RoAlgo Contest 6 — Editorial

8 Octombrie 2023

1 Mulțumiri

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

- Rares-Andrei Buzdugan, Alexandru Lensu, Cosmin-Gabriel Pascale, Eduard-Lucian Pîrțac, laureați la concursurile de informatică și membri activi ai comunității RoAlgo.
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova
- Andrei Cherteș, care a dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei, precum și ajutor la crearea unor teste.
- Ștefan-Cosmin Dăscălescu, coordonatorul rundei și autorul soluțiilor video.
- Comunității RoAlgo, pentru participarea la acest concurs.

2 Problema Quickmath

Autor: Ștefan-Cosmin Dăscălescu

Pentru a rezolva această problemă, trebuie să observăm că numerele au un modul între n și $2 \cdot n$ ceea ce înseamnă că singurele triplete valide sunt $n + n - 2 \cdot n$, respectiv $2 \cdot n - n - n$, ambele cazuri putând fi verificate manual după aflarea frecvențelor fiecărei valori.

2.1 Cod sursă și soluție video

Cod sursă Soluție video

3 Problema Sliniare

Autor: Buzdugan Rareș-Andrei

3.1 Subtask 1 - 11 puncte

Vom fixa cele două capete ale fiecărei subsecvențe și vom parcurge fiecare element dintre cele două capete, verificând la final condiția.
Complexitatea temporală - $O(N^3)$
Sursa de la această soluție poate fi accesată aici.

3.2 Subtask 2 - 13 puncte

Vom fixa capătul din stânga al unei subsecvențe, după care ne vom plimba cu capătul din dreapta până ajungem la sfârșitul șirului, actualizând în același timp valoarea expresiei $aU - Z$. De fiecare dată, vom verifica dacă condiția este îndeplinită.
Complexitatea temporală - $O(N^2)$
Sursa de la această soluție poate fi accesată aici.

3.3 Subtask 3 - 17 puncte

Observăm că dacă b nu este divizibil cu a, atunci răspunsul este 0. Altfel, știm că avem nevoie de $\frac{b}{a}$ valori de 1. Astfel, răspunsul este numărul de subsecvențe de lungime $\frac{b}{a}$, care este egal cu $n - \frac{b}{a} + 1$. Dacă acest număr este negativ, vom afișa 0.
Sursa de la această soluție poate fi accesată aici.

3.4 Subtask 4 - 59 puncte

Încercăm să determinăm numărul de subsecvențe liniare cu capătul din dreaptă la poziția i. Vom nota cu U și Z numărul de valori de 1, respectiv de 0 din subsecvența $[1, i]$. Încercăm sa "tăiem" din această subsecvență o altă subsecvență $[1, j]$, $j < i$ astfel încât subsecvența $[j + 1, i]$ să fie liniară. Să notăm cu U' și Z' numărul de valori de 1, respectiv de 0 din subsecvența $[1, j]$. Astfel, vrem ca $a(U - U') - (Z - Z') = b$. Manevrăm această expresie si vom obține $aU - Z = b + aU' - Z'$. Din acest lucru, trebuie să ținem minte câte valori $b + aU' - Z'$ sunt egale cu $aU - Z$ până la i. Putem face acest lucru cu ajutorul unui **map**, să îl notăm cu mp. Vom crește numărul de subsecvențe cu $mp[aU - Z]$ după care vom incrementa $mp[b + aU - Z]$. De asemenea, avem grijă să luăm în considerare și subsecvențele liniare de forma $[1, i]$.
Complexitatea temporală - $O(NlogN)$
Sursa de la această soluție poate fi accesată aici.

3.5 Soluție video

Soluție video

4 Problema Transformare

Autor: Buzdugan Rareș-Andrei și Eduard-Lucian Pîrțac

4.1 Subtask 1 - 21 puncte

Vom parcurge fiecare insulă și vom determina câți vecini cu apă are pe direcțiile normale și direcția lui specială. Dacă îndeplinește condiția să devină insulă, vom crește un contor.

Complexitatea temporală - $O(P)$

4.2 Subtask 2 - 23 puncte

Vom ține minte în matricea $ani[i][j]$ anul în care insula de pe coordonatele (i, j) devine apă. Vom repeta același proces ca la Subtask 1 până când nu mai avem nicio insulă care poate să devină apă, doar că vom șterge la fiecare pas fiecare insulă care devine apă și nu o mai luăm în considerare mai departe.

Complexitatea temporală - $O(P^2 + Q)$

4.3 Subtask 3 - 56 puncte

Observăm că o insulă ce devine apă poate influența insulele din jurul lui, mai exact:

1. Toate insulele de pe direcțiile N, S, E, V

2. Toate insulele de pe direcțiile speciale NE, SV, SE, NE, cu condiția că din aceste insule putem ajunge în insula care a devenit apă pe direcția sa specială.

Dacă una din insulele influențate îndeplinește condiția de transformare, atunci este următorul candidat care poate influența alte insule. Astfel, putem face un algoritm asemănător algoritmului lui **Lee**, folosind o coadă în care bagăm fiecare insulă ce devine apă. Atunci când băgăm o insulă în coadă, actualizăm și anul în care acesta devine apă, care va fi egal cu $ani[i][j] + 1$, unde insula (i, j) este insula curentă scoasă din coadă.

Complexitatea temporală - $O(N * M + Q)$

Sursa de la această soluție poate fi accesată aici.

4.4 Soluție video

Soluție video

5 Problema Sugubatz

Autor: Cosmin-Gabriel Pascale

5.1 Subtask 1 - 22 de puncte

Pentru fiecare indice i de la 1 la N, putem calcula cel mai mare divizor comun dintre acesta și numerele din șirul A, de pe pozițiile din intervalele $[1, i-1]$ și $[i+1, N]$. După, pentru fiecare indice i de la 1 la N, putem porni cu un alt indice j de la i la N și putem calcula suma "sau" a secvenței $[i, j]$. Dacă această sumă devine mai mare sau egală cu X la un pas k, atunci numărul total de secvențe crește cu $N - k + 1$, iar i poate fi incrementat.
Complexitatea temporală: $O(N^2 \cdot log_2 N)$
Complexitatea spațială: $O(N)$
Sursa de la această soluție poate fi accesată aici.

5.2 Subtask 2 - 31 de puncte

Observația principală pe care o facem este aceea că mai multe numere au același divizor comun i dacă aceștia sunt multiplii ai lui i. Observând totodată că numerele date sunt relativ mici, putem folosi un vector de frecvență pentru a ști frecvența fiecărui număr din vectorul inițial. Astfel, pentru fiecare număr i de la 1 la VALMAX, unde VALMAX este valoarea maximă întâlnită în vectorul inițial, pornim cu un indice j de la i, care este mereu incrementat cu i, astfel putând să aflăm dacă numărul i are cel puțin doi multipli în vectorul A. Dacă are, vom parcurge din nou cu indicele j acești multipli, pentru a putea economisi memorie. În același timp, vom menține un alt vector de dimensiune VALMAX, în care vom marca răspunsul optim pentru fiecare număr de la 1 la VALMAX. La fiecare pas menționat anterior actualizăm în noul vecotr valorile tututror multiplilor lui i cu valoarea i dacă numărul acestora este mai mare sau egal cu 2. După, fiecare $a[i]$, cu i de la 1 la N, va lua valoarea răspunsului calculat pentru valoarea $a[i]$. Ne rămâne doar să calculăm numărul de secvențe cu proprietatea cerută, lucru ce poate fi efectuat cu un algoritm de tip Range Minimum Query, adaptat la *sau*.
Complexitatea temporală: $O(VALMAX + N * log2(N))$
Complexitatea spațială: $O(VALMAX + N * log2(N))$
Sursa de la această soluție poate fi accesată aici.

5.3 Subtask 3 - 47 de puncte

Formarea vectorului B se va realiza la fel ca la subtask-ul 2. Optimizarea pentru acest subtask va consta în faptul că în locul algoritmului de tip Range Minimum Query vom folosi un algoritm de tip two pointers. Acesta reduce drastic atât memoria utilizată, cât și timpul de rulare a progamului. Astfel, vom simula suma *sau* a unei secvențe, folosindu-ne de doi indici. Contorizăm într-un vector de frecvență de dimensiune $log2(VALMAX)$ numărul de biți care apar pe aceeași poziție. Dacă la un moment dat suma *sau* a secvenței considerate este mai mare sau egală cu X, atunci toate secvențele care o includ pe cea curentă și au același început ca și aceasta vor avea suma *sau* mai mare sau egală cu X. Astfel, vom începe să scoatem numere de la începutul secvenței considerate, pentru a ajunge la o sumă *sau* mai mică decât X. Pe parcurs vom actualiza continuu răspunsul final. Repetăm acest algoritm până când secvența considerată are suma *sau* mai mică decât X și nici nu mai există numere pe care le putem adăuga la această secvență.
Observație: atunci când adăugăm sau eliminăm un număr din secvența curentă, vom incrementa, respectiv decrementa frecvența poziției pe care se află biții numărului respectiv. Dacă această frecvență devine în acel moment 1, atunci la suma *sau* a secvenței vom adăuga puterea lui 2 la poziția bitului respectiv. Astfel, dacă aceasta devine 0, atunci din suma *sau* a secvenței vom scădea puterea lui 2 la poziția bitului respectiv.
Complexitatea temporală: $O(VALMAX * log2(VALMAX) + N)$
Complexitatea spațială: $O(VALMAX * log2(VALMAX) + N)$
Sursa de la această soluție poate fi accesată aici.

5.4 Soluție video

Soluție video

6 Problema Divizibilitate

Autor: Lensu Alexandru

În această problema putem distinge 3 cazuri:

1. Vrem toate numerele de k cifre care au restul 0.
2. Vrem toate numerele de k cifre care au restul 1.
3. Vrem toate numerele de k cifre care au restul 2.

Notăm:
$dp[0][k]$ = numerele de k cifre care au restul 0 la împărțirea cu 3.
$dp[1][k]$ = numerele de k cifre care au restul 1 la împărțirea cu 3.
$dp[2][k]$ = numerele de k cifre care au restul 2 la împărțirea cu 3.

Pentru cazul de bază în care $k = 1$, avem:
$dp[0][1] = 2$ (6 și 9), $dp[1][1] = 1$ (1), $dp[2][1] = 1$ (8).

Se poate observa o recurență între aceste relații.

Pentru $X = 0$:

$dp[0][k+1] = dp[0][k-1] \cdot 2 + dp[1][k-1] + dp[2][k-1]$;

(pentru un număr de $k-1$ cifre se poate păstra restul 0 adăugându-l la sfârșit pe 6 sau 9, obținând restul 0 din restul 1 adăugând la sfârșit 8 sau obținând restul 0 din restul 1, adăugând la sfârșit 6).

Analog pentru $X = 1$ și $X = 2$:

$dp[1][k+1] = dp[1][k-1] * 2 + dp[0][k-1] + dp[2][k-1]$;
$dp[2][k+1] = dp[2][k-1] * 2 + dp[0][k-1] + dp[1][k-1]$;

Folosirea acestor relații obține 52 de puncte.

Pentru 100 de puncte, putem profita de faptul că recurențele noastre sunt **liniare**. Astfel, putem găsi o matrice de recurență, pe care dacă o înmulțim cu matricea $(dp[0][i-1], dp[1][i-1], dp[2][i-1])$ să obținem matricea $(dp[0][i], dp[1][i], dp[2][i])$. Folosind proprietățile de înmulțire a matricilor, matricea de recurență este:

$$M = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Astfel, vom lua matricea inițială $(dp[0][1], dp[1][1], dp[2][1])$ și o vom înmulți cu M^{n-1}. Pentru a calcula M^{n-1}, vom folosi exponențiere rapidă.
Complexitatea temporală - $O(QlogN * 27)$

6.1 Cod sursă și soluție video

Cod sursă Soluție video

7 Problema Sap

Autor: Buzdugan Rareș-Andrei

7.1 Subtask 1 - 21 puncte

Vom parcurge fiecare nod din subarborele nodului X cu un DFS și vom determina dacă acestea formează o permutare. Pentru a verifica acest lucru, se poate folosi un vector de frecvență.

Complexitatea temporală - $O(N * Q)$

Sursa de la această soluție poate fi accesată aici.

7.2 Subtask 2 - 23 puncte

Observăm că dacă în subarborele unui nod i nu avem o permutare, atunci niciun ascendent al acestuia nu va avea în subarbore o permutare. Astfel, vom parcurge elementele de la dreapta la stânga și vom ține minte dacă sufixul curent este o permutare sau nu. Pentru a afla acest lucru, vom ține minte maximul curent și dacă elementele sunt distincte, folosind un vector de frecvență. Dacă maximul curent este egal cu lungimea sufixului și elementele sunt distincte, atunci pentru nodul curent avem o permutare.

Complexitate temporală - $O(N + Q)$

Sursa de la această soluție poate fi accesată aici.

7.3 Subtask 3 - 27 puncte

Vom băga în vectorul $DFSTraversal$ valorile nodurilor obținute atunci când facem parcurgerea DFS. Un Query pe nodul X va deveni un Query pe intervalul $[Position[X], Position[X] + SubTreeSize[X] - 1]$, unde $Position[X]$ este poziția nodului X în $DFSTraversal$, iar $SubTreeSize[X]$ este dimensiunea subarborelui nodului X.

Pentru a verifica dacă elementele din acest interval formează o permutare, trebuie să verificăm daca toate elementele sunt distincte și maximul din acest interval este egal cu lungimea intervalului.

Pentru a verifica dacă elemente sunt distincte, vom construi vectorul $FurthestDistinct$, unde $FurthestDistinct[i]$ reprezintă cel mai din stânga indice pentru care subsecvența $[FurthestDistinct[i], i]$ conține elemente distincte. Acest lucru se poate face cu ajutorul unui algoritm de tip **Two Pointers**, împreună cu un vector de frecvență. Condiția ca intervalul $[left, right]$ să conțină elemente distincte este ca $FurthestDistinct[right] \leq left$

Pentru a afla maximul din intervalul curent, putem folosi **RMQ**.

De asemenea, o soluție care determină MEX-ul intervalului ar putea intra în acest Subtask, implementat cu atenție.

Complexitatea temporală - $O(NlogN + Q)$

Sursa de la această soluție poate fi accesată aici.

7.4 Subtask 4 - 29 puncte

Putem observa că dacă un șir este o permutare, atunci suma elementelor acelui șir este egală cu $\frac{n*(n+1)}{2}$, unde n este lungimea șirului. Astfel, dacă elementele

din intervalul de Query sunt distincte și suma elementelor din interval este egală cu $\frac{n*(n+1)}{2}$, unde n reprezintă lungimea intervalului de Query, atunci șirul din intervalul de Query formează o permutare. Putem determina suma dintr-un interval de Query folosind sume parțiale.

Complexitatea temporală - $O(N + Q)$

De asemenea, o soluție cu arbori de intervale cu complexitatea temporală $O(N + QlogN)$ poate lua 100 de puncte.

Sursa de la această soluție poate fi accesată aici.

7.5 Soluție video

Soluție video