

Metoda Backtracking. Algoritmul greedy.

1. Obiectivele lucrării

În această lucrare se vor studia aspecte legate de metoda backtracking și algoritmul greedy. Se vor prezenta probleme tipice în care se aplică aceste metode.

2. Breviar teoretic

Metoda backtracking se aplică la probleme în care soluția se poate reprezenta sub forma unui vector. Dacă notăm acest vector cu X ($X = X_0, X_1, X_2, \dots, X_{n-1}$), atunci fiecare componentă X_k a soluției poate să ia valori într-o mulțime finită S_k .

În general, în problemele în care se aplică această metodă, se cer fie toate soluțiile posibile (fiecare soluție fiind un vector cu N componente), fie o singură soluție: cea mai bună, după un anumit criteriu.

Sunt două variante de implementare a acestor metode:

1. Varianta iterativă
2. Varianta recursivă (mai ușor de implementat).

Prin **metoda backtracking iterativă**, orice vector soluție este construit progresiv, începând cu prima componentă și mergând către ultima, cu eventuale reveniri. Odată, stabilită valoarea curentă atribuită lui X_k , din mulțimea de valori S_k , nu se trece direct la atribuirea unei valori următoarei componente X_{k+1} , ci se verifică dacă soluția parțială găsită ($X_0, X_1, X_2, \dots, X_k$) este bună. Dacă este bună, se trece la următoarea atribuire pentru X_{k+1} . Dacă nu este bună, se revine asupra lui X_k încercându-se o nouă atribuire din mulțimea valorilor necunoscute din S_k . Dacă toată mulțimea S_k este consumată (toate valorile posibile din S_k au fost atribuite lui X_k), se revine cu noi atribuiri componente anterioare X_{k-1} . După ce s-a atribuit și ultimei componente X_{n-1} o valoare, se afișează soluția găsită (vectorul X) și apoi procesul se reia, încercând o nouă atribuire din mulțimea de valori posibile rămase, pentru ultima componentă.

Astfel, aplicând această metodă pentru a genera toate permutările multimii $\{1,2,3\}$, se obțin, în ordine, următoarele soluții:

{1,2,3}

{1,3,2}

{2,1,3}

{2,3,1}

{3,1,2}

{3,2,1}

Prin **metoda backtracking recursivă** se construiește complet vectorul soluție (toate cele n componente) și în final se verifică dacă este o soluție validă. Verificarea validității nu se face după fiecare nouă componentă adăugată, ca la metoda backtracking iterativă, ci doar în final, când au fost construite toate cele n componente. Generarea soluțiilor prin metoda recursivă consumă mai mult timp decât prin metoda iterativă, însă varianta recursivă se transpune mai ușor în limbajul de programare.

Algoritmul greedy (greedy = lacom) este un concept folosit pentru a desemna o mulțime finită de operații, complet ordonată în timp, care pornind de la date de intrare produce într-un timp finit date de ieșire. Cu alte cuvinte, algoritmul redă metoda de rezolvare a unei probleme într-un număr finit de pași. Strategia Greedy este strategia în care optimul local se consideră, optim general. Este o strategie constructivă prin adăugarea treptată de soluții locale care construiesc soluția finală. Așadar, la Greedy se cere o mulțime a datelor de intrare care să îndeplinească niște condiții. Dacă sunt mai multe soluții posibile metoda găsește optimul (orice cale spre soluție se acceptă imediat).

Observație:

Algoritmul de tip greedy nu garantează obținerea celei mai bune soluții.

3. Probleme rezolvate

Se vor edita și apoi executa programele descrise în continuare.

1. Afișarea tuturor permutărilor mulțimii $A = \{1, 2, 3, \dots, N\}$. Rezolvare folosind metoda backtracking **recursivă**.

$$P_n = n!$$

Sursa programului:

```
#include <stdio.h>
```

```
#include <conio.h>
#include<iostream.h>
#define N 3 //dimensiunea vectorului de elemente a
int a[N]={10,5,15};
int x[N]; //vectorul solutie

int esteSol(int k)
{
    //x[0], ..., x[k-1] sunt valide. Adaugam si x[k].
    //Pentru a fi valida trebuie ca x[k] sa fie diferit
    //de celelalte componente ale solutiei.
    int i;
    for(i=0;i<=k-1;i++)
        if(x[k]==x[i]) return 0;
    return 1;
}

void afisare()
{
    cout<<endl;
    for(int i=0;i<N;i++)
        cout<<a[x[i]]<<" ";
    cout<<endl;
}

void bk(int k)
{
    if(k==N)
        afisare();
    else
        for(int i=0;i<N;i++)
        {
            x[k]=i;
            if(esteSol(k))
                bk(k+1);
        }
}

void main()
{
    clrscr();
    bk(0);
    getch();
}
```

2. Afișarea tuturor permutărilor mulțimii $A = \{1, 2, 3, \dots, N\}$.
Rezolvare folosind metoda backtracking **iterativă**.

$$P_n = n!$$

Sursa programului:

```
#include <stdio.h>
#include <conio.h>

#define N 3 //numarul de componente
int X[N]; //X[k]=1 sau 2 sau 3 sau ... sau N

//prototipuri functii
void afisare();
int esteSol(int k); //testeaza daca elementul X[k]
//este bun pentru solutia finala
void back();

void main()
{
    clrscr();
    back();
    getch();
}

void back()
{
    //Initializare vector X cu solutia vida
    int i,k;
    for(i=0;i<N;i++)X[i]=0;
    k=0;
    while (k>=0){
        //s-a completat vectorul X?
        if(k==N) //s-a construit o solutie
        {
            afisare();
            k=N-1;//se revine la utima
            //componenta
        }
        else{
            //Nu s-a completat. Sunt 2 posibilitati: sunt / nu
            //sunt valori disponibile pt. X[k]
            if(X[k]+1<=N){
                //sunt valori disponibile.
```

```
        X[k]=X[k]+1;
        if(esteSol(k)==1) k++; //da
//valori urmatoarei componente
//daca nu este buna se ramane tot la X[k]
    }
    else{
//daca nu mai sunt valori disponibile, face pasul
//inapoi:
        X[k]=0; //se atribuie lui
//X[k] valoarea imposibila - valoarea de start
        k=k-1;}
    } //else
} //while
} //back()

int esteSol(int k)
{
    int i;
    for(i=0; i<k; i++) //componenta X[k] este buna
daca este diferita de X[0], de //X[1], de ..., de
X[k-1].
        if(X[k]==X[i]) return 0; //nu este buna
    return 1;
}

void afisare()
{
    int i;
    for(i=0; i<N; i++)
        printf("%d ", X[i]);
    printf("\n");
}
```

3. Se citește un cuvânt de la tastatură. Să se afișeze toate anagramările posibile ale cuvântului.

Observație: În unele situații, mulțimea X construită prin metoda backtracking, nu este soluția propriu-zisă, ci conține doar indecșii soluției. Această aplicație exemplifică acest lucru.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <string.h>
```

```
//variabile globale
int N;//lungimea cuvantului tastat
int x[20];//indecsii solutiei, deci X[k] este din
//multimea {0,1,...,N-1}
//vectorul X este o permutare a numerelor 0, 1, N-1
char cuvant[21];//cuvantul citit de la tastatura

int esteSol(int k)
{
    int i;
    for(i=0;i<=k-1;i++)
        if(x[k]==x[i]) return 0;//nu sunt
//distincte
//daca se ajunge aici atunci vectorul X contine o
//permutare valida
    return 1;
}

void afisare()
{
    cout<<endl;
    for(int i=0;i<N;i++)
        cout<<cuvent[x[i]];
    cout<<endl;
}

void bk(int k)
{
    if(k==N)
        afisare();
    else
    {
        for(int i=0;i<N;i++)
        {
            x[k]=i;
            if(esteSol(k))
                bk(k+1);
        }
    }
}

void main()
{
    clrscr();
```

```
        cout<<"Tastati un cuvant:"<<endl;
        gets(cuvant);
        N=strlen(cuvant);
        bk(0);
        getch();
    }
```

4. Afișarea tuturor aranjamentelor mulțimii $A = \{1, 2, 3, \dots, N\}$.
Rezolvare folosind metoda backtracking **recursivă**.

$$A_n^k = \frac{n!}{(n-k)!}$$

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>

#define N 3 //dimensiunea vectorului de elemente a
int p; //nr. de elemente din grupa, luate din cele
//N ale multimii a
int a[N]={10,5,15};
int x[N]; //vectorul solutie

int esteSol(int k)
{
    //x[0], ..., x[k-1] sunt valide. Adaugam si x[k].
    //Pentru a fi valida trebuie ca x[k] sa fie diferit
    //de celelalte componente ale solutiei.
    int i;
    for(i=0;i<=k-1;i++)
        if(x[k]==x[i]) return 0;
    return 1;
}

void afisare()
{
    cout<<endl;
    for(int i=0;i<p;i++)
        cout<<a[x[i]]<<" ";
    cout<<endl;
}

void bk(int k)
{

```

```

        if (k==p)
            afisare();
        else
            for (int i=0; i<N; i++)
            {
                x[k]=i;
                if (esteSol(k))
                    bk(k+1);
            }
    }

void main()
{
    clrscr();
    cout<<"p=";
    cin>>p;
    bk(0);
    getch();
}

```

5. Afișarea tuturor combinațiilor mulțimii $A = \{1, 2, 3, \dots, N\}$.
Rezolvare folosind metoda backtracking **recursivă**.

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Sursa programului:

```

#include <stdio.h>
#include <conio.h>
#include <iostream.h>

#define N 3 //dimensiunea vectorului de elemente a
int p; //nr. de elemente din grupa, luate din cele
//N ale multimii a
int a[N]={10,5,15};
int x[N]; //vectorul solutie

int esteSol(int k)
{
    //x[0], ..., x[k-1] sunt valide. Adaugam si x[k].
    //Pentru a fi valida trebuie ca x[k] sa fie diferit
    //de celelalte componente ale solutiei.
    int i;
    for (i=0; i<=k-1; i++)
        if (x[k]<=x[i]) return 0;
}

```

```
        return 1;
    }

    void afisare()
    {
        cout<<endl;
        for(int i=0;i<p;i++)
            cout<<a[x[i]]<<" ";
        cout<<endl;
    }

    void bk(int k)
    {
        if(k==p)
            afisare();
        else
            for(int i=0;i<N;i++)
            {
                x[k]=i;
                if(esteSol(k))
                    bk(k+1);
            }
    }

    void main()
    {
        clrscr();
        cout<<"p=";
        cin>>p;
        bk(0);
        getch();
    }
```

6. Fiind dată o hartă cu N țări, se cer toate soluțiile de colorare a hărții, utilizând cel mult 4 culori, astfel încât oricare două țări, ce au frontiera comună, să fie colorate diferit.

Observație: Pentru memorarea informațiilor despre frontierele comune dintre oricare două țări, folosim ca structură de date, matricea esteFrontiera. Astfel: esteFrontiera[i][j]=1 – dacă există frontieră între țara i și țara j. Dacă nu există, esteFrontiera [i][j]=0.

Sursa programului:

```
#include<stdio.h>
```

```
#include<conio.h>
#include<iostream.h>
#include<string.h>

#define NR_TARI 6//numarul de tari
#define NR_CULORI 4
typedef char stringCuloare[10];

//variabile globale
int esteFrontiera[NR_TARI][NR_TARI];
stringCuloare
culori[NR_CULORI]={"rosu","galben","albatru","verde"};
stringCuloare x[NR_CULORI]; //vectorul solutie X
//este un vector de stringuri

int esteSol()
{
    int i,j;
    for(i=0;i<NR_TARI;i++)
        for(j=i+1;j<NR_TARI;j++)
            if(esteFrontiera[i][j]==1)//daca
//tarile i si j au granita comuna
            if(strcmp(x[i],x[j])==0)//daca au
//aceeasi culoare
                return 0;//fals, nu este
//solutie
//daca se ajunge aici atunci avem o permutare valida
    return 1;
}

void afisare()
{
    cout<<endl;
    for(int i=0;i<NR_TARI;i++)
        cout<<i<<" = "<<x[i];
    cout<<endl;
}

void BkRecurziv(int k)
{
    if(k==NR_TARI)
    {
        if(esteSol()){afisare();}
```

```

    }
    else
    {
        for(int i=0;i<NR_CULORI;i++)
        {
            strcpy(x[k],culori[i]);
            BkRecurziv(k+1);
        }
    }
}

void main()
{
    int i,j;    clrscr();
    //citire informatii despre frontiere intre tari
    for(i=0;i<NR_TARI;i++)
        for(j=i+1;j<NR_TARI;j++){
            cout<<"Este frontiera intre "<<i<<"
            si "<<j<<"? 0 - nu ; 1 - da";
            cin>>esteFrontiera[i][j]; }
    //datorita simetriei, nu a fost necesara decat
    initializarea pe jumătate a matricii esteFrontiera
    BkRecurziv(0);
    getch();
}

```

7. Se dau N tipuri de monede. Să se plătească o sumă dată S, folosind un număr total de monede, minim. Se consideră că există un număr suficient de monezi din fiecare tip.

Sursa programului:

```

#include<stdio.h>
#include<conio.h>
#include<iostream.h>

#define N 4//numarul de monezi distincte
int S;//suma data
int valoareMonezi[N]={100,50,20,10};//prima moneda
//are valoarea 100, a doua 50, s.a.m.d.
int x[N];//solutia curenta. Astfel x[0]=numar de
//monezi luate din tipul //valoareMoneda[0]
int xOptim[N];
int nrminmonezi;
int semafor=0;//sa vad daca suma se poate calcula
//exact cu monezile pe care le avem la dispozitie

```

```
int esteSolutie()
{
    int i, sumacrt;
    //este solutie? Este daca se realizeaza plata
    //exacta a sumei S cu vectorul x
    sumacrt=0;
    for(i=0;i<N;i++)
        sumacrt=sumacrt+x[i]*valoareMonezi[i];
    if(sumacrt==S) return 1; //se realizeaza plata
    //exacta
    return 0;
}

void BkRecurziv(int k)
{
    int i, nrcrtmonezi;
    if(k==N) {
        if(esteSolutie()) {
            //comparare cu cea mai buna solutie
            nrcrtmonezi=0;
            for(i=0;i<N;i++) nrcrtmonezi+=x[i];
            if(nrcrtmonezi<nrminmonezi) {
                //solutia curenta este mai buna
                nrminmonezi=nrcrtmonezi;
                for(i=0;i<N;i++) xOptim[i]=x[i];
            }
        }
    }
    else
        for(i=0;i<=S/valoareMonezi[k];i++)
            { //i parcurge valorile posib. ale componentei x[k]
                x[k]=i;
                BkRecurziv(k+1);
            }
}

void main()
{
    int i;
    clrscr();
    cout<<"Introduceti valoarea sumei de bani:";
    cin>>S;
    //initializare nrminmonezi cu cel mai mare
    //numar de monezi cu care
```

```

        //se poate plati suma S
        //presupunem, pentru simplificarea codului, ca
        //cea mai mica moneda
        //este ultima din vectorul valoareMoneda
        nrminmonezi=S/valoareMonezi[N-1];
        //initializarea vectorului solutieOptima
        xOptim[N-1]=nrminmonezi;
        for (i=0;i<N-1;i++) xOptim[i]=0;
        BkRecurziv(0);
        printf("\nNr minim de monezi este: %d\n",
nrminmonezi);
        for (i=0;i<N;i++){
            cout<<endl<<xOptim[i]<<" monezi";
            cout<<" de valoare "<<valoareMonezi[i];
        }
        getch();
    }
}

```

8. Aceeași problemă ca la punctul precedent - (problema nr. 7) – utilizând însă un algoritm de tip Greedy.

Soluție:

- Se începe prin ordonarea descrescătoare a vectorului monezi.
- Se vede câte monezi din cea mai mare valoare se pot utiliza.
- Se merge descrescător până se completează toată suma.

Observație: Această tehnică, pentru acest tip de problemă, nu garantează găsirea celei mai bune soluții.

Ex:

Suma=111 – trebuie plătită având la dispoziție monezi de 100, 55, 1.

Utilizând **algoritmul greedy** obținem soluția: $S = 1 \cdot 100 + 11 \cdot 1 = 12$ monezi.

Utilizând **metoda Backtracking** obținem soluția optimă: $S = 2 \cdot 55 + 1 \cdot 1 = 3$ monezi.

Sursa programului:

```

#include<stdio.h>
#include<conio.h>
#include<iostream.h>

#define N 3//numarul de monezi distincte
int S;//suma data

```

```

int valoareMonezi[N]={100,55,1};//prima moneda are
//valoarea 100, a doua 50, s.a.m.d.
int solutie[N];

void main()
{
    int i;
    clrscr();
    cout<<"Introduceti valoarea sumei de bani:";
    cin>>S;
    for(i=0;i<N;i++)
    {
        solutie[i]=S/valoareMonezi[i];
        S=S-solutie[i]*valoareMonezi[i];
    }
    for(i=0;i<N;i++)
        cout<<endl<<" Din moneda
"<<valoareMonezi[i]<<" s-au folosit
"<<solutie[i]<<" bucati";
    getch();
}

```

9. Următoarea aplicație este o problemă clasică de backtracking, fiind cunoscută sub numele de “*Problema celor 8 regine*”. Se cere să se afișeze toate posibilitățile de a aranja 8 regine pe o tablă de șah, astfel încât să nu se atace între ele. Se va afișa și numărul total de posibilități.

Modurile in care pot fi asezate pe tabla de sah 8 regine fara a se ataca :

R	-	-	-	-	-	-	-
-	-	-	-	R	-	-	-
-	-	-	-	-	-	-	R
-	-	-	-	-	R	-	-
-	-	R	-	-	-	-	-
-	-	-	-	-	-	R	-
-	R	-	-	-	-	-	-
-	-	-	R	-	-	-	-

Posibilitatea 1_

Observație: Este evident că pentru a nu se ataca, fiecare regină trebuie plasată într-o altă linie (nu pot fi 2 regine pe aceeași linie). De aceea vom afișa doar numerele coloanelor. Vom numerota cele 8 coloane ale tablei de șah cu: 0,1,2,3,4,5,6,7.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>

#define stsus 0xda
#define oriz 0xc4
#define drsus 0xbf
#define drjos 0xd9
#define stjor 0xc0
#define vert 0xb3

int v[8], m=0; // m este contorul pentru numarul de
// solutii gasite
void charxy(int x, int y, int c);
void linieh(int x1, int x2, int y, int c);
void liniev(int y1, int y2, int x, int c);
void drepts(int x1, int y1, int x2, int y2);
void determ(int k);

void charxy(int x, int y, int c)
{
    gotoxy(x, y);
    printf("%c", c);
}

void linieh(int x1, int x2, int y, int c)
{
    int i;
    for (i=x1; i<=x2; i++)
        charxy(i, y, c);
}

void liniev(int y1, int y2, int x, int c)
{
    int i;
```

```
        for (i=y1;i<=y2;i++)
            charxy(x,i,c);
    }

void drepts(int x1,int y1,int x2,int y2 )
{
    int i;
    charxy(x1,y1,stsus);
    linieh(x1+1,x2-1,y1,oriz);
    for (i=1;i<=7;i++)
        linieh(x1+1,x2-1,y1+2*i,oriz);
    charxy(x1,y2,stjos);
    linieh(x1+1,x2-1,y2,oriz);
    charxy(x2,y1,drsus);
    liniev(y1+1,y2-1,x1,vert);
    for (i=1;i<=7;i++)
        liniev(y1+1,y2-1,x1+5*i,vert);
    charxy(x2,y2,drjos);
    liniev(y1+1,y2-1,x2,vert);
}

int afisare(void)
{
    int l,c;
    char p;
    for (c=0;c<8;c++)
        for (l=0;l<8;l++)
        {
            gotoxy(20+5*l,5+2*c);
            if (v[c]==l)
                printf("R");
            else
                printf("-");
        }
    gotoxy(10,22);
    m++;
    printf("Posibilitatea %d",m);
    getch();
    p=getch();
    if (p=='x') exit(0);
    return 0;
}
```

```
int detect(int k)
{
    int q,y;
    y=1;
    for(q=0;q<k&& y!=0;q++)
        y=(abs(q-k)!=abs(v[q]-v[k]))&&(v[q]!=v[k]);
    if(y&& k==7)
        return afisare();
    else
        return y;
}

void determ(k)
{
    int i;
    for(i=0;i<8;i++)
    {
        v[k]=i;
        if (detect(k))
            determ(k+1);
    }
}

void main(void)
{
    int k;
    clrscr();
    printf("Modurile in care pot fi asezate pe
tabla de sah 8 regine \nfara a se ataca :");
    k=0;
    drepts(17,4,17+5*8,4+2*8);
    determ(k);
    printf("\nNr total de posibilitati
este:%d",m);
    getch();
}
```

10. Se citește o matrice pătrată de dimensiune M cunoscută. Să se calculeze și afișeze suma maximă ce constă din M valori, fiecare valoare fiind luată din matrice din linii și coloane diferite.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
```

```
#include<iostream.h>

#define N 3//dimensiune matrice
int A[N][N]={2,5,1,
             1,3,6,
             4,5,2};
int sumaMax;
int x[N];//pentru rezolvare propunem utilizarea
unui vector cu N componente x[N]=0,1,2,...N si
//facem toate permutarile posibile pentru acest
//vector

void calcul_maxim()
{
    int suma=0;
    for(int i=0;i<N;i++)
        suma+=A[x[i]][i];
    if(suma>sumaMax) sumaMax=suma;
}

int esteSolutie(int k)
{
    for(int i=0;i<=k-1;i++)
        if(x[k]==x[i]) return 0;
    return 1;
}

void BkRecurziv(int k)
{
    if(k==N)
        calcul_maxim();
    else
        for(int i=0;i<N;i++)
        {
            x[k]=i;
            if(esteSolutie(k))
                BkRecurziv(k+1);
        }
}

void main()
{
    int i;
    clrscr();
```

```

    sumaMax=0;
    BkRecursiv(0);
    cout<<"Maximul este: "<<sumaMax;
    getch();
}

```

4. Probleme propuse

1. Să se afișeze toate configurațiile posibile în care 8 ture așezate pe tabla de șah nu se atacă. De asemenea, să se afișeze și numărul total de soluții găsite.
2. Fie N și p două constante definite cu directiva `#define` și $p \leq N$. Se citesc N caractere de la tastatură. Să se afișeze toate șirurile formate din p caractere dintre cele N citite știind că trebuie respectate simultan următoarele restricții:
 - a. Nu pot fi două vocale alăturate.
 - b. Nu trebuie să fie nici un caracter cifră.
3. Se dă un vector A cu N componente numere întregi. Să se determine o submulțime - un subșir al lui A , ale cărui elemente sunt luate în ordine dintre componentele vectorului A – dar nu obligatoriu elementele consecutive din A – de lungime maximă, ale cărui componente sunt în ordine crescătoare.
Ex.: Dacă $A=\{4, 3, 1, 7, 2, 4, 5\}$, atunci submulțimea căutată este: $B=\{1, 2, 4, 5\}$, deci lungimea maximă este 4.
4. Să se afișeze toate posibilitățile de a realiza perforarea unui bilet de autobuz și numărul total de posibilități, pornind de la următoarea figură:

0	1	2
x	x	x
3	4	5
x	x	x
6	7	8
x	x	x

$X[k]=0$ dacă nu este perforat locul k

$X[k]=1$ dacă este perforat locul k .

5. Se dorește construirea unei țevi de lungime totală L știind că avem la dispoziție N tipuri de țevi. Toate țevile de tipul i au

aceeași lungime notată cu li . Pentru fiecare tip de țevă avem la dispoziție un anumit număr de bucăți, notat cu $bucăți[i]$.

- a. Țeava de lungime L trebuie obținută prin folosirea cel puțin a unei țevi din fiecare tip. Să se afișeze toate variantele de obținere a țevii de lungime L .

Ex.:

$L=100m$ tip 0: $l[0]=10$ tip 1: $l[1]=20$

tip 2: $l[2]=50$

$N=3$ $bucăți[0]=5$ $bucăți[1]=4$

$bucăți[2]=2$

Răspuns: $100 = 3*10 + 1*20 + 1*50 = 1*10 + 2*20 + 1*50$

- b. Țeava de lungime L trebuie obținută prin folosirea unui număr cât mai mic de țevi. Să se afișeze toate variantele de obținere a țevii de lungime L .
- c. Țeava de lungime L trebuie obținută prin folosirea unui număr cât mai mic de țevi. Să se afișeze soluția obținută prin algoritmul Greedy.
-