Editorial RoAlgo PreOJI 2024

4-11 Martie 2024

Cuprins

1 Multumiri

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

- Luca Valentin Mureșan, Alexandru Gheorghieș și Matei Neacșu, 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;
- Rareș Buzdugan, Andrei Chertes, Tudor Iacob, testerii concursului, care au dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei;
- Alexandru Gheorghieș, coordonatorul comisiei claselor 11-12;
- Comunității de informatică din România, pentru participarea la acest concurs, precum și tuturor celor care ne-au ajutat să promovăm concursul.

2 Problema Dominant

Autor: Luca Valentin Mureșan

Subtask 1 (Dificultate: ★☆☆☆☆)

Un șir este xy-dominant dacă satisface simultan următoarele condiții:

1. Există cel puțin x elemente distincte în șir.
2. Suma celor mai mari x frecvențe din șir este mai mare sau egală cu k.

Astfel, o verificare brută a fiecărei subsecvențe are complexitatea totală: $O(n^2 \cdot nlog(n)) = O(n^3 \cdot log(n))$.

Soluție de 25 de puncte

Subtask 2 (Dificultate: ★★☆☆☆)

Se observă că dacă subsecvența (i, j) este *xy-dominantă* atunci și subsecvența $(i, j + 1)$ e *xy-dominantă*.

Acest lucru se întâmplă deoarece adăugarea unui element nou poate doar să crească suma celor mai mari x frecvențe din subsecvență.

Prin urmare, algoritmul de la subtask-ul anterior poate fi optimizat folosind tehnica two-pointers, reducând complexitatea la
$O(n \cdot nlog(n)) = O(n^2 \cdot log(n))$.

Soluție de 47 de puncte

Subtask 3 (Dificultate: ★☆☆☆☆)

Deoarece $y = 1$, acest subtask este echivalent cu găsirea numărului de subsecvențe care conțin cel puțin x elemente distincte.

O soluție optimă pentru acest subtask folosește tehnica two-pointers împreună cu un vector de frecvență și un contor care reține numărul de valori distincte din subsecvența curentă.

Complexitate: $O(n)$

Soluție de 16 puncte

Subtask 4 (Dificultate: ★★☆☆☆)

Deoarece $x = 1$, acest subtask este echivalent cu găsirea numărului de subsecvențe care au frecvența maximă mai mare sau egală cu y.

O soluție optimă pentru acest subtask folosește tehnica two-pointers împreună cu un `std::multiset` pentru a menține frecvența maximă din subsecvența curentă.

Complexitate: $O(n \cdot log(n))$

Soluție de 66 de puncte

Subtask 5 (Dificultate: ★★★☆☆)

Generalizând ideea de la subtask-urile precedente, ne trebuie o structură de date care suportă operațiile:

1. Crește / scade o valoare cu 1.
2. Verifică dacă există cel puțin x elemente nenule.
3. Află suma celor mai mari x valori.

Cum x este constant, putem folosi tot un `std::multiset`, împreună cu un pointer la a x-a cea mai mare valoare din multiset.

Folosind tehnica two-pointers, împreună cu structura de date menționată anterior aplicată pe frecvențele numerelor din subsecvența curentă, obținem un algoritm cu complexitatea $O(n \cdot log(n))$.

Soluție de 100 de puncte cu multiset

O altă structură de date care suportă aceste operații este un arbore indexat binar, pe care putem căuta binar a x-a cea mai mare valoare. În funcție de cum este implementată căutarea binară, acest algoritm poate avea complexitatea $O(n \cdot log(n))$ sau $O(n \cdot log^2(n))$.

Soluție în $O(n \cdot log^2(n))$ cu aib

Provocare: Rezolvați problema în $O(n)$.

3 Problema Morcovi

Autor: Alexandru Gheorghieș

Subtask 1 (Dificultate: ★☆☆☆☆)

Soluția pentru acest subtask verifică brut pentru fiecare pereche (a, b) cu $1 \leq a < b \leq n$ dacă satisface $gcd(a, b) + k = lcm(a, b)$.

Complexitate: $O(n^2 \cdot log(n))$ per testcase.

Soluție de 25 de puncte

Subtask 2 (Dificultate: ★★☆☆☆)

Deoarece $n, k \leq 3000$, se poate precalcula răspunsul pentru toate valorile posibile ale lui n și ale lui k.

Fie $ans[n][k]$ răspunsul pentru un anumit n și k. Pentru un n anume se pot calcula simultan toate $ans[n][k]$ în felul următor:

```
for(int i=1;i<=k;i++)
{
    ans[n][k]=ans[n-1][k];
}
for(int i=1;i<n;i++) /// luam in calcul toate perechile (i,n)
{
    int k2=lcm(n,i)-gcd(n,i);
    if(k2<=k)
        ans[n][k2]++;
}
```

Complexitate: $O(nk \cdot log(n))$ precalulare + $O(1)$ per testcase.

Soluție de 46 de puncte

Subtask 3 (Dificultate: ★★☆☆☆)

Se poate observa că putem fixa valoarea lui a și valoarea lui $gcd(a, b) = d$ în $O(n \cdot log(n))$ moduri.

Pentru fiecare pereche (d, a), avem $d + k = \frac{ab}{d} \Leftrightarrow b = \frac{d^2+k \cdot d}{a}$. Dacă b respectă simultan următoarele condiții:

1. b este un număr întreg
2. $1 \leq b \leq n$
3. $gcd(a, b) = d$

atunci (a, b) este o soluție validă. Deoarece fiecare pereche (a, b) este numărată de două ori, răspunsul va trebui împărțit la 2.

Complexitate: $O(n \cdot log(n))$ per testcase.

Soluție de 64 de puncte

Subtask 4 (Dificultate: ★★★☆☆)

Acest subtask există pentru a puncta soluțiile aproximativ corecte cu complexitatea ideală sau aproape ideală.

Complexitate: $O(k^{\frac{5}{6}})$

Soluție de 12 puncte

Subtask 5 (Dificultate: ★★★☆☆)

Notăm $d = gcd(a, b)$, $x = \frac{a}{d}$ și $y = \frac{b}{d}$:

$$d + k = \frac{ab}{d} \Leftrightarrow$$
$$d + k = x \cdot y \cdot d \Leftrightarrow$$
$$1 + \frac{k}{d} = x \cdot y$$

Prin urmare, $d = gcd(a, b)$ trebuie obligatoriu să fie un divizor al lui k.

Pentru fiecare divizor d al lui k va trebui să verificăm pentru toate perechile (x, y) care satisfac $x \cdot y = 1 + \frac{k}{d}$ dacă:

1. $1 \leq x \cdot d, y \cdot d \leq n$

2. $gcd(x \cdot d, y \cdot d) = d \Leftrightarrow gcd(x, y) = 1$

Dacă aceste condiții suplimentare sunt îndeplinite, atunci perechea $(x \cdot d, y \cdot d)$ este o soluție validă. Deoarece fiecare pereche este numărată de două ori, răspunsul final trebuie împărțit la 2.

Estimând numărul maxim de divizori ai unui număr n cu $n^{\frac{1}{3}}$, obținem următoarea complexitate:

$$O(\sqrt{k} + k^{\frac{1}{3}} \cdot (\sqrt{k} + k^{\frac{1}{3}} \cdot log(k)) = O(k^{\frac{5}{6}} + k^{\frac{4}{6}} \cdot \log(k)) = O(k^{\frac{5}{6}})$$

Soluție de 100 de puncte fără ciur

Folosind ciurul lui eratostene, determinarea divizorilor lui $1 + \frac{k}{d}$ poate fi optimizată de la $O(\sqrt{k})$ la $O(\frac{\sqrt{k}}{\log(k)})$, aducând la complexitatea finală per testcase la:

$$O\left(\frac{k^{\frac{5}{6}}}{log(k)}\right)$$

.

Notă: Această optimizare nu este necesară pentru a lua 100 de puncte.

Soluție de 100 de puncte cu ciur

4 Problema Warb

AUTOR: MATEI NEACȘU

Subtask-urile 1,2,4 (Dificultate: ★☆☆☆☆)

La aceste subtask-uri putem colora toate cele n noduri cu aceeași culoare. Prin urmare, răspunsul este egal cu: $max(w_1, w_2, \ldots, w_m) \cdot n$.

Complexitate: $O(m)$

Soluție de 18 puncte

Subtask 3 (Dificultate: ★★☆☆☆)

Putem încerca toate colorările posibile folosind metoda backtracking. Acest algoritm are complexitatea $O(m^n \cdot n)$.

Soluție de 30 de puncte

Subtask 5 (Dificultate: ★★★☆☆)

Vom parcurge arborele/lanțul de la un capăt la altul. Fie $dp[u][c][0/1]$ suma maximă a ponderilor culorilor până la nodul u inclusiv, astfel încât:

1. Nodul u să aibă culoarea c.
2. Următorul nod după u să fie obligat (sau nu) să aibă culoarea $c - 1$.

Pentru fiecare stare, vom încerca să colorăm următorul nod cu fiecare culoare de la 1 la m și verificăm dacă acest lucru este posibil.

Complexitate: $O(n \cdot m^2)$

Subtask 6 (Dificultate: ★★★☆☆)

Soluția de la subtask-ul anterior poate fi optimizată dacă calculăm $dp[u][c][0/1]$ în funcție de stările posibile ale nodului precedent.

Acest lucru se poate face în $O(1)$ folosindu-ne de maximele pe prefix, respectiv pe sufix, ale șirului $dp[prv][1][0], dp[prv][2][0], \ldots, dp[prv][m][0]$.

Complexitate: $O(n \cdot m)$

Subtask 7 (Dificultate: ★★★☆☆)

Vom identifica nodul cu gradul $n-1$, fie acesta u. Pentru fiecare culoare posibilă a nodului u de la 2 la m, vom folosi următorul algoritm:

Pentru fiecare vecin v al lui u, vom calcula două valori:

- $a[v]$ - Ponderea maximă posibilă a culorii lui v (adică $w[c[v]]$) care totuși satisface $c[v]+1 \neq c[u]$.
- $b[v] = w[c[u]-1]$

Pentru a avea exact $req[u]$ vecini cu culoarea $c[u]-1$, va trebui să alegem exact $req[u]$ noduri care vor avea ”costul” $b[v]$, iar restul vor avea ”costul” $a[v]$.

Pentru a maximiza suma costurilor, vom sorta vectorul de perechi $(a[i], b[i])$ descrescător după $b[i]-a[i]$. Pentru primele $req[u]$ valori din șir vom adăuga $b[i]$ la costul total, iar pentru celelalte vom adăuga $a[i]$.

În cazul în care culoarea lui u este 1, celelalte noduri pot fi colorate independent unul de celelalte.

Complexitate: $O(n \cdot m \cdot log(n))$

Soluție de 14 puncte

Subtask 8 (Dificultate: ★★★★☆)

Similar cu subtask-urile 5 și 6, fie $dp[u][c][0/1]$ valoarea maximă posibilă a subarborelui lui u, astfel încât:

- Culoarea lui u să fie egală cu c.
- Părintele lui u să fie obligat (sau nu) să aibă culoarea $c - 1$.

Astfel, noi știm că pentru o anumită stare, ori $req[u]$, ori $req[u] - 1$ copii ai lui u trebuie să aibă culoarea $c - 1$. Această subproblemă este foarte similară cu subtask-ul 7.

În loc să sortăm după $b[i] - a[i]$ (ca la subtask-ul 7), soluția pentru acest subtask are nevoie de un rucsac suplimentar pentru a calcula $dp[u][c][0/1]$ în funcție de dp-urile copiilor lui u.

Complexitate: $O(\sum_{i=1}^{n}(grad(i) \cdot req[i]) \cdot m) = O(n \cdot max(req[i]) \cdot m)$

Soluție de 74 de puncte

Subtask 9 (Dificultate: ★★★★★)

Acest subtask există pentru a puncta sursele care nu tratează corect cazul $req_i = 0$.

Complexitate: $O(n \cdot m \cdot log(n))$

Subtask 10 (Dificultate: ★★★★★)

Combinând dp-ul de la subtask-ul 8 și ideea de la subtask-ul 7, se obține complexitatea $O(n \cdot m \cdot log(n))$.

O sursă de 100 de puncte; O altă sursă de 100 de puncte