Descrieri ale soluțiilor problemelor de la secțiunea liceu

Drag (individual, clasa a 9-a)

Numărul de rânduri care se pot completa (pe care îl notăm cu R) este egal cu numărul total de valori 1 împărțit la numărul de coloane (câtul acestei valori).

Notăm cu Z numărul de valori 0 de pe ultimele R rânduri. Toate pozițiile pe care se află acest elemente 0 trebuie completate.

Alegem cele mai de jos Z valori de 1 de pe rândurile de deasupra celor R și pe acestea le cuplăm cu zerourile care trebuie completate. Practic s-ar putea sa trebuiască mutate și altele, din ultimele R rânduri, dar costul va fi același.

Am putea sorta șirul S1 al valorilor ce reprezintă liniile elementelor de completat, apoi șirul S2 al liniilor ce reprezintă elementele de adus și le combinăm apoi 1 la 1 (adică adunăm la soluție valorile S2[i]-S1[i]).

Color (individual, clasa a 9-a)

Putem codifica fiecare culoare distinctă într-un mod care să permită ca prin amestecarea culorilor să obținem tot valori distincte. De exemplu, la fiecare opreație putem să considerăm drept culoare următorul număr prim nefolosit și să înmulțim valoarea lui în fiecare celulă a matricei.
Din restricții deducem că aceste valori care se pot obține se încadrează pe tipul long long (de exemplu al 150-lea număr prim este de cel mult 3 cifre).
Așadar putem aplica operațiile brut iar la final să vedem câte valori distincte avem în matrice.
Un mod basic de a afla acest lucru este să punem toate elementele matricei într-un vector, să îl sortăm și apoi să determinăm numărul de secvențe maximale cu elemente egale.

Pathinter (individual, clasa a 9-a)

Soluție 40 de puncte

Vom parcurge liniile formate din traseul lui Radu. Pentru fiecare dintre acestea, putem parcurge toate liniile din traseul lui Petru pentru a număra câte se intersectează cu linia curentă din traseul lui Radu. Complexitatea acestei soluții va fi O(n^2).
Două linii (x1, x2) și (y1, y2), unde x1 și x2 reprezintă punctele de pe bordura de jos, iar y1 și y2 cele de pe bordura de sus, se vor intersecta dacă se îndeplinește una din următoarele condiții:

- x1 < y1 și x2 > y2
- x1 > y1 și x2 < y2

Soluție 100 de puncte

Observația necesară pentru a optimiza soluția anterioară este aceea că fiecare linie din traseul lui Radu va intersecta o subsecvență continuă de linii din traseul lui Petru. Astfel, pentru o linie din traseul lui Radu putem folosi două căutări binare pentru a determina prima, respectiv ultima linie din traseul lui Petru cu care aceasta se intersectează. Complexitatea obținută va fi O(n log n).
Putem optimiza și mai mult această soluție folosind tehnica two pointers în locul căutărilor binare. Vom ține 2 pointeri [l, r] care reprezintă intervalul din traseul lui Petru pe care îl intersectează linia curentă a lui Radu. Când trecem la următoarea linie, vom incrementa corespunzător l și r pentru a acoperi noul interval intersectat. Astfel, am obține complexitate O(n) însă aceasta nu este necesară pentru punctajul maxim.

Towers (individual, clasa a 10-a)

La fiecare interogare între pozițiile p și p+1 putem itera înapoi începând cu poziția p, determinând maximul elementelor întâlnite. De fiecare dată când se schimbă maximul avem un element care se vede și vom contoriza aceste momente. Complexitatea este de ordinul (numar de interogări) X (lungimea șirului dat) și nu este suficientă pentru toate testele.

Pentru optimizare putem folosi o stivă pe care o actualizăm pe măsură ce vizităm elementele șirului dat, de la stânga la dreapta. La un moment dat avem în stivă doar elementele care mai pot fi văzute privind dinspre dreapta. Elementul curent elimină așadar din stivă elementele mai mici sau egale cu el aflate în vârf, apoi se așează și el în stivă. În acest moment, numărul de elemente din stivă este chiar valoarea cu care vom răspunde la o interogare în această poziție, deci o putem memora și apoi răspundem la fiecare interogare în timp constant. Acum avem complexitate de ordinul: (numar de interogări) + (lungimea șirului dat).

Editsmart (individual, clasa a 10-a)

Subtask 1 (24 de puncte) - N <= 100

Soluția brută folosește **o coadă** pentru a găsi drumul minim de la poziția de start la poziția de final. Tratăm fiecare posibilitate de mișcare în mod direct, la fiecare pas din parcurgere:

- Săgețile normale ne permit să ne deplasăm o poziție în oricare din direcțiile sus, jos, stânga sau dreapta.
- Home și End ne deplasează direct la începutul sau sfârșitul rândului, având costul 1.
- Ctrl + Stânga: trebuie să găsim începutul cuvântului anterior de pe același rând. Dacă în poziția curentă suntem pe un caracter, vom parcurge rândul curent spre stânga până ajungem la începutul cuvântului curent, apoi continuăm să ne deplasăm spre stânga până ajungem la începutul cuvântului anterior.
- Ctrl + Dreapta: ne deplasăm spre dreapta până ajungem la începutul cuvântului următor.

Complexitate: Parcurgerea are complexitate O(n^2), Operațiile Ctrl + Stânga / Dreapta în parcurgerii au complexitate O(n). In total, avem complexitatea **O(n^3)**.

Subtask 2 (12 puncte) - textul nu contine spații

Dacă nu avem spații, avem mai multe variante de a forma un traseu de la poziția de start la poziția de final. Mai întâi, ne deplasăm cu săgeata sus / jos până ajungem pe rândul poziției de final. Numărul de operații fiind |x_start - x_final|. Acum suntem în poziția (x_final, y_start). Pentru a ajunge pe coloana y_final, avem mai multe posibilități:

- Ne deplasăm cu săgețile stânga / dreapta până ajungem la coloana poziției de final. Număr de operații: |y_start - y_final|
- Aplicăm operația Home, apoi mergem cu săgeata dreapta până ajungem la coloana y_final. Număr de operații: y_start
- Aplicăm operația End, apoi mergem cu săgeata stânga până ajungem la coloana y_final. Număr de operații: 1 + n - y_end

În final, soluția optimă este minimul dintre variantele de deplasare menționate anterior.
Complexitate: O(1)

Subtask 3 (8 puncte) - poziția de început și final pe aceeași linie

Dacă linia de început și de final sunt egale, coada pe matrice se reduce la coadă pe un singur rând, deoarece nu trebuie să explorăm și alte zone ale matricei. Mutările posibile vor fi:: Săgeata stânga sau dreapta, Home, End, Ctrl + Săgeată stânga / dreapta.
Complexitate: O(n^2).

Subtask 4 (5 puncte) - poziția de început și final pe aceeași coloană

Dacă poziția de început și poziția de final sunt pe aceeași coloană, nu avem nevoie de mutări pe orizontală. Ne deplasăm doar vertical cu săgeata sus / jos până ajungem la destinație. Numărul de operații va fi |x_start - x_end|.
Complexitate: O(1)

Subtask 5 (51 de puncte) - N <= 1000

Pentru soluția completă, optimizăm algoritmul astfel încât să nu mai parcurgem linia pentru a determina unde să ne deplasăm cu Ctrl + Stânga / Dreapta. Vom face acest lucru prin două precalculări.

Left[i][j] = coloana primului caracter al cuvântului aflat în stânga poziției (i, j), pe linia i. Dacă (i, j) se află deja la începutul unui cuvânt, atunci Left[i][j] este poziția acestuia.

Right[i][j] = Right[i][j] reține coloana primului caracter al următorului cuvânt de pe linia i, situat la dreapta poziției (i, j). Dacă nu există un astfel de cuvânt, marcăm cu -1 și vom avea grija sa nu ne deplasam.

Complexitate: O(n^2), deoarece deplasarea Ctrl + Săgeată stânga/dreapta se face in timp constant.

Quick (individual, clasa a 10-a)

Dacă generăm toate permutările de lungime n și aplicăm implementarea din enunț, căutând permutarea care generează numărul minim de incrementări, putem obține 15 puncte.

Observăm mai întâi că funcția pivot aplicată pe un interval alege primul element din interval (notăm valoarea X) apoi plasează toate valorile mai mici decât X din interval în fața lui X și plasează toate valorile mai mari după X, păstrând ordinea relativă.

Notăm cu cnt_min[i] numărul minim de incrementări ale lui cnt pentru o permutare de lungime i. Dacă prima valoare din permutare este X, la primul apel al funcției pivot se realizeaza i-1 incrementări, apoi funcția este apelată recursiv pe intervale de lungime X-1 și i-X. Întrucât aceste intervale au elemente distincte le putem considera (dupa o re-etichetare) echivalente cu niște permutări.

Astfel obținem următoarea recurență:

- cnt_min[1] = 0
- cnt_min[i] = min(i - 1 + cnt_min[X - 1] + cnt_min[i - X] | 1 <= X <= i)
- mai reținem și best[i] cel mai mic X pentru care a fost obținută valoarea din cnt_min[i]
- pentru reconstituire punem best[n] pe prima poziție, apoi reconstituim recursiv cele mai bune permutări de lungime best[n] - 1 si n - best[n]; pentru a obține permutarea minim lexicografică prima permutare conține doar valorile mai mici decât best[n], iar a doua le conține pe cele mai mari (pentru aceasta avem nevoie de o re-etichetare).

Această soluție are complexitate O(n^2) și obține aproximativ 65 de puncte.

Pentru O(n) putem observa urmatorul pattern in best[i]:

- 1, 1, 2, 2, 2, 3, 4, 4, 4, 4, 4, 5, 6, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 10, 11, …
- Pe scurt șirul este format din numere consecutive, iar puterile lui 2 (2^k) apar de (2^k + 1) ori.
- Putem precalcula acest șir pana la lungimea n, apoi sa reconstituim solutia.

Această soluție obține 100 de puncte.

Antena (individual, clasele a 11-a și a 12-a)

OBS: Toate adunările și calculele se vor face modulo 10^9 + 7.

Soluție O(N^3)
Observația principală pentru a rezolva problema este că pentru o secvență de la i la j (de pe *“linia”* din arbore pe care o formează nodurile de la 1 la N) pe care vrem să o considerăm ca subarbore conex trebuie doar să decidem în cate moduri putem păstra nodurile laterale.

Astfel că dacă pentru un subarbore conex cu noduri de la i la j de pe linia principală a antenei pentru care avem în total **L** noduri laterale legate de la oricare nod de la i la j, atunci vom avea 2^L moduri de a selecta nodurile laterale pe care trebuie să le numărăm la soluție, deoarece pentru fiecare nod lateral avem 2 opțiuni:

- Îl păstrăm în subarbore
- Nu îl păstrăm în subarbore

Pentru soluția de O (N^3) fixăm fiecare subsecventa de noduri de la i la j în N^2, după care parcurgem secvența pentru a calcula numărul total de vecini laterali L și adunăm la numărul total de arbori conecși 2^L**.**
La final mai trebuie să adunăm la răspuns și numărul total de noduri laterale ale arborelui original/întreg, care este echivalentul cu a considera pe rând ca subarbore fiecare nod lateral ca nod izolat.

Soluție O(N^2)
Similar cu soluția anterioară, fixăm fiecare subsecvență de la i la j, doar că de această dată vom precalcula în timp liniar un vector de frecvență f[i] = numărul de vecini laterali pentru subsecvența de la 1 la i de pe linia principală a antenei. Astfel că, L-ul subsecvenței de la i la j va fi egal cu L = f[j] - f[i - 1], reducând complexitatea la O(N^2), iar la final adunăm numărul total de vecini laterali.

Soluție O(N)
Plecând de la observația principală pentru rezolvarea problemei, putem să calculăm numărul de subarbori liniar dacă folosim următoarea dinamică:
D[i] = numărul de subarbori al caror lanț principal din antenă se termină cu i (nu mai conține niciun nod la dreapta lui i)

Unde vom avea următoarea recurență:
D[1] = 2 ^ (numărul de vecini laterali ai lui 1)
D[i] = (D[i -1] + 1) * 2^(numărul de vecini laterali ai lui i)

Ideea din spatele recurenței de bază este că pentru nodul i, avem 2 posibilități:

- Alegem să îl legam de i-1 în D[i - 1] modrui

- Alegem să nu îl conectăm de i-1, și pornim un nou subarbore cu i cel mai din stânga nod într-un singur fel (1)
- Pentru ambele cazuri avem exact 2^(numărul de vecini laterali ai lui i) moduri în care putem considera vecinii laterali ai lui i, fără a strica proprietatea de conexitate a subarborelui

Rezultatul final va fi:
D[1] + D[2] + … + D[n] + numărul total de vecini laterali (din același motiv prezentat anterior)

Soluție caz particular nv = 0

Pentru acest caz, se garantează că nu există niciun vecin lateral în arborele antena din input, cea ce însemnă că arborele nostru este de fapt un lanț cu n noduri, astfel problema reducându-se la a calcula numărul total de subsecvențe al unui șir de lungime n, care este egal cu:

$$n * (n - 1) / 2$$

Trenuri (individual, clasele a 11-a și a 12-a)

Se observă că două trenuri se vor intersecta dacă ordinea în care ajung la cel mai apropiat strămoș comun al nodurilor de plecare este diferită de ordinea în care ajung în nodul **1**.

Se calculează strămoșii fiecărui nod: **P[p][i] = strămoșul aflat la distanța 2^p de nodul i**.

Pentru o interogare, se calculează cel mai apropiat strămoș comun (**lca**) în **O(log n)** folosind precalcularea strămoșilor. Apoi, pe drumul de la **lca** la **1**, se folosesc din nou strămoșii pentru a căuta cel mai apropiat nod **x** de rădăcină, cu proprietatea că trenurile ajung la **x** în aceeași ordine în care au ajuns la **lca**. Dacă trenurile se vor intersecta, o vor face pe muchia dintre **x** și părintele său.

Trebuie tratate cazurile în care trenurile se intersectează exact în **lca** sau în părintele lui **x**.

Pentru a calcula ordinea în care trenurile ajung într-un nod în **O(1)**, se vor precalcula sume parțiale **dist[i] = distanța dintre nodul i și nodul 1**. Astfel, putem calcula timpul în care un tren ajunge la un nod strămoș, împărțind distanța dintre cele două trenuri la viteza trenului.

Complexitate - timp: **O(n log n + m log n)**, memorie: **O(n log n)**.

Lencycles (individual, clasele a 11-a și a 12-a)

Solutie 18 puncte:
Generam cu backtracking toate permutarile de lungime N si apoi verificam daca acestea au exact K cicli de lungimi egale cu lungimile date, asa cum se cere in enunt
Complexitate: O(N! * N)

Solutie subtask 2 (42 puncte):
Fie lungimile ciclilor egale cu p1, p2, … pk.
Pentru a numara permutarile cerute, putem calcula mai intai in cate moduri putem selecta pozitile elementelor primului ciclu de lungime p1. Observam ca avem C(N, p1) moduri de a selecta aceste p1 elemente din cele N. (Am folosit notatia C(a, b) = combinari de a luate cate b).
Apoi, pentru a selecta pozitile celui de-al 2-lea ciclu, avem C(N-p1, p2) moduri (practic am ‘rezervat’ p1 elemente pentru primul ciclu, iar din restul de N-p1 selectam p2 elemente pentru al 2-lea ciclu).
Pentru al 3-lea ciclu avem asadar C(N-p1-p2, p3) moduri si asa mai departe.
O data selectate pozitiile elementelor ciclilor, trebuie sa vedem in cate moduri le putem permuta intre ele in asa fel incat acestea sa formeze un ciclu.
Vrem deci sa calculam dp[n] = nr de moduri de a forma o permutare care este un ciclu de lungime n.
Subproblema este echivalenta cu a numara cate siruri de forma : 1 => v2 => v3 => … => vn exista, cu v2..n distincte din multimea {2, 3, .. n}. Cu alte cuvinte, putem reprezenta ciclii ca siruri de forma de mai sus, cu semnificatia: 1 se afla pe pozitia v2, v2 se afla pe pozitia v3, …, vn-1 se afla pe pozitia vn, vn se afla pe pozitia v1.
Observam ca nu exista vreo restrictie asupra ordinii valorilor v2..n, deci avem in total (n-1)! moduri de a forma un ciclu de lungime n.

Solutia problemei este asadar: C(N, p1) * (p1-1)! * C(N-p1, p2) * (p2-1)! * … * C(N-p1-...-p(k-1), pk) * (pk - 1)!

Putem precalcula toate combinarile in N^2 folosind recurenta C(a, b) = C(a-1, b) + C(a-1, b-1), complexitatea finala fiind O(N^2 + N) = O(N^2).

Solutie subtask 2 si 3 (63 puncte in total):
La fel ca la subtaskul anterior, doar ca putem calcula combinarile folosindu-ne de formula de la invers modular astfel:

C(a, b) = a! / [b! * (a-b)!] = fact[a] / fact[b] / fact[a-b] = fact[a] * fact[b]^-1 * fact[a-b]^(-1)
Unde fact[n] = n!

Putem asadar sa precalculam toate factorialele, si inversele lor modulare, reducand complexitatea la O(N * logN) daca vom calcula manual pentru fiecare n, fact[n]^(-1) in logN.
Alternativ, putem reduce complexitatea la O(N) observand urmatoarea relatie:
Fie inv[n] = fact[n]^(-1) = [(n-1)! * n]^(-1) = [(n-1)!]^(-1) * n^(-1) = inv[n-1] * n^(-1)
inv[n] = inv[n-1] * n^(-1) => inv[n-1] = inv[n] * n;

Putem asadar sa calculam doar inv[n] folosind ridicare la putere in timp logaritmic, si apoi sa calculam restul inverselor factorialelor folosind recurenta de mai sus.
Complexitate: O(N * logN) sau O(N + logN) = O(N).

Observatie: putem implementa backtracking pentru primul subtask + solutia precedenta pentru a obtine 81 de puncte.

Solutie 100 puncte:
Problema care apare la solutia precedenta este atunci cand avem cicli de lungimi egale, nu putem sa ii distingem intre ei. Cu rationamentul precedent selectam pozitiile elementelor primului ciclu si apoi pozitile celuilalt ciclu. Dar numaram de 2 ori deoarece am fi putut sa selectam pentru primul ciclu pozitiile din al 2-lea ciclu si vice versa. Exemplu: n = 6, k = 2, p1 = 3, p2 = 3
Avem 2 asignari de pozitii posibile astfel (1 inseamna ca pozitia a fost asignata primului ciclu, la fel si pentru 2:
1 2 1 1 2 2 vs 2 1 2 2 1 2
Trebuie deci sa eliminam aceste duplicate. Observam ca daca avem x cicli de o anumita lungime L, avem x! moduri de a permuta ordinea in care asignam pozitiile acestor cicli.
Trebuie asadar sa tinem minte pentru fiecare lungime L, cati cicli de lungime L avem, si sa impartim raspunsul de la solutia precedenta la L! pentru fiecare lungime L posibila.
Complexitate: O(N) sau O(N * logN)