IIOT 2022-23 Round 3 - Problem Solutions

Comisia IIOT

January 2023

1 Problem Patrol 2

Autor: Edoardo Morassutto

Observația 1: este întotdeauna optim să ajungem la un nod intermediar cât mai repede posibil.

Să presupunem că avem o un drum a de la 0 la x, un drum b de la x la $N-1$ și un drum c mai rapid decât a de la 0 la x.

Dacă ab este o soluție, atunci putem construi un drum nou cb și dacă, în orice moment, deplasarea la următoarul manhole ne-ar face prinși, atunci așteptăm 1 minut.

Noua soluție nu poate fi mai lentă decât cea inițială, deoarece dacă sunt introduse suficiente așteptări, vom fi în același punct cu soluția originală, astfel încât nu se vor mai adăuga așteptări.

Apoi putem folosi algoritmul lui Dijkstra pentru a calcula distanța de la origine la orice alt nod.

Când procesăm o muchie de la a la b, verificăm că această mișcare nu ne va face sa fim prinsi din urma, altfel așteptăm 1 minut înainte de a muta.

2 Problem Interval XOR

Autor: Stefan Dascalescu

Pentru primele cateva subtaskuri putem folosi o metoda bazata pe brute force pentru a raspunde la fiecare query, fie direct, fie precalculand toate raspunsurile posibile.

Pentru solutia de 100 de puncte, putem observa faptul ca x $\oplus$ (x+1) $\oplus$ (x+2) $\oplus$ (x+3) = 0, pentru toate valorile x care sunt multipli de 4. Acest fapt ne va ajuta sa calculam cu usurinta valoarea XOR-ului de la 0 la $a[i]$.

Pentru cea de-a doua parte a raspunsului, putem trece prin fiecare bit ce poate

aparea in raspuns pentru a verifica daca exista in xor sau nu, incepand de la cel mai semnificativ. Daca nu exista si nu am depasi limita superioara a query-ului, luam bit-ul respectiv, altfel nu.

3 Problem Keyboard

Autor: Davide Bartoli

Putem rezolva problema folosind programare dinamica.

Definim $dp[pos][i][j]$ ca fiind numarul minim de miscari ale degetului pentru a scrie $s[pos...N]$ daca noi deja am schimbat cheile i si j.

La inceput $i = j = -1$, iar la fiecare pas daca i si j sunt ambele egale cu -1, putem incerca fiecare schimbare.

Complexitatea de timp ar ajunge $O(N * 100)$

4 Problem Scount

Autor: Vlad Mihai Bogdan

Vom defini F ca fiind un vector astfel incat $F[i]$ = frecventa celei de-a i-a valoare distincta din V.

Mai intai vom rezolva problema pentru o tinta fixa, reprezentand frecventa fiecarei valori din subset. Raspunsul pentru o tinta fixa va fi atunci produsul tuturor (nCk($F[i]$, $target$) + 1) - 1, pentru fiecare i astfel incat $target \leq F[i]$, unde $nCk(x, y)$ este combinari de x luate cate y.

Observatia finala pe care trebuie sa o facem este aceea ca vor fi cel mult $N/target$ pozitii i in F prin care va trebui sa iteram pentru a calcula raspunsul final pentru un target dat. Dupa aceasta observatie, complexitatea de timp ar deveni $O(N * logN)$. Abordari care merg in $O(N * logN * logN)$ ar lua si ele 100 de puncte.

5 Problem Magical Forest

Autor: Aron Noszaly

Vom defini $dp[i][c]$ ca fiind numarul de drumuri magice care se termina in nodul i astfel incat ultima muchie are o valoare magica c.

Daca consideram muchiile in ordine crescatoare a valorilor magice si presupunem ca am calculat deja valorile dp pentru primele k muchii, atunci pentru a obtine valorile pentru primele $k + 1$ muchii vom putea actualiza operatiile in $O(1)$.

Daca valoarea celei de-a $k+1$-a muchii este w si leaga nodurile a si b, atunci singurele valori ce se schimba sunt $dp'[a][w] += dp[a][w-1]+1$ and $dp'[b][w] += dp[b][w-1]+1$ (ori continuam sau incepem un nou traseu magic).

Solutia va fi astfel suma valorilor dp finale minus M (deoarece consideram fiecare drum de doua ori ca fiind un drum de lungime 1).

Daca folosim o structura de date eficienta precum map/unordered map, putem calcula valorile in $O(M)$ sau $O(MlogM)$, dar complexitatea finala va fi tot $O(MlogM)$ din cauza sortarii.

6 Problem Xor Tree 2

Autor: Bence Deak

Vom nota diferenta nodurilor u si v cu $d(u,v)$.

Observatia principala este aceea ca pentru fiecare pereche de noduri $(u,v), d(u,v) = d(1,u) \oplus$ d(1, v) (unde $\oplus$ reprezinta operatia XOR), trebuie doar sa calculam $d(1,i)$ pentru fiecare i de la 1 la n, lucru ce se poate face in $O(N)$ folosind un DFS pe arborele dat.

Astfel, daca am avea doar insertii, solutia ar fi echivalenta cu gasirea valorii maxime a XOR-ului intre doua valori din vector folosind un trie. (Problema clasica. Complexitatea de timp ar fi $O(Q * logmax(w_i))$.)

Acum trebuie doar sa ne asiguram ca efectuam operatiile de stergere suficient de rapid: Deoarece complexitatea insertiilor ar fi $O(logmax(w_i))$, iar query-urile sunt offline, putem folosi inca un truc standard(Problema pentru a raspunde la query-uri in $O(Q * logmax(w_i) * logQ)$.

Astfel, pentru un numar x, scapam de stergeri prin considerarea doar a intervalelor in care x este activ. Prin construirea unui arbore de intervale pe langa query-urile date, fiecare interval ar fi reprezentat ca fiind un update in acest arbore de intervale (in fiecare nod din arborele de intervale u, vom stoca un vector $upd[u]$ cu valorile active.

Dupa ce efectuam toate actualizarile, vom rula un DFS pe arborele de intervale, pastrand si o trie cu numerele. Cand intram intr-un nod u corespunzator intervalului $[l,r]$, adaugam toate elementele din $upd[u]$ in trie. Apoi, daca u nu este o frunza, vom traversa ambii fii ai lui u. La final, cand terminam cu nodul curent, vom inversa adaugarile facute la inceput. Astfle, intre intrare si iesire, trie-ul va contine doar numerele care erau active de-a lungul intregului interval.

De fiecare data inainte sa adaugam un numar y in trie, calculam $max y \oplus x_i | i = 1, ..., m$, unde $x_1, ..., x_m$ sunt valorile curente din trie.

Daca u este o frunza, query-ul corespunzator lui u poate fi calculat pe baza acestor valorii calculate, tinand cont de maximul lor de la radacina la nodul

curent.

Complexitatea totala ar fi $O(N + Q * logmax(w_i) * logQ)$, iar memoria necesara ar fi $O(N * logmax(w_i) + Q * logQ)$.

7 Problem Not Divisible

Autor: Rabeeh Jouni

Mai intai vom incerca sa rezolvam un query pe un vector. Cerinta ar fi sa aflam valoarea minima care nu se imparte la vreo valoare din intervalul $[L, R]$ din vector.

Prima observatie ar fi ca raspunsul la query ar trebui sa fie un numar prim, daca raspunsul ar fi un numar compus, inseamna ca ar trebui sa contina un factor prim care nu e divizibil cu niciun numar din interval.

Cea de-a doua observatie ar fi faptul ca raspunsul e cel mai mic numar prim care nu se afla in interval.

Putem reformula problema sa aflam acum MEX-ul Mex) numerelor prime din intervalul [L,R].

Aceasta problema poate fi rezolvata folosind un arbore de intervale sau algoritmul lui MO daca am avea mai multe query-uri pe un vector. O varianta a acestui algoritm, MO pe arbori, MO on Trees) poate fi folosita pentru a rezolva problema pe un arbore.

Pe scurt, pasii ar fi sa liniarizam arborele, apoi sa transformam fiecare query conform regulilor descrise in linkul de mai sus, pentru a reduce problema la MO pe vector, ceea ce ar duce la o solutie in $O(Nsqrt(Q)log(N))$. Putem optimiza solutia scapand de factorul de log, prin actualizarea unui vector de frecventa in timp ce procesam query-urile folosind un MO obisnuit, iar apoi sa il impartim in blocuri de lungime $sqrt(max_value)$ pentru a obtine mex-ul in $O(Qsqrt(max_value))$ iar complexitatea totala ar deveni $O(Nsqrt(Q) + Q * sqrt(max_value))$.

Putem optimiza si mai mult algoritmul prin folosirea unui criteriu de sortare mai avansat al query-urilor, precum (Hilbert curve sorting: Hilbert Curve Optimization), dar aceasta optimizare nu era necesara pentru a obtine 100 de puncte.