Descrieri ale soluțiilor problemelor de la secțiunea echipe

Universitate (echipe, 9-10)

Soluție - O(1) pentru adăugare, O(n) pentru ștergere și afișare în O(nr_studenți)

Putem ține un vector cu studenții. La operațiile de tip adaugă în universitate, adăugăm studentul la finalul vectorului. Pentru ștergeri, putem căuta studentul în O(n) și îl ștergem în O(n). Afișăm vectorul (care poate avea dimensiune maxim 1000) la finalul fiecărui an.

Soluție - O(1) pentru adăugare/ștergere și afișare în O(n)

Putem avea un vector de bool-uri de dimensiune n, unde studenti[i] = se află sau nu studentul “i” în universitate. La adăugare student[id] = 1, la ștergeri student[id] = 0. La afișare trecem prin tot vectorul în O(n) și afisăm pozițiile pentru care studenti[i] == 1.

Solutie - O(log(n)) adăugare/ștergere și afișare în O(nr_studenți)

Putem ține un set de studenți. Inserarea și ștergerea din set va deveni atunci O(log(n)) iar la finalul fiecărui an putem afișa elementele din set în ordine aleatoare (care pot fi maxim 1000)

Sloutie - O(1) adăugare/ștergere și afișare în O(nr_studenți)

1. Unordered_set

Dacă la soluția precedentă schimbăm structura de date folosită de la set la unordered_set atunci complexitatea pe adăugari și ștergeri devine O(1). Se poate folosi și un unordered_map pentru aceasta soluție.

2. Soluția cu 2 vectori

Putem ține doi vectori:
studenti[i] = pe ce poziție se afla studentul “i” în universitate
universitate = vectorul cu studenții.

Atunci avem:

- Pentru adaugare: îl adăugăm pe student la finalul vectorului “universitate” iar studenti[id] = poziția pe care a fost adaugat.
- Pentru ștergere: Aflăm pe ce poziție se afla studentul “id” în universitate cu ajutorul vectorului “studenti”. Facem un swap studentului “id” cu sfârșitul vectorului “universitate” și apoi putem face ștergerea în O(1). Trebuie sa fim atenți să actualizăm și vectorului “studenti” cu noile poziții obținute în urma swap-ului.
- Pentru afișare: afișăm vectorul “universitate”.

Vedere (echipe, 9-10)

Pentru fiecare abscisă **x** se poate determina de la ce **ys[x]** trebuie privit pentru a vedea vârful blocului din stânga. Acest **ys[x]** este o valoare mai mare sau egală cu **h[x]**.

Dintre toate pozițiile **i** de la **2** la **n** ne interesează cea pentru care valoarea **(h[i]-h[1]) / (i-1)** (reprezentând panta dreptei care unește vârful **1** de vârful **i**) este maximă. Acest lucru se poate determina pentru fiecare poziție ca și cum am calcula maximul dintr-un vector. Astfel, noi putem determina, pentru fiecare poziție **i**, înălțimea **ys[i]** de la care putem vedea vârful blocului **1**.

Cu un raționament similar putem determina, pentru fiecare poziție **i**, înălțimea **yd[i]** de la care putem vedea vârful blocului **n**.

Pentru o anumită poziție **i**, putem vedea ambele capete de la înălțimea **max(ys[i], yd[i])**. Așadar alegem minimul acestor valori drept soluție. În funcție de implementare, este posibil ca soluțiile în care punctul este ales la abscisa **1** sau **n** să fie tratate separat.

Complexitate timp: **O(n)**

Conform restricțiilor din enunț pot fi implementate și soluții parțiale:

- punctul căutat se află într-o mulțime restrânsă de posibilități (fie valorile ordonatelor sunt mici, fie știm că se află în vârful unui bloc), așadar nu avem nevoie să implementăm intersecția de drepte, ci doar avem de comparat pante; astfel se pot obține până la 55 de puncte
- nu precalculăm maximele pantelor pe sufix/prefix ci iterăm de fiecare dată de la început, obținând astfel o complexitate pătratică care poate aduce până la 49 de puncte

Aproapeperm (echipe, 11-12)

Solutie 19 puncte:
Se vor genera toate permutarile de lungime N cu backtracking si se vor verifica individual daca respecta conditiile din problema.
Complexitate: O(N! * N) sau O(N!) in functie de optimizari

Solutie 41 puncte:
La fel ca la solutia de 20 de puncte, doar ca procesul de generare se poate optimiza semnificativ daca observam ca pe pozitia i a permutarii putem pune doar una valorile i-1, i sau i+1, in loc sa incercam orice valoare posibila. De asemenea, putem verifica in timp ce generam permutarea daca avem 3 puncte fixe consecutive, si in caz afirmativ sa oprim generarea celorlalte elemente din permutare.
Complexitate: ~O(nr permutari valide)

Solutie 100 puncte:
Fiind o problema de numarare, aceasta ne duce cu gandul la programare dinamica. De asemenea, cum valoarea de pe pozitia i poate sa fie doar o valoare din multimea {i-1, i i+1}, inseamna ca valoarea de pe pozitia i poate sa intre in conflict (sa fie egala) doar cu valorile puse pe pozitiile i-1 sau i-2, deci trebuie sa tinem cont doar de aceste pozitii cand suntem la pasul i.
Mai mult, putem sa encodam valorile puse pe aceste pozitii folosind doar 3 stari, acestea reprezentand ca pentru o pozitie p am pus fie p-1, fie p fie p+1.

Avem asadar urmatoarea dinamica pe stari:
dp[i][s1][s2] (0 <= s1, s2 <= 2) = in cate moduri putem genera prefixe de lungime i valide ale aproape permutarilor, astfel incat valoarea de pe pozitia i-1 este egala cu [(i-1) + (s1-1)] iar valoarea de pe pozitia i este egala cu [i + (s2-1)]

Evident, trebuie sa avem grija sa nu punem o valoare invalida (de exemplu 0 sau N+1) pe o pozitie si sa nu avem 2 valori egale in permutare)

Cand vrem sa generam prefixe de lungime i+1, adunam pe dp[i][s1][s2] la dp[i+1][s2][s3] doar daca noul prefix de lungime i+1 respecta in continuare conditiile, adica nu avem valori egale in permutare, nu am pus o valoare invalida pe pozitia i+1 si tripletul (i-1, i, i+1) nu contine 3 puncte fixe).

Raspunsul va fi suma tuturor valorilor de forma dp[N][s1][s2] cu 0 <= s1,s2 <= 2
Complexitate: O(N * constanta) = O(N)

Faleza (echipe, 11-12)

Solutia 1
Putem folosi o soluție Monte-Carlo pentru a estima aria dreptunghiului și a mării.
Soluția este $\frac{Arie\ uscat}{Aria\ mare}$.
Trebuie luat un număr cât mai mare de puncte, apoi văzut fiecare punct unde se află pe hartă(uscat sau mare respectiv în ce fâșii intervine) si calculat $\frac{Nr\ puncte\ uscat}{Nr\ puncte\ mare}$

Această soluție este foarte aproape de valoarea reală, deoarece nu există erori de calcul pentru numerele reale care să schimbe rezultatul.

O problemă asemănătoare pe care o puteți încerca: https://www.geeksforgeeks.org/estimating-value-pi-using-monte-carlo/

Soluția 2
Altă soluție posibilă reprezintă idea cu ajutorul căreia se calculează integrala Riemann. Trebuie sa împărțim dreptunghiul mare în mai multe mini fâșii și apoi putem găsi un trapez/dreptunghi/altă formă geometrică pe care o putem calcula matematic.
Totuși este de remarcat faptul că aici calculăm ***aria mării***, pentru că e vorba de aria de sub grafic.
Mare atenție la calculul acestei “pseudo integrale”, deoarece funcția poate să nu fie în interiorul dreptunghiului pe anumite porțiuni.
In plus, atenție la faptul că va trebui să scădeți yStart - 1 la lungimea trapezului dacă nu calculați integrala de yStart = 0.

Putem să aflăm și aria uscatului folosind idea că: Arie totala = Arie mare + Arie uscat.

De remarcat: soluția aceasta e posibil să aibă o eroare de calcul mai mare, deoarece erorile de calcul pentru numere reale sunt mai accentuate. Cu toate acestea, ea se va încadra în eroare relativă de 0.05.