Descrierea soluţiilor

Problema 1. Foc

Autor: prof. Marinel Şerban, Colegiul Naţional "Emil Racoviţă" Iaşi

Soluţia 1

Vom numi zonă o mulţime de parcele cultivate cu proprietatea că oricare ar fi două parcele `x` sau `y` din zonă fie `x` şi `y` sunt vecine, fie există o succesiune de parcele z_1, z_2, ..., z_k astfel încât `x` este vecin cu z_1, z_i este vecină cu z_{i+1} ($1 \le i < k$) şi z_k este vecină cu `y`.

Printr-un algoritm de *fill*, identificăm toate zonele de pe hartă, şi reţinem pentru fiecare zonă dimensiunea acesteia.

Plasăm un dreptunghi imaginar de dimensiuni `X*Y` în colţul `(1,1)` al matricii care reprezintă terenul.

Vom utiliza un vector de frecvenţă `nr` pentru a contoriza numărul de parcele din fiecare zonă "acoperită" de dreptunghi (`nr[i]`=numărul de parcele din zona i acoperite de dreptunghi).

Într-o variabilă denumită `cat` însumăm dimensiunile zonelor distincte acoperite de dreptunghi.

Vom deplasa acest dreptunghi "şerpuit" (pe liniile cu număr de ordine impar dreptunghiul este deplasat de la stânga la dreapta, pe liniile cu număr de ordine par dreptunghiul este deplasat de la dreapta la stânga). Pentru fiecare poziţie a dreptunghiului în variabila `cat` se află suma dimensiunilor parcelelor distincte acoperite (total sau parţial) de dreptunghi şi de fiecare dată, comparăm, evident variabila `cat` cu maximul.

Când dreptunghiul ajunge la capătul unei linii (adică pe coloana `m-y+1` pentru liniile impare sau pe coloana 1 pentru liniile pare), dreptunghiul este deplasat cu o linie mai jos.

Când deplasăm un dreptunghi nu trebuie să recalculăm vectorul de frecvenţă `nr`, trebuie doar să-l actualizăm.

De exemplu, dacă dreptunghiul are colţul din stânga-sus în poziţia (`lin`, `col`) şi îl mutăm:

- cu o coloană spre dreapta: dispare latura din stânga a dreptunghiului (de pe coloana `col` elementele situate pe liniile `lin`, ..., `lin+X-1`) şi apare o nouă latură la dreapta (pe coloana `col+Y`, elementele situate pe aceleaşi linii);
- cu o coloană spre stânga: dispare latura din dreapta a dreptunghiului (de pe coloana `col+Y-1` elementele situate pe liniile `lin`, ..., `lin+X-1`) şi apare o nouă latură la stânga (pe coloana `col-1`, elementele situate pe aceleaşi linii)
- cu o linie în jos: dispare latura de sus (elementele situate pe linia `lin` şi coloanele `col`, ..., `col+Y-1`) şi apare o nouă latură jos (elementele de pe linia `lin+X`, aceleaşi coloane).

Complexitatea algoritmului este O(`n*(m*X+Y)`).

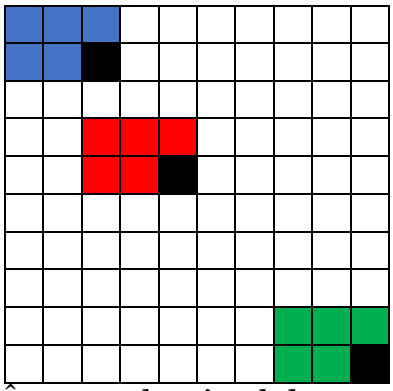
În funcţie de implementare, această soluţie poate obţine maximum 93 de puncte.

Soluția 2 – 100 de puncte

Să analizăm abordarea standard a rezolvării şi să observăm că putem să identificăm un dreptunghi prin poziția colțului său din dreapta-jos.

Exemplu: `n=10`, `m=10`, `X=2`, `Y=3`

1. Primul dreptunghi: Albastru, poziţia de definiţie (2,3)
2. Al doilea dreptunghi : Roşu, poziția de definiție (5,5)
3. Al treilea dreptunghi : Verde, poziția de definiție (10,10)



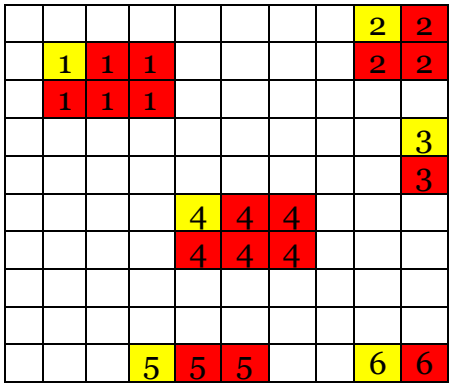
În general, primul dreptunghi are poziția de definiție (`X`,`Y`), ultimul dreptunghi are poziția de definiție (`N`,`M`), iar un dreptunghi oarecare are poziția de definiție (`i`,`j`) cu `X≤i≤N` şi `Y≤j≤M`.

Fie o poziție (`lin`,`col`) a unei zone Z care conține `A` poziții cultivate. Dacă un dreptunghi cu poziția de definiție (`i`,`j`) conține poziția (`lin`,`col`) atunci spunem că dreptunghiul se găsește în zona de afectare a poziției (`lin`,`col`). În acest caz, `i-lin≤X-1`, `lin≤i≤lin+X-1` şi `i≤n`; `j-col<=Y-1`, `col≤j≤col+Y-1` şi `j≤n`. Mulțimea de afectare a unei poziții (`lin`,`col`) este dreptunghiul delimitat:

- sus de linia `U=lin`,
- jos de linia `D=min(n,lin+X-1)`,
- stânga de coloana `L=col`,
- dreapta de coloana `R=min(m,col+Y-1)`.

În imaginea următoare, pentru `n=10`, `m=10`, `X=2`, `Y=3`, am exemplificat 6 mulțimi de afectare.

Toate acestea sunt dreptunghiulare şi au poziția (`lin`,`col`) care determină mulțimea de afectare evidențiată cu culoare galbenă. De exemplu, mulțimea 1 are (`lin`,`col`) =(2,2), mulțimea 3 are (`lin`,`col`)=(4,10), iar mulțimea 6 are (`lin`,`col`)=(10,9). Pentru mulțimea 3: U=4, L=10, D=5, R=10. Pentru mulțimea 4: U=6, L=5, D=8, R=7. Pentru mulțimea 5: U=10, L=4, D=10, R=6.



Dacă vom considera toate poziţiile (`lin`,`col`) pentru o zonă Z, vom putea defini mulţimea de afectare a zonei Z drept reuniunea mulţimilor de afectare pentru toate poziţiile (`lin`,`col`).

Pentru toate dreptunghiurile care au poziţia de definiţie în mulţimea de afectare a unei zone Z avem măcar o poziţie a zonei în interiorul zonei Z deci va trebui să adăugăm la suprafaţa incediată a dreptunghiului aria A a zonei Z. Consemnăm acest lucru adunând la poziţia de definiţie a dreptunghiului valoarea A. Când parcurgem toate zonele şi facem toate adunările, vom găsi la poziţia de definiţie a fiecărui dreptunghi exact suprafaţa incendiată când se alege acel dreptunghi.

Problema este că mulţimea de afectare a unei zone Z este o reuniune de dreptunghiuri şi noi vom dori să adunăm valoarea A în fiecare poziţie a mulţimii de afectare exact o singură dată. În mod normal, valoarea A ar putea fi adunată pe fiecare dreptunghi folosind adunarea lui A într-o matrice Mars 2D. Din păcate această metoda nu ar da soluţia corectă pentru că mulţimea de afectare a unei zone este o reuniune de dreptunghiuri, nu neapărat disjuncte şi atunci valoarea A ar putea să fie adunata de mai multe ori la acelaşi element al mulţimii de afectare.

Ce ne dorim acum este să transformăm mulţimea de afectare într-o reuniune de dreptunghiuri disjuncte. Pentru asta va trebui ca în momentul în care dispunem de două dreptunghiuri să le transformăm în alte dreptunghiuri care să fie disjuncte. Am transformat astfel problema în altă problemă: Fie o mulţime de dreptunghiuri într-o matrice. Să se transforme mulţimea într-o mulţime de dreptunghiuri disjuncte.

În cazul nostru problema mai are şi o particularitate care este utilă în rezolvare: Fie două dreptunghiuri delimitate sus-stânga-jos-dreapta de (U1,L1,D1,R1) şi respectiv (U2,L2,R2,D2). Dacă $U1 \leq U2$ avem $D1 \leq D2$ iar dacă $L1 \leq L2$ avem $R1 \leq R2$.

Vom parcurge dreptunghiurile ordonate după U şi apoi după L.

Astfel, pentru oricare dreptunghi suntem asiguraţi că dreptunghiul curent are linia superioară U maximă deci şi linia inferioară D maximă prin comparaţie cu cele anterioare.

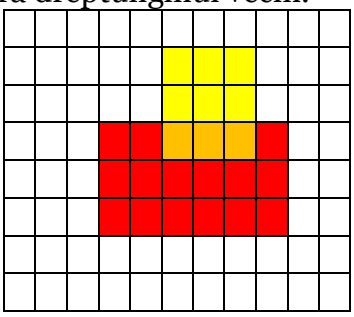
Vom menţine dreptunghiurile procesate într-o structură de date astfel încât să fie menţinută următoarea proprietate:

(P): În dreptul fiecărei coloane avem cel mult un dreptunghi în structură (dreptunghiurile din structură să fie poziţionate pe benzi verticale disjuncte)

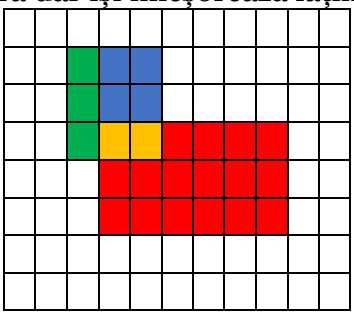
Pentru fiecare adăugare a unui dreptunghi nou vom realiza următoarea procesare. Fie banda verticală determinată de dreptunghiul curent (banda nouă).

1. Dacă un dreptunghi vechi este situat pe o bandă disjunctă de banda nouă atunci la adăugarea dreptunghiului se păstrează proprietatea (P).
2. Dacă un dreptunghi vechi este situat strict deasupra dreptunghiului nou (are linia de jos strict deasupra liniei de sus a dreptunghiului nou), atunci scoatem complet dreptunghiul vechi din structură şi îl adăugăm la reuniunea de dreptunghiuri disjuncte, deoarece acel dreptunghi nu se mai poate intersecta cu alte dreptunghiuri în viitor.
   Observăm că eliminarea unui element din structură păstrează proprietatea (P)
3. Dacă un dreptunghi vechi are banda inclusă în banda nouă atunci vom decupa din dreptunghiul vechi zona în care se intersectează cu dreptunghiul nou.
   Se adaugă la soluţie porţiunea rămasă şi se elimină dreptunghiul vechi din structură.
   În exemplul din figura de mai jos: Dreptunghiul nou-roşu, dreptunghiul vechi-galben cu zona comuna portocaliu. Se elimină din dreptunghiul vechi zona portocalie, se adaugă la reuniunea

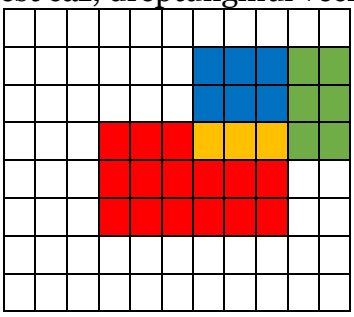
finală zona galbenă situată strict deasupra dreptunghiului nou. Se elimină din structură dreptunghiul vechi.



4. Dacă banda veche intersectează parţial banda nouă, pe stânga avem situaţia prezentată în imaginea de mai jos:
   Cele două dreptunghiuri se intersectează pe zona portocalie care va fi preluată de dreptunghiul nou.
   Din ce mai rămâne din dreptunghiul vechi distingem două zone:
   Cea verde se obţine din dreptunghiul vechi prin mutarea coloanei din dreapta.
   Cea albastră care se adaugă la reuniunea de dreptunghiuri.
   În acest caz, din dreptunghiul vechi rămâne doar zona verde. Dreptunghiul vechi rămâne în structură dar îşi micşorează lăţimea. Astfel proprietatea (P) se va păstra.



5. Dacă banda veche intersectează parţial banda nouă, pe dreapta, avem situaţia prezentată în imaginea de mai jos şi explicaţiile sunt exact aceleaşi ca la cazul anterior cu următoarea modificare:
   Zona verde se obţine din dreptunghiul vechi prin mutarea coloanei din dreapta.
   Şi in acest caz, dreptunghiul vechi rămâne în structură dar îşi micşoreaza lăţimea.



După ce s-au procesat toate dreptunghiurile (mulţimile de afectare), toate dreptughiurile rămase în structură se adaugă la reuniunea de dreptunghiuri disjuncte şi se scot din structură.

În realitate nu se creează reuniunea de dreptunghiuri disjuncte. Nu este necesar. Atunci când din structură se scoate un dreptunghi delimitat de U,L,D,R se aplică pe matricea Mars 2D valoarea ariei A a zonei Z pe dreptunghiul U,L,D,R.

Mai ramâne de explicat cum detectăm zonele şi cum implementăm structura de date care menţine dreptunghiurile disjuncte.

Pentru detectarea zonelor aplicăm algoritmul FILL. Când algoritmul intră într-o poziție (`lin`,`col`) a zonei se incrementează aria A a zonei și se adaugă în șirul dreptunghiurilor mulțimea de afectare pe 4 vectori `U[]`,`L[]`,`D[]`,`R[]` reprezentând delimitările dreptunghiului adăugat.

La finalul FILL-ului se sortează șirul dreptunghiurilor crescător după `U[]` și apoi după `L[]`.

Din acel moment fiecare dreptunghi se va identifica prin poziția sa `i` în șirul sortat, $1 \le i \le A$.

Structura de date este construită astfel:

- un vector de marcaj de dimensiune `m` pe care atunci când adăugăm un dreptunghi identificat prin poziția sa `i` vom adăuga pe pozițiile `L[i]` și `R[i]` a vectorului valoarea `i`.
- un bitset de dimeniune tot `m` în care vom marca 1 o poziție de câte ori adăugăm o valoare în vectorul de marcaj.

Când eliminăm un dreptunghi din structură, ștergem din structură capetele `L[i]` și `R[i]`, aplicând valoarea 0 la aceste poziții, atât în vectorul de marcaj cât și în bitset.

Această structura ad-hoc ne permite să procesăm rapid fiecare dreptunghi nou astfel:

Dacă dreptunghiul `i` are banda corespunzătoare `st=L[i]` și dreapta `dr=R[i]` pe noi ne interesează să descoperim care sunt dreptunghiurile active (adică cele care există acum în structură) care au măcar un capăt între `st` și `dr`. Acestea se identifică prin toate valorile existente în vectorul de marcaj între pozițiile `st` și `dr`. Aceste poziții sunt marcate în bitset și pot fi identificate foarte ușor în bitset folosind metodele `_Find_first()` și/sau `_Find_next()` ale clasei bitset. Odată identificați toți biții dintre `st` și `dr` se elimină din structură toate dreptunghiurile cu excepția eventual a două dreptunghiuri (unul – cel mai din stânga dacă se încadrează în situația 4 și altul – cel mai la dreapta dacă se încadrează pe situația 5). Pentru aceste cazuri speciale se vor reintroduce dreptunghiurile în structură.

Analiza complexității:

Algoritmul FILL va avea complexitatea $O(N^2)$ dar la aceasta se mai adaugă un factor $\log(N^2)$ pentru sortarea dreptunghiurilor de afectare.

Pe partea de procesare avem tot $O(N^2)$ deoarece operațiile care trebuie numărate la procesarea unui dreptunghi sunt: intrarea dreptunghiului în structură, ieșirea dreptunghiului din structură, decuparea eventuală a încă două dreptunghiuri anterioare. De remarcat că atunci când dreptunghiul nou elimină complet unul vechi operația se numără ca ieșire pentru dreptunghiul vechi, deci nu și la dreptunghiul nou și ca un dreptunghi chiar dacă ar putea fi decupat chiar și de Y ori, acele operații se numara la dreptunghiul nou (cel care decupează) și nu la cel vechi (cel decupat).

Și în acest pas se poate invoca un factor suplimentar, acela în care se detectează fiecare bit al intervalului [st,dr] dar datorită dimensiunii mici a bitset-ului cele doua metode ale clasei bitset au o complexitate apropiată de numărul de biți 1 situați în zona respectivă deci pot fi estimate la numărul de operatii deja efectuate.

În cele din urmă complexitatea poate fi estimată la O(N*N*log(N*N)).

Problema 2. Teren

Autor: stud. Cociorvă Ana-Miruna, Facultatea de Informatică, Universitatea „Al. I. Cuza” Iaşi

Soluția cu backtracking – K=0 (12 puncte)

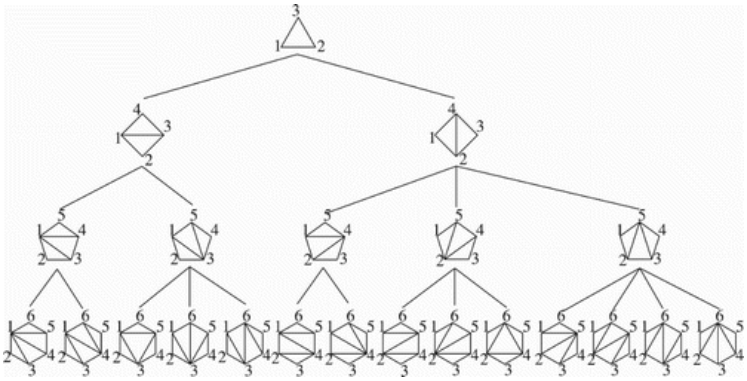
Se generează cu un algoritm de tip *backtracking* cu toate modalitățile de partiționare a terenului.

Soluția cu numere Catalan K = 0 (36 puncte)

Observații

1) Oricum am construi un gard (o diagonală) în interiorul unui poligon convex, cele două porțiuni rezultate sunt de asemenea poligoane convexe.

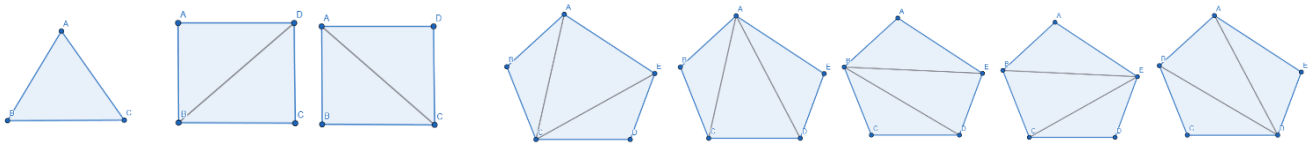
2) Orice partiționare în triunghiuri a unui poligon convex va avea 2 zone triunghiulare cu 2 laturi ale poligonului convex inițial și 1 latură o diagonală, iar restul de n-4 zone triunghiulare vor avea 1 latură a poligonului convex inițial și 2 laturi diagonale.



Numărul de posibilități de a partiționa în triunghiuri un poligon convex cu `n` vârfuri utilizând `n-3` diagonale care nu se intersectează în interiorul poligonului este `Catalan(n-2)`.

În articolul http://campion.edu.ro/arhiva/www/arhiva_2009/papers/paper20.pdf

puteți găsi diferite formule pentru numerele lui Catalan.



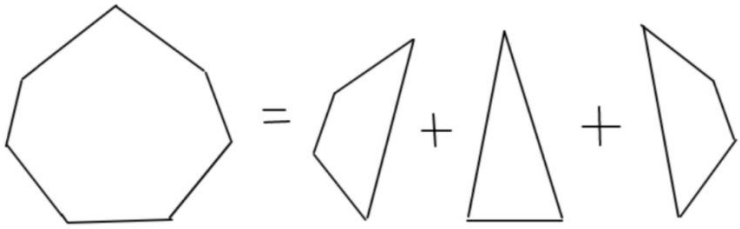
1 2 5

O modalitate de calcul eficientă se bazează pe formula

$$Catalan(n) = \frac{1}{n+1} C_{2n}^{n}$$

Este necesară o precalculare a inverselor modulare ale factorialelor.

Soluția cu programare dinamică - 100 puncte



Vom formula o subproblemă a problemei date astfel: să se determine numărul de posibilităţi de a partiţiona în triunghiuri un poligon convex cu vârfurile Pi, ..., Pj cu exact x triunghiuri sfinte (cu aria număr întreg).

Vom memora soluţia subproblemei în dp[x][i][j].

Iniţializare

$$pentru\ fiecare\ punct\ i\ (1 \le i < n)\ dp[0][i][i+1] = 1;$$

Relaţia de recurenţă

$$\text{dp[x][i][j]} = \sum_{mij=i+1}^{j-1} dp[0]][i][mij] * dp[0][mij][j], x = 0$$

$$dp[x][i][j] = \sum_{mij=i+1}^{j-1} \sum_{t=0}^{x} dp[x-t-r][i][mij] * dp[t][mij][j], x \ge 1$$

Funcția isSfint(i, mij, j) returnează

1 dacă $X_i * (Y_{mij} - Y_j) + X_{mij} * (Y_j - Y_i) + X_j * (Y_i - Y_{mij})$ este un număr par

0 altfel

r = valoarea returnată de funcția isSfint(i, mij, j), 0 sau 1

Complexitatea este $O(n^5)$ – se comportă foarte bine în pratică.