# Editorial RoAlgo Educational Contest 3


18 August 2024

# Cuprins

# 1 Multumiri

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

- Matei Ionescu, autorul problemelor și laureat la concursurile de informatică și membru activ al comunității RoAlgo;
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova;
- Ștefan Alecu, creatorul acestui șablon LaTeXpe care îl folosim;
- Eduard Lucian Pîrțac, Raul Ardelean, Cosmin Pascale si Andrei Chertes, testerii concursului, care au dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei;
- Ștefan Dăscălescu, coordonatorul rundei;
- Comunității RoAlgo, pentru participarea la acest concurs.

# 2 Problema 1

Autor: Ionescu Matei

## 2.1 Soluția oficială

Fie $p$ prima poziție a unui număr impar. Este destul de clar că nu este optim să utilizăm o operație pe valori mai mici ca $p$, deoarece vom transforma un număr par într-un număr impar, ceea ce duce la folosirea în plus a unei operații. Atunci, putem deduce următorul algoritm : căutăm prima poziție a unui număr impar, fie ea $q$, și folosim o operație pe $q$. Cum valorile lui $q$ pe care le vom alege pe parcursul algoritmului sunt crescătoare, putem parcurge vectorul de la început la sfârșit, menținând pe parcurs dacă numarul curent este par sau impar.

### 2.1.1 Cod sursă

Soluție de 100

# 3 Problema 2

Autor: Ionescu Matei

## 3.1 Soluția oficială

Considerăm funcția :

$f(x)$ = HP-ul minim al lui Bahoi pe parcursul jocului dacă inițial P = x.

Este destul de evident faptul că funcția $f$ este strict crescătoare, i.e. $f(x-1) < f(x)$. Pentru a demonstra afirmația, vom lua în considerare poziția $q$ unde vom atinge minimul pentru $P = x - 1$.

$$R - \sum_{i=1}^{q} a_i + (x-1) * (q-1) < R - \sum_{i=1}^{q} a_i + x * (q-1) \Rightarrow$$
$$\Rightarrow (x-1) < x, \text{ adevărat pentru oricare x numar natural}$$

Ca să aflăm x-ul minim pentru care $f(x) > 0$ vom folosi cautare binară, complexitatea finală fiind $\mathcal{O}\left(N \cdot \log N\right)$.

### 3.1.1 Cod sursă

Soluție de 100

# 4 Problema 3

Autor: Ionescu Matei

## 4.1 Soluția oficială

Pentru două poziții $i$ și $j$ ($i < j$), $xr_i$ va fi egal cu $xr_j$ dacă
$A_i \oplus A_{i+1} \oplus \cdots \oplus A_j = 0$.
Observăm că primul prefix cu suma xor egală cu 0 nu ne încurcă. Adică dacă $xr_i = 0$, dar 0 nu mai apare până acum în vectorul $xr$, putem da skip la interval. Așadar scopul nostru este să scăpam de toate secvențele cu suma xor 0 care se află strict în interioriul vectorului.

### 4.1.1 Soluția I

Pentru fiecare poziție $i$ vom calcula care este $j$-ul maxim mai mic ca $i$ astfel încât $xr_i = xr_{j-1}$. Astfel ne formăm o mulțime de intervale. Evident e faptul că este necesară o operație per interval, prin urmare, pentru a ne minimiza numărul de operații folosite, va trebui să grupăm intervalele în componente cât mai compacte, cu condiția că toate intervalele din fiecare componentă să aibă un element comun.

Pentru a afla componentele, vom sorta mai întâi intervalale după capătul

drept. Apoi vom parcurge intervalele de la stânga la dreapta și vom menține intersecția lor până-n momentul actual. Dacă intersecția intervalelor conține măcar un element comun cu intervalul curent, vom actualiza intersecția, iar dacă nu, vom creea o nouă componentă cu intervalul curent.
Rezultatul este numărul de componente. Complexitatea este $\mathcal{O}\left(N \cdot \log N\right)$.
Soluție de 100

### 4.1.2 Soluția II

Vom menține xorul pe prefixe până în momentul $i$ într-un set. Odată ce găsim un indice $p$ pentru care $xr_p$ aparține de set, facem o operație în $p$ și eliminăm restul elementelor curente din setul nostru.
Complexitatea este $\mathcal{O}\left(N \cdot \log N\right)$. Soluție de 100

# 5 Problema 4

Autor: Ionescu Matei

## 5.1 Soluția oficială

Considerăm un vector cu 2 elemente. Ca să respecte proprietatea din enunț, diferența absolută dintre cele două numere trebuie să fie mai mică sau egală cu 1. Cum orice vector cu mai mult de două elemente conține măcar o secvență cu exact două elemente, deducem că singura posibilitate pentru vectorul nostru este ca $|a_i - a_{i+1}| \leq 1, 1 \leq i < n$.

### 5.1.1 Soluția I

Fie matricea: $\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & ... & N \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ \vdots & & & & & & & & & \\ M & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \end{bmatrix}$

Rezultatul este numărul de drumuri pe care le putem face dacă începem de pe prima coloană (i.e. coloana cu numărul 1) și ajungem pe ultima coloană ( i.e. coloana cu numărul $N$), facând următoarele operații:

1. $(x, y) \rightarrow (x, y + 1)$
2. $(x, y) \rightarrow (x - 1, y + 1)$
3. $(x, y) \rightarrow (x + 1, y + 1)$

Evident că la oricare moment, nu avem voie să ieșim din matrice.
Ca să rezolvam problema ne putem folosi de programare dinamică. Fie $dp(x, y)$ = numărul de drumuri de a ajunge în celula $(x, y)$. Atunci:

$$dp(x, y) = dp(x - 1, y - 1) + dp(x, y - 1) + dp(x + 1, y - 1)$$

Vom optimiza soluția folosindu-ne de **divide and conquer**.
Aici, $dp_{l,r}(x, y)$ va reprezenta numărul de drumuri dacă am începe de pe coloana $l$, linia $x$, și am termina pe coloana $r$, linia $y$. Împărțim intervalul în două, $mid = \frac{l+r}{2}$.

$$dp_{l,r}(x, y) = \sum_{k=1}^{M} dp_{l,mid}(x, k) \cdot dp_{mid,r}(k, y)$$

Dacă $r - l + 1$ este par, atunci $dp_{l,mid} = dp_{mid,r}$, iar dacă este impar, putem să adaugăm coloana care este în plus la unul dintre $dp$-uri în $\mathcal{O}\left(M^2\right)$.

Odată ce $r - l + 1 \leq 2$, putem opri algoritmul pentru că în ambele cazuri intervalul $(mid, r)$ va coincide cu intervalul $(l, r)$. Pentru lungimi așa de mici putem calcula matricea $dp$ brut.

Rezultatul este $\sum_{i,j \leq M} dp_{1,N}(i, j)$. Complexitatea finală este $\mathcal{O}\left(M^3 \cdot \log N\right)$.
*Observație.* Felul în care este calculat $dp_{l,r}$ aduce foarte bine cu înmulțirea a două matrici. De fapt, putem rescrie formula de mai sus ca $dp_{l,r} = dp_{l,mid} \cdot dp_{mid,r}$.

Soluție de 100.

### 5.1.2 Soluția II

Fie $f_0 = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix}$. Sunt în total exact $M$ elemente în matricea $f_0$.

Fie $T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 1 & 1 & \dots & 0 \\ \vdots & & & & & & \\ 0 & 0 & \dots & 1 & 1 & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & 1 \\ 0 & 0 & \dots & 0 & 0 & 1 & 1 \end{bmatrix}$. Matricea $T$ este de $M \cdot M$ elemente.

Fie $f = f_0 \cdot T^N$. Rezultatul este $\sum_{i=1}^{M} f_i$. Complexitatea finală este $\mathcal{O}\left(M^3 \cdot \log N\right)$.
Soluție de 100.

### 5.1.3 Fun fact

Există algoritmi care ar putea înmulții două matrici de $N \cdot N$ elemente mai rapid decât algoritmul clasic de $N^3$. Un asemenea algoritm este Strassen care poate rezolva problema în $\mathcal{O}\left(N^{\log_2 7}\right) \approx \mathcal{O}\left(N^{2.8074}\right)$, însă în practică nu va face o diferență semnificativă, facând uneori algoritmul clasic să se comporte mai bine în unele cazuri.

# 6 Problema 5

Autor: Ionescu Matei

## 6.1 Soluția oficială

Dacă trasăm muchii de la $i$ la $i + A_i$, atunci vectorul nostru va fi alcatuit din mai multe arborescențe.

Pentru un interval $(l, r)$, numim rădăcinile intervalului $(l, r)$ acele noduri care nu au parintele în interioriul intervalului, adică $i \leq r$ și $i + A_i > r$. Putem precalcula pentru fiecare nod suma valorilor nodurilor pe lanțul spre rădăcina arborelui în care se află, fie ea $sp_{nod}$.

Astfel, rezultatul pentru un interval $(i, j)$ este

$$\sum_{r \in S} \sum_{d \in T} sp_d - ((sp_r + A_r) \cdot |T|)$$

unde $S$ reprezintă mulțimea rădăcinilor pentru intervalul $(i, j)$, iar $T$ reprezintă mulțimea nodurilor din subarborele lui $r$ incluse în interval. Putem răspunde la întrebări în mod **offline**, parcurgând vectorul de la dreapta spre stânga, 'activând' pe rând fiecare nod. Ca să calculăm eficient suma $sp$-urilor nodurilor din subarborele fiecarei rădăcini, vom liniariza

fiecare arbore folosindu-ne de câte un **dfs**. Pentru update și query putem utiliza fie un arbore de intervale, fie un arbore indexat binar.
Cum fiecare interval are maxim $Amax \leq 30$ rădăcini, complexitatea finala este $\mathcal{O}\left((N+Q) \cdot Amax \cdot \log N\right)$.

### 6.1.1 Cod sursă

Soluție de 100

# 7 Problema 6

Autor: Ionescu Matei

## 7.1 Soluția oficială

Dacă intervalul $(l, r)$ este sigma, nu înseamnă ca și intervalul $(l, r+1)$ este sigma. Nu avem garanția că maximul din vectorul $A$ pe intervalul $(l, r+1)$ rămâne între minimul din $B$ și maximul din $B$. Cu toate acestea, daca $max(A_l, \dots, A_r) = max(A_l, \dots, A_{r+1})$, atunci putem spune că dacă $(l, r)$ este sigma, garantat și $(l, r+1)$ este.

Pentru a calcula cu ușurință rezultatul, ne putem folosi de un arbore cartezian, sau treap implicit construit pe vectorul nostru, unde cheile sunt pozițiile elementelor din vector, iar prioritățile sunt valorile din vector. Pentru a ne avantaja și mai mult, prioritățile vor fi în ordine descrescătoare. Cu astea fiind spuse, putem simula un algoritm de tip small-to-large, unde vom fixa pe rând câte un nod, și vom căuta binar în sensul opus poziției lui față de radăcina actuală. Astfel ne formăm o mulțime de mai multe intervale de forma $(l, r_1, r_2)$ cu semnificația că intervalele care încep la $l$ și se termina în intervalul $(r_1, r_2)$ sunt sigma, sau, $(l_2, l_1, r)$ cu același înțeles doar că cu ordinea schimbată. Cum avem $\mathcal{O}\left(N \cdot \log N\right)$ noduri pe care le vom parcurge prin algoritmul de tip small-to-large, vom avea tot $\mathcal{O}\left(N \cdot \log N\right)$ intervale ca

cele menționate mai sus.

Pentru a calcula suma propriu-zisă, vom parcurge vectorul în ambele sensuri, și cu un arbore de intervale și o stivă putem face update-uri de tipul : $A_p = max(A_p, x)$ pe un interval și queriuri de tipul : $A_l + A_{l+1} + \cdot + A_r$.. Altă posibilitate ar fi să ne folosim de un segment tree beats, însă complexitatea rămâne la fel, și anume $\mathcal{O}\left(2N \log N + N \log N^2\right)$

### 7.1.1 Cod sursă

Soluție de 100