Editorial RoAlgo PreOJI 2026
Clasa a-VII-a

Februarie 2026

Cuprins

1 Oximoron

Autor: Flaviu Dimitriu

1.1 Discutare a ideii generale de rezolvare

În primul rând, observația ce este esențială pentru rezolvarea problemei este oferită de restricția "Nu va apărea aceeași valoare *div* în descrierea a oricăror două operații diferite.". Astfel, este un lucru bine cunoscut și care este prezentat și în cadrul altor algoritmi, precum Ciurul lui Eratostene, că aceasta varianta a sumei armonice poate fi aproximată la $X \cdot log(X)$ și arată astfel:

$$\frac{X}{1} + \frac{X}{2} + \frac{X}{3} + \cdots + \frac{X}{X}$$

Astfel, dat fiind că valorile *div* sunt distincte, ne permitem să parcurgem, pentru fiecare operație, toți multiplii valorii *div* din cadrul operației. Dacă valorile *div* nu ar fi fost distincte, atunci acestea ar fi putut fi, spre exemplu, toate egale cu 1, caz în care am fi putut face, în cel mai rău caz, $N \cdot M \cdot Q$ operații de adunare în matricea B in total față de cele $N \cdot M \cdot log(N \cdot M)$ pe care le vom face, în cel mai rău caz, datorită sumei armonice.

Acum, noi vrem ca în cadrul parcurgerii fiecărui multiplu, pentru un multiplu M, să nu trebuiască să căutam de fiecare dată prin matrice linia și

coloana la care se afla valoarea M, ci vrem să ne stocăm aceste lucruri. Astfel, ne vom menține, înafară de cele două matrici, doi vectori: lin și col, de mărime $N \cdot M$, cu semnificația lin_i = linia pe care se află valoarea i și col_i = coloana pe care se află valoarea i, pentru orice $1 \leq i \leq N \cdot M$. De asemenea, acest lucru ne este permis deoarece ne este garantat că fiecare valoare de la 1 la $N \cdot M$ va apărea exact odată în matrice, deci nu vom avea de a face cu repetiții de valori. Astfel, acum putem adăuga în matricea B cu ușurință și fără a afecta complexitatea soluției valoarea 1 la pozițiile unde se regăsesc toți multiplii valorii div din cadrul fiecărei operații.

De asemenea, observăm că nu ne sunt oferite limite clare pentru N și M, ci doar pentru produsul $N \cdot M$, care este ≤ 200 000. Astfel, nu ne putem declara o matrice clasică, deoarece nu știm concret valorile N și M maxime și pentru că, dacă am declara-o ca având numărul de linii egal cu 200 000 și numărul de coloane egal cu 200 000, am avea parte de o depășire clară a limitei de memorie (și chiar o posibilă eroare de compilare din cauza dimensiunii uriașe). Astfel, există două abordări ce au fost cele mai populare pentru stocarea în memoriei a celor două matrici, A și B. Una dintre acestea este prin a folosi vector<vector> din STL sau prin a declara în interiorul main-ului matricea, lucru ce dădea, în probă, 100 de puncte, însă este nerecomandat din cauza declarării vectorului pe stivă, respectiv pe heap în cazul în care este vector STL, și una dintre acestea fiind prin a folosi doar doi vectori de mărime 200 000 pentru cele două matrici. Datorită simplității și ideii frumoase, precum și a diverselor cazuri de utilizare ale acesteia, o vom discuta pe a doua dintre acestea.

Ne vom ține elementele din matricea A într-un vector, astfel încât al i-lea element din vector reprezintă al i-lea element obținut prin parcurgerea

matricei de la stânga spre dreapta, de la primie linie spre ultima. Cu alte cuvinte, matricea:

17	3	5	4	2
15	19	18	1	11
9	6	13	8	10
20	16	12	14	7

Se transformă în vectorul:

$$\mathbf{v} = \left[17, 3, 5, 4, 2, 15, 19, 18, 1, 11, 9, 6, 13, 8, 10, 20, 16, 12, 14, 7\right]$$

Astfel, mai rămâne de văzut cum obținem linia, respectiv coloana unui element dacă stocăm pozițiile prin metoda menționată. Astfel, nu vom mai avea nevoie de 2 vectori, *lin* și *col*, ci doar de un vector, *poz*, ce reprezinta poziția valorii în vectorul în care se transformă matricea. Elementul de pe linia i și coloana j va ocupa, în vector, poziția $(i-1) \cdot M + j$, deoarece avem $(i-1)$ linii complete a câte M elemente fiecare, și înca j elemente pe linia curentă.

De asemenea, vrem să verificăm dacă o poziție *poz* în vectorul dat se află într-o submatrice dată, adică avem nevoie și de procesul invers de transformare: dată fiind o poziție din vector, să determinăm care este linia pe care se afla aceasta în matricea originală, precum și coloana pe care se afla aceasta inițial. Astfel, avem următoarele funcții, ale căror logică este în mod direct inversă celei menționate anterior, la procesul de transformare din poziție în matrice în poziție în vector.

```
1 int lin(int x)
2 {
```

```
    return (x - 1) / m + 1;
}
int col(int x)
{
    return (x % m == 0 ? m : (x % m));
}
```

Astfel, funcția *lin* returnează linia pe care se afla în matricea inițială valoarea de pe poziția x în vector, iar funcția *col* returnează coloana pe care se afla în matricea inițială valoare de pe poziția x în vector.

De asemenea, logica pentru actualizarea matricei B este relativ simplă: pentru fiecare multiplu care se situează în submatricea dată, adăugăm unu la poziția acestuia în vectorul obținut din matrice. Trebuie să avem grijă ca, atunci când îl afișăm pe B, să ne ocupăm de separarea pe linii și coloane, dar acest lucru este relativ simplu.

Astfel, se obține soluția de 100 de puncte, regăsită mai jos, la secțiunea „Cod sursă”. Pentru subtaskul de 20 de puncte se putea face un brut ce trece prin toate elementele submatricii și verifică divizibilitatea, iar subtaskul de 50 de puncte este dat deoarece permite menținerea matricei ca având dimensiunile laturilor maxim 400, nefiind nevoie de logica cu menținerea matricei drept un vector pentru acest subtask.

1.1.1 Cod sursă

Soluție de 100 de puncte ce reflectă raționamentul prezentat anterior.

2 Liga 1

Autor: Mihai Georgescu

2.1 Soluție 1 - 10 pct, $O(2^n * n * C)$

- Folosind metoda backtracking se generează toate submulțimile și se alege răspunsul optim.

2.2 Soluție 2 - 20 pct, $O(1e9 * n)$

- Fie funcția $f(x)$ = numărul maxim de intervale disjuncte cu $z_i \leq x$ care pot fi alese.
- Se iterează de la 1 la 1e9 si se va afișa x minim astfel incât $X \leq f(x)$ (nr de meciuri cerut). Pentru a calcula pe $f(x)$ se va folosi o abordare greedy clasică. (Problema Spectacolelor)

2.3 Soluție 3 - 100 pct, $O(n * log(n))$

- Se poate observa că funcția f este crescătoare, ceea ce ne permite să folosim căutare binară. Astfel, vom căuta cel mai mic x astfel incât $X \leq f(x)$. (Căutare binară de tip lowerbound)

- Notă: în funcție de calitatea greedy-ului folosit s-au putut obține diverse punctaje parțiale în intervalul 0-90 pct.

2.4 Cod sursă

Soluție de 100

3 Multumiri

Acest concurs nu ar fi putut avea loc fără următoarele persoane:

- Flaviu Dumitru si Mihai Georgescu, autorii problemelor și laureați la concursurile de informatică și membri activi ai comunității RoAlgo;
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova;
- Ștefan Alecu, creatorul acestui șablon LaTeXpe care îl folosim;
- Susan Margine și Robert Moldovan, coordontatori PreOJI;
- Comunității RoAlgo, pentru participarea la acest concurs.