Olimpiada Națională de Informatică, Etapa Națională
Clasa a IX-a
Descrierea Soluțiilor

Comisia Științifică

24 aprilie 2024

Problema 1: Apgreid

Propusă de: Stud. Andrei-Robert Ion, TU Delft

Cerința 1. Pentru această cerință este suficient să parcurgem linear cu un `for` șirul de obiecte date, calculând pe parcurs datele cerute.

Cerința 2. Pentru această cerință vom face mai întâi niște observații.

Observație 1. *Pentru $P, Q, x \in \mathbb{N}^+$, dacă $P \leq Q$ atunci $P + x \leq Q + x$, $Px \leq Qx$, $P^x \leq Q^x$.*

Din această observație reiese că fiecare subsecvența a unei ordonări optime de obiecte este și ea optimă pentru acea submulțime de obiecte. De exemplu, dacă ordonăm optim obiectele, și eliminăm ultimul obiect, atunci ordonarea rezultată va fi optimă pentru toate obiectele rămase. Așadar, putem construi ordonarea optimă pas cu pas.

Observație 2. *Pentru $P, x, y \in \mathbb{N}^+$, $(P + x)y \geq Px + y$.*

Demonstrație. $(P + x)y = Px + xy \geq Px + y$. □

Din această observație reiese că este optim să considerăm mai întâi obiecte ce adună cu x, apoi obiecte ce înmulțesc cu x — într-adevăr, dacă o pereche adiacente de obiecte nu sunt în ordinea aceasta, le putem interschimba și vom îmbunătăți ordonarea.

Observație 3. *Pentru $P, x, y \in \mathbb{N}^+$, $(P + x)^y \geq P^y + x$.*

Demonstrație. $(P + x)^y = P^y + x^y + \sum_{i=1}^{y-1} \binom{y}{i} P^i x^{(y-i)} \geq P^y + x^y \geq P^y + x$. □

La fel, vedem că mai întâi vrem să adunăm, apoi vrem să ridicăm la putere.

Observație 4. *Pentru $P, x, y \in \mathbb{N}^+$, $(Px)^y \geq P^y x$.*

Demonstrație. $(Px)^y = P^y x^y \geq P^y x$. □

În ultimul rând, mai întâi vrem să înmulțim, apoi să ridicăm la putere.

Așadar reiese că într-o ordonare optimă, mai întâi se folosesc toate obiectele ce adaugă la P, apoi toate obiectele ce îl înmulțesc pe P, apoi toate obiectele ce îl ridică pe P la o putere dată. Cum $P + x + y = P + y + x$, $Pxy = Pyx$, $(P^x)^y = P^{(xy)} = (P^y)^x$, nu contează cum sunt ordonate obiectele în cadrul acestor 3 grupe.

Cerința 3. Pentru această cerință, este suficient să ordonăm conform cerinței 2, și apoi să efectuăm operațiile. Pentru a efectua operațiile de ridicare la putere, este necesară folosirea ridicării la putere în timp logaritmic.

Există o optimizare posibilă ce câștigă niște puncte suplimentare în cazul în care concurentul nu cunoaște ridicarea la putere în timp logaritmic. Mai exact, pentru oricare număr prim p, $a^x \bmod p = a^{x \bmod (p-1)} \bmod p$. Așadar, putem face doar o singură ridicare la putere — în loc să calculăm $(\ldots(P^{x_1})^{x_2}\ldots)^{x_n}$, putem calcula P^x unde $x = x_1 \cdots x_n \bmod (p-1)$.

Problema 2: Eras

Propusă de: Stud. Andrei Onuț, Yale University

Soluție pentru primul subtask. Utilizând o matrice de mărime $N \times M$, inițial conținând doar elemente egale cu 0, fiecare modificare dintre cele U poate fi efectuată în complexitatea $O(M)$ (pentru tipul (L, a, v)) sau $O(N)$ (pentru tipul (C, a, v)), iterând prin toate celulele ce își schimbă valoarea. Pentru fiecare dintre cele Q întrebări, se poate afla răspunsul în complexitatea $O(N \times M)$, iterând prin toate celulele prezente în dreptunghiul (se cunoaște $K = 1$) din întrebarea curentă. Astfel, complexitatea totală de timp este de $O(U \times max(N, M) + Q \times N \times M)$.

Soluție pentru al doilea subtask. Cele U modificări pot fi reținute într-un vector. Pentru fiecare dintre cele Q întrebări, se cunoaște $K = 1$: fie (x_1, y_1) și (x_2, y_2) cele două colțuri ale submatricei din întrebarea curentă. Mai mult, să definim *înălțimea* $h = (x_2 - x_1 + 1)$ și *lățimea* $w = (y_2 - y_1 + 1)$. Astfel, o modificare de tipul (L, a, v) contribuie cu $+(v \cdot w)$ la suma ce trebuie determinată, dacă $x_1 \le a \le x_2$; similar, o modificare de tipul (C, a, v) contribuie cu $+(v \cdot h)$ la suma ce trebuie determinată, dacă $y_1 \le a \le y_2$. Deci, pentru fiecare întrebare putem itera prin toate cele U modificări efectuate și, dacă este cazul, actualizăm răspunsul, în funcție de tipul modificării. Complexitatea totală este de $O(U + Q \times U) \le O(Q \times U)$.

Soluție pentru al treilea subtask. Pentru fiecare întrebare, știm: $K \le 2$. Astfel, putem folosi soluția de la subtask-ul precedent pentru a calcula suma pentru fiecare dintre cele două submatrice (pentru $K = 2$). Dacă cele două submatrice au cel puțin o celulă în comun (adică, mai specific, se întâmplă simultan ca $max(x_{1,1}, x_{2,1}) \le min(x_{1,2}, x_{2,2})$ și $max(y_{1,1}, y_{2,1}) \le min(y_{1,2}, y_{2,2})$), atunci trebuie să determinăm câte brățări se află în interiorul sau pe conturul acestei submatrice (rezultată din **intersecția** celor două submatrice din întrebare). Astfel, din nou, putem aplica aceeași metodă ca la subtask-ul precedent. Complexitatea totală este de $O(U + Q \times U) \le O(Q \times U)$.

Soluție pentru al cincilea subtask. Într-o manieră similară cu cea de la paragraful precedent, pentru fiecare dintre cele Q întrebări putem aplica *Principiul Includerii și Excluderii*, folosind operații cu numere în baza doi. Trebuie avut grijă ca, în loc să parcurgem toate cele U modificări pentru fiecare dintre cele $O(2^K)$ sume calculate, alegem să sortăm (în funcție de valoarea a) modificările, înainte de citirea întrebărilor, iar apoi, pentru fiecare sumă ce trebuie să o calculăm (în cadrul întrebărilor), folosim căutări binare și sume parțiale (pe prefix sau sufix). Astfel, complexitatea totală este de $O(U \times \log U + Q \times 2^K \times (K + \log U))$.

Soluție pe matrice mică. Mai întâi, să ne imaginăm că matricea (notată) A din input (după efectuarea celor U modificări) este mică, de mărime $N \times M$. Putem, atunci, să calculăm răspunsul la o întrebare cu K dreptunghiuri folosind $O(K + N \times M)$ operații. *Cum implementăm această metodă?* Vom utiliza o matrice auxiliară $t[i][j]$, unde scopul nostru este ca $t[i][j]$ să reprezinte numărul de dreptunghiuri (din întrebarea curentă) în care se află celula (i, j). Pentru a calcula această matrice $t[i][j]$, vom folosi tehnica *2D Difference Array* (cunoscută în România și sub numele de *Șmenul lui Mars 2D*): fiecare dreptunghi din întrebare va fi procesat în timp constant, apoi o parcurgere a matricei ne va da fiecare $t[i][j]$. Având $t[i][j]$ calculat, soluția constă în a parcurge matricea și a calcula suma elementelor $A[i][j]$ pentru toate pozițiile (i, j) unde se întâmplă: $t[i][j] > 0$.

Soluție completă. Ideea pentru soluția de 100 de puncte este să încercăm să reducem problema dată la problema din paragraful anterior. Mai exact, dacă considerăm K matrice, vom reduce problema la o subproblemă pe o matrice de mărime $(2K+1) \times (2K+1)$.

Să presupunem că matricele din întrebare au coordonatele $(x_{i,1}, y_{i,1})$ și $(x_{i,2}, y_{i,2})$ pentru $i = 1, \ldots, K$. Considerăm acum două linii *adiacente* x, x' din matrice. Dacă nu există nicio latură a vreunei matrice din input (din întrebare) care să separe linia x de linia x', atunci soluția pentru întrebare poate fi calculată „unind" linia x cu linia x'. Mai exact, liniile x, x' vor fi înlocuite de o nouă linie, a cărei valori va fi suma valorilor de pe liniile x, x'.

Aplicând această operație de mai multe ori, ajungem la o matrice cu cel mult $(2K+1)$ linii și coloane. Să considerăm o celulă arbitrară din aceasta matrice *comprimată*. Să presupunem că această celulă corespunde cu submatricea între liniile x, x' și coloanele y, y' din matricea inițială. *Care va fi valoarea din celulă?* Observăm că fiecare operație $(\texttt{L}, a, v)$ cu $x \le a \le x'$ incrementează valoarea din celulă cu $v \cdot (y' - y + 1)$; similar, operațiile de forma $(\texttt{C}, a, v)$ cu $y \le a \le y'$ incrementează valoarea din celulă cu $v \cdot (x' - x + 1)$. Așadar, pentru a calcula valoarea din matrice, trebuie să aflăm suma valorilor tuturor operațiilor ce *afectează* liniile $x, \ldots, x'$, respectiv coloanele $y, \ldots, y'$.

Cum implementăm această metodă? Vom descrie acest lucru pentru linii; coloanele se tratează identic. Vom sorta toate operațiile în funcție de linia *afectată*. Vom calcula, pentru fiecare operație, suma valorilor operațiilor de la ea până la sfârșit (sau de la ea până la început); pentru o operație o, fie $s(o)$ acea sumă. Dacă vrem să vedem suma operațiilor ce *afectează* liniile $x, \ldots, x'$, găsim prima operație ce *afectează* o linie mai mare sau egală decât x, respectiv decât $(x'+1)$. Fie o, respectiv o' aceste operații. Atunci suma cerută este egală cu $s(o) - s(o')$. Operațiile o, o' pot fi găsite cu căutări binare, în timp logaritmic.

Așadar, pentru fiecare interogare (dintre cele Q) vom face $O(K)$ căutări binare pe șiruri de lungime U și o parcurgere a unei matrice de mărime $(2K+1) \times (2K+1)$. Precalcularea constă într-o sortare și calcul de sume parțiale, ce va lua timp $O(U \log U)$. Așadar complexitatea finală este de $O((U + Q \times K) \log U + Q \times K^2)$.

Problema 3: Spirala

Propusă de: Stud. Alexandra Udrişţoiu, Facultatea de Matematică-Informatică, Univ. Bucureşti

Subtask 1:

Pentru acest subtask se poate parcurge matricea în spirală pornind din centru și să verificăm, pentru fiecare element parcurs, dacă valoarea sa respectă sau nu proprietățile date. Dacă nu găsim niciun element din matrice care încalcă proprietățile, răspunsul va fi 0. În caz contrar, înseamnă că răspunsul trebuie să fie 1.

Subtask 2:

O posibilă abordare pentru acest subtask este să fixăm un element $A_{i,j}$ care își va păstra valoarea la final. Apoi, vom proceda asemănător cu subtaskul anterior și vom parcurge elementele matricei verificând ce elemente ar trebui înlocuite. De data aceasta însă vom parcurge matricea începând cu $A_{i,j}$, iar pentru a verifica elementele de dinaintea sa, vom parcurge apoi prima parte a spiralei invers de la $A_{i,j}$ până în centrul matricei. Atunci când întâlnim un element care trebuie schimbat, îl înlocuim în matrice cu orice valoare care respectă proprietățile și continuăm verificarea următoarelor elemente.

Complexitatea acestei soluții este $O(N^4)$.

Subtask 3:

Deoarece ordinea cifrelor nu contează, putem să generăm toate submulțimile de cifre nenule și să presupunem că elementul din mijloc conține aceste cifre. Apoi, vom parcurge matricea în spirală și vom verifica ce elemente trebuie înlocuite ca în subtaskul anterior.

Complexitatea acestei soluții este $O(N^2 * 2^C)$, unde C este numărul de cifre pentru care generăm submulțimii, în cazul nostru egal cu 10.

Subtask 4:

Observația necesară pentru soluția de 100 de puncte este că putem rezolva independent pentru fiecare cifră, deoarece aparițiile unei cifre în matrice nu influențează apariția celorlalte.

Astfel, pentru fiecare cifră c de la 1 la 9, vom presupune că elementul din centrul matricei conține c și vom parcurge restul elementelor în spirală pentru a determina care dintre ele trebuie să conțină c. Elementele care trebuie să conțină cifra c în acest caz nu ar putea să o conțină dacă nici elementul din centru nu conține c și viceversa.

Apoi, pentru fiecare element $A_{i,j}$ din matrice, putem determina ce cifre ar trebui să conțină elementul din centru pentru ca $A_{i,j}$ să își păstreze valoarea neschimbată. Vom reprezenta configurația obținută într-o mască pe biți, unde bitul i este 1 dacă elementul din centru conține cifra i și 0 altfel. Astfel, fiecărui $A_{i,j}$ îi va corespunde o configurație unică $X_{i,j}$ a elementului din centru pentru care $A_{i,j}$ poate să rămână neschimbat.

Răspunusul final se va obține scăzând din numărul total de elemente N^2 numărul maxim de apariții al unei configurații în matricea X obținută. Acest număr se poate afla folosind un vector de frecvență. Complexitatea finală a acestei soluții este $O(N^2)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Anton Cristina, Colegiul Național "Gheorghe Munteanu Murgoci", Brăila
- Stud. Ariciu Toma, Universitatea Politehnica București
- Inst. Dăscălescu Ștefan-Cosmin, RoAlgo
- Lect. dr. Diac Paul, Facultatea de Informatică Iași, Universitatea "A.I. Cuza"
- Stud. Ion Andrei-Robert, TU Delft
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova, Ploiești
- Stud. Onuț Andrei, Yale University
- Prof. Pascu Olivia-Cătălina, Colegiul Național "Nichita Stănescu", Ploiești
- Stud. Udrițoiu Alexandra, Facultatea de Matematică-Informatică, Univ. București
- Drd. Nakajima Tamio-Vesa, Department of Computer Science, University of Oxford