**LICEUL „CONSTANTIN BRÂNCOVEANU” HOREZU**

LUCRARE PENTRU OBRȚINEREA ATESTATULUI PROFESIONAL LA INFORMATICĂ

METODA GREEDY

PROFESOR,

Prof.Chișevescu Cristian

ELEV,

Lungu Elena Izabela

2018-2019

CUPRINS

Cap.I. Prezentarea generală a temei…………………………………………………3

* Sinteză
* Algortim general
* Principiul metodei Greedy
* Modul de lucru al algoritmilor

Cap.II. Schema generală și implementare………………………………......………5

Cap.III. Resurse hardware și software necesare………………….........……………7

Cap.IV. Realizarea aplicației.........................………………….................................8

* Pașii algoritmului
* Implementarea strategiei Greedy în exemple de probleme
* Complexitatea algoritmului

Cap.V. Probleme care se pot rezolva cu ajutorul metodei Greedy………….……...11

Cap.VI. Codul sursă.......................…………………....……………....………......14

Cap.VII. Bibliografie....………………………………………………....................24

Cap.I. Prezentarea generală a temei

Metoda de programare **Greedy** se aplică problemelor de optimizare. Această metodă constă în faptul că se construieşte soluția optimă pas cu pas, la fiecare pas fiind selectat în soluție elementul care pare „**cel mai bun/cel mai optim**” la momentul respectiv, în speranța că această alegere locală va conduce la **optimul globa**l. Algoritmii Greedy sunt foarte eficienți, dar nu conduc în mod necesar la o soluție optimă. Şi nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda Greedy rezolvă sau nu o anumită problemă de optimizare. Din acest motiv, orice algoritm Greedy trebuie însoțit de o demonstrație a corectitudinii sale. Demonstrația faptului că o anumită problemă are proprietatea alegerii Greedy se face de obicei prin inducție matematică. Algoritmii Greedy sunt caracterizați de metoda lor de funcționare: la fiecare pas se alege cel mai bun candidat posibil, după evaluarea tuturor acestora. Metoda determină întotdeauna o singură soluție, asigurând un optim local, dar nu întotdeauna și global. Tehnica Greedy este una de optimizare, rulând mai rapid decât un Backtracking, dar nefiind întotdeauna cea mai bună.  
 Metoda **Greedy** se aplică problemelor pentru care se dă o **mulţime A** cu **n** elemente şi pentru care trebuie determinată o **submulţime** a sa, **S** cu **m** elemente, care îndeplinesc anumite condiţii, numite și **condiții de optim**.

Algoritm general:

* se dă o mulțime A ;
* se cere o submulțime S din mulțimea A care:
* să îndeplinească anumite condiţii interne (să fie acceptabilă)
* să fie optimală (să realizeze un maxim sau un minim)

Principiul metodei Greedy:

* se iniţializează mulţimea soluţiilor S cu mulţimea vidă, S=Ø
* la fiecare pas se alege un anumit element x∈A (cel mai promiţător element la momentul respectiv) care poate conduce la o soluţie optimă
* se verifică dacă elementul ales poate fi adăugat la mulţimea soluţiilor:
* dacă da atunci

⚫ va fi adăugat şi mulţimea soluţiilor devine S=S∪{x} - un element introdus în mulţimea S nu va mai putea fi eliminat

* altfel

•el nu se mai testează ulterior

* procedeul continuă, până când au fost determinate toate elementele din mulţimea soluţiilor

Modul de lucru al algoritmilor de tip Greedy:

* Construirea mulțimii S se face printr-un șir de decizii. Inițial se pornește cu mulțimea vidă (S = Ø). Fiecare decizie constă în alegerea unui element din mulțimea A, analiza lui și eventual introducerea lui în mulțimea S. În funcție de modul în care se iau aceste decizii, mulțimea S se va apropia mai mult sau mai puțin de soluția optima S\*. În cazul ideal vom avea S= S\*.
* Algoritmii de tip greedy nu urmăresc să determine toate soluțiile posibile, ci să aleagă dintre ele, conform criteriului de optimalitate impus, soluțiile optime. După cum spune și numele, algoritmii de tip greedy sunt caracterizați prin lăcomie și nu au răbdarea să investigheze toate variantele posibile de alegere a soluției. Ei încep construirea unei soluții pornind de la mulțimea vidă, apoi lucrează în pași, într-un mod cât se poate de hotărât: la fiecare pas se ia câte o decizie și se extinde soluția cu câte un element.
* La fiecare pas se analizează câte un element din mulțimea A și se decide dacă să fie sau nu inclus în mulțimea S care se construiește. Astfel se progresează de la Ø cu un șir de mulțimi intermediare (Ø, S0, S1, S2, ), până când se obține o soluție finală S.

Cap.II. Schema generală și implementare

Schema generală a unui algoritm greedy poate fi redată cu ajutorul unui ciclu:

* While existăElemente do
* Begin
* AlegeUnElement(x);
* includeElementul(x);
* End.

Funcția AlegeUnElement extrage un element din mulțimea A, un element x.

Funcția includeElementul înscrie elementul selectat în submulțimea S.

Pentru implementarea metodei greedy se vor folosi două subprograme:

→ subprogramul sort() -care ordonează mulțimea A după criteriul candidatului optim;

→subprogramul greedy() -care implementează metoda propriu-zisă;

Ca și schema generală de lucru, există doua variante de implementare a algoritmilor de tip Greedy.

* Prima variantă folosește două funcții caracteristice: ‘alege’ și ‘posibil’. ‘alege’ este o funcție care are rolul de a selecta următorul element din mulțimea A care să fie prelucrat, iar funcția ‘posibil’ verifică dacă un element poate fi adăugat soluției intermediare S(i) astfel încât noua soluție S(i+1) care s-ar obține să fie o soluție validă.

În continuare este prezentat pseudocodul pentru aceasta primă variantă greedy.

* Se consideră că numărul de elemente al mulțimii A este n.
* S = mulțimea vidă
* for (i=0; i<n; i++)
* Dificultatea la această primă variantă constă în scrierea funcției ‘alege’. Dacă funcția ‘alege’ este bine concepută, atunci putem fi siguri că soluția S găsită este o soluție optimă. Dacă funcția ‘alege’ nu este foarte bine concepută, atunci soluția S găsită va fi doar o soluție posibilă și nu va fi optimă. Ea se poate apropia însă mai mult sau mai puțin de soluția optimă S\*, în funcție de criteriul de selecție implementat.
* A doua variantă de implementare diferă de prima prin faptul că face o etapă inițială de prelucrare a mulțimii A. Practic se face o sortare a elementelor mulțimii A,conform unui anumit criteriu. După sortare, elementele vor fi prelucrate direct în ordinea rezultată.

În continuare este prezentat pseudocodul pentru această a doua variantă greedy.

* S = mulțimea vidă
* prelucrează(A)
* for (i=0; i<n; i++)

La a doua variantă, dificultatea funcției ‘alege’ nu a dispărut, ci s-a transferat funcției ‘prelucrează’. Dacă prelucrarea mulțimii A este bine făcută, atunci se va ajunge în mod sigur la o soluție optimă. Altfel se va obține doar o soluție posibilă, mai mult sau mai puțin apropiată de optim.

Cap.III. Resurse hardware și software necesare

Resurse de hard şi soft necesare aplicației a fost realizat cu ajutorul Code Blocks. Trebuie instalat pe un calculator care are minim următoarele componente hardware:

* Procesor (CPU) : 500 MHz +
* Memorie (RAM) : 32 MB
* Spaţiu liber pe hard-disk : 30 MB
* Rezoluție monitor :1024\*768
* sistem de operare : Windows 7

Cap.IV. Realizarea aplicației

Pașii algoritmului:

PAS1. Se inițializează mulțimea S cu mulțimea vidă S←Ø

PAS2. Cât timp S nu este soluție a problemei și A ≠Ø execută:

│PAS3. Se alege din mulțimea A elementul a care este candidatul optim al soluției.

│PAS4. Se elimină elementul a din mulțimea A.

│PAS5. Dacă el poate fi element al soluției, atunci elementul a se adaugă la mulțimea S. Se revine la PAS2.

PAS6. Dacă mulțimea S este soluția problemei, atunci se afișează soluția; astfel, se afișează mesajul “Nu s-a găsit soluție”

Implementarea strategiei Greedy în exemple de probleme:

Exemplul 1. Problema planificării optime a activitățiilor.

Strategia **greedy** implementată:

PAS1. Se inițializează vectorul **s** cu indicele primului element din vectorul **a**(se selectează, ca primă activitate, activitatea care are ora de terminare cea mai mică).

PAS2.**Pentru** următoarele **n-1** elemente ale vectorului a, **execută**

│PAS3. **Dacă** ora la care începe activitatea din elementul curent al vectorului **a** este mai mare sau egală cu ora la care se termină ultima activitate adăugată la soluție (în vectorul **s**), **atunci** activitatea este adăugată la soluție.

**Mulțimea activitatiilor**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **12** | **8** | **10** | **16** | **14** | **20** | **19** |
| **11** | **13** | **10** | **12** | **18** | **16** | **22** | **21** |

**Activitatea 1. 2. 3. 4. 5. 6. 7. 8.**

**Ora de începere**

**Ora de terminare**

**Mulțimea activităților – după ce a fost sortată**

**Activitatea 3. 1. 4. 2. 6. 5. 8. 7.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | **9** | **10** | **12** | **14** | **16** | **19** | **20** |
| **10** | **11** | **12** | **13** | **16** | **18** | **21** | **22** |

**Ora de începere**

**Ora de terminare**

**Soluția problemei**

**Activitatea 3. 4. 2. 6. 5. 8.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **8** | **10** | **12** | **14** | **16** | **19** |
| **10** | **12** | **13** | **16** | **18** | **21** |

**Ora de începere**

**Ora de terminare**

Exemplul 2. Problema ocupării optime a mijlocului de transport(problema rucsacului).

PAS1. Se inițializează greutatea **Gr** cu valoarea **G** și selectarea obiectelor începe cu obiectul cu cea mai mare eficiență a transportului (primul obiect din vectorul **a**).

PAS2. **Cât timp**  mai există obiecte care nu au fost selectate și greutatea obiectelor selectate nu este greutatea maximă, **execută**

|PAS3. **Dacă** greutatea obiectului din elementul curent al vectorului **a** este mai mică sau egală cu greutatea rămasă, **Gr**, **atunci** obiectul este adăugat la soluție, luându-se întreaga cantitate disponibilă din obiect; **astfel**, se ia din obiect fracțiunea egală cu greutatea rămasă **Gr**. Se actualizează greutatea **Gr** diminuând-o cu greutatea obiectului care a fost adăugat în rucsac.

**Mulțimea obiectelor – G=20**

**Obiectul 1. 2. 3. 4. 5.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | **4** | **4** | **8** | **10** |
| **10** | **20** | **10** | **10** | **22** |
| **2** | **5** | **2,5** | **1,25** | **2,2** |

**Greutatea**

**Profitul**

**Eficiența**

**Mulțimea obiectelor după ce a fost sortată**

**Obiectul 1. 2. 3. 4. 5.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **4** | **4** | **10** | **5** | **8** |
| **20** | **10** | **22** | **10** | **10** |
| **5** | **2,5** | **2,2** | **2** | **1,25** |

**Greutatea**

**Profitul**

**Eficiența**

**Soluția problemei**

**Obiectul 2. 3. 5. 1. 4.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **4** | **4** | **10** | **5** | **8** |
| **1** | **1** | **1** | **0,2** | **0** |
| **4** | **4** | **10** | **2** | **0** |

**Greutatea**

**Fracțiunea**

**Greutatea**

Complexitatea algoritmului:

Algoritmul greedy propriu-zis are complexitatea **O(n),** deoarece se parcurg cele n elemente ale mulțimii **A** pentru a alege fiecare dintre cele **m** elemente ale soluției. Algoritmul de sortare folosit pentru aranjarea elementelor mulțimii **A** după criteriul candidatului optim are complexitatea **O(n×log2n)** dacă se alege un algoritm de sortare eficient, cum este de explemplu algoritmul **QuickSort** . Complexitatea **agoritmului greedy** este **O(n)+O(n× log2(n))=O(n×log2(n))** .

Metoda **greedy** se recomandă în următoarele cazuri:

> Se dorește numai obținerea **soluției optime** și suntem **siguri** că aplicând strategia greedy se obține soluția optima;

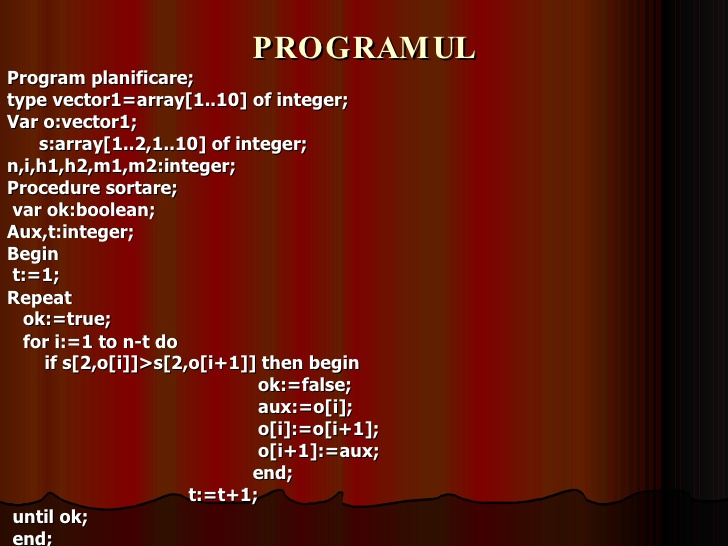
> Se dorește **obținerea unei soluții acceptabile,** nu neapărat optime, și algoritmul greedy este mult mai eficient decât alți algoritmi – care pot duce la obținerea mai multor soluții sau a soluției optime.

Cap.V. Probleme care se pot rezolva cu ajutorul metodei Greedy

* **Problema Spectacolelor:**

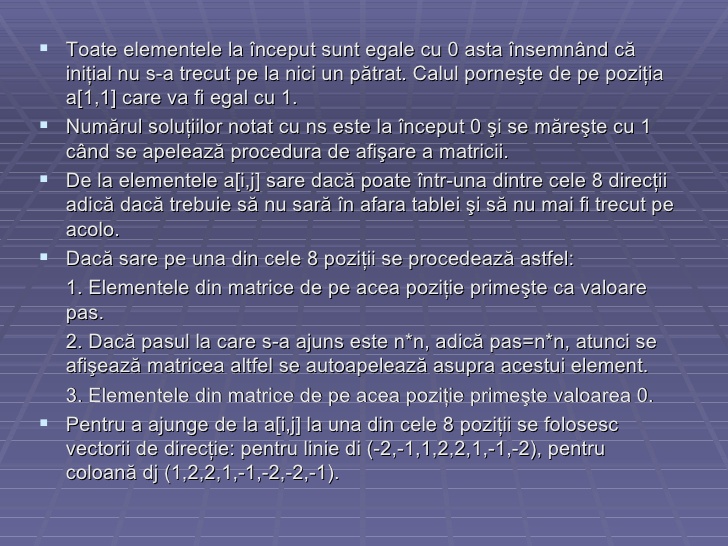
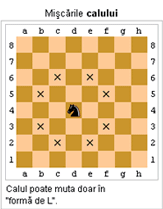
Managerul artistic al unui festival trebuie să selecteze o mulțime cât mai amplă de spectacole ce pot fi jucate într-o singură sală pe care o are la dispoziție. Știind că i s-au propus n spectacole și pentru fiecare spectacol i-a fost anunțat intervalul în care se poate desfașura [început,sfârșit]. Scrieți un program care să permită spectatorilor vizionarea unui numar cât mai mare de spectacole.





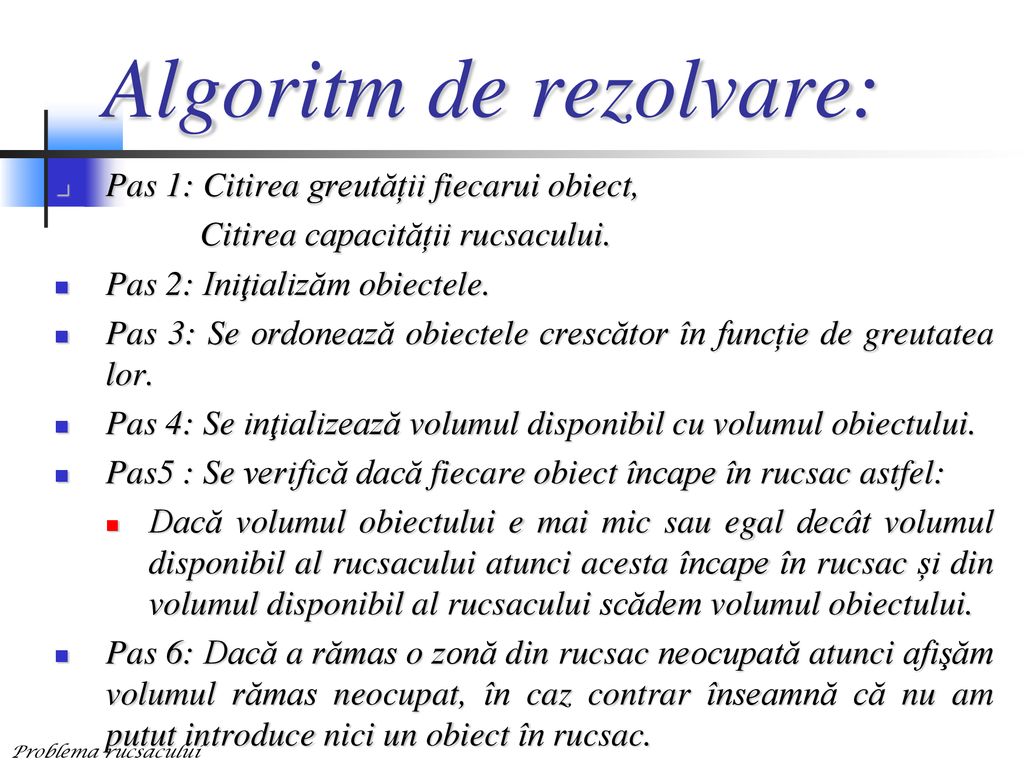
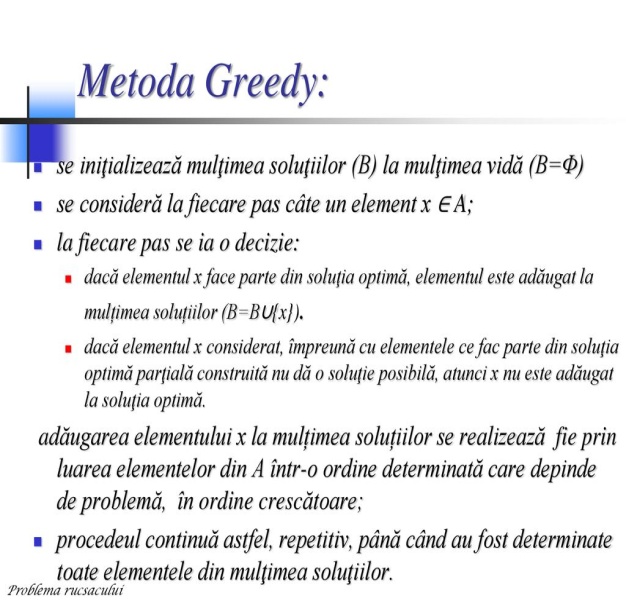
* **Problema Calului:**

Se dă o tablă de șah cu dimensiunea nxn. Un cal se găsește în linia 1 și coloana 1. Găsiți un șir de mutări ale calului astfel încât acesta să acopere întreaga tablă de șah fără a trece printr-o căsuță de 2 ori.



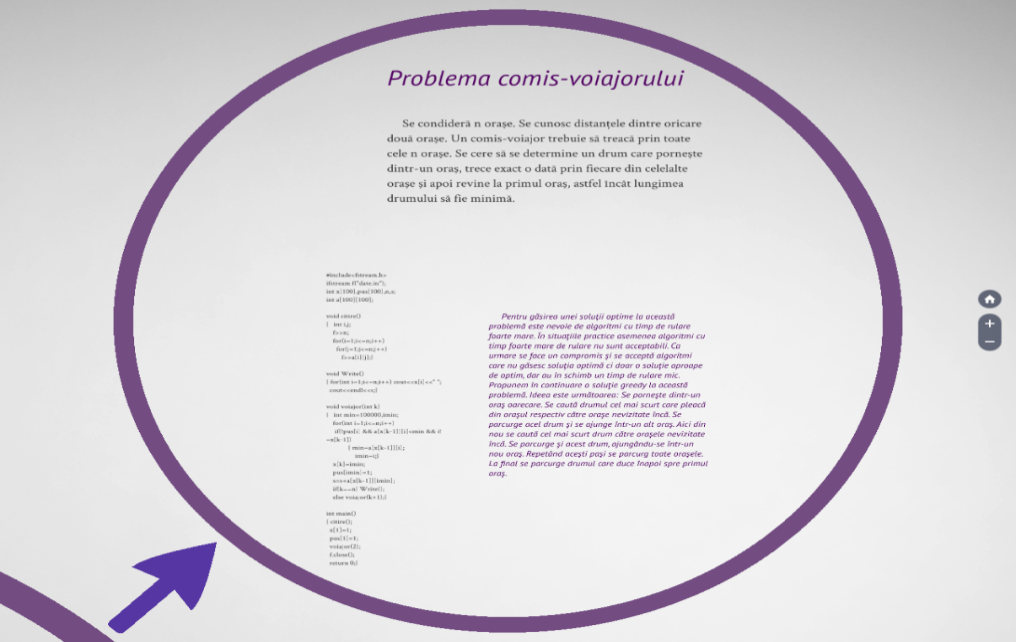
* **Problema Rucsacului:**

Se consideră că dispunem de un rucsac cu capacitatea M și de N obiecte, definite fiecare prin greutate și valoare, ce trebuie introduse în rucsac. Se cere o modalitate de a umple rucsacul cu obiecte , astfel încât valoarea totală să fie maximă. Este posibil ca oricâte obiecte și bucăți din obiecte sa fie introduse.



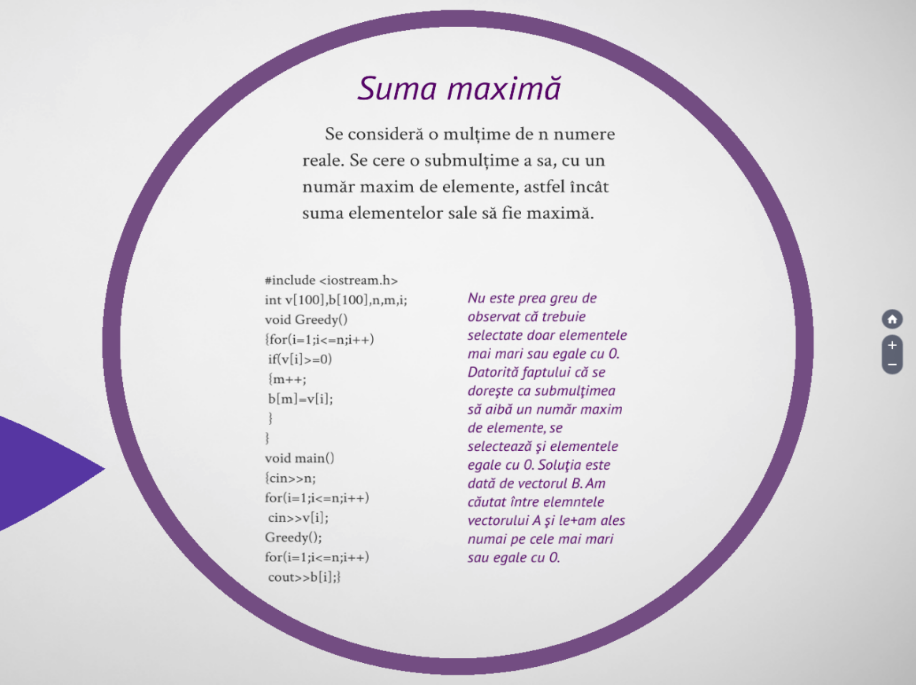
* **Problema Comisului-Voiajor:**

Un comis-voiajor pleacă dintr-un oraș, trebuie să viziteze un număr de orașe și să nu se întoarca în orașul de unde a plecat cu efort minim. Orice oraș este legat printr-o șosea i de orice alt oraș j printr-un drum de A[i,j] kilometri. Se cere traseul pe care trebuie să-l urmeze comisul-voiajor, astfel încât să parcurgă un număr minim de kilometri.



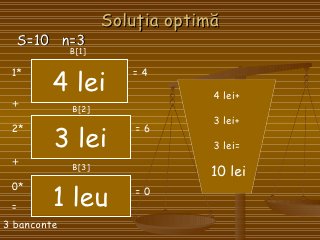
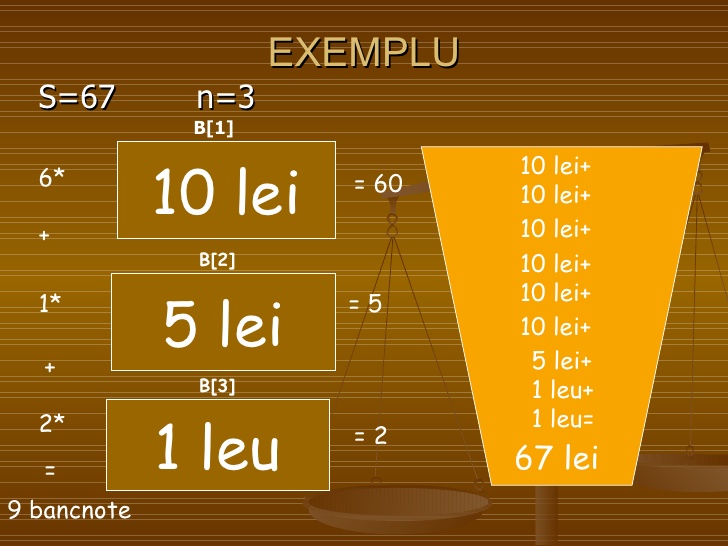
* **Problema Sumei Maxime:**

Se consideră o mulțime de n numere reale. Se cere o submulțime a sa cu un număr maxim de elemente, astfel încât suma elementelor sale să fie maximă.



* **Problema Banilor:**

Se efectuează plata unei sume S utilizând un număr minim de bancnote și monezi. Se cunoaște valoarea fiecărei bancnote și monezi. Să se găsească modul cel mai eficient în care să se achite suma cu un număr minim de bancnote sau monezi.



Cap.VI. Codul sursă

#include <bits/stdc++.h>

using namespace std;

constint x[8]={-1,1,2,2,1,-1,-2,-2};

constint y[8]={2,2,1,-1,-2,-2,-1,1};

float A[100],B[100];

double c[9],g[9],ef[9],gv,mani,castig;

int t[50][50],a[100],Z[30],Mutari,E,man,inv,vs1,ora,mint;

int cost,man1;

int i,j,n,p,v,s,m,h1,m1,h2,m2,vs;

void Mut(intl,int c);

int Numar(intl,int c);

void Greedy();

void sortare();

void Sort(intk,int X[20]);

void parasire\_program();

int problema\_spectacolelor()

{cout<<"n=";

cin>>n;

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

{ a[i]=i;

cout<<"ora de inceput pentru spectacolul"<<i<<" (hh mm)=";

cin>>h1>>m1;

t[0][i]=h1\*60+m1;

cout<<"ora de sfarsit pentru spectacolul"<<i<<" (hh mm)=";

cin>>h2>>m2;

t[1][i]=h2\*60+m2;}

sortare();

cout<<"ordinea spectacolelor este "<<endl<<a[1]<<endl;

ora=t[1][a[1]];

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

if(t[1][a[1]]>=ora)

{cout<<a[i]<<endl;

ora=t[1][a[i]]; }

return 0; }

int problema\_rucsacului()

{cout<<"Greutatea ce poate fi transportata=";

cin>>gv;

cout<<"Numar de obiecte=";

cin>>n;

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

{

cout<<"c["<<i<<"]=";cin>>c[i];

cout<<"g["<<i<<"]=";cin>>g[i];

a[i]=i;

ef[i]=c[i]/g[i];

}

do{inv=0;

for(i=1;i<=n-1;i++)

if(ef[i]<ef[i+1])

{

mani=ef[i];

ef[i]=ef[i+1];

ef[i+1]=mani;

mani=c[i];

c[i]=c[i+1];

c[i+1]=mani;

mani=g[i];

g[i]=g[i+1];

g[i+1]=mani;

inv=1;

man1=a[i];

a[i]=a[i+1];

a[i+1]=man1;

}

}while(inv);

i=1;

while(gv>0&&i<=n)

{if(gv>g[i])

{

cout<<"obiectul"<<a[i]<<' '<<1<<endl;

gv-=g[i];

castig+=+c[i];

}

else

{cout<<"obiectul"<<a[i]<<' '<<gv/g[i]<<endl;

castig+=c[i]\*gv/g[i];

gv=0; }

i++;}

cout<<"Castig total="<<castig;

return 0;}

int problema\_calului()

{cout<<"Dati dimensiunea tablei de sah: ";

cin>>n;

t[1][1]=1;

Mut(1,1);

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

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

cout<<t[i][j]<<" ";

cout<<endl;}

return 0;}

int problema\_comisului\_voiajor()

{cout<<"Numar noduri=";

cin>>n;

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

for(j=i+1;j<=n;j++)

cout<<"t["<<i<<"]["<<j<<"]=";

cin>>t[i][j];

t[j][i]=t[i][j];

cout<<"Nod de pornire:";

cin>>v;

a[v]=1;

vs1=v;

cout<<"Drumul trece prin:"<<v<<' ';

p=v;

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

{ mint=3000;

for(j=1;j<=n;j++)

if(t[v][j]!=0&&t[j]==0&&mint>t[v][j])

{ mint=t[v][j];

vs=j; }

cost+=t[v][vs];

cout<<vs<<' '; a[vs]=1; v=vs; }

cout<<p;

cost+=t[vs1][v];

cout<<endl<<"Cost="<<cost;

return 0; }

int problema\_banilor()

{cout<<"Suma=";

cin>>s;

cout<<"Numarul de bancnote: ";

cin>>n;

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

{cout<<"a["<<i<<"]=";

cin>>a[i];}

do{inv=0;

for(i=1;i<=n-1;i++)

if(a[i]<a[i+1])

{man=a[i];

a[i]=a[i+1];

a[i+1]=man;}}while(inv);

i=1;

cout<<s<<endl;

do{if(s/a[i]>0)

cout<<s/a[i]<<" bancnote cu valoarea "<<a[i]<<endl;

s=s%a[i];

i++; }while(i<=n);

return 0;}

int problema\_sumei\_maxime()

{

cout<<"n=";

cin>>n;

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

{

cout<<"A["<<i<<"]=";

cin>>A[i];}

Greedy();

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

cout<<a[i]<<" ";

return 0;}

int problema\_de\_maxim()

{cout<<"m=";

cin>>m;

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

cin>>Z[i];

cout<<"n=";

cin>>n;

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

cin>>a[i];

Sort(m,Z);

Sort(n,a);

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

E+=Z[i]\*a[n-m+i];

cout<<" Emax="<<E;

return 0;}

int Numar(int l,int c)

{int nr=0,i;

for(i=0;i<=7;i++)

if(l+x[i]>=1&&

l+x[i]<=n&&

c+y[i]>=1&&

c+y[i]<=n&&

t[l+x[i]][c+y[i]]==0)

nr++;

return nr;}

void Mut(int l,int c)

{ t[l][c]=Mutari+1;

gasit=0; min=9;

for(i=0;i<=7;i++)

if(l+x[i]>=1&&

l+x[i]<=n&&

c+y[i]>=1&&

c+y[i]<=n&&

t[l+x[i]][c+y[i]]==0)

{ v=Numar(l+x[i],c+y[i]);

if(v<min)

{ min=v;

linie=l+x[i];

coloana=c+y[i];

gasit=1;}}

if(gasit)

{Mutari++;

Mut(linie,coloana);}}

void sortare()

{ ta=1;

for(i=1;i<=n-1;i++)

if(t[1][a[i]]>t[1][a[i+1]])

{ m=a[i];

a[i]=a[i+1];

a[i+1]=m;

gata=0; }

}while(!gata); }

void Sort(int k,int X[20])

{

int inversari,man;

do

{

inversari=0;

for(i=1;i<=k-1;i++)

if(X[i]>X[i+1])

{

man=X[i];

X[i]=X[i+1];

X[i+1]=man;

inversari=1;

}

}while(inversari);

}

void Greedy()

{

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

if(A[i]>=0)

{

m++;

a[m]=A[i];

}

}

void parasire\_program()

{

cout<<" Am parasit programul ";

}

int main()

{

char selector;

do{ cout<<"Introduceti comanda: \n";

cout<<" 'p' pentru problema spectacolelor\n";

cout<<" 'r' pentru problema rucsacului\n";

cout<<" 'c' pentru problema calului\n";

cout<<" 'v' pentru problema comisului-voiajor\n";

cout<<" 'b' pentru problema banilor\n";

cout<<" 's' pentru problema sumei maxime\n";

cout<<" 'm' pentru problema de maxim\n";

cout<<" 'x' pentru parasirea meniului\n";

cin.get(selector);

cin.clear();

cin.ignore(INT\_MAX,'\n');

switch(selector)

{case 'p': problema\_spectacolelor();

cin.get();

break;

case 'r': problema\_rucsacului();

cin.get();

break;

case 'c': problema\_calului();

cin.get();

break;

case 'v': problema\_comisului\_voiajor();

cin.get();

break;

case 'b': problema\_banilor();

cin.get();

break;

case 's': problema\_sumei\_maxime();

cin.get();

break;

case 'm': problema\_de\_maxim();

cin.get();

break;

case 'x': parasire\_program();

cin.get();

break;

default: cout<<"COMANDA INVALIDA"<<endl;

}}while(selector!='x');

return 0;}

Cap. VII. Bibliografie

* Manual pentru clasa a XI-a, Informatică intensiv C++, Filiera teoretică, profilul real, specializarea matematică-informatică, intensiv informatică, Mariana Miloșescu
* <https://ro.m.wikipedia.org/wiki/Algoritm_greedy>
* <https://sites.google.com/site/eildegez/home/clasa-xi/prezentarea-metodei-greedy>
* <http://www.worldit.info/articole/algoritmica-articole/metoda-greedy/>
* <https://forum.b-zone.ro/topic/373320-c-metoda-greedy/>