Referat

INFORMATICA

Tema: Tehnica Greedy

A efectuat: Carlasuc Sabina Cl. XI„C''
Profesor: Gutu Maria

Chisinau, 2020

Cuprins:

1.Introducere

Metoda **Greedy** este una dintre cele mai directe tehnici de proiectare a algoritmilor care poate fi aplicată la o gamă largă de probleme. În general, această metodă se aplică **problemelor de optimizare**. Majoritatea acestor probleme constau în determinarea unei submulţimi **B**, a unei mulţimi **A** cu **n** elemente care să îndeplinească anumite condiţii pentru a fi acceptată. Orice astfel de submulţime care respectă aceste restricţii se numeşte ***soluţie posibilă***. Din mulţimea tuturor soluţiilor posibile se doreşte determinarea unei soluţii care maximizează sau minimizează o funcţie de cost. O soluţie posibilă care realizează acest lucru se numeşte ***soluţie optimă***. Considerăm că soluţiile posibile au următoarea proprietate: dacă **B** este o soluţie posibilă, atunci orice submulţime a sa este soluţie posibilă.

Specificul acestei metode constă în faptul că se construieşte **soluţia optimă pas cu pas**, la fiecare pas fiind selectat (sau "înghiţit") în soluţie elementul care pare "cel mai bun" la momentul respectiv, în speranţa că va duce la soluţia optimă globală.

Având în vedere cele de mai sus, se poate construi un algoritm care operează în etape, considerând pe rând câte un element din A. În fiecare etapă se decide dacă o soluţie este sau nu optimală. Acest lucru este realizat dacă se consideră pe rând elementele din A într-o ordine dată de o procedură de selecţie. Dacă adăugarea acestui element conduce la o soluţie imposibilă se renunţă la adăugarea sa la soluţia parţială. Procedura de selecţie are la bază un cost. Acest cost poate fi sau nu identic cu costul care stabileşte dacă o soluţie posibilă este optimă sau nu.

Un dezavantaj al acestei tehnici este faptul ca pentru multe alte probleme, algoritmii greedy nu reușesc să producă soluția optimă, și poate chiar produce cea mai proastă soluție. Un exemplu este problema comis-voiajorului: pentru orice număr de orașe, există o atribuire de distanțe între orașe pentru care euristica celui mai apropiat vecin cel mai rău ciclu posibil. Algoritmii greedy produc soluții bune la unele probleme de matematică, dar nu la altele.[3]

2.Descrierea metodei

Exemplu: Se dă o mulţime **A** cu ***n*** elemente şi se cere să se determine o submulţime a sa **B** care satisface anumite restricţii. Această submulţime se numeşte ***soluţie posibilă*.** Se cere să se determine o soluţie posibilă care fie să maximizeze fie să minimizeze o anumită ***funcţie obiectiv*** dată. Această soluţie posibilă se numeşte ***soluţie optimă*.**

Metoda **Greedy** lucrează în paşi astfel:

1. se iniţializează mulţimea soluţiilor (B) la mulţimea vidă (B=Φ)
2. la fiecare pas se alege un anumit element $x \in A$ (cel mai promiţător element la momentul respectiv) care poate conduce la o solutie optimă la pasul i ;
3. se verifică dacă elementul ales poate fi adăugat la mulţimea soluţiilor; dacă da atunci va fi adăugat şi mulţimea soluţiilor devine $B=B \cup \{x\}$. Dacă un element se introduce în mulţimea **B**, el nu va fi niciodată eliminat de acolo. Dacă se alege un element din **A** care nu se poate adăuga mulţimii **B,** el nu se mai testează ulterior;
4. procedeul continuă astfel, repetitiv, până când au fost determinate toate elementele din mulţimea soluţiilor

În rezolvarea problemelor, de multe ori este utilă ordonarea mulţimii **A** înainte ca algoritmul propriu-zis să fie aplicat.

Observaţie: Metoda **Greedy** nu caută să determine toate soluţiile posibile (care ar putea fi prea numeroase) şi apoi să aleagă din ele pe cea optimă, ci caută să introducă direct un element ***x*** în soluţia optimă. Acest lucru duce la eficienţa algorimilor Greedy, însa nu conduc în mod necesar la o soluţie optimă şi nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili excat dacă metoda **Greedy** rezolvă sau nu o anumită problemă de optimizare. Acest motiv duce la completarea fiecărei rezolvări prin metoda **Greedy** cu o demonstraţie matematică.

Este foarte dificil de a scrie forma generală a unei probleme rezolvate prin tehnica **Greedy**. Totuşi procedura de mai jos vine să sintetizeze cele afirmate:

```
procedure Greedy(A, n, B)
  Begin
    B ←□
    for i=1 to n do
      begin
        ALEGE(A, x)
        If POSIBIL(B, x) then
          ADAUG(B, x)
      end
  end
```

Procedura **ALEGE(A, x)** selectează un element $x = a_j$ din mulţimea $A=\{ a_1, \ldots, a_n \}$ şi efectuează, eventual, interschimbarea $a_i \leftarrow a_j$. Funcţia **POSIBIL(B, x)** verifică dacă $B \cup \{x\}$ este soluţie posibilă, caz în care returnează valoarea booleană true, altfel va returna valoarea false.

Procedura **ADAUG(B, x)** înlocuieşte pe **B** cu **B**U**{x}**. Procedura **ALEGE(A, x)** nu precizează deloc cum se selectează un element x, de aceea trebuie stabilită o procedură de prelucrare (PREL), care va preciza ordinea în care vor fi introduse elementele lui **A** în soluţie - procedură specifică fiecărei probleme în parte .

Dupa cum se vede, metoda **GREEDY** nu încearcă să determine toate soluţiile posibile şi să o stabilească apoi pe cea optimală, comparând costurile. Ea alege pe rând câte un element urmând ca eventual să-l agauge în mulţimea soluţie. De aici vine şi numele metodei de **greedy** (in engleza lacom).

3.Exemple de programe

1. Suma maxima

Se dă o mulţime $X=\{x_1, x_2, \ldots, x_n\}$ *cu elemente reale. Să se determine o submulţime a lui* X *astfel încât suma elementelor submulţimii să fie maximă.*

Observaţie : evident că pentru a maximiza suma unui şir de numere acestea trebuie să fie, în primul rând, pozitive. Deci condiţia de alegere a unui element din şir ca să facă parte din mulţimea soluţie este ca acesta să fie pozitiv. Dacă, în plus, am adăuga şi condiţia suplimentară ca mulţimea soluţie să conţină un număr dat, **m**, de elemente (m<=n) atunci apare necesară ordonarea elementelor din mulţimea **X** în ordine descrescătoare, astfel ca la fiecare alegere să adăugăm la soluţie un element cu valoare maximă. În acest caz algoritmul se termină când în mulţimea soluţie au fost introduse numărul cerut de elemente.

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. K 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.
```

Metoda **Greedy** pare atât de simplă încât, la început, ne miră faptul că a fost evidenţiată ca tehnică. La o analiză atentă, vom observa însă, că lucrurile nu stau chiar aşa. Exemplele prezentate sunt didactice (le rezolvăm şi fără să ştim că există această tehnică), ele nu au alt rol decât de a evidenţia caracteristicile tehnicii.

NU întotdeauna există un algoritm de tip Greedy care găseşte soluţia optimă.

Există probleme pentru care nu se cunosc astfel de algoritmi. Mai mult, pentru cele mai multe probleme, nu se cunosc algoritmi Greedy.

3. Problema banilor

Este o problemă pentru care metoda Greedy determină soluţia optimă

*Scrieţi un program, care afişează modalitatea de plată, a unei sume întregi **S** de lei (S<20000), folosind un număr minim de bancnote. 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 linii, pe fiecare fiind indicate numărul de bancnote respectiv de 1, 5, 10, 50, 100, 200 şi 500 de lei.*

Date de Intrare**:* Fişierul text ***BANI.IN şi de la tastatură se citeşte suma ***S***.

Date de Ieşire: Dacă e posibil de plătit această sumă ***S***, atunci pe ecran se va afişa valoarea bancnotei şi numărul de bancnote respective utilizate la plată. Bancnotele nefolosite la plata sumei nu sunt afişate. Dacă nu este posibil de efectuat plata cu bancnotele indicate – afişaţi mesajul corespunzător.

(Menţionăm, că probleme asemănătoare, sunt mai multe. De obicei se presupune că dispunem de un număr nelimitat de bancnote de fiecare fel. Această problemă ne limitează însă numărul de bancnote de o anumită valoare.)

Ideea algoritmului de rezolvare a acestei 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", număr 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 folosite la plată, practic ceea ce trebuie aflat. Calculul se va începe cu coloana a 7, adică începem de la sfârşit.

```
Program problema_banilor;
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.
```

4. 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.*

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 $\mathbf{i_1, i_2,...,i_k \in \{1,2,…N\},}$ şi care îndeplinesc condiţia că spectacolul $\mathbf{i_{j+1}}$ începe după terminarea spectacolului $\mathbf{i_j}$.

Vom construi o soluţie după următorul algoritm:

P1. Sortăm spectacolele după ora terminării lor;

P2 . Primul spectacol programat este cel care se termină cel mai devreme;

P3. 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;

P4. 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 **P3**.

Algoritmul poate fi descris folosind diferite structuri de date:

- tablouri cu două linii şi **N** coloane în care să memorăm ora de început şi sfârşit al spectacolului de pe coloana **j**
- un vector de înregistrări în care să memorăm atât numărul de ordine al spectacolului cât şi ora de început şi sfârşit al lui.

Vom aborda algoritmul folosind a doua variantă.

```
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 ;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.
```

5. Problema rucsacului [2]

O persoană are la dispoziţie un rucsac cu ajutorul căruia poate transporta o greutate maximă ***G*** *şi urmează să efectueze un transport în urma căruia să obţină un câştig. Persoana are la dispoziţie* ***n*** *obiecte şi cunoaşte pentru fiecare obiect greutatea şi câştigul care se obţine în urma transportului său la destinaţie. Se cere să se precizeze ce obiecte (şi în ce cantităţi) trebuie să transporte persoana în aşa fel încât câştigul sa fie maxim şi care este acest câştig.*

„Problema rucsacului” comportă două abordări în raport cu modul în care pot fi transportate obiectele şi anume daca acestea pot fi sau nu tăiate pentru transportul la destinaţie. In prima situaţie (vezi enunţul problemei), problema poartă numele de ***problema continuă a rucsacului***, iar în a doua avem ***problema discreta a rucsacului***. Aceste două probleme se rezolvă diferit O soluţie pentru varianta continuă a problemei rucsacului este dată mai jos, aplicând principiile metodei Greedy, iar varianta discretă se rezolvă cu ajutorul programării dinamice.

Problema continuă a rucsacului, în care persoana are posibilitatea să încarce în rucsac fracţiuni de obiecte, conduce la o încărcare mai eficientă a rucsacului, dar necesită date suplimentare referitoare la eficienţa transportului fiecărui obiect.

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 şi se preiau în calcul în această ordine;

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

• atâta timp cât nu a fost completată greutatea maximă a rucsacului şi nu au fost luate în 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 100%, 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.

Presupunem că greutatea care poate fi transportată cu ajutorul rucsacului aste 5 şi că avem la dispoziţie 6 obiecte. Greutatea şi câştigul pentru fiecare obiect sunt prezentate în tabelul de mai jos:

Obiect	1	2	3	4	5	6
Greutate	2	2	3	1	2	1
Câştig	3	4	3	5	6	4
Eficienţă						

Dacă vom calcula eficienţa de transport după formula

Eficienţă = câştig / greutate

vom obţine rezultatele:

Obiect	1	2	3	4	5	6
Greutate	2	2	3	1	2	1
Câştig	3	4	3	5	6	4
Eficienţă	1.5	2	1	5	3	4

Se observă că obiectul 4 are cea mai bună eficienţă, urmat fiind de obiectele 6, 5, 2, 1 şi 3. Aşadar, vom încărca rucsacul astfel:

Obiectul 4 → 100%

Obiectul 6 → 100%

Obiectul 5 → 100%

Obiectul 2 → 50%

Câştigul total maxim = 17

In scrierea programului am folosit tipul **rucsac** definit ca un vector în cara fiecare element este un vector cu 5 componente în care am reţinut, în ordine:

– codul obiectului (numărul de ordine)

– greutatea obiectului

– câştigul obiectului

– eficienţa calculată ca raport dintre câştig şi greutate

– cota parte din greutatea obiectului, încărcată în rucsac

Datele de intrare se citesc din fişierul **rucsac.in**, în care, pe prima linie avem greutatea totala ce poate fi încărcată în rucsac iar pe următoarele linii avem câte trei numere reale, separate prin spaţii, reprezentând codul, greutatea şi respectiv câştigul fiecărui obiect. Programul afiseaza structura rucsacului, pentru fiecare obiect incarcat indicând:

Cod obiect, greutatea încărcată, câştigul obţinut, eficienţa efectivă, procent din greutatea obiectului

Precum şi câştigul maxim realizat.

```
program rucsac_greedy;
type coloana=array[1..5] of real;
     rucsac=array[1..100] of coloana;
var r:rucsac; n:integer; g:real;

procedure citire_obiecte (var a:rucsac; var
nr_ob:integer; var gr:real);
var f:text;
begin
  assign(f,'rucsac.in');  reset(f);
  readln(f,gr);  {citeste greutatea totala}
  nr_ob:=0;
  while not eof(f) do
    begin
      Inc(nr_ob);
      readln(f,a[nr_ob,1],a[nr_ob,2],a[nr_ob,3]
);
      a[nr_ob,4]:=a[nr_ob,3]/a[nr_ob,2];      {se
calculeaza eficienta}
    end;
  close(f);
end;

procedure afisare(var x:rucsac; n:integer);
var i,j:integer;
begin
  writeln('obiect               ','greutate      ','castig
','eficienta');
  for j:=1 to n do
    begin
      for i:=1 to 4 do
        write(x[j,i]:6:2,'   ');
      writeln;
    end;
end;

procedure sortare(var x:rucsac; n:integer);
var  i,j:integer;  c:coloana;
begin
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if x[i,4]<x[j,4] then
        begin
          c:=x[i];
          x[i]:=x[j];
          x[j]:=c;
        end;
end;

procedure  greedy(var  x:rucsac;  n:integer;
gr_max:real);
var castig_total, gr_rucsac, rest:real;
    nr_ob:integer;
begin
  castig_total:=0;  gr_rucsac:=0;  nr_ob:=1;
  while (gr_rucsac<gr_max) and (nr_ob<=n) do
    begin
      if x[nr_ob,2]<=gr_max-gr_rucsac then
        begin
          gr_rucsac:=gr_rucsac+x[nr_ob,2];
          castig_total:=castig_total+x[nr_ob,3];
          x[nr_ob,5]:=100;
        end
      else
        begin
          rest:=gr_max-gr_rucsac;
      castig_total:=castig_total+rest*x[nr_ob,4];
      x[nr_ob,3]:=rest*x[nr_ob,3]/x[nr_ob,2];
          gr_rucsac:=gr_rucsac+rest;
          x[nr_ob,5]:=rest*100/x[nr_ob,2];
          x[nr_ob,2]:=rest;
        end;
      Inc(nr_ob);
    end;
  if gr_rucsac<gr_max then
    writeln('Rucsacul nu este plin')
  else
    begin
      writeln('In  rucsac  au  fost  incarcate
obiectele:');
      writeln('cod  obiect  greutate  castig
eficienta  procent');
      for nr_ob:=1 to n do
        if x[nr_ob,5]<>0 then
writeln(x[nr_ob,1]:9:2,x[nr_ob,2]:10:2,x[nr_ob,3]
:8:2,x[nr_ob,4]:11:2,x[nr_ob,5]:10:2);
      writeln('Castigul total=',castig_total:10:2);
    end;
end;

begin
  citire_obiecte(r,n,g);
  writeln('Inainte de sortare');
  afisare(r,n);
  sortare(r,n);
  writeln('Dupa sortare');
  afisare(r,n);
  greedy(r,n,g);
  readln;
end.
```

4.Concluzii:

Metoda Greedy este foarte eficientă atunci cînd dorim să aflăm rezultatul optim în cît mai scurt timp posibil, deoarece algoritmii sunt polinomiali. Cu regret, aceasta poate fi aplicată numai atunci cînd din enunțul problemei poate fi dedusă regula care asigură selecția directă a elementelor necesare din mulțimea dată. [1]

5.Bibliografie

1. http://caterinamacovenco.blogspot.com/p/metoda-greedy.html
2. file:///D:/Users/D.Doctor/Downloads/XI_Informatica%20(in%20limba%20romana).pdf
3. https://ro.wikipedia.org/wiki/Algoritm_greedy#cite_note-3