Editorial FMI NO STRESS 12

The Power of Pizza

Propunători: Giulian Buzatu, Dumitru Ilie
Developer: Dumitru Ilie
Editorialist: Dumitru Ilie
First solve: mihai145 după 0:02:02

Trebuie să se calculeze

$$\sum_{i=1}^{N} (h_{2_i} - h_{1_i}) \cdot (1 + pizza_i)$$

Complexitatea timp este $O(N)$, iar complexitatea spațiu este $O(1)$.

Light bulbs

Propunători: Giulian Buzatu, Dumitru Ilie
Developer: Dumitru Ilie
Editorialist: Dumitru Ilie
First solve: Gheal după 0:03:11

Obs.: Dacă primul bec este aprins, primul întrerupător trebuie apăsat, deoarece acesta este singurul care va stinge acest bec.

Putem astfel obtine o soluție pătratică, iterăm din stânga în dreapta și, de fiecare dată când găsim un bec aprins, apăsăm întrerupătorul corespunzător și le schimbăm pe toate celelalte din dreapta.

Pentru a optimiza soluția este necesar să vedem că dacă facem două operații, i și j $(i < j)$, atunci acestea vor avea efectul combinat de a schimba starea becurilor $i, i+1, i+2, \ldots, j-1$. Putem astfel obține secvențele contigue de 1 și pe baza acestora determina unic întrerupătoarele pe care trebuie să le apăsăm.

Complexitatea timp este $O(N)$, iar complexitatea spațiu este $O(N)$.

San Francisco

Propunător: Ilie-Danie Apostol
Developer: Ilie-Danie Apostol, Vlad-Mihai Bogdan
Editorialist: Dumitru Ilie
First solve: andrei_C1 după 0:28:41

Problema ne cere să aflăm pentru fiecare muchie dintr-un multigraf ponderat câte drumuri minime dintre nodurile 1 și 2 folosesc acea muchie. La sfârșit se dorește afișarea sumei acestor numere.

Pentru început vom folosi un algoritm similar algoritmului lui Dijkstra, care va calcula pentru fiecare nod distanța minimă de la un nod sursă până la acesta și numărul de drumuri minime de la nodul sursă la acesta. Vom aplica acest algoritm atât pentru nodul 1, cât și pentru nodul 2.

Acum, pentru fiecare muchie, vom lua cele două capete, u și v. Dacă distanța de la nodul 1 la u + distanța de la nodul 2 la v + costul muchiei (u, v) este egală cu distanța de la nodul 1 la nodul 2, atunci muchia face parte din drumul minim. Numărul de drumuri este egal cu produsul numărului de drumuri de la 1 la u și numărului de drumuri de la 2 la v. Trebuie considerat și cazul celălalt, $1 - v - u - 2$, care este analog.

Rămâne să facem doar suma acestor numere. Complextitatea timp este $O(M \cdot \log_2 N)$, iar compexitatea spațiu este $O(N + M)$.

Rachete

Propunător: Ștefan-Alexandru Popescu
Developer: Ștefan-Alexandru Popescu
Editorialist: Vlad-Mihai Bogdan
First solve: VSebastian8 după 1:00:12

Vom considera o rețea de flux, în care nodul S este nod sursă, iar nodul D este nod destinație. Vom avea N_1 noduri, al i-lea din ele fiind reprezentativ pentru al i-lea motor, alte N_3 noduri, al i-lea din ele fiind reprezentativ pentru al i-lea echipaj. În ceea ce privește capsulele, va fi nevoie să avem $2N_2$ noduri. Al i-lea astfel de nod va fi nodul de intrare reprezentativ capsulei i, iar al $i+N_2$-lea astfel de nod va fi nodul de ieșire reprezentativ capsulei i.

Pentru început, vom lega sursa de nodurile de tip motor, unde capacitatea unei muchii (dintre S și nodul i reprezentativ motoarelor) este numărul de motoare de tipul i puse la dispoziție. Analog, vom lega nodurile de tip echipaj de nodul destinație. De asemenea, vom lega nodurile de tip capsulă de intrare de nodurile de tip capsulă de ieșire, unde legătura dintre nodul i și $i + N_2$ este egală cu numărul de capsule de tipul i puse la dispoziție.

Pentru restricțiile de tip (motor, capsulă), vom conecta nodurile de tip motor cu nodurile de intrare ale capsulelor, iar pentru restricțiile de tip (capsulă, echipaj), vom conecta nodurile de tip capsulă de ieșire cu nodurile de tip echipaj.

Pentru a obține punctaj maxim este nevoie să implementați cu grijă un algoritm eficient de flux. În practică, o implementare în $O(N^2 \cdot M)$ a algoritmului lui Dinic ar trebui să obțină punctaj maxim fără probleme. Complexitatea spațiu este $O(N + M)$.

Personality Test

Propunător: Marius-Stelian Brabete
Developer: Marius-Stelian Brabete
Editorialist: Vlad-Mihai Bogdan
First solve: Gheal după 0:12:49

Problema este în mare una de implementare. Trebuie tratate cazurile de egalitate cu atenție. Punctajele pot fi diferite în funcție de cât de eficient se face factorizarea numerelor. Important este doar ca factorizarea să se realizeze în cel mult $O(\sqrt{x})$, pentru un număr x oarecare.

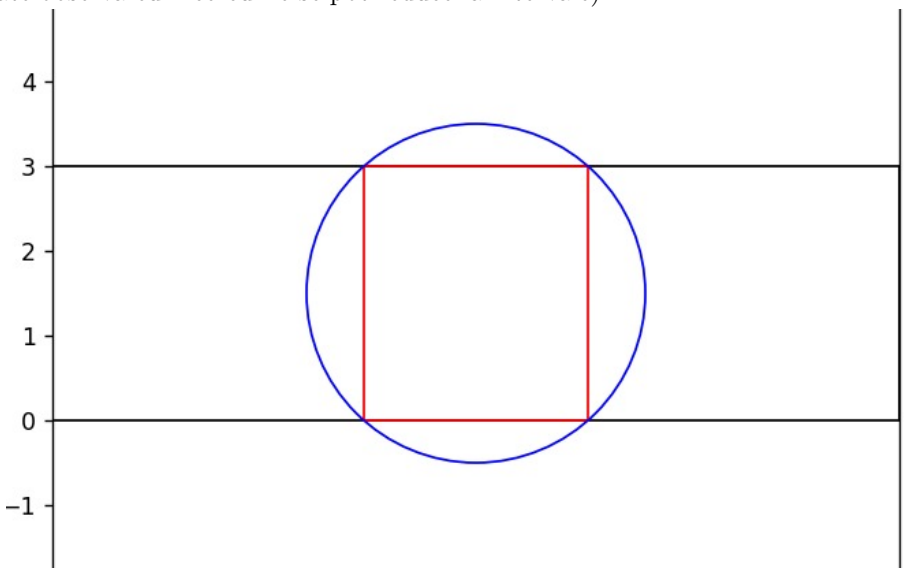
Complexitatea timp este $O(N \cdot \sqrt{max\_value})$, iar cea spațiu este $O(1)$.

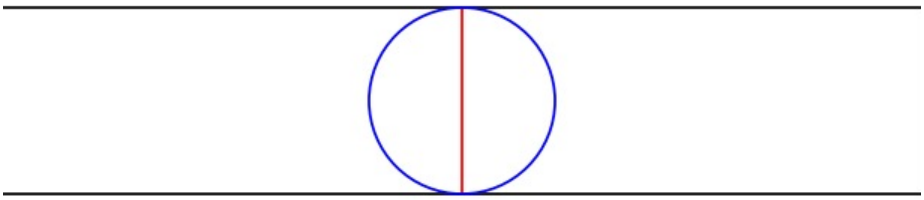
Aspersoare

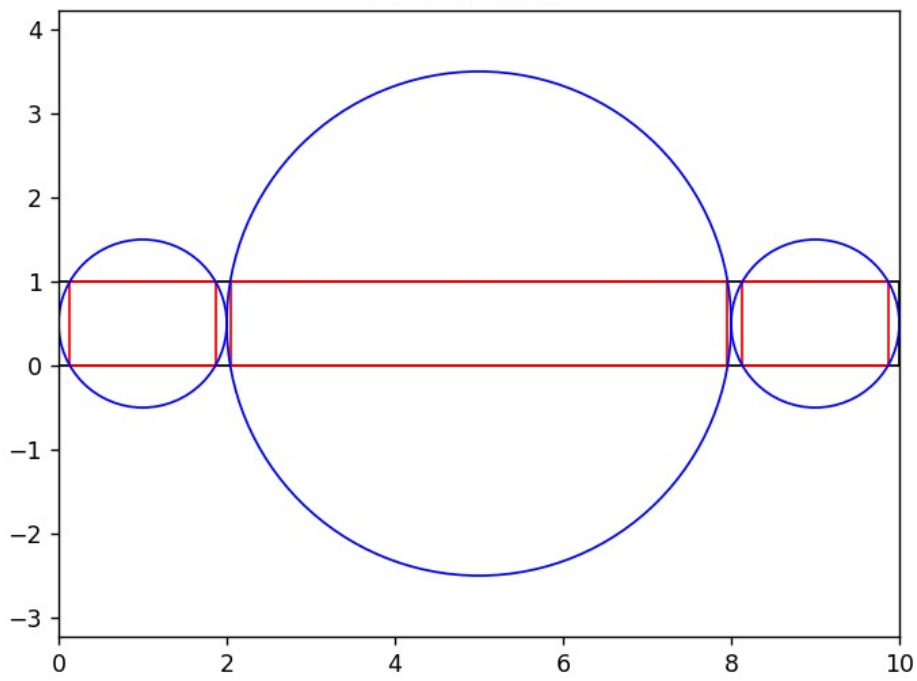
Propunător: Marius-Stelian Brabete
Developer: Marius-Stelian Brabete
Editorialist: Vlad-Mihai Bogdan
First solve: Ricardo03 după 3:46:24

Putem observa că pentru un cerc oarecare este de interes doar unde se intersectează cu dreptunghiul. Să numim coordonatele x unde un cerc C se intersectează cu dreptunghiul x_start și x_end. Pentru ca un alt cerc C_2 să poată fi parte din soluție și să urmeze imediat înainte de C în aceasta, este nevoie ca punctul de intersecție din partea dreaptă a cercului C_2 cu dreptunghiul să fie $\geq x\_start$.

Prin urmare, după ce determinăm punctele de intersecție cu dreptunghiul pentru fiecare dintre cele N cercuri, problema se reduce la a găsi un număr minim de intervale care să acopere intervalul $[0, L]$. (Pe următoarele desene se poate observa cum cercurile se pot reduce la intervale)







De aici, problema se poate rezolva cu un algoritm greedy, folosind următoarea observație: dacă trebuie să găsim un interval care să acopere punctul l, este mereu optim să alegem un interval care are capătul stânga mai mic decât l și capătul dreapta maxim. Demonstrația corectitudinii rămâne ca exercițiu pentru cititor.

Complexitatea timp este $O(N \cdot logN)$, iar cea spațiu este $O(N)$. Există și soluții care se folosesc de programare dinamică și au aceeași complexitate timp și spațiu, dar algoritmul greedy este mai ușor și mai scurt de implementat.

insert1insert2

Propunător: Sebastian Popa
Developer: Sebastian Popa
Editorialist: Vlad-Mihai Bogdan
First solve: alexdumitru după 2:11:30

Pentru început, vom procesa operațiile în ordine inversă. Astfel, putem

determina pozițiile reale ale elementelor, în felul următor: vom ține un șir, inițial plin cu valoarea 1. Când vrem să aflăm o poziție reală (poziția relativă la query-uri era *pos*), tot ce trebuie să facem este să găsim primul indice, l, din șir, cu proprietatea că suma valorilor din șir de pe prefixul de lungime l este egală cu *pos*. După aceasta, vom pune valoarea 0 pe poziția l în șir.

După ce am aflat pozițiile reale ale operațiilor, trebuie doar să trecem prin operații (de această dată, în ordinea în care apar în input) și să folosim o structură de date care poate suporta update pe o poziție și sumă pe interval.

Un arbore de intervale sau un arbore indexat binar poate fi folosit să rezolve ambele părți ale problemei în $O(logQ)$ per operație.

Complexitatea timp este $O(Q \cdot logQ)$, iar cea spațiu este $O(Q)$. Există soluții alternative cu aceleași complexități timp, dar folosesc structuri de date mai complexe (de exemplu, treap-uri).

Hipocamp

Propunător: Vlad-Mihai Bogdan
Developer: Vlad-Mihai Bogdan
Editorialist: Vlad-Mihai Bogdan
First solve: Dap după 4:42:05

Vom spune că o permutare are k maxime dacă își schimbă maximul pe prefix de k ori. Pentru început, vrem să calculăm $count_{n,k}$ = numărul de permutări ale mulțimii $\{1, 2, \ldots, n\}$ cu k maxime.

Încercăm să trecem de la permutările mulțimii $\{1, 2, \ldots, n\}$ la cele ale mulțimii $\{1, 2, \ldots, n+1\}$. Pentru o permutare a mulțimii $\{1, 2, \ldots, n\}$, creștem fiecare element al său cu 1 și obținem o permutare a mulțimii $\{2, 3, \ldots, n+1\}$. La această mulțime trebuie să adăugăm elementul 1. Avantajul este că 1 poate să crească numărul de maxime doar în cazul în care se află pe prima poziție în permutare. Pentru restul de n poziții, 1 nu va contribui cu nimic la numărul de maxime pe prefix.

Prin urmare, $count_{n,k} = count_{n-1,k-1} + (n-1) \cdot count_{n-1,k}$. Recurența de mai sus este aceeași ca și recurența pentru numerele lui Stirling de speța I. De altfel, există o bijecție destul de simplă între numărul de permutări ale mulțimii $\{1, 2, \ldots, N\}$ cu K ciclii și numărul de permutări ale aceleiași mulțimi care au K maxime. Ideea generală este să rotim fiecare ciclu astfel încât maximul este primul. Detaliile acestei bijecții sunt lăsate ca exercițiu pentru cititor.

Trebuie, așadar, să calculăm în mod eficient $s(N, 0), s(N, 1), \ldots, s(N, K)$. Numerele lui Stirling sunt, de fapt, coeficienții binomiali ai dezvoltării $x(x-1)(x-2)\ldots(x-n+1)$. Cum în acest caz nu avem nevoie de numerele lui Stirling cu semn, putem să schimbăm din minus în plus în interiorul parantezelor. Înmulțirea a două polinoame de gradul N se poate face în timp $O(N \cdot logN)$, folosind FFT. Pentru punctaj complet, va trebui să înmulțim polinoamele într-o ordine mai eficientă, așa că vom folosi Divide & Conquer.

Soluția are complexitate timp $O(N \cdot log^2 N)$ și complexitatea spațiu $O(N)$.

3D Sums

Propunători: Giulian Buzatu, Dumitru Ilie
Developer: Dumitru Ilie
Editorialist: Dumitru Ilie
First solve: lbadea1000 după 1:06:06

Problema ne cere să facem niște actualizări într-un spațiu 3D, urmate de niște calcule de sumă pe anumite subparalelipipede. Pentru a rezolva eficient problema vom folosi conceptul de difference array (cunoscut în România drept Șmenul lui Mars), aplicat în 3 dimensiuni. Să presupunem că vrem să adăugăm valoarea v în paralelipipedul determinat de coordonatele (x_1, y_1, z_1) și (x_2, y_2, z_2). Vom face următoarele operații:

$$\begin{aligned}
a[x_1][y_1][z_1] &+= v \\
a[x_1][y_1][z_2+1] &-= v \\
a[x_1][y_2+1][z_1] &-= v \\
a[x_1][y_2+1][z_2+1] &+= v \\
a[x_2+1][y_1][z_1] &-= v \\
a[x_2+1][y_1][z_2+1] &+= v \\
a[x_2+1][y_2+1][z_1] &+= v \\
a[x_2+1][y_2+1][z_2+1] &-= v
\end{aligned}$$

Încurajăm cititorul să deseneze un paralelipiped dreptunghic și să observe cum arată aceste coordonate.

După procesarea tuturor operațiilor de actualizare vom rula următoarele operații, care vor calcula pentru fiecare poziție din paralelipiped care este valoarea finală.

$$\begin{aligned}
a[i][j][k] += {} & a[i-1][j][k] \\
& + a[i][j-1][k] \\
& + a[i][j][k-1] \\
& - a[i-1][j-1][k] \\
& - a[i-1][j][k-1] \\
& - a[i][j-1][k-1] \\
& + a[i-1][j-1][k-1]
\end{aligned}$$

Pentru a răspunde rapid întrebărilor de tip sumă vom folosi conceptul de sume parțiale, dar extinse la 3 dimensiuni. Pentru a le calcula vom folosi din nou calculul precedent.

Răspunsul pentru paralelipipedul dreptunghic determinat de coordonatele (x_1, y_1, z_1) și (x_2, y_2, z_2) este calculat după următoarea formulă:

$$
\begin{aligned}
&+ a[x_2][y_2][z_2] \\
&- a[x_2][y_2][z_1 - 1] \\
&- a[x_2][y_1 - 1][z_2] \\
&+ a[x_2][y_1 - 1][z_1 - 1] \\
&- a[x_1 - 1][y_2][z_2] \\
&+ a[x_1 - 1][y_2][z_1 - 1] \\
&+ a[x_1 - 1][y_1 - 1][z_2] \\
&- a[x_1 - 1][y_1 - 1][z_1 - 1]
\end{aligned}
$$

Complexitatea timp este $O(L \cdot l \cdot H)$, iar complexitatea spațiu este $O(L \cdot l \cdot H)$. Avem un factor adițional de 8. Pe cazul general, cu D dimensiuni, avem un factor adițional de 2^D.

Hemodinamică

Propunător: Vlad-Mihai Bogdan
Developer: Vlad-Mihai Bogdan
Editorialist: Vlad-Mihai Bogdan
First solve: Razvan48 după 0:31:00

Mereu vom compara $A_{i,j}$ cu $B_{i,j}$. Deci, dacă $A_{i,j} = B_{i,j}$, atunci celula (i, j) nu va contribui deloc la sumă. Dacă $A_{i,j} \neq B_{i,j}$, atunci celula (i, j) va contribui cu 1 la toate submatricele care o conțin.

Complexitatea timp și spațiu este $O(N \cdot M)$.

Antisense

Propunător: Vlad-Mihai Bogdan
Developer: Vlad-Mihai Bogdan
Editorialist: Vlad-Mihai Bogdan
First solve: robert.barbu27 după 0:02:26

Vom construi un nou șir, C, unde $C_i = A_i - B_i$. Acum, răspunsul este egal cu subsecvența contiguă din șirul C care are suma maximă.

Orice algoritm cu complexitate timp $O(N)$ sau $O(N \cdot logN)$ ar trebui să obțină punctaj maxim. Complexitatea spațiu este $O(N)$.

Reactor Nuclear

Propunător: Dumitru Ilie
Developer: Dumitru Ilie
Editorialist: Dumitru Ilie
First solve: Gheal după 0:18:49

Citat inspirațional din timpul concursului: "vreau să spun că la nuclear reactor îmi ia mie meltdown serverul" - *CNC*

Vom arăta cum se poate obține puterea pentru toate materialele de un anume tip. Fie $(i_1, j_1), (i_2, j_2), \ldots, (i_K, j_K)$ pozițiile din matrice la care se găsește acest material, sortate în ordine lexicografică. Vom procesa aceste poziții în ordine. Pentru poziția (i_a, j_a) ne interesează câte poziții $b \leq a$ îndeplinesc proprietatea $j_b \leq j_a$. Proprietatea $i_b \leq i_a$ este implicită din ordinea de procesare. Trebuie deci să calculăm câte numere $\leq j_a$ apar în șirul $j_1, j_2, \ldots, j_{a-1}$.

Pentru a calcula acest număr rapid, vom modifica puțin definiția. Fie cnt_n numărul de apariții ale numărului n de până acum. Atunci noi vrem

$$\sum_{i=1}^{a} cnt_i$$

Observăm că acest număr reprezintă suma pe prefix a vectorului *cnt*. Această cantitate poate fi calculată și actualizată rapid folosind un arbore indexat binar sau un arbore de intervale (deși acesta este mai încet). După ce am procesat materialele de un anumit tip, vom reseta arborele și vom repeta acest procedeu pentru toate celelalte materiale.

Complexitatea finală este $O(N^2 \cdot \log_2 N)$. Soluțiile care folosesc un arbore de intervale sau un arbore indexat binar 2D pot obține punctaj maxim, dacă sunt implementate atent.

Red Splay BTreap

Propunător: Ilie-Daniel Apostol
Developer: Vlad-Mihai Bogdan
Editorialist: Vlad-Mihai Bogdan
First solve: :(

Problema este în mare doar una de implementare. Pentru ușurință în exprimare, vom folosi abrevierea RBT pentru un Red Black Tree și abrevierea RSBT pentru un Red Splay BTreap.

Proprietatea cea mai importantă a unui RBT este că adâncimea sa este de ordinul $logN$. Prin urmare, și înălțimea unui RSBT (care poate fi colorat ca un RBT) este tot de ordinul $logN$.

Vom rezolva problema folosind programare dinamică, pornind de la frunzele arborelui și urcând în sus. Așadar, vom defini $count_{u,h,color,is\_rbt\_or\_not}$ ca fiind numărul de colorări ale subarborelui cu rădăcina în nodul u, care au black height-ul h, colorând nodul u cu culoarea *color* și știind despre colorare că este o colorare RBT sau o colorare $RSBT$ (caz în care știm că avem black height-ul maxim egal cu h și black height-ul minim egal cu $h - 1$).

Este nevoie de atenție la scrierea relațiilor de recurență. Răspunsul pentru problemă este $\sum_{h=0}^{depth} count_{1,h,black,not\_rbt}$.

Complexitatea timp și spațiu este $O(N \cdot logN)$.

Eritrocit

Propunător: Vlad-Mihai Bogdan
Developer: Vlad-Mihai Bogdan
Editorialist: Vlad-Mihai Bogdan
First solve: RobBobBot după 2:29:03

Pentru a împărți arborele în K componente, trebuie, de fapt, să tăiem $K-1$ muchii din arbore, astfel încât să nu avem o muchie (u, v) cu $color_u \neq color_v$.

Prin urmare, în orice soluție suntem forțați să tăiem toate muchiile (u, v) cu $color_u \neq color_v$. Vom nota cu b numărul de muchii care trebuie neapărat șterse. După ștergerea celor b muchii, am rămas cu o pădure de arbori. Trebuie să mai ștergem $K - b - 1$ muchii din această pădure.

Acum, orice mod de a alege $K - b - 1$ muchii pentru a fi șterse ne dă o împărțire unică în componente a arborelui. Prin urmare, răspunsul este $\sum_{i=0}^{K-b-1} \binom{N-b-1}{i}$.

Orice soluție cu complexitatea timp $O(N)$ sau $O(N \cdot logN)$ ar trebui să obțină punctaj complet. Complexitatea spațiu este $O(N)$.

Comisia:

- Ilie-Daniel Apostol
- Vlad-Mihai Bogdan
- Marius-Stelian Brabete
- Giulian Buzatu
- Dumitru Ilie
- Sebastian Popa
- Ștefan-Alexandru Popescu