RoAlgo Contest 7 - Editorial

Ștefan-Cosmin Dăscălescu, Matei Ionescu,
Theodor Pirnog, Traian Mihai Danciu

5 Noiembrie 2023

1 Mulțumiri

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

- Matei Ionescu, Theodor Ioan Pirnog, Traian Mihai Danciu, autorii problemelor și laureați la concursurile de informatică și membri activi ai comunității RoAlgo.
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova
- Andrei Chertes, Ioana Franț, testerii concursului, care au dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei.
- Ștefan-Cosmin Dăscălescu, coordonatorul rundei, și autorul soluțiilor video.
- Comunității RoAlgo, pentru participarea la acest concurs.

2 Problema nr9

Autor: Matei Ionescu

2.1 Soluția oficială

Observația necesară este că dacă n nu este divizibil cu 9, oricum am rearanja cifrele lui n obținem un număr care nu este divizibil cu 9. Acest lucru se întâmplă deoarece restul împărțirii lui n la 9 este dat de restul împărțirii sumei cifrelor lui n la 9. Atunci, dacă n nu este divizibil cu 9, putem afișa direct n, altfel vom afișa $n - 1$, deoarece $n - 1$ nu este divizibil cu 9 dacă n este. Complexitate: $O(1)$ timp și $O(1)$ memorie.

2.2 Cod sursă și soluție video

Soluție de 100

Soluție video

3 Problema divnr

Autor: Traian Mihai Danciu

3.1 Soluția oficială

Soluția se bazează pe faptul că putem afișa un număr de forma $x \cdot 10^k$ (x urmat de k zero-uri), unde x este cel mai mic multiplu comun al tuturor numerelor din șir, iar $k = m -$ numărul de cifre al lui$x$. Din moment ce x este cel mai mic astfel de număr, și există un număr pentru fiecare test, deducem că $m \geq$ numărul de cifre al lui$x$. Deci soluția este corectă. Complexitate: $O(n+m)$ timp și $O(1)$ memorie, pentru fiecare test.

Desigur există și alte soluții, precum afișarea numărului $10^{m-1} + (x - (10^{m-1} \bmod x))$.

3.2 Cod sursă și soluție video

Soluție de 100

Soluție video

4 Problema muxusetre

Autor: Traian Mihai Danciu

4.1 Soluția oficială

Fie $z_i =$ numărul de 0 până la i. Fie $u_i =$ numărul de 1 până la i. Pentru ca secvența $[l, r]$ să fie bună, trebuie să avem $z_{l-1} - u_{l-1} = z_r - u_r \pmod 2$.

O observație importantă este că $z_i - u_i \neq z_{i+1} - u_{i+1}$. Acest lucru se întâmpla, deoarece când adăugam al $i+1$-lea caracter, exact unul dintre z_{i+1} și u_{i+1} își schimbă paritatea când trecem de la i la $i+1$ (deoarece adunăm 1 la exact unul dintre ele). Atunci diferența își va schimba paritatea.

Încă o observație care ne duce spre soluție este că $z_1 - u_1 = \pm 1$. Atunci rezultă că $z_i - u_i = i \pmod 2$.

Deci problema se reduce la aceasta: Se dă un număr n, în câte moduri putem lua 2 indici care să aibă aceeași paritate (observați că aici este inclus și 0). Pentru a rezolva această problemă nouă, trebuie doar să numărăm câți indici pari avem (notat cu p), și câti indici impari avem (notat cu i):

$p = \lfloor \frac{n}{2} \rfloor + 1$

$i = \lceil \frac{n}{2} \rceil$

Iar acum putem lua doi indici pari în $\frac{p \cdot (p-1)}{2}$, iar cu impari în mod asemănător: $\frac{i \cdot (i-1)}{2}$. Și adunând aceste două numere, obținem rezultatul. Complexitate: $O(n)$ timp (deoarece trebuie să citim și șirul, ca să aflăm următorul n) și $O(1)$ memorie, pentru fiecare subtest.

Soluția care reține numerele z_i și u_i în 2 variabile, pe care le actualizeaza, va lua și ea punctajul maxim, dar având în vedere ce am spus mai sus, nu este nevoie de acele 2 variabile.

4.2 Cod sursă și soluție video

Soluție de 100

Soluție video

5 Problema 3secv

Autor: Traian Mihai Danciu

5.1 Soluția oficială

Ne putem folosi câteva variabile, în care reținem cele 3 valori distincte din secvența curentă, și unde au apărut ultima oară. Când întâlnim o valoare diferită de cele 3, o înlocuim pe cea care a apărut ulima oară cel mai târziu cu noua valoare. Complexitate: $O(n)$ timp, $O(1)$ memorie.

5.2 Cod sursă și soluție video

Soluție de 100

Soluție video

6 Problema cuvinte

Autor: Theodor Ioan Pirnog

6.1 Soluția oficială

Expresia se va evalua folosind o stivă sau cu ajutorul recursivității indirecte. Se vor extrage parametrii fiecărei funcții (care pot fi simpli: $reverse(\text{"}exemplu\text{"})$ sau compuși: $reverse(order(\text{"}exemplu\text{"})))$. Odată știute funcția și parametrii ei, se poate trece la prelucrarea aferentă fiecărei funcții. Complexitate: $O(N * 100)$.

6.2 Cod sursă și soluție video

Soluție de 100

Soluție video

7 Problema joingraf

Autor: Traian Mihai Danciu

7.1 Soluții parțiale

Pentru subtaskurile $1, 2, 3, 4$ este suficientă o abordare care folosește arbori de intervale cu lazy.

7.2 Soluția oficială

Pentru a rezolva această problemă, trebuie mai întâi să observăm că componentele conexe sunt ca niște intervale. De exemplu, să luăm $n = 7$. Atunci, la început intervalele vor fi: $[1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6], [7, 7]$. Dacă unim muchiile de la 3 la 6, intervalele vor deveni: $[1, 1], [2, 2], [3, 6], [7, 7]$.

Atunci, putem folosi o structură de tip pădure de mulțimi disjuncte. Vom reține $par_i =$ "părintele" nodului i, sau mai ușor de înțeles, capătul stânga al intervalului în care este nodul i. Este nevoie să reținem doar capătul dreapta, deoarece capătul dreapta al secvenței curente este predecesorul capătului stânga al secvenței următoare. Vom reține și $nxt_i =$ capătul stânga al secvenței de după secvența în care este i.

Iar atunci când avem update cu x, y, mergem la fiecare secvență până la y (adică când avansăm de la p la următoarea, facem $p = nxt_p$) și o reunim cu secvența în care este x.

Iar la query, verificăm dacă intervalul în care este x este egal cu cel în care este y. Complexitate: $O(N + Q\ log\ N)$ timp, $O(n)$ memorie.

7.3 Soluție alternativă

Putem, în loc să folosim păduri de mulțimi disjuncte, să folosim un set. Iar update-ul și query-ul se implementează asemănător cu soluția oficială.

7.4 Cod sursă și soluție video

Soluție de 100

Soluție alternativă

Soluție video

8 Problema suxumetre

AUTOR: MATEI IONESCU

Vom simplifica mai intai problema prin precalcularea costului pe fiecare interval în parte.

O idee ar fi să fixăm un i și să reținem suma pe intervalul $[i, j]$ mod m, și atunci putem pentru fiecare k de la 1 la m să numărăm câte interval cuprinse intre $[i, j]$ se termină în j și au suma mod $m = k$. Dacă știm că $suma[j] = \sum_{k=i}^{j} v[k] = A \bmod m$, atunci numarul de intervale care se termină in j și au $suma \mod m = k$ va fi dat de pozitiile unde $suma[h] = (A + m - k) \bmod m, h \leq j$. Ca să le numărăm putem să reținem un vector de frecvență cu M elemente.

8.1 Soluții parțiale

8.1.1 Subtask 1: 10 puncte

Putem sa folosim backtracking ca sa găsim o împărțire optimă în k secvențe disjuncte și nevide sau ne putem ajuta de bitmask-uri iar biții setați vor delimita cele K intervale. Observăm că vom avea nevoie de $K + 2$ biți (vom pune un bit pe pozitia 0 si unul pe pozitia $n + 1$); Complexitatea temporală: $O(2^n \cdot n)$;

8.1.2 Subtask 2 si 3: 40 puncte

Ne vom folosi de dinamica $dp_{i,j}$ = costul minim ca să împărțim primele j elemente în i intervale. Recurența reiese astfel:
$dp_{i,j} = min(dp_{i-1,k} + C(k + 1, j)$; Complexitate temporală: $O(n^2 \cdot k)$

8.2 Soluția oficială

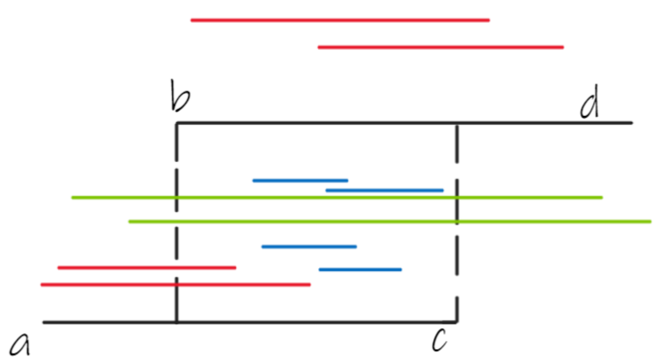
Dinamicele de genul sunt foarte clasice, chiar au o optimizare foarte faină. Pentru un dp cum ar fi $dp_{i,j} = min(dp_{i-1,k} + C(k+1, j)$ denotăm $opt(i, j)$ ca fiind "the optimum splitting point" și egal cu k-ul care minimizeaza $dp_{i,j}$.

Atunci putem spune că $opt(i, j-1) \leq opt(i, j)$ pt oricare (i, j) doar dacă oricum am alege patru indici $(a, b, c, d), C(a, b) + C(b, d) \leq C(a, d) + C(b, c)$;

$$C(x, y) = \sum_{x \leq i \leq j \leq y} sp(i, j), \text{ unde } sp(i, j) = \sum_{k=i}^{j} v_k \mod m$$

Demonstrația inegalității în cazul nostru (de *Andrei Chertes*)

Conform definiției avem $sp(x, y) \geq 0$ și $C(x, y) \geq 0$.



În primul rand vom analiza ce intervale contribuie la costurile din inegalitatea de mai sus. Distingem trei tipuri de intervale:

1. interval roșu: inclus în $[a, c]$ sau $[b, d]$, dar nu în $[b, c]$.

2. interval albastru: inclus în $[b, c]$

3. interval verde: inclus în $[a, d]$, dar nu în $[a, c]$ sau $[b, d]$

Vom scrie membrul stâng și membrul drept al inegalității în funcție de cele trei tipuri de intervale:

$$C(a, c) + C(b, d) = \sum sp(interval_{roșu}) + 2 \sum sp(interval_{albastru})$$

$$C(a, d) + C(b, c) = \sum sp(interval_{roșu}) + 2 \sum sp(interval_{albastru}) + \sum sp(interval_{verde})$$

$$C(a,c) + C(b,d) \leq C(a,d) + C(b,c) \Leftrightarrow 0 \leq \sum sp(interval_{verde})$$

Ceea ce este adevărat din definiția funcției $sp(x, y)$.

Putem deci sa dezvoltăm un algoritm tip "divide and conquer" unde vom împărți succesiv vectorul în 2 intervale $[st, mij], [mij + 1, dr]$ și să calculăm dp-ul pentru mij în intervalul $[opt(i, st), opt(i, dr)]$.

Această optimizare se numește "divide and conquer dp" si puteți citi mai multe de pe linkul acesta. Complexitatea finală: $O(k \cdot n \ log \ n)$

8.3 Cod sursă și soluție video

Soluție de 100

Soluție video

9 Problema cucuruz

Autor: Matei Ionescu

Practic problema ne întreabă care este suma valorilor de pe fiecare lanț cu lungimea mai mică sau egală cu P. Restricția cu $W < D$ este pusă pentru a sugera că nu se numără același drum de 2 ori.

9.1 Soluții parțiale

9.1.1 Subtaskturile [1, 2, 3, 4], 30 puncte

Putem sa precalculăm sumele parțiale pentru fiecare nod și să fixăm oricare 2 noduri, verificăm dacă distanța dintre cele 2 noduri este mai mică sau egală cu p și adunăm la rezultat suma de pe lantul respectiv; Complexitate: $O(n^2 \ log \ n)$

9.1.2 Subtask 5 : 20 puncte

Putem să ne calculăm în $cnt(nod, i)$ suma de pe toate lanțurile care încep în nod și se termina într-un nod din subarborele lui cu distanța i și în $pl(nod, i)$ câte noduri din subarborele lui nod sunt la distanță i față de nod.

Acuma putem pentru fiecare nod să ne calculăm suma totala pentru lanțurile care încep în subarborele lui nod, trec prin nod și se termină tot în subarborele lui nod.

Fie $f_1, f_2, f_3, \ldots, f_k$ toți fii lui nod. Dacă ne alegem un nod f_i și o distanță p_i, atunci putem să aflăm suma $cnt(f_k, p_k)$ și $pl(f_k, p_k)$, cu $k < i$ și $p_i + p_k \leq P$ și să adunăm la rezultat $suma + cnt(f_i, p_i) \cdot suma1$; Pentru a optimiza aflarea sumei și a numărului de noduri aferente, putem să folosim 2 arbori indexați binar. Complexitate: $O(p \cdot n \ log \ n)$

Putem să reținem doar sumele parțiale și atunci complexitatea devine $O(n * p)$.

9.1.3 Subtask 6: 10 puncte

Putem face același lucru ca la subtask ul anterior, doar că $P = 2$ și atunci putem să calculăm totul in $O(n)$

9.2 Soluția oficială

Putem reimplementa ideea de la subtask-ul 5 doar ca vom aborda diferit problema și ne vom folosi de o tehnică foarte utilă numită "Centroid Decomposition". O idee bună ar fi să aflăm centroidul arborelui inițal, aplicăm ideea de la subtaskul 5, eliminăm centroidul din arbore și apelăm recursiv pentru fiecare subarbore rezultat.

De ce complexitatea devine $O(n \cdot log^2 \ n)$? Mai întâi hai să clarificăm ce este un centroid. Un centroid este un nod pe care dacă-l eliminăm, numărul de noduri din subarborii rezultați sunt mai mici sau egali cu $N/2$, unde N e numărul de noduri din arborele inițial.

Astfel dacă descompunem arborele eliminând centroizii, atunci fiecare nod va aparține de maxim $O(log \ n)$ componente conexe pe parcursul descompunerii. Astfel complexitatea se poate exprima ca $O(log \ n) \cdot f(N)$ unde $f(N)$ este timpul necesar pentru a afla toate lanțurile bune care trec prin fiecare centroid.

$f(N) = O(n \ log \ n)$ deorece trebuie să precalculăm matricea *cnt* pentru fiecare nod din componenta respectiva $\cdot log \ n$ (de la aib-uri). Putem scăpa de matricea *cnt* și *pl* dacă luam doar suma nodurilor de la A la centroid, unde A este un nod din aceeași componentă cu centroidul.

9.3 Cod sursă și soluție video

Soluție de 100

Soluție video