IIOT 2022-23 Runda 1 - Soluții

Comisia IIOT

Noiembrie 2022

1 Problema Monopoly

Autor: Carlo Collodel, Davide Bartolli

1.1 Descrierea soluției

Dificultate: Medie

Pentru a rezolva aceasta problema se va folosi o solutie bazata pe metoda programarii dinamice.

Vom defini dp[i][j][k] ca fiind suma maxima de bani pe care Carlo ar pierde-o dupa i ture, daca se afla la pozitia j si a avut k zaruri cu duble la rand. Pentru a actualiza celelalte stari, ne vom duce din starea curenta fie intr-o stare care ne-ar duce la urmatoarea tura, fie la o stare la care am putea ajunge daca mai dam o dubla cu zarul.

2 Problema Dream

Autor: Daniel Apostol

Dificultate: Medie

2.1 Solutia Brute-force $X \leq 8$ (15 puncte)

Pentru acest subtask se pot verifica in ordine crescatoare toate numerele de X cifre daca sunt palindroame sau nu

2.2 Solutia fara manipularea numerelor mari $X \leq 18$ (35 de puncte)

Este nevoie sa facem o observatie care sa ne permita sa generam direct toate palindroamele de X cifre, anume ca al (i + 1)-lea palindrom de X cifre este de forma:

10...01 (X - 2 cifre de 0) + i * $10^{X/2+1}$ + $rasturnat(i) * 10^{X/2-nrCifre(i)}$

In alte cuvinte numarul va avea la mijloc in partea dreapta cifre care reprezinta numarul i si la stanga cifre care reprezinta rasturnatul lui i.

Se observa faptul ca aceasta formula functioneaza si atunci cand i are X / 2 cifre.

Exemplu: al 12-lea numarul de 4 cifre este $2112 = 1001 + 11 * 10^{4/2+1} + 11 * 10^{4/2-2}$

Astfel, putem calcula foarte usor suma primelor K palindroame de X. Luam fiecare i de la 0 la K - 1 si insumam rezultatul formulei de mai sus.

2.3 Solutia $O(X * K)$ (70 de puncte)

Acest subtask este creat pentru solutii care folosesc tehnici asemanatoare cu cele din solutia de 100 de puncte, calculand totul modulo 666013, dar nu o fac in cel mai optim mod. De exemplu, nu se precalculeaza puterile de 10 si nici nu se foloseste exponentiere in timp logaritmic, se ia fiecare cifra din numar neobservandu-se ca la X-uri mari vor fi foarte multi de 0, se folosesc operatii pe numere mari etc.

2.4 Solutia $O(X + K)$ (100 de puncte)

Pentru acest subtask va trebui sa scriem solutia in mod optim folosind tehnici enuntate la subtaskul de mai sus.

3 Problema Chess Tournament

Autor: Stefan Dascalescu

3.1 Descrierea soluției

Dificultate: Ușoară

Pentru fiecare jucător se va calcula valoarea medie a mutărilor și se va decide pentru fiecare jucător dacă a trișat sau nu.

Trebuie un pic de atenție la cazurile limită, de exemplu 6001/5 ¿ 1200, chiar dacă folosind împărțirea cu numere întregi ar da egal.

4 Problema Patrol

Autor: Edoardo Morassutto

4.1 Descrierea soluției

Dificultate: Medie

Pentru a rezolva problema, vom face un BFS din starea initiala $(0, 0)$. Vom tine $dist[i][j]$ ca fiind distanta minima de a ajunge de la $(0, 0)$ la (i, j), unde prima dimensiune poate lua valori pana la n iar cea de-a doua dimensiune, pana la 420, acesta fiind CMMMC-ul numerelor de la 1 la 7, lungimile corespunzatoare ale ciclurilor pe care le putem avea.

Nu in ultimul rand, trebuie avut grija sa evitam sa vizitam starile de mai multe ori decat trebuie, pentru a obtine punctaj maxim.

5 Problema Dristor

Autor: Vlad Mihai Bogdan

5.1 Descrierea soluției

Dificultate: Medie-Grea

Soluție $O(N^M)$

Vom defini $match[u][v]$ ca fiind compatibilitatea dintre al u-lea om și a v-a locație secretă.

Pentru această soluție vom genera, folosind backtracking, toate șirurile de perechi de forma (a, b), cu $match[a][b] = 1$, a din $[1, N]$, b din $[1, M]$, valorile a din subșir distincte și valorile b din subșir distincte . Dintre aceste șiruri, se numără câte dintre ele au lungime maximală. De asemenea, aceste șiruri se pot genera și sub forma numerelor cu N cifre în baza $M + 1$.

Această soluție obține 20 de puncte.

Soluție $O(2^N * M * N)$

Vom folosi programarea dinamică. Definim $count[mask][k]$ ca fiind numărul de moduri prin care putem trimite oamenii din $mask$ în primele k locații secrete. Astfel, $count[mask][k] =$
$count[mask][k-1] + \sum_{i=1}^{N} count[mask - \{i\}][k-1] \ | match[i][k] = 1.$

6 Problema Panama Sum

Autor: Stefan Dascalescu

6.1 Descrierea soluției

Dificultate: Medie-Grea

Pentru toate subtaskurile, vom tine doi vectori, unul avand elementele cu semne alternante pe pozitiile pare, respectiv impare. Cu alte cuvinte, prima valoare va fi pozitiva in primul vector, iar in al doilea vector, prima valoare va fi negativa iar mai apoi semnele ar alterna.

Pentru primul subtask, se poate calcula pentru fiecare query Panama Sum-ul pentru fiecare subsecventa si sa se gaseasca raspunsul in $O(n^2)$ pentru fiecare query, pentru 20 de puncte.

Pentru cel de-al doilea subtask, putem folosi orice algoritm eficient pentru a gasi subsecventa de suma maxima pentru a afla raspunsul pentru fiecare query in $O(n)$, obtinand 40 de puncte.

Pentru a obtine punctaj maxim, putem folosi arbori de intervale pentru a rezolva problema. Cu alte cuvinte, putem tine in fiecare nod urmatoarele informatii:

sum - suma valorilor din acel nod pmax - subsecventa de suma maxima care e un prefix in intervalul nodului dat smax - subsecventa de suma maxima care e un sufix in intervalul nodului dat maxi - subsecventa de suma maxima din intervalul nodului dat

Pentru a combina doua noduri, pentru suma adunam sumele de pe cei doi fii, pentru pmax fie luam pmax-ul de pe nodul din stanga, fie luam suma din stanga la care adunam pmax-ul din dreapta.

Pentru smax fie luam smax-ul de pe nodul din dreapta, fie luam suma din dreapta la care adunam pmax-ul din dreapta.

Pentru maxi, luam maximul dintre pmax, smax si maximul din fiecare din cei doi fii.

Pentru a rezolva query-urile, vom folosi un algoritm similar cu algoritmul de subsecventa maxima pentru fiecare nod.

7 Problema Sum Tree

Autori: Vlad Mihai Bogdan

7.1 Descrierea soluției

Dificultate: Grea

Soluție $O(N^3)$

Pentru această soluție vom fixa două noduri, u și v, $u < v$. Verificăm dacă $gcd(val[u], val[v]) > 1$. În caz afirmativ, vom merge din părinte în părinte și vom aduna valorile din noduri, până când ajungem la LCA-ul celor două noduri.

Soluție $O(N^2 * logN)$

Se va proceda ca la soluția anterioară, dar vom folosi binary lifting pentru a afla LCA-ul dintre cele două noduri.

Aceasta soluție obține 20 de puncte.

Soluție $O(N^2 * logVALMAX)$**)**

Vom parcurge arborele din fiecare nod. Așadar, vom precalcula $distance[u][v]$ ca fiind distanța de la nodul u la nodul v. Tot ce rămâne de făcut este să facem suma $distance[u][v]$ pentru perechile de noduri u și v, unde $u < v$ și $gcd(val[u], val[v]) > 1$.

Soluție $O(N^2 * logVALMAX)$

Se poate proceda la fel ca la soluția de complexitate $O(N^3)$, dar vom calcula LCA folosind o precalculare în $O(N * logN)$ cu Range Minimum Query. Astfel, putem obține LCA-ul a două noduri în $O(1)$.

Aceasta soluție obține tot 20 de puncte.

Soluție $O(2^F * N * logN)$**, unde** F **e numărul maxim de factori primi ai unui număr din șirul de valori.**

Pentru un număr liber de pătrate, vom defini $TSum(K)$ ca fiind suma costurilor tuturor perechilor de noduri distincte care au în descompunerea lor toți factorii primi din descompunerea lui K.

Astfel, putem sa reformulăm problema, folosind principiul includerii si al excluderii, în felul următor: suma costurilor tuturor perechilor de noduri distincte cu $gcd(val[u], val[v]) > 1$ este egală cu
$\sum (-1)^{countPrimeFactors(K)+1} * TSum(K)$ $|K$ este liber de pătrate.

În cele ce urmează, vom prezenta cum calculăm $TSum(K)$. Pentru început, vom reține o listă cu toate nodurile care au în descompunerea valorilor lor toți factorii primi ai lui K. Vom nota această listă cu T. Pentru a calcula costul dintre două noduri, putem calcula costul de la rădăcină la cele două noduri, din care scădem 2* costul de la rădăcină la LCA-ul lor, iar mai apoi adăugam $val[lca]$ Putem folosi această abordare pentru mai mult de două noduri. Așadar, vom calcula inițial toate costurile de tip $(rădăcină, nod)$, urmând ca mai apoi sa scădem costurile de tip $(rădăcina, lca)$.

Observația cheie în acest calcul este că, în momentul în care reținem nodurile din T în ordinea parcurgerii lor DFS, este suficient să inserăm in lista noastră doar LCA-urile dintre două noduri consecutive din T. Așadar, pentru o listă cu N noduri de interes, vom avea un arbore cu maxim $2 * N - 1$ noduri (care

sa conțină si LCA-urile).

Vom reține triplete de forma (u, v, lca), care se vor sorta descrescător după adâncimea lca-ului. Inițial, cele N noduri de interes (din T) vor reprezenta fiecare câte o mulțime cu câte un element. În timpul procesării, pentru o tripletă (u, v, lca), trebuie sa scădem
$2 * size[u] * size[v] * costuldelarădăcinălalca$, unde $size[u]$ este dimensiunea mulțimii din care face parte nodul u. Rămâne să adunăm
$size[u] * size[v] * val[lca]$ la răspuns (deoarece prin formula de mai sus am scos $val[lca]$ din costurile calculate), după care să unim mulțimile nodurilor u și v.

În ceea ce privește implementarea, trebuie folosită o structură de date care să permită aceste calcule cu mulțimi în $O(1)$ per operație. De asemenea, se recomandă folosirea calculării LCA-ului cu răspuns in $O(1)$ (Euler tour + RMQ).

Problema admite, de asemenea, soluții care se bazează pe Centroid Decomposition, Virtual Trees sau Small to Large.