# METODA GREEDY

Bulhac Bogdan-Grigorie

# 1. Aspecte teoretice

## 1.1. Definitia

*Metoda Greedy* este una din cele mai directe tehnici de proiectare a algoritmilor care se aplica la o varietate larga de probleme.In general,aceasta metoda se aplica problemelor de optimizare.Specificul acestei metode consta in faptul ca se construieste solutia optima pas cu pas,la fiecare pas fiind selectat(sau „inghitit") in solutie elementul care pare „cel mai bun"la momentul respectiv,in speranta ca va duce la solutie optima globala [3].

## 1.2. Ideea generala

Se da o multime A cu n elemente si se cere sa se determine o submultime a sa(B) care satisface anumite restrictii. Aceasta submultime se numeste solutie posibila. Se cere sa se determine o solutie posibila care fie sa maximizeze fie sa minimizeze o anumita functie obiectiv data. Aceasta solutie posibila se numeste solutie optima [3].

Metoda Greedy lucreaza in pasi astfel:

1. Multimea B este vida la inceput

2. Se alege un element din A care pare a fi solutia optima la pasul i

3. Se verifica daca elementul ales poate fi adaugat la multimea solutiilor, daca da atunci va fi adaugat

4. Procedeul continua astfel, repetitiv, pana cand au fost determinate toate elementele din multimea solutiilor

Observatie*:* Metoda Greedy nu cauta sa determine toate solutiile posibile ( care ar putea fi prea numeroase) si apoi sa aleaga din ele pe cea optima, ci cauta sa introduca direct un element x in solutia optima.Acest lucru duce la eficienta algorimilor Greedy,insa nu conduc in mod necesar la la o solutie optima si nici nu este posibila formularea unui criteriu general conform caruia sa putem stabili excat daca metoda Greedy rezolva sau nu o anumita problema de optimizare.Acest motiv duce la insotirea fiecarei rezolvari prin metoda Greedy a unei demonstratii matematice(in general prin inductie) [3].

## 1.3. Forma generala

Este foarte dificil de a scrie forma generală a unei probleme rezolva folosind tehnica Greedy.
Să considerăm o mulţime A cu n elemente. Se cere o submulţime a sa cu m elemente (în cazul m=n este importantă ordinea alegerii elementelor), astfel încât să fie îndeplinite anumite condiţii (acestea diferă de la o problemă la alta).
Exemplu: se consideră o mulţime de n numere reale. Se cere o submulţime a sa, astfel încât suma elementelor sale să fie maximă.
Pentru rezolvare, vom alege un prim element al mulţimii de numere reale. Dacă este posibil, acesta va fi adăugat soluţiei, iniţial vide. Posibilitatea ca acesta să fie adăugat este dată de semnul numărului (acesta trebuie să fie mai mare ca O). Se alege un al doilea număr, cu care se procedează în mod asemănător. Algoritmul se încheie când au fost alese şl eventual adăugate toate elementele mulţimii.
Pentru a rezolva o problemă cu Greedy, soluţia se construieşte, după regula:
Pentru fiecare element care urmează să fie adăugat soluţiei finale, se efectuează o alegere a sa din elementele mulţimii A (după un mecanism specific fiecărei probleme în parte), iar dacă este posibil, acesta este adăugat. Algoritmul se termină fie când a fost găsită soluţia cerută, fie când s-a constatat inexistenţa acesteia.
Intuitiv, alegem un element, al doilea,....până când obţinem ce dorim sau până când au fost testate toate elementele mulţimii. De aici provine si numele metodei (greedy=lacom).
Greedy pare atât de simplă încât, la început, ne miră faptul că a fost evidenţiată ca tehnică. La o analiză atentă, vom observa că lucrurile nu stau chiar aşa. Exemplul prezentat este didactic (îl rezolvam şi fără să ştim că există această tehnică), el nu are alt rol decât de a evidenţia caracteristicile tehnicii [2].

## 1.4. Particularitati

Tehnica Greedy conduce la timp de calcul polinomial.
Motivul care conduce la acest timp de calcul, tine de mecanismul tehnicii. Să presupunem că mulţimea din care se face alegerea are n elemente si că soluţia are tot n elemente (caz maxim). Se fac n alegeri, la fiecare alegere se fac n teste, rezulta un algoritm cu timp O(n2).
De multe ori, este necesar ca elementele mulţimii. A să fie sortate, pentru ca apoi să alegem din acestea, iar sortarea necesita un timp minim O(n x log2n). Insă sortarea se efectuează la început. Prin urmare, acest timp se adună, deci nu influenţează rezultatul [2].

O întrebare firească: fiind dată o problemă, există întotdeauna un algoritm de tip Greedy care găseşte soluţia optimă? Evident, NU. Există probleme pentru care nu se cunosc astfel de algoritmi. Mai mult, pentru cele mai multe probleme nu se cunosc algoritmi Greedy.
Avantajul timpului polinomial, conduce la necesitatea utilizării tehnicii Greedy. Din alt punct de vedere, nu tuturor problemelor li se pot aplica algoritmi de acest tip. Ce este de făcut?
=> Pentru problemele pentru care nu se cunosc algoritmi care necesită timp polinomial, se caută soluţii, chiar dacă nu obţine, dar apropiate de acestea, dar care au fost obţinute în timp util. Multe din aceste

soluţii sunt obţinute cu Greedy [2].
Astfel de algoritmi se numesc algoritmi euristici [2].

# 2. Probleme rezolvate

## 1.

Problema banilor

Scrieţi un program, care afişează modalitatea de plată, folosind un număr minim de bancnote, a unei sume întregi S de lei (S<20000). Plata se efectuează folosind bancnote cu valoarea 1, 5, 10, 50, 100, 200 şi 500 de lei. Numărul de bancnote de fiecare valoare se citeşte din fişierul text BANI.IN, care conţine 7 rânduri, în fiecare din care sunt indicate numărul de bancnote respectiv de 1, 5, 10, 50, 100, 200 şi 500 de lei [2].
Intrare: Fişierul text BANI.IN şi de la tastatură se citeşte suma S.
Ieşire: Dacă e posibil de plătit această sumă S, atunci la ecran se va afişa valoarea bancnotei şi numărul de bancnote respective utilizate la plată. Dacă bancnote de careva valoare nu se folosesc, atunci nu se afişează această valoare. Dacă nu este posibil de efectuat plata cu bancnotele indicate – afişaţi mesajul respective [2].

Menţionăm, că probleme asemănătoare, cu banii, sunt mai multe. De obicei se presupune că bancnote de fiecare fel avem oricât de multe. Această problemă ne limitează numărul de bancnote.
Ideea algoritmului de rezolvare a aceste probleme constă în faptul că trebuie să începem eliberarea restului de la cea mai mare bancnotă. Există 2 variante, dacă suma necesară e mai mare ca produsul dintre numărul de bancnote şi nominalul atunci se iau toate bancnotele de acest nominal, dacă nu – atunci se iau atâtea bancnote, câte „încap”, care se află prin împărţirea sumei rămase la nominal. Pentru rezolvare se foloseşte un tablou cu 3 rânduri şi 7 coloane (pentru fiecare nominal câte o coloană). În primul rând al tabloului vom păstra nominalul bancnotelor, în al doilea rând - numărul bancnotelor citite din fişier şi în al treilea rând - numărul bancnotelor obţinute la schimb, practice ceea ce aflăm.
Calculul se va începe cu coloana a 7, adică începem de la sfârşit [2].

```
Program V3P7_02;
type tablou=array[1..3,1..7] of integer;
var s,ss,i : integer; a:tablou; f:text;
{In primul rind al tabelului vom pastra nominalul bancnotelor}
{In al doilea rind - numarul bancnotelor citite din fisier}
{In al treilea rind - numarul bancnotelor obtinute la schimb}
Procedure Afisare(sa:integer);
begin writeln('suma ',s);
if sa<>0
then writeln('nu poate fi transformata cu bancnotele date ')
else
begin writeln('se plateste cu urmatoarele bancnote');
for i:=1 to 7 do
if a[3,i]<>0
then writeln('bancnote de ',a[1,i]:6,' sau folosit ',a[3,i]);
end
end; { Afisare }
Procedure calcul(var sa:integer);
var nb:integer;
begin
i:=7;
while (i>=1) and (sa>0) do
```

```
begin nb:=sa div a[1,i];
if nb<>0 then if nb>= a[2,i]
then a[3,i]:=a[2,i]
else a[3,i]:=nb;
sa:=sa-a[3,i]*a[1,i];
i:=i-1;
end;
end; { calcul }
begin
a[1,1]:=1; a[1,2]:=5; a[1,3]:=10; a[1,4]:=50;
a[1,5]:=100; a[1,6]:=200; a[1,7]:=500;
assign (f,'bani.in'); reset(f);
for i:=1 to 7 do readln(f,a[2,i]);
write ('introduceti suma de lei S ');readln(s);
ss:=s; calcul(ss); Afisare(ss);
end.
```

## 2.

**Problema rucsacului**

O persoană are un rucsac cu care poate transporta o greutate maximă G. Persoana are la dispoziţie n obiecte si cunoaşte pentru fiecare obiect greutatea si câştigul care se obţine în urma transportului său la destinaţie. Se cere să se precizeze ce obiecte trebuie să transporte persoana în aşa fel încât câştigul sa fie maxim [2].

O precizare în plus transformă această problema în alte două probleme distincte. Această precizare se referă la faptul că obiectele pot fi sau nu tăiate pentru transportul la destinaţie. In prima situaţie, problema poartă numele de problema continuă a rucsacului, iar în a doua avem problema discreta a rucsacului. Aceste două probleme se rezolvă diferit Varianta continuă a problemei rucsacului este rezolvată mai jos, iar cea discretă se rezolvă cu ajutorul programării dinamice [2].

Problema continuă a rucsacului, în care persoana are posibilitatea să taie obiectele. În acest fel, se poate obţine o încărcare mai eficientă a rucsacului.

Algoritmul este următorul:

• se calculează, pentru fiecare obiect în parte, eficienţa de transport rezultată prin împărţirea câştigului la greutate (de fapt, acesta reprezintă câştigul obţinut prin transportul unităţii de greutate);

• obiectele se sortează în ordine descrescătoare a eficienţei de transport si se preiau în calcul în această ordine;

• câştigul iniţial va fi 0, iar greutatea rămasă de încărcat va fi greutatea rucsacului;

• atât timp cât nu a fost completată greutatea maximă a rucsacului si nu au fost luate in considerare toate obiectele, se procedează astfel:

• dintre obiectele neîncărcate se selectează acela cu cea mai ridicată eficienţă de transport şi avem două posibilităţi:

• acesta încape în totalitate în rucsac, deci se scade din greutatea rămasă de încărcat greutatea obiectului, la câştig se cumulează câştigul datorat transportului acestui obiect; se tipăreşte 1, în sensul că întregul obiect a fost încărcat;

• obiectul nu încape în totalitate în rucsac, caz în care se calculează ce parte din el poate fi transportată, se cumulează câştigul obţinut cu transportul acestei părţi din obiect, se tipăreşte procentul care s-a încărcat din obiect, iar greutatea rămasă de încărcat devine 0.

Vom considera un exemplu numeric.
Greutatea care poate fi transportată cu ajutorul rucsacului aste 3
Avem la dispoziţie 3 obiecte. Greutatea şi câştigul pentru fiecare obiect sunt prezentate mai jos:
Eficienţa de transport este 1 pentru primul obiect, 4 pentru al doilea si 2 pentru al treilea.
In concluzie, obiectul 2 se încarcă în întregime în rucsac, obţinând un câştig de 4 şi rămâne o capacitate de transport de două unităţi de greutate.
Se încarcă 2/3 din obiectul 3 (care este al doilea în ordinea eficienţei de transport) pentru care se obţine câştigul 4. Câştigul obţinut în total este 8. Se remarca strategia Greedy prin alegerea obiectului care va fi transportat, alegere asupra căreia nu se revine [2].

```
program rucsac;
type tablou=array [1..4] of real;
matrice=array [1..10] of tablou;

{In prima coloana se inscrie costul,
In a II - greutatea, in a III - eficienta,
si in a IV - a cata parte se ia
tabloul c il folosim la sortare, "al treilea pahar"}

var a:matrice; c:tablou; f:text;
loc,n,g,i,j:integer; max,castig,dg:real;
begin
assign (f,'rucsac.txt'); reset (f);
readln(f,n,g);
for i:=1 to n do
begin readln(f,a[i,1],a[i,2]);
a[i,3]:=a[i,1]/a[i,2]; a[i,4]:=0;
end;
{sortam tabloul dupa eficienta}
for i:=1 to n-1 do
begin max:=a[i,3];loc:=i;
for j:=i+1 to n do
if a[j,3]>max then begin max:=a[j,3]; loc:=j; end;
c:=a[i]; a[i]:=a[loc]; a[loc]:=c;
end;
{Aflam cat din fiecare obiect se pune in rucsac si calculam castigul}
castig:=0;
i:=1; dg:=g;
writeln ('greutatea ','costul ','eficienta ','rucsac');
while (i<=n) and (dg>0) do
begin;
if dg>=a[i,2]
then begin castig:=castig+a[i,1];
dg:=dg-a[i,2]; a[i,4]:=1;
end
else begin castig:=castig+dg*a[i,3];
a[i,4]:=dg/a[i,2];dg:=0;
end;
writeln (a[i,1]:6:2,a[i,2]:8:2,a[i,3]:12:2,a[i,4]:10:2);
i:=i+1;
end;
writeln ('greutatea rucsacului este ',g-dg:0:2);
writeln ('costul este ',castig:0:2);
```

```
end.
```

## 3.

Se consideră mulţimea $A=\{a_1, a_2, ..., a_i, ..., a_n\}$ elementele căreia sînt numere reale, iar cel puţin unul din ele satisface condiţia $a_i>0$. Elaboraţi un program care determină o submulţime $B$, $B \subseteq A$, 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 $B$, $B \subseteq A$, conţine un element $b \neq 0$, atunci suma elementelor submulţimii $B \setminus \{b\}$ 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 $A$.

În programul ce urmează mulţimile $A$ şi $B$ sînt reprezentate prin vectorii (tablourile unidimensionale) A şi B, iar numărul de elemente ale fi ecărei mulţimi ? prin variabilele întregi, respectiv n şi m. Iniţial $B$ este o mulţime vidă, respectiv $m=0$ [1].

```
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.
```

# 3. Concluzii

După cum se vede, în metoda *Greedy* soluţia problemei se caută prin testarea consecutivă a elementelor din mulţimea *A* şi prin includerea unora din ele în submulţimea *B*. Într-un limbaj plastic, submulţimea *B* încearcă să „înghită” elementele „gustoase” din mulţimea *A*, de unde provine şi denumirea metodei [1]. Si cu tote ca nu pare a fi o situatie complicate, in unele cazuri poate parea inaplicabila.

# Bibliografie


1. Anatol Gremalschi ‘’Manual pentru clasa a 11,, editura Stiinta 2014
2. http://dasinika.blogspot.com/2009/04/tehnica-greedy-pentru-problemele-pentru.html
3. http://www.worldit.info/uncategorized/metoda-greedy/