PROIECT IDIVIDUAL

LA INFORMATICĂ

TEMA: TEHNICA GREEDY

A REALIZAT: Apostol Cristina, clasa a XI-a ”C”

A VERIFICAT: Maria Guțu

Chișinău 2019

Informaţie generală:

Metoda de programare Greedy se aplică problemelor de optimizare.

Algoritmii Greedy sunt caracterizati de metoda lor de functionare: la fiecare pas se alege cel mai bun candidat posibil, dupa evaluarea tuturor acestora. Metoda determina întotdeauna o singura solutie, asigurand un optim local, dar nu intotdeauna si global.

Algoritm Greedy:

-se dă o mulţime A
-se cere o submulţime S din multimea A care sa:
-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 optima;

-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 eliminate. **Altfel** el nu se mai testează ulterior.

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

Avantaje:

-poate fi aplicata multor probleme:
-determinarea celor mai scurte drumuri in grafuri (Dijkstra),
-determinarea arborelui minimal de acoperire (Prim, Kruskal),
-codificare arborilor Huffmann,
-planificarea activitatilor,
-problema spectacolelor si problema fractionara a rucsacului.

Dezavantaje:

-Algoritmii Greedy nu conduc în mod necesar la o solutie optimă.
-Nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda - Greedy rezolvă sau nu o anumită problemă de optimizare.

Exemple de probleme REZOLVATE:

1. SUMA MAXIMĂ

Se dă o mulţime X={x1, x2, . . ., xn } cu elemente reale. Să se determine o submulţime a lui X astfel încât suma elementelor submulţimii să fie maximă.

REZOLVARE: Pentru ***rezolvarea problemei*** reprezentăm atât mulţimea **X** cât şi mulţimea soluţiilor S sub forma a doi vectori de numere reale. Alegerea unui element din **X** se face in ordine, de la **1 la n**. Funcţia **POSIBIL(B, x)** se reduce la comparaţia **x[i]>0**, iar procedura **ADAUG(B, x)** va consta din adăugarea unui element **x[i]>0** la vectorul **S** în funcţie de contorul **k**.

```
program suma_maxima;
var s,x:array[1..20] of real;
i,k,n:integer;
begin
    write('Numarul de elemente n = ');
    readln(n);
    for i:=1 to n do
        begin
            write('x[',i,']= ');
            readln(x[i]);
        end;
    k:=0;
    for i:=1 to n do
        if x[i]>0 then
            begin
                k:=k+1;
                s[k]:=x[i]
            end;
    for i:=1 to k do
        write(s[i]:5:2,' ');
    readln;
end.
```

2. PROBLEMA SPECTACOLELOR

Într-un oraş de provincie se organizează un festival de teatru. Oraşul are o singură sală de spectacole, iar la festival şi-au anunţat participarea mai multe trupe. Aşadar, în sală, într-o zi, trebuie planificate N spectacole. Pentru fiecare spectacol se cunoaşte intervalul în care se desfăşoară: [ora_inceput, ora_sfarsit]. Se cere să se planifice un număr maxim de spectacole care, bineînţeles, nu se pot suprapune.

REZOLVARE: Pentru descrierea algoritmului convenim că spectacolele sunt codificate cu numere întregi din mulţimea **{1,2,…N}** iar ora de început şi sfârşit al fiecărui spectacol este exprimată în minute scurse de la miezul nopţii

O planificare optimă a spectacolelor presupune alegerea unui număr maxim **k** de spectacole $\mathbf{i_1, i_2,...,i_k}$ unde $\mathbf{i_1, i_2,...,i_k \in \{1,2,\ldots N\}}$, şi care îndeplinesc condiţia că spectacolul $\mathbf{i_{j+1}}$ începe după terminarea spectacolului $\mathbf{i_j}$.

Vom construi o soluţie după următorul algoritm:

1-Sortăm spectacolele după ora terminării lor;

2-Primul spectacol programat este cel care se termină cel mai devreme;

3- Alegem primul spectacol dintre cele care urmează în şir după ultimul spectacol programat care îndeplineşte condiţia că începe după ce s-a terminat ultimul spectacol programat;

4- Dacă tentativa de mai sus a eşuat (nu am găsit un astfel de spectacol) algoritmul se termină; altfel se programează spectacolul găsit şi algoritmul se reia de la **3**.

```
Program spectacole;
Type spectacol=record
                 ora_inc, ora_sf:integer;
                 ord:integer;
             end;
Var v:array[1..30] of spectacol;
n, ultim, nr:integer;

procedure sortare;
var i,j :integer; aux:spectacol;
begin
    for i:=1 to n-1 do
        for j:=i+1 to n do
            if v[j].ora_sf < v[i].ora_sf then
                begin
                    aux:=v[j];
                    v[j]:=v[i];
                    v[i]:=aux;
                end;
end;

procedure citire;
var hh, mm, i:integer;
begin
    write('Numarul de spectacole:');
    readln(n);
    for i:=1 to n do
        begin
            write('Spectacolul, i, incepe la:');
            readln(hh,mm);
            v[i]. ora_inc:=hh*60+mm;
            write ('Spectacolul, i, se termina la:');
            readln(hh,mm);
            v[i].ora_sf:=hh*60+mm;
            v[i].ord:=i;
        end;
end;
procedure greedy;
var i:integer;
begin
    writeln('Ordinea spectacolelor este:');
    ultim:=1;
    nr:=1;
    write(v[1].ord,' ');
    for i:=2 to n do
        if v[i].ora_inc>v[ultim].ora_sf then
            begin
                write(v[i].ord,' ');
                ultim:=i;
                Inc(nr);
            end;
```

```
    writeln('Se pot juca ', nr, ' spectacole');
end;

begin
    citire;
    sortare;
    greedy;
end.
```

3. DIVIZORI NATURALI

Fiind dat numărul natural **k > 1**, se cere să se determine cel mai mic număr natural **n** având exact **k** divizori naturali proprii (diferiţi de 1 şi n).

```
program k_divizori_naturali;
var v:boolean;
      k,n,s,i:integer;

procedure VERIF(n,k:integer;var v:boolean);
var j,i:integer;
begin
    i:=0;
    for j:=2 to n-1 do
        if n mod j = 0 then
            i:=i+1;
    if i = k then
        v:=true
    else
        v:=false;
end;

begin
    write('Numarul de divizori k > 1 ');
    readln(k);
    write('Cel mai mic numar care are exact ',k,' divizori este ');
    n:=k+2;
    s:=0;
    while s = 0 do
        begin
            VERIF(n,k,v);
            if v = true then
                begin
                    write(n);
                    s:=1;
                end;
            n:=n+1;
        end;
    readln;
end.
```

4.PROBLEMA CONTINUĂ A RUCSACULUI

Se consideră n obiecte. Pentru fi ecare obiect i (i=1, 2, ..., n) se cunoaște greutatea gi și cîștigul ci care se obţine în urma transportului său la destinaţie. O persoană are un rucsac cu care pot fi transportate unul sau mai multe obiecte greutatea sumară a cărora nu depășește valoarea Gmax. Elaboraţi un program care determină ce obiecte trebuie să transporte persoana în așa fel încît cîștigul să fi e maxim. În caz de necesitate, unele obiecte pot fi tăiate în fragmente mai mici.

Algoritm de rezolvare:

-Citirea greutății fiecarui obiect;

-Citirea capacității rucsacului;

-Iniţializăm obiectele;

-Se ordonează obiectele crescător în funcție de greutatea lor;

-Se inţializează volumul disponibil cu volumul obiectului;

-Se verifică dacă fiecare obiect încape în rucsac astfel:Dacă volumul obiectului e mai mic sau egal decât volumul disponibil al rucsacului atunci acesta încape în rucsac și din volumul disponibil al rucsacului scădem volumul obiectului;

-Dacă a rămas o zonă din rucsac neocupată atunci afişăm volumul rămas neocupat, în caz contrar înseamnă că nu am putut introduce nici un obiect în rucsac.

```
program rucsac1;
Var     g:array [1..10] of integer;
          i,n,Gm,R, aux : integer;
         ok:boolean;

begin
 writeln('nr obiecte'); readln(n);
writeln('capacitate  rucsac'); readln(R);
 writeln(' Obiectele de luat in rucsac:' );
 for  i:=1 to n do
        read (g[i]);
ok:=false;
while(ok=false) do
 begin
 ok:=true;
 for i:=1 to n-1 do
 if g[i]>g[i+1]   then
        begin
         aux:=g[i];
         g[i]:=g[i+1];
         g[i+1]:=aux;
         ok:=false;
        end;
  end;
writeln; for  i:=1 to n do    write( g[i], '*');
Gm:=0 ;
  i:=1;
     while ( Gm +g[i]<=R ) do
     begin
      Gm:=Gm+g[i];
      i:=i+1;
     end;
  writeln('sunt' ,i-1,'obiecte cu greutate', Gm) ;
   writeln ( '  a ramas' , R-Gm,' loc liber' ) ;end.
```

5.Se consideră mulţimea A={a1, a2, ..., ai, ..., an}

Elementele sale sînt numere reale, iar cel puţin unul din ele satisface condiţia ai>0. Elaboraţi un program care determină o submulţime B, astfel încît suma elementelor din B să fi e maximă. De exemplu, pentru A={21,5; -3,4; 0; -12,3; 83,6} avem B={21,5; 83,6}.

Rezolvare: Se observă că dacă o submulţime B, conţine un element b<=0, atunci suma elementelor submulţimii B \{b} este mai mare sau egală cu cea a elementelor din B. Prin urmare, regula de selecţie este foarte simplă: la fi ecare pas în submulţimea B se include un element pozitiv arbitrar din mulţimea A.

```
Program P153;
 { Tehnica Greedy }
const nmax=1000;
var A : array [1..nmax] of real;
 n : 1..nmax;
 B : array [1..nmax] of real;
 m : 0..nmax;
 x : real;
 i : 1..nmax;
Function ExistaElemente : boolean;
var i : integer;
begin
 ExistaElemente:=false;
 for i:=1 to n do
 if A[i]>0 then ExistaElemente:=true;
 end; { ExistaElemente }
procedure AlegeUnElement(var x : real);
var i : integer;

begin
 i:=1;
 while A[i]<=0 do i:=i+1;
 x:=A[i];
 A[i]:=0;
end; { AlegeUnElement }
procedure IncludeElementul(x : real);
begin
 m:=m+1;
 B[m]:=x;
end; { IncludeElementul }
begin
 write('Daţi n='); readln(n);
 writeln('Daţi elementele mulţimii A:');
 for i:=1 to n do read(A[i]);
 writeln;
 m:=0;
 while ExistaElemente do
 begin
 AlegeUnElement(x);
 IncludeElementul(x);
 end;
 writeln('Elementele mulţimii B:');
 for i:=1 to m do writeln(B[i]);
 readln;
end.
```

Concluzie:

Metoda Greedy este una dintre cele mai directe tehnici de proiectare a algoritmilor care poate fi aplicată la o gamă largă de probleme. Insa cu regret, metoda Greedy poate fi aplicată numai atunci cînd din enunţul problemei poate fi dedusă regula care asigură selecţia directă a elementelor necesare din mulţimea A.

Date bibliografice:

http://www.worldit.info/articole/algoritmica-articole/metoda-greedy/

https://sites.google.com/site/eildegez/home/clasa-xi/prezentarea-metodei-greedy

https://www.infoarena.ro/metoda-greedy-si-problema-fractionara-a-rucsacului

https://ru.scribd.com/doc/43454385/Metoda-Greedy#download