Metode de programare

Sumar

Competenţe generale

- *elaborarea algoritmilor de rezolvare a problemelor*
- *implementarea algoritmilor într-un limbaj de programare*

Competenţe specifice

- *analiza problemei în scopul identificării metodei de programare adecvate pentru rezolvarea problemei*
- *aplicarea creativă a metodelor de programare pentru rezolvarea unor probleme intradisciplinare sau interdisciplinare, sau a unor probleme cu aplicabilitate practică*
- *analiza comparativă a eficienţei diferitelor metode de rezolvare a aceleiaşi probleme şi alegerea unui algoritm eficient de rezolvare a unei probleme*
- elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării
- utilizarea tehnicilor moderne în implementarea aplicaţiilor

2. Descrierea generală a metodei

Metoda Greedy[1] (metoda optimului local) este o metodă de programare care se foloseşte în probleme de optimizare şi care furnizează o singură soluţie (optimul global), obţinută prin alegeri succesive ale optimului local.

Metoda 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.

[1] Iacom

Problemele care se pot rezolva cu această metodă sunt de forma:
- se dă o mulţime A
- se cere o submulţime $S \subseteq A$ care:
  - să îndeplinească anumite condiţii interne (să fie ***acceptabilă***) şi
  - 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
  - dacă nu**,** el nu se mai testează ulterior
- procedeul continuă, până când au fost determinate toate elementele din mulţimea soluţiilor

Algoritmul metodei Greedy:

```
S=∅
cât timp !Solutie(S) și A≠∅ execută
   Alege(A,b)
   Elimină(A,b)
   dacă Posibil(S,b) atunci
      Adauga(S,b)

dacă Solutie(S) atunci
    scrie S
  altfel
    scrie ”Nu există soluție”
```

unde: `Solutie(S)` testează dacă `S` este o soluție optimă
`Alege(A,b)` extrage cel mai promițător element `b` din `A`
`Posibil(S,b)` testează dacă este posibil să se obțină o soluție

1. Sumă maximă

Se dă o mulţime $A=\{a_1, a_2, \ldots, a_n\}$ cu elemente reale. Să se determine o submulţime **S** astfel încât suma elementelor submulţimii să fie maximă.

```
#include <fstream>
using namespace std;

ifstream in("suma.in");
ofstream out("suma.out");

int main()
{
    float a[101],s[101];
    int i,n,m,nr_neg=0,el_max;
    in>>n;
    for(i=1;i<=n;i++)
    {
        in>>a[i];
        if(a[i]<0)
            nr_neg++;
    }
    m=0;
    if(nr_neg!=n)
    {
        for(i=1;i<=n;i++)
            if(a[i]>0)
            {
                m++;
                s[m]=a[i];
            }
    }
    else
    {
        el_max=a[1];
        for(i=2;i<=n;i++)
            if(a[i]>el_max)
                el_max=a[i];
        m++;
        s[m]=el_max;
    }
    for(i=1;i<=m;i++)
        out<<s[i]<<' ';
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

suma.in	suma.out
6 12 -4 78 -21 5 18	12 78 5 18
5 -12 -7 -895 -54 -231	-7

2. Cele mai mari două elemente ale unui şir

Se dă o mulţime $A=\{a_1, a_2, \ldots, a_n\}$ cu elemente întregi. Să se determine cele mai mari două elemente ale mulţimii.

```
#include <fstream>
using namespace std;

ifstream in("maxime.in");
ofstream out("maxime.out");

int main()
{
    int n,a,max1,max2,i,t;
    max1=max2=INT_MIN;
    in>>n;
    for(i=1;i<=n;i++)
    {
        in>>a;
        if(a>max1)
            max1=a;
        if(max2<max1)
        {
            t=max1;max1=max2;max2=t;
        }
    }
    out<<max1<<' '<<max2;
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

maxime.in	maxime.out
7 5 44 -6 12 47 488 54	54 488

3. Festivitate de premiere

La o festivitate de premiere, dirigintele clasei are **n** obiecte (**n≤1000**), de valori cunoscute, mai mici decât **100** lei. Ştiind că elevului care a obţinut premiul I, îi va fi înmânat **m** obiecte, realizaţi un program care identifică valoarea maximă a premiului I şi care obiecte au fost selectate.

```
#include <fstream>
using namespace std;

ifstream in("premiu.in");
ofstream out("premiu.out");

int main()
{
    int i,n,m,s,a[1001],sortat,t;
    in>>n>>m;
    for(i=1;i<=n;i++)
        in>>a[i];
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(a[i]<a[i+1])
            {
                t=a[i];a[i]=a[i+1];a[i+1]=t;sortat=1;
            }
    }while(sortat==1);
    s=0;
    for(i=1;i<=m;i++)
    {
        out<<a[i]<<' ';
        s=s+a[i];
    }
    out<<'\n'<<s;
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

premiu.in	premiu.out
8 4 3 7 8 1 6 8 9 5	9 8 8 7 32

4. Timp de aşteptare

La o casă de marcat sunt serviţi **n** clienţi (**n≤1000**). Cunoscându-se timpul necesar de servire pentru fiecare client, să se determine o ordine de servire a clienţilor, astfel încât timpul mediu de aşteptare să fie minim.

```
#include <fstream>
using namespace std;

ifstream in("clienti.in");
ofstream out("clienti.out");

int main()
{
    int i,n,a[1001],c[1001],sortat,t;
    in>>n;
    for(i=1;i<=n;i++)
    {
        in>>a[i];
        c[i]=i;
    }
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(a[i]>a[i+1])
            {
                t=c[i];c[i]=c[i+1];c[i+1]=t;
                t=a[i];a[i]=a[i+1];a[i+1]=t;sortat=1;
            }
    }while(sortat==1);
    for(i=1;i<=n;i++)
        out<<c[i]<<' ';
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

clienti.in	clienti.out
5 3 1 1 4 2	2 3 5 1 4

5. Problema spectacolelor

Într-o sală de spectacole, într-o zi, trebuie planificate **n** spectacole. Pentru fiecare spectacol se cunoaşte intervalul în care se desfăşoară. Se cere să se planifice un număr maxim de spectacole astfel încât să nu se suprapună.

Soluţie:

- mulţimea **A** este formată din cele **n** spectacole;
- fiecare spectacol are un interval de timp **[a,b]**,(**a<b**) şi un număr de ordine **nr**;
- două spectacole **i** şi **j** sunt compatibile dacă intervalele lor de ocupare (**[a_i,b_i],[a_j,b_j]**) sunt disjuncte (**b_i<a_j sau b_j<a_i**);
- se sortează spectacolele crescător după ora de încheiere a spectacolului;
- se afişează spectacolele pentru care ora de început a spectacolului **i** este mai mare sau egală cu ora de sfârşit a ultimului spectacol programat;

```cpp
#include <fstream>
using namespace std;

ifstream in("spectacol.in");
ofstream out("spectacol.out");

int main()
{
    int n,i,ora,minut,a[101],b[101],nr[101],sortat,t,k;
    in>>n;
    for(i=1;i<=n;i++)
    {
        nr[i]=i;
        in>>ora>>minut;
        a[i]=ora*60+minut;
        in>>ora>>minut;
        b[i]=ora*60+minut;
    }
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(b[nr[i]]>b[nr[i+1]])
            {
                t=nr[i];nr[i]=nr[i+1];nr[i+1]=t;sortat=1;
            }
    }while(sortat==1);
    k=1;
    out<<nr[1]<<' ';
    for(i=2;i<=n;i++)
        if(a[nr[i]]>=b[nr[k]])
        {
            out<<nr[i]<<' ';
            k=i;
        }
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

spectacol.in	spectacol.out
5 12 30 16 30 15 0 18 0 10 0 18 30 18 0 20 45 12 15 13 0	5 2 4

6. Expresie de valoare maximă

Pentru două mulţimi de numere întregi nenule, **C** cu **n** elemente şi **A** cu **m** elemente, (**n≤m**), se cere să se formeze o submulţime de **n** elemente din mulţimea **A**, astfel încât expresia:

$E=c_1 \cdot x_1+c_2 \cdot x_2+\ldots+c_n \cdot x_n$ să aibă valoarea maximă, ($x_i \in A$).

Soluţie:

- se sortează crescător cel două mulţimi;
- se parcurg mulţimile de la ultimul la primul element, cât timp elementele sunt pozitive şi nu au fost găsiţi **n** termeni ai expresiei **E**;
- se reţin în **p** şi **q** poziţiile la care s-a întrerupt parcurgerea în cele două mulţimi;
- se parcurg mulţimile de la primul la ultimul element cât timp elementele sunt negative şi nu au fost găsiţi **n** termeni ai expresiei **E**;
- dacă elementul din mulţimea **C** este negativ se continuă cu parcurgerea mulţimilor până au fost găsiţi **n** termeni ai expresiei **E**;
- dacă elementul din mulţimea **C** este pozitiv se continuă cu parcurgerea mulţimilor de la **p**, respectiv **q**, până au fost găsiţi **n** termeni ai expresiei **E**;

```
#include <fstream>
using namespace std;

ifstream in("expresie.in");
ofstream out("expresie.out");

void bubble_sort(int v[], int n)
{
    int sortat,i,t;
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(v[i]>v[i+1])
            {
                t=v[i];v[i]=v[i+1];v[i+1]=t;sortat=1;
            }
    }while(sortat==1);
}

int main()
{
    int a[101],c[101],n,m,i,j,p,q,E,k,t;
    in>>n>>m;
    for(i=1;i<=n;i++)
        in>>c[i];
    for(i=1;i<=m;i++)
        in>>a[i];
    bubble_sort(c,n);
    bubble_sort(a,n);
    E=0; k=0; i=n; j=m;
    while(c[i]>0 && a[j]>0 && k<n)
    {
        E=E+c[i]*a[j];
        i--; j--; k++;
    }
    p=i; q=j; i=1; j=1;
    while(c[i]<0 && a[j]<0 && k<n)
    {
        E=E+c[i]*a[j];
        i++; j++; k++;
    }
    if(c[i]<0)
        while(k<n)
        {
            E=E+c[i]*a[j];
            i++; j++; k++;
        }
    else
    {
        i=p; j=q;
        while(k<n)
        {
            E=E+c[i]*a[j];
            i--; j--; k++;
        }
    }
    out<<E;
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

expresie.in	expresie.out
6 7 -2 -1 3 4 5 5 -6 -5 -4 -3 -1 1 2	19
5 7 -2 -1 3 4 5 -5 -4 -1 2 5 7 8	97
6 7 -5 -4 -3 -1 4 5 -5 -4 1 2 3 4 5	77

7. Plata unei sume de bani

Să se plătească o sumă **s** cu un număr minim de bancnote cu valori date. Se consideră că din fiecare tip de bancnotă se poate folosi un număr nelimitat de bancnote, iar pentru ca problema să aibă soluţie, vom considera că există şi bancnote cu valoarea **1**.

Soluţie:

- se sortează descrescător vectorul de bancnote;
- atâta timp cât suma **s** este diferită de zero se determină ce bancnote se folosesc şi câte astfel de bancnote;

```cpp
#include <fstream>
using namespace std;

ifstream in("suma.in");
ofstream out("suma.out");

int main()
{
    int a[101],n,i,sortat,t,s;
    in>>n>>s;
    for(i=1;i<=n;i++)
        in>>a[i];
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(a[i]<a[i+1])
            {
                t=a[i];a[i]=a[i+1];a[i+1]=t;sortat=1;
            }
    }while(sortat==1);
    i=1;
    while(s!=0)
    {
        if(s/a[i])
        {
            out<<a[i]<<'x'<<s/a[i]<<'\n';
            s=s%a[i];
        }
        i++;
    }
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

suma.in	suma.out
4 147 5 1 50 10	50x2 10x4 5x1 1x2

8. Numere frumoase

Fiind dat un număr natural n, afişaţi pe ecran primele n numere frumoase. Numerele frumoase sunt numerele care au ca factori primi doar pe 2, 3 şi 5.

Soluţie:

- **n2** reprezintă cel mai mic multiplu de **2** din şir, neadăugat;
- **i** reprezintă indicele elementului din vector care îl conţine pe elementul din care s-a obţinut prin înmulţirea cu **2**;
- **n3** reprezintă cel mai mic multiplu de **3** din şir, neadăugat;
- **j** reprezintă indicele elementului din vector care îl conţine pe elementul din care s-a obţinut prin înmulţirea cu **3**;
- **n5** reprezintă cel mai mic multiplu de **5** din şir, neadăugat;
- **k** reprezintă indicele elementului din vector care îl conţine pe elementul din care s-a obţinut prin înmulţirea cu **5**;
- se completează primul element din vectorul cu valoarea **1**;
- atâta timp cât vectorul soluţie nu are **n** elemente, se adaugă în vector elementul minim dintre elementele **n2**, **n3** şi **n5**, apoi se măreşte variabila respectivă prin înmulţirea cu poziţia următoare celei de unde a fost obţinută;

Exemple

```
#include <fstream>
using namespace std;

ifstream in("numere.in");
ofstream out("numere.out");

int main()
{
    int a[101],n,n2,n3,n5,i,j,k,l,t;
    in>>n;
    t=1;  l=1;
    a[l]=t;
    n2=2; i=1;
    n3=3; j=1;
    n5=5; k=1;
    while(l!=n)
    {
        t=n2;
        if(n3<t)
            t=n3;
        if(n5<t)
            t=n5;
        l++;  a[l]=t;
        if(n2==t)
        {
            i++;  n2=2*a[i];
        }
        if(n3==t)
        {
            j++;  n3=3*a[j];
        }
        if(n5==t)
        {
            k++;  n5=5*a[k];
        }
    }
    for(i=1;i<=l;i++)
        out<<a[i]<<' ';
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

numere.in	numere.out
40	1 2 3 4 5 6 8 9 10 12 15 16 18 20 24 25 27 30 32 36 40 45 48 50 54 60 64 72 75 80 81 90 96 100 108 120 125 128 135 144

9. Problema rucsacului

Se consideră un rucsac cu care se poate transporta o greutate maximă **Gmax** şi mai multe obiecte de greutăţi g_1, g_2,…, g_n, la transportul cărora se obţin câştigurile c_1, c_2,…, c_n. Se cere să se încarce rucsacul astfel încât să se obţină un câştig maxim.

Problema poate fi transformată în două probleme distincte:
- problema discretă a rucsacului (obiectele nu pot fi tăiate);
- problema continuă a rucsacului (obiectele pot fi tăiate).

Observaţie: problema discretă a rucsacului se poate rezolva optim cu ajutorul programării dinamice.

a. Problema rucsacului – varianta discretă

Soluţie:

- se determină eficienţa fiecărui obiect **i**, astfel: $e_i = c_i / g_i$;
- se sortează obiectele descrescător după eficienţă;
- iniţial, greutatea obiectelor transportate este **G**;
- se alege un obiect în ordinea descrescătoare a eficienţei;
- verificăm dacă putem adăuga obiectul, adică dacă prin adăugare nu se depăşeşte greutatea admisă;
- repetăm procedeul până când s-au terminat obiectele sau până s-a încărcat greutatea admisă;

Exemple

```cpp
#include <fstream>
using namespace std;

ifstream in("rucsac.in");
ofstream out("rucsac.out");

int main()
{
    int Gmax,G,C,n,i,j,t,sortat;
    int c[101],g[101],e[101],nr[101];
    in>>n>>Gmax;
    for(i=1;i<=n;i++)
        in>>g[i];
    for(i=1;i<=n;i++)
        in>>c[i];
    for(i=1;i<=n;i++)
    {
        e[i]=c[i]/g[i];
        nr[i]=i;
    }
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(e[i]<e[i+1])
            {
                t=g[i];g[i]=g[i+1];g[i+1]=t;
                t=c[i];c[i]=c[i+1];c[i+1]=t;
                t=e[i];e[i]=e[i+1];e[i+1]=t;
                t=nr[i];nr[i]=nr[i+1];nr[i+1]=t;
                sortat=1;
            }
    }while(sortat==1);
    G=0;  C=0;
    for(i=1;i<=n;i++)
        if(G+g[i]<=Gmax)
        {
            out<<nr[i]<<' ';
            G=G+g[i];
            C=C+c[i];
        }
    out<<'\n'<<C;
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

rucsac.in	rucsac.out
4 10 3 2 8 1 6 8 8 5	4 2 1 19

b. Problema rucsacului – varianta continuă

Soluţie:

- se determină eficienţa fiecărui obiect **i**, astfel: $e_i = c_i / g_i$;
- se sortează obiectele descrescător după eficienţă;
- se alege un obiect în ordinea descrescătoare a eficienţei;
- verificăm dacă putem adăuga obiectul în întregime, adică dacă prin adăugare nu se depăşeşte greutatea admisă; în caz contrar, se taie obiectul păstrând greutatea admisă la transport;
- repetăm procedeul până când s-au terminat obiectele sau până s-a încărcat greutatea admisă;

```cpp
#include <fstream>
using namespace std;

ifstream in("rucsac.in");
ofstream out("rucsac.out");

int main()
{
    int n,i,t,sortat;
    float c[101],g[101],e[101],C,Gmax,p;
    int nr[101];
    in>>n>>Gmax;
    for(i=1;i<=n;i++)
        in>>g[i];
    for(i=1;i<=n;i++)
        in>>c[i];
    for(i=1;i<=n;i++)
    {
        e[i]=c[i]/g[i];
        nr[i]=i;
    }
    do
    {
        sortat=0;
        for(i=1;i<n;i++)
            if(e[i]<e[i+1])
            {
                t=g[i];g[i]=g[i+1];g[i+1]=t;
                t=c[i];c[i]=c[i+1];c[i+1]=t;
                t=e[i];e[i]=e[i+1];e[i+1]=t;
                t=nr[i];nr[i]=nr[i+1];nr[i+1]=t;
                sortat=1;
            }
    }while(sortat==1);
    C=0;  i=1;
    while(Gmax>0 && i<=n)
    {
        if(g[i]<=Gmax)
        {
            out<<nr[i]<<' '<<1<<'\n';
            Gmax=Gmax-g[i];
            C=C+c[i];
        }
        else
        {
            p=Gmax/g[i];
            out<<nr[i]<<' '<<p<<'\n';
            C=C+c[i]*p;
            Gmax=0;
        }
        i++;
    }
    out<<C;
    in.close();
    out.close();
    return 0;
}
```

Exemplu:

rucsac.in	rucsac.out
3 8 4 6 2 4 3 1	1 1 2 0.666667 6

Fişă de lucru

- Aplicaţii metoda de programare *Greedy*

5. Bibliografie şi webografie

1. Miloşescu M., *Informatică. Manual pentru clasa a XI*, Editura Didactică şi Pedagogică, Bucureşti, 2006
2. Lica D., *Informatică. Fundamentele programării. Culegere de probleme pentru clasa a XI*, Editura L&S Soft, Bucureşti, 2006
3. Sorin T., *Informatica. Tehnici de programare*, Editura L&S Infomat, Bucureşti, 2002
4. Logofătu G., *C++. Probleme rezolvate şi algoritmi*, Editura Polirom, Iaşi, 2001
5. http://en.wikipedia.org/wiki/Greedy_algorithm