează dacă $S_{niv_i} \dots S_k$ apare printre șirurile formate). Similar, $num(i,0/1)$ va fi numărul de moduri de a completa valorile lipsă din subarbore pentru a obține $d(i,0/1)$.

Pentru a calcula $d(i,1)$ trebuie luat în considerare doar cazul când val_i este egal cu S_{niv_i} și $S_{niv_{i+1}} \dots S_k$ apare cel puțin unul dintre fii. Pentru a nu număra un mod de mai multe ori, vom fixa fiu_j primul fiu în care apare șirul. Astfel, $d(i,1)$ va fi minim din

$$1 + d(fiu_1,0) + \dots + d(fiu_{j-1},0) + d(fiu_j,1) + d(fiu_{j+1},0/1) + \dots + d(fiu_G,0/1).$$

Iar $num(i,1)$ va fi sumă pentru valorile j care dau $d(i,1)$ minim, din

$$num(fiu_1,0) + \dots + num(fiu_{j-1},0) + num(fiu_j,1) + num(fiu_{j+1},0/1) + \dots + num(fiu_G,0/1).$$

Pentru a avea complexitate finală $O(N)$, trebuie să ținem sume parțiale pentru a calcula cele două valori de mai sus.

Când $niv_i = K$, pentru a calcula $d(i, 0)$ se consideră doar primele două cazuri. $d(i, 1)$ va fi egal cu $d(fiu_1, 0) + \ldots + d(fiu_G, 0)$ și $num(i, 1) = num(fiu_1, 0) \cdot \ldots \cdot num(fiu_G, 0)$, dacă val_i poate fi S_K.

Dacă $niv_i > K$, putem considera că toate șirurile formate vor fi mai mari, deci $d(i, 0) = 0$ și $num(i, 1)$ va fi numărul de moduri de a completa valorile din subarborele lui i.

Complexitatea finală va rămâne $O(N)$.

Subtaskurile 1 și 4 pot fi rezolvate prin metoda backtracking.

Pentru subtaskul 2, observăm că, după ce am fixat un nod i pentru care vrem să se obțină un șir egal cu S și am completat corespunzător valorile de pe lanțul de la rădăcină la i, pentru ca acest șir să apară cât mai repede în ordine lexicografică, nodurilor rămase fără valori le putem da tuturor valoarea M. Așadar, vom încerca toate posibilitățile de a alege nodul i, vom completa valorile, și vom număra cu ajutorul unei parcurgeri DFS câte șiruri ar fi mai mici decât nodul S pentru această completare, alegând minimul. Complexitatea acestei soluții este $O(N^2)$.

6.2.D Problema „Miyuki vrea să împăturească arborigami”

Prima observație pe care trebuie să o facem este că o operație este validă doar dacă nodurile a_i și b_i împăturite în cadrul operației sunt fie vecine, fie au un vecin comun. Altfel, operația introduce un ciclu în arbore. Mai observăm și că, dacă există vreo muchie între două noduri A și B din arborele original pentru care niciunul dintre A și B nu a făcut parte dintr-o operație de împăturire, atunci arborele obținut în urma tuturor operațiilor de împăturire nu are cum să fie stea. Astfel, problema se reduce la a afla un subset minim de noduri S astfel încât fiecare muchie să aibă cel puțin unul din capete într-un nod ce aparține lui S. Acest subset S se numește acoperirea minimă cu noduri a arborelui[3]

Pentru a implementa acest lucru, putem face o parcurgere în adâncime pe arborele original, și să aplicăm următoarea strategie greedy: după ce am parcurs toți subarborii aferenți unui nod x, îl vom selecta pe x ca făcând parte din subsetul S dacă și numai dacă cel puțin unul dintre fii direcți ai lui x nu aparține lui S (pentru a evita situația descrisă în a doua observație). Soluții alternative care folosesc programarea dinamică sau algoritmul de cuplaj în graf bipartit sunt de asemenea suficiente pentru a obține punctaj maxim.

Pentru a reconstitui, pe baza setului S, operațiile care duc la împăturirea arborelui inițial într-unul stea, este suficient să luăm nodurile din S în ordinea adâncimii lor în arborele inițial, și să le unim la fiecare pas cu cel mai de sus nod aflat în arbore la momentul respectiv. Acesta este, la prima operație, cel mai de sus nod aflat în arborele inițial, iar apoi, nodul creat în urma precedentei operații de împăturire. Observăm că, aplicând această strategie, operațiile respectă mereu prima observație. Complexitatea finală a acestei soluții este $O(N)$.

Subtask 1 (10 puncte)

Soluția parțială pentru primul subtask presupune selectarea subsetului S cu backtracking în $O(2^N)$ și apoi simularea operațiilor în $O(N^2)$ pentru a ne asigura că ele sunt valide, pentru o complexitate totală de $O(2^N N^2)$.

Subtask 2 (20 de puncte)

Al doilea subtask admite orice algoritm polinomial pentru determinarea setului S (de exemplu, programare dinamică în $O(N^2)$).

Subtask 3 (10 puncte)

Pentru al treilea subtask, observăm că arborele dat este un lanț, prin urmare putem selecta setul S ca fiind format din toate nodurile cu indice par.

[3] https://en.wikipedia.org/wiki/Vertex_cover

6.2.E Problema Guguștiuc

Propusă de: stud. Matei Tinca — Vrije Universiteit Amsterdam

Pentru primul subtask, putem simula efectiv toate operațiile în complexitate $O(N^2)$.

Pentru următoarele două subtaskuri, pentru fiecare interval ne putem menține un set în care stocăm toate intervalele rămase. Atunci când vine o operație de `split` luăm intervalele care îl conțin pe x și le împărțim în câte două intervale. Complexitatea soluției este $O(QN \log N)$.

O observație importantă pentru găsirea soluției este faptul că dacă avem două operații de `split` la pozițiile x și y și după avem o operație de `skip` la o poziție z, atunci din toate intervalele care îl conțin pe z se va șterge intervalul (x, y). Putem să reformulăm problema astfel: se dau N intervale și Q updateuri definite prin triplete de numere (x, y, z), iar un update înseamnă că pentru fiecare interval care îl conține pe z se șterge intervalul (x, y).

Pentru a ne aduce problema la noua formă, putem simula operațiile inițiale pe intervalul $(1, \infty)$, folosind soluția de la subtaskul anterior. Când avem o operație de `skip`, vedem primul `split` din stânga punctului în care se aplică operația și la fel și pe partea dreaptă. Astfel, ne putem scoate tripletele de numere (x, y, z).

O proprietate importantă a acestor triplete ce poate fi dedusă din modul de construcție este faptul că intervalele (x, y) specifice tripletelor sunt fie disjuncte, fie incluse unul în altul.

În această formă a problemei, tripletele devin updateuri, iar cele N intervale inițiale devin queryuri: pentru un interval (x, y), vrem să vedem, după ce aplicăm toate updateurile, cu cât contribuie la răspuns intervalul respectiv.

Pentru cazul în care există mai multe triplete care au același z, putem lua în considerare doar primul triplet găsit (și anume cel care are intervalul cel mai mare).

Pe această formă, putem obține soluții pentru restul subtaskurilor.

Ne putem menține un arbore de intervale, unde pentru un segment de lungime 1, adică de forma $(x, x + 1)$ reținem dacă acel segment a fost acoperit. Ca și detalii de implementare, acest arbore de intervale va menține minimul pe un interval și de câte ori apare acesta, iar când vrem să marcăm faptul că am acoperit intervalul (x, y), atunci adăugăm 1 pe intervalul $(x, y - 1)$.

Pentru un interval din cele N inițiale, pentru a vedea cât rămâne din acesta după ce aplicăm operațiile definite de triplete, luăm toate tripletele cu z care aparține intervalului respectiv și acoperim în arborele de intervale fiecare interval asociat tripletelor. Lungimile rămase vor fi numărul de apariții ale lui 0 pe intervalul $x, y - 1$.

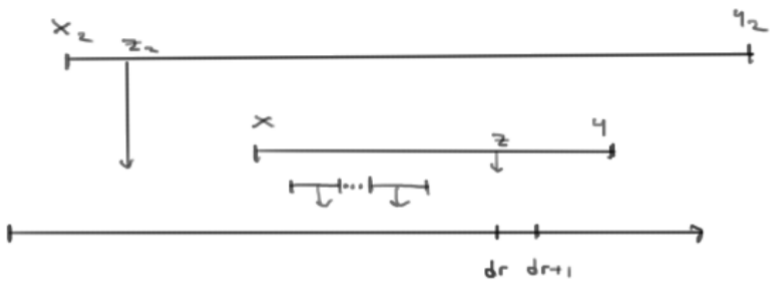
Pentru următoarele două subtaskuri, presupunem că am rezolvat intervalul (x, y) și vrem să rezolvăm alt interval. Putem să ne mișcăm pointerii x și y până când se potrivesc cu cei ai intervalului pe care vrem să îl rezolvăm. Mișcarea unui pointer înseamnă incrementarea sau decrementarea acestuia, iar după fiecare modificare se activează sau dezactivează tripletele care intră sau ies din acoperirea intervalului (x, y). Dacă rezolvăm intervalele în ordinea din input, rezolvăm subtaskul 4. Dacă folosim Algoritmul lui Mo, atunci rezolvăm și subtaskul 5. Complexitățile sunt $O(val_{max} \log N)$, respectiv $O(N\sqrt{N})$.

Pentru a rezolva toată problema, putem încerca următoarea abordare: iterăm cu capătul dreapta al unui interval și vedem cum se schimbă răspunsul în funcție de capătul stâng. Presupunând că suntem la pasul dr, ne calculăm șirul res_{st} ca fiind rezultatul queryului (st, dr). Dacă

avem șirul calculat pentru pasul dr, trebuie să vedem cum se modifică res_{st} dacă trecem de la dr la $dr + 1$.

Trebuie să vedem intervalul $(dr, dr + 1)$ la ce elemente din șir contribuie. Pentru acest caz, observăm că se adaugă 1 la toate elementele de pe pozițiile $z', z' + 1, \ldots dr$, unde z' este cel mai mare z al unui triplet în care intervalul acestuia acoperă intervalul $(dr, dr + 1)$, deoarece toate intervalele cu capătul stâng până la acel punct nu vor fi afectate de vreun update, iar după acel punct, intervalul $(dr, dr + 1)$ va fi șters.

Acuma trebuie ținut cont de cazul în care la poziția dr avem un triplet care afectează șirul.



Pe desenul de deasupra, (x_2, y_2, z_2) este un triplet prin care am trecut deja care are z-ul cât mai mare, iar intervalul lui, (x_2, y_2) îl include în totalitate pe intervalul (x, y). Astfel, putem observa următoarele două modificări: pe intervalul $(x, dr - 1)$, elementele lui *res* devin 0, deoarece acestea vor include tripletul (x, y, z), deci toate intervalele cu capătul între x și $dr - 1$ vor fi șterse. A doua modificare asupra șirului *res* va fi faptul că în intervalul $(z_2, x - 1)$ se va scădea *sub*, unde *sub* este lungimea intervalului (x, dr) din care scădem toate intervalele mai mici care sunt incluse în (x, dr). Practic, pentru toate intervalele cu acele capete, trebuie să ținem cont de faptul că se șterg toate bucățile neacoperite din acel interval.

Aceste modificări se pot implementa folosi un arbore de intervale care suportă următoarele operații:

- $\texttt{add}(x, y, z)$ — Pe intervalul (x, y) se adaugă valoarea z;
- $\texttt{set}_0(x, y)$ — Intervalul (x, y) se setează pe 0.

Complexitatea acestei soluții va fi $O((val_{max} + Q + N) \log N)$ și obține scor maxim.

6.2.F Problema Hoață

Propusă de: asist. doctorand Andrei-Costin Constantinescu — ETH Zürich

Subtask 1 (11 puncte)

Se folosește metoda backtracking. Restricția $g_i \geq 2$ garantează încadrarea soluției în limita de timp.

Subtask 2 (18 puncte)

Se poate proceda după cum urmează: rând pe rând determinăm acțiunile fiecărui hoț, alegând de fiecare dată acțiuni care nu ar declanșa nicio alarmă date fiind acțiunile hoților procesați în prealabil și care ar duce la o captură maximă pentru hoțul curent. Mai exact, pentru fiecare pereche (i, g), unde $1 \leq i \leq N$ și $0 \leq g \leq G$ vom menține o variabilă $x_{i,g}$ reprezentând numărul de hoți care ar mai putea trece prin camera i cu greutate a rucsacului g fără a declanșa alarma. Inițial, vom seta $x_{i,g} = x_i$, iar pe parcurs vom scădea aceste valori după fiecare hoț ale cărui acțiuni au fost decise. Acum, pentru a determina acțiunile optime ale unui hoț, vom face o dinamică $dp_{i,g} =$ care este captura maximă posibilă a hoțului curent până în camera i astfel încât acesta împreună cu hoții anteriori să nu declanșeze alarma iar rucsacul hoțului să fie umplut cu obiecte de greutate totală g, unde vom lua $dp_{i,g} = -\infty$ dacă o alarmă s-ar declanșa indiferent de acțiunile hoțului curent. Acestă dinamică se calculează pentru $1 \leq i \leq N+1$ și $0 \leq g \leq G$. Definim cantitățile

$$A = \begin{cases} v_i + dp_{i,g-g_i} \text{ dacă } i \leq N \text{ și } g_i \leq g, \\ -\infty \text{ altfel;} \end{cases}$$

$$B = \begin{cases} dp_{i-1,g} \text{ dacă } i \geq 2 \text{ și } x_{i-1,g} > 0, \\ -\infty \text{ altfel.} \end{cases}$$

Atunci avem relația de recurență $dp_{i,g} = \max(A, B)$. Inițializarea dinamicii este dată de cazul excepțional $dp_{1,0} = 0$. După calculul dinamicii se poate reconstitui șirul optim de acțiuni ale hoțului curent, după care se trece la hoțul următor. Complexitatea soluției este $O(NKG)$. Lăsăm ca temă cititorului motivul pentru care algoritmul prezentat este corect (adică de ce este corect sa determinăm acțiunile hoților unul câte unul).

Subtaskurile 3 și 4 (69 de puncte, respectiv, 100 de puncte)

În acest caz, camerele pot avea alarme de valori x_i variate, așa că algoritmul folosit pentru rezolvarea subtaskului 2 nu mai este corect, însă ne putem în continuare folosi de graful de stări al dinamicii pentru a determina simultan acțiunile optime ale tuturor hoților. Mai exact, pentru rezolvarea subtaskurilor 3 și 4 vom folosi tehnica flux maxim de cost minim, [4] după cum urmează. Vom avea câte un nod $n_{i,g}$ pentru fiecare $1 \leq i \leq N+1$ și $0 \leq g \leq G$. Fiecare muchie orientată $a \to b$ din rețeaua de flux va avea o capacitate pozitivă x și un cost nepozitiv y. În acest caz vom nota $a \xrightarrow[y]{x} b$. Acestea fiind spuse, în rețeaua de flux introducem următoarele muchii:

[4] https://infoarena.ro/problema/fmcm

- Pentru $i \leq N$ și $g_i \leq g$ adaugăm muchia $n_{i,g-g_i} \xrightarrow[-v_i]{\infty} n_{i,g}$;
- Pentru $i \geq 2$ adaugăm muchia $n_{i-1,g} \xrightarrow[0]{x_{i-1}} n_{i,g}$.

Sursa se alege nodul $n_{1,0}$ și se introduc introduc în rețea K unități de flux (dacă acest lucru nu este posibil, atunci se va afișa -1) astfel încât costul total al fluxului astfel determinat sa fie minim. Răspunsul va fi atunci -1 înmulțit cu costul fluxului.

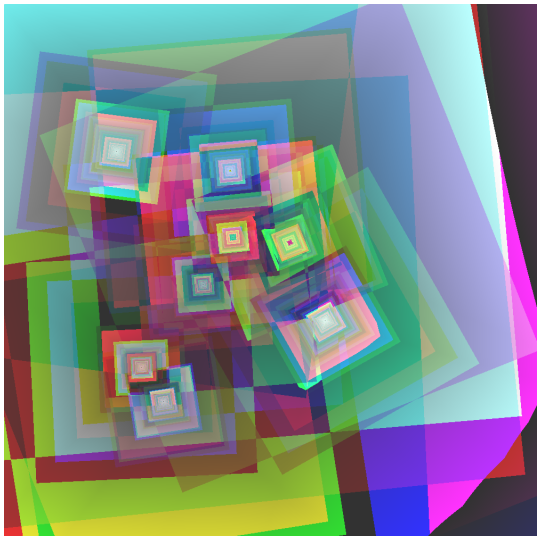
În funcție de implementare, acestă abordare poate obține între 69 și 100 de puncte. Pentru 100 de puncte, următoarele optimizări sunt necesare:

- Graful de flux are $(N+1)(G+1) \leq 90\,601$ noduri, deci implementarea nu se poate face cu matrice de adiacență, ci sunt necesare listele de adiacență.
- Găsirea drumurilor minime în graful rezidual trebuie făcută folosind tehnica Dijkstra cu potențiale,[5] în locul mai uzualului Bellman-Ford cu coadă.[6] Calculul potențialelor se poate face cu programare dinamică, deoarece graful rețelei de flux este aciclic.

Într-o implementare eficientă, complexitatea este $O\left(NKG\log(NG)\right)$.

[5] https://en.wikipedia.org/wiki/Johnson%27s_algorithm
[6] https://infoarena.ro/problema/bellmanford

Partea III
Surse Oficiale



Capitolul A Olimpiada Județeană de Informatică

A.1 Clasa a V-a

A.1.A Problema Ceas

```
/// 100p
#include <fstream>
using namespace std;
ifstream fin("ceas.in");
ofstream fout("ceas.out");
int n, C, k;
int main() {
    fin >> C >> k >> n;
    /// cerinta C=1
    int nr_k = 0, nr_t = 0, x;
    if (C == 1) {
        for (int i = 1; i <= n; i++) {
            fin >> x;
            if (x == k)
                nr_k++;
            else
                while (x > 0) {
                    if (x % 10 == k) nr_k++;
                    x /= 10;
                }
        }
        fout << nr_k;
    }
    if (C == 2)  /// cerinta C=2
    {
        for (int i = 1; i <= n; i++) {
            fin >> x;
            while (x > 12) {
                if (x % 100 <= 12 && x % 100 >= 10)
                    x /= 100;
                else
```

```
                    x /= 10;
                nr_t++;
            }
        }
        fout << nr_t;
    }
    fin.close();
    fout.close();
    return 0;
}
```

A.1.B Problema Sss

```
/// 100p
#include <fstream>
#include <iostream>
using namespace std;
int n, c, sum, L, sol1, sol2, x, k, sc, nr, i;
int main() {
    ifstream fin("sss.in");
    ofstream fout("sss.out");
    fin >> c;
    fin >> n;

    L = 0;
    sum = 0;
    while (sum < n) {
        L++;
        sum += L;
    }

    for (i = 1; i <= n; i++) {
        fin >> x;

        if (i == 1) {
            k = x;
            while (k % 10 == 0) k /= 10;
            k = k % 10;
        }

        if (i + k - 1 >= n) sol1 += x;

        sc += x;
        nr++;
        if (nr == L) {
            if (sc > sol2) {
                sol2 = sc;
            }
            L--;
            nr = 0;
            sc = 0;
        }
```

```
    }
    if (c == 1)
        fout << sol1 << "\n";
    else
        fout << sol2 << "\n";
    return 0;
}
```

A.2 Clasa a VI-a

A.2.A Problema Cmmdc

```
/**
Implementare Dan Pracsiu - 100p
*/
#include <bits/stdc++.h>
using namespace std;

ifstream fin("cmmdc.in");
ofstream fout("cmmdc.out");
long long a[100002], st[100002], dr[100002];
int n;

int main() {
    int i, j, T;
    long long c, d, M;
    /// citire
    fin >> T >> n;
    for (i = 1; i <= n; i++) fin >> a[i];

    /// cmmdc partial de la stanga la dreapta
    st[1] = a[1];
    for (i = 2; i <= n; i++) st[i] = __gcd(st[i - 1], a[i]);

    /// cmmdc partial de la dreapta la stanga
    dr[n] = a[n];
    for (i = n - 1; i >= 1; i--) dr[i] = __gcd(a[i], dr[i + 1]);

    if (T == 1)  /// rezolvare a)
    {
        fout << st[n];
    } else if (T == 2)  /// rezolvare b)
    {
        M = 0;
        for (i = 1; i <= n; i++) {
            d = __gcd(st[i - 1], dr[i + 1]);
            M = max(M, d);
        }
        fout << M;
    } else  /// /// rezolvare c)
```

```
    {
        /// aflam cmmdc-ul maxim obtinut prin eliminarea lui a[i] si a[j]
        M = 0;
        for (i = 1; i < n; i++) {
            c = 0;  /// c = cmmdc(a[i+1..j])
            for (j = i + 1; j <= n; j++) {
                d = __gcd(st[i - 1], c);
                d = __gcd(d, dr[j + 1]);
                M = max(M, d);
                c = __gcd(c, a[j]);
            }
        }
        fout << M;
    }
    fin.close();
    fout.close();
    return 0;
}
```

A.2.B Problema Vecine

```
/*
Implementare Dan Pracsiu - 100p
*/
#include <fstream>
#include <iostream>
using namespace std;

ifstream fin("vecine.in");
ofstream fout("vecine.out");
int n, a[100003];

void F1() {
    int i, cnt = 0;
    for (i = 1; i < n; i++)
        if (a[i] + 1 == a[i + 1]) cnt++;
    fout << cnt << "\n";
}

void F2() {
    int i, j, L, k;
    long long x, y, ans = -1;
    L = min(10, n / 2);
    for (k = L; k >= 1; k--) {
        /// cautam daca exista doua numere alaturate consecutive de k
        /// cifre
        for (i = 1; i <= n - 2 * k + 1; i++)
            if (a[i] != 0) {
                x = y = 0;
                for (j = 0; j < k; j++) x = x * 10 + a[i + j];
                for (j = 0; j < k; j++) y = y * 10 + a[i + k + j];
                if (x + 1 == y) ans = max(ans, x);
```

```
                /// verific daca x si y nu sunt de forma x=999, y=1000
                if (i + 2 * k <= n) {
                    y = y * 10 + a[i + 2 * k];
                    if (x + 1 == y) ans = max(ans, x);
                }
            } else if (k == 1 && a[i + 1] == 1)
                ans = max(ans, 0LL);
    }
    fout << ans << "\n";
}

int main() {
    int task;
    fin >> task >> n;
    for (int i = 1; i <= n; i++) fin >> a[i];
    if (task == 1)
        F1();
    else
        F2();
    fin.close();
    fout.close();
    return 0;
}
```

A.3 Clasa a VII-a

A.3.A Problema Patratele

```
/**
Autor: Bogdan-Ioan Popa, FMI Universitatea din Bucuresti
Scor: 100p
**/

#include <bits/stdc++.h>

#define N_MAX 60

using namespace std;

ifstream fin("patratele.in");
ofstream fout("patratele.out");

int N, M, t;

int A[N_MAX + 5][N_MAX + 5];
int up[N_MAX + 5][N_MAX + 5];
int ri[N_MAX + 5][N_MAX + 5];
int dwn[N_MAX + 5][N_MAX + 5];
int le[N_MAX + 5][N_MAX + 5];
int ans[N_MAX + 5];
```

```
int di[] = {-1, 0, 1, 0};
int dj[] = {0, 1, 0, -1};
int dir[] = {2, 3, 0, 1};
int p2[] = {1, 2, 4, 8};

char side[][8] = {"SUS", "DREAPTA", "JOS", "STANGA"};

bool check_side(int conf, int side) { return conf / p2[side] % 2; }

void flip_side(int &conf, int side) {
    if (check_side(conf, side)) {
        conf -= p2[side];
    } else {
        conf += p2[side];
    }
}

int compute_ans() {
    memset(ans, 0, sizeof(ans));
    memset(le, 0, sizeof(le));
    memset(ri, 0, sizeof(ri));
    memset(up, 0, sizeof(up));
    memset(dwn, 0, sizeof(dwn));

    int total_ans = 0;

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= M; j++) {
            if (check_side(A[i][j], 1)) {
                up[i][j] = 1 + up[i - 1][j];
            }

            if (check_side(A[i][j], 2)) {
                le[i][j] = 1 + le[i][j - 1];
            }
        }
    }

    for (int i = N; i >= 1; i--) {
        for (int j = M; j >= 1; j--) {
            if (check_side(A[i][j], 0)) {
                ri[i][j] = 1 + ri[i][j + 1];
            }

            if (check_side(A[i][j], 3)) {
                dwn[i][j] = 1 + dwn[i + 1][j];
            }
        }
    }

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= M; j++) {
            int max_k =
                min(min(dwn[i][j], ri[i][j]), min(N - i + 1, M - j + 1));
```

```
            for (int k = 1; k <= max_k; k++) {
                int ii = i + k - 1;
                int jj = j + k - 1;
                if (k <= up[ii][jj] && k <= le[ii][jj]) {
                    ans[k]++;
                    total_ans++;
                }
            }
        }
    }
    return total_ans;
}

int main() {
    fin >> N >> M >> t;
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= M; j++) {
            fin >> A[i][j];
        }
    }

    int sol = compute_ans();
    if (t == 3) {
        int max_ans = sol;
        int ans_i, ans_j, ans_d;
        bool ok = false;
        for (int i = 1; i <= N; i++) {
            for (int j = 1; j <= M; j++) {
                for (int d = 0; d < 4; d++) {
                    if (!check_side(A[i][j], d)) {
                        int ii = i + di[d];
                        int jj = j + dj[d];
                        flip_side(A[i][j], d);
                        flip_side(A[ii][jj], dir[d]);
                        int curr = compute_ans();
                        flip_side(A[i][j], d);
                        flip_side(A[ii][jj], dir[d]);
                        if (max_ans < curr) {
                            ok = true;
                            max_ans = curr;
                            ans_i = i;
                            ans_j = j;
                            ans_d = d;
                        }
                    }
                }
            }
        }
        if (!ok) {
            fout << "0\n0 0 NU\n";
        } else {
            fout << max_ans << "\n"
                 << ans_i << " " << ans_j << " " << side[ans_d] << "\n";
        }
```

```
    } else {
        if (t == 1) {
            fout << sol << "\n";
        } else {
            for (int i = 1; i <= N; i++) {
                if (ans[i] != 0) {
                    fout << i << " " << ans[i] << "\n";
                }
            }
        }
    }
    return 0;
}
```

A.3.B Problema Pseudocomp

```
/**
Autor: Bogdan-Ioan Popa, FMI Universitatea din Bucuresti
Scor: 100p
**/

#include <bits/stdc++.h>

#define N_MAX 100000
#define V_MAX 1000000

using namespace std;

ifstream fin("pseudocmp.in");
ofstream fout("pseudocmp.out");

int P, N;
int A[N_MAX + 5];
int freq[V_MAX + 5];

int dig_sum(int x) {
    int ret = 0;
    while (x) {
        ret += x % 10;
        x /= 10;
    }
    return ret;
}

int main() {
    fin >> P >> N;
    for (int i = 1; i <= N; i++) {
        fin >> A[i];
        freq[A[i]]++;
    }
```

```
    int L = 0;
    for (int i = 1; i <= V_MAX; i++) {
        while (freq[i]--) {
            A[++L] = i;
        }
        freq[i] = 0;
    }

    if (P == 1) {
        for (int i = 2; i <= N; i++) {
            if (dig_sum(A[i - 1]) > dig_sum(A[i])) {
                fout << A[i - 1] << " " << A[i] << "\n";
                return 0;
            }
        }
        fout << "-1\n";
        return 0;
    }

    if (P == 2) {
        long long ans = 0;
        for (int i = 1; i <= N; i++) {
            int sum = dig_sum(A[i]);
            for (int j = sum + 1; j <= 54; j++) {
                ans += freq[j];
            }
            freq[sum]++;
        }
        fout << ans << "\n";
    }
    return 0;
}
```

A.4 Clasa a VIII-a

A.4.A Problema Pelican

```
// Nistor Eugen Mot - O(k+p) 100 puncte
#include <fstream>
using namespace std;
#define M 10024
#define MM 100004
int px[M], py[M], pu[M];
char cx[MM];
int vx[MM];
int main() {
    ifstream fi("pelican.in");
    ofstream fo("pelican.out");
    int i, n, p, k, is = 0, u, x, y;
    fi >> n >> p >> k;
```

```
for (i = 1; i <= p; i++) {
    fi >> px[i] >> py[i] >> u;
    pu[i] = u - 1;
}
for (i = 1; i <= k; i++) {
    fi >> cx[i] >> vx[i];
    if (cx[i] == 'Z') is = i;
}
if (is > 0) {
    x = vx[is] / n;
    y = vx[is] % n;
    u = 0;
    for (i = 1; i < is; i++)
        if (cx[i] == 'R') u += vx[i];
    for (i = 1; i <= p; i++) {
        px[i] = x;
        py[i] = y;
        pu[i] = (pu[i] + u) % 4;
    }
}
u = 0;
x = 0;
y = 0;
for (i = is + 1; i <= k; i++) {
    if (cx[i] == 'R') u = (u + vx[i]) % 4;
    if (cx[i] == 'A') {
        if (u == 0)
            x += (n - vx[i]);
        else if (u == 1)
            y += vx[i];
        else if (u == 2)
            x += vx[i];
        else
            y += (n - vx[i]);
    }
}
x %= n;
y %= n;
for (i = 1; i <= p; i++) {
    if (pu[i] == 0) {
        px[i] = (px[i] + x) % n;
        py[i] = (py[i] + y) % n;
    }
    if (pu[i] == 1) {
        px[i] = (px[i] + y) % n;
        py[i] = (py[i] + n - x) % n;
    }
    if (pu[i] == 2) {
        px[i] = (px[i] + n - x) % n;
        py[i] = (py[i] + n - y) % n;
    }
    if (pu[i] == 3) {
        px[i] = (px[i] + n - y) % n;
        py[i] = (py[i] + x) % n;
```

```
        }
        fo << px[i] << ' ' << py[i] << '\n';
    }
    return 0;
}
```

A.4.B Problema Strips

```
// Em. Cerchez 100 puncte
#include <fstream>
#define NRMAX 50002
using namespace std;
ifstream fin("strips.in");
ofstream fout("strips.out");

struct zona {
    int inc, sf;
};
zona Z[2][NRMAX];  // 0 Ana 1 Bogdan
int nrz[2];

int N, L, Nr, poz, C;
int p[2], lgmax[2];
int cautbinar(int poz);
bool valid(int poz, int unde, int culoare);
void plaseaza(int poz, int cine);
int main() {
    int i, poz, j;
    fin >> C >> N >> Nr >> L;
    for (i = 1; i <= Nr; i++) {
        fin >> poz;
        plaseaza(poz, 0);
        fin >> poz;
        plaseaza(poz, 1);
    }
    if (C == 1)
        fout << p[0] << ' ' << p[1] << '\n';
    else {
        for (j = 0; j < 2; j++)
            for (i = 1; i <= nrz[j]; i++)
                if (lgmax[j] < Z[j][i].sf - Z[j][i].inc + 1)
                    lgmax[j] = Z[j][i].sf - Z[j][i].inc + 1;
        fout << lgmax[0] << ' ' << lgmax[1] << '\n';
    }
    fout.close();
    return 0;
}

int cautbinar(int poz, int unde)
/// invariant Z[unde][st].inc<poz<=Z[unde][dr].inc
{
    int st = 0, dr = nrz[unde] + 1, mij;
```

```
    while (dr - st > 1) {
        mij = (st + dr) / 2;
        if (Z[unde][mij].inc >= poz)
            dr = mij;
        else
            st = mij;
    }
    return dr;
}

void plaseaza(int poz, int cine) {
    int unde, adv, alipit, j;
    if (poz + L - 1 >= N) {
        p[cine]++;
        return;
    }
    adv = 1 - cine;
    unde = cautbinar(poz, adv);
    if (unde <= nrz[adv] && poz + L - 1 >= Z[adv][unde].inc - 1) {
        p[cine]++;
        return;
    }
    if (unde > 1 && poz <= Z[adv][unde - 1].sf + 1) {
        p[cine]++;
        return;
    }
    /// mutare valida
    unde = cautbinar(poz, cine);
    // alipire
    alipit = 0;
    if (unde - 1 > 0)
        if (poz <= Z[cine][unde - 1].sf + 1) {
            Z[cine][unde - 1].sf =
                max(poz + L - 1, Z[cine][unde - 1].sf);
            alipit = 1;
        }

    if (unde <= nrz[cine])
        if (poz + L - 1 >= Z[cine][unde].inc - 1) {
            Z[cine][unde].inc = min(poz, Z[cine][unde].inc);
            alipit = 1;
        }
    // eliminare
    if (unde <= nrz[cine] && unde > 1 &&
        Z[cine][unde].inc <= Z[cine][unde - 1].sf + 1) {
        Z[cine][unde - 1].sf = Z[cine][unde].sf;
        for (j = unde; j < nrz[cine]; j++) Z[cine][j] = Z[cine][j + 1];
        nrz[cine]--;
    }
    if (!alipit)  // inserare
    {
        for (j = nrz[cine]; j >= unde; j--) Z[cine][j + 1] = Z[cine][j];
        nrz[cine]++;
        Z[cine][unde].inc = poz;
```

```
        Z[cine][unde].sf = poz + L - 1;
    }
}
```

A.5 Clasa a IX-a

A.5.A Problema Balba

```
// Andrei Arhire
// 100 de puncte
// O(N)
#include <bits/stdc++.h>
using namespace std;
const int NR = 1e5 + 15;
int n, c, v[NR], sol1, sol2, fr[NR], highestDigit;
ifstream in("balba.in");
ofstream out("balba.out");

void printPalilindrom(int digit, bool alreadySet, bool check) {
    for (int j = 1; j <= (fr[digit] >> 1); ++j) {
        out << digit;
    }
    if (check && !alreadySet && fr[digit] % 2 && fr[digit] > 1) {
        out << digit;
        --fr[digit];
        alreadySet = true;
    }
    if (check && fr[digit] % 2 && highestDigit == -1) {
        highestDigit = digit;
    }
    if (!digit) {
        if (highestDigit != -1) {
            out << highestDigit;
        }
    } else {
        printPalilindrom(digit - 1, alreadySet, check);
    }
    for (int j = 1; j <= (fr[digit] >> 1); ++j) {
        out << digit;
    }
}

void tryFirstCase() {
    highestDigit = -1;
    int ret = 0;
    for (int i = 1; i < 10; ++i) {
        ret |= fr[i] % 2 && fr[i] > 1;
    }
    if (!ret) {
        return;
```

```
    }
    printPalilindrom(9, false, true);
    exit(0);
}

void trySecondCase() {
    highestDigit = -1;
    int ret = 1, ret2 = 0;
    for (int i = 0; i < 10; ++i) {
        ret &= !(fr[i] % 2);
    }
    for (int i = 1; i < 10; ++i) {
        ret2 |= fr[i] > 2;
    }
    if (!ret || !ret2) {
        return;
    }
    for (int i = 9; i; --i) {
        if (fr[i] > 2) {
            highestDigit = i;
            --fr[i];
            break;
        }
    }
    printPalilindrom(9, false, true);
    exit(0);
}

void tryThirdCase() {
    highestDigit = -1;
    int ret = 0;
    for (int i = 1; i < 10; ++i) {
        ret |= fr[i] > 1 && fr[i] % 2 == 0;
    }
    if (!ret) {
        return;
    }
    printPalilindrom(9, false, true);
    exit(0);
}

void tryFourthCase() {
    int ret = 1;
    for (int i = 1; i < 10; ++i) {
        ret &= fr[i] < 2;
    }
    if (!ret) {
        return;
    }
    for (int i = 9; i; --i) {
        if (fr[i]) {
            out << i;
            exit(0);
        }
```

```
    }
}

signed main() {
    ios::sync_with_stdio(false);
    in.tie(nullptr);
    in >> c >> n;
    for (int i = 1; i <= n; ++i) {
        in >> v[i];
    }
    for (int i = 1, j; i <= n; i = j) {
        for (j = i; j <= n && v[i] == v[j]; ++j)
            ;
        ++sol1;
        j - i > 1 ? sol2++ : true;
    }
    if (c == 1) {
        out << sol1 << '\n' << sol2 << '\n';
        return 0;
    }
    for (int i = 1; i <= n; ++i) {
        ++fr[v[i]];
    }
    tryFirstCase();
    trySecondCase();
    tryThirdCase();
    tryFourthCase();
    return 0;
}
```

A.5.B Problema Oneout

```
// Gheorghe Liviu Armand
// 100 de puncte
// Aceasta este solutia 3 din descrierea solutiilor
#include <bits/stdc++.h>
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("Ofast")
using namespace std;
ifstream f("oneout.in");
ofstream g("oneout.out");
pair<int, int> v[500002];
int kk = 25, p[] = {0,  2,  3,  5,  7,  11, 13, 17, 19, 23, 29, 31, 37,
                    41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97};
bool pp[1000002];
char m[1000002];
int nr[32], x;

inline bool liber() {
    if (m[x] != 0) return m[x] - 1;
    int i, y = 0;
```

```
    nr[++y] = x;
    if (pp[x]) {
        m[x] = 1;
        return 0;
    }
    for (i = 1; i <= kk && p[i] * p[i] * p[i] <= x; i++) {
        if (x % (p[i] * p[i]) == 0) {
            for (i = 1; i <= y; i++) m[nr[i]] = 1;
            return 0;
        }
        if (x % p[i] == 0) x /= p[i];
        if (pp[x]) m[x] = 1;
        if (m[x] != 0) {
            for (i = 1; i <= y; i++) m[nr[i]] = m[x];
            return m[x] - 1;
        }
        if (x != nr[y]) nr[++y] = x;
    }
    for (i = 1; i <= y; i++) m[nr[i]] = 2;
    return 1;
}

int main() {
    int n, i, l = 0, k = 0, maxim = 0, poz = 1, cate = 0, c;
    f >> c >> n;
    for (i = 2; i <= 1000; i++) pp[i * i] = 1;
    if (c == 1) {
        for (i = 1; i <= n; i++) {
            f >> x;
            k += liber();
        }
        g << k;
    } else {
        for (i = 1; i <= n; i++) {
            f >> x;
            if (liber())
                k++;
            else
                l = k, k = 0, poz = i - 1;
            if (k != 0 && l != 0 && poz == i - k - 1) {
                if (l + k > maxim) {
                    maxim = l + k;
                    cate = 0;
                    v[++cate] = {poz - l + 1, i};
                } else if (l + k == maxim)
                    v[++cate] = {poz - l + 1, i};
            }
        }
        if (maxim == 0)
            g << -1;
        else {
            g << maxim << " " << cate << '\n';
            for (i = 1; i <= cate; i++)
                g << v[i].first << " " << v[i].second << '\n';
```

```
        }
    }
    return 0;
}
```

A.5.C Problema Pergament

```
#include <algorithm>
#include <fstream>
using namespace std;
ifstream in("pergament.in");
ofstream out("pergament.out");
int n, k, q, a, b, c, d, prm, lun, x, y, z, s;
struct str {
    int c, l, v;
} query[300000];
int sume[100], v[100];
bool cmp(str a, str b) { return a.l < b.l; }
int main(void) {
    in >> n >> k >> q;
    in >> a >> b >> c >> d;
    in >> prm >> lun;
    for (int i = 1; i <= q; i++) {
        in >> x >> y >> z;
        s++;
        query[s].l = y;
        query[s].c = x;
        query[s].v = 1;

        s++;
        query[s].l = y + z;
        query[s].c = x;
        query[s].v = -1;
    }

    sort(query + 1, query + s + 1, cmp);

    q = 1;
    int total = 0;
    for (int i = 1; i <= n; i++) {
        bool ok = false;
        while (query[q].l == i) {
            v[query[q].c] += query[q].v;
            q++;
            ok = true;
        }
        if (ok) {
            for (int j = 1; j <= k; j++) {
                sume[j] = sume[j - 1] + v[j];
            }
        }
```

```
        total += sume[prm + lun - 1] - sume[prm - 1];

        prm = (prm * a + b) % k + 1;
        lun = (lun * c + d) % (k - prm + 1) + 1;
    }

    out << total;

    return 0;
}
```

A.6 Clasa a X-a

A.6.A Problema Circular

```
// prof Carmen Popescu - CNGL Sibiu
#include <fstream>
#include <iostream>

using namespace std;

ifstream f("circular.in");
ofstream g("circular.out");

char s[50005], v[30], s1[100005];
short d[30][30];

int main() {
    int t, i;
    long long timp = 0, nr = 1, mn, ct, d1, j, k, l = 0;

    for (int c1 = 0; c1 < 26; c1++)
        for (int c2 = c1; c2 < 26; c2++)
            d[c1][c2] = d[c2][c1] =
                ((26 - c2 + c1 < c2 - c1) ? 26 - c2 + c1 : c2 - c1);

    f >> t;
    if (t == 1) {
        f >> s;
        timp = d[0][s[0] - 'A'];  // dist de la 'A' la s[0]
        for (i = 1; s[i] != '\0'; i++)
            timp += d[s[i - 1] - 'A'][s[i] - 'A'];
        g << timp;
        return 0;
    }
    if (t == 2) {
        l = 0;
        f >> s >> v;
        timp = d[0][s[0] - 'A'];
        for (i = 0; s[i + 1] != '\0'; i++) {
```

```
            mn = 50;
            s1[l] = s[i];
            l++;
            for (j = 0; v[j] != '\0'; j++) {
                d1 = d[s[i] - 'A'][v[j] - 'A'] +
                     d[v[j] - 'A'][s[i + 1] - 'A'];
                if (d1 < mn) {
                    mn = d1;
                    ct = 1;
                    k = j;
                } else if (d1 == mn)
                    ct++;
            }
            s1[l] = v[k];
            l++;
            timp = timp + mn;
            nr = nr * ct % 666013;
        }
        s1[l] = s[i];
        l++;
        s1[l] = '\0';
        g << timp << "\n";
        g << nr << "\n";
        g << s1 << "\n";
    }
}
```

A.6.B Problema Pulsar

```
/// Andrei Cotor - Universitatea Babes Bolyai Cluj Napoca

#include <algorithm>
#include <fstream>
#include <iostream>
#include <queue>

using namespace std;

struct PULSAR {
    int x, y, r, t;
};

struct QSTATE {
    int cost, x, y, st;

    bool operator<(const QSTATE &other) const {
        return (this->cost) > (other.cost);
    }
};

int getLCM(int x, int y) { return (x * y) / __gcd(x, y); }
```

```
int manhattanDist(int x, int y, int xx, int yy) {
    return abs(x - xx) + abs(y - yy);
}

PULSAR PS[15005];
bool state[65][505][505];

const int dx[] = {-1, 0, 1, 0, 0};
const int dy[] = {0, 1, 0, -1, 0};

void fillDist(int x, int y, int xs, int ys, int dist, int st, int N) {
    state[st][x][y] = 1;

    if (manhattanDist(x, y, xs, ys) < dist) {
        for (int d = 0; d < 4; d++) {
            if (x + dx[d] > 0 && x + dx[d] <= N && y + dy[d] > 0 &&
                y + dy[d] <= N &&
                manhattanDist(x + dx[d], y + dy[d], xs, ys) >
                    manhattanDist(x, y, xs, ys)) {
                fillDist(x + dx[d], y + dy[d], xs, ys, dist, st, N);
            }
        }
    }
}

int Cost[65][505][505];

int main() {
    ifstream fi("pulsar.in");

    int C, N, P;
    fi >> C >> N >> P;

    int lcm = 1;
    for (int i = 0; i < P; i++) {
        int x, y, r, t;
        fi >> x >> y >> r >> t;

        lcm = getLCM(r, lcm);
        PS[i] = {x, y, r, t};
    }

    int xs, ys, xd, yd;
    fi >> xs >> ys >> xd >> yd;

    fi.close();

    for (int i = 0; i < lcm; i++) {
        for (int j = 0; j < P; j++) {
            fillDist(PS[j].x, PS[j].y, PS[j].x, PS[j].y, PS[j].t, i, N);
            PS[j].t = (PS[j].t + 1) % PS[j].r;
        }
    }
```

```
if (C == 1) {
    int rez1 = 0;
    for (int i = 0; i < lcm; i++) {
        int nr = 0;
        for (int j = 1; j <= N; j++) {
            for (int k = 1; k <= N; k++) {
                nr += state[i][j][k];
            }
        }
        rez1 = max(rez1, nr);
    }

    ofstream fo("pulsar.out");
    fo << rez1 << "\n";
    fo.close();
    return 0;
}

for (int i = 0; i < lcm; i++) {
    for (int j = 1; j <= N; j++) {
        for (int k = 1; k <= N; k++) {
            Cost[i][j][k] = 1000000000;
        }
    }
}

priority_queue<QSTATE> Q;
Q.push({0, xs, ys, 0});
Cost[0][xs][ys] = 0;

while (!Q.empty()) {
    QSTATE cr = Q.top();
    Q.pop();

    if (Cost[cr.st][cr.x][cr.y] < cr.cost) continue;

    for (int d = 0; d < 5; d++) {
        QSTATE nxt;
        nxt = {cr.cost + 1, cr.x + dx[d], cr.y + dy[d],
               (cr.st + 1) % lcm};

        if (nxt.x > 0 && nxt.x <= N && nxt.y > 0 && nxt.y <= N &&
            !state[nxt.st][nxt.x][nxt.y]) {
            if (Cost[nxt.st][nxt.x][nxt.y] > nxt.cost) {
                Cost[nxt.st][nxt.x][nxt.y] = nxt.cost;
                Q.push(nxt);
            }
        }
    }
}

int rez = 1000000000;
for (int i = 0; i < lcm; i++) {
    rez = min(rez, Cost[i][xd][yd]);
```

```
    }

    ofstream fo("pulsar.out");
    fo << rez << "\n";
    fo.close();
    return 0;
}
```

A.6.C Problema Transport

```
/**
 *   David Coroian
 *   Problema Transport
 */

#include <bits/stdc++.h>

#define MAX_N 500000

using namespace std;

typedef long long lint;

const lint MOD = (lint)(1e9) + 7;

inline void modd(lint &a) {
    if (a >= MOD) a -= MOD;

    if (a < 0) a += MOD;
}

inline void modd(int &a) {
    if (a >= MOD) a -= MOD;

    if (a < 0) a += MOD;
}

unordered_map<lint, lint> ap;
unordered_map<lint, int> last;

lint rez;

lint c;
int q, n;

lint d[MAX_N + 1];
lint a[MAX_N + 1];

lint p2x[MAX_N + 1];

void readFile() {
    ifstream f("transport.in");
```

```
    f >> q;
    f >> n >> c;

    for (int i = 1; i <= n; i++) f >> a[i] >> d[i];

    f.close();
}

void upd(lint x, int i) { ap[x] = ap[x] * p2x[i - last[x]] % MOD; }

void add(lint x, int i) {
    upd(x, i);

    ap[x]++;
    modd(ap[x]);
    last[x] = i;
}

void inc(lint x) { ap[x]++; }

void getP2() {
    p2x[0] = 1;
    for (int i = 1; i <= n; i++) {
        p2x[i] = p2x[i - 1] << 1;
        modd(p2x[i]);
    }
}

void solve() {
    getP2();

    for (int i = 1; i <= n; i++) {
        rez += (q == 1 ? ap[c * a[i] - d[i]]
                       : ap[c * a[i] - d[i]] *
                             p2x[i - last[c * a[i] - d[i]] - 1] % MOD);
        modd(rez);
        (q == 1 ? inc(c * a[i] + d[i]) : add(c * a[i] + d[i], i));
    }
}

void printFile() {
    ofstream g("transport.out");

    g << rez << "\n";

    g.close();
}

int main() {
    readFile();

    solve();
```

```
    printFile();

    return 0;
}
```

A.7 Clasele XI–XII

A.7.A Problema Dulciuri

```
/* Tamio-Vesa Nakajima, 100p */
#include <cassert>
#include <fstream>
#include <iomanip>
#include <iostream>
using namespace std;

constexpr int maxn = 1e6 + 1;

class aib {
    int buf[maxn] = {};

   public:
    void update(int x, int delta) {
        for (++x; x < maxn; x += x & -x) buf[x] += delta;
    };
    long long query(int x) {
        long long r = 0;
        for (++x; x; x -= x & -x) r += buf[x];
        return r;
    }
    long long query(int st, int dr) {
        return query(dr - 1) - query(st - 1);
    }

    double solve(int st, int dr) {
        if (st > dr) swap(st, dr);
        return st == dr ? query(st, dr + 1)
                        : query(st, dr) / double(dr - st);
    }
};

aib ax, ay;

int main() {
    ifstream f("dulciuri.in");
    ofstream g("dulciuri.out");

    int q;
    f >> q;
```

```
    assert(1 <= q && q <= 100000);

    while (q--) {
        int t;
        f >> t;

        if (t == 1) {
            int x, v;
            f >> x >> v;
            assert((0 <= x && x <= 1000000));
            assert(0 <= v && v <= 1000);
            ax.update(x, v);
        } else if (t == 2) {
            int y, v;
            f >> y >> v;
            assert(0 <= y && y <= 1000000);
            assert(0 <= v && v <= 1000);
            ay.update(y, v);
        } else {
            int x, y, xx, yy;
            f >> x >> y >> xx >> yy;
            assert((0 <= x && x <= 1000000));
            assert(0 <= y && y <= 1000000);
            assert(0 <= xx && xx <= 1000000);
            assert(0 <= yy && yy <= 1000000);
            g << fixed << setprecision(10)
              << ax.solve(x, xx) + ay.solve(y, yy) << endl;
        }
    }
    return 0;
}
```

A.7.B Problema Investitie

```
/* Tamio-Vesa Nakajima, 100p */
#include <fstream>
#include <iostream>
#include <vector>
using namespace std;

using ll = long long;

constexpr int maxn = 1e5 + 10;

ifstream f("investitie.in");
ofstream g("investitie.out");
int n, m, q, v[maxn] = {}, which_c[maxn] = {}, pos_in_c[maxn] = {};
vector<ll> cycle_sp[maxn] = {};

ll query(int col, int upto) {
    int which = which_c[col], len = cycle_sp[which].size() - 1;
    upto += pos_in_c[col];
```

```
    return cycle_sp[which].back() * (upto / len) +
           cycle_sp[which][upto % len];
}

int main() {
    f >> n >> m;

    for (int i = 1; i <= n; ++i) f >> v[i];

    for (int i = 1, c = 1; i <= n; ++i)
        if (which_c[i] == 0) {
            cycle_sp[c].push_back(0);
            for (ll j = i, sp = 0; which_c[j] == 0; j = v[j]) {
                pos_in_c[j] = cycle_sp[c].size();
                cycle_sp[c].push_back(sp += j);
                which_c[j] = c;
            }
            ++c;
        }
    f >> q;

    while (q--) {
        int sus, jos, st, dr;
        f >> sus >> jos >> st >> dr;

        ll ret = 0;
        for (int i = st; i <= dr; ++i)
            ret = ret + query(i, jos) - query(i, sus - 1);
        g << ret << '\n';
    }
}
```

A.7.C Problema Superhedgy

```
/* Mihaela Cismaru, 100p */
#include <bits/stdc++.h>

using namespace std;
ifstream fin("superhedgy.in");
ofstream fout("superhedgy.out");
const int N = 1000002;
int n, m, i;
int64_t Lmax;
long long Lu, Ld;
struct cladire {
    int L, H, E;
    cladire() { L = H = E = 0; }
};

cladire U[N], D[N];
int main() {
    assert(fin >> n);
```

```
    assert(1 <= n && n <= 100000);
    for (i = 1; i <= n; i++) {
        assert(fin >> U[i].L >> U[i].H >> U[i].E);
        Lmax += U[i].L;
        Lu += U[i].L;
        assert(1 <= U[i].L && U[i].L <= 2000000000);
        assert(1 <= U[i].H && U[i].H <= 1000000000);
        assert(0 <= U[i].E && U[i].E <= 1000000000);
    }
    assert(fin >> m);
    assert(1 <= m && m <= 100000);
    for (i = 1; i <= m; i++) {
        assert(fin >> D[i].L >> D[i].H >> D[i].E);
        Lmax += D[i].L;
        Ld += D[i].L;
        assert(1 <= D[i].L && D[i].L <= 2000000000);
        assert(1 <= D[i].H && D[i].H <= 1000000000);
        assert(0 <= D[i].E && D[i].E <= 1000000000);
    }
    Lmax /= 2;
    assert(Lu == Ld);
    int i = 0, j = 0;
    int64_t solU = 0, solD = 0;
    while (i <= n && j <= m) {
        int lg = min(U[i].L, D[j].L);
        solU += lg;
        U[i].L -= lg;
        solD += lg;
        D[j].L -= lg;
        int64_t elevator = U[i].E + D[j].E;
        int64_t vertU = U[i].L ? 0 : abs(U[i].H - U[i + 1].H);
        int64_t vertD = D[j].L ? 0 : abs(D[j].H - D[j + 1].H);
        int64_t auxU = solU;
        int64_t auxD = solD;
        solU = min(auxU + vertU, auxD + elevator + vertU);
        solD = min(auxD + vertD, auxU + elevator + vertD);
        i = i + (U[i].L == 0);
        j = j + (D[j].L == 0);
    }
    fout << min(solU, solD);
    return 0;
}
```

Capitolul B Olimpiada Națională de Informatică

B.1 Clasa a V-a

B.1.A Problema Culori

```
#include <fstream>
using namespace std;
ifstream fin("culori.in");
ofstream fout("culori.out");
int cerinta, N, nr_culori, culoare1, culoare, ok;
short x[505], y[505];
int main() {
    fin >> cerinta >> N;
    if (cerinta == 1)  // cel mai lung rand ce contine NU contine 2
                       // patratele alaturate colorate la fel
    {
        int Lmax = 0, Kmax = 0;
        for (int i = 1; i <= N; i++) {
            fin >> nr_culori;
            ok = 1;
            fin >> culoare;
            for (int j = 1; j <= nr_culori - 1; j++) {
                fin >> culoare1;
                if (culoare == culoare1) ok = 0;
                culoare = culoare1;
            }
            if (ok)
                if (nr_culori > Lmax) {
                    Lmax = nr_culori;
                    Kmax = 1;
                } else if (Lmax == nr_culori)
                    Kmax++;
        }
        fout << Lmax << ' ' << Kmax << '\n';
    } else if (cerinta == 2) {
        fin >> x[0];  // citim primul numar
```

```
        for (int i = 1; i <= x[0]; i++) fin >> x[x[0] - i + 1];

        for (int i = 2; i <= N; i++) {
            fin >> y[0];  // citim urmatorul numar
            for (int j = 1; j <= y[0]; j++) fin >> y[y[0] - j + 1];
            // Compararea celor doua numere
            ok = 1;
            if (x[0] != y[0])  // cele doua numere au lungimi diferite
            {
                if (x[0] < y[0]) ok = -1;
            } else  // cele doua numere au aceeasi lungime
            {
                // comparam lexicografic
                int j = x[0];
                while ((x[j] == y[j]) && j > 0) j--;
                if (x[j] < y[j]) ok = -1;
            }
            if (ok == -1)  // al doilea numar e mai mare
                for (int j = 0; j <= y[0]; j++) x[j] = y[j];
        }
        for (int i = 1; i <= x[0]; i++) fout << x[x[0] - i + 1];
        fout << '\n';
    }
    return 0;
}
```

B.1.B Problema Joc

```
#include <bits/stdc++.h>
using namespace std;
ifstream in("joc.in");
ofstream out("joc.out");
int c, n, apoz, bpoz, i, mutare, vap[12001], vbp[12001], nra, nrb,
    cif[7] = {0}, maxx = 0, a, b, X;
int main() {
    in >> c >> n;
    /// calculez numarul de divizori ai lui n

    int d = 2, p = 1, nr = 0, x = n;
    while (x > 1) {
        nr = 0;
        while (x % d == 0) nr++, x /= d;
        p *= (nr + 1);
        d++;
        if (d * d > x and x > 1) x = 1, p *= 2;
    }
    if (c == 1)
        out << p;
    else {
        a = 1;
        b = 0;
```

```
        apoz = 1, bpoz = 1;  /// pun pionii pe prima pozitie
        vap[1] = vbp[1] = nra = nrb =
            1;  /// pun in vectori pozitiile pionilor
        while (apoz < n && bpoz < n) {
            mutare++;  /// de cate ori il calculeaza pe X copii
            if (mutare % 2 == 1)
                X = (mutare + (a * apoz + b * bpoz + n) % 10) % 6 + 1;
            else
                X = ((mutare + 1) % 5 + (a * apoz + b * bpoz + n) % 10) %
                        6 +
                    1;

            cif[X]++;
            if (maxx < cif[X]) maxx = cif[X];
            apoz += a * X;  /// daca a=1 si b=0
            bpoz += b * X;  /// daca a=0 si b=1
            if (a == 1) {
                if (apoz <= n)  /// sunt in tabla
                    vap[++nra] = apoz;
                else
                    vap[++nra] = n, apoz = n;  /// am iesit din tabla
                if (apoz == bpoz)
                    bpoz = 1,
                    vbp[++nrb] = 1;  /// sa nu uit sa trec pozitia de
                                     /// intoarcere in vector
            }

            else {
                if (bpoz <= n)
                    vbp[++nrb] = bpoz;
                else
                    vbp[++nrb] = n, bpoz = n;
                if (apoz == bpoz)
                    apoz = 1,
                    vap[++nra] = 1;  /// sa nu uit sa trec pozitia de
                                     /// intnrarcere in vector
            }

            if (X == 6)
                a %= 2, b %= 2;  /// ramane ordinea mutarii copiilor
            else
                a = (a + 1) % 2,
                b = (b + 1) % 2;  /// schimb ordinea mutarii copiilor
        }
        if (c == 2)
            out << maxx;
        else {
            if (apoz == n)
                for (i = 1; i <= nra; i++) out << vap[i] << " ";
            else
                for (i = 1; i <= nrb; i++) out << vbp[i] << " ";
        }
    }
}
```

B.1.C Problema Rotire25

```cpp
#include <fstream>

using namespace std;

ifstream f("rotire25.in");
ofstream g("rotire25.out");

int c, x, k;

int main() {
    f >> c >> x >> k;

    if (c == 1) {
        int first = x;
        while (first >= 10) {
            first /= 10;
        }
        x %= 10;
        // perioada 1
        if (x == 0 || x == 1 || x == 5 || x == 6) {
            k = 1;
        }
        // perioada 2
        else if (x == 4 || x == 9) {
            k %= 2;
            if (k == 0) k = 2;
        }
        // perioada 4
        else {
            k %= 4;
            if (k == 0) k = 4;
        }

        int ans = x;
        for (int i = 1; i < k; i++) {
            ans = (ans * x) % 10;
        }

        g << ans * first << '\n';
    } else {
        int last1 = -1, last2 = -1;
        bool is_5 = true;
        while (x != last2 && k) {
            last2 = last1;
            last1 = x;

            // aplicam transformarea
            int p = (is_5 ? 5 : 2);
            x *= p;
```

```
            int ans = 0;
            while (x) {
                if (x % 10) {
                    ans = ans * 10 + x % 10;
                }
                x /= 10;
            }

            x = ans;

            k--;
            is_5 = !is_5;
        }

        if (k % 2 == 1) {
            int p = (is_5 ? 5 : 2);
            x *= p;

            int ans = 0;
            while (x) {
                if (x % 10) {
                    ans = ans * 10 + x % 10;
                }
                x /= 10;
            }

            x = ans;
        }

        g << x << '\n';
    }

    return 0;
}
```

B.2 Clasa a VI-a

B.2.A Problema Iluminat

```
/*
Iluminat - Raluca Costineanu
C == 1 in O(k)
C == 2 in O(k)
C == 3 in O(n^2)
*/
#include <bits/stdc++.h>

using namespace std;

#define nMax 1010
```

```
ifstream f("iluminat.in");
ofstream g("iluminat.out");

long long a[nMax][nMax];
int C, n, k, mx;
int ap[3][nMax * nMax];
bool linii[nMax], coloane[nMax];

int main() {
    f >> C >> n >> k;
    int i, j;
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j) {
            f >> a[i][j];
            mx = max(1LL * mx, a[i][j]);
            ap[0][a[i][j]] = 1;
            ap[1][a[i][j]] = i;
            ap[2][a[i][j]] = j;
        }
    if (C == 1) {
        int bec = mx;
        int nrBecuri = 0;
        for (int etapa = 1; etapa <= k; ++etapa) {
            for (; bec >= 1; --bec)
                if (ap[0][bec] && linii[ap[1][bec]] == 0 &&
                    coloane[ap[2][bec]] == 0) {
                    nrBecuri = bec;
                    linii[ap[1][bec]] = 1;
                    coloane[ap[2][bec]] = 1;
                    break;
                }
        }
        g << nrBecuri << '\n';
    } else if (C == 2) {
        int bec = mx, lin, col;
        int nrBecuri = 0, cate = 0;
        for (int etapa = 1; etapa <= k - 1; ++etapa) {
            for (; bec >= 1; --bec)
                if (ap[0][bec] && linii[ap[1][bec]] == 0 &&
                    coloane[ap[2][bec]] == 0) {
                    nrBecuri = bec;
                    linii[ap[1][bec]] = 1;
                    coloane[ap[2][bec]] = 1;
                    break;
                }
        }
        for (; bec >= 1; --bec)
            if (ap[0][bec] && linii[ap[1][bec]] == 0 &&
                coloane[ap[2][bec]] == 0) {
                nrBecuri = bec;
                lin = ap[1][bec];
                col = ap[2][bec];
                break;
```

```
            }
        for (j = 1; j <= n; ++j)
            if (coloane[j] == 0) cate += a[lin][j];
        for (i = 1; i <= n; ++i)
            if (linii[i] == 0) cate += a[i][col];
        cate -= nrBecuri;
        g << cate << '\n';
    } else {
        for (i = 1; i <= n; ++i)
            for (j = 1; j <= n; ++j)
                a[i][j] += a[i - 1][j] + a[i][j - 1] - a[i - 1][j - 1];
        long long maxim = 0, s;
        for (i = k; i <= n; ++i)
            for (j = k; j <= n; ++j) {
                s = a[i][j] - a[i - k][j] - a[i][j - k] +
                    a[i - k][j - k];
                maxim = max(maxim, s);
            }
        g << maxim << '\n';
    }
    return 0;
}
```

B.2.B Problema Inundație

```
/* Task: Inundatie
 * Author: Tulba-Lecu Theodor-Gabriel
 * Time complexity:
 * Task 1: O(N) -- find minimum and maximum
 * Task 2: O(P) -- use a stack and keep track of the sum of
 *              elements popped until index P is encountered
 * Task 3: O(N) -- use algorithm from task 2 and simulate
 *              until more than S units of time are needed
 * Task 4: O(P) -- after processing, get the row with the
 *              biggest width from the stack
 * Asserted input for checking test validity
 */
#include <assert.h>

#include <cstdio>
#include <vector>

#define MAXN 100000
#define MAXH 20000
#define MAXS 100000

using namespace std;

int C, N, P, S;
vector<int> h;
vector<pair<int, int>> s;
vector<int> idx;
```

```
int main() {
    freopen("inundatie.in", "r", stdin);
    freopen("inundatie.out", "w", stdout);

    assert(scanf("%d%d%d%d", &C, &N, &P, &S) == 4);
    assert(C == 1 || C == 2 || C == 3 || C == 4);
    assert(3 <= N && N <= MAXN);
    assert(1 <= P && P <= N);
    assert(1 <= S && S <= MAXS);

    h.resize(N, 0);

    for (int i = 0; i < N; i++) {
        assert(scanf("%d", &h[i]) == 1);
        assert(1 <= h[i] && h[i] <= MAXH);
    }

    if (C == 1) {
        int minim = MAXH + 1, maxim = 0;
        for (int el : h) {
            minim = min(minim, el);
            maxim = max(maxim, el);
        }
        printf("%d\n", maxim - minim);
    } else if (C == 2) {
        int T = 0;
        for (int i = 0; i < P; i++) {
            int width = 1;
            while (!s.empty() && s.back().first < h[i]) {
                T += (h[i] - s.back().first) * s.back().second;
                width += s.back().second;
                s.pop_back();
            }
            s.push_back({h[i], width});
        }
        printf("%d\n", T);
    } else if (C == 3) {
        int D = 0;
        for (int i = 0; i < N; i++) {
            int width = 1;
            while (!s.empty() && s.back().first < h[i]) {
                S -= (h[i] - s.back().first) * s.back().second;
                width += s.back().second;
                s.pop_back();
            }
            s.push_back({h[i], width});
            if (S > 0) {
                D = i + 1;
            } else {
                break;
            }
        }
        printf("%d\n", D);
```

```
    } else {
        int R = 0, maxim = 0;
        for (int i = 0; i < P; i++) {
            int width = 1;
            while (!s.empty() && s.back().first < h[i]) {
                width += s.back().second;
                s.pop_back();
                idx.pop_back();
            }
            s.push_back({h[i], width});
            idx.push_back(i + 1);
        }

        for (int i = 0; i < (int)s.size(); i++) {
            if (maxim < s[i].second) {
                if (i < (int)s.size() - 1 &&
                    s[i].first == s[i + 1].first) {
                    continue;
                }
                maxim = s[i].second;
                R = idx[i];
            }
        }

        printf("%d\n", R);
    }

    return 0;
}
```

B.2.C Problema Șiruri

```
/* Task: siruri
 * Author: Tulba-Lecu Theodor-Gabriel
 * Time complexity: O(n * log10(n))
 * Space complexity: O(n)
 * Asserted input for checking test validity
 */
#include <cassert>
#include <cstdio>
#include <string>
#include <vector>

#define MAXN 100000

using namespace std;

int c, n, cnt_distinct;
long long p10[10];
vector<int> v;

long long mirror(long long x) {
```

```
    long long y = 0;
    while (x) {
        y = y * 10 + x % 10;
        x /= 10;
    }
    return y;
}

long long keep_distinct(long long x) {
    int fr[10] = {};
    long long y = 0;
    x = mirror(x);

    while (x) {
        if (fr[x % 10] == 0) {
            fr[x % 10]++;
            y = y * 10 + x % 10;
        }
        x /= 10;
    }

    return y;
}

bool has_zeros(long long x) {
    while (x) {
        if (x % 10 == 0) {
            return true;
        }
        x /= 10;
    }
    return false;
}

long long get_len(long long x) {
    int cnt = 0;
    while (x) {
        cnt++;
        x /= 10;
    }

    return cnt;
}

long long get_last(long long x) { return x % 10; }

long long get_first(long long x) {
    while (x > 10) {
        x /= 10;
    }
    return x;
}

long long merge(long long x, long long y) {
```

```
    return x * p10[get_len(y)] + y;
}

int main() {
    freopen("siruri.in", "r", stdin);
    freopen("siruri.out", "w", stdout);

    assert(scanf("%d%d", &c, &n) == 2);
    assert(c == 1 || c == 2 || c == 3);
    assert(1 <= n && n <= MAXN);

    p10[0] = 1;
    for (int i = 1; i < 10; i++) {
        p10[i] = p10[i - 1] * 10;
    }

    for (int i = 1; i <= n; i++) {
        long long x;
        assert(scanf("%lld", &x) == 1);
        assert(1 <= x && x <= 1000000000LL);
        assert(!has_zeros(x));

        if (x == keep_distinct(x)) cnt_distinct++;

        v.push_back(x);
    }

    int it = 0;
    for (int i = 1; i < n; i++) {
        if (get_last(v[it]) == get_first(v[i])) {
            v[it] = keep_distinct(merge(v[it], v[i]));
        } else {
            v[it] = keep_distinct(v[it]);
            it++;
            v[it] = v[i];
        }
    }
    v[it] = keep_distinct(v[it]);

    if (c == 1) {
        printf("%d\n", cnt_distinct);
    } else if (c == 2) {
        printf("%d\n", it + 1);
    } else {
        int maxim = 0, cnt = 0;
        for (int i = 0; i <= it; i++) {
            int len = get_len(v[i]);
            if (len > maxim) {
                maxim = len;
                cnt = 1;
            } else if (maxim == len) {
                cnt++;
            }
        }
```

```
        printf("%d %d\n", maxim, cnt);
    }

    return 0;
}
```

B.3 Clasa a VII-a

B.3.A Problema Microbuz

```
#include <bits/stdc++.h>

#define INF (INT_MAX >> 2)
#define K_MAX 10
#define S_MAX (K_MAX * (K_MAX + 1) / 2)
#define C_MAX 100

using namespace std;

ifstream fin("microbuz.in");
ofstream fout("microbuz.out");

int T;
int N;
int C[K_MAX + 5];
pair<int, int>
    D[S_MAX +
      5];  /// D[i] = perechea (cost, conf) unde cost reprezinta costul
           /// minim sa obtinem distanta i si conf reprezinta
           /// configuratia cu care acest cost a fost obtinut
bitset<C_MAX * K_MAX + 5> A[(1 << K_MAX) + 5];

pair<int, int> compute_conf(int conf) {
    int cost = 0;
    int dist = 0;
    for (int i = 0; i < K_MAX; i++) {
        if (conf & (1 << i)) {
            dist += i + 1;
            cost += C[i];
        }
    }
    return make_pair(cost, dist);
}

int main() {
    fin >> T;
    for (int i = 0; i < K_MAX; i++) {
        fin >> C[i];
    }
    fin >> N;
```

```
    if (T <= 2) {
        for (int i = 1; i <= S_MAX; i++) {
            D[i].first = INF;
        }

        for (int conf = 0; conf < (1 << K_MAX); conf++) {
            int cost = 0, dist = 0;
            tie(cost, dist) = compute_conf(conf);
            if (D[dist].first > cost) {
                D[dist] = make_pair(cost, conf);
            }
        }

        int ans = INF;
        int ans_conf[4];
        for (int dist_1 = 0; dist_1 <= S_MAX; dist_1++) {
            for (int dist_2 = 0; dist_2 <= S_MAX && dist_1 + dist_2 <= N;
                 dist_2++) {
                int dist_3 = N - dist_1 - dist_2;
                if (dist_3 > S_MAX) {
                    continue;
                }
                if (ans > D[dist_1].first + D[dist_2].first +
                              D[dist_3].first) {
                    ans = D[dist_1].first + D[dist_2].first +
                          D[dist_3].first;
                    ans_conf[1] = D[dist_1].second;
                    ans_conf[2] = D[dist_2].second;
                    ans_conf[3] = D[dist_3].second;
                }
            }
        }

        if (T == 1) {
            fout << ans << "\n";
        } else {
            for (int i = 1; i <= 3; i++) {
                for (int j = 0; j < K_MAX; j++) {
                    if (ans_conf[i] & (1 << j)) {
                        fout << j + 1 << " " << C[j] << "\n";
                    }
                }
            }
        }
    } else {
        for (int conf = 1; conf < (1 << K_MAX); conf++) {
            int cost = 0, dist = 0;
            tie(cost, dist) = compute_conf(conf);
            for (int i = 0; i < K_MAX; i++) {
                if (conf & (1 << i)) {
                    A[conf] |= A[conf ^ (1 << i)];
                }
            }
```

```
            A[conf][cost] = 1;
        }

        int ans = 0, ans_conf = 0;
        for (int conf = 1; conf < (1 << K_MAX) - 1; conf++) {
            int cost = 0, dist = 0;
            tie(cost, dist) = compute_conf(conf);
            int flip = ((1 << K_MAX) - 1) ^ conf;
            if (A[flip][cost]) {
                if (cost > ans) {
                    ans = cost;
                    ans_conf = conf;
                }
            }
        }

        fout << ans << "\n";
        for (int i = 0; i < K_MAX; i++) {
            if (ans_conf & (1 << i)) {
                fout << i + 1 << " ";
            }
        }
        fout << "\n";

        int flip = ((1 << K_MAX) - 1) ^ ans_conf;
        for (int conf = 1; conf <= flip; conf++) {
            if ((conf & flip) ==
                conf) {  /// conf este submultime a lui flip
                int cost = 0, dist = 0;
                tie(cost, dist) = compute_conf(conf);
                if (cost == ans) {
                    for (int i = 0; i < K_MAX; i++) {
                        if (conf & (1 << i)) {
                            fout << i + 1 << " ";
                        }
                    }
                    return 0;
                }
            }
        }
    }
    return 0;
}
```

B.3.B Problema Raza

```
/*
    Autor: prof. Alin Burta, Colegiul National "B.P. Hasdeu", Buzau
*/

#include <fstream>
#include <iostream>
```

```
#define FIN "raza.in"
#define FOU "raza.out"
#define SMAX 10001     // maximum de rovere
#define SecMax 100001  // maximum de secunde

using namespace std;

int Prd[SMAX];
int S;                      // numarul roverelor
int M;                      // numarul roverelor ramase
int Sec;                    // numar maxim secunde
int Unu[SMAX], Doi[SMAX];   // intersectia cu diagonale
int C1[SecMax], C2[SecMax];
int Cate[SecMax];

int main() {
    ifstream in(FIN);
    int i, j, M, x, y, r;
    int minim, maxim, aux, timp;
    int task;

    minim = SecMax + 1;
    maxim = -1;
    // citire date de intrare
    in >> task >> S >> Sec;
    M = 0;
    for (i = 1; i <= S; ++i) {
        in >> x >> y >> r;
        // verific daca se intersecteaza cu diagonala
        if (x <= y + r - 1 && x + r - 1 >= y) {
            M++;
            Prd[M] = 4 * r - 4;
            // calculez intersectiile fiecaei traiectorii cu diagonala
            if (x == y)  // varfurile pe diagonala
            {
                Unu[M] = 1;
                Doi[M] = 2 * r - 1;
            } else if (x > y)  // intersectie cu prima si a doua latura
            {
                Unu[M] = x - y + 1;
                Doi[M] = 2 * r + y - x - 1;
            } else  // intersectie cu celelalte doua laturi
            {
                Unu[M] =
                    2 * r + y - x -
                    1;  // 2 * r - 2 + y  + r - 1 - ( x + r - 1) + 1;
                Doi[M] = 4 * r - y + x - 3;
            }
            if (Unu[M] == Doi[M]) Doi[M] = 0;
            if (Unu[M] > maxim) maxim = Unu[M];
            if (Unu[M] < minim) minim = Unu[M];
            if (Doi[M] > maxim && Doi[M]) maxim = Doi[M];
            if (Doi[M] < minim && Doi[M]) minim = Doi[M];
```

```
        }
    }
    in.close();

    for (i = 1; i <= M; ++i) C1[i] = Unu[i], C2[i] = Doi[i];
    for (i = 1; i <= Sec; ++i) Cate[i] = 0;

    for (i = 1; i <= M; ++i) {
        for (j = C1[i]; j <= Sec; j += Prd[i]) Cate[j]++;
        if (Doi[i])
            for (j = C2[i]; j <= Sec; j += Prd[i]) Cate[j]++;
    }

    maxim = 0;
    timp = 0;
    for (i = 1; i <= Sec; ++i)
        if (maxim < Cate[i]) maxim = Cate[i], timp = i;

    ofstream out(FOU);
    if (task == 1)
        out << M << '\n';
    else
        out << maxim << " " << timp << '\n';
    out.close();

    return 0;
}
```

B.3.C Problema Text

```
/// Dana Lica - C.J. Ex. PH
#include <cassert>
#include <cstring>
#include <fstream>

using namespace std;

ifstream f("text.in");
ofstream g("text.out");

const int NMAX = 2e5 + 1;
const int INF = 1e9;

int P;

int T;
char V[NMAX];

int N, M;

int pal_Max = 0;
string Container, Sol;
```

```
int H[NMAX];
int K, Stack[NMAX];
int Left[NMAX], Right[NMAX];

void Read() {
    assert(f >> P);
    assert(P == 1 || P == 2 || P == 3);

    assert(f.get());

    bool spaces = 0;
    bool minn = 0;
    bool maxn = 0;
    bool en = 0;

    int TOTAL = 0;

    char Ch = 0;
    while (f.get(Ch)) {
        if (Ch == '\n') {
            en = 1;

            break;
        }

        ++TOTAL;

        assert(Ch == ' ' || (Ch >= 'a' && Ch <= 'z') ||
               (Ch >= 'A' && Ch <= 'Z'));

        spaces |= (Ch == ' ');
        minn |= (Ch >= 'a' && Ch <= 'z');
        maxn |= (Ch >= 'A' && Ch <= 'Z');

        if ((Ch >= 'a' && Ch <= 'z') || (Ch >= 'A' && Ch <= 'Z'))
            V[++T] = Ch;
    }

    assert(spaces && minn && maxn && en);
    assert(TOTAL >= 2 && TOTAL <= 2e5);

    return;
}

void Max_Pal(string S) {
    int Size = (int)S.size() - 1;

    /// LUNGIME == 1 (MODULO 2):
    for (int i = 1; i <= Size; ++i) {
        int lg = 1;
        int left = i - 1, right = i + 1;

        while (left >= 1 && right <= Size && S[left] == S[right])
```

```
            lg += 2, --left, ++right;

        string now;
        for (int pos = left + 1; pos < right; ++pos) now += S[pos];

        if (lg > pal_Max || (lg == pal_Max && now > Sol))
            pal_Max = lg, Sol = now;
    }
    ///

    /// LUNGIME == 0 (MODULO 2):
    for (int i = 1; i < Size; ++i)
        if (S[i] == S[i + 1]) {
            int lg = 2;
            int left = i - 1, right = i + 2;

            while (left >= 1 && right <= Size && S[left] == S[right])
                lg += 2, --left, ++right;

            string now;
            for (int pos = left + 1; pos < right; ++pos) now += S[pos];

            if (lg > pal_Max || (lg == pal_Max && now > Sol))
                pal_Max = lg, Sol = now;
        }
    ///

    return;
}

int MAX(int a, int b) { return ((a > b) ? a : b); }

int main() {
    Read();

    int Min = INF;
    for (int d1 = 1; d1 * d1 <= T; ++d1)
        if (T % d1 == 0) {
            int d2 = T / d1;

            if ((d2 - d1) < Min) Min = (d2 - d1), N = d1, M = d2;
        }

    assert(N > 1);

    char A[N + 1][M + 1];
    for (int i = 1; i <= N; ++i)
        for (int j = 1; j <= M; ++j) A[i][j] = V[((i - 1) * M + j)];

    if (P == 1) {
        for (int i = 1; i <= N; ++i) {
            for (int j = 1; j <= M; ++j) g << A[i][j];

            g << '\n';
```

```
        }

        return 0;
    }

    if (P == 2) {
        for (int i = 1; i <= N; ++i) {
            Container = "0";
            for (int j = 1; j <= M; ++j) Container.push_back(A[i][j]);

            Max_Pal(Container);
        }

        for (int j = 1; j <= M; ++j) {
            Container = "0";
            for (int i = 1; i <= N; ++i) Container.push_back(A[i][j]);

            Max_Pal(Container);
        }

        g << Sol << '\n';

        return 0;
    }

    int ans = 0;

    for (int i = 1; i <= N; ++i) {
        for (int j = 1; j <= M; ++j) {
            char X = A[i][j];
            if (X >= 'A' && X <= 'Z') X = tolower(X);

            if (X == 'a' || X == 'e' || X == 'i' || X == 'o' || X == 'u')
                ++H[j];
            else
                H[j] = 0;
        }

        K = 0;
        for (int j = 1; j <= M; ++j) {
            while (K && H[j] < H[Stack[K]]) Right[Stack[K]] = j, --K;

            Stack[++K] = j;
        }
        for (int x = 1; x <= K; ++x) Right[Stack[x]] = (M + 1);

        K = 0;
        for (int j = M; j >= 1; --j) {
            while (K && H[j] < H[Stack[K]]) Left[Stack[K]] = j, --K;

            Stack[++K] = j;
        }
        for (int x = 1; x <= K; ++x) Left[Stack[x]] = 0;
```

```
        for (int j = 1; j <= M; ++j) {
            int now = ((Right[j] - 1) - (Left[j] + 1) + 1);

            ans = MAX(ans, H[j] * now);
        }
    }

    g << ans << '\n';

    return 0;
}
```

B.4 Clasa a VIII-a

B.4.A Problema Proeminența

```
#include <stdio.h>
/*
autor         Paul Diac
idee          iteram toate inaltimile, retinem o varfurile in ordine
scrict descrescatoare si cele mai adanci vai de dupa ele un varf nou
adaugat elimina toate varfurile mai mici decat el, deci vom folosi o
stiva pentru varfuri in ordine pentru a evita alte cazuri, la inceput
calculam proeminenta la stanga, si apoi la dreapta dupa o oglindire la
final afisam minimul dintre cele doua proeminente calculate memory O(N)
time          O(N)
score         100
*/
#include <stdio.h>
#define NMax 1000005
long N, C, s;
long a[NMax];

long min_(long i, long j) {
    if (i < j) return i;
    return j;
}
long peak[NMax], valley[NMax],
    top = -1;  // stiva de varfuri "peak" si cele mai adanci vai "valley"
long prom[2][NMax];  // proeminenta la stanga varfului de pe pozita i
                     // este prom[0][i], si la dreapta este
                     // prom[0][n-1-i] datorita oglindirii

void popTop() {  // eliminam un varf din stiva, actualizam cea mai adanca
                 // vale din dreapta daca e cazul
    if (top < 0) {
        return;
    }
    if (top >= 1 && valley[top - 1] > valley[top]) {
        valley[top - 1] = valley[top];
```

```
    }
    top--;
}

int main() {
    freopen("proeminenta.in", "r", stdin);
    freopen("proeminenta.out", "w", stdout);

    scanf("%ld %ld", &C, &N);
    for (int i = 0; i < N; i++) {
        scanf("%ld", &a[i]);
        if (i > 0 && a[i] == a[i - 1]) {
            i--;
            N--;  // eliminam duplicatele
        }
    }
    if (C == 1) {
        for (int i = 1; i < N - 1; i++) {
            if (a[i - 1] < a[i] && a[i] > a[i + 1]) {
                s++;
            }  // este varf
        }
        printf("%ld\n", s);
        return 0;
    }

    for (int d = 0; d <= 1;
         d++) {  // d=0 va fi parcurgerea stanga-dreapta, si la d=1 dupa
                 // oglindire, adica dreapta-stanga
        for (int i = 1; i < N - 1; i++) {
            if (a[i - 1] < a[i] && a[i] > a[i + 1]) {  // este varf
                prom[d][i] = a[i];
                while (top >= 0 &&
                       a[peak[top]] <= a[i]) {  // elimim varfurile mai
                                                // mici din stiva pe rand
                    popTop();
                }
                if (top >= 0 && a[peak[top]] > a[i]) {
                    prom[d][i] = a[i] - valley[top];
                }  // am gasit primul varf mai mare din stanga
                peak[++top] = i;
                valley[top] = a[i];  // adaugam noul varf
            }
            if (top >= 0 && valley[top] > a[i]) {
                valley[top] = a[i];
            }
        }
        for (int i = 0; i < N / 2; i++) {  // oglindim altitudinile
            long aux = a[i];
            a[i] = a[N - 1 - i];
            a[N - 1 - i] = aux;
        }
        top = -1;
    }
```

```
    for (int i = 1; i < N - 1; i++)
        if (a[i - 1] < a[i] && a[i] > a[i + 1]) {
            if (prom[0][i] >
                prom[1][N - 1 - i]) {  // retinem maximul in prom[0][i]
                prom[0][i] = prom[1][N - 1 - i];
            }
        }

    for (int i = 1; i < N - 1; i++) {
        if (prom[0][i] > 0) {
            printf("%ld ", prom[0][i]);
        }  // si il afisam
    }

    printf("\n");
    fclose(stdout);
    return 0;
}
```

B.4.B Problema RGB

```
#include <fstream>

using namespace std;

ifstream fin("rgb.in");
ofstream fout("rgb.out");

const int NMAX = 500000;
int r[NMAX + 5], g[NMAX + 5], b[NMAX + 5];

int solve_task_1(int normal[], int weak[], int strong[]) {
    int p = normal[normal[0]], ans = normal[0] - 1;

    for (int w = 1; w <= weak[0]; ++w) {
        if (2 * p < weak[w]) {
            break;
        }
        ++ans;
    }

    for (int s = 1; s <= strong[0]; ++s) {
        if (p < 2 * strong[s]) {
            break;
        }
        ++ans;
    }

    return ans;
}

void solve_task_2(int normal[], int weak[], int strong[]) {
```

```
    int s = 1, w = 1;
    for (int n = 1; n <= normal[0]; ++n) {
        while (s <= strong[0] && strong[s] * 2 < normal[n]) {
            ++s;
        }
        while (w <= weak[0] && weak[w] < 2 * normal[n]) {
            ++w;
        }
        fout << n - 1 + s - 1 + w - 1 << " ";
    }
    fout << "\n";
}

int main() {
    int t;
    fin >> t >> r[0] >> g[0] >> b[0];
    for (int i = 1; i <= r[0]; ++i) {
        fin >> r[i];
    }
    for (int i = 1; i <= g[0]; ++i) {
        fin >> g[i];
    }
    for (int i = 1; i <= b[0]; ++i) {
        fin >> b[i];
    }

    if (t == 1) {
        int w_r = solve_task_1(r, g, b);
        int w_g = solve_task_1(g, b, r);
        int w_b = solve_task_1(b, r, g);

        int ans = w_r, p = r[r[0]];
        if (w_g > ans || (w_g == ans && g[g[0]] < p)) {
            ans = w_g;
            p = g[g[0]];
        }
        if (w_b > ans || (w_b == ans && b[b[0]] < p)) {
            ans = w_b;
            p = b[b[0]];
        }

        fout << p << "\n";
    } else {
        solve_task_2(r, g, b);
        solve_task_2(g, b, r);
        solve_task_2(b, r, g);
    }

    fin.close();
    fout.close();

    return 0;
}
```

B.4.C Problema Subșir

```
#include <fstream>
using namespace std;
ifstream fin("subsir.in");
ofstream fout("subsir.out");
int C, N, S[10005], poz[10005][10], M, Q, x, p, a, b, nr, nw, w[12];
struct pereche {
    int poz, s;
} v[10000005];
int main() {
    fin >> C >> N;
    for (int i = 1; i <= N; i++) {
        fin >> S[i];
    }
    for (int j = 0; j <= 9; j++) {
        poz[N + 1][j] = N + 1;
        poz[N + 2][j] = N + 1;
    }
    for (int i = N; i >= 1; i--) {
        for (int j = 0; j <= 9; j++) {
            poz[i][j] = poz[i + 1][j];
        }
        poz[i][S[i]] = i;
    }
    /// poz[x][cif] = prima pozitie din S[x;N] unde gasesc cif

    if (C == 1) {
        /// O(M)
        fin >> M;
        for (int i = 1; i <= M; i++) {
            fin >> x >> p;
            nw = 0;
            do {
                nw++;
                w[nw] = x % 10;
                x = x / 10;
            } while (x);
            int ok = 1;
            int j = 0;
            for (int k = nw; k >= 1; k--) {
                /// caut w[k] incepand cu S[j]
                j = poz[j + 1][w[k]];
                if (j > p) {
                    ok = 0;
                    break;
                }
            }
            fout << ok << "\n";
        }
    } else {
        /// cazul C==2
```

```
        for (int i = 0; i <= 9; i++) {
            v[i].poz = poz[1][i];
            if (v[i].poz == N + 1)
                v[i].s = 0;
            else
                v[i].s = 1;
            if (i > 0) v[i].s += v[i - 1].s;
        }
        for (int i = 1; i < 1000000; i++) {
            p = v[i].poz;
            x = i * 10;
            for (int j = 0; j <= 9; j++) {
                v[x].poz = poz[p + 1][j];
                if (v[x].poz <= N) {
                    v[x].s = v[x - 1].s + 1;
                } else {
                    v[x].s = v[x - 1].s;
                }
                x++;
            }
        }

        /// O(Q)
        fin >> Q;
        for (int i = 1; i <= Q; i++) {
            fin >> a >> b;
            nr = 0;
            if (a > 0)
                nr = v[b].s - v[a - 1].s;
            else
                nr = v[b].s;
            fout << nr << "\n";
        }
    }
    fin.close();
    fout.close();
    return 0;
}
```

B.5 Clasa a IX-a

B.5.A Problema Colibri

```
/* Autor: Andrei Arhire, Universitatea A.I. Cuza, Iasi
 * Complexitate: O(NlogN)
 */
#include <bits/stdc++.h>

using namespace std;
```

```
struct fraction {
    int b, c, i;

    fraction(int b, int c, int i) {
        this->b = b;
        this->c = c;
        this->i = i;
    }
};

vector<fraction> listsOfCategories[8];
vector<int> fractions;
int n;

/*
 * (-oo, 1) - category I
 * [-1, -1] - category II
 * (-1,  0) - category III
 * [0,   0] - category IV
 * (0,   1) - category V
 * [1,   1] - category VI
 * (1, +oo) - category VII
 */

bool checkFirstCase() {
    for (auto it : listsOfCategories[7]) {
        fractions.emplace_back(it.i);
    }
    for (int i = 0; i + 1 < listsOfCategories[1].size(); i += 2) {
        fractions.emplace_back(listsOfCategories[1][i].i);
        fractions.emplace_back(listsOfCategories[1][i + 1].i);
    }
    if (listsOfCategories[1].size() % 2 == 1) {
        if (!listsOfCategories[2].empty()) {
            fractions.emplace_back(listsOfCategories[1].back().i);
            fractions.emplace_back(listsOfCategories[2].back().i);
            return true;
        }
        if (!listsOfCategories[3].empty() &&
            1LL * listsOfCategories[1].back().b *
                    listsOfCategories[3][0].b >
                1LL * listsOfCategories[1].back().c *
                    listsOfCategories[3][0].c) {
            fractions.emplace_back(listsOfCategories[1].back().i);
            fractions.emplace_back(listsOfCategories[3][0].i);
        }
    }
    return !fractions.empty();
}

bool checkSecondCase() {
    if (!listsOfCategories[6].empty()) {
        fractions.emplace_back(listsOfCategories[6].back().i);
```

```
        return true;
    }
    if (listsOfCategories[2].size() > 1) {
        fractions.emplace_back(listsOfCategories[2][0].i);
        fractions.emplace_back(listsOfCategories[2][1].i);
        return true;
    }
    if (!listsOfCategories[1].empty() && !listsOfCategories[3].empty()) {
        auto fr1 = listsOfCategories[1][0];
        auto fr2 = listsOfCategories[3][0];
        if (1LL * fr2.b * fr1.b == 1LL * fr1.c * fr2.c) {
            fractions.emplace_back(fr1.i);
            fractions.emplace_back(fr2.i);
            return true;
        }
    }
    return false;
}

bool checkThirdCase() {
    long long numerator = -1, denominator = -1;
    if (!listsOfCategories[5].empty()) {
        numerator = listsOfCategories[5].back().b;
        denominator = listsOfCategories[5].back().c;
        fractions.emplace_back(listsOfCategories[5].back().i);
    }
    if (!listsOfCategories[1].empty() && !listsOfCategories[3].empty()) {
        auto fr1 = listsOfCategories[1][0];
        auto fr2 = listsOfCategories[3][0];
        long long currNumerator = 1LL * fr1.b * fr2.b;
        long long currDenominator = 1LL * fr1.c * fr2.c;
        if (numerator == -1 || 1LL * currNumerator * denominator >
                               1LL * numerator * currDenominator) {
            fractions.clear();
            fractions.emplace_back(fr1.i);
            fractions.emplace_back(fr2.i);
            return true;
        }
    }
    if (!listsOfCategories[2].empty() && !listsOfCategories[3].empty()) {
        long long currNumerator = listsOfCategories[3][0].b;
        long long currDenominator = listsOfCategories[3][0].c;
        if (numerator == -1 || 1LL * currNumerator * denominator >
                               1LL * numerator * currDenominator) {
            fractions.clear();
            fractions.emplace_back(listsOfCategories[2][0].i);
            fractions.emplace_back(listsOfCategories[3][0].i);
            return true;
        }
    }
    if (listsOfCategories[3].size() > 1) {
        long long currNumerator =
            1LL * listsOfCategories[3][0].b * listsOfCategories[3][1].b;
        long long currDenominator =
```

```
            1LL * listsOfCategories[3][0].c * listsOfCategories[3][1].c;
        if (numerator == -1 || 1LL * currNumerator * denominator >
                                1LL * numerator * currDenominator) {
            fractions.clear();
            fractions.emplace_back(listsOfCategories[3][0].i);
            fractions.emplace_back(listsOfCategories[3][1].i);
            return true;
        }
    }
    return !fractions.empty();
}

bool checkFourthCase() {
    for (int i = 4; i; --i) {
        if (!listsOfCategories[i].empty()) {
            fractions.emplace_back(listsOfCategories[i].back().i);
            return true;
        }
    }
    return false;
}

signed main() {
    ifstream in("colibri.in");
    ofstream out("colibri.out");
    int a, b, c, gcd;
    in >> n;
    for (int i = 1; i <= n; ++i) {
        in >> a >> b >> c;
        if (b != 0) {
            gcd = __gcd(b, c);
            b /= gcd;
            c /= gcd;
        }
        fraction current(b, c, i);
        if (b && a % 2 == 0 && b > c) {
            listsOfCategories[7].emplace_back(current);
        }
        if (b && a % 2 == 0 && b == c) {
            listsOfCategories[6].emplace_back(current);
        }
        if (b && a % 2 == 0 && b < c) {
            listsOfCategories[5].emplace_back(current);
        }
        if (b == 0) {
            listsOfCategories[4].emplace_back(current);
        }
        if (b && a % 2 == 1 && b < c) {
            listsOfCategories[3].emplace_back(current);
        }
        if (b && a % 2 == 1 && b == c) {
            listsOfCategories[2].emplace_back(current);
        }
        if (b && a % 2 == 1 && b > c) {
```

```
            listsOfCategories[1].emplace_back(current);
        }
    }

    for (int i = 1; i <= 7; ++i) {
        if (i < 4) {
            sort(listsOfCategories[i].begin(),
                 listsOfCategories[i].end(),
                 [](const fraction i, const fraction j) {
                     return 1ll * i.b * j.c > 1LL * j.b * i.c;
                 });
        } else {
            sort(listsOfCategories[i].begin(),
                 listsOfCategories[i].end(),
                 [](const fraction i, const fraction j) {
                     return 1ll * i.b * j.c < 1LL * j.b * i.c;
                 });
        }
    }

    checkFirstCase() || checkSecondCase() || checkThirdCase() ||
        checkFourthCase();

    string sol(n, '0');
    for (auto it : fractions) {
        sol[it - 1] = '1';
    }
    out << sol << '\n';
    in.close();
    out.close();
    return 0;
}
```

B.5.B Problema Geogra

```
// Gheorghe Liviu Armand
// 100 de puncte
#include <bits/stdc++.h>
using namespace std;
ifstream f("geogra.in");
ofstream g("geogra.out");
long long n, z[100002][32], r[100002][32], nrr[100002], nr[100002],
    ri[100002], t[32], rr[32], sol[100002], sola[100002], solb[100002],
    s[100002], p[100002], zz[100002], jj;
long long x[100002], y[100002], w[100002];
long long a[32][100002], nrz[32][100002];

inline bool cmp(const long long &aa, const long long &bb) {
    return nr[aa] < nr[bb];
}

inline bool cmp2(const long long &aa, const long long &bb) {
```

```
    if (nr[aa] == nr[bb]) return nrr[aa] < nrr[bb];
    return nr[aa] < nr[bb];
}

inline bool comp(const long long &aa, const long long &bb) {
    return nrz[jj][aa] < nrz[jj][bb];
}

inline bool cmpx(const long long &aa, const long long &bb) {
    return x[aa] < x[bb];
}

inline bool cmpy(const long long &aa, const long long &bb) {
    return y[aa] < y[bb];
}

long long caut_bin_st(long long niv, long long val) {
    long long st = 1, dr = n, mij;
    while (st <= dr) {
        mij = (st + dr) / 2;
        if (val <= nrz[niv][a[niv][mij]])
            dr = mij - 1;
        else
            st = mij + 1;
    }
    return dr + 1;
}

long long caut_bin_dr(long long niv, long long val) {
    long long st = 1, dr = n, mij;
    while (st <= dr) {
        mij = (st + dr) / 2;
        if (val >= nrz[niv][a[niv][mij]])
            st = mij + 1;
        else
            dr = mij - 1;
    }
    return st - 1;
}

int main() {
    long long c, l, i, q, j, k = 0, A, B, st, dr;
    f >> c >> n >> l;
    for (i = 1; i <= n; i++) {
        f >> x[i] >> y[i];
        if (c == 2) f >> w[i];
        for (j = 1; j <= l; j++) f >> z[i][j];
    }
    f >> q;
    for (i = 1; i <= q; i++) {
        for (j = 1; j <= l; j++)
            f >> r[i][j],
                nr[i] +=
                (r[i][j] !=
```

```
                -1);  // numaram cate valori sunt diferite de -1
        ri[i] = i;
    }
    sort(ri + 1, ri + q + 1,
         cmp);  // sortam rundele dupa nr valorilor diferite de -1 pentru
                // a putea calcula sirul T
    for (j = 1; j <= l; j++)
        if (r[ri[1]][j] != -1) t[++k] = j;  // adugam pozitii in sirul T
    for (i = 1; i < q; i++)
        if (nr[ri[i]] != nr[ri[i + 1]]) {
            for (j = 1; j <= l; j++)
                if (r[ri[i]][j] == -1 && r[ri[i + 1]][j] != -1)
                    t[++k] = j;  // adugam pozitii in sirul T
        }
    for (j = 1; j <= l; j++)
        if (r[ri[q]][j] == -1) t[++k] = j;  // adugam pozitii in sirul T
    for (i = 1; i <= q; i++) {
        for (j = 1; j <= l; j++)
            rr[j] =
                r[i][t[j]];  // schimbam sirurile R in functie de sirul T
        for (j = 1; j <= l; j++) r[i][j] = rr[j];
    }
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= l; j++)
            zz[j] =
                z[i][t[j]];  // schimbam sirurile Z in functie de sirul T
        for (j = 1; j <= l; j++) z[i][j] = zz[j];
    }
    for (i = 1; i <= q; i++) {
        j = 1;
        while (j <= l && r[i][j] != -1) {
            nrr[i] *= 2;
            nrr[i] +=
                r[i][j];  // calculam numerele formate din valorile
                          // diferite de -1 din fiecare sir R (adica nr
                          // formate din primii cativa biti)
            j++;
        }
        ri[i] = i;
    }
    for (i = 1; i <= n; i++) {
        a[0][i] = i;
        nrz[0][i] = 0;
        for (j = 1; j <= l; j++) {
            nrz[j][i] =
                nrz[j - 1][i] * 2;  // pentru fiecare sir Z calculam
                                    // numerele formate din primii j biti
            nrz[j][i] += z[i][j];
            a[j][i] = i;
        }
    }
    for (j = 1; j <= l; j++)  // nu mai e nevoie sort pt 0 :)
    {
        jj = j;
```

```
        sort(a[j] + 1, a[j] + n + 1,
             comp);  // sortam fiecare nivel j, sortam locatiile dupa
                     // numarul format din primii j biti din sirul Z
                     // asociat fiecarei locatii
    }
    sort(ri + 1, ri + q + 1,
         cmp2);  // ne sortam rundele dupa nr de valori diferite de -1 si
                 // in caz de egalitate dupa nr format din acesti cativa
                 // biti, pentru a putea vedea daca sirurile R a doua
                 // runde coincid(ca sa nu calculam de 2 ori pentru
                 // acelasi sir R)
    for (i = 1; i <= q; i++) {
        st = caut_bin_st(
            nr[ri[i]],
            nrr[ri[i]]);  // cautam binar capatul stanga al intervalului
                          // format din locatiile ce respecta sirul R
                          // actual
        dr = caut_bin_dr(
            nr[ri[i]], nrr[ri[i]]);  // cautam binar capatul dreapta al
                                     // intervalului format din locatiile
                                     // ce respecta sirul R actual
        if (c == 1) {
            k = dr - st + 1;  // raspunsul pentru cerinta 1 este lungimea
                              // intervalului
            sol[ri[i]] = k;
        } else {
            k = 0;
            for (j = st; j <= dr; j++)
                p[++k] =
                    a[nr[ri[i]]]
                     [j];  // ne punem locatiile din interval intr-un sir
                           // ca fie mai usor de codat in continuare
            sort(p + 1, p + k + 1,
                 cmpx);  // sortam locatiile dupa coordonata X
            for (j = 1; j <= k; j++)
                s[j] = s[j - 1] + w[p[j]];  // facem sume partiale dupa W
            for (j = 1; j <= k; j++)
                if (s[j] >=
                    s[k] / 2 +
                        s[k] % 2)  // daca aici se afla valoarea mediana
                {
                    A = x[p[j]];  // aceasta este coodonata X optima
                    break;
                }
            sort(p + 1, p + k + 1,
                 cmpy);  // sortam locatiile dupa coordonata Y
            for (j = 1; j <= k; j++)
                s[j] = s[j - 1] + w[p[j]];  // facem sume partiale dupa W
            for (j = 1; j <= k; j++)
                if (s[j] >=
                    s[k] / 2 +
                        s[k] % 2)  // daca aici se afla valoarea mediana
                {
                    B = y[p[j]];  // aceasta este coodonata Y optima
```

```
                    break;
                }
            sola[ri[i]] = A;
            solb[ri[i]] = B;
        }
        while (i < q && nr[ri[i]] == nr[ri[i + 1]] &&
               nrr[ri[i]] ==
                   nrr[ri[i + 1]])  // daca runda actuala are acelasi sir
                                    // R ca runda urmatoare, atunci nu
                                    // mai trebuie sa calculam si pentru
                                    // runda urmatoare, ci doar sa punem
                                    // solutia obtinuta in runda actuala
                                    // si in runda urmatoare
        {
            i++;
            if (c == 1)
                sol[ri[i]] = k;
            else
                sola[ri[i]] = A, solb[ri[i]] = B;
        }
    }
    for (i = 1; i <= q; i++) {
        if (c == 1)
            g << sol[i] << '\n';
        else
            g << sola[i] << " " << solb[i] << '\n';
    }
    return 0;
}
// YEY
```

B.5.C Problema Schi

```
/* Autor: Bogdan Iordache, Universitatea din Bucuresti
 * Complexitate: O(N)
 */
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int MAXN = 200005;
const int INF = 0x3f3f3f3f;

enum Dir { CUP = 2, CAP = 0, FULL = 1 };

int l[MAXN], h[MAXN];
Dir dir[MAXN];

ll height[MAXN];  // inaltimea fetei superioare a cubului i

int s_restrict[MAXN], k_restrict;
```

```
// mentinem sirul de cutii cu golul in sus care inca mai pot "acoperi"
// cutii de deasupra, alaturi de un maxim partial pe prefix (inaltimea
// maxima pana la care mai pot acoperi primele i cutii din acest sir)
pair<int, ll> s_cups[MAXN];
int k_cups;

// intervalul [seg_bot[i], seg_top[i]] descrie intervalul de inaltimi pe
// care cubui i este vizibil
ll seg_bot[MAXN], seg_top[MAXN];

// idem pentru cutiile cu golul in jos
pair<int, ll> s_caps[MAXN];
int k_caps;

ll restrict(int under_l, int under_h, Dir under_dir, int over_l,
            int over_h, Dir over_dir, ll curr_height) {
    /* In aceasta functie consideram o cutie `under` deja plasata in
     * turn, pentru care latura superioara (eventual capacul lipsa) se
     * afla la inaltimea `curr_height`. Incercam sa determinam o
     * restrictie H pentru o cutie noua (`over`) cu semnificatia ca
     * latura superioara a noii cutii nu poate fi mai jos decat H din
     * cauza cutiei `under`.*/

    if (over_dir == CUP) {
        if (under_dir == CUP) {
            if (under_l < over_l)
                return curr_height + over_h;
            else
                return curr_height - under_h + over_h;
        } else if (under_dir == CAP) {
            return curr_height + over_h;
        } else {
            return curr_height + over_h;
        }
    } else if (over_dir == CAP) {
        if (under_dir == CUP) {
            if (under_l < over_l)
                return curr_height;
            else
                return curr_height - under_h + over_h;
        } else if (under_dir == CAP) {
            if (under_l < over_l)
                return curr_height;
            else
                return curr_height + over_h;
        } else {
            if (under_l < over_l)
                return curr_height;
            else
                return curr_height + over_h;
        }
    } else {
        if (under_dir == CUP) {
```

```
            if (under_l < over_l)
                return curr_height + over_h;
            else
                return curr_height - under_h + over_h;
        } else if (under_dir == CAP) {
            return curr_height + over_h;
        } else {
            return curr_height + over_h;
        }
    }
}

int main() {
    ifstream cin("schi.in");
    ofstream cout("schi.out");

    int C;
    cin >> C;

    int N;
    cin >> N;
    for (int i = 1; i <= N; ++i) {
        int d;
        cin >> l[i] >> h[i] >> d;
        dir[i] = Dir(d);
    }

    /* Cerinta 1 */

    l[0] = INF;
    h[0] = 0;
    dir[0] = FULL;
    height[0] = 0;

    k_restrict = 1;
    s_restrict[1] = 0;

    for (int i = 1; i <= N; ++i) {
        ll max_h = h[i];  // restrictia calculata pentru podea
        while (l[s_restrict[k_restrict]] < l[i]) {
            int under = s_restrict[k_restrict];
            max_h =
                max(max_h, restrict(l[under], h[under], dir[under], l[i],
                                    h[i], dir[i], height[under]));
            k_restrict--;
        }
        int under = s_restrict[k_restrict];
        max_h = max(max_h, restrict(l[under], h[under], dir[under], l[i],
                                    h[i], dir[i], height[under]));
        height[i] = max_h;
        s_restrict[++k_restrict] = i;
    }

    ll h_max = *max_element(height + 1, height + N + 1);
```

```
    /* Cerinta 2 */

    for (int i = 1; i <= N; ++i) {
        seg_bot[i] = height[i] - h[i];
        seg_top[i] = height[i];
    }

    // actualizam intervalele de vizibilitate conform cutiilor cu golul
    // in sus
    k_cups = 0;
    for (int i = 1; i <= N; ++i) {
        while (k_cups > 0 && l[s_cups[k_cups].first] < l[i]) k_cups--;
        if (k_cups > 0)
            seg_bot[i] = max(seg_bot[i], s_cups[k_cups].second);
        if (dir[i] == CUP) {
            ll maxim = (k_cups == 0 ? 0 : s_cups[k_cups].second);
            maxim = max(maxim, height[i]);
            s_cups[++k_cups] = {i, maxim};
        }
    }

    // actualizam intervalele de vizibilitate conform cutiilor cu golul
    // in jos
    k_caps = 0;
    for (int i = N; i >= 1; --i) {
        while (k_caps > 0 && l[s_caps[k_caps].first] < l[i]) k_caps--;
        if (k_caps > 0)
            seg_top[i] = min(seg_top[i], s_caps[k_caps].second);
        if (dir[i] == CAP) {
            ll minim = (k_caps == 0 ? h_max : s_caps[k_caps].second);
            minim = min(minim, height[i] - h[i]);
            s_caps[++k_caps] = {i, minim};
        }
    }

    // numaram cate cutii au ramas vizibile dupa acoperiri
    int vis = 0;
    for (int i = 1; i <= N; ++i)
        if (seg_bot[i] < seg_top[i]) vis++;

    if (C == 1)
        cout << h_max;
    else
        cout << vis;
    cout << "\n";

    return 0;
}
```

B.6 Clasa a X-a

B.6.A Problema ChangeMin

```cpp
/**
Autor: Bogdan-Ioan Popa
Complexitate: O(N)
Scor: 100p
**/

#include <bits/stdc++.h>

#define INF 2000000000
#define MOD 666013
#define N_MAX 1000000
#define ll long long

using namespace std;

ifstream fin("changemin.in");
ofstream fout("changemin.out");

int T, N;
int A[N_MAX + 5];
int L;
int S[N_MAX + 5];

int main() {
    fin >> T >> N;
    for (int i = 1; i <= N; i++) {
        fin >> A[i];
        assert(A[i] >= 1 && A[i] <= 1000000000);
    }
    assert(T == 1 || T == 2);
    assert(1 <= N && N <= 1000000);

    ll cnt = 0;
    for (int i = N; i >= 1; i--) {
        while (L && A[S[L]] >= A[i]) {
            L--;
        }
        S[++L] = i;
        cnt += L;
    }

    if (T == 1) {
        fout << cnt << "\n";
        return 0;
    }
```

```
    L = 0;
    ll sum_all = 0, sum_coef = 0;
    int score = 0;
    for (int i = N; i >= 1; i--) {
        while (L && A[S[L]] >= A[i]) {
            sum_coef -= sum_all;
            sum_all -= A[S[L]];
            L--;
        }
        S[++L] = i;
        sum_all += A[i];
        sum_coef += sum_all;
        cnt -= L;
        score =
            (score + (1ll * cnt * sum_all) % MOD + sum_coef % MOD) % MOD;
    }
    fout << score << "\n";

    return 0;
}
```

B.6.B Problema Dragonfruit

```
/* Task: Dragon Fruit
 * Author: Tulba-Lecu Theodor-Gabriel
 * Time Complexity: O(T * N * S * K)
 * Approach: dynamic progamming
 */
#include <cassert>
#include <cstdio>
#include <cstring>
#include <string>
#include <vector>

#define INF 2140000000
#define MOD 1000000007

using namespace std;

pair<int, int> d[2][11][1001][2];

void join(pair<int, int> &s1, pair<int, int> s2, bool inc) {
    if (inc) {
        s2.first++;
    }

    if (s2.first < s1.first) {
        s1 = s2;
    } else if (s2.first == s1.first) {
        s1 = {s1.first, (s1.second + s2.second) % MOD};
    }
}
```

```
pair<int, int> solve(int n, int steps, int sum, vector<int> v) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j <= steps; j++) {
            for (int k = 0; k <= sum; k++) {
                d[i][j][k][0] = d[i][j][k][1] = {INF, 0};
            }
        }
    }

    // dp[i][j][k][0/1] = minimum number of fruits and number of ways to
    // obtain that if we consider only the first i elements from the
    // array and only take j sequences of contignuous elements and the
    // sum of the selected elements is k and 0 if no current segment is
    // selected and 1 is there is a current segment selected

    d[0][0][0][0] = {0,
                     1};  // one way to make 0 fruits and takes 0 cells
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= steps; j++) {
            for (int k = 0; k <= sum; k++) {
                d[i & 1][j][k][0] = d[i & 1][j][k][1] = {INF, 0};
                join(d[i & 1][j][k][0], d[(i - 1) & 1][j][k][0], 0);
                join(d[i & 1][j][k][0], d[(i - 1) & 1][j][k][1], 0);

                if (k >= v[i]) {
                    join(d[i & 1][j][k][1],
                         d[(i - 1) & 1][j][k - v[i]][1], 1);
                    if (j > 0) {
                        join(d[i & 1][j][k][1],
                             d[(i - 1) & 1][j - 1][k - v[i]][1], 1);
                        join(d[i & 1][j][k][1],
                             d[(i - 1) & 1][j - 1][k - v[i]][0], 1);
                    }
                }
            }
        }
    }

    // answer is the best of d[n][j][k][0/1] for any j from 0 to s
    pair<int, int> ans = {INF, 0};
    for (int j = 0; j <= steps; j++) {
        for (int bit = 0; bit < 2; bit++) {
            if (d[n & 1][j][sum][bit].first < ans.first) {
                ans = d[n & 1][j][sum][bit];
            } else if (d[n & 1][j][sum][bit].first == ans.first) {
                ans.second =
                    (ans.second + d[n & 1][j][sum][bit].second) % MOD;
            }
        }
    }

    if (ans.first == INF) {
        return {0, 0};
```

```
    }

    return ans;
}

int main() {
    freopen("dragonfruit.in", "r", stdin);
    freopen("dragonfruit.out", "w", stdout);

    int t, n, k, s;
    vector<int> v;

    assert(scanf("%d", &t) == 1);
    assert(1 <= t && t <= 5);

    while (t--) {
        assert(scanf("%d%d%d", &n, &k, &s) == 3);
        assert(1 <= n && n <= 100000);
        assert(1 <= k && k <= 1000);
        assert(1 <= s && s <= 10);
        v.resize(n + 1);
        for (int i = 1; i <= n; i++) {
            assert(scanf("%d", &v[i]) == 1);
            assert(0 <= v[i] && v[i] <= 1000);
        }

        pair<int, int> ans = solve(n, s, k, v);
        printf("%d %d\n", ans.first, ans.second);
    }

    return 0;
}
```

B.6.C Problema Munte

```
/* Task: Munte
 * Author: Tulba-Lecu Theodor-Gabriel
 * Time Complexity: O(N * log(N))
 * Approach: greedy
 * Asserted input for checking test validity
 */
#include <algorithm>
#include <cassert>
#include <cstdio>
#include <vector>

#define MOD 111181111LL
#define MAXN 200000

using namespace std;

bool check_mountain(vector<int> v) {
```

```
    int n = v.size();
    int pos = 0;
    for (int i = 0; i < n; i++) {
        if (v[i] == n) {
            pos = i;
            break;
        }
    }

    if (pos == 1 || pos == n) return false;

    for (int i = 1; i <= pos; i++) {
        if (v[i - 1] >= v[i]) return false;
    }

    for (int i = pos + 1; i < n; i++) {
        if (v[i - 1] <= v[i]) {
            return false;
        }
    }

    return true;
}

vector<int> get_vec(vector<pair<int, int>> p) {
    vector<int> v;
    for (int i = 0; i < (int)p.size(); i++) {
        if (p[i].first != 0) v.push_back(p[i].first);
    }
    for (int i = p.size() - 1; i >= 0; i--) {
        if (p[i].second != 0) v.push_back(p[i].second);
    }

    return v;
}

int main() {
    freopen("munte.in", "r", stdin);
    freopen("munte.out", "w", stdout);

    int n, c;
    int full_pairs;
    int partial_pairs;
    vector<pair<int, int>> pairs;
    vector<int> found;

    assert(scanf("%d%d", &c, &n) == 2);
    assert(c == 1 || c == 2);
    assert(3 <= n && n <= MAXN);

    found.resize(n + 1, 0);
    full_pairs = n / 2;
    partial_pairs = (n + 1) / 2;
    pairs.resize(partial_pairs + 1);
```

```
    for (int i = 1; i <= n; i++) {
        int x;
        assert(scanf("%d", &x) == 1);
        assert(found[x] == 0);
        assert(1 <= x && x <= n);
        if (i > partial_pairs)
            pairs[full_pairs + 1 - (i - partial_pairs)].second = x;
        else
            pairs[i].first = x;
    }

    for (int i = 1; i <= full_pairs; i++) {
        if (pairs[i].first > pairs[i].second) {
            swap(pairs[i].first, pairs[i].second);
        }
    }

    sort(pairs.begin() + 1, pairs.begin() + 1 + full_pairs);

    int ans = 2;
    if (check_mountain(get_vec(pairs))) {
        for (int i = 2; i <= full_pairs; i++) {
            if (pairs[i].first > pairs[i - 1].second) {
                ans = (ans * 2) % MOD;
            }
        }
    } else {
        ans = 0;
    }

    if (c == 2) {
        printf("%d\n", ans);
    } else {
        printf(ans == 0 ? "NU\n" : "DA\n");
    }

    return 0;
}
```

B.7 Clasele XI–XII

B.7.A Problema Lupușor și Mielu

```
// Andrei Constantinescu, ETH Zurich
// nlog
#include <bits/stdc++.h>

using namespace std;
```

```
const int MOD = 1000000000 + 7;
const int INV2 = 500000000 + 4;
const int INF = 2E9;

inline void AddMod(int &res, int a, int b) {
    res = a + b;
    if (res >= MOD) {
        res -= MOD;
    }
}

inline void MulMod(int &res, int a, int b) { res = (1LL * a * b) % MOD; }

inline void SubMod(int &res, int a, int b) {
    res = a - b;
    if (res < 0) {
        res += MOD;
    }
}

struct Change {
    int index, a, b;
};

struct Instance {
    vector<int> a, b;
    vector<Change> updates;

    Instance() {}
    Instance(vector<int> &&a_, vector<int> &&b_,
             vector<Change> &&updates_)
        : a(a_), b(b_), updates(updates_) {
        assert(a.size() == b.size());
    }

    int N() const { return a.size(); }
    int M() const { return updates.size(); }
};

Instance Read(istream &stream) {
    int N;
    stream >> N;
    vector<int> a(N), b(N);
    for (int i = 0; i < N; ++i) {
        stream >> a[i];
    }
    for (int i = 0; i < N; ++i) {
        stream >> b[i];
    }
    int M;
    stream >> M;
    vector<Change> updates(M);
    for (int i = 0; i < M; ++i) {
        stream >> updates[i].index >> updates[i].a >> updates[i].b;
```

```
    }
    return Instance(move(a), move(b), move(updates));
}

namespace task1 {
struct DataStructureSegmentTree {
    struct Node {
        int l, r;
        int maximum, lazy;
    };
    vector<Node> nodes;

    void Combine(int node) {
        nodes[node].maximum =
            max(nodes[2 * node].maximum, nodes[2 * node + 1].maximum);
    }
    void AddLazy(int node, int val) {
        nodes[node].maximum += val;
        if (nodes[node].l != nodes[node].r) {
            nodes[node].lazy += val;
        }
    }
    void Propagate(int node) {
        if (nodes[node].lazy != 0) {
            AddLazy(2 * node, nodes[node].lazy);
            AddLazy(2 * node + 1, nodes[node].lazy);
            nodes[node].lazy = 0;
        }
    }

    void Build(const vector<int> &v, int l, int r, int node) {
        if (l > r) return;
        nodes[node].l = l, nodes[node].r = r, nodes[node].lazy = 0;
        if (l == r) {
            nodes[node].maximum = v[l];
            return;
        }
        const int mid = (l + r) / 2;
        Build(v, l, mid, 2 * node);
        Build(v, mid + 1, r, 2 * node + 1);
        Combine(node);
    }

    DataStructureSegmentTree(const vector<int> &v_)
        : nodes(4 * v_.size() + 1) {
        Build(v_, 1, v_.size() - 1, 1);
    }

    void Update(int l, int r, int val, int node) {
        if (l == nodes[node].l && r == nodes[node].r) {
            AddLazy(node, val);
            return;
        }
        Propagate(node);
```

```
        const int mid = (nodes[node].l + nodes[node].r) / 2;
        if (r <= mid) {
            Update(l, r, val, 2 * node);
        } else if (l > mid) {
            Update(l, r, val, 2 * node + 1);
        } else {
            Update(l, mid, val, 2 * node);
            Update(mid + 1, r, val, 2 * node + 1);
        }
        Combine(node);
    }

    void AddVal(int l, int r, int val) {
        if (l > r) return;
        Update(l, r - 1, val, 1);
    }
    int QueryMax() { return nodes[1].maximum; }
};

template <typename DataStructure>
vector<int> SolveMaxNNorNlog(Instance inst) {
    const int N = inst.N();
    const int M = inst.M();
    vector<int> &a = inst.a;
    vector<int> &b = inst.b;
    const vector<Change> &updates = inst.updates;

    vector<int> s_part(2 * N + 2 * M + 1);
    for (int j = 0; j < N; ++j) {
        if (a[j] < b[j]) {
            ++s_part[a[j]];
            --s_part[b[j]];
        }
    }
    for (int j = 1; j <= 2 * N + 2 * M; ++j) {
        s_part[j] += s_part[j - 1];
    }

    DataStructure ds(move(s_part));
    vector<int> ans(M + 1);
    for (int i = 0; i <= M; ++i) {
        if (i > 0) {
            ds.AddVal(a[updates[i - 1].index - 1],
                      b[updates[i - 1].index - 1], -1);
            a[updates[i - 1].index - 1] = updates[i - 1].a;
            b[updates[i - 1].index - 1] = updates[i - 1].b;
            ds.AddVal(a[updates[i - 1].index - 1],
                      b[updates[i - 1].index - 1], 1);
        }
        ans[i] = N - ds.QueryMax();
        if (ans[i] == N) {
            ans[i] = -1;  // Can not make happen.
        }
    }
```

```
    return ans;
}

}  // namespace task1

namespace task2 {

struct DataStructureSegmentTree {
    vector<bool> active;

    struct Node {
        int l, r;
        int sum, sum_active, lazy;
    };
    vector<Node> nodes;

    void Combine(int node) {
        AddMod(nodes[node].sum, nodes[2 * node].sum,
               nodes[2 * node + 1].sum);
        AddMod(nodes[node].sum_active, nodes[2 * node].sum_active,
               nodes[2 * node + 1].sum_active);
    }

    void AddLazy(int node, int val) {
        MulMod(nodes[node].sum, val, nodes[node].sum);
        MulMod(nodes[node].sum_active, val, nodes[node].sum_active);
        if (nodes[node].l != nodes[node].r) {
            MulMod(nodes[node].lazy, val, nodes[node].lazy);
        }
    }
    void Propagate(int node) {
        if (nodes[node].lazy > 1) {
            AddLazy(2 * node, nodes[node].lazy);
            AddLazy(2 * node + 1, nodes[node].lazy);
            nodes[node].lazy = 1;
        }
    }

    void Build(const vector<int> &v, int l, int r, int node) {
        if (l > r) return;
        nodes[node].l = l, nodes[node].r = r, nodes[node].lazy = 1;
        if (l == r) {
            nodes[node].sum = v[l];
            nodes[node].sum_active = active[l] * v[l];
            return;
        }
        const int mid = (l + r) / 2;
        Build(v, l, mid, 2 * node);
        Build(v, mid + 1, r, 2 * node + 1);
        Combine(node);
    }

    DataStructureSegmentTree(const vector<int> &v_,
                             vector<bool> &&active_)
```

```
        : active(active_), nodes(4 * active_.size() + 1) {
        assert(v_.size() == active.size());
        Build(v_, 1, active.size() - 1, 1);
    }

    void Toggle(int index, int node) {
        if (nodes[node].l == nodes[node].r) {
            active[nodes[node].l] = active[nodes[node].l] ^ 1;
            nodes[node].sum_active =
                active[nodes[node].l] * nodes[node].sum;
            return;
        }
        Propagate(node);
        const int mid = (nodes[node].l + nodes[node].r) / 2;
        if (index <= mid) {
            Toggle(index, 2 * node);
        } else {
            Toggle(index, 2 * node + 1);
        }
        Combine(node);
    }

    void Update(int l, int r, int val, int node) {
        if (l == nodes[node].l && r == nodes[node].r) {
            AddLazy(node, val);
            return;
        }
        Propagate(node);
        const int mid = (nodes[node].l + nodes[node].r) / 2;
        if (r <= mid) {
            Update(l, r, val, 2 * node);
        } else if (l > mid) {
            Update(l, r, val, 2 * node + 1);
        } else {
            Update(l, mid, val, 2 * node);
            Update(mid + 1, r, val, 2 * node + 1);
        }
        Combine(node);
    }

    void Add(int l, int r) {
        if (l > r) return;
        Toggle(l, 1);
        Update(l, r, 2, 1);
    }
    void Remove(int l, int r) {
        if (l > r) return;
        Toggle(l, 1);
        Update(l, r, INV2, 1);
    }
    int Query() { return nodes[1].sum_active; }
};

template <typename DataStructure>
```

```
vector<int> SolveCountNNorNlog(Instance inst) {
    const int N = inst.N();
    const int M = inst.M();
    vector<int> &a = inst.a;
    vector<int> &b = inst.b;
    const vector<Change> &updates = inst.updates;

    // Values are between 1 and 2 * N + 2 * M.
    vector<int> aux(2 * N + 2 * M + 2, 1);
    vector<bool> active(2 * N + 2 * M + 2, false);
    for (int i = 0; i < N; ++i) {
        if (a[i] < b[i]) {
            active[a[i]] = true;
            AddMod(aux[a[i]], aux[a[i]], aux[a[i]]);
            MulMod(aux[b[i] + 1], INV2, aux[b[i] + 1]);
        }
    }
    for (int j = 1; j <= 2 * N + 2 * M; ++j) {
        MulMod(aux[j], aux[j - 1], aux[j]);
    }

    DataStructure ds(move(aux), move(active));
    vector<int> ans(M + 1);
    for (int i = 0; i <= M; ++i) {
        if (i > 0) {
            ds.Remove(a[updates[i - 1].index - 1],
                      b[updates[i - 1].index - 1]);
            a[updates[i - 1].index - 1] = updates[i - 1].a;
            b[updates[i - 1].index - 1] = updates[i - 1].b;
            ds.Add(a[updates[i - 1].index - 1],
                   b[updates[i - 1].index - 1]);
        }
        MulMod(ans[i], INV2, ds.Query());
    }
    return ans;
}

}  // namespace task2

int main() {
    ifstream fin("lupusor.in");
    ofstream fout("lupusor.in");
    int task_id;
    fin >> task_id;
    const Instance inst = Read(fin);
    vector<int> sol;
    if (task_id == 1) {
        sol = task1::SolveMaxNNorNlog<task1::DataStructureSegmentTree>(
            inst);
    } else {
        sol = task2::SolveCountNNorNlog<task2::DataStructureSegmentTree>(
            inst);
    }
    for (int i = 0; i < sol.size(); ++i) {
```

```
        fout << sol[i] << '\n';
    }
    return 0;
}
```

B.7.B Problema Regate și Alianțe

```
// Stelian Chichirim
// O((N + M) * log*(N + M) + (N + M) * log(N + M))
// Expected: 100p

#include <bits/stdc++.h>

using namespace std;

const int Nmax = 2e5, Mmax = 2e5, Cmax = 1e9, Rmax = 1e9;

struct edge {
    int x, c, pos;
};

struct bridge {
    int x, y, c;
};

int lvl[Nmax + 5], lvlMin[Nmax + 5], comp[Nmax + 5], root[Nmax + 5],
    cnt[Nmax + 5], where[Nmax + 5], nrcomp;
bool isBridge[Nmax + 5];
long long val[Nmax + 5], ans[Nmax + 5];
vector<edge> g[Nmax + 5];
vector<int> v[Nmax + 5];
vector<bridge> bridges;

void getBridges(int nod, int parent) {
    for (edge& vec : g[nod])
        if (lvl[vec.x] == 0) {
            lvl[vec.x] = lvlMin[vec.x] = lvl[nod] + 1;
            getBridges(vec.x, nod);
            if (lvlMin[vec.x] == lvl[vec.x]) {
                isBridge[vec.pos] = true;
                bridges.push_back({nod, vec.x, vec.c});
            }
            lvlMin[nod] = min(lvlMin[nod], lvlMin[vec.x]);
        } else if (vec.x != parent)
            lvlMin[nod] = min(lvlMin[nod], lvl[vec.x]);
}

void dfs(int nod) {
    comp[nod] = nrcomp;
    cnt[nrcomp]++;
    for (edge& vec : g[nod])
        if (!isBridge[vec.pos] && !comp[vec.x]) dfs(vec.x);
```

```
}

int getRoot(int x) {
    int y = x;
    while (y != root[y]) y = root[y];
    while (root[x] != y) {
        int aux = root[x];
        root[x] = y;
        x = aux;
    }
    return y;
}

void unite(int x, int y, int cst) {
    x = getRoot(x);
    y = getRoot(y);
    if (x == y) return;
    if (cnt[x] > cnt[y]) swap(x, y);
    v[y].push_back(x);
    val[y] += 1LL * cst * cnt[x];
    val[x] += 1LL * cst * cnt[y] - val[y];
    root[x] = y;
    cnt[y] += cnt[x];
}

void propagate(int nod, int parent) {
    for (int& vec : v[nod])
        if (vec != parent) {
            val[vec] += val[nod];
            propagate(vec, nod);
        }
}

int main() {
    ifstream in("regate.in");
    ofstream out("regate.out");
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int N, M, x, y, c;
    in >> N >> M;
    assert(1 <= N && N <= Nmax);
    assert(1 <= M && M <= Mmax);

    for (int i = 1; i <= N; ++i) {
        in >> x;
        assert(1 <= x && x <= Rmax);
        bridges.push_back({-1, i, x});
    }

    for (int i = 1; i <= M; ++i) {
        in >> x >> y >> c;
        assert(1 <= x && x <= N);
        assert(1 <= y && y <= N);
        assert(x != y);
```

```
        assert(1 <= c && c <= Cmax);
        g[x].push_back({y, c, i});
        g[y].push_back({x, c, i});
    }

    // find bridges
    lvl[1] = lvlMin[1] = 1;
    getBridges(1, 0);

    // find components
    for (int i = 1; i <= N; ++i)
        if (!comp[i]) {
            nrcomp++;
            cnt[nrcomp] = 0;
            root[nrcomp] = nrcomp;
            dfs(i);
        }

    sort(bridges.begin(), bridges.end(),
         [](const bridge& e1, const bridge& e2) -> bool {
             if (e1.c == e2.c) return e1.x < e2.x;
             return e1.c > e2.c;
         });
    for (bridge& e : bridges) {
        if (e.x == -1) {
            int rt = getRoot(comp[e.y]);
            where[e.y] = rt;
            ans[e.y] = 1LL * e.c * (cnt[rt] - 1) - val[rt];
        } else
            unite(comp[e.x], comp[e.y], e.c);
    }

    propagate(getRoot(1), 0);
    for (int i = 1; i <= N; ++i) {
        ans[i] += val[where[i]];
        out << ans[i] << "\n";
    }
    return 0;
}
```

B.7.C Problema Schema și Investițiile

```
// Tinca - permutare
// O(N*G)
// Expected 100
#include <bits/stdc++.h>

const int MAX_N = 2000;
const int MAX_S = 5000;

bool sePoate[1 + MAX_N][1 + MAX_S];
int sp[1 + MAX_N + 1];
```

```
int a[1 + MAX_N];

bool cmp(int a, int b) { return a > b; }

int main() {
    int N, G;

    FILE *fin = fopen("schema.in", "r");
    FILE *fout = fopen("schema.out", "w");

    fscanf(fin, "%d%d", &N, &G);

    for (int i = 1; i <= N; ++i) fscanf(fin, "%d", &a[i]);

    std::sort(a + 1, a + 1 + N, cmp);

    for (int i = N; i >= 1; --i) sp[i] = sp[i + 1] + a[i];

    int res = G - sp[1];

    sePoate[0][0] = true;

    for (int i = 1; i <= N; i++) {
        for (int j = 0; j <= G; j++) {
            if (j - a[i] >= 0) sePoate[i][j] |= sePoate[i - 1][j - a[i]];
            sePoate[i][j] |= sePoate[i - 1][j];
        }

        int ssmall = sp[i + 1];

        for (int j = 0; j <= G; ++j) {
            int stotal = G - ssmall - j;
            if (0 <= stotal && stotal < a[i] && sePoate[i - 1][j] &&
                stotal > res)
                res = std::max(res, stotal);
        }
    }

    fprintf(fout, "%d", res);

    return 0;
}
```

Capitolul C Proba de Baraj

C.1 Juniori

C.1.A Problema Autostradă

```
/* Task - autostrada
 * Author: Tulba-Lecu Theodor-Gabriel
 * Complexity: O(N^2 + K)
 * Approach, simulate with difference array the path and
 * encode each cellin binary 1 - N, 2 - E, 4 - S, 8 - W
 */

#include <iostream>

#define MAXN 2005

using namespace std;

int n, k, xs, ys, cz, ct, cp, type;
int dir, p;
int mat[MAXN][MAXN][4];

int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};

void add_dir(int x, int y, int d, int len) {
    mat[x][y][d]++;
    mat[x + dx[d] * len][y + dy[d] * len][d]--;

    mat[x][y][d ^ 2]--;
    mat[x + dx[d] * len][y + dy[d] * len][d ^ 2]++;
}

int bit_count(int x, int y) {
    int cnt = 0;
    for (int i = 0; i < 4; i++) {
        if (mat[x][y][i] > 0) {
            cnt++;
        }
```

```
    }
    return cnt;
}

int main() {
    freopen("autostrada.in", "r", stdin);
    freopen("autostrada.out", "w", stdout);

    scanf("%d %d %d %d %d %d %d", &n, &k, &xs, &ys, &cz, &ct, &cp);

    for (int step = 0; step < k; step++) {
        scanf("%d %d", &dir, &p);

        if (xs + dx[dir] * p < 1 || xs + dx[dir] * p > n ||
            ys + dy[dir] * p < 1 || ys + dy[dir] * p > n) {
            printf("TRASEU INVALID\n%d", step + 1);
            return 0;
        }

        add_dir(xs, ys, dir, p);
        xs += dx[dir] * p;
        ys += dy[dir] * p;
    }

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            mat[i][j][1] += mat[i][j - 1][1];
            mat[i][j][2] += mat[i - 1][j][2];
        }
    }

    for (int i = n; i > 0; i--) {
        for (int j = n; j > 0; j--) {
            mat[i][j][0] += mat[i + 1][j][0];
            mat[i][j][3] += mat[i][j + 1][3];
        }
    }

    int cntz = 0, cntt = 0, cntp = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            int cnt = bit_count(i, j);
            if (cnt == 4) {
                cntp++;
            } else if (cnt == 3) {
                cntt++;
            } else if (cnt > 0) {
                cntz++;
            }
        }
    }

    printf("TRASEU VALID\n%d\n", cntz * cz + cntt * ct + cntp * cp);
```

```
    return 0;
}
```

C.1.B Problema Bug

```
/*
Problema: bug
Autor: Cerchez Emanuela
Complexitate: O(len(N))
*/
#include <cassert>
#include <fstream>
#define LGMAX 100004
using namespace std;
ifstream fin("bug.in");
ofstream fout("bug.out");

int lg, cerinta, t;
int a[LGMAX];
int uz[12];
int inc[LGMAX];
int rez[LGMAX];
int lgrez;

void citire();
void afisare();
void rezolvare();

int main() {
    citire();
    rezolvare();
    afisare();
    return 0;
}

void citire() {
    char c;
    fin >> cerinta;
    while (fin >> c) a[++lg] = c - '0';
}

void rezolvare() {
    int k, i, c, nr, j;
    int nrbloc = 0;
    inc[0] = lg + 1;  /// inceputul ultimului bloc gasit
    nr = 0;
    for (k = lg; k > 0; k--) {
        if (uz[a[k]] == 0) {
            uz[a[k]]++;
            nr++;
        }
```

```
        if (nr == 10) {
            nrbloc++;
            for (i = 0; i < 10; i++) uz[i] = 0;
            inc[nrbloc] = k;
            nr = 0;
        }
    }
    /// toate numerele de nrbloc cifre se pot obtine
    /// determin cel mai mic numar de nrbloc+1 cifre care nu se poate
    /// obtine. caut cea mai mica cifra care nu apare in ultima secventa
    /// de cifre
    c = 1;
    while (uz[c]) c++;
    /// numerele <=(c-1)9999...9 pot fi obtinute
    rez[1] = c;
    lgrez = 1;
    t = 0;
    if (c == 10) {
        c = 0;
        t = 1;
    }
    for (i = nrbloc; i > 0; i--) {
        for (j = 0; j < 10; j++) uz[j] = 0;
        /// determin pozitia ultimei cifre din rez in blocul curent -
        /// prima aparitie
        for (k = inc[i]; a[k] != rez[lgrez]; k++)
            ;
        /// construiesc uz pentru sufixul blocului curent
        for (k++; k < inc[i - 1]; k++) uz[a[k]] = 1;
        for (c = 0; uz[c]; c++)
            ;
        rez[++lgrez] = c;
    }
}

void afisare() {
    int i;
    if (cerinta == 1)
        fout << t + lgrez << '\n';
    else {
        for (i = 1; i <= lgrez; i++) fout << rez[i];
        fout << '\n';
    }
}
```

C.1.C Problema Triprime

```
/* Autor: Cristian Francu
 * Timp: O(N log log N)
 * Memorie: O(N) - approx 20MB
 *
 * Idee: cel mai mare numar prim din componenta unui numar triprim
```

```
 * este maxim N/6. De aceea calculam ciurul lui Eratostene pana la 65
 * milioane. Pentru optimizare vom folosi un bit per element si nu vom
 * stoca decat numerele impare. Memoria necesara pentru ciur: 4MB. Vom
 * colecta numerele prime in alt vector si apoi vom numara numerele
 * triprime in O(N) folosind doi indici.
 *
 * Nota importanta: calculul ciurului si colectarea numerelor intr-un
 * vector constituie 95% din timpul de executare. De aceea trebuie sa
 * ne concentram pe acesti algoritmi, nu pe numararea numerelor
 * triprime.
 */
#include <stdio.h>
#include <sys/time.h>
#include <time.h>

#define DEBUG 0

#define MAXN 390000000
// ciur pana la MAXN/6, doar cu numere impare, pe biti, patru octeti pe
// int rezulta MAXN/384 intregi adica 1015626
#define MAXNPER384 1015626
#define NPRIME 4000000  // numarul de numere prime pana la 65 000 000

unsigned ciur[MAXNPER384];  // ciur pe biti circa 4MB
int n, prime[NPRIME];       // aici memoram numerele prime circa 16MB

static inline int getBit(int x) {
    int i = x >> 6;         // 32 bits per intreg dar numai impare
    int o = (x >> 1) & 31;  // offset in cadrul intregului

    return (ciur[i] >> o) & 1;
}

static inline void setBit(int x) {
    int i = x >> 6;         // 32 bits per intreg
    int o = (x >> 1) & 31;  // offset in cadrul intregului

    ciur[i] |= (((unsigned)1) << o);
}

int triprime(int x) {
    int ntriprime, i, j, k;

    ntriprime = 0;
    i = 0;
    while (i + 2 < n &&
           prime[i] * prime[j = i + 1] * prime[k = i + 2] <= x) {
        while (prime[i] * prime[j] * prime[k] <= x) k++;

        k--;  // ajustam k
        // pornim cautarea cu doi indici, j creste si k scade
        while (k > j) {          // cata vreme avem un k
            ntriprime += k - j;  // adunam numarul de perechi
            j++;                 // avansam j
```

```
            // scadem k pentru ca produsul sa redevina <= x
            while (k > j && prime[i] * prime[j] * prime[k] > x) k--;
        }
        i++;
    }
    return ntriprime;
}

int main() {
    FILE *fin, *fout;
    int a, b, c, i, d, tria, trib;

    fin = fopen("triprime.in", "r");
    fscanf(fin, "%d%d", &a, &b);
    fclose(fin);

    // ciurul lui Eratostene pana la b / 6
    c = (b / 6) | 63;  // vrem sa fie divizibil cu 64
    // ciurul lui Eratostene doar pentru numere impare
    for (i = 3; i * i <= c; i += 2)
        if (getBit(i) == 0)
            for (d = i * i; d <= c; d += 2 * i) setBit(d);

    // colectam numerele prime
    prime[n++] = 2;  // stocam 2, este singurul numar prim par
    for (i = 3; i <= c; i += 2)
        if (getBit(i) == 0) prime[n++] = i;
    prime[n] = c + 1;  // santinela

    trib = triprime(b);
    tria = triprime(a - 1);

    fout = fopen("triprime.out", "w");
    fprintf(fout, "%d\n", trib - tria);
    fclose(fout);

    return 0;
}
```

C.2 Seniori

C.2.A Problema 3dist

```
#include <bits/stdc++.h>

#define N_MAX 250000
#define INF (INT_MAX - 2)

using namespace std;
```

```
ifstream fin("3dist.in");
ofstream fout("3dist.out");

struct Point {
    int x, y, yn;
    int min_dist;
    int pos;
    bool operator<(Point &oth) const {
        return this->x < oth.x || (this->x == oth.x && this->y < oth.y);
    }
};

int N;
int tree_low[4 * N_MAX + 5], tree_high[4 * N_MAX + 5];
Point P[N_MAX + 5];
vector<int> ind[N_MAX + 5];
int L;
int srt[2 * N_MAX + 5];

void reset_trees() {
    for (int i = 1; i <= 4 * L; i++) {
        tree_low[i] = tree_high[i] = INF;
    }
}

void update(int tree[], int node, int le, int ri, int pos, int val) {
    if (le == ri) {
        tree[node] = min(tree[node], val);
        return;
    }
    int mid = (le + ri) / 2;
    if (pos <= mid) {
        update(tree, 2 * node, le, mid, pos, val);
    } else {
        update(tree, 2 * node + 1, mid + 1, ri, pos, val);
    }
    tree[node] = min(tree[2 * node], tree[2 * node + 1]);
}

int mi;
void query(int tree[], int node, int le, int ri, int a, int b) {
    if (a <= le && ri <= b) {
        mi = min(mi, tree[node]);
        return;
    }
    int mid = (le + ri) / 2;
    if (a <= mid) {
        query(tree, 2 * node, le, mid, a, b);
    }
    if (b > mid) {
        query(tree, 2 * node + 1, mid + 1, ri, a, b);
    }
}
```

```
map<int, vector<Point> > d_min;
map<int, vector<Point> > diag;
void process(map<int, vector<Point> > &diag, int d) {
    for (auto diagonal : diag) {
        int dif = diagonal.first;
        vector<Point> A = diagonal.second;
        vector<Point> B, C;
        if (diag.find(dif - d) != diag.end()) {
            B = diag[dif - d];
        }
        if (diag.find(dif + d) != diag.end()) {
            C = diag[dif + d];
        }
        int le_B = 0, ri_B = 0, le_C = 0, ri_C = 0;
        for (int i = 0; i < (int)A.size(); i++) {
            while (ri_B < (int)B.size() && B[ri_B].x <= A[i].x) {
                ri_B++;
            }
            while (le_B < ri_B && B[le_B].x < A[i].x - d) {
                le_B++;
            }

            while (ri_C < (int)C.size() && C[ri_C].x <= A[i].x + d) {
                ri_C++;
            }
            while (le_C < ri_C && C[le_C].x < A[i].x) {
                le_C++;
            }

            for (int j = le_B; j < ri_B; j++) {
                ind[A[i].pos].push_back(B[j].pos);
            }
            for (int j = le_C; j < ri_C; j++) {
                ind[A[i].pos].push_back(C[j].pos);
            }
        }
    }
}

set<pair<int, int> > check;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> N;
    assert(1 <= N && N <= 250000);
    for (int i = 1; i <= N; i++) {
        cin >> P[i].x >> P[i].y;
        assert(P[i].x >= 0 && P[i].y >= 0 && P[i].x <= 1000000000 &&
               P[i].y <= 1000000000);
        srt[++L] = P[i].y;
        check.insert(make_pair(P[i].x, P[i].y));
    }
```

```
    assert((int)check.size() == N);

    sort(srt + 1, srt + L + 1);
    L = unique(srt + 1, srt + L + 1) - srt - 1;
    for (int i = 1; i <= N; i++) {
        P[i].yn = lower_bound(srt + 1, srt + L + 1, P[i].y) - srt;
        P[i].min_dist = INF;
    }

    sort(P + 1, P + N + 1);

    reset_trees();
    for (int i = 1; i <= N; i++) {
        mi = INF;
        query(tree_low, 1, 1, L, 1, P[i].yn);
        if (mi != INF) {
            P[i].min_dist = min(P[i].min_dist, P[i].x + P[i].y + mi);
        }
        mi = INF;
        query(tree_high, 1, 1, L, P[i].yn, L);
        if (mi != INF) {
            P[i].min_dist = min(P[i].min_dist, P[i].x - P[i].y + mi);
        }

        update(tree_low, 1, 1, L, P[i].yn, -P[i].x - P[i].y);
        update(tree_high, 1, 1, L, P[i].yn, -P[i].x + P[i].y);
    }

    reset_trees();
    for (int i = N; i >= 1; i--) {
        mi = INF;
        query(tree_low, 1, 1, L, 1, P[i].yn);
        if (mi != INF) {
            P[i].min_dist = min(P[i].min_dist, -P[i].x + P[i].y + mi);
        }
        mi = INF;
        query(tree_high, 1, 1, L, P[i].yn, L);
        if (mi != INF) {
            P[i].min_dist = min(P[i].min_dist, -P[i].x - P[i].y + mi);
        }

        update(tree_low, 1, 1, L, P[i].yn, P[i].x - P[i].y);
        update(tree_high, 1, 1, L, P[i].yn, P[i].x + P[i].y);
    }

    for (int i = 1; i <= N; i++) {
        P[i].pos = i;
        d_min[P[i].min_dist].push_back(P[i]);
    }

    for (auto it : d_min) {
        vector<Point> points = it.second;
        diag.clear();
```

```
        int d = it.first;
        for (auto it : points) {
            diag[it.x + it.y].push_back(it);
        }
        process(diag, d);
        // cout << d << "\n";
        diag.clear();
        for (auto it : points) {
            diag[it.x - it.y].push_back(it);
        }
        process(diag, d);
    }

    int ans = 0;
    for (int i = 1; i <= N; i++) {
        sort(ind[i].begin(), ind[i].end());
        ind[i].resize(unique(ind[i].begin(), ind[i].end()) -
                      ind[i].begin());
        for (int j = 0; j < (int)ind[i].size(); j++) {
            for (int k = j + 1; k < (int)ind[i].size(); k++) {
                if (abs(P[ind[i][j]].x - P[ind[i][k]].x) +
                        abs(P[ind[i][j]].y - P[ind[i][k]].y) ==
                    P[i].min_dist) {
                    ans++;
                }
            }
        }
    }
    cout << ans / 3 << "\n";
    return 0;
}
```

C.2.B Problema Piezișă

```
// rapeanu_assert5.cpp
#include <bits/stdc++.h>

#include <cassert>

using namespace std;

const int NMAX = 5e5;
const int BUCK = 1000;
const int inf = 1e9;

class BucketDecomposition {
    int n;
    int bucket_size;
    vector<int> values;
    vector<int> bucket_values;
    vector<int> bucket;
```

```
    public:
     /// 0 indexed
     BucketDecomposition(int n, int bucket_size) {
         this->n = n;
         this->bucket_size = bucket_size;
         values = vector<int>(n + 1, inf);
         bucket = vector<int>(n + 1, 0);

         int current_bucket = 0;
         for (int i = 0; i < n; i += bucket_size) {
             for (int j = i; j < n && j < i + bucket_size; j++) {
                 bucket[j] = current_bucket;
             }
             current_bucket++;
         }
         bucket_values = vector<int>(current_bucket, inf);
     }

     void update(int pos, int value) {
         values[pos] = value;
         bucket_values[bucket[pos]] =
             min(bucket_values[bucket[pos]], value);
     }

     int query(int r) {
         r--;
         int ans = inf;
         for (int i = bucket[r] - 1; i >= 0; i--) {
             ans = min(ans, bucket_values[i]);
         }
         for (int i = r; i >= 0 && bucket[i] == bucket[r]; i--) {
             ans = min(ans, values[i]);
         }
         return ans;
     }
};

struct query_t {
     int id;
     int l;
     int r;
     int ans;

     query_t() {
         id = l = r = 0;
         ans = inf;
     }

     bool operator<(const query_t& other) const {
         return this->r > other.r;
     }
};

void solve(int n, int v[], int q, int l[], int r[], int solutions[]) {
```

```
    assert(n <= 500000);
    assert(q <= 500000);

    for (int i = 0; i < n; i++) {
        assert(0 <= v[i] && v[i] < (1 << 30));
    }

    for (int i = 0; i < q; i++) {
        assert(0 <= l[i] && l[i] <= r[i] && r[i] < n);
    }

    vector<int> prefix_sums(n + 1, 0);
    vector<pair<int, int>> normalizare;

    normalizare.push_back({prefix_sums[0], 0});
    for (int i = 0; i < n; i++) {
        prefix_sums[i + 1] = prefix_sums[i] ^ v[i];
        normalizare.push_back({prefix_sums[i + 1], i + 1});
    }

    sort(normalizare.begin(), normalizare.end());

    int last = 0;
    for (int i = 0; i < (int)normalizare.size(); i++) {
        if (i > 0 && normalizare[i].first != normalizare[i - 1].first) {
            last++;
        }
        prefix_sums[normalizare[i].second] = last;
    }

    normalizare.clear();

    vector<int> fr(last + 1, 0);

    for (int i = 0; i < (int)prefix_sums.size(); i++) {
        fr[prefix_sums[i]]++;
    }

    vector<query_t> queries(q);

    for (int i = 0; i < q; i++) {
        queries[i].l = l[i] + 1;
        queries[i].r = r[i] + 1;
        queries[i].ans = inf;
        queries[i].id = i;
    }

    vector<int> nxt(n + 1, -1);
    vector<int> ant(n + 1, -1);

    for (int value = 0; value <= last; value++) {
        if (fr[value] > BUCK) {
            for (int i = 0; i <= n; i++) {
                ant[i] = (prefix_sums[i] == value
```

```
                                ? i
                                : (i > 0 ? ant[i - 1] : -1));
            }
            for (int i = n; i >= 0; i--) {
                nxt[i] = (prefix_sums[i] == value
                                ? i
                                : (i < n ? nxt[i + 1] : -1));
            }

            for (auto& it : queries) {
                if (ant[it.l - 1] != -1 && nxt[it.r] != -1) {
                    it.ans = min(it.ans, nxt[it.r] - ant[it.l - 1]);
                }
            }
        }
    }

    vector<vector<int>> positions(last + 1, vector<int>());
    for (int i = 0; i <= n; i++) {
        positions[prefix_sums[i]].push_back(i);
    }

    sort(queries.begin(), queries.end());

    BucketDecomposition batog(n + 1, BUCK);
    int idx = n;

    for (auto& query : queries) {
        while (idx >= query.r) {
            if (fr[prefix_sums[idx]] <= BUCK) {
                for (auto it : positions[prefix_sums[idx]]) {
                    batog.update(it, idx - it);
                    if (it >= idx) {
                        break;
                    }
                }
            }
            idx--;
        }
        query.ans = min(query.ans, batog.query(query.l));
    }

    for (auto& query : queries) {
        if (query.ans == inf) {
            query.ans = -1;
        }
        solutions[query.id] = query.ans;
    }
}

#include <stdio.h>
#include <stdlib.h>

#define LEN (1 << 12)
```

```
static char buff[LEN];
static int ind = LEN - 1;

static int int32() {
    int ans = 0;

    while (buff[ind] < '0' || buff[ind] > '9') {
        if (++ind >= LEN) {
            fread(buff, 1, LEN, stdin);
            ind = 0;
        }
    }

    while (!(buff[ind] < '0' || buff[ind] > '9')) {
        ans = ans * 10 + buff[ind] - '0';
        if (++ind >= LEN) {
            fread(buff, 1, LEN, stdin);
            ind = 0;
        }
    }
    return ans;
}

int main() {
    int n, q;
    n = int32();
    int* v = (int*)malloc(sizeof(int) * n);

    for (int i = 0; i < n; i++) {
        v[i] = int32();
    }

    q = int32();

    int* l = (int*)malloc(sizeof(int) * q);
    int* r = (int*)malloc(sizeof(int) * q);
    for (int i = 0; i < q; i++) {
        l[i] = int32();
        r[i] = int32();
    }

    int* answers = (int*)malloc(sizeof(int) * q);

    solve(n, v, q, l, r, answers);

    for (int i = 0; i < q; i++) {
        printf("%d\n", answers[i]);
    }

    free(answers);
    free(l);
    free(r);
    free(v);
```

```
    return 0;
}
```

C.2.C Problema Portocal

```
#include <algorithm>
#include <cassert>
#include <cstdio>
#include <fstream>
#include <iostream>
#include <vector>
#define DIM 500005
#define mod 1000000009
using namespace std;
int n, k, m, i, j, x, y, minim, nr, cerinta;
vector<int> v[DIM];
int dp[DIM][2], num[DIM][2], t[DIM], s[DIM], val[DIM], len[DIM],
    nrp[DIM], total[DIM], nums[DIM], dps[DIM];
int vizitat[DIM];
void mai_mare(int nod, int nr) {
    dp[nod][0] = 0;
    num[nod][0] = nr * 1LL * total[nod] % mod;
}
void mai_mic(int nod, int nr) {
    dp[nod][0] = nrp[nod];
    num[nod][0] = total[nod] * 1LL * nr % mod;
}
void egal_k(int nod) {
    dp[nod][1] = 0;
    num[nod][1] = total[nod];
}
void egal(int nod) {
    int d, nr;
    d = 1;
    nr = 1;
    for (int i = 0; i < v[nod].size(); i++) {
        int vecin = v[nod][i];
        if (vecin != t[nod]) {
            d += dp[vecin][0];
            nr = nr * 1LL * num[vecin][0] % mod;
        }
    }
    if (dp[nod][0] > d) {
        dp[nod][0] = d;
        num[nod][0] = nr;
    } else if (dp[nod][0] == d) {
        num[nod][0] = (nr + num[nod][0]) % mod;
    }
    d = dps[v[nod].size()] = 0;
    nr = nums[v[nod].size()] = 1;
    for (int i = v[nod].size() - 1; i >= 0; i--) {
        int vecin = v[nod][i];
```

```
        dps[i] = dps[i + 1];
        nums[i] = nums[i + 1];
        if (vecin != t[nod]) {
            dps[i] += min(dp[vecin][0], dp[vecin][1]);
            if (dp[vecin][0] == dp[vecin][1]) {
                nums[i] = nums[i] * 1LL *
                          (num[vecin][0] + num[vecin][1]) % mod;
            } else {
                if (dp[vecin][0] < dp[vecin][1]) {
                    nums[i] = nums[i] * 1LL * num[vecin][0] % mod;
                } else {
                    nums[i] = nums[i] * 1LL * num[vecin][1] % mod;
                }
            }
        }
    }
    for (int i = 0; i < v[nod].size(); i++) {
        int vecin = v[nod][i];
        if (vecin != t[nod]) {
            if (d + dps[i + 1] + dp[vecin][1] < dp[nod][1]) {
                dp[nod][1] = d + dps[i + 1] + dp[vecin][1];
                num[nod][1] =
                    nr * 1LL * nums[i + 1] % mod * num[vecin][1] % mod;
            } else if (d + dps[i + 1] + dp[vecin][1] == dp[nod][1]) {
                num[nod][1] =
                    (nr * 1LL * nums[i + 1] % mod * num[vecin][1] +
                     num[nod][1]) %
                    mod;
            }
            d += dp[vecin][0];
            nr = nr * 1LL * num[vecin][0] % mod;
        }
    }
    dp[nod][1]++;
}
void dfs(int nod, int tata) {
    t[nod] = tata;
    len[nod] = len[tata] + 1;
    nrp[nod] = 1;
    total[nod] = 1;
    vizitat[nod] = 1;
    for (int i = 0; i < v[nod].size(); i++) {
        int vecin = v[nod][i];
        if (vecin != tata) {
            dfs(vecin, nod);
            nrp[nod] += nrp[vecin];
            total[nod] = (total[vecin] * 1LL * total[nod]) % mod;
            if (val[vecin] == -1) {
                total[nod] = total[nod] * 1LL * m % mod;
            }
        }
    }
    if (len[nod] > k) {
        dp[nod][0] = 0;
```

```
        dp[nod][1] = n + 1;
        if (val[nod] != -1) {
            num[nod][0] = 1;
        } else {
            num[nod][0] = m;
        }
        for (i = 0; i < v[nod].size(); i++) {
            int vecin = v[nod][i];
            if (vecin != tata) {
                num[nod][0] = num[nod][0] * 1LL * num[vecin][0] % mod;
            }
        }
        return;
    }
    dp[nod][0] = dp[nod][1] = n + 1;
    if (val[nod] == -1) {
        if (s[len[nod]] != m) {
            mai_mare(nod, m - s[len[nod]]);
        } else {
            mai_mic(nod, s[len[nod]] - 1);
        }
        if (len[nod] == k) {
            egal_k(nod);
        } else {
            egal(nod);
        }
    } else {
        if (val[nod] == s[len[nod]]) {
            if (len[nod] == k) {
                egal_k(nod);
            } else {
                egal(nod);
            }
        } else {
            if (val[nod] > s[len[nod]]) {
                mai_mare(nod, 1);
            } else {
                mai_mic(nod, 1);
            }
        }
    }
}
int main() {
    assert(scanf("%d", &cerinta));
    assert(cerinta == 1 || cerinta == 2);
    assert(scanf("%d%d%d", &n, &m, &k));
    assert(1 <= n && n <= 500000);
    assert(1 <= m && m <= 500000);
    assert(1 <= k && k <= n);
    for (i = 1; i <= n; i++) {
        assert(scanf("%d", &val[i]));
        assert(val[i] == -1 || (val[i] >= 1 && val[i] <= m));
    }
    for (i = 1; i <= k; i++) {
```

```
        assert(scanf("%d", &s[i]));
        assert(s[i] >= 1 && s[i] <= m);
    }
    for (i = 1; i < n; i++) {
        assert(scanf("%d%d", &x, &y));
        assert(1 <= x && x <= n);
        assert(1 <= y && y <= n);
        v[x].push_back(y);
        v[y].push_back(x);
    }
    dfs(1, 0);
    for (i = 1; i <= n; i++) {
        assert(vizitat[i] == 1);
    }
    assert(dp[1][1] < n);
    if (cerinta == 1) {
        cout << dp[1][1] + 1;
    } else {
        cout << num[1][1];
    }
}
```

C.2.D Problema „Miyuki vrea să împăturească arborigami”

```
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using cd = complex<double>;

constexpr ll mod = 1e9 + 7, maxn = 5e5 + 10;

int n, curr_node;
vector<int> vec[maxn] = {};

vector<pair<int, int>> operatii;

int fold(int x, int y) {
    operatii.emplace_back(x, y);
    return curr_node++;
}

// impatura subarborele lui x la o stea
// returneaza (centrul stelei, centru e radacina).
pair<int, bool> dfs(int x) {
    vector<int> centers;
    bool is_center = false;

    for (auto y : vec[x]) {
        vec[y].erase(find(begin(vec[y]), end(vec[y]), x));
        if (vec[y].empty())
            is_center = true;
```

```
        else {
            auto t = dfs(y);
            centers.push_back(t.first);
            if (!t.second) is_center = true;
        }
    }

    return !is_center
               ? make_pair(accumulate(begin(centers) + 1, end(centers),
                                      centers[0], fold),
                           false)
               : make_pair(
                     accumulate(begin(centers), end(centers), x, fold),
                     true);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n;
    if (n < 4) {
        cout << 0 << "\n";
        return 0;
    }

    curr_node = n + 1;

    for (int i = 0, x, y; i < n - 1; ++i) {
        cin >> x >> y;
        vec[x].push_back(y);
        vec[y].push_back(x);
    }

    dfs(1);

    cout << operatii.size() << "\n";
    for (auto x : operatii) cout << x.first << ' ' << x.second << "\n";
}
```

C.2.E Problema Guguștiuc

```
#include <bits/stdc++.h>

const int MAX_N = 500000;
const int MAX_X = 1000000;

const int BUFF = 1 << 12;
int curs = BUFF - 1;
char buffer[BUFF];

char getChr() {
```

```
    ++curs;
    if (curs == BUFF) {
        curs = 0;
        fread(buffer, 1, BUFF, stdin);
    }
    return buffer[curs];
}

int getNr() {
    char chr = getChr();
    while (!isdigit(chr)) chr = getChr();

    int nr = 0;
    while (isdigit(chr)) {
        nr = nr * 10 + chr - '0';
        chr = getChr();
    }
    return nr;
}

int aint[1 + 4 * MAX_X];
bool lazyset0[1 + 4 * MAX_X], lazyadd[1 + 4 * MAX_X];
int lazy[1 + 4 * MAX_X];

/********************* SEGMENT TREE *************************/
void pushLazy(int nod, int l, int r) {
    if (l < r) {
        if (lazyset0[nod]) {
            lazyset0[2 * nod] = lazyset0[2 * nod + 1] = true;
            lazy[2 * nod] = lazy[2 * nod + 1] = 0;
            lazyadd[2 * nod] = lazyadd[2 * nod + 1] = false;
        }

        if (lazyadd[nod]) {
            lazyadd[2 * nod] = lazyadd[2 * nod + 1] = true;
            lazy[2 * nod] += lazy[nod];
            lazy[2 * nod + 1] += lazy[nod];
        }
    } else {
        if (lazyset0[nod]) aint[nod] = 0;
        if (lazyadd[nod]) aint[nod] += lazy[nod];
    }
    lazyadd[nod] = false;
    lazy[nod] = 0;
    lazyset0[nod] = false;
}

void set0(int i, int j, int l = 1, int r = MAX_X, int nod = 1) {
    pushLazy(nod, l, r);
    if (j < l || r < i || j < i) return;

    if (i <= l && r <= j) {
        lazyset0[nod] = true;
        lazyadd[nod] = false;
```

```
        lazy[nod] = 0;
        pushLazy(nod, l, r);
    } else {
        int mid = (l + r) / 2;
        set0(i, j, l, mid, 2 * nod);
        set0(i, j, mid + 1, r, 2 * nod + 1);
    }
}

void addRange(int i, int j, int val, int l = 1, int r = MAX_X, int nod = 1) {
    pushLazy(nod, l, r);
    if (j < l || r < i || j < i) return;

    if (i <= l && r <= j) {
        lazy[nod] += val;
        lazyadd[nod] = true;
        pushLazy(nod, l, r);
    } else {
        int mid = (l + r) / 2;
        addRange(i, j, val, l, mid, 2 * nod);
        addRange(i, j, val, mid + 1, r, 2 * nod + 1);
    }
}

int queryAint(int pos, int l = 1, int r = MAX_X, int nod = 1) {
    pushLazy(nod, l, r);

    if (l == r)
        return aint[nod];
    else {
        int mid = (l + r) / 2;
        if (pos <= mid)
            return queryAint(pos, l, mid, 2 * nod);
        else
            return queryAint(pos, mid + 1, r, 2 * nod + 1);
    }
}
/******************** /SEGMENT TREE ********************/

std::set<std::pair<int, int>> axis;

struct RangeCut {
    int l, r;
    int x;
};

std::vector<RangeCut> cuts;
std::vector<int> cutPos[1 + MAX_X], rightCut[1 + MAX_X];
std::vector<int> query[1 + MAX_X];

// true daca x il include pe y <=> y este inclus in x
bool includes(std::pair<int, int> x, std::pair<int, int> y) { return x.first <= y.
    ↪ first && y.second <= x.second; }
```

```
int checkCoverage(std::vector<std::pair<int, int>> &x, int i) {
    int res = 0;
    std::pair<int, int> last_range = {-1, -1};
    std::reverse(x.begin(), x.end());

    for (auto it : x)
        if (!includes(last_range, it)) {
            last_range = it;
            res += std::min(i, it.second) - it.first;
        }

    return res;
}

std::vector<RangeCut> rangeStack;
std::priority_queue<std::pair<int, int>> activeRanges;
std::vector<bool> isActive;

int getLastCut() {
    while (activeRanges.size() > 0 && !isActive[activeRanges.top().second])
    ↪ activeRanges.pop();
    if (activeRanges.size() > 0) return activeRanges.top().first;
    return 1;
}

int main() {
    int N, Q;

    N = getNr();
    Q = getNr();

    for (int i = 0; i < N; ++i) {
        int x, y;
        x = getNr();
        y = getNr();
        query[y].push_back(x);
    }

    axis.insert({1, MAX_X});

    for (int i = 0; i < Q; ++i) {
        int t, x;
        t = getNr();
        x = getNr();
        std::set<std::pair<int, int>>::iterator it = axis.lower_bound({x + 1, -1});
        if (it != axis.begin()) {
            it--;
            if (it->first < x && x < it->second) {
                int l = it->first;
                int r = it->second;

                if (t == 1) {
                    axis.erase({l, r});
                    axis.insert({l, x});
```

```
                axis.insert({x, r});
            } else {
                rightCut[r].push_back(cuts.size());
                cutPos[x].push_back(cuts.size());
                cuts.push_back({l, r, x});
                isActive.push_back(false);
            }
        }
    }
}

long long res = 0LL;

for (int i = 1; i <= MAX_X; ++i) {
    // Prima taietura care contine intervalul (i - 1, i)
    int lastCut = getLastCut();
    addRange(lastCut, i - 1, 1);

    // Avem taietura care influenteaza raspunsurile
    // Prima din vector e singura relevanta
    if (cutPos[i - 1].size() > 0) {
        int c = cutPos[i - 1][0];

        // Scot intervalele care devin irelevante din stiva de
        // intervale
        std::vector<std::pair<int, int>> aux;
        while (rangeStack.size() > 0 &&
               includes({cuts[c].l, cuts[c].r}, {rangeStack.back().l,
↪ rangeStack.back().r})) {
            aux.push_back({rangeStack.back().l, rangeStack.back().r});
            isActive[rangeStack.back().x] = false;
            rangeStack.pop_back();
        }

        int sub = checkCoverage(aux, i);

        //  Taietura intervalului care il domina pe c
        lastCut = getLastCut();
        // Toate taieturile care il contin pe c pana la capatul
        // stanga devin 0
        set0(cuts[c].l, cuts[c].x - 1);

        // Intervalele care initial nu erau acoperite de c trebuie
        // scazute din restul
        addRange(lastCut, cuts[c].l - 1, -(std::min(i, cuts[c].r) - cuts[c].l -
↪  sub));

        activeRanges.push({i - 1, c});
        rangeStack.push_back({cuts[c].l, cuts[c].r, c});
        isActive[c] = true;
    }

    for (auto it : rightCut[i]) isActive[it] = false;
```

```
        for (auto it : query[i]) res += queryAint(it);
    }

    printf("%lld", res);

    return 0;
}
```

C.2.F Problema Hoață

```
// Autor: Ioan-Bogdan Iordache, Universitatea din Bucuresti
// Scor: 100
#include <bits/stdc++.h>
using namespace std;

const int INF = 0x3f3f3f3f;

struct Edge {
    int from;
    int to;
    int cost;
    int capacity;
    int flow;

    Edge() {}
    Edge(int from, int to, int cost, int capacity, int flow)
        : from(from),
          to(to),
          cost(cost),
          capacity(capacity),
          flow(flow) {}
};

int encode_node(int room, int weight, int G) {
    return room * (G + 1) + weight + 2;
}

vector<int> compute_potential(const vector<int> &value,
                              const vector<int> &weight, int N, int G,
                              int source = 0, int destination = 1) {
    vector<int> potential(2 + N * (G + 1), INF);

    potential[source] = 0;
    potential[encode_node(0, 0, G)] = 0;
    for (int room = 0; room < N; ++room) {
        for (int w = 0; w <= G; ++w) {
            int curr = encode_node(room, w, G);
            if (w >= weight[room]) {
                int prev = encode_node(room, w - weight[room], G);
                potential[curr] =
                    min(potential[curr], potential[prev] - value[room]);
            }
```

```
            if (room > 0) {
                int prev = encode_node(room - 1, w, G);
                potential[curr] = min(potential[curr], potential[prev]);
            }
        }
    }
    for (int w = 0; w <= G; ++w) {
        potential[destination] = min(
            potential[destination], potential[encode_node(N - 1, w, G)]);
    }

    return potential;
}

pair<int, int> maxflow_mincost(vector<Edge> &edges,
                               const vector<vector<int>> &graph,
                               vector<int> &potential, int source = 0,
                               int destination = 1) {
    int N = (int)graph.size();
    vector<int> d(N, INF), d_real(N, INF), prev(N, -1);
    d[source] = d_real[source] = 0;

    priority_queue<pair<int, int>, vector<pair<int, int>>,
                   greater<pair<int, int>>>
        heap;
    heap.emplace(0, source);

    while (!heap.empty()) {
        int cst = heap.top().first, node = heap.top().second;
        heap.pop();
        if (d[node] != cst) continue;

        for (int edge_idx : graph[node]) {
            auto &edge = edges[edge_idx];
            if (edge.flow == edge.capacity) continue;
            int new_d = d[node] + edge.cost + potential[node] -
                        potential[edge.to];
            if (new_d < d[edge.to]) {
                d[edge.to] = new_d;
                d_real[edge.to] = d_real[node] + edge.cost;
                prev[edge.to] = edge_idx;
                heap.emplace(d[edge.to], edge.to);
            }
        }
    }

    potential = d_real;
    if (prev[destination] == -1) return {0, 0};

    int minimum = INF, cst = 0;
    for (int node = destination; node != source;
         node = edges[prev[node]].from) {
        auto &edge = edges[prev[node]];
        minimum = min(minimum, edge.capacity - edge.flow);
```

```
        cst += edge.cost;
    }

    for (int node = destination; node != source;
         node = edges[prev[node]].from) {
        auto &edge = edges[prev[node]];
        auto &rev_edge = edges[prev[node] ^ 1];
        edge.flow += minimum;
        rev_edge.flow -= minimum;
    }

    return {minimum, cst};
}

int main() {
    int T;
    assert(cin >> T);
    assert(1 <= T && T <= 900);

    int S_N = 0;
    while (T--) {
        int N, K, G;
        assert(cin >> N >> K >> G);
        assert(1 <= N && N <= 300);
        assert(1 <= K && K <= 50);
        assert(1 <= G && G <= 300);
        S_N += N;

        vector<int> value(N), weight(N), alarm(N);
        for (int i = 0; i < N; ++i) {
            assert(cin >> value[i] >> weight[i] >> alarm[i]);
            assert(1 <= value[i] && value[i] <= 300);
            assert(1 <= weight[i] && weight[i] <= 300);
            assert(1 <= alarm[i] && alarm[i] <= 50);
        }

        int source = 0, destination = 1;
        vector<Edge> edges;
        vector<vector<int>> graph(2 + N * (G + 1));

        auto add_edge = [&](int from, int to, int cost, int capacity) {
            edges.emplace_back(from, to, cost, capacity, 0);
            graph[from].push_back((int)edges.size() - 1);
            edges.emplace_back(to, from, -cost, 0, 0);
            graph[to].push_back((int)edges.size() - 1);
        };

        // from source
        add_edge(source, encode_node(0, 0, G), 0, K);

        // same room
        for (int room = 0; room < N; ++room) {
            for (int old_w = 0, new_w = weight[room]; new_w <= G;
                 ++old_w, ++new_w)
```

```
                add_edge(encode_node(room, old_w, G),
                         encode_node(room, new_w, G), -value[room], K);
        }

        // from room to next room
        for (int room = 0; room + 1 < N; ++room) {
            for (int w = 0; w <= G; ++w)
                add_edge(encode_node(room, w, G),
                         encode_node(room + 1, w, G), 0, alarm[room]);
        }

        // last room to destination
        int room = N - 1;
        for (int w = 0; w <= G; ++w)
            add_edge(encode_node(room, w, G), destination, 0,
                     alarm[room]);

        vector<int> potential = compute_potential(value, weight, N, G);

        int sol = 0, flow = 0;
        while (true) {
            auto ret = maxflow_mincost(edges, graph, potential);
            if (ret.first == 0) break;
            flow += ret.first;
            sol -= ret.first * ret.second;
        }
        cout << (flow == K ? sol : -1) << "\n";
    }

    assert(1 <= S_N && S_N <= 900);

    string str;
    assert(!(cin >> str));

    return 0;
}
```