

Problema 2 <traseu>

Descriere a unei/unor soluții posibile

Gradul de dificultate: 2

Structuri de date utilizate: matrice, vectori, alocare dinamica, liste dublu inlantuite

Solutia 1 – 40 puncte -- $O(N*M*N*M)$

(sursa: traseu40.cpp)

Pentru fiecare pozitie $(x1,y1)$ din matrice se parcurg pozitiile $(x2,y2)$ aflate spre Sud si spre Est, $x1 \leq x2 \leq N$ si $y1 \leq y2 \leq M$ si se verifica daca $a[x1][y1] < a[x2][y2]$.

Solutia 2 – 90 puncte -- $O(N*M*N)$

(sursa: traseu90_1.cpp)

Faptul ca avem numere distincte si ca apar toate numerele de la 1 la $N*M$, simplifica oarecum problema.

Putem parcurge valorile in ordine crescatoare si pentru o anumita valoare v , vom incerca sa gasim un traseu optim ce incepe la pozitia valorii v . Fie aceasta pozitie (p, q) . Evident, finalul unui astfel de traseu se va afla in submatricea $A[p..N][q..M]$.

Daca eliminam din matrice valorile tratate anterior, cu exceptia valorii v , toate valorile ramase vor fi mai mari, si deci vor satisface monotonia de altitudine din enunt. Avem cu o constrangere mai putin.

Astfel, vom modela liniile matricei cu liste inlantuite, preferabil dublu inlantuite si circulare, pentru a avea acces usor la ultimul element al liniei. Avem deci un vector de liste $row[1..N]$. Fiecare nod modeleaza o celula din matrice si va contine (r, c) , linia si coloana celulei respective.

Pentru a putea sterge un nod, cel ce corespunde valorii v , este util sa stim pentru fiecare valoare $1..N*M$ ce nod ii corespunde. Acest lucru se poate face cu usurinta in etapa de citire a datelor din fisier, atunci cand se construiesc listele.

Pozitiile ce candideaza pentru finalul traseului ce incepe la pozitia (p, q) , se pot afla pe liniile $p, p+1, \dots, N$. Intrucat vrem ca traseele sa fie cat mai lungi, pentru fiecare dintre aceste linii va fi suficient sa analizam doar ultimul element al listei, i.e. cel mai apropiat de capatul dreapta al matricei. Mai ramane de verificat daca avem un drum valid, adica coloana nodului candidat este $\geq q$.

Reiese usor ca avem un algoritm $O(N*M*N)$. O aparitie destul de neasteptata a listelor inlantuite!

Solutia 3 – 90 puncte -- $O(N*M*\min(M, N))$

(sursa: traseu90_2.cpp)

Sa ne concentram atentia asupra formei 1D a problemei, i.e. atunci cand avem o singura linie.

Astfel, presupunem ca avem un vector $A[1..n]$ cu elemente distincte si dorim sa identificam doua pozitii $i < j$ cu $A[i] < A[j]$, care maximizeaza $j - i$.

O prima observatie e ca daca avem $A[p] < A[q]$ cu $p < q$, o solutie (q, r) , $q < r$ nu ar fi optima intrucat (p, r) este o solutie mai buna, $r-p > r-q$. Putem concluziona ca lista candidatilor, ca puncte de start, este descrescatoare (de la stanga la dreapta). In consecinta, daca pozitia p este un candidat bun de start, atunci $A[p]$ va trebuie sa fie mai mic decat toate elementele din stanga lui.

O observatie similara in ceea ce priveste punctele de oprire arata ca daca $A[p] < A[q]$ cu $p < q$, atunci o solutie (r, p) cu $r < p$ ar fi suboptima intrucat (r, q) este superioara. Din nou avem de-a face cu o lista descrescatoare de candidati. Un candidat p ca punct de oprire este bun, daca $A[p]$ este mai mare decat toate valorile din dreapta lui.

Bazandu-ne pe aceste observatii putem defini:

$left[i] = \min\{A[k] \mid 1 \leq k \leq i\}$

$right[i] = \max\{A[k] \mid i \leq k \leq n\}$

Calculul acestor vectori se face pur si simplu prin determinarea minimului respectiv a maximului prin doua parcurgeri, una de la stanga la dreapta si una in sens invers.

Acum putem folosi "two pointers technique". Pornim cu $i = j = 1$, si pentru un i fixat, atata timp cat $left[i] < right[j]$ putem incrementa j , extinzand astfel solutia curenta. Cand vom trece la urmatorul candidat $i' > i$ cu $left[i'] < left[i]$, din cauza monotoniei se vede ca j nu va mai trebui resetat, putand continua din pozitia lui curenta. Avem astfel o solutie $O(N)$.

Sa incercam mai departe sa adaptam solutia 1D la varianta bidimensionala.

O abordare foarte utila in multe situatii, este fixarea a doua linii u si v , $u \leq v$, si cautarea solutiilor ce au punctul de start pe linia u si punctul de stop pe linia v .

Odata cu fixarea celor doua linii, am stabilit numarul de celule vizitate prin pasi Nord->Sud, ramanand de rezolvat problema pe directia Vest->Est.

Similar cu solutia anterioara, putem calcula $left[]$ in linia u , respectiv $right[]$ in linia v . Implementarea decurge similar, in plus pentru o solutie valida trebuind sa fie satisfacuta relatia $i \leq j$.

Complexitatea finala devine $O(M*N*\min(M, N))$, daca transpunem matricea a.i. numarul liniilor sa fie mai mic decat numarul coloanelor, in cazul in care acest lucru nu este deja respectat. De observat ca prin transpunere, liniile devin coloane si reciproc, si in consecinta vom schimba directiile de mers, Sud devenind Est si invers, insa lungimea traseului optim nu se schimba.

Solutia 4 – 90 puncte -- $O(N*M*(M+N))$
(sursa: traseu90_3.cpp)

Similar cu solutia precedenta, avem ca un traseu ce incepe la pozitia (p, q) nu poate fi optim, daca exista (p', q') , $p' \leq p$, $q' \leq q$ si $A[p'][q'] < A[p][q]$. In acest caz traseul ar putea incepe la pozitia (p', q') si ar fi mai lung. Deci o pozitie (p, q) reprezinta un candidat bun ca punct de start, daca $A[p][q]$ este mai mic decat toate elementele din subtabloul $A[1..p][1..q]$, bineinteles exceptand $A[p][q]$.

Conform observatiei de mai sus, are sens sa definim:

$$\min[i][j] = \min\{A[p][q] \mid 1 \leq p \leq i \ \&\& \ 1 \leq q \leq j\}$$

Se observa ca minimul nu are cum sa creasca daca ne deplasam de la stanga la dreapta sau de sus in jos. Deci, liniile si coloanele matricei $\min[i][j]$ sunt descrescatoare. Avem practic un "tablou Young" descrescator, i.e. $\min[i-1][j] \geq \min[i][j]$, $\min[i][j-1] \geq \min[i][j]$.

Tablourile Young au numeroase proprietati interesante, una dintre ele fiind faptul ca operatia de cautare a unei valori se poate efectua in timp liniar, $O(M + N)$. Vom adapta ideea din spatele acestui algoritm mai jos.

In cazul problemei noastre, pentru fiecare pozitie (i, j) vom incerca sa identificam un traseu optim ce se termina in aceasta pozitie. Punctul de start va fi pozitionat evident in submatricea $A[1..i][1..j]$.

Distingem urmatoarele cazuri:

(a) $\min[1][j] < A[i][j]$ -- Putem renunta sa cautam candidati in coloana j , intrucat $A[1][j]$ este cel mai departat posibil si verifica proprietatea din enunt.

(b) $\min[1][j] > A[i][j]$ -- Putem renunta sa cautam candidati in linia 1 , intrucat din cauza monotoniei lui $\min[i][j]$, valorile din stanga lui $\min[1][j]$ sunt mai mari.

(c) cazul de egalitate poate fi evitat, intrucat valorile sunt distincte si $A[i][j]$ nu poate fi egal cu alte elemente.

Deci putem elimina fie o linie, fie o coloana, reducand zona de cautare la una mai mica. Procesul se repeta in aceasta maniera pana cand zona de cautare devine vida. Mai concret, se porneste cu coltul dreapta-sus, $(p, q) = (1, j)$, si in urma comparatiei dintre $\min[p][q]$ si $A[i][j]$, fie se incrementeaza p , ori se decrementeaza q . Se vor face maxim $i+j$ pasi, ceea ce conduce la o solutie finala $O(M*N*(M+N))$.

Solutia 5 – 90 puncte -- $O(N * M * N)$
(sursa: traseu90_4.cpp)

Vom procesa elementele in ordine crescatoare, si daca la un moment dat avem o valoarea v , aflata la coordonatele (x, y) vrem sa aflam cel mai lung traseu care se termina la aceasta pozitie.

Asta inseamna ca trebuie ca traseul sa inceapa la una din valorile mai mici, valori care au fost deja procesate. Iar dintre acestea se doreste cea la distanta maxima de (x, y) mai sus si mai la stanga.

Putem tine astfel pentru fiecare linie cea mai din stanga pozitie procesata: $\text{best}[i] = y$ minim astfel incat (i, y) a fost deja procesat

Cand suntem la valoarea v , de la pozitia (x, y) putem verifica toate liniile de la 1 la x , si important este doar cea mai din stanga pozitie procesata.

Deci pentru linia i , ne uitam la $\text{best}[i]$

- (a) Daca $best[i] \leq y$, atunci traseul care incepe la $(i, best[i])$ si se termina la (x, y) e un potential traseu maxim. Daca este mai lung decat cel mai lung traseu gasit pana la momentul actual actualizam raspunsul
- (b) Daca $best[i] > y$, sau $best[i]$ inca nu a fost calculat nu facem nimic.

Deoarece pentru fiecare valoare s-au procesat maxim N linii complexitatea finala este $O(N * M * N)$.