Soluţie-part

Sol. 1 (in O(n^2))
Pentru ca cele n piese sa realizeze o **partitie** trebuie sa se verifice urmatoarele conditii:

- fiecare piesa (triunghiulara) trebuie sa aiba toate varfurile in interiorul sau pe triunghiul desenat pe tabla magnetica
- fiecare doua piese trebuie sa nu se suprapuna, adica sa nu existe doua laturi, una in primul trunghi si a doua in al doilea triunghi, care sa se intersecteze sau sa existe doua triunghiuri unul in interiorul celuilalt
- suma ariilor celor doua triunghiuri trebuie sa fie egala cu aria triunghiului desenat pe tabla.

Pentru a verifica aceste conditii este suficient un algoritm de complexitate O(n^2), care se incadreaza in timpul de executie.

Sol. 2 (in O($n$log n))
Se iau toate punctele ce sunt varfuri ale cel putin unui triunghi dat. Pentru fiecare astfel de punct trebuie verificat ca exista triunghiuri din pavaj de jur imprejur.
Pentru punctele interioare triunghiului mare avem doua cazuri corecte :

- sa existe triunghiur de jur imprejur avand un varf in acel punct
- sa existe triunghiuri cu varful in acel punct acoperind un semiplan, iar celalalt semiplan sa fie acoperit de un triunghi avand punctul considerat pe o latura

Daca un punct nu se incadreaza in cazurile de mai sus, atunci pavajul nu este corect. Daca toate punctele se incadreaza, atunci pavajul e corect.
Observam ca putem sa testam doar daca triunghiurile avand varfurile intr-un punct dat acopera fie tot planul de jur imprejur fie un semiplan, fara a mai testa in al doilea caz existenta unui triunghi avand punctul dat pe o latura: daca nu ar exista un astfel de triunghi atunci s-ar incalca cealalta conditie intr-un punct de-a lungul acelei drepte.
Pentru marginile triunghiului de pavat consideram niste triunghiuri care acopera exteriorul triunghiului de pavat, astfel incat sa nu trebuiasca sa consideram varfurile sau laturile triunghiului mare ca si cazuri particulare.
Complexitatea O($n$log n) vine din sortarea varfurilor triunghiurilor dupa coordonate in vederea gruparii si din sortarea triunghiurilor in jurul fiecarui punct.

Problema EVO – soluţie

Rezolvarea problemei se bazeaza pe metoda programarii dinamice.

Pentru fiecare dintre cele m siruri de intrare vom calcula o matrice OK[i][j] cu urmatoarea semnificatie :
- OK[i][j]=1 daca si numai daca subsecventa sirului de intrare (pe care il vom nota in continuare cu w) cuprinsa intre indicii i si j (subsecventa va fi notata prin w[i..j]) este generata prin hairpin, aplicat de un numar arbitrar de ori, dintr-un sir initial;
- OK[i][j]=0, altfel.

O strategie « naiva » pentru calculul acestei matrice este: completam in primul rand OK[i][j]=1, daca w[i..j] este un sir din multimea initiala.
Apoi, pentru lungime = 3 pana la lungime(w) verificam daca se verifica vreunul din cazurile:
- OK[i][i+lungime-1]=1
- exista o descompunere a sirului w[i..i+lungime-1] in cinci secvente w0w1w2w3w4, astfel incat: w0w1=c(w3w4)^R, lungime(w0w1)>1, OK[i+lungime(w0)][i+lungime-1]=1. In acest caz, vom pune OK[i][j]=1.
- exista o descompunere a sirului w[i..i+lungime-1] in cinci secvente w0w1w2w3w4, astfel incat w0w1=c(w3w4)^R, lungime(w0w1)>1, OK[i][i+lungime-lungime(w4)-1]=1. In acest caz, vom pune OK[i][i+lungime-1]=1.
In acest caz, va fi dat raspunsul "da" daca si numai daca OK[1][n]=1.
Complexitatea acestei rezolvari este O(lungime(w)^3), si ar fi primit 55 de puncte.
Aceasta strategie poate fi imbunatatita prin precalcularea unor valori :
P[i][j]=lungimea celui mai lung prefix al lui w[i..j] care, dupa complementare si oglindire, este si sufix al lui w[i..j].
right[i]=pozitia unde se termina cel mai lung sir din limbaj care incepe pe pozitia i, calculat pana la un moment dat
left[i]= pozitia unde incepe mai lung sir din limbaj care se termina pe pozitia i, calculat pana la un moment dat
Aceste valori se vor calcula tot prin programare dinamica, concomitent cu calculul matricei OK, pornind de la subsecventele scurte catre cele lungi.

De data aceasta OK[i][j]=1 daca right[j]<i + P[i][j] sau left[i]>j-P[i][j].

Datorita limitei de spatiu nu se pot tine in memorie ambele matrice OK si P. Se observa ca in calculul matricei P sunt necesare numai ultimele 3 linii, la fiecare pas.

Aceasta solutie are ca timp de rulare O(lungime(w)^2), si ar fi primit 100 de puncte.

Acolor – descrierea solutiei

Rezolvăm problema prin programare dinamică. Arborele din problema este un arbore binar de căutare şi observăm imediat că predecesorul ca ordine este cel mai din dreapta nod al subarborelui de la stânga, iar succesorul este cel mai din stânga nod al subarborelui din dreapta. Pornind de la această observaţie calculăm ca subprobleme numărul de moduri pentru a colora un anumit subarbore cu K culori, având fixate culoarea celui mai din stânga, culoarea rădăcinii şi culoarea nodului cel mai din dreapta a acelui subarbore.

$C_{n,st,r,dr}$ = numărul de moduri pentru a colora cu K culori subarborele având ca rădăcină nodul n şi colorând nodul cel mai din stânga, nodul n şi nodul cel mai din dreapta cu respectiv culorile st, r, dr.

$$C_{n,st,r,d} = \begin{cases} \\ nfrunza: 1 \text{ pentru } st = r = dr \text{ si } 0 \text{ altfel} \\ \text{n nod interior : notand cu f1 fiul din stanga si cu f2 fiul din dreapta avem} \\ \qquad \sum_{\substack{i \neq r \\ j \neq r}} C_{f1,st,i,j} \times \sum_{\substack{i \neq r \\ j \neq r}} C_{f2,i,j,dr} \end{cases}$$

O implementare directă va avea complexitatea T = O(N*K^3*K^2).
Putem folosi principiul includerii şi excluderii pentru a calcula sumele:

$$\sum_{\substack{i \neq r \\ j \neq r}} C_{n,st,i,j} = \sum_{\substack{i=1,K \\ j=1,K}} C_{n,st,i,j} - \sum_{i=1,K} C_{n,st,i,r} - \sum_{j=1,K} C_{n,st,r,j} + C_{n,st,r,r}$$

Precalculând sumele care apar în membrul drept obţinem o soluţie de complexitate T = O(N*K^3).

Observăm ca avem subprobleme inutile, de exemplu: $C_{n,1,2,3} = C_{n,7,1,10}$ şi putem calcula doar $C_{n,1,1,1}, C_{n,1,1,2}, C_{n,1,2,1}, C_{n,2,1,1}, C_{n,1,2,3}$ prin calcularea sumelor din dreapta prin descompunerea in aceste tipuri şi determinarea coeficienţilor corespunzători.
Obţinem o soluţie cu T = O(N).