Olimpiada Națională de Informatică, Etapa județeană
Clasa a X-a
Descrierea soluțiilor

Comisia științifică

24 aprilie 2024

Problema 1: Dominoes

Propusă de: Stud. Apostol Ilie-Daniel, Universitatea București

Pregatită de: Stud. Apostol Ilie-Daniel, Universitatea București, Stud. Popescu Ștefan-Alexandru, Universitatea București

Vom considera urmatoarele tipuri de coloane (fie $type_i$ tipul celei de a i-a coloana):

- Daca doar prima linie a coloanei este ocupata, aceasta va fi de tipul SUS.
- Daca doar a doua linie este ocupata, aceasta va fi de tipul JOS.
- Daca amandoua liniile sunt ocupate, va fi de tipul $FULL$.
- Daca niciuna dintre linii nu este ocupata, atunci va fi de tipul $EMPTY$.

Solutie 1 (9 puncte) $k, n, q \leq 8$

Pentru acest subtask, orice solutie care genereaza pentru fiecare interogare toate variantele de pavari va lua punctajul maxim.

Solutie 2 (6 puncte) $k = 0$

Daca $y1 = y2$, atunci se va crea o coloana $FULL$. Se poate pava la stanga si la dreapta cu domino-uri de 2×1.

Altfel, stim ca vom putea pava coloanele y ¡ y1 si y ¿ y2 cu domino-uri de 2×1. Vrem sa vedem daca putem pava zona dintre cele 2 spatii noi ocupate. Stim ca tipurile celor doua coloane pot fi ori SUS, ori JOS. Cazurile sunt urmatoarele:

Daca $type_{y1} \neq typey2$, atunci se poate pava doar daca y1 si y2 au aceeasi paritate.

In cazul in care $type_{y1} = type_{y2}$, atunci se poate pava doar daca y1 si y2 au parități diferite.

Soluția 3 (20 puncte) $n, k, q \leq 5000$

Se observă faptul că raționamentul de la ”Soluția 2” poate fi aplicat cuplând coloanele consecutive de tip SUS și JOS (în sensul în care ignorăm coloanele de tip $EMPTY$ care s-ar putea găsi între ele). Între coloanele unei perechi cuplate aplicăm logica de la ”Soluția 2”, iar între 2 perechi consecutive umplem cu domino-uri de 2×1.

Astfel, pentru fiecare interogare, problema se reduce la a verifica dacă există un astfel de cuplaj valid (neavând coloane de tip $FULL$ între coloanele unei perechi cuplate).

Soluția 4 (32 puncte) $k, q \leq 5000$

Asemănător cu "Soluția 3", doar că se aplică o normalizare a coloanelor din matricea inițială și din interogări.

Soluția 5 (6 puncte) $y1 = y2$

Pentru acest subtask, trebuie să verificăm între care coloane consecutive de tip FULL se vor insera cele două poziții noi. Acest lucru îl putem afla cu o căutare binară.

După inserarea celor două poziții, se formează două noi subsegmente delimitate de coloane de tip FULL. Putem verifica dacă cele două subsegmente se pot pava folosind o precalculare de sume parțiale pe prefix respectiv sufix, care ne spune câte perechi de poziții ocupate dintr-un subsegment se cuplează valid.

Soluția 6 (100 puncte)

Solutia pentru 100 de puncte va optimiza "Soluția 4". Pentru acest lucru va trebui sa aflam care sunt noile cuplari, dupa adaugarea celor 2 pozitii ocupate. Este nevoie de atentie la cazuri. Dupa aflarea celor maxim 2 cuplari, intervalul $[1, N]$ se va sparge in maxim 3 intervale (un prefix, un sufix, un interval in mijloc). Este nevoie sa se verifice (in maxim $O(logK)$ daca aceste intervale pot fi pavate.

Putem asigna pentru fiecare poziție ocupată valoarea 1 dacă aceasta este pe prima linie, pe coloană pară sau pe a doua linie, pe coloană impară și valoarea 0 dacă aceasta este pe prima linie, pe coloană impară sau pe a doua linie, pe coloană pară. Pentru ca grid-ul să se poată pava, trebuie ca pozițiile ocupate consecutive să aibă semne diferite (+1 și -1 sau -1 și +1). Putem precalcula pentru gridul inițial, pentru fiecare subsegment delimitat de două coloane FULL, câte perechi valide avem atât pe prefix, cât și pe sufix.

La fiecare interogare, verificăm dacă intervalele rezultate după adăugarea noilor poziții ocupate se pot pava folosind precalcularea de mai sus. De asemenea, verificăm perechile în care intră pozițiile din interogare.

Problema 2: Seqstr

Propusă de: Prof. Piț-Rada Ionel-Vasile, Colegiul Național Traian, Drobeta-Turnu Severin

Cerința 1 - Soluția 1

Complexitate $O(m^2)$

Avem construite $urmA[0][x]$= poziția următorului element din A egal cu 0, după poziția x, respectiv $urmA[1][x]$ = poziția următorului element din A egal cu 1, după poziția x.

La etapa 1 se construiesc două liste ordonate strict crescător cu pozițiile din șirul B, lista B_0 a elementelor cu valoarea 0 și lista B_1 a elementelor cu valoarea 1. Pentru fiecare listă păstrăm ca poziție de continuare prima poziție din A, $p_0 = urmA[0][0] < n+1$ respectiv $p_1 = urmA[1][0] < n+1$.

La etapa 2 parcurgem lista B_0 și pentru fiecare poziție x din B_0, dacă $B[x+1] = 0$, atunci păstrăm poziția $x+1$ în lista $B_{0,0}$, iar dacă $B[x+1] = 1$, atunci păstrăm poziția $x+1$ în lista B_{01}. Analog din B_1 obținem listele $B_{1,0}$ și $B_{1,1}$. Pentru fiecare listă păstrăm ca poziție de continuare prima poziție $p_{00} = urmA[0][p_0]$, $p_{01} = urmA[1][p_0]$, $p_{10} = urmA[0][p_1]$, $p_{11} = urmA[1][p_1]$, dacă sunt cel mult egale cu n.

La etapa 3 continuăm și obținem listele B_{000}, B_{001}, B_{010}, B_{011}, B_{100}, B_{101}, B_{110}, B_{111} Unele liste este posibil să fie vide și vor fi neglijate cele pentru care nu există poziție de continuare în A. Se continuă până la pasul m când se va obține o singură listă, cu un singur element poziția

m. La final numărul pozițiilor păstrate va fi egal cu numărul subsecvențelor distincte. Procesul este asemănător celui de la metoda radixsort.

Cerința 1 - Soluția 2

Complexitate $O(m^3)$

Pentru fiecare secvență $B[p \dots q]$ verificăm dacă este egală cu una din secvențele anterioare $B[i \dots j]$ de aceeași lungime $q - p + 1 = j - i + 1$ și $1 \le i < p$. Marcăm secvențele distincte. În paralel cu procesul de verificare putem verifica și existența subșirului corespunzător din A folosind $urmA[]$.

Cerința 1 - Soluția 3

Aplicăm metoda programării dinamice. Calculăm $lsb_{i,j}$, reprezintând lungimea celui mai lung sufix comun care se termină pe pozițiile i respectiv j în șirul B. Folosindu-ne de aceste valori, putem determina ls_i, lungimea celui mai lung sufix care se termină pe poziția i în șirul B și a fost deja întâlnit la stânga lui. Ambele precalculări se pot realiza în $O(m^2)$, relațiile de recurență fiind ușor de dedus.

Șirul ls face posibilă verificarea în $O(1)$ dacă o anumită subsecvență a mai fost întâlnită sau nu, dacă le parcurgem lexicografic după capătul stâng, pe urmă după cel drept. Mai exact, dacă avem $ls_{dr} < dr - st + 1$, atunci subsecvența $B[st \dots dr]$ este una nouă.

Folosind tabloul urm de la soluția 1 obținem complexitatea $O(m^2)$.

Cerința 2 - Soluția 1

Complexitate $O(m \cdot n)$ Pentru secvența dată se poate parcurge procesul de determinare a celui mai lung subșir comun dintre $B[p \dots q]$ și $A[1 \dots n]$. Numărul subșirurilor comune de lungime maximă este soluția căutată dacă lungimea subșirului comun de lungime maximă este $q - p + 1$.

Cerința 3 - Soluția 1

Se parcurg subsecvențele și se calculează pentru fiecare dintre ele numărul subșirurilor corespunzător egale, complexitate $O(m^2 \cdot n)$. $d[i][j][k]$=numărul subșirurilor din $A[k \dots n]$ egale cu subsecvența $B[i \dots j]$.

Apoi pentru $i = j - 1, \dots, 1$, $d[i][j][k] = d[i+1][j][k+1] + d[i][j][k+1]$, dacă $B[i]$ egal cu $A[k]$ respectiv $d[i][j][k] = d[i][j][k+1]$, dacă $B[i]$ diferit de $A[k]$. Se însumează apoi rezultatele doar pentru subsecvențele distincte, $O(m^2)$.

Problema 3: Teze

Propusă de: Lect. univ. Pătcaș Csaba, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai, Cluj-Napoca

Căutăm o partiționare a lui n în f termeni pozitivi $l_1, \dots, l_f$, în așa fel încât $n = l_1 + \dots + l_f$ și $m \left(\sum_{i=1}^{f} (p + \sum_{j=1}^{l_i} (k + t_j)) \right)$ să fie minim.

Observăm, că este de ajuns să determinăm timpul minim necesar corectării unei singure teze, valoare care va fi înmulțită cu m la final.

Soluția 1 (5 - 15 puncte)

Putem genera recursiv toate partiționările posibile ale lui n. O posibilă optimizare este să generăm doar variantele în care $l_1 \leq l_2 \leq \ldots \leq l_f$. Complexitatea acestei soluții este exponențială.

Soluția 2 (35 puncte)

Aplicăm metoda programării dinamice. Notăm cu dp_i timpul total minim pentru a corecta exact i probleme. Avem $dp_i = \min\limits_{j=1}^{i}(dp_{i-j} + p + j \cdot k + \sum\limits_{w=1}^{j} t_w)$. Precalculând sumele primelor w elemente din șirui t pentru $\forall w \in [1, n]$ obținem complexitatea $O(n^2)$

Soluția 3 (50 puncte)

Observăm, că pentru un f fixat partiționarea optimă nu poate avea $l_i - l_j > 1, \forall i, j \in [1, f]$.

Demonstrație. Să presupunem prin absurd, că pentru un anumit f fixat avem o partiționare optimă $n = l_1 + \ldots + l_f$ având $l_i - l_j > 1$ pentru un anumit i și j. Dacă înlocuim l_i cu $l'_i = l_i - 1$ și l_j cu $l'_j = l_j + 1$ avem o nouă partiție $n = l_1 + \ldots + l'_i + \ldots + l'_j + \ldots + l_f$. Calculând diferența dintre timpurile necesare în cazul celor două variante, avem:

$\sum\limits_{j=1}^{l_i} t_j + \sum\limits_{j=1}^{l_j} t_j - \sum\limits_{j=1}^{l_i-1} t_j - \sum\limits_{j=1}^{l_j+1} t_j = t_{l_i} - t_{l_j}$

Fiindcă potrivit presupunerii inițiale $l_i > l_j$ și din restricțiile problemei $t_1 < \ldots < t_n$, este clar, că diferența $t_{l_i} - t_{l_j}$ este pozitivă, deci prin înlocuirea lui l_i și l_j cu l'_i și l'_j am obținut o partiționare mai bună, ceea ce contrazice presupunerea că partiționarea inițială este optimă. □

Folosind acest rezultat, deducem, că pentru f fixat în partiționarea optimă mereu vom avea $f - n \bmod f$ bucăți de termeni egale cu $[n/f]$ și $n \bmod f$ bucăți de termeni egale cu $[n/f] + 1$.

Fixăm pe rând f la toate valorile posibile de la 1 la n, iar pentru fiecare f iterăm de la 1 la $[n/f]$ pentru a calcula timpurile necesare. La complexitate avem $O(\frac{n}{1} + \frac{n}{2} + \ldots + \frac{n}{n}) = O(n \log n)$.

Soluția 4 (55 - 60 puncte)

Precalculăm sumele parțiale pentru șirul t, astfel scăpăm de al doilea ciclu din soluția precedentă și obținem complexitatea $O(n)$.

Soluția 5 (65 - 80 puncte)

Notăm cu opt_f timpul optim de corectare a celor n probleme în exact f faze. Dacă ne uităm la șirul opt, observăm că acesta este unimodal, adică pe cazul general pe intervalul $f \in [1, n]$ prima dată scade, pe urmă crește (exceptând cazurile, când are minimul în una dintre cele două capete).

Demonstrație. Pentru a simplifica calculele, putem adăuga p la t_1. Trebuie să minimizăm funcția $z(x) = \sum\limits_{i=1}^{x} \sum\limits_{j=1}^{l_i} t_j$ definită pe numerele naturale din intervalul $x \in [1, n]$.

Vom arăta, că dacă $z(x) < z(x-1)$, atunci $z(x-1) < z(x-2)$.

Pentru a trece de la x faze la $x - 1$ faze, putem redistribui toate problemele corectate într-o fază de dimensiune $\lfloor \frac{n}{x} \rfloor$ la celelalte faze.

Notăm cu $h = \lfloor \frac{n}{x} \rfloor$. Redistribuind o fază în care corectăm h probleme, scădem contribuțiile $t_1, \ldots, t_h$ la cost ale acestor probleme, dar adunăm $t'_1, \ldots, t'_h$ contribuțiile corespunzătoare după redistribuire, unde t'_i este termenul corespunzător celei de-a i-a problemă după redistribuire, iar i' este indicele celei de-a i-a problemă după redistribuire.

Relația $z(x) < z(x-1)$ este echivalentă cu $-t_1 - t_2 - \ldots - t_h + t'_1 + t'_2 + \ldots + t'_h > 0$.

Notăm cu $g = \lfloor \frac{n}{x-1} \rfloor$. Vrem să arătăm că $-t_1 - t_2 - \ldots - t_g + t''_1 + t''_2 + \ldots + t''_g > 0$, unde t''_i este termenul corespunzător de cost al celei de-a i-a problemă după redistribuire de la $x-1$ faze la $x-2$ faze.

$-t_1 - t_2 - \ldots - t_g + t''_1 + t''_2 + \ldots + t''_g =$
$-t_1 - t_2 - \ldots - t_h - t_{h+1} - \ldots - t_g + t''_1 + t''_2 + \ldots + t''_h + t''_{h+1} + \ldots + t''_g = (-t_1 - t_2 - \ldots - t_h + t''_1 + t''_2 + \ldots + t''_h) + (-t_{h+1} - \ldots - t_g + t''_{h+1} + \ldots + t''_g) > (-t_1 - t_2 - \ldots - t_h + t'_1 + t'_2 + \ldots + t'_h) + (-t_{h+1} - \ldots - t_g + t''_{h+1} + \ldots + t''_g) > 0 + (-t_{h+1} - \ldots - t_g + t''_{h+1} + \ldots + t''_g) > 0 + 0 = 0$

Explicație: în inegalitatea a patra am folosit faptul, că $t'_i \le t''_i$ pentru orice i, în inegalitatea a cincea am folosit ipoteza de lucru, iar în inegalitatea șasea am folosit faptul, că $t_i \le t''_i$ pentru orice $i > 1$. Inegalitățile $t_i \le t'_i \le t''_i$ pentru orice $i > 1$ provin din modul în care redistribuim problemele. Poziția pe care se află o problemă înainte de redistribuire va fi mai mic decât cea pe care se află după redistribuire, de unde și costul asociat corectării problemei crește după redistribuire. Acest lucru se întâmplă doar pentru $i > 1$ din cauza faptului, că am adunat p la t_1; totuși $i' > 1$, deoarece o problemă nu va fi niciodată redistribuită pe prima poziție a unei faze, iar $h + 1 > 1$.

Am demonstrat că dacă $z(x) < z(x-1)$, atunci $z(x-1) < z(x-2)$. Inductiv $z(x) < z(x-1) < \ldots < z(1)$.

Rezultă, că în afară de cazurile speciale, în care funcția z crește pe tot intervalul $[1, n]$, sau descrește pe tot intervalul $[1, n]$, pe cazul general funcția z descrește strict de la 1 până într-un punct de minim, eventual stă pe un platou, apoi crește până la n. $\square$

Bazându-ne pe acest fapt, putem optimiza soluția precedentă oprind din iterare în momentul în care detectăm o scădere pe urmă o creștere, adică dacă avem $opt_{i-1} > opt_i \le opt_{i+1}$ știm imediat, că i este numărul optim de faze.

Dacă implementăm obținerea sumei parțiale $\sum_{j=1}^{i} t_j$ pentru orice i dat în timp constant folosindu-ne de modul de construire al șirului t, complexitatea soluției rămâne liniară, dar în practică se comportă mai bine ca cel precedent.

Soluția 6 (100 puncte)

Observăm că numărul de valori distincte h din soluția precedentă este de ordinul $O(\sqrt{n})$, deci putem evalua opt doar în valorile f minime corespunzătoare fiecărui h. Pentru a obține aceste perechi (h, f), prima oară iterăm prin valorile h de la 1 la $\lfloor \sqrt{n} \rfloor$, iar f-ul minim corespunzător o să fie egal cu $f = \lfloor \frac{n}{h} \rfloor$, iar apoi iterăm prin valorile f de la 1 la $\lfloor \sqrt{n} \rfloor$, iar h-ul o să fie conform definiției $h = \lfloor \frac{n}{f} \rfloor$.

Complexitatea soluției este $O(\sqrt{n})$.

Soluția 7 (100 puncte)

Am văzut la soluția precedentă că de fapt căutăm minimul unui șir unimodal, ceea ce se poate face în timp logaritmic aplicând căutarea ternară, sau o variantă ușor modificată a căutării binare. Complexitate $O(\log n)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Nodea Gheorghe-Eugen, Centrul Județean de Excelență Gorj
- Lect. univ. Pătcaș Csaba, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai, Cluj-Napoca
- Prof. Piț-Rada Ionel-Vasile, Colegiul Național Traian, Drobeta-Turnu Severin
- Prof. Moț Nistor, Școala ”Dr.Luca”, Brăila
- Stud. Gavrilă-Ionescu Vlad-Alexandru, Zurich, Elveţia
- Stud. Apostol Ilie-Daniel, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Oprea Mihai-Adrian, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Pagu Tudor Ștefan, Delft University of Technology Delft, Olanda
- Stud. Popa Sebastian, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Popescu Ioan, Facultatea de Automatică și Calculatoare, Universitatea Politehnică București
- Stud. Popescu Ștefan-Alexandru, Facultatea de Matematică-Informatică, Universitatea București