Editorial NiceContest (RoAlgo Contest #12)

25 Ianuarie 2025

Cuprins

1 Multumiri

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

- Robert Moldovan și Ștefan Dăscălescu, coordonatorii rundei;
- Robert Moldovan, autorul problemelor;
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova;
- Ștefan Alecu, creatorul acestui șablon LaTeXpe care îl folosim;
- Raul Ardelean, Ștefan Vîlcescu, Radu Vasile, Rareș Hanganu, Matei Ionescu și Andrei Chertes, testerii concursului, care au dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei;
- Comunității RoAlgo, pentru participarea la acest concurs.

2 A. Nice PP

Autor: Robert Moldovan
Cuvinte cheie: implementare

2.1 Soluția oficială

Problema este una de lucru cu numere reale. Nu trebuie decat sa se aplice formula din enunt pentru a obtine Aim PP-ul, Speed PP-ul si Accuracy PP-ul, dupa care se afiseaza $(AimPP + SpeedPP) * AccuracyPP$.

Cod sursă

Soluție de 100

3 B. Nice Mustache Growth

Autor: Robert Moldovan
Cuvinte cheie: two pointers, set, deque

3.1 Soluția oficială

Problema cere să numărăm toate intervalele pentru care $a \leq max([l,r]) - min([l,r]) \leq b$. Observăm că dacă $F(k) =$ numărul de intervale pentru care $max([l,r]) - min([l,r]) \leq k$, putem scrie răspunsul ca $F(b) - F(a-1)$. Rămâne să calculăm $F(k)$ eficient.

Observăm că, având un interval $[l,r]$ și crescând r, max poate doar să crească și min poate doar să scadă. Analog, dacă creștem l, max poate doar să scadă și min poate doar să crească. În alte cuvinte, crescând r, diferența dintre maxim și minim poate doar să crească. Analog, dacă creștem l, diferența dintre maxim și minim poate doar să scadă.

Astfel, ne vine ideea să rezolvăm problema cu tehnica celor doi pointeri. Fixăm capatul r și atât timp cat $max([l,r]) - min([l,r]) > k$, creștem l. Acest interval contribuie cu $r - l + 1$, fiindcă orice sufix al său este bun datorită monotoniei. Rămâne de calculat $max[l,r]$ și $min[l,r]$ eficient. Pentru asta există o multitudine de abordări, dintre care vom explora două.

1. Set-uri

Putem ține un `set` (o mulțime ordonată de numere, de exemplu un `std::multiset<int>` (numerele se pot repeta)), unde putem accesa minimul și maximul. Astfel, când creștem r, inserăm A_r în set, iar când creștem l, ștergem A_l din set **(atenție: doar una dintre aparițiile sale!)**. Complexitate: $O(NlogN)$.

2. Deque

Observăm de asemenea că ambii pointeri l și r pot doar să crească. Astfel, ne vine ideea să folosim două deque-uri monotonice, unul pentru minime și unul pentru maxime. Astfel, `dq_min.front()` va ține indicele minimului din $[l, r]$, iar `dq_max.front()` va ține indicele maximului din $[l, r]$. Când creștem r, eliminăm toate elementele din spatele cozii care sunt mai mici decat A_r, după care punem indicele r în spatele cozii. Când creștem l, scoatem din fața cozii toți indicii care sunt mai mici decât l. Se procedează analog pentru coada de maxime. Complexitate: $O(N)$.
Ambele soluții ar trebui să ia cu ușurință punctaj maxim.

Cod sursă

Soluție de 100 folosind set-uri
Soluție de 100 folosind deque

4 C. Nice Boring Problem

Autor: Robert Moldovan
Cuvinte cheie: matematică, implementare

4.1 Soluția oficială

Observăm că putem calcula rezultatul pentru fiecare dimensiune independent. Astfel, am redus problema la $\sum_{i=1}^{N} \sum_{j=1}^{i-1} (a_i - a_j)^2$.

Folosind formula $(a_i - a_j)^2$, obținem $\sum_{i=1}^{N} \sum_{j=1}^{i-1} (a_i^2 - 2 * a_i * a_j + a_j^2)$. Putem descompune suma în 3 sume pe care le putem calcula mai eficient, anume:

$A = \sum_{i=1}^{N} \sum_{j=1}^{i-1} (a_i^2)$

$B = \sum_{i=1}^{N} \sum_{j=1}^{i-1} (2 * a_i * a_j)$

$C = \sum_{i=1}^{N} \sum_{j=1}^{i-1} (a_j^2)$

Observăm că termenul lui A nu se schimbă, așadar se reduce la $\sum_{i=1}^{N} (i-1) * (a_i^2)$.

Dând factor comun pe $2 * a_i$, obtinem $B = \sum_{i=1}^{N} (2 * a_i * \sum_{j=1}^{i-1} a_j)$.

Folosind sumele parțiale $P1_i = \sum_{i=1}^{N} a_i$ și $P2_i = \sum_{i=1}^{N} a_i^2$, calculul rezultatului se poate face în $O(N * K)$.

Cod sursă

Soluție brută de 30 de puncte

Soluție de 100 de puncte folosind sume parțiale

5 D. Nice Compatibility

Autor: Robert Moldovan
Cuvinte cheie: combinatorică, PIE

5.1 Soluția brută (5 puncte)

Procesăm toate submulțimile de jucării și dacă au cel puțin o jucărie in comun creștem un contor.

Complexitate: $O(2^N)$ sau $O(2^N * K)$, în funcție de implementare

5.2 Soluție cu bitmask DP / memoizare la brut (30 puncte)

Vom defini $dp[i][mask]$ = numărul de submultimi din primele i jucării care au culorile reprezentate de masca de biti $mask$ în comun. De asemenea, vom defini $cmask(i)$ ca fiind masca de biți corespunzătoare culorilor jucăriei i. Astfel, recurența este următoarea:

$dp[i+1][mask\&cmask(i)] = dp[i+1][mask\&cmask(i)] + dp[i][mask]$ (luăm jucăria i în submulțime);
$dp[i+1][mask] = dp[i+1][mask] + dp[i][mask]$ (nu o luăm).

Presupunând că jucăriile sunt indexate de la 0, rezultatul se află în suma a tuturor $dp[N][i]$, unde i reprezintă o mască nevidă.

De asemenea, acest subtask se poate obține pur și simplu adăugând memoizare soluției recursive.

Complexitate: $O(N * 2^K)$

5.3 Soluție cu PIE (100 puncte)

Vom defini $S(i)$ ca fiind submulțimile corespunzătoare culorii i. Astfel, rezultatul nostru este reuniunea tuturor $S(i)$. Putem calcula cardinalul acestei reuniuni folosind principiul includerii și excluderii. Astfel, adunăm toate $S(i)$ la rezultat, dar scădem intersecțiile de câte două $S(i)$-uri ca să nu le numărăm de mai multe ori. În mod asemănător, adunăm intersecțiile de câte trei $S(i)$-uri ca să nu numărăm prea puțin, dar scădem intersecțiile de câte patru $S(i)$-uri ca să nu numărăm prea mult ș.a.m.d. De asemenea, trebuie să avem grijă la cazurile când obținem o mulțime vidă de jucării sau alegem o submulțime de culori care conduce la o mulțime vidă.

Complexitate: $O(2^K * logN)$

Cod sursă

Soluție de 5 puncte folosind o soluție recursivă
Soluție de 30 de puncte folosind memoizare
Soluție de 100 de puncte folosind PIE

6 E. Nice Limbo

Autor: Robert Moldovan
Cuvinte cheie: combinatorică, DP, exponențiere rapidă de matrici

6.1 Soluție brută (10 puncte)

Aplicăm un algoritm de backtracking pentru a genera toate combinările de mutări posibile.

Complexitate: $O(8^N)$

6.2 Soluție DP (60 puncte, 100 puncte)

Observăm că nu are rost să aplicăm transformările pe matrice efectiv, fiindcă ne interesează doar de poziția $(0, 0)$. Astfel, la un pas putem aplica una dintre transformările din enunț. Se va calcula pentru fiecare transformare, pentru fiecare poziție, unde va ajunge aceasta după transformarea respectivă. Vom asocia fiecărei poziții un număr, reprezentând indicele stării. Pe lânga asta, vom construi un multigraf orientat în care $G[i][j]$ = numărul de modalități de a ajunge din starea i în starea j folosind o mutare.

Acum, problema s-a redus la a afla, dintre toate succesiunile posibile de transformări, câte duc din starea i în starea j. Observăm că dacă matricea A

reprezintă numărul de drumuri de lungime X de la i la j, iar matricea B reprezintă numărul de drumuri de lungime Y de la i la j, produsul acestor matrici, C, va reprezenta numărul de drumuri de lungime $X + Y$ de la i la j, deoarece $C[i][j] = \sum_{k=1}^{8} A[i][k] * B[k][j]$, iar drumurile din A (cele de lungime X) se compun cu drumurile din B (cele de lungime Y). Astfel, deoarece matricea G deasemenea reprezintă numărul de drumuri de lungime 1 de la i la j, ridicând matricea G la puterea K, putem obține numărul de drumuri de lungime K de la i la j.

Nu mai rămâne decât a aplica formula pentru E.V (expected value), adică să împărțim $G[0][j]$ la numărul total de succesiuni de transformari, anume T^K, unde T este numărul de tipuri de transformări din enunț, adica 8. Bineînteles, vorba fiind de aritmetică modulara, vom folosi inversul modular pentru împărțiri.

Complexitate: $O(S^3 * logK)$, unde S e numarul de stări posibile, în cazul nostru 8.

Cod sursă

Soluție de 10 puncte folosind un DFS brut
Soluție de 60 puncte folosind DP fără exponențiere de matrici
Soluție de 100 puncte folosind DP si exponentiere de matrici

7 F. Nice Scam

Autor: Robert Moldovan
Cuvinte cheie: cuplaj/flux, bitset

7.1 Soluția oficială

Prima observație este că dacă construim un graf de relații între toate perechile de elevi, unde adăugăm o muchie între prietenii i și j dacă și numai dacă distanța Hamming între ID-urile lor este exact 1, soluția se poate obține găsind cea mai mare submulțime de noduri cu proprietatea că nu există muchie între nicio pereche de noduri i și j.

A doua observație și cea mai importantă este faptul că acest graf de relații este mereu bipartit. Potrivit teoremei lui Konig, cardinalul min vertex cover-ului unui graf bipartit este egal cu cardinalul cuplajului maxim al său. Mai departe, cardinalul max independent set-ului este egal cu $N - |VC|$, unde cu $|VC|$ notăm cardinalul vertex cover-ului menționat anterior.

Bineînțeles, înaintea rulării algoritmului de cuplaj, va trebui să împărțim graful în două părți, și să tragem muchie între două noduri doar dacă culoarea lor diferă.

Singurul lucru care mai rămâne de optimizat este calcularea distanței Hamming. Acest lucru se poate face folosind hashuri și căutare binară, sau

mai simplu, bitseturi.

Astfel, obținem complexitatea finală de $O(\frac{N^2M}{64})$.

Cod sursă

Soluție euristică de 20 folosind un DFS
Soluție de 100 folosind cuplaj și bitset

8 G. Nice Apple Tree

Autor: Robert Moldovan
Cuvinte cheie: arbori de intervale, RMQ, diametrul unui arbore

8.1 Soluția oficială

Vom începe prin a defini diametrul unei mulțimi ca fiind perechea de noduri din acea mulțime care se află la distanță maximă în arbore.

O observație crucială pentru rezolvarea problemei este că putem combina doi diametri A si B, luând maxim 6 perechi candidate din fiecare (anume: $(A.x, A.y)$, $(B.x, B.y)$, $(A.x, B.x)$, $(A.x, B.y)$, $(A.y, B.x)$, $(A.y, B.y)$) și luând perechea de distanță maximă.

Ne vine ideea să precalculăm diametre pe intervale folosind un RMQ, după care putem găsi diametrul nodurilor din intervalul $[l, r]$ in $O(1)$ timp.

Deoarece avem update-uri în care activăm/dezactivăm intervale de noduri, ne vine ideea să folosim un arbore de intervale cu lazy propagation, unde într-un nod ținem diametrul nodurilor activate și de asemenea diametrul nodurilor dezactivate.

Pentru a da handle la update-urile de tip 3 (unde schimbăm starea nodurilor dintr-un interval), observăm ca folosind doi arbori de intervale (unul pentru

nodurile bune, altul pentru nodurile rele) și puțin case-work, putem implementa și această operatie în conjuncție cu primele două tipuri de operații.

De asemenea, pentru a asigura punctaj maxim, este necesar ca toate LCA-urile să fie calculate in $O(1)$ timp.

Complexitate finală: $O((N + Q)logN)$

Cod sursă

Soluție de 100

9 H. Nice Heap

Autor: Robert Moldovan
Cuvinte cheie: amortizare, treap, implementare

9.1 Soluția brută (15-25 puncte)

Se simulează brut operatiile. Pentru obținerea subtaskului 2, se poate folosi, de exemplu, un `std::set`, pentru menținerea set-ului cu operații de extragere și inserare, dar această soluție funcționează doar pentru k mic.

Complexitate finală: $O((N+Q) * log(N+Q) * \Sigma K)$

9.2 Soluția ? (69 puncte)

Acest subtask este doar un failsafe, în care pot intra anumite soluții mai eficiente decât soluția menționată anterior, dar nu suficient de eficiente pentru 100 de puncte.

9.3 Soluția optimă (100 puncte)

Observăm că dacă avem două șiruri sortate (descrescător) A si B, și fără pierdere a generalității primul element al lui A este mai mare decat cel al lui

B, putem căuta binar ultimul element din A care este mai mare sau egal cu primul. Extragem acest prefix din A, iar interschimbăm șirurile A și B. Astfel, se formează un număr de alternări sau interschimbări între A și B. Acest număr total de alternări pe parcursul tuturor operațiilor este chiar foarte mic, ceea ce înseamna că dacă avem o cale eficientă de a "lipi" două șiruri A și B ordonate descrescător, unde $min(A) > max(B)$, putem rezolva problema eficient. De exemplu, o structură de date care poate face aceste operații eficient este un treap cu lazy propagation.

Complexitate finală: $O((N + Q) * log(Vmax) * log(N + Q))$

Deși complexitatea nu pare deosebit de bună (în special $log(Vmax)$-ul), și faptul că constanta treap-urilor deobicei nu e foarte bună, rulează foarte bine în practică.

Mai multe informații și o demonstrație pot fi găsite aici.

Cod sursă

Soluție de 25
Soluție de 100

10 X. Nice Guess

Autor: Robert Moldovan

10.1 Soluția oficială

Ca să putem obține soluția cu 0 guess-uri, va fi nevoie să găsim o vulnerabilitate în grader. Un exemplu este faptul că x este declarat global în `grader.cxx`. Având în vedere că `grader.cxx` și `interaction.cxx` se compilează într-un binary, putem redeclara în scop global în `interaction.cxx` variabila x ca tip `extern`. Astfel, putem accesa direct variabila x din `interaction.cxx`.

10.1.1 Cod sursă

Soluție de 10
Soluție de 100
Încercare de păcălire a sistemului