TABĂRA DE PREGĂTIRE ȘI SELECȚIE A LOTULUI NAȚIONAL DE INFORMATICĂ MAI 2023, SLATINA

DESCRIEREA SOLUȚIILOR, BARAJ 2, JUNIORI

Ziua 2 — Problema 1: Bossfight

Propusă de: Stud. Tulbă-Lecu Theodor Gabriel, Universitatea Politehnică din București

Notăm cu x o poziție din șir și cu P_x valoarea de la poziția x.

Soluție în $\mathcal{O}(N^3)$: O prima soluție constă în verificarea tuturor tripletelor de numere posibile și numărarea celor valide. Această soluție ar trebui sa obțină în jur de 11 puncte.

Soluție în $\mathcal{O}(N^2 \log N)$: Plecăm de la soluția precedentă, în care selectăm toate tripletele de numere (x, y, z) cu proprietatea că $x < y < z$, $P_x < P_y < P_z$ și $P_x + P_y + P_z \leq S$. Observăm că odată cu selectarea pozițiilor x și y, putem alege orice valoare P_z din dreapta lui y cuprinsă între $[P_y + 1, S - P_x - P_y]$ (sau similar pentru y și z fixate, iar x determinat). O soluție bazată pe această idee poate obține de la 38 de puncte în sus, în funcție de implementare.

Arbore Fenwick (AIB). Iterăm prin fiecare boss central, de la stînga la dreapta. Pe măsură ce facem asta, menținem o *colecție* cu boșii din stînga centrului. Pentru un centru fixat y, iterăm prin fiecare boss din dreapta, z. Pentru fiecare pereche (y, z) cu $P_y < P_z$, un boss corect din stînga trebuie să aibă puterea cel mult egală cu $(P_y - 1)$ și cel mult egală cu $(S - P_y - P_z)$. După ce terminăm de procesat centrul y, adăugăm P_y la colecție și trecem la centrul $y + 1$.

Cum implementăm colecția? Avem nevoie de o structură care să admită în $\mathcal{O}(\log N)$ operațiile (1) adaugă o putere la colecție și (2) raportează numărul de puteri mai mici sau egale cu un P dat. Un AIB atinge acest scop, cu observația că puterile pot fi 10^9, deci trebuie normalizate. O tehnică simplă de normalizare le ține sortate într-un vector. Prin diferența $(S - P_y - P_z)$ pot lua naștere și valori care nu sînt printre cele N puteri date la intrare. Pe acestea le căutăm binar în vectorul sortat de puteri.

O optimizare care scade timpul de rulare cu circa 40% pornește de la următoarea observație. Dacă $P_y - 1 < S - P_y - P_z$, atunci știm că valoarea care ne limitează este $P_y - 1$ și nu mai este nevoie să căutăm binar poziția lui $(S - P_y - P_z)$ printre puteri.

Căutare binară. Putem mai întâi să selectăm valoarea de la mijloc (y), iar apoi sortăm crescător secvența de valori din dreapta lui y. Pentru orice x din stânga, putem căuta binar cea mai mică (P_{z1}), respectiv cea mai mare (P_{z2}) valoare pe care o poate lua P_z. Numărul de soluții pentru z este egal cu numărul de valori din șir cuprinse între z_1 și z_2.

Diverse optimizări se pot face la acestă soluție. Odată cu mutarea poziției y, valorile din dreapta nu trebuiesc sortate din nou. În schimb, se poate adăuga / șterge direct valoarea care este afectată de modificarea poziției y în complexitate $\mathcal{O}(N)$. Această soluție ar trebui sa obțină până la 78 de puncte.

Soluție în $\mathcal{O}(N^2)$: Plecăm de la soluția precedentă, în care mai întâi setăm valoarea y, apoi x, iar la final căutam binar valorile posibile ale lui P_z. Motivul pentru care este nevoie să căutăm binar este acela că valorile lui P_x alternează. Dar dacă am sorta crescător valorile posibile ale lui P_x, respectiv valorile posibile ale lui z, atunci am putea identifica tripletele folosind o tehnică de tipul „two-pointers". Pe măsură ce creștem valoarea lui P_x, vom scădea valoarea lui P_z.

Pornim inițial cu vectorul de valori posibile ale lui x care va conține strict prima valoare din șir, în timp ce vectorul de valori posibile ale lui z va conține ultimele $n - 2$ valori din șir, sortate crescător. Plecăm cu y de pe poziția a doua, și calculăm toate tripletele posibile cu

tehnica prezentată mai sus. Apoi, când trecem la y pe poziția a treia, ștergem această valoare din vectorul de valori posibile ale lui z, și inserăm valoarea de pe poziția a doua în vectorul de valori posibile ale lui x. Putem face ștergerea și inserarea naiv, în $\mathcal{O}(n)$. Nu este nevoie să căutăm soluții mai eficiente, întrucât tehnica „two-pointers" necesită oricum $\mathcal{O}(n)$. Repetăm acest proces pentru fiecare valoare posibilă a lui y.

Această soluție, cât timp este implementată eficient, ar trebui să obțină 100 de puncte.

Ziua 2 — Problema 2: Veverițe

Propusă de: Prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian", Botoșani

- Pentru $N = 3$ avem 2 trasee.
- Pentru $N = 5$ avem 16 trasee.
- Pentru $N = 7$ avem 564 trasee.
- Pentru $N = 9$ avem 93866 trasee.

Soluția pentru toate subtask-urile este backtracking. Se generează prin backtracking toate traseele lui Chip în ordine lexicografică având grijă ca pentru orice mutare face Chip să consemnăm și mutarea simetrică față de centrul matricei pe care o face Dale. Punctajele obținute depind de modul în care se implementează backtracking-ul după cum urmează.

Soluție bactracking neoptimizat. Se consideră o matrice în care vom marca pozițiile prin care a trecut una dintre veverițe. La fiecare moment verificăm ce poziții vecine celei în care se găsește Chip nu au fost vizitate și îl mutăm pe Chip acolo. Avem grijă sa marcăm drept vizitate poziția lui Chip și poziția simetrică în care se găsește Dale. Deoarece numărul de soluții este foarte mic pentru $N \leq 7$ cu această abordare se vor obține punctele corespunzătoare subtaskurilor 1, 2, 3 și 4.

Soluție bactracking neoptimizat, dar oprit la poziția maximă din query-uri. Este exact aceeași abordare de mai sus cu observația suplimentară că la fiecare soluție validă obținută incrementăm un contor și la fiecare apel al funcției recursive de backtracking verificăm dacă acel contor a atins poziția maximă din toate query-urile (situație în care returnăm).

În acest fel nu vor fi generate toate soluțiile, ci doar primele în ordine lexicografică de care avem nevoie pentru a răspunde la întrebările din test.

Această abordare rezolvă subtask-urile 5 și 6.

Backtracking optimizat. Pentru a rezolva subtask-ul maxim vom implementa în alt mod backtracking-ul. În loc să folosim o „matrice de vizitare", vom construi inițial o matrice în care vom consemna numărul de vecini nevizitați. Astfel pozițiile din colț vor avea 2 vecini, celelalte poziții de pe liniile și coloanele exterioare 3 vecini, iar restul pozițiilor vor avea 4 vecini. În momentul în care una dintre veverițe va pleca dintr-o poziție, numărul vecinilor acelei poziții va deveni 0 iar numărul vecinilor pentru fiecare poziție vecină nevizitată va descrește cu 1.

În acest mod vom putea controla dacă o poziție a fost vizitată (are 0 vecini) sau nu.

O primă optimizare ar fi aceea că, dacă o poziție are la intrarea în ea un vecin marcat cu 2, din poziția curentă va trebui să trecem obligatoriu în acel vecin (altfel nu s-ar mai putea ajunge mai târziu la acel vecin). Implementată îngrijit, această optimizare este suficientă pentru a obține 100p.

A doua optimizare este mai subtilă și se referă la situația când traseul nu va fi valid pentru că separă matricea în două zone "neconexe" și astfel, dacă la un moment dat intrăm în una dintre ele, cealaltă nu va mai putea fi parcursă.

Pentru a surprinde astfel de situații, trebuie să fim foarte atenți atunci când traseul lui Chip se găsește la un moment dat pe o linie sau coloană exterioară. Pentru a nu „deconecta" pozițiile nevizitate în zone inaccesibile una alteia, se poate observa că pe pozițiile de pe prima linie și în continuare de pe ultima coloană trebuie să apară (dacă apar) în traseu strict în ordinea de la Chip spre Dale. Similar pentru pozițiile primei coloane urmate apoi de ultima

linie. Altfel se va crea o zonă nevizitată în care Chip nu ar trebui să intre, iar Dale nu mai are cum să intre după cum urmează:

- Pe de o parte, dacă Chip ar ajunge acolo, atunci el însuși ar avea traseul spre centrul matricei blocat de propriul să traseu.
- Pe de altă parte, Dale are deja blocat traseul spre acea poziție de traseul curent al lui Chip.

Ziua 2 — Problema 3: Ashima

Propusă de: Prof. Bunget Mihai, Colegiul Național „Tudor Vladimirescu", Târgu Jiu

Soluție brută 16p. Se parcurge matricea M pe diagonale, din stânga-jos spre dreapta-sus, completând-o cu elementele șirului A. Apoi, pentru fiecare cerință, se calculează suma elementelor submatricei determinate de liniile l_1 și l_2, respectiv coloanele c_1 și c_2.

Complexitate O($L \cdot C + Q \cdot L \cdot C$)

Soluție brută cu sume parțiale 37p. Se parcurge matricea M pe diagonale, din stânga-jos spre dreapta-sus, completând-o cu elementele șirului A. Apoi se calculează sumele parțiale pentru matricea M folosind recurențele $SC_j = SC_j + M_{i,j},\ M_{i,j} = M_{i,j-1} + SC_j$, pentru fiecare $i \in \{1, 2, ..., L\},\ j \in \{1, 2, ..., C\}$, unde SC_j este suma elementelor pe coloana j până la poziția curentă. Calculul sumei elementelor submatricei determinate de liniile l_1 și l_2, respectiv coloanele c_1 și c_2, se face cu formula $M_{l_2,c_2} - M_{l_2,c_1-1} - M_{l_1-1,c_2} + M_{l_1-1,c_1-1}$.

Complexitate O($L \cdot C + Q$)

Soluție optimă 100p. Pentru a răspunde la o cerință, se determină pentru fiecare diagonală stânga-jos dreapta-sus a submatricei determinate de liniile l_1 și l_2, respectiv coloanele c_1 și c_2, suma elementelor de pe această diagonală. Pentru aceasta se determină poziția în șirul A a primului și ultimului element de pe diagonală. Pentru a determina poziția elementului $M_{i,j}$ în șirul A, se află câte elemente sunt situate pe diagonalele anterioare ale matricei până la poziția acestui element. Să notăm cu h, respectiv u, pozițiile primului, respectiv ultimului element de pe o diagonală în șirul A. Atunci suma elementelor de pe această diagonală a submatricei va fi $S_u - S_{h-1}$, unde am notat cu S_n suma primelor n elemente din șirul A.

În funcție de valoarea exponentului K, valoarea lui S_n este:

- pentru $K = 0$ avem $S_n = 2^{n+1} - 2$;
- pentru $K = 1$ avem $S_n = 2^{n+1} \cdot (n-1) + 2$;
- pentru $K = 2$ avem $S_n = 2^{n+1} \cdot (n^2 - 2 \cdot n + 3) - 6$;
- pentru $K = 3$ avem $S_n = 2^{n+1} \cdot (n^3 - 3 \cdot n^2 + 9 \cdot n - 13) + 26$.

Calculul acestor sume se poate face astfel:

- Pentru $K = 0$ avem $S_n = 2 + 2^2 + 2^3 + ... + 2^n$. Înmulțind această relație cu 2 obținem $2 \cdot S_n = 2^2 + 2^3 + ... + 2^n + 2^{n+1}$. Scăzând din aceasta relația inițială obținem $S_n = 2^{n+1} - 2$.
- Pentru $K = 1$ avem $S_n = 1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + ... + n \cdot 2^n$. Înmulțind această relație cu 2 obținem $2 \cdot S_n = 1 \cdot 2^2 + 2 \cdot 2^3 + ... + (n-1) \cdot 2^n + n \cdot 2^{n+1}$. Scăzând din aceasta relația inițială obținem $S_n = n \cdot 2^{n+1} - 2 - 2^2 - ... - 2^n = n \cdot 2^{n+1} - (2^{n+1} - 2) = 2^{n+1} \cdot (n-1) + 2$.
- Pentru $K = 2$ avem $S_n = 1^2 \cdot 2 + 2^2 \cdot 2^2 + 3^2 \cdot 2^3 + ... + n^2 \cdot 2^n$. Înmulțind această relație cu 2 obținem $2 \cdot S_n = 1^2 \cdot 2^2 + 2^2 \cdot 2^3 + ... + (n-1)^2 \cdot 2^n + n^2 \cdot 2^{n+1}$. Scăzând din aceasta relația inițială obținem $S_n = n^2 \cdot 2^{n+1} - 1 \cdot 2 - 3 \cdot 2^2 - ... - (2 \cdot n - 1) \cdot 2^n = n^2 \cdot 2^{n+1} - 2 \cdot (1 \cdot 2 + 2 \cdot 2^2 + ... + n \cdot 2^n) + 2 + 2^2 + ... + 2^n = n^2 \cdot 2^{n+1} - 2 \cdot (2^{n+1} \cdot (n-1) + 2) + (2^{n+1} - 2) = 2^{n+1} \cdot (n^2 - 2 \cdot n + 3) - 6$.
- Pentru $K = 3$ avem $S_n = 1^3 \cdot 2 + 2^3 \cdot 2^2 + 3^3 \cdot 2^3 + ... + n^3 \cdot 2^n$. Putem scrie $S_n = 2 \cdot S_n - S_n = 1^3 \cdot 2^2 + 2^3 \cdot 2^3 + 3^3 \cdot 2^4 + ... + n^3 \cdot 2^{n+1} - (1^3 \cdot 2 + 2^3 \cdot 2^2 + 3^3 \cdot 2^3 + ... + n^3 \cdot 2^n) = 2^{n+1} \cdot n^3 - 3 \cdot (1^2 \cdot 2 + 2^2 \cdot 2^2 + 3^2 \cdot 2^3 + ... + n^2 \cdot 2^n) + 3 \cdot (1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + ... + n \cdot 2^n) - (2 + 2^2 + 2^3 + ... + 2^n)$ și se folosesc formulele anterioare.

Complexitate O($Q \cdot max(l_2 - l_1 + c_2 - c_1) \cdot log(L \cdot C)$)

Optimizare (prof. Adrian Panaete, Cătălin Frâncu). O optimizare elegantă, dar care nu este necesară pentru a obține 100p, calculează puterile lui 2 în timp constant, eliminînd factorul logaritmic. Pentru fiecare dintre cele $L + C - 1$ diagonale precalculăm puterea lui 2 necesară pentru primul termen de pe acea diagonală (așadar pe prima coloană și pe ultima linie din matrice). Cînd avem nevoie de puterea 2^k pentru cel de-al k-lea termen, înmulțim puterea precalculată pe diagonala lui k cu 2^d, unde d este distanța dintre elementul k și primul element de pe acea diagonală. Această optimizare necesită doi vectori de cîte 200.000 de elemente, deci 1,6 MB de memorie.

Această soluție poate fi împinsă mai departe pornind de la Mica Teoremă a lui Fermat, care ne spune că $2^{MOD-1} = 1 \pmod{MOD}$. Așadar, pentru a calcula 2^k, putem începe prin a calcula $k' = k \mod (MOD - 1)$. Astfel exponentul maxim va fi $1\,000\,000\,005 < 2^{30}$. Acum construim doi vectori de cîte $2^{15} = 32\,768$ de elemente fiecare: $a[i] = 2^{32\,768 \times i}$ și $b[i] = 2^i$. Atunci $2^k = a[k'/32\,768] \times b[k' \mod 32\,768]$. Această optimizare necesită $2 \times 32\,768 \times 4 = 256$ KB de memorie.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Panaete Adrian, Colegiul Național „A.T. Laurian”, Botoșani
- Prof. Șerban Marinel, Colegiul Național „Emil Racoviță”, Iași
- Prof. Chește Ciprian, Liceu Tehnologic „Grigore C. Moisil”, Buzău
- Prof. Nodea Gheorghe Eugen, Colegiul Național „Tudor Vladimirescu”, Târgu Jiu
- Prof. Pițt-Rada Ionel-Vasile, Colegiul Național „Traian”, Drobeta-Turnu Severin
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova
- Prof. Costineanu Raluca, Colegiul Național „Ștefan cel Mare”, Suceava
- Prof. Cerchez Emanuela, Colegiul Național „Emil Racoviță”, Iași
- Prof. Bunget Mihai, Colegiul Național „Tudor Vladimirescu”, Târgu Jiu
- Prof. Boian Flavius, Colegiul Național „Spiru Haret”, Târgu Jiu
- Prof. Szabo Zoltan, Inspectoratul Școlar Mureș
- Inst. Frâncu Cătălin, Clubul Nerdvana, București
- Stud. Tulbă-Lecu Theodor Gabriel, Universitatea Politehnică din București
- Stud. Bogdan Vlad Mihai, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Pop Ioan Cristian, Universitatea Politehnică din București
- Stud. Onuț Andrei, Universitatea Yale, S.U.A.

Cursurile de pregătire au fost susținute de:

- Prof. Marius Nicoli, Colegiul Național „Frații Buzești”, Craiova
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova
- Inst. Frâncu Cristian, Clubul Nerdvana, București
- Stud. Popa Bogdan Ioan, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Dăscălescu Ștefan-Cosmin, Facultatea de Matematică și Informatică, Universitatea București

Comisia tehnică a fost formată din:

- Prof. Opriță Petru Simion, Liceul „Regina Maria”, Dorohoi
- Prof. Bolohan Mihai, Liceul „Regina Maria”, Dorohoi