IIOT 2021-22 Runda 1 - Soluții

Stefan Dascalescu

Noiembrie 2021

1 Problema Array Counting

Autor: stud. Ștefan-Cosmin Dăscălescu, Universitatea din București

1.1 Descrierea soluției

Dificultate: Medie

Problema sugerează o soluție ce are la bază combinatorică, precum și structuri de date pentru a procesa cele q query-uri date. Pentru a rezolva partea de combinatorică, va trebui să observăm următorul lucru - trebuie mai întâi să aducem toate valorile din subsecvența din query la o valoare cel puțin egală cu k. Pentru a afla cât trebuie să adaugăm valorilor din intervalul $[l, r]$, vom folosi un arbore de intervale în care vom memora pentru fiecare nod această sumă, pe care o vom nota $xtrsum$.

După ce am rezolvat această parte a problemei, problema se reduce la următoarea problemă:

Dat fiind R - L + 1 numere egale cu 0, care este numărul de moduri de a crea o sumă egală cu S - $xtrsum$, dacă fiecare valoare trebuie să fie cel puțin egală cu 0?

Pentru a rezolva această subproblemă, vom folosi o formulă inspirată din formulele Stars and Bars, astfel încât răspunsul va fi egal cu $C(R - L + 1 + sum - xtrsum - 1, R - L)$. După ce avem grijă să putem aplica formula în acest caz, vom putea calcula combinările folosind precalcularea factorialelor și a inverselor modulare.

Soluția va avea o complexitate de $O((n + q)logn + slogs)$

2 Problema Chess Tournament

Autor: stud. Ioan Popescu, Universitatea Politehnica din București, stud. Ștefan-Cosmin Dăscălescu, Universitatea din București

2.1 Descrierea soluției

Dificultate: Ușoară

Problema se reducea la a afla dacă avem un caracter egal cu la sfârșitul șirului, caz în care trebuia verificat dacă primul, respectiv cel de-al doilea jucător a câștigat. În caz contrar, meciul s-a terminat la egalitate.

3 Problema Martian War

Autor: stud. Bogdan Ioan Popa, Universitatea din București

3.1 Descrierea soluției

3.1.1 Solutie 30 de puncte

Dificultate: Medie

Pentru 30 de puncte se pot afla componentele biconexe după fiecare muchie adăugată, folosind un algoritm de tipul celui pentru aflarea Componentelor biconexe

3.1.2 Solutie 100 de puncte

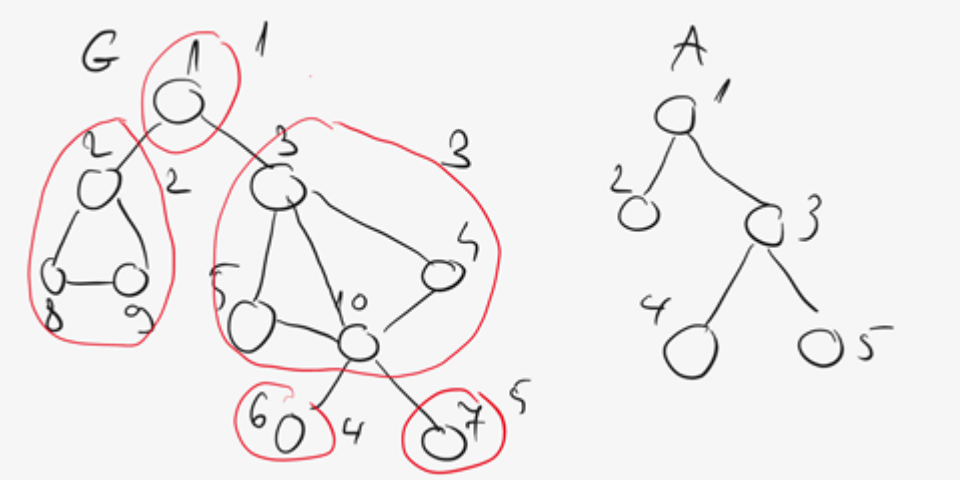
Dificultate: Grea

Vom considera că avem la început un graf cu N noduri, 0 muchii și $M + Q$ operații de adăugare, astfel vom afișa doar rezultatul ultimelor Q operatii. Ne vom referi la linie de cale ferată strategică de acum încolo ca muchie critică. Numim componentă fmc (fără muchii critice) un subgraf maximal care nu conține muchii critice. Considerăm că un graf arbitrar G se poate transforma într-un graf A al componentelor fmc asociat lui prin următorul procedeu:

1) Pentru fiecare componentă fmc din G asociem un nod în A.

2) Dacă avem muchie între un nod dintr-o componentă fmc din G în altă componentă fmc din G vom trage o muchie între nodurile asociate lor în A.

Se observă că A este o pădure de arbori (fiecare componentă conexă a sa este un arbore). Demonstrația este trivială.



În stânga avem graful G, în dreapta avem arborele A. Componentele fmc sunt cele încercuite cu rosu iar nodul asociat lor în A este scris în imediata lor apropiere.

O muchie nou adăugată poate avea următoarele efecte:

a) Unește două noduri din aceeași componentă conexă, dar nu din aceeași componentă fmc, caz în care cel puțin o muchie critică își pierde statutul de muchie critică.

b) Unește două noduri din două componente conexe diferite, caz în care această muchie adăugată este muchie critică.

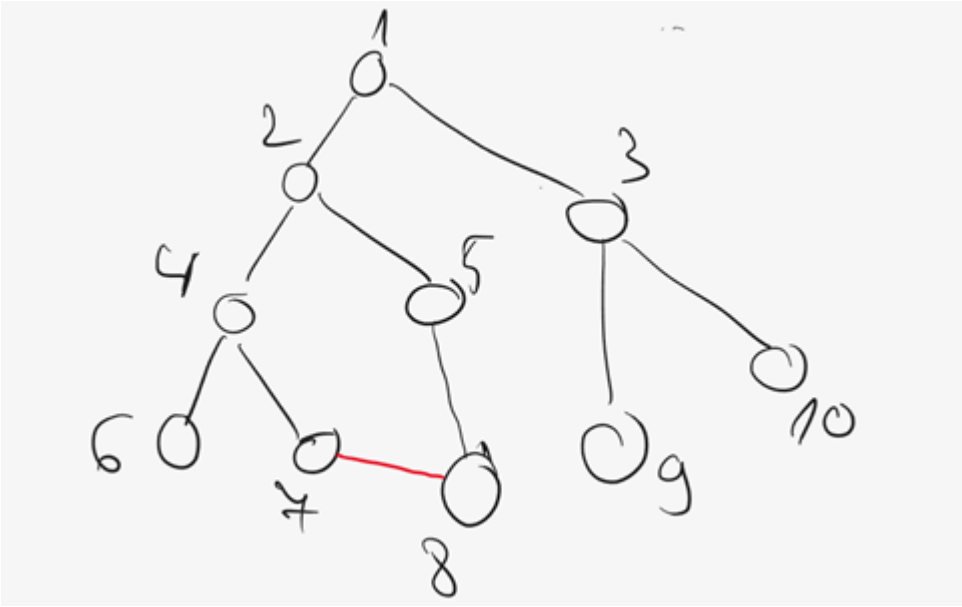
c) Unește două noduri din aceeași componentă fmc, caz în care numărul de muchii critice nu se modifică.

Cazul a)

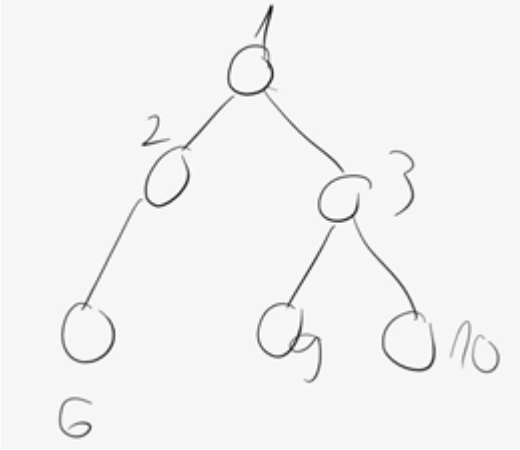
Fie x și y cele două componente fmc care urmează între care s-a tras muchia nouă. Atunci toate muchiile de pe drumul de la x la y în A își vor pierde statutul de muchie critică și toate nodurile de pe acel drum vor deveni o singură componentă fmc. Pentru a determina ușor nodurile de pe drumul de la x la y va fi necesar să ținem o rădăcină pentru fiecare componentă din A.

Având o rădăcină vom putea calcula $LCA(x, y)$ = cel mai apropiat strămoș comun între x și y (lowest common ancestor). Toate nodurile pe drumul de la x la $LCA(x, y)$ și de la y la $LCA(x, y)$ vor deveni o singură componentă fmc. În poza de mai jos dacă se adaugă muchia roșie, atunci componentele fmc 2, 4, 5, 7 și 8 vor deveni o singură componentă fmc.

Înainte



După

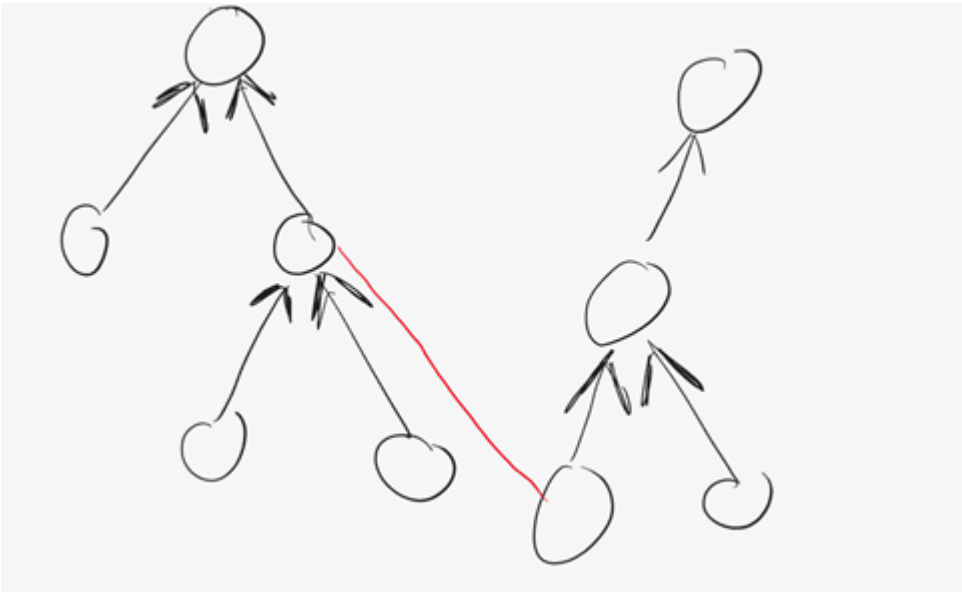


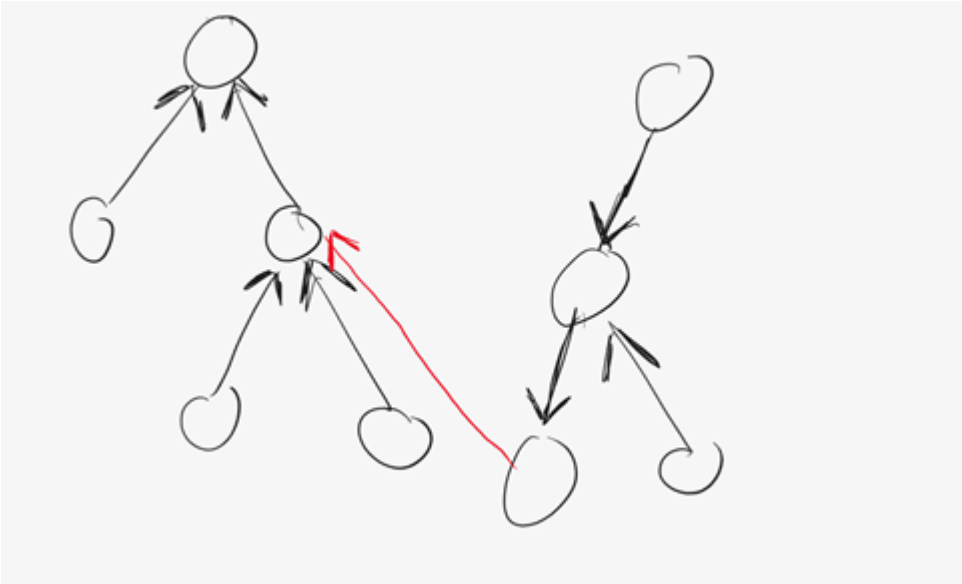
Pentru determinarea LCA-ului se poate folosi algoritmul liniar, având grijă ca atunci când urcăm în arbore să unim nodurile (putem folosi păduri de mulțimi disjuncte). Fiecare muchie este retrogradată maxim o dată, deci complexitatea se amortizează.

Cazul b)

La unirea celor doi arbori trebuie să avem grijă la reorientarea vectorului de tați al arborelui (deoarece arborele va deveni subarborele celuilalt). Avem mai jos un exemplu înainte și după orientare.

Înainte





Vom avea grijă să reorientăm vectorul de tați în arborele de dimensiune mai mică, astfel încât vom obține o complexitate finală $O(N * logN)$.

4 Problema Prime Sums

Autor: stud. Vlad Andrei Andrieș, Universitatea din București

4.1 Descrierea soluției

Dificultate: Ușoară

Pentru a obține punctaje parțiale, se pot folosi diverse soluții bazate pe metoda programării dinamice.

Pentru început, amintim enunțul conjecturii lui Goldbach: "Orice număr par mai mare decat 2 se poate scrie ca sumă de două numere prime. Dacă însumăm 3 la un număr par, obținem un număr impar, deci, cum orice număr par mai mare decat 2 se poate scrie ca sumă de două numere prime, deducem că orice număr mai mare decat 5 se poate scrie ca sumă a trei numere prime. Adunând în mod repetat 2, obținem că orice număr mai mare decât 7 se poate scrie ca suma de patru numere prime ș.a.m.d. Cazul general se demonstrează folosind procedeul inducției matematice.

Astfel, în urma raționamentului inductiv demonstrat anterior, obținem răspunsul ca fiind $n - 2 * k + 1$.

Complexitatea soluției este $O(1)$

5 Problema Another Boring Problem

Autor: stud. Ioan Popescu, Universitatea Politehnica din București

5.1 Descrierea soluției

5.1.1 Solutie 30 de puncte

Dificultate: Medie

Observăm că valorile $b * i^c$ pot deveni extrem de mari, iar lucrul cu numere mari este foarte lent în acest caz. Putem logaritma,
$log(b * i^c) = log(b) + c * log(i)$ fiind o valoare mică, logaritmul putând să îl calculăm chiar cu funcția log din cmath. Astfel, comparăm valorile logaritmate si ținem minte valorile lui b si c, ca să putem afișa răspunsul modulo $10^9 + 7$.

5.1.2 Solutie 100 de puncte

Dificultate: Grea

Prima solutie care este posibilă, se bazează pe observația că, pentru poziții mai mari decât radical de ordin k din n, nu contează decât cele mai mari k valori ale lui c. Daca alegem k favorabil, putem obtine 100 de puncte, rezolvând ca la soluția precedentă pentru primele radical de ordin k din n valori, și cu un arbore de intervale pentru restul valorilor.

O a doua solutie care este mai tehnică, se folosește de Lichao Tree, în cazul acesta funcțiile fiind de tipul $f(x) = a * ln(x) + b$. Update-urile fiind pe interval, nu pe tot vectorul, trebuie folosită o variantă care este descrisă aici: Li Chao Tree Extended

6 Problema Bytes

Autor: stud. Ioan Popescu, Universitatea Politehnica din București

6.1 Descrierea soluției

6.1.1 Solutie 50 de puncte

Dificultate: Medie

Încercam să rezolvăm problema când nu apar variabile negate.Dacă ne uităm la un monom, vedem ce variabile apar în el, și creăm un număr care are pe poziția bitului i valoarea 0 dacă caracterul $('a' + i)$ nu apare in monom, respectiv 1 dacă apare. Practic asociem unei măști nevoia apariției unei variabile ca monomul sa aibă valoarea 1. Observăm că pentru toate supramaștile unui monom, monomul are valoarea 1, deci am putea calcula pentru fiecare mască supramasca ei in $O(2^k)$, solutia având complexitatea $O(2^k * nrmonoame)$, o astfel de soluție obținând 20 de puncte. Pentru 50 de puncte, putem rezolva problema cu SOS DP, această tehnică fiind descrisă aici: SOS DP

6.1.2 Solutie 100 de puncte

Dificultate: Grea

Ca să putem adapta soluțția anterioară, trebuie să construim diferit măștile. O să le construim în felul următor: bitul i o să aibă valoarea 0, dacă litera asociată nu apare, 1 dacă litera apare negată, 2 dacă litera apare normal. Putem face un SOS Dp ca la soluția anterioară, complexitatea fiind $O(k * 3^k)$, fiind prea lentă ca să obțină punctaj maxim. O observație importantă este aceea că biții unei măști nu o să se schimbe din 1 în 2 sau din 2 în 1, ci doar biții egali cu 0 se pot schimba fie în 1, fie în 2.

O altă observație este că pentru fiecare mască, nu are sens să facem tranziții către o supramască, decât dacă schimbăm cel mai nesemnficativ bit, și că este suficient să schimbăm doar acest bit. Deci este suficient să facem o parcurgere a măștilor. O soluție cu aceste observații are complexitatea $O(3^k)$, un factor important fiind modul în care vedem pentru o mască cel mai nesemnficativ bit.

7 Problema UCL

Autor: stud. Ștefan-Cosmin Dăscălescu, Universitatea din București

7.1 Descrierea soluției

Dificultate: Medie

Soluția se bazează pe metoda Backtracking, astfel noi vom genera toate variantele valide posibile de rezultate, respectând condițiile din enunț, având grijă să separăm variantele unice de cele neunice. Pentru ușurința implementării, se pot precalcula toate jocurile între cele n echipe, iar mai apoi putem fixa rezultatul fiecărui joc dintre două echipe A și B (victorie A, egal sau victorie B).

Pentru obținerea punctajului maxim, trebuie precalculate toate variantele valide și memorate apoi într-o structură de tipul map sau set.

Complexitatea soluției va fi $O(4^{(n*(n-1))} * logn)$