Olimpiada Națională de Informatică, Baraj selecție lot juniori Descrierea soluțiilor

Comisia științifică

March 25, 2026

Problema 1. Fibopower

Propusă de: prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj
stud. Andrei Boacă, Universitatea "Alexandru Ioan Cuza", Facultatea de Informatică Iași

Observații și precalculări preliminarii

Observăm că sunt 43 de numere Fibonacci mai mici decât 10^9. Prin urmare, toate numerele fibopower pot fi generate utilizând trei parcurgeri imbricate. Adăugând într-un vector aceste numere, putem determina dacă un număr din șirul A este fibopower, căutând binar în vectorul menționat mai sus. Astfel, transformăm șirul A într-un șir binar, unde $A_i = 1$ dacă numărul A_i din șirul inițial era fibopower, respectiv $A_i = 0$ în caz contrar. Pentru soluțiile prezentate mai jos vom folosi această interpretare a vectorului A, fiind interesați de secvențele cu suma K.

Soluție O($N^2 \times Q$)

Construim un vector de sume parțiale peste șirul A. Pentru o interogare (x, y), parcurgem toate perechile (i, j) cu $x \leq i \leq j \leq y$ și verificăm, cu ajutorul vectorului de sume parțiale, dacă suma pe intervalul $[i, j]$ este K.

Soluție O($N \times Q \times log(N)$)

Observăm că pentru un i fixat sumele pe intervalele cu capătul stâng pe poziția i sunt monoton crescătoare, prin urmare pentru fiecare i putem căuta binar intervalul în care suma pe interval este egală cu K.

Soluție O($N \times Q$)

Folosind ideea de la subpunctul anterior, putem utiliza un algoritm de tip Two Pointers, în locul căutărilor binare.

Soluție O((N+Q) $\times log(N)$)

Reținem un vector POZ ce conține acele poziții j ($1 \leq j \leq N$) pentru care $A_j = 1$. Orice secvență cu exact K elemente egale cu 1 în care cea mai din stânga poziție egală cu 1 este $POZ[i]$ va conține cu siguranță elementele $POZ[i]$, $POZ[i+1]$, $POZ[i+2]$, ... $POZ[i+K-1]$. Prin urmare, numărul de secvențe ce conțin exact pozițiile $POZ[i]$, $POZ[i+1]$, $POZ[i+2]$, ... $POZ[i+K-1]$ este $BLOCK[i] = (POZ[i] - POZ[i-1]) \times (POZ[i+K] - POZ[i+K-1])$.

Pentru un interval (x,y) corespunzător unei interogări vom nota cu P_1, P_2, ... P_M pozițiile elementelor cu valoare 1 care apar în intervalul [x,y]. Pentru P_2, P_3, ... P_{M-K} (dacă există vreuna de acest tip) este suficient să însumăm valorile $BLOCK$ corespunzătoare acelor poziții. Pentru P_1, contribuția acestei poziții la numărul total de secvențe este $(P_1 - x + 1) \times (P_{K+1} - P_K)$, iar în mod asemănător putem calcula și contribuția poziției P_{M-K+1}. Un caz particular ce trebuie tratat este atunci când $P_1 = P_{M-K+1}$.

Întrucât valorile P_1, P_2, ... P_M reprezintă, de fapt, un interval din vectorul POZ, acestea pot fi determinate folosind căutare binară pentru fiecare interogare, iar calculul sumei totale va utiliza, de asemenea, un vector de sume parțiale pentru $BLOCK$.

Problema 2. Iscodituri și primeneli (IP)

Propusă de: prof. Dan Pracsiu, Liceul Teoretic "Emil Racoviță" Vaslui

Vom nota prin $a[i..j]$ secvența formată din elementele $a[i]$, $a[i+1]$, ... $a[j]$. Vom nota cu $R = sqrt(n)$ și numim *bucket* o secvență $a[i..j]$ de lungime R.

O observație importantă este că dacă numerele din șir sunt ≤ 200000, atunci numărul maxim de divizori al unui număr natural este de cel mult 160. În consecință, de la început vom construi cu un algoritm de tip Ciurul lui Eratostene vectorul d de lungime 200000 în care $d[i]$ va memora numărul de divizori ai lui i, pentru orice $1 \leq i \leq n$.

În vector, la orice poziție i ($0 \leq i \leq n-1$) valoarea $a[i]$ o vom înlocui cu numărul de divizori ai lui $a[i]$, iar fiecare actualizare (primeneală) a valorii x de pe poziția i se va înlocui cu numărul de divizori, adică cu $d[x]$.

Subtask 1 - soluție propusă de stud. Alin-Gabriel Răileanu

Utilizăm tehnica *Square Root Decomposition* pentru a împărți șirul a în bucket-uri de lungimi $R = \sqrt{n}$.
Pentru fiecare bucket i, vom reține următorul tablou:

- $fr[i][j]$ = câte numere din bucket-ul i au exact j divizori

În continuare:

- Pentru fiecare operație de tip primeneală, tabloul fr poate fi actualizat în $O(1)$ pentru bucket-ul în care se află poziția din operație. Vom scădea frecvența numărului vechi de divizori pentru această poziție și vom incrementa frecvența numărului de divizori pentru noua valoare.

- Pentru fiecare operație de tip iscoditură, fiecare bucket inclus complet în intervalul dat vaa actualiza răspunsul în $O(d)$ (parcurgând fiecare număr de diviori posibil), iar pentru extremitățile care nu sunt luate în calcul prin acestă procedură, răspunsul poate fi actualizat în $O(1)$.

Cum pentru fiecare operație de tip primeneală, numărul de divizori poate fi calculat în $O(\sqrt{V}al)$, complexitatea finală pentru acest tip de operații va fi $O(\sqrt{V}al))$.
Pentru fiecare operație de tip iscoditură, vom parcurge cel mult $O(N/R)$ bucket-uri și $O(R)$ elemente individuale. Astfel, complexitatea finală pentru acest tip de operație va fi $O(d \times (N/R) + R)$.
Diferența dintre acest subtask și soluția pentru 100 este dată, în principal, de optimizarea operațiilor de tip iscoditură.

Subtask 2 - soluție propusă de stud. Daniel Gheorghe

Pentru subtask-ul 2, nu există update-uri și putem aplica următorul algoritm. Acumulăm răspunsul pentru query-ul i în $ans[i]$. Fixăm pe rând numărul de divizori d (luând valori de la 1 la numărul maxim de divizori). Construim un nou șir unde punem 1 pe pozițiile j unde $A[j]$ are exact d divizori, iar 0 pe celelalte, și îi calculăm sumele parțiale. Apoi, considerând query-ul i, adunăm la $ans[i]$ suma pe subsecvența $x[i] \ldots y[i]$ doar dacă $d \leq d[i]$. Suma pe subsecvență se află ușor din sumele parțiale în O(1). Complexitatea algoritmului este O(numărul maxim de divizori $\times$(n + m)).

Subtask 3

Deoarece n și m sunt cel mult 1000, fiecare actualizare (primeneală) 1 i x se face în O(1) scriind $a[i] = d[x]$, iar fiecare interogare (iscoditură) 2 x y D se face în complexitate O(n) parcurgând secvența $a[x..y]$ și numărând valorile mai mici sau egale cu D. Complexitatea totală în acest subtask este O($n \times m$).

Subtask 4

Utilizăm tehnica *Square Root Decomposition* pentru a împărți șirul a în bucket-uri de lungimi $R = sqrt(n)$. Deoarece $n \leq 122500$, atunci numărul maxim de bucket-uri este 350.
Construim două matrice fr și sp, ambele cu 350 de linii (câte o linie pentru fiecare bucket) și 160 de coloane (câte o coloană pentru fiecare divizor $1 \leq D \leq 160$), în care
$fr[i][j]$ = câte numere din bucket-ul i ($0 \leq i \leq n/R$) au exact j divizori
$sp[i][j] = fr[i][0] + fr[i][1] + \ldots + fr[i][j]$ (sumele parțiale)

Fiecare operație de actualizare (primeneală) 1 i x presupune identificarea bucket-ului b din care face parte elementul $a[i]$, apoi decrementarea frecvenței $fr[b][a[i]]$, incrementarea frecvenței $fr[b][d[x]]$, abia apoi se actualizează $a[i] = d[x]$. Până acum avem O(1) complexitate, dar, pentru că s-a modificat fr în bucketul b, trebuie refăcute sumele parțiale în acest bucket, deci vom avea O(R) pe operație. În concluzie, fiecare operație primeneală va avea complexitatea O($sqrt(n)$).

Pentru fiecare operație de interogare (iscoditură) 2 x y D, trebuie să identificăm bucket-urile b_x și b_y unde se află pozițiile x și y. Avem două cazuri:

- b_x și b_y sunt egale, deci x și y sunt în același bucket, caz în care parcurgem efectiv secvența $a[x..y]$ pentru a afla câte valori sunt mai mici sau egale cu D. Complexitate O(R);
- b_x și b_y sunt diferite, deci x și y sunt în bucket-uri diferite. Identificăm bucket-urile care sunt întregi, acestea fiind b_{x+1}, b_{x+2}, ... b_{y-1} și calculăm în O(R) numărul de elemente din aceste bucket-uri folosind sumele parțiale. Rămân eventual câteva elemente din bucket-urile b_x și b_y. Aceste două părți de bucket-uri se parcurg brut, în O(R), folosind cele două secvențe corespunzătoare din vectorul a. În consecință fiecare operație de tip iscoditură se rezolvă în complexitate O($sqrt(n)$).

Complexitatea finală va fi O($m \times sqrt(n)$), suficientă pentru a obține 100 de puncte.

Soluție alternativă

Problema se poate rezolva tot de 100 de puncte dacă în loc să utilizăm vectori de frecvențe, împărțim vectorul în bucket-uri care se păstrează sortate. Fiecare operație de primeneală înseamnă că se modifică o valoare dintr-un bucket, iar în acel bucket elementele se reordonează în complexitate O(R), deoarece nu sortăm efectiv tot bucket-ul, ci eliminăm valoarea veche a lui $a[i]$ și facem inserție pentru noua valoare a lui $a[i]$. Fiecare operație de iscoditură înseamnă că vom face pe fiecare bucket câte o căutare binară în O($R \times logR$). Avantajul la această soluție este că nu folosim decât vectorul a și un alt vector b, tot de lungime n, în care păstrăm bucket-urile sortate. Dezavantajul este că se mărește puțin complexitatea pe operația de iscoditură, dar se pot obține astfel tot 100 de puncte.

Problema 3. Model

Propusă de: prof. Claudiu-Cristian Gorea-Zamfir, Liceul Teoretic de Informatică "Grigore Moisil" Iași

Foaia de matematică este parcursă plecând din pătrățelul (1,1), deplasarea realizându-se în 8 direcții posibile:

- spre SE, atingem ultima linie, direcția se schimbă spre NE;
- spre SE, atingem ultima coloană, direcția se schimbă spre SV;
- spre SV, atingem ultima linie, direcția se schimbă spre NV;
- spre SV, atingem prima coloană, direcția se schimbă spre SE;
- spre NE, atingem prima linie, direcția se schimbă spre SE;
- spre NE, atingem ultima coloană, direcția se schimbă spre NV;
- spre NV, atingem prima linie, direcția se schimbă spre SV;

- spre NV, atingem prima coloană, direcția se schimbă spre NE.

Se poate demonstra că, aplicând acest procedeu, întotdeauna vom ajunge într-un colț al foii de matematică. Pentru aceasta, putem vizualiza această problemă într-un mod mai simplu. În loc să "îndoim" traiectoria lui Octav când lovește o margine, ne putem imagina că foaia se oglindește la infinit, iar Octav merge în linie dreaptă pe o grilă infinită. În această grilă extinsă, Octav pleacă din $(1, 1)$ și se deplasează în diagonală. Un "colț" în grila noastră extinsă apare la fiecare poziție multiplu de $(N - 1)$ pe verticală și $(M - 1)$ pe orizontală. Deci, Octav ajunge într-un colț atunci când traiectoria sa liniară atinge un punct (x, y) unde $x - 1$ este un multiplu de $M - 1$ (marginea stângă/dreaptă) și $y - 1$ este un multiplu de $N - 1$ (marginea de sus/jos). Deci Octav va ajunge într-un colț după $cmmmc(N - 1, M - 1) + 1$ pași.

Cerința 1.

Se simulează pas cu pas deplasarea și se marchează într-o matrice pătrățelele prin care trecem. Se contorizează la final numărul de pătrățele marcate (acestea vor fi considerate în continuare ziduri, adică poziții prin care trecerea nu este posibilă).

Cerințele 2, 3, 4.

Pentru a determina total numărul de modele obținute, parcurgem matricea în care am marcat zidurile, pe linii de sus în jos, iar fiecare linie de la stânga la dreapta. Când identificăm o poziție care nu este zid și nici nu aparține unui alt model deja identificat, deducem că acolo începe un nou model. Datorită modului în care parcurgem matricea, deducem că poziția curentă corespunde pătrățelului reprezentativ al acestui model. Contorizăm modelul identificat (pentru cerința 2), apoi identificăm modelul printr-un algoritm de tip *Fill*. Pe parcursul algoritmului de tip FILL vom reține într-un vector pozițiile pătrățelelor care fac parte din acest model, marcându-le în același timp în matricea în care am marcat atât zidurile, cât și modelele deja parcurse. Numărul de elemente memorate în vector reprezintă aria modelului.

Pentru a rezolva cerințele 3 și 4 vom compara modelul curent cu toate modelele distincte identificate deja. Dacă modelul curent nu este egal cu niciunul dintre acestea, el constituie o nouă categorie, pe care o contorizăm (pentru cerința 3) și o reținem (pentru cerința 4). Dacă modelul curent este egal cu un model din una dintre categoriile deja memorate, doar incrementăm numărul de exemplare din categoria respectivă.

Pentru a verifica dacă două modele sunt egale vom compara mai întâi ariile acestora. Dacă ariile sunt egale, pentru a verifica dacă cele două modele se suprapun perfect, facem o translație imaginară. Pentru aceasta vom determina două diferențe: diferența dintre indicii liniilor pătrățelelor reprezentative ale celor două modele, respectiv diferența dintre indicii coloanelor pătrățelelor reprezentative. Pentru a verifica dacă toate celelalte pătrățele din model au aceleași diferențe pe linie, respectiv coloană cu pătrățelul corespondent din celălalt model trebuie să reținem pătrățelele din fiecare model în ordinea crescătoare a pozițiilor acestora. Astfel, este suficient să parcurgem simultan cele două modele și să comparăm diferențele.

Complexitatea algoritmului de *Fill* este $O(N \times M)$, la care se adaugă operațiile de comparare a modelului curent cu modelele distincte deja determinate, dar numărul de modele distincte care se pot obține în condițiile problemei este cel mult egal cu 7.

Echipa

Problemele pentru această etapă au fost pregătite de:

- prof. Emanuela Cerchez, Colegiul Național "Emil Racoviță" Iași
- prof. Dan Pracsiu, Liceul Teoretic "Emil Racoviță" Vaslui
- prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj
- prof. Claudiu-Cristian Gorea-Zamfir, Liceul Teoretic de Informatică "Grigore Moisil" Iași
- șef lucrări dr. Mihai Nan, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica Bucureşti
- prof. Ionel-Vasile Piț-Rada, Colegiul Național "Traian", Drobeta Turnu Severin
- prof. Sandor Lukacs, Colegiul Național "Onisifor Ghibu", Oradea
- prof. Marinel Paul Șerban, Centrul Județean de Excelență
- stud. Aureliu Valentin Antonie, Universitatea Națională de Știință și Tehnologie Politehnica Bucureşti, Facultatea de Automatică și Calculatoare
- stud. Andrei Boacă, Universitatea Alexandru Ioan Cuza, Facultatea de Informatică Iași
- stud. Eduard-Lucian Pîrțac, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași
- stud. Alin Gabriel Răileanu, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași
- stud. Daniel Gheorghe, Facultatea de Matematică-Informatică, Universitatea București
- prof. Dorin-Mircea Rotar, Colegiul Național "Samuil Vulcan" Beiuș
- prof. Adrian Doru Pintea, Inspectoratul Școlar Județean Cluj
- Bogdan-Ioan Popa, Societatea pentru Performanță și Excelență în Informatică
- stud. Valentin Porumb, Universitatea București, Facultatea de Matematică și Informatică
- stud. Rareș-Andrei Cotoi, Universitatea "Babeș-Bolyai", Facultatea de Matematică și Informatică Cluj-Napoca
- stud. Robert-Cristian Mitri, Universitatea Națională de Științe și Tehnologie Politehnica Bucureşti, Facultatea de Automatică și Calculatoare
- stud. Jonathan Mogovan, Universitatea "Babeș-Bolyai", Facultatea de Matematică și Informatică Cluj-Napoca
- stud. Răzvan-Alexandru Rotaru, Universitatea "Alexandru Ioan Cuza", Facultatea de Informatică Iași