Editorial RoAlgo Back to School 2024

8 Septembrie 2024

Cuprins

1 Multumiri

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

- Raul Ardelean, Lucian Badea, Rares Buzdugan, Alex-Robert David, Ștefan Dăscălescu, Tudor Iacob, Andrei Ivan, Bogdan-Alexandru Luca, Iulia Ela Morariu, Radu-Teodor Prosie, Mircea-Stefan Tomita, Stefan Vilcescu, autorii și propunătorii problemelor, 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;
- Pe lângă autorii menționați mai sus, Fiodar Ziuzin, testerul concursului, care a dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei;
- Ștefan Dăscălescu, coordonatorul rundei;
- Comunității RoAlgo, pentru participarea la acest concurs.
- Algolymp, pentru oferirea premiilor generoase pe care le-am putut prezenta concurentilor care au castigat primele locuri.

2 Problema Perlă

Autor: Prosie Radu-Teodor

2.1 Soluție subtask 1

Pentru ca T să apară cel puțin o dată in S, T trebuie să fie o subsecvență de-a lui S. Astfel, putem fixa o subsecvență și îi calculăm scorul. Complexitate $O(|S|^3)$ sau $O(|S|^4)$, în funcție de cum se calculează numarul de apariții.

2.2 Soluție completă

Observăm că un șir de lungime k apare de maxim $|S| - k + 1$ ori in S. Astfel, scorul maxim posibil este $|S| + 1$, și îl putem obține mereu alegând $T = S$. Complexitate $O(|S|)$.

3 Problema Antidivizibil

Autor: Stefan Dascalescu

Soluție

Pentru a rezolva această problemă, trebuie să ne gândim care ar fi un fel de număr care s-ar potrivi cu cerința problemei date.
Deoarece nu putem avea un număr care să înceapă cu 1, deoarece 1 este un divizor al oricărui număr mai mare decât el, numărul nostru trebuie să înceapă cu 2.
Cel mai mic număr care începe cu 2 este de forma 2000 ... 0, dar 2 este un divizor al acestui număr mai mare. Pentru a evita această problemă, vom putea afișa 2000 ... 01, deoarece numărul final este impar, iar toate prefixele sale sunt pare.
Soluția de 100

4 Problema costMin

Autor: Ștefan Vîlcescu

4.1 Observație generală

Putem observa destul de ușor că răspunsul nostru va fi $max(n, m)$ - numărul de perechi care au $a_i = b_i$, unde $i \leq i \leq min(n, m)$. Deci, noi trebuie să inteschimbăm elemente din a și din b astfel încât avem un număr maxim de poziții i care au $a_i = b_i$. Pentru restul acestui editorial, vom defini ”rezultat” ca numărul maxim de perechi $a_i = b_i$ pe care trebuie să le avem.

4.2 Soluție de 40 de puncte

Pentru această soluție, algoritmul este simplu:
Pentru fiecare $1 \leq i \leq n$, găsim un indice j, $1 \leq j \leq m$ astfel încat $a_i = b_j$.
Dacă am găsit un astfel de indice, vom aduna la rezultat 1 și vom nota că indicele j are pereche, altfel nu facem nimic
Complexitate timp: $O(n \cdot m)$
Complexitate memorie: $O(n + m)$
Soluție de 40 de puncte

4.3 Soluție de 100 de puncte

Există multe abordări la această problema, voi prezenta doar două dintre ele.

4.3.1 Prima soluție de 100 de puncte

Deoarece trebuie să avem $a_i = b_i$, putem pune într-un vector de frecvență frv șirul a, iar pentru fiecare element din b verificăm dacă $frv_{b_i} > 0$, dacă este adunăm la rezultat 1 si scădem din frecvență, altfel nu facem nimic
Complexitate timp: $O(n + m + AMAX)$, unde $AMAX = 10^6$
Complexitate memorie $O(AMAX)$
O implementare de 100 pe baza acestei soluții

4.3.2 Cea de-a doua soluție de 100 de puncte

Putem sorta vectorii și vom face algoritmul clasic de interclasare a doi vectori. Dacă în timpul interclasării dăm de două elemente a_i și b_j care sunt egale, adunăm la rezultat 1, continuând algoritmul.
Complexitate timp: $O(n \cdot log(n) + m \cdot log(m))$
Complexitate memorie: $O(n + m)$
O implementare de 100 pe baza acestei soluții

5 Crocks

Autor: Iacob Tudor

5.1 Soluție subtask 1

Pentru acest subtask putem să fixăm st și dr și să iterăm brute-force pentru a calcula suma pentru toate perechile in $O(n^3)$.

5.2 Soluție subtask 2

Pentru acest subtask ajunge să fixăm st și dr și cu o sumă parțială să calculăm în $O(1)$ suma pe interval, în total algoritmul este $O(n^2)$.

5.3 Soluție completă

Pentru solutia de 100 de puncte trebuie să observăm că în realitate dacă am desface parantezele problema se rezumă la a afla $\sum_{1 \le x < y < z \le N} a_x \cdot a_y \cdot a_z$. Dacă rescriem un pic suma ajungem la
$\sum_{1 < y < N} a_y \cdot (a_1 + a_2 + ... + a_{y-1}) \cdot (a_{y+1} + a_{y+2} + ... + a_N)$. Pentru un y fixat, putem calcula suma în $O(1)$ folosind sume parțiale.
Soluție de 100

6 Problema Crystallized

Autor: Prosie Radu-Teodor

Soluție subtask 1 respectiv 2

Pentru primul subtask este suficientă o soluție care parcurge toate submulțimile iar pentru al doilea subtask este necesară o arbordare prin programare dinamică.

Soluție completă

Fie S mulțimea puterilor lui 2 mai mici sau egale cu N. Acum, dacă luam orice submulțime a lui $\{1, 2, 3, ...N\} \setminus S$, vom avea exact un mod de a alege elemente din S astfel încât suma xor a mulțimii rezultate să fie X. Astfel, rezultatul, indiferent de X, este $2^{N-|S|}$. Ramâne doar cazul particular în care nu putem face suma X și rezultatul este 0. De asemenea, deși nu e de preferat, formula putea fi dedusă ajungând la concluzia că răspunsul nu depinde de X analizând rezultatele obținute prin programare dinamică. Complexitate : $O(log_2(N))$

7 Problema prod3

Autor: Ștefan Vîlcescu

7.1 Soluția de 13 puncte

Se poate implementa un brut, unde pentru fiecare query de tip 1 vom înlocui a_poz cu x, iar pentru fiecare query de tipul 2, vom merge peste fiecare triplet (i, j, k), unde $l \leq i, j, k \leq r$, și vom afișa un triplet astfel încât $a_i = a_j \cdot a_k$, altfel vom afișa 000
Complexitate timp: $O(q \cdot n^3)$
Complexitate memorie: $O(n)$
Soluție de 13 de puncte

7.2 Soluția de 31 de puncte

Pentru fiecare query de tip 2, vom sorta subsecvența $[l, r]$, și vom aplica tehnica two pointers astfel:
Ne vom fixa o poziție i Pentru fiecare poziție i fixată, vom avea doi indici j, k cu proprietatea că $a_i \cdot a_j \leq a_k$. Când trecem de la poziția j la $j + 1$, atunci vom aduna la k până când $a_i \cdot a_j \leq a_k$
Complexitate timp: $O(q \cdot n^2)$
Complexitate memorie: $O(n)$

Soluție de 31 de puncte

7.3 Soluția de 53 de puncte

Observație. Nu ne trebuie toate numerele din intervalul $[l, r]$, ci doar $AMAX \cdot 3$, unde $AMAX = 50$
Această observație se poate face foarte ușor, deoarece în ecuația noastră in queryurile de tipul 2 avem doar 3 variabile a_i, a_j și a_k, deci nu o să ne trebuiască niciodata mai mult de 3 variabile. Deoarece există maxim $AMAX$ numere distince, noi trebuie să ținem cont doar de $AMAX \cdot 3$.
Știind asta, putem să calculăm 3 poziții ale fiecărui element x,
$1 \leq x \leq AMAX$, în intervalul $[l, r]$, apoi vom merge prin fiecare triplet (i, j, k), unde $1 \leq i, j, k \leq AMAX$ și verificăm dacă există poziția lui i, j și k în intervalul $[l, r]$ și că $i = j \cdot k$
Complexitate timp: $O(q \cdot n)$
Complexitate memorie: $O(n)$
Soluție de 53 de puncte

7.4 Soluția de 100 de puncte

Observație. Deoarece se admite orice soluție, pentru fiecare x,
$1 \leq x \leq AMAX$, trebuie doar să aflăm cele mai mici 3 poziții ale lui x din intervalul $[l, r]$
Această observație ne duce la gândul să ținem pentru fiecare x,
$1 \leq x \leq AMAX$, pozițiile acestuia în ordine crescătoare
Deci, problema se reduce la a ține crescător pozițiile pentru $x - uri$, chiar și cu updateuri
Putem ține $v_{(i,j)}$=de câte ori apare elementul j în primele i poziții

Când avem queryuri de tipul 1, vom da update la $v_{a_{poz}}$ și la v_x, facând si $a_{poz} = x$

Pentru queryurile de tipul 2, pentru fiecare element x, vom afla de câte ori apare pe primele $l - 1$ poziții, și vom căuta binar pozițiile următoare, deoarece dacă x apare la poziția p, atunci $v_{(p,x)}=v_{(p-1,x)} + 1$, deci trebuie să căutăm binar $p - urile$ minime astfel încât $v_{(p,x)} = v_{(l-1,x)} + k$, unde $1 \leq k \leq 3$, iar noi trebuie să facem asta eficient.

Acest lucru se poate face cu aib. Totuși, există soluții cu set și aint pe care le-au descoperit testării, mergând pe aceeași idee cu căutare binară

Complexitate timp: $O(n \cdot log(n) + q \cdot AMAX \cdot log(n))$

Complexitate memorie: $O(n \cdot AMAX)$ sau $O(n)$, depinde de soluție

Soluție de 100 de puncte cu aib

Soluție de 100 de puncte cu set

Soluție de 100 de puncte cu aint

8 Șanțuri și poduri

Autor: Alex-Robert David

8.1 Observație generală

Din cerința problemei, reiese că drumurile solicitate sunt defapt punți în graful dat, iar aceasta va fi terminologia pentru restul editorialului.

8.2 Soluție subtask 1

Pentru primul subtask putem afla punțile din graf și pentru fiecare pereche de noduri, să adunăm punțile întâlnite în parcurgerea DFS. Complexitatea este $O((N + M) + N^3)$.

8.3 Soluție subtasks 2 & 3

Cum se garantează faptul că graful dat este un arbore, observația importantă este că toate muchiile sunt punți, ceea ce ne permite să găsim o soluție mai eficientă.

8.3.1 Pentru subtask 2

Putem optimiza soluția de la subtask-ul 1 cu un algoritm de aflare a LCA-ului dintre 2 noduri. O posibilă abordare presupune construierea unui șir d, unde $d[u]$ reprezintă suma muchiilor de la rădăcina arborelui până la nodul u. Fiecare pereche de noduri $(u,\ v)$ contribuie la sumă cu $d[u] + d[v] - d[lca(u,\ v)]$, unde cu $lca(u,\ v)$ s-a notat LCA-ul celor două noduri.
O altă abordare ar fi să numărăm de câte ori este folosită fiecare muchie. Putem aplica un algoritm asemănător cu „șmenul lui Mars" adaptat pentru arbore astfel: pentru fiecare pereche $(u,\ v)$ vom incrementa în șirul $diff$ valorile $diff[u]$ și $diff[v]$ și vom scădea 2 din $diff[lca(u,\ v)]$. La final, vom calcula șirul $count$ în care $count[u]$ este suma $diff[v]$ pentru nodurile v din subarborele lui u. Răspunsul va fi $\sum_{i=1}^{N} count[i] \cdot w[i]$, unde cu $w[i]$ s-a notat costul muchiei de la părintele lui i la i.
Complexitatea devine $O(N \cdot \log N + N^2 \cdot \log N)$ sau $O(N \cdot \log N + N^2)$, în funcție de implementarea algoritmului pentru aflarea LCA-ului a două noduri.
Soluție ce ilustrează a doua abordare

8.3.2 Pentru subtask 3

Putem calcula răspunsul în $O(N)$ observând că muchia incidentă de la părinte la un nod u poate fi folosită în toate lanțurile cu capătul într-un nod din subarborele lui u și un nod din afara subarborelui. Muchia contribuie la sumă cu $(s[u] \cdot (N - s[u])) \cdot w[u]$, unde cu $s[u]$ s-a notat numărul de noduri din subarborele lui u, iar cu $w[u]$ costul muchiei de la părintele lui u la u.

8.4 Soluția completă

Pentru a putea rezolva complet problema, trebuie să găsim o metodă de a transforma graful dat într-un arbore, iar apoi să aplicăm algoritmul pentru subtask-ul 3. Suma cerută face abstracție de restul muchiilor din arbore, deci putem construi un graf compresat pe baza celui inițial, în care singurele muchii sunt punțile, iar nodurile constituie componentele conexe rămase prin eliminarea punților din graful inițial.
Graful compresat este conex, deoarece presupunând prin absurd că ar fi cel puțin două componente conexe rezultate în urma compresiei, ar duce la existența a cel puțin două componente conexe în graful inițial - fiindcă punțile leagă doar componentele rezultate prin eliminarea lor; lucru care contrazice enunțul și datele problemei.
Graful compresat este defapt arbore, deoarce presupunând prin absurd că ar exista cel puțin un ciclu, ar rezulta că cel puțin una dintre muchii nu era punte în graful inițial, lucru care contrazice construcția grafului compresat.
Acest graf special poartă numele de arbore de compresie și puteți găsi mai multe informații și probleme la acest link.
Pentru a finaliza algoritmul, fiecărui nod u din arborele de compresie îi vom atribui prin $p[u]$ numărul de noduri din componenta conexă pe care o reprezintă, din graful inițial. Pentru un nod u și muchia incidentă de la părinte, acesta va contribui la răspuns cu $(s[u] \cdot (N - s[u])) \cdot w[u]$, unde $s[u]$ este suma din $p[v]$, pentru toate nodurile v din subarborele lui u.
Complexitatea finală este $O(N + M)$.
Soluție de 100

9 Remove Inversions

Autor: Morariu Iuli

9.1 Solutie subtask 1

Iterăm fiecare două poziții (i, j) și verificăm dacă numărul de inversiuni este exact K după ce le eliminăm. Dacă numărul inițial de inversiuni este K, adaugam 1 la răspuns. Complexitate: $O(N^4)$

9.2 Solutie subtask 2

Ținem o listă inversions[]. inversions[i] este numărul de inversiuni cauzate de numărul de la poziția i (numărul de poziții j < i cu A[i] > A[j] și numărul de poziții j > i cu A[i] > A[j]).
Numărul de inversiuni pe care trebuie să le eliminăm va fi numărul total de inversiuni minus K.
Pentru a calcula numărul de moduri de a elimina un element, vom număra numărul de valori din lista inversions[i] care sunt egale cu numărul de inversiuni necesare, deoarece eliminând elementul de pe poziția i, numărul de inversiuni scade cu inversions[i].
Pentru a calcula numărul de moduri de a elimina două elemente, vom itera fiecare pereche (i, j), verificând dacă eliminând ambele rezultă exact K

inversiuni. Complexitate: O(N log N + N^2)

9.3 Solutie subtask 3

Calculăm numărul de inversiuni inițial folosind un AIB.
Construim lista inversions[] (cu aceeași semnificație ca la subtaskul precedent) în complexitate O(N log N) (cu AIB).
Pentru a calcula numărul de moduri de a elimina un element, se procedează la fel ca la subtaskul 2.
Pentru a calcula numărul de moduri de a elimina două elemente, vom itera fiecare poziție i și valoarea celui de-al doilea element. Să numim această valoare j. Dacă j <= A[i], la răspuns se adaugă numărul de poziții înainte de i la care se află valoarea j, și inversions[poz] = numărul necesar - K - inversions[i].
Dacă j > A[i], la răspuns se adaugă numărul de poziții înainte de i la care se află valoarea j, și inversions[poz] = numărul necesar - K - inversions[i] + 1.
Se menține numărul de poziții cu un vector de frecvență bidimensional.
Complexitate: O(N log N + N * VMAX)

9.4 Solutie subtask 4

Pentru a rezolva acest subtask, se procedează ca la subtaskul precedent, doar că numărul de poziții se menține cu un ordered set sau un aint dinamic, în loc de vector de frecvență. Complexitate: O(N log N)

10 Crucks

Autor: Iacob Tudor

10.1 Soluție subtask 1

Pentru acest subtask este îndeajuns să se genereze toate parcurgerile și să se verifice toate într-o manieră brute-force.

10.2 Soluție subtask 3

Fie $dp[i][j]$ numărul de parcurgeri dacă considerăm doar subarborele lui i care dau numar par de inversiuni când $j = 0$ și număr impar de inversiuni când $j = 1$. Având în vedere că fiecare nod are cel mult 2 fii avem două cazuri: Fie prima oară mergem pe fiul stâng, fie prima oară mergem pe fiul drept. Dacă alegem prima oară să mergem pe fiul stâng știm cu siguranță că toate valorile din fiul stang vor fi înaintea celor din fiul drept, astfel se vor crea obligatoriu niște inversiuni.
În total, numarul de inversiuni dintr-o parcuregere va fi suma dintre numărul de inversiuni care le obținem în fiul stâng cu numărul de inversiuni care se obțin în fiul drept și cu numarul de inversiuni obligatorii. Analog si pentru celalalt caz.

Spre exemplu, dacă suntem în nodul 1, alegem să mergem mai întâi pe fiul stang și numarul de inversiuni obligatorii este impar, vom avea $dp[1][0] = dp[Fiu_1][0] * dp[Fiu_2][1] + dp[Fiu_1][1] * dp[Fiu_2][0]$. Dacă în schimb, numarul de inversiuni obligatorii este par, pentru ca în nodul 1 la sfârșit să avem numar par de inversiuni, unul dintre fii trebuie sa aiba numar impar de inversiuni.
Atenție la implementare: când se calculează numarul de inversiuni obligatorii, trebuie ținut cont și de valoarea nodului i care este primul în secvență.

10.3 Soluție subtask 4

Se va proceda în mod similar cu subtaskul 3, singura diferență fiind că acum putem avea cel mult 6 cazuri în loc de 2.

10.4 Soluție completă

Pentru a obține punctajul maxim, putem trata cele 24 de cazuri pe care le avem ca urmare a numărului maxim de fii pe care îi putem avea.
Soluție de 100

10.5 Soluție alternativă

Pentru a evita toate cazurile care se pot obține folosind abordarea anterioară, o metodă alternativă de parcurgere a permutărilor fiilor este aceea de a folosi măști pe biți pentru fiecare din ele, pentru a fixa pe de o parte ordinea în care sunt parcurși fiii nodului curent, iar pe de altă parte, pentru a fixa paritatea

numărului de inversiuni pentru permutarea generată de subarborele oricărui fiu pe care îl parcurgem.
Pentru o configurație de fii, tot ce trebuie să facem este să aflăm numărul de inversiuni ai fiecărei perechi de subarbori, iar mai apoi putem adăuga la răspunsul parității rezultate numărul de soluții obținut. Trebuie avut grijă la implementare deoarece pe lângă inversiunile obținute la acest pas, trebuie să fim atenți și la inversiunile care au fost deja prezente în subarborii anteriori. Pentru mai multe detalii, recomandăm citirea codului de mai jos.
Soluție alternativă de 100