**Descrierea soluţiilor**

**Problema 1 esentiale**

lector. dr. Paul Diac, Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza" Iaşi

*Subtask 1*

Cerința 1 este problema clasică a arborelui parțial de cost minim, care se poate rezolva prin implementarea algoritmilor cunoscuți: Kruskal sau Prim. În cazul algoritmului Kruskal, pentru menținerea componentelor conexe se poate folosi structura union-find, dar se pot încadra în timp și metode mai ineficiente. Răspunsul poate depăși valoarea maximă care încape pe 32 biți, deci *int* nu este suficient de mare.

*Subtask 2*

Folosim arborele partial de terminat la prima cerință. Eliminăm temporar câte o muchie a acestui arbore. Pe graful rezultat, se rulează iar aceelași algoritm și se compară costul noului arbore partial de cost minim cu cel obținut pe graful inițial. Dacă acest cost este strict mai mare, înseamnă că muchia eliminată este esențială, deoarece fără ea, nu s-a găsit un arbore parțial de aceelași cost minim. După eliminarea temporară a fiecărei muchii, aceasta trebuie adăugată înapoi în graf. Pentru a rezolva problema cu o singură implementare a algoritmului ales, muchia ștearsă poate fi doar marcată ca fiind interzisă iar muchiile interzise să nu fie ignorate de algoritm.

**Problema 2 gems**

prof. Emanuela Cerchez, Colegiul Naţional "Emil Racoviţă" Iaşi

*Subtask 1*

O abordare recursivă (de tip *backtracking*) poate obţine maximum 26 de puncte, în funcţie de implementare.

*Subtask 2*

Gems este o problemă clasică de programare dinamică.

*Subproblemă*: valoarea totală maximă pe care Gemini o poate colecta până în poziţia (i, j)

*Relaţia de recurenţă*

cmax[i][j]=max{cmax[i-1][j-1], cmax[i-1][j], cmax[i-1][j+1]}+valoare(i,j)

Pentru a nu trata special cazurile j=1 şi j=P (am bordat matricea cmax cu 0).

Soluţia va fi cea mai mare valoare care se poate obţine în cmax la una dintre ieşiri.

Cu valoare(i,j) am notat valoarea pietrei preţioase din poziţia (i,j). Aceasta se poate calcula astfel:

int valoare(int lin, int col)

{int poz=((lin-1)\*p+col)%n;

return v[poz];

}

Pentru subtaskul 2 există memorie suficientă pentru a reţine matricea cmax integral.

*Subtask 3*

Trebuie să optimizăm memoria. Observăm că pentru a determina linia i din cmax este suficient să reţinem linia i-1. Prin urmare nu este necesară toată matricea, este suficient ca la fiecare pas să reţinem două linii consecutive din matrice.

int pd()

{int i, j, lcrt=1, lprec=0, maxim=-1;

for (i=1; i<=p; i++) cmax[0][i]=valoare(1,i);

for (i=2; i<=p; i++)

{for (j=1; j<=p; j++)

cmax[lcrt][j]=valoare(i,j)+max3(cmax[lprec][j-1],cmax[lprec][j],cmax[lprec][j+1]);

lprec=1-lprec; lcrt=1-lcrt;

}

for (i=1; i<=nr; i++)

if (maxim<cmax[lprec][out[i]]) maxim=cmax[lprec][out[i]];

return maxim;

}