OLIMPIADA NATIONALA DE INFORMATICA, FAZA NATIONALA, 2023, CLASELE 11-12

Problema 1: Biom

Propusă de: stud. Matei Tinca

Subtask 1 (12 puncte). Putem genera recursiv toate drumurile și să alegem drumul care are costul minim. O observație importantă este să avem grijă să nu trecem de mai multe ori prin același cub, altfel programul va intra într-o buclă infinită.

Subtask 2 (8 puncte). Dacă ne uităm la structura șirului de caractere, putem observa că orice mișcare facem se va deplasa într-o celulă adiacentă. Astfel, nu va avea rost să mergem niciodată în stânga. Când suntem într-un grup de litere, mișcările posibile pe care le putem face sunt să mergem cu o poziție la dreapta cu cost A sau C, iar optim este să alegem costul minim dintre ele. Când vrem să trecem de la un grup de litere, singura mișcare pe care trebuie să o facem este cea de cost A. Soluția acestei soluții este $O(N)$.

Subtask 3 (26 puncte). Pentru acest subtask, cum B și D sunt foarte mari, nu va avea niciodată rost să ne mișcăm înspre stânga. Astfel, toate mișcările pe care le facem vor fi spre dreapta. Putem modela o soluție folosind programare dinamică. Fie d_N costul minim pentru a ajunge în cubul cu indicele i. Fie x un cub astfel încât dacă fac mișcarea cu cost C, atunci Steve ajunge în cubul i. Dacă acesta există, atunci:
$d_i = min(d_x + C, d_{i-1} + A)$.
Dacă x nu există, atunci:
$d_i = d_{i-1} + A$
Șirul d va fi inițializat cu valoarea $d_1 = 0$, iar restul șirului se va calcula cu recurența de mai sus. Soluția problemei va fi valoarea d_N. Complexitatea acestei soluții este $O(N)$.

Subtask 4 (24 puncte). În acest subtask, nu va avea niciodată rost să folosim o mișcare de cost 10^9, deci toate mișcările pe care le facem vor avea cost 1.

Din acest punct, vom considera șirul de caractere ca și un graf. Fiecare cub i va fi reprezentat printr-un nod i, de la care avem 4 muchii: de la i la $i+1$, de la i la $i-1$, de la i la $next_i$ și de la i la $last_i$, unde $next_i$ este primul nod de la dreapta cu aceeași literă și $last_i$ este primul nod de la stânga cu aceeași literă. Muchiile le includem în graf dacă acestea există și dacă au costul 1.

Astfel, acest subtask este o aplicație a algoritmului de BFS pe graful de mai sus. Soluția acestei soluții este $O(N)$.

Subtask 5 (11 puncte). Ca și la subtaskul 4, vom construi același graf, doar că includem și muchiile care au cost 0. Practic, avem un graf cu muchii care au cost 0 sau 1.

Pentru acest subtask, atunci când ne aflăm într-un nod, trebuie să explorăm toate nodurile în care putem ajunge folosind doar muchii de cost 0. Putem combina soluția anterioară cu un DFS. Când ne aflăm la nodul i, explorăm cu algoritmul DFS componenta conexă plimbându-ne doar prin muchiile de cost 0, iar din DFS să adăugăm în coada nodurile care parcurg muchii de cost 1.

Putem implementa soluția de mai sus elegant, fără a implementa algoritmul DFS folosind un deque în loc de coadă. Atunci când parcurgem o muchie de cost 0, vom adăuga nodul explorat la începutul deque-ului, altfel vom adăuga nodul la sfârșitul deque-ului.

Soluția aceasta are complexitate O(N).

Subtask 6 (12 puncte). Din acest punct, vom considera același graf, incluzând toate muchiile care există cu costul lor. Problema astfel devine o aplicație a găsirii drumului minim în graf. Pentru acest subtask, putem aplica algoritmul Bellman-Ford: când ne aflăm într-un nod, luăm toate muchiile care ies din acest nod, iar de fiecare dată când drumurile noi create sunt mai bune decât costurile din fiecare nod, adăugăm nodul în coadă.

Complexitatea algoritmului Bellman-Ford este $O(NM^2)$, iar pentru că numărul de muchii este cel mult $4N$, atunci complexitatea algoritmului o să fie $O(N^3)$. În practică, această soluție se comportă mult mai bine.

Subtask 7 (8 puncte). Pentru această soluție, vom folosi algoritmul lui Dijkstra. Asemănător ca și la soluția precedentă, vom folosi o structură de date în care ținem nodurile vizitate. De fiecare dată când scoatem un nod din această structură, vom scoate nodul care are cel mai mic cost și actualizăm costurile vecinilor.

Algoritmul lui Dijkstra este de obicei implementat cu un heap, deoarece acesta are complexitatea $O(logN)$ pentru operațiile de care avem nevoie, deci complexitatea acestui algoritm va fi $O(NlogN)$.

Subtask 8 (12 puncte). În graful construit, avem doar patru costuri distincte. O altă observație importantă ce provine din modul în care funcționează algoritmul lui Dijkstra, este că șirul costurilor minime în ordinea în care parcurgem nodurile va fi un șir crescător. Adăugând o constantă la acest șir, vom obține în continuare un șir crescător.

Practic, dacă am vizitat un nod și vrem să actualizăm costurile vecinilor, adăugând la costul nodului în care ne aflăm costul A, avem garanția că toate celălalte costuri la care am adăugat A vor fi mai mici sau egale.

Astfel, putem să ținem minte patru cozi în loc de un heap. Costurile la care adăugăm A se vor duce în prima coadă, costurile la care adăugăm B se duc în a doua coadă ș.a.m.d. Pentru a afla nodul cu cost minim, vom lua nodul minim pentru toate cele patru cozi.

Complexitatea acestei soluții este $O(N)$ și obține punctaj maxim.

Problema 2: XidarTros

Propusă de: Daniel Posdărăscu

Subtask 1 (6 puncte). Știind că orice test care admite soluție conține valori de până in 30, putem încerca toate cele 30^4 posibilități de șiruri inițiale și să aplicăm Radix Sort pentru verificare.

Subtask 2 (6 puncte). Știind că numărul de cifre disponibile este mai mare decât numărul de numere, putem atribui, la fiecare pas, fiecărui număr câte o cifră distinctă de la 0 la $N-1$, în ordinea dată de permutare.

Subtask 3 (11 puncte). Pentru $K = 1$, trebuie generat un șir în care fiecare element este o cifră în baza B. Fie P permutarea primită în input și V valorile șirului inițial. Pentru fiecare $i < N$ cu $P[i] > P[i+1]$, trebuie ca $V[P[i]] < V[P[i+1]]$, iar în cazul în care $P[i] < P[i+1]$, putem avea $V[P[i]] \leq V[P[i+1]]$. Astfel, observăm că nu există soluție doar în cazul în care există cel puțin B astfel de perechi. În cazul în care există soluție, putem pune cifrele începând cu 0 pentru $V[P[1]]$, după care $V[P[i+1]] = \begin{cases} V[P[i]] & \text{dacă } P[i] < P[i+1] \\ V[P[i]]+1 & \text{dacă } P[i] > P[i+1] \end{cases}$

Subtask 7 (28 puncte). Putem observa că sortarea pe fiecare cifră (unități, zeci, etc.) este independentă. Astfel, vom rezolva atribuirea cifrelor în același mod ca la subtaskul 3, cu diferența că vom ține cont de pozițiile precedente ale indicilor. În final, numerele trebuie transformate în baza 10.

Problema 3: Keidei

Propusă de: stud. Vlad-Adrian Ulmeanu

Subtaskurile 1 si 2. Permutam in toate modurile posibile multimea muchiilor si facem bfs/dfs pe arborii rezultati. Complexitate: $O(n!)$.

Subtaskurile cu c = 1 (dfs) ($O(n^3)$). Exista multe solutii cu diferite abordari. Aici o sa ne ocupam de cele care calculeaza involuntar raspunsul pentru orice k.

In general, vrem sa vedem daca un nod particular poate sa fie pe o pozitie anume in parcurgere. Daca intram in dfs intr-un nod x, trebuie sa vizitam oricare $nn \in$ subarborele lui x inainte sa vizitam alt frate de-al lui x.

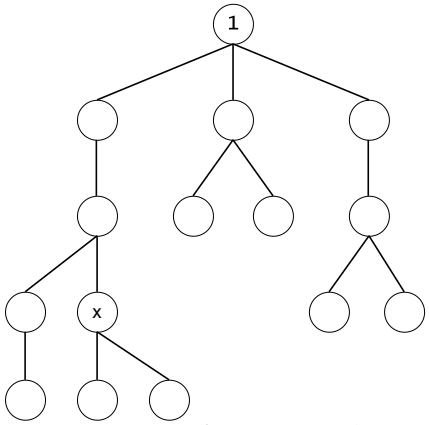

Figure 1. Tatal lui x mai are inca un fiu, cu 2 noduri in subarborele lui. Dupa ce il vizitam pe tatal lui x, putem sa intram direct in x, sau sa intrarziem intrarea cu 2 noduri, vizitandu-l intai pe fratele lui x.

In aceeasi maniera, daca x ar fi avut 3 frati, cu marimile subarborilor a, b, c, atunci am fi putut intarzia intrarea in x cu $0, a, b, c, a+b, a+c, b+c$ sau $a+b+c$ noduri.

Daca avem un rucsac pentru tatal lui x in care am calculat deja pe orice pozitie ar putea fi el intr-un dfs (n pozitii de $0/1$, de fapt conteaza doar cele pana in k inclusiv), atunci putem sa calculam rucsacul lui x astfel:

- adauga pe rand a, b, c la rucsacul tatalui lui x.
- shifteaza oricare valoare din rucsacul tatalui la dreapta cu 1 (pentru a simula coborarea din tata in x).

Initial, rucsacul lui 1 este contine doar valoarea 1. Presupunand ca rucsacul tatalui este calculat, calculam recursiv rucsacurile fiilor lui.

Daca un nod are s fii, atunci trebuie sa aduagma in oricare fiu $s-1$ valori. Consideram ca operatia inserarii unei valori intr-un rucsac este $O(n)$.

In cel mai rau caz, o sa adaugam in oricare nod in afara 1 $n-2$ valori, deci $(n-1)(n-2)$ inserari.

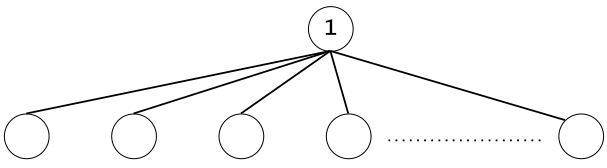

Figure 2. Arbore cu $O(n^2)$ inserari.

Celelalte puncte se pot obtine reducand numarul de inserari in rucsacuri necesare.

Subtaskurile cu c = 1 (dfs) ($O(n^{2.5})$). Intai vom arata o optimizare generica: daca am un nod x cu m fii, x_1, .., x_m.

Notam cu $\circ$ operatia care ne permite adaugarea unei valori *val* intr-un rucsac r, obtinand astfel un rucsac r': $r \circ val = r'$. Notam marimea subarborelui lui y cu $szSub_y$.

Vreau sa calculez $rucsac_x \circ szSub_{x_1} \circ szSub_{x_2} \circ ..szSub_{x_m}$, apoi sa obtin cumva inversul operatiei $\circ$ astfel incat sa pot scoate $szSub_{x_1}$ si sa obtin $rucsac_{x_1}$, etc.

Voi imparti fiii lui x in $O(\sqrt{m})$ galeti. O sa pun intr-o galeata $p \geq [\sqrt{m}]$ fii.

$$buck_0 = rucsac_x \circ szSub_{x_{p+1}} \circ szSub_{x_{p+2}} \circ .. \circ szSub_{x_m}$$

$$buck_1 = rucsac_x \circ szSub_{x_1} \circ .. \circ szSub_{x_p} \circ szSub_{x_{2p+1}} \circ .. \circ szSub_{x_m}$$

..

Intr-o galeata sunt toti fiii in afara celor care ar fi trebuit sa fie acolo.

$$rucsac_{x_1} = buck_0 \circ szSub_{x_2} \circ szSub_{x_3} \circ .. \circ szSub_{x_p}$$

Ca sa calculez galetile pentru tot arborele am nevoie de $O(n\sqrt{n})$ inserari. Ca sa calculez rucsacurile in parte tot de $O(n\sqrt{n})$ inserari am nevoie. Acum complexitatea totala devine $O(n^2\sqrt{n})$. Daca se folosesc optimizari tip bitset se poate obtine punctaj maxim.

Subtaskurile cu c = 1 (dfs) ($O(n^2 \log n)$). Se poate face o optimizare asupra solutiei de $O(n^3)$ specifica problemei. Daca 2 fii au aceeasi marime a subarborilor lor, atunci vor avea acelasi rucsac, deci trebuie sa calculam doar un rucsac si sa-l copiem apoi pentru celalalt fiu.

Incercam sa calculam o recurenta $T(n)$ = numarul maxim de inserari pentru un arbore cu n noduri.

Construim un subarbore de marime n astfel: $[n/2]$ fii de marime 1, iar restul fiilor au marime $2, 3, 4, ..$ (m alti fii).

$$T(n) = m(n/2 + m) + T(2) + T(3) + .. + T(m-1)$$

Pentru $m = \sqrt{n} \Rightarrow T(n) \in \Omega(n^{2.5})$. Totusi, aceasta optimizare se comporta mai bine in practica decat cea cu galeti si poate sa ia si ea punctaj maxim cu bitset.

Cealalta optimizare care se poate face este urmatoarea: daca vreau sa bag intr-un rucsac 3 valori identice x, x, x, pot sa obtin acelasi rezultat daca adaug doar doua valori, x si $2x$ (pentru ca obtin toate rezultatele partiale x, $2x$, $3x$).

Astfel, daca am p valori identice, pot sa fac doar 2 inserari pentru fiecare exponent al lui 2, cel mult $2\log_2 p$ in total.

Intr-un nod cu marimea subarborelui n pot sa fac maxim $\sqrt{n} + 2\log_2 n$ inserari ($\sqrt{n}$ din fiii de marime distincta si maxim $2\log_2 n$ din duplicate).

Relatia de recurenta devine:

$$T(n) = \sqrt{n} + 2\log_2 n + T(szfiu_1) + T(szfiu_2) + .. + T(szfiu_{ult})$$

Punem $2\log_2 n$ pentru fiecare nod (in total $O(n \log n)$ din duplicate) si acum cel mai rau caz se intampla cand arborele este construit astfel incat fiecare $szSub_{fiu}$ este distinct.

FIGURE 3.

Numarul de inserari pe stratul de sub 1 este $O(\sqrt{n}) \cdot (\sqrt{n}-1)$ ($O(\sqrt{n})$ fii, in fiecare bag $\sqrt{n}-1$ valori).

Pe urmatorul strat am $O(\sqrt{n} \cdot \sqrt[4]{n}) \cdot O(\sqrt[4]{n})$ inserari.

..

Suma lor apartine $O(n \log\log n)$.

Partea de la duplicate $O(n \log n)$ domina $O(n \log\log n)$ de la valori distincte, iar complexitatea totala iese $O(n^2 \log n)$.

Subtaskurile cu c = 1 (dfs) ($O(n^2)$). In loc sa tin intr-un rucsac daca pot sa obtin sau nu valorile, tin minte in cate moduri pot sa fiecare valoare in parte.

In acest fel, pot sa scot o valoare din rucsac in $O(n)$.

```
void adD(int x) {
    for (int i = k; i >= x; i--) {
        d[i] += d[i-x];
    }
}

void rmD(int x) {
    for (int i = x; i <= k; i++) {
        d[i] -= d[i-x];
    }
}

//valoare x in rucsac <=> d[x] != 0.
```

Testele nu sunt generate adversarial astfel incat numarul de valori dintr-un rucsac sa fie divizibil cu un MOD particular.

Exista si alta solutie care se foloseste de urmatoarea structura de date.

Subtaskurile cu c = 2 (bfs). Consideram urmatorul exemplu:

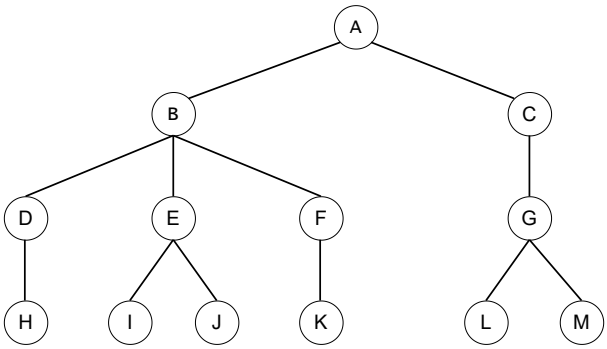

FIGURE 4. Nu putem sa avem in coada urmatoarea configuratie pe ultimul strat: $(K)(ML)(H)(IJ)$. Nu putem avea (ML) intre K si H, pentru ca inserarea in coada a lui B inaintea lui C garanteaza ca H, I, J sunt inaintea lui M, L.

Putem calcula pe ce substrat s se pot afla nodurile care pot sa fie ale k-lea intr-o parcurgere.

Consideram o notiune mai abstracta, greutatea unui nod. La $c = 1$, w_{nod} = marimea subarborelui lui. Aici luam w_x = numarul de noduri din subarborele lui x care se afla pe substratul s.

Daca B intra in coada inaintea lui C, nodurile de pe substratul s care sunt stra-fiii lui C vor avea o intarziere egala cu 4 (H, I, J, K sunt inaintea lui M, L).

Algoritmul este la fel ca la dfs, doar ca nu mai shiftam cu 1 rucsacul cand coboram intr-un fiu.

As vrea sa le multumesc lui Tamio Nakajima, Costin Oncescu, Adrian Budau, Bogdan Sitaru si Andrei Constantinescu pentru multitudinea de solutii si contraexemple oferite.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Zoltan Szabo, Inspectoratul Şcolar Judeţean Mureş
- Drd. Andrei Costin Constantinescu, ETH Zurich
- Prof. Mihai Bunget, Colegiul Naţional "Tudor Vladimirescu", Târgu Jiu
- Stud. Bogdan Sitaru, University of Oxford
- Stud. Vlad-Adrian Ulmeanu, Universitatea Politehnica Bucureşti
- Stud. Andrei Cotor, Universitatea "Babeş Bolyai", Cluj-Napoca
- Stud. Tudor-Ştefan Măgirescu, DELFT University of Technology, Delft
- Stud. Alexandra-Mihaela Nicola, DELFT University of Technology, Delft
- Stud. Costin-Andrei Oncescu, Harvard University, Cambridge
- Stud. Matei Tinca, Vrije Universiteit Amsterdam
- Asist. Eugenie Daniel Posdărăscu, Youni
- Adrian Budău, Infoarena