Tabăra de pregătire a lotului național de informatică - juniori, Zalău 22-27 mai
Baraj 4
Descrierea soluțiilor

Comisia științifică

May 28, 2025

Problema 1. Căsuța

Propusă de: Stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Politehnica București

Subtaskul 1

Dacă toată matricea este valoroasă, atunci întreaga arie a fiecărui triunghi este valoroasă, deci trebuie doar să tipărim, pentru fiecare triunghi, valoarea

$$\frac{(x_2 - x_1)(y_2 - y_1)}{2}$$

Subtaskul 2

Dacă aria tuturor triunghiurilor este mică, ne permitem să calculăm, pentru fiecare celulă din submatricea acoperită de triunghi, aria acelei celule, care fie va fi complet acoperită, fie va avea forma unui trapez.

Subtaskul 3

Următoarele subtaskuri necesită precalcularea sumelor parțiale în V, pe linii, pe coloane sau pe două dimensiuni, după caz. Acestea ne oferă în $O(1)$ valoarea unui dreptunghi de mărime $k \times 1$, $1 \times k$, respectiv $k_1 \times k_2$.

De asemenea, în toate subtaskurile următoare vom defini **panta** unui triunghi ca fiind $(x_2 - x_1)/(y_2 - y_1)$, invers decît în sensul strict geometric.

Dacă T este suficient de mic, ne permitem o abordare în $O(TN)$, mai exact $O(\text{suma\_lungimilor})$. Pentru un triunghi dat, baleiem x de la x_1 la x_2 și adăugăm la răspuns aria valoroasă pe coloana $[x-1, x]$. Această coloană constă dintr-o coloană de pătrate (pentru care avem precalculată suma valoroasă) și dintr-un trapez inclus într-o celulă a matricei.

Această implementare poate trece și alte teste, în funcție de eficiență. Iată, de exemplu, o rutină care nu efectuează decît două împărțiri per triunghi, una pentru a afla panta și una la final, pentru a calcula aria.

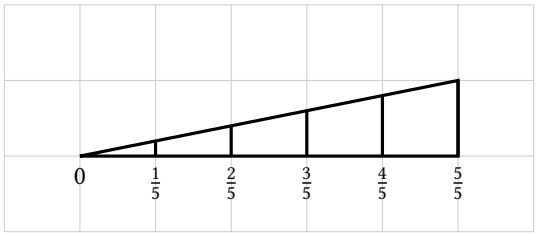
```
double query(int x1, int y1, int x2, int y2) {
  const int slope = (x2 - x1) / (y2 - y1);
  int r_area = 0; // aria dreptunghiurilor de sub triunghiuri
  int t_area = 0; // aria trapezelor înmulțită cu numitorul 2*slope

  for (int x = x1, y = y1; x < x2; x += slope, y++) { // un triunghi
    for (int frac = 0; frac < slope; frac++) {
      // Trapez cu colțul din stînga-jos în (x+frac,y) de lățime 1
      // și cu bazele de înălțime frac/slope și (frac+1) / slope
      int right_x = x + frac + 1;
      r_area += col[y][right_x] - col[y1][right_x];
      // Valoarea celulei care conține trapezul,
      // ca diferență de sume parțiale.
      int rich = col[y + 1][right_x] - col[y][right_x];
      t_area += (frac * 2 + 1) * rich;
    }
  }

  return r_area + t_area / (2.0 * slope);
}
```

Subtaskurile 4 și 6

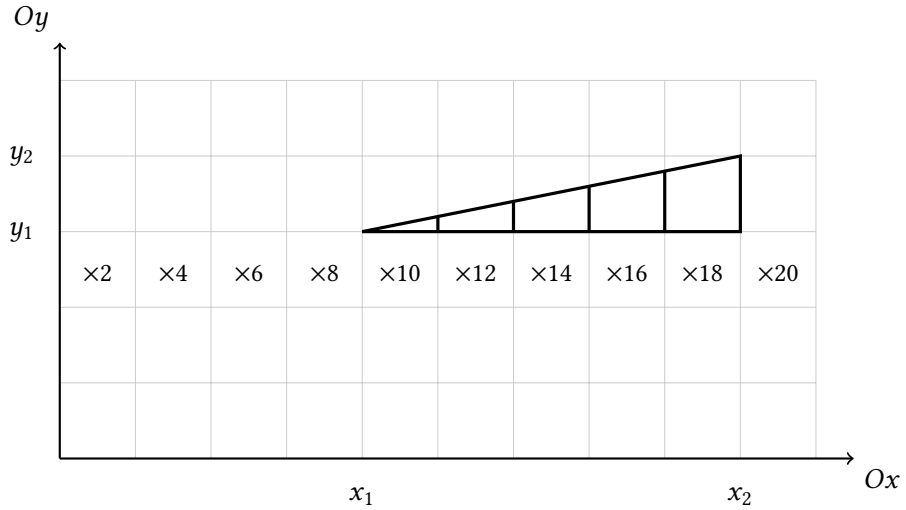
Să vedem cum putem calcula în $O(1)$ valoarea unui triunghi de înălțime 1. Vom exemplifica pe un caz particular ($x_2 - x_1 = 5$) pentru a nu încărca formulele.



Cele 5 trapeze au ariile 1/10, 3/10, 5/10, 7/10 și 9/10. Valoarea triunghiului este însă dată de suma produselor dintre aceste arii și celulele corespunzătoare din matrice. Pentru concizie, fie $c_1, c_2, \ldots, c_n$ valorile matricei V pe linia y_2 (linia cu triunghiul). Atunci dorim să calculăm:

$$\frac{1}{2 \cdot 5} \cdot (1 \cdot c_{x_1+1} + 3 \cdot c_{x_1+2} + 5 \cdot c_{x_1+3} + 7 \cdot c_{x_1+4} + 9 \cdot c_{x_2})$$

Acum, să spunem că triunghiul este așezat la $x_1 = 4$.



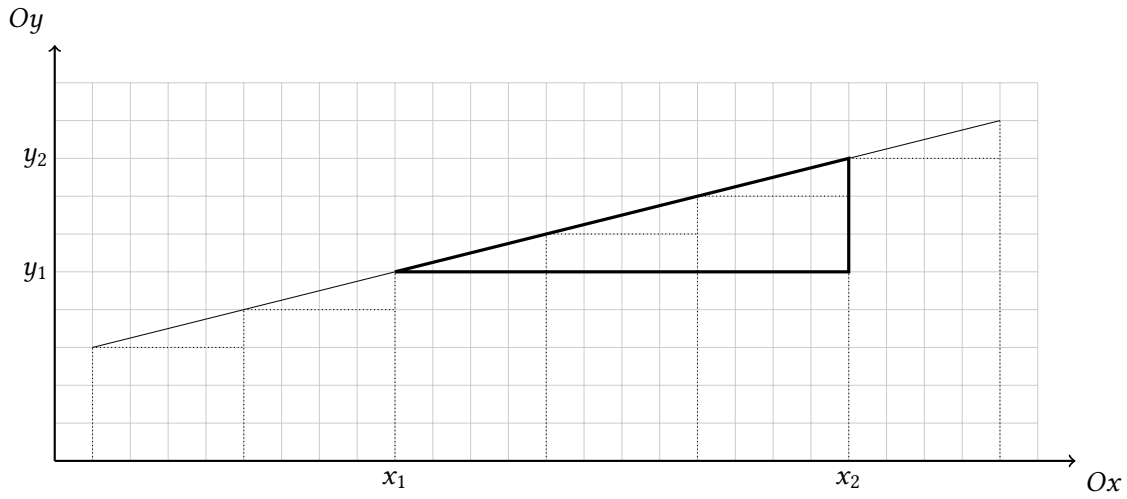
Să precalculăm pe linia curentă, pe fiecare coloană x, suma

$$S_x = 2 \cdot c_1 + 4 \cdot c_2 + \cdots + 2 \cdot x \cdot c_x$$

Atunci $S_{x_2} - S_{x_1}$ este o sumă pe intervalul acoperit de triunghi. Această sumă este, pe toate pozițiile, mai mare cu 9 ($= 2(x_2 - x_1) - 1$) decît aria valoroasă dorită. De aceea, putem scădea de 9 ori aria valoroasă a dreptunghiului $(x_1, y_1) - (x_2, y_2)$.

Subtaskurile 4 și 5

Acum putem aborda și triunghiurile de înălțime mai mare decît 1. Cînd N este suficient de mic, ne permitem o soluție în $O(T + N^3)$. Să considerăm triunghiurile în ordinea pantei (le putem sorta în timp liniar deoarece există cel mult N pante distincte). Pentru o pantă fixată p și pentru fiecare punct (x, y) să calculăm valoarea totală a triunghiurilor de $p \times 1$ care încap în matrice la stînga lui x, precum și a dreptunghiurilor de sub ele (așadar, valoarea unui trapez maximal).



Putem calcula această matrice în $O(N^2)$. Valoarea trapezului care se termină la (x, y) provine din:

- valoarea triunghiului $(x - p, y - 1) - (x, y)$;
- valoarea dreptunghiului $(x - p, 0) - (x, y - 1)$;
- valoarea trapezului anterior, care se termină la $(x - p, y - 1)$.

Atunci putem calcula valoarea unui triunghi cu panta p în $O(1)$ ca fiind:

- valoarea trapezului care se termină la (x_2, y_2);
- minus valoarea trapezului care se termină la (x_1, y_1);
- minus valoarea dreptunghiului $(x_1, 0) - (x_2, y_1)$

Subtaskul 7

Cînd N este suficient de mare, tratăm triunghiurile diferit după cum panta lor este mai mare sau mai mică decît o pantă $P = O(\sqrt{N})$. În practică orice valoare a lui P între 25 și 64 poate obține punctaj maxim.

Cînd panta este mică, procedăm ca la subtaskul anterior și obținem complexitatea $O(T + N^2\sqrt{N})$ pentru toate triunghiurile care au pante mici.

Cînd panta este mare, remarcăm că fiecare triunghi se compune din cel mult N/P triunghiuri de înălțime 1, deci putem însuma, folosind teoria de la subtaskul 6, toate aceste triunghiuri și dreptunghiurile de sub ele, în $O(\sqrt{N})$ per interogare.

Rezultă o complexitate totală de $O((T + N^2)\sqrt{N})$.

Problema 2. Nrk

Propusă de: Stud. Alin-Gabriel Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
Prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin

Soluția oficială - Programare dinamică

Notă: Pentru obținerea răspunsului corect într-o complexitate timp decentă, este necesară colectarea numerelor care pot „candida” la un loc în răspuns, iar apoi sortarea acestora.

Sortarea reprezintă elementul-cheie în cadrul acestei soluții, întrucât în funcție de criteriul folosit, se pot obține rezultate diferite.

Criteriul corect de ordonare a două șiruri x și y este: $x \leq y$ dacă și numai dacă $x \oplus y \leq y \oplus x$, unde $x \oplus y$ denotă concatenarea numerelor x și y în ordinea aceasta.

Intuiție: Când rescriem criteriul d.p.d.v. matematic, obținem:

$$\begin{gathered} x \oplus y \leq y \oplus x \iff \\ x \cdot 10^{|y|} + y \leq y \cdot 10^{|x|} + x \iff \\ \frac{x}{10^{|x|} - 1} \leq \frac{y}{10^{|y|} - 1} \end{gathered}$$

, criteriu în care fiecare număr este independent. Prin $|x|$ ne referim la lungimea numărului x.

Evident, putem sorta folosind acest criteriu toate numerele naturale din intervalul $[A, B]$, dar asta ar duce la TLE din pricina volumului mare.

Observație: Din fiecare grupă de dimensiune a numerelor ($10^L \leq x < 10^{L+1}$) ne interesează doar cele mai mari $\frac{N}{L}$ numere (în cazul în care există). Dacă selectăm mai multe, depășim N cifre în total, deci nu vom folosi numerele selectate în plus.

Colectăm aceste numere într-un vector V, pe care îl sortăm **descrescător**. Acum avem o ordonare favorabilă a numerelor, ceea ce înseamnă că orice soluție va fi un subșir de lungime K al lui V cu suma lungimilor numerelor egală cu N.

În continuare vom utiliza următoarea structură de programare dinamică:

$dp[s][i][j]$ = cel mai mare număr de lungime i care se poate obține prin alipirea a j numere distincte dintre primele s

Pentru a adăuga fiecare număr $V[s] = NR$ în structură vom utiliza următoarea recurență:

1: **for** $s \leftarrow 1$ to lungime(V) **do**
2: $\quad dp[s] \leftarrow dp[s-1]$ ▷ atribuire de matrice, în caz că nu folosim NR
3: $\quad$ **for** $i \leftarrow N$ to $|NR|$ **do**
4: $\quad\quad$ **for** $j \leftarrow k$ to 1 **do**
5: $\quad\quad\quad dp[s][i][j] \leftarrow \max\{dp[s][i][j], dp[s-1][i-|NR|][j-1] \oplus NR\}$

În practică, ne dăm seama că nu ne este necesară o matrice tridimensională dp, ci ne este suficient un singur strat în care adunăm, pe rând, fiecare $V[s]$. În final, complexitatea algoritmului descris mai sus se calculează astfel:

- Partea de selectare: În total vom selecta cel mult $\sum_{i=1}^{i \leq N} \frac{N}{i} = O(N \cdot \log N)$ numere. Suma lungimilor numerelor va fi $O(N^2)$, căci la fiecare lungime ne oprim când depășim N cifre colectate. Generarea numerelor presupune decrementări în $O(1)$. Astfel complexitatea este $O(N^2)$.
- Partea de sortare: $O(N^2 \cdot \log N \cdot \log(N \cdot \log N)) = O(N^2 \cdot \log^2 N)$. Aceasta deoarece sortăm un vector cu $N' = N \log N$ numere făcând $O(N' \log N')$ comparații, iar fiecare comparație durează $O(N)$.
- Programarea dinamică: $O(K \cdot N^3 \cdot \log N)$. Aceasta deoarece completăm o matrice de dimensiuni $N \log N \times N \times K$, iar completarea fiecărei celule necesită o comparare de șiruri.

Complexitatea timp finală va fi $O(K \cdot N^3 \cdot \log N)$, mult amortizată din pricina lungimilor numerelor.

Complexitatea memorie va fi $O(N^2(K + \log N))$: un vector de $O(N \log N)$ șiruri de lungime $O(N)$ și matricea dp care reține $N \cdot K$ șiruri de lungime $O(N)$.

Optimizare: Pentru a reduce complexitatea timp, se pot menține coduri hash pentru prefixele numerelor, astfel compararea se poate face prin căutare binară pe rezultat în $O(\log |NR|)$, fapt care duce la o complexitate timp de $O(K \cdot N^2 \cdot \log^2 N)$.

Problema 3. Passepartout

Propusă de: Instr. Cristian Frâncu, Nerdvana București

Definim distanța Manhattan între două poziții din matrice, (L_1, C_1) și (L_2, C_2) ca fiind $|L_2 - L_1| + |C_2 - C_1|$, unde $|A|$ este valoarea absolută a lui A.

Subtaskul 1

Când avem o singură țară răspunsul este distanța Manhattan de la (1, 1) la cea mai apropiată valoare 1, la care se adaugă 1 pentru poziția de pornire. Complexitate $O(N^2)$ timp și memorie.

Subtaskul 2

Când avem două țări, putem calcula distanța până la țara 1 conform primului subtask. Apoi, pentru fiecare valoare 1, calculăm distanța la cel mai apropiat 2. Pentru aceasta putem folosi un algoritm BFS în matrice (zis și Lee) cu multiple puncte de pornire în toate valorile 2. Pe măsură ce atingem valori 1 marcăm distanțele.

În final răspunsul este minimul sumei celor două distanțe pentru toate valorile 1. Complexitate $O(N^2)$ timp și memorie.

Subtaskul 4

Când N este foarte mic putem folosi o parcurgere în adâncime, similară cu un backtracking, dar care nu eșuează: parcurgem, pe rând fiecare poziție 1. Pentru fiecare poziție 1, încercăm fiecare poziție 2 și așa mai departe. Complexitate $O((N^2/M)^M)$ ca timp, $O(N^2)$ memorie.

Subtaskul 5

O soluție de complexitate $O(N^5)$ sau $O(N^4 \times M)$ ar trebui să ia punctele acestui subtask. Iată o soluție $O(N^4 \times M)$: Pentru fiecare țară $2 \leq K \leq M$ și pentru fiecare pereche de coordonate (L_1, C_1) și (L_2, C_2), dacă (L_1, C_1) aparține țării $K-1$, iar (L_2, C_2) aparține țării K, atunci încercăm să îmbunătățim distanța până la (L_2, C_2) prin (L_1, C_1).

Subtaskul 6

Să presupunem că am calculat lungimile minime ale drumurilor până la toate pozițiile țării $K-1$. Dorim să calculăm minimele pentru pozițiile țării K. Pentru fiecare poziție (L, C) unde avem o valoare K vom calcula:

$$D[L][C] = \min\{D[L_i][C_i] + |L - L_i| + |C - C_i|\}$$

unde (L_i, C_i) sunt pozițiile în matrice ale țării $K-1$.

Pentru o implementare de complexitate $O(N^4/M)$ va trebui să colectăm pentru fiecare țară pozițiile unde apare valoarea ei în matricea inițială.

Această soluție va lua punctaj parțial. Pentru a lua toate punctele acestui subtask observăm că nu este necesar să luăm în considerare toate pozițiile unei țări, ci doar cele de pe frontieră.

O altă îmbunătățire este ca, la fiecare trecere de la țara $K-1$ la țara K, să luăm în considerare doar punctele din țara $K-1$ ce au valori minime ale drumului, relativ la vecinii lor. Aceste „puncte speciale" sunt suficiente deoarece orice drum de la o altă poziție din țara $K-1$ la orice poziție din țara K poate fi ajustat ca să treacă printr-un punct special. Această optimizare va trece și teste din subtaskul următor.

Subtaskul 7

Putem gândi problema puțin diferit: am putea aplica BFS pe matrice (zis și Lee). Dar cum procedăm când, în parcurgere, ajungem la țara 1? Am putea să calculăm distanțele la țara 1, apoi să reluăm un Lee modificat, în care pornim cu lungimile drumurilor sortate crescător. Când

introducem o nouă valoare în coadă, o vom insera în ordine crescătoare. Vom folosi o coadă de priorități (heap). Algoritmul seamănă cu cel al lui Dijkstra, de drum minim în graf.

Pentru a nu complica algoritmul Lee, putem face altceva: atunci când dăm de o valoare 1, să ne deplasăm **în jos**. Ne putem imagina o matrice 3D, în care prima „felie" orizontală, cea de sus, corespunde țării 1, următoarea țării 2 și așa mai departe. Când parcurgem BFS această matrice, în „felia" K, vom genera vecinii în același plan dacă poziția curentă nu are valoarea K în matricea originală. Altfel vom rămâne la aceeași poziție și vom trece la următoarea „felie", $K + 1$.

Algoritmul trebuie implementat cu grijă pentru a nu depăși memoria. Complexitatea este $O(N^2\dot{M})$ ca timp și memorie.

Subtaskul 8

Ne propunem să calculăm, pe rând, drumurile minime către țara 1, apoi către toate țările $2, 3, \ldots, M$. Să presupunem că am calculat drumurile minime până la țara $K - 1$ și dorim să le calculăm pentru țara K. Pentru orice astfel de poziție, drumul optim poate veni de la o poziție anterioară ce poate fi în direcția unuia din colțurile matricei. Putem, deci, evalua cele patru drumuri optime posibile, apoi selectăm minimul dintre ele. Vom arăta cum rezolvăm problema pentru colțul (1, 1).

Parcurgem matricea pe linii. Pentru fiecare poziție vom calcula drumul minim până la acea poziție, indiferent dacă acea poziție aparține țării K ce se dorește a fi calculată. Atunci:

$$P_1[L][C] = \begin{cases} D[L][C] & \text{dacă } A[L][C] = K - 1, \text{ sau} \\ \min\{P_1[L-1][C] + 1, P_1[L][C-1] + 1\} & \text{altfel} \end{cases}$$

Similar vom calcula P_2, P_3 și P_4, lungimile minime pentru cele patru direcții. Parcurgerile se modifică pentru a calcula minimele parțiale corect.

La final calculăm matricea D pentru pozițiile unde avem valori K:

$$D[L][C] = \min\{P_1[L][C], P_2[L][C], P_3[L][C], P_4[L][C]\}$$

Complexitatea acestei soluții este tot $O(N^2 \times M)$ ca timp, dar $O(N^2)$ memorie. Constanta este mai mică, drept care va lua 100p.

Există și o soluție $O(N^2 \log N)$. Ideea este similară cu soluția anterioară, dar ne propunem să facem trecerea de la țara $K - 1$ la țara K în $O(N \log N)$. Considerând că avem drumurile calculate pentru țara $K - 1$, dorim să calculăm drumurile pentru țara K. Vom proceda similar, împărțind problema în patru subprobleme, drumurile optime ce vin din cele patru direcții.

Din nou, să considerăm direcția colțului (1, 1).

Să considerăm următoarea structură vectorială $V[]$: să presupunem că pe coloana C avem poziții cu valori $K-1$. Fie ele L_i, cu drumurile minime precalculate $D[L_i][C]$. Atunci, la poziția $V[C]$ vom memora $\min(D[L_i][C] - L_i - C)$.

Acum, parcurgem matricea pe linii. Atunci când la poziția (L, C) întâlnim valoarea $K-1$ în A, vom actualiza elementul $V[C]$ cu minimul corespunzător. Atunci când la poziția (L, C) întâlnim valoarea K în A, răspunsul pentru $P_1[L][C]$ va fi:

$$P_1[L][C] = \min\{L + C + V[C_i]\} = L + C + \min\{V[C_i]\}$$

unde $C_i \leq C$. De ce? Deoarece drumul optim este:

$$P_1[L][C] = \min\{D[L_i][C_i] + L - L_i + C - C_i\} = L + C + \min\{D[L_i][C_i] - L_i - C_i\}$$

pentru $L_i \leq L$ și $C_i \leq C$. Deoarece introducem punctele $K-1$ în V prin parcurgere pe linii, ne asigurăm că ambele condiții sunt îndeplinite.

Dacă vom căuta minimul liniar, obținem un algoritm $O(N^3)$. Însă putem folosi o structură de calcul rapid al minimului pe intervale, cum ar fi un arbore de intervale sau un arbore indexat binar. Complexitatea scade la $O(N^2 \log N)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- Stud. Alin-Gabriel Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin
- Instr. Cristian Frâncu, Nerdvana București
- Prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani
- Prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil” Buzău
- Instr. Cătălin Frâncu, Nerdvana București
- Stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Prof. Mihai Bunget, Colegiul Național Tudor Vladimirescu, Târgu-Jiu
- Prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj, Târgu-Jiu
- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași

- Stud. Răzvan Alexandru Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Stud. Rareș-Andrei Cotoi, Universitatea Babeș-Bolyai, Cluj, Facultatea de Matematică și Informatică
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Dan Pracsiu, Liceul Teoretic Emil Racoviță, Vaslui
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- Stud. Petruț-Rareș Gheorghieș, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- Stud. Ioan-Cristian Pop, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București