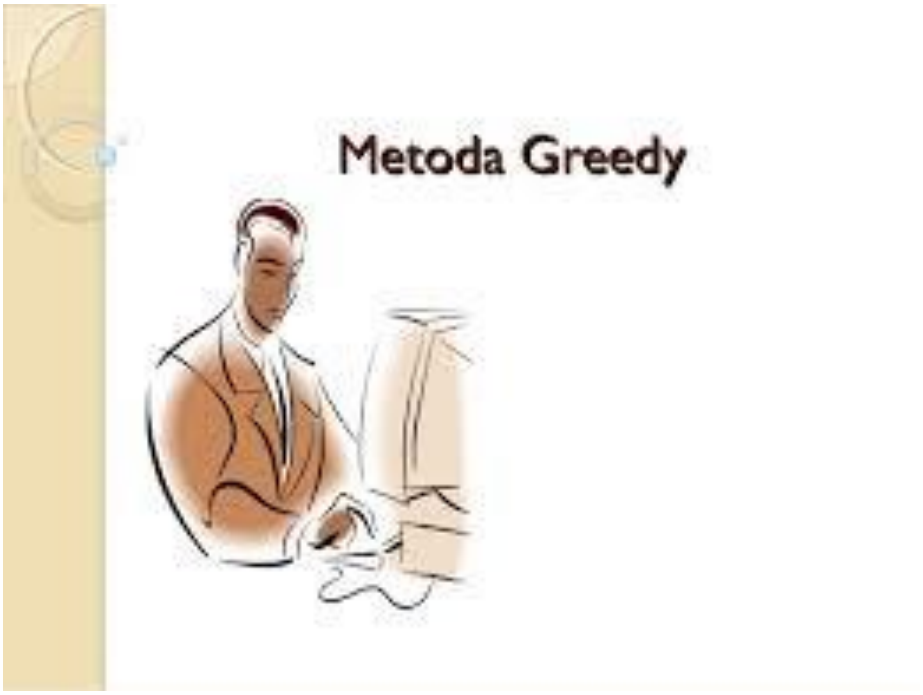
Metoda Greedy



Coșciug Andrian, cl XI-a "D"

Cuprins

Descrierea metodei Greedy

Această metodă presupune că problemele pe care trebuie să le rezolvăm au următoarea structură: – se dă o mulţime A={a1, a2, ..., an} formată din n elemente; – se cere să determinăm o submulţime B, B⊆A, care îndeplineşte anumite condiţii pentru a fi acceptată ca soluţie. În principiu, problemele de acest tip pot fi rezolvate prin metoda trierii, generînd consecutiv cele 2n submulţimi Ai ale mulţimii A. Dezavantajul metodei trierii constă în faptul că timpul cerut de algoritmii respectivi este foarte mare. Pentru a evita trierea tuturor submulţimilor Ai , Ai ⊆A, în metoda Greedy se utilizează un criteriu (o regulă) care asigură alegerea directă a elementelor necesare din mulţimea A. De obicei, criteriile sau regulile de selecţie nu sînt indicate explicit în enunţul problemei şi formularea lor cade în sarcina programatorului. Evident, în absenţa unor astfel de criterii metoda Greedy nu poate fi aplicată.

Probleme zilnice care pot fi rezolvate folosind această metodă:

Urmărirea cât mai multor spectacole într-o anumită perioadă de timp.

Plata unei sume în monede de mai multe tipuri.

Alegerea cât mai multor preparate din lista unui meniu din restaurant.

Alegerea unui număr maxim de articole care se pot încadra într-un rucsac.

Problema selectării activităților este caracteristică acestui tip de problemă, dacă obiectivul este alegerea numărului maxim de activități care nu intră în conflict.

Într-un computer Macintosh, jocul Crystal Quest își propune să colecteze cristale. Jocul are un mod demo, caz în care jocul folosește un algoritm lacom pentru a merge la fiecare cristal. Dar inteligența artificială nu ține cont de obstacole, astfel încât modul de demonstrație se termină adesea rapid.

Algoritmul și principiul de operare a metodei Greedy

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 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
- **altfel**
- el nu se mai testează ulterior
- procedeul continuă, până când au fost determinate toate elementele din mulţimea soluţiilor

În general, algoritmii *greedy* au cinci componente:

1. O mulțime de candidați, din care se creează o soluție
2. Funcția de selecție, care alege cel mai bun candidat pentru a fi adăugat la soluție
3. O funcție de fezabilitate, care este folosită pentru a determina dacă un candidat poate fi utilizat pentru a contribui la o soluție
4. O funcție obiectiv, care atribuie o valoare unei soluții sau unei soluții parțiale, și
5. O funcție de soluție, care va indica atunci când s-a descoperit o soluție completă

Algoritmii *greedy* produc soluții bune la unele probleme de matematică[d], dar nu la altele. Cele mai multe probleme la care funcționează vor avea următoarea proprietate:

Proprietatea alegerii *greedy*

Putem face orice alegere pare mai bună pe moment și apoi se pot rezolva subproblemele care apar mai târziu. Alegerea făcută de către un algoritm *greedy* poate depinde de alegerile făcute până atunci, dar nu de viitoarele alegeri sau de toate soluțiile subproblemelor. El face iterativ o alegere *greedy* după alta, reducând fiecare problemă dată într-una mai mică. Cu alte cuvinte, un algoritm greedy nu își reconsideră alegerile. Aceasta este principala diferență față de programarea dinamică[d], care este exhaustivă găsește garantat soluția.

După fiecare etapă, programarea dinamică ia decizii pe baza tuturor deciziilor luate în etapa anterioară, și poate reconsidera calea găsită în etapa algoritmică anterioară.

Probleme rezolvate

1. 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.
   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 respectiv.

```
Program P1;
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.
```

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.

2 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.

```
program P2;
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 Programul dat stabileste numărul de ordine al unui spectacol și numarul maxim de spectacole ce poate fi afișat pînă la miazul nopții în functie de durata lor.

```
Program Teatru
Program P3;
type teatru=record
ins, sfs:integer; (ora de inceput si de sfarsit a unui spectacol calculata in minute
scurse fata de
miezul noptii)
ord:integer; (numarul de ordin al spectacolului)
end;
Var v:array [1..30] of teatru;
n, ultim, nr:integer; (n=numarul de spectacole, in variabila ultim avem in
permanenta ultimul
spectacol selectat, nr=numarul maxim de spectacole)
Procedure sortare_piese;
Var i,j:integer;
temp:teatru;
Begin
For i:=1 to n-1 do
for j:=i+1 to n do
if v[j].sfs<v[i].sfs then
begin
temp:=v[i];
v[i]:=v[j];
v[j]:=temp;
end;
Procedure citire_piese;
Var hh,mm,i:integer;
begin
Write (‘Numarul de piese de teatru n= ‘); Readln (n);
for i:=1 to n do begin
Write (‘Piesa cu nr ‘,i, cand incepe? (ora si minutul)’);
Readln (hh,mm);
v[i].ins:=hh*60+mm;
Write (‘Piesa cu nr ‘,i, cand se termina? (ora si minutul)’);
Readln (hh,mm);
v[i].ins:=hh*60+mm;
v[i].ord:=i;
end; end;
Procedure afis_piese;
Var i:integer;
Begin
Write (‘Inceputurile si sfarsiturile pieselor in minute scurse de la miezul noptii: ‘);
for i:=1 to n do
```

```
write (‘(‘,v[i].ins,’,’,v[i].sfs,’,’,v[i].ord,’)’);
writeln;
end;
Procedure algo_greedy;
Var i:integer;
Begin
Write (‘Piesele posibile, in ordine: ‘);
ultim:=1; nr:=1;
write (v[i], ‘ ‘);
for i:=2 to n do
If (v[i].ins>v[ultim].sfs) then
Begin
Write (v[i].ord, ‘ ‘);
ultim:=i;
nr:=nr+1; end;
Writeln (‘In total se pot alege maxim’,nr,’ piese’);
end;
Begin
citire_piese;
afis_piese;
sortare_piese;
afis_piese;
algo_greedy;
end.
```

4 Programul dat stabilește numarul de client în ordine ce pot să fie deserviți la o benzinărie în funcție de timpul inserat de către noi în cadrul programului.

```
Program p4;
Type benz=record
ins, sfs:integer;
ord:integer; end;
Var v:array [1..100] of benz;
n, ultim, nr:integer;
Procedure citire_clienti;
Var hh, mm, i:integer;
begin
Write (‘n= ‘); Readln (n);
for i:=1 to n do begin
Write (‘Clientul cu nr. ‘,i,’cand este servit? (ora si minutul)’);
Readln (hh, mm);
v[i].ins:=hh*60+mm;
Write (‘clientul cu nr ‘, i, ‘ cand a terminat alimentarea ? ‘);
Readln (hh, mm);
v[i].sfs:=hh*60+mm;
v[i].ord:=i; end; end;
Procedure afisare_clienti;
Var i:integer;
Begin
Write (‘ cand incepe sa fie servit si cand a terminat alimentarea: ‘);
for i:=1 to n Do
Write (‘(‘v[i].ins,’,’,v[i].sfs, ‘,’,v[i].ord’)’);
Writeln; end;
Procedure sortare_clienti;
Var i,j:integer;
t:benz;
Begin
for i:=1 to n-1 Do
for j:=i+1 to n Do
if (v[j].sfs<v[i].sfs) then
Begin
t:=v[i]; v[i]:=v[j];
v[j]:=t; end; end;
Procedure alg_greedy;
var i:integer;
Begin
Write (‘posibilii clienti, in ordine: ‘);
ultim:=1;
nr:=1;
Write (v[i].ord, ‘ ‘);
```

```
for i:=2 to n do
if (v[i].ins>v[ultim].sfs) then
begin
Write (v[i].ord, ‘ ‘);
ultim:=i;
nr:=nr+1;
end;
Writeln (‘in total se pot alege maxim’,nr, ‘clienti’);
begin
citire_clienti;
afisare_clienti;
sortare_clienti;
afisare_clienti;
alg_greedy;
END.
```

5 Programul dat stabilește prin metoda Greedy maximul dintre mai multe numere și le interschimbă

```
Program P5;
Var n, a1, a2, c:Integer;
Begin
a1:=-MAXINT; (initializam primele 2 numere si n cu o constanta predefinita)
a2:=-MAXINT;
n:=-MAXINT;
While n<>0 Do Begin
If (n>a1) Then a1:=n; (daca numarul n este mai mare decat primul cel mai mare numar atunci
maximul este n)
If (a2<a1) Then Begin
c:=a1;
a1:=a2;
a2:=c; end; (interschimbare)
Readln (n); end;
Writeln (‘a1, ‘ ‘,a2’);
end.
```

Concluzii

În concluzie putem afirma că metoda de programare **Greedy** se aplică problemelor de optimizare. Aceasta metoda constă în faptul că se construieşte solutia optimă pas cu pas, la fiecare pas fiind selectat în solutie elementul care pare „**cel mai bun/cel mai optim**” la momentul respectiv, în speranta că această alegere locală va conduce la **optimul globa**l.

Algoritmii Greedy sunt foarte eficienti, dar nu conduc în mod necesar la o solutie optimă. Şi nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda Greedy rezolvă sau nu o anumită problemă de optimizare. Din acest motiv, orice algoritm Greedy trebuie însotit de o demonstratie a corectitudinii sale . Demonstratia faptului că o anumită problemă are proprietatea alegerii Greedy se face de obicei prin inductie matematică.

Metoda **Greedy** 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, numite si **conditii de optim**.

Bibliografie

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

https://www.pbinfo.ro/articole/16619/metoda-greedy

https://www.slideshare.net/LuminiaMihailov/metoda-greedy-47936020

http://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-02

https://prezi.com/getfspxmrtwg/metoda-greedy/