Instituția Publică Liceul Teoretic Spiru Haret.

Analiză:
Metoda Reluării.

1. Date Generale.
2. Avantaje.
3. Dezavantaje.
4. Concluzii.
5. Aplicații(Exemple de probleme).

10.05.2019

Realizat de: Colomiicenco Sanda,

clasa a XI-a „D”

Profesor: Maria Guțu

Date Generale:

Metoda reluării implcă in sine faptul ca soluția unei probleme poate fi reprezentată printr-un vector.

$$X=(x1, x_{2,} ...,x_{k,}...,x_n)$$

Fiecare componentă a vectorului X luând valori dintr-o anumtă mulțime A_k, k=1,2,3,...,n. Elementele fiecărei mulțimi A_k fiind ordonate conform unui criteriu stabilit.

La baza acestei metode stau anumite condiții de continuare specifice problemei, aceste condiții ne permit să evităm calcularea inutila a soluțiilor ce nu corespund cerințelor.

Aceste condiții stablesc situații in care are sens să trecem la calculul x_{k+1}. Dacă condițiile nu sunt satisfăcute programul revine la funcția anterioare și 1-Alege un alt x_k sau 2-Micșorează pe k cu o unitate încercând o noua alegere pentru x_{k-1}.

Micșorarea lui k a dat nume metodei, cuvântul „reluare” semnificând revenirea la alte variante de alegere a variabilelor x1, x2,..., x_{k-1}. Aceeași semnificație o are și denumirea engleză a metodei de studiu-backtracking (back-înapoi, track-urmă).

Metoda poate fi folosită la rezolvarea urmatoarelor probleme:

1. Generarea permutărilor unei mulţimi.
2. Generarea aranjamentelor unei mulţimi.
3. Generarea submulţimilor unei mulţimi.
4. Generarea submulţimilor cu m elemente ale unei mulţimi (combinări)
5. Generarea produsului cartezian a n mulţimi.
6. Aranjarea a n regine pe o tablă de şah de dimensiune n fără ca ele să se atace etc.

Algoritmul general:

```
Procedure Reluare(k:integer);
 begin
if k<=n then
   begin
X[k]:=primulelement(k);
if Continuare(k) then Reluare(k+1);
while ExistaSuccesor(k) do
       begin
X[k]:=Succesor(k);
 If Continuare(k) then Reluare(k+1)
    End;
   End ;
Else PrelucrareaSolutiei;
End;
```

Reluare: Comunică cu programul principal și subprogramele apelate prin variabilele globale.

Primulelement(k): Returnează primul element din mulțimea A_k.

Continuare(k):Returnează valoarea true dacă elementele inscrise in primele componente k sale X satisfac condițiile și false în caz contrar.

ExistaSuccesor(k): True daca elemental memorat x_k are succesori în mulțmea A_k, false în caz contrar.

Succesor(k): Returnează succesorul elementului memorat în componența x_k

PrelucrareaSolutiei: Soluția reținută este afișata pe ecran.

Metoda se aplica astfel :

1) se alege prima valoare sin A_1 si i se atribuie lui x1 ;

2) se presupun generate elementele x1…x[k-1], cu valori din A1..A[k-1]; pentru generarea lui x[k] se alege primul element din A[k] disponibil și se testeaza îndeplinirea condițiilor de continuare.

Apar astfel urmatoarele situații :

a) x[k] îndeplineste conditiile de continuare. Daca s-a ajuns la solutia finala (k = n) atunci se afiseaza solutia obținuta. Daca nu s-a ajuns la solutia finala se trece la generarea elementului urmator : x [k-1];

b) x[k] nu îndeplineste conditiile de continuare. Se încearca urmatoarea valoare disponibila din A[k]. Daca nu se gaseste nici o valoare în A[k] care sa îndeplineasca conditiile de continuare, se revine la elementul x[k-1] si se reia algoritmul pentru o noua valoare a acestuia. Algoritmul se încheie când au fost luate in considerare toate elementele lui A1.

Avantaje vs Dezavantaje:

+ Se evită generarea tuturor soluțiilor posibile acestea fiind triate datorita condițiilor aplicate, astfel reducând timpul efectuării programului în cazul în care acesta are date de intrare cu dimensiuni rezonabil de mici.

+ Rezolvarea unei probleme prin reluare garantează obţinerea soluţiei programele date fiind considerate relative simple, iar depanarea lor nu necesita verificari complexe .

-Nu există algoritm de generare a condițiilor necesare sau a condițiilor optime pentru rezolvarea unei probleme alegerea și aplicarea acestora depinzând de utilizator, fapt ce elimină garanția unui studiu corect din prima incercare sau poate duce la blochaj.

-Timpul de execuţie este mare, datorită revenirilor specifice metodei mai ales în cazul datelor de intrare cu dimensiuni mari .

Concluzii:

Folosirea acestei metode ca și metodă principală, deși are caracteristici positive precum condițiile abordate, nu este recomandată. Un lucru ce trebuie examinat înainte de elaborarea unei metode backtracking sunt datele de intrare, mai exact dimensiunile lor. Ținând cont că principala regulă ar fi ca o metoda de parcurgere trebuie să ne ofere un rezultat în timp util, datele studiate de metoda reluarii trebuie sa fie de dimensiuni relativ mici.

Totuși metoda aceasta ne oferă posibilitatea de a selecta soluțiile ce se încadrează in condiții specific si ne oferă un mod de rezolvare pentru tipuri de probleme precum(vedeți pagina 2).

Probleme rezolvate:

Să se plaseze n regine pe o tablă de şah de dimensiune nxn astfel încât oricare două regine să nu se atace.Pentru ca două regine să nu se atace, ele nu trebuie să fie situate pe aceeaşi linie, pe aceeaşi coloană sau pe aceeaşi diagonală. Cum numărul de regine este egal cu dimensiunea tablei de şah, deducem că pe fiecare linie trebuie plasată o regină. Deci, pentru ca poziţia reginelor să fie complet determinată, este suficient să reţinem pentru fiecare linie coloana în care este plasată regina. Pentru aceasta vom utiliza un vector C, cu n componente având următoarea semnificaţie: C [i] reprezintă coloana în care este plasată regina de pe linia i. Condiţii interne

- C[i] =[l,2,...,n], pentru I =[l,2...,n] (elementele vectorului C sunt indici de linii)
- C[i]<> C[j]; i<> j; i, j = [l,2,...,n] (două regine nu pot fi plasate pe aceeaşi coloană)
- C[i]-C[j] | <> | i-j |; Vi <> j; i, j = [l,2,...,n] (două regine nu pot fi plasate pe aceasi diagonală)

  Comentariu: Fiecare regina are un parametru k egal cu nr. liniei pe care se afla aceasta.

  Pentru inceput se compara indicele k cu n, daca k<n rezulta că solutia nu este completa, incepându-se procedura Plaseaza_Regina, în caz contrar se incepe procedura Afisare.

  Analizăm procedura Plaseaza_Regina:

  Se compara indicele k cu n, k<n rezulta că solutia nu este completa. I se atribuie variabilei locale i valoarea 1, se verifica daca pe pozitia (1,1) poate fi plasata o regină. Verificarea are succes, regina este plasata, se apeleaza prin urmare recursiv procedura Plaseaza_Regina(k+1).

  Daca verificarea eşueaza si pe linia k+1 nu mai poate fi plasată nici o regină, programul se intoarce la apelul precedent marind variabila i cu 1, verificarea reâncepe.

  Verificările au loc pâna când k=n si începe procedura Afişare.

```
program Regine;
const NrMaxRegine = 30;
type   Indice = 0 .. NrMaxRegine;
       Solutie = array[Indice] of 0..NrMaxRegine;
 var   C: Solutie; n: Indice;  NrSol: word;
 procedure Afisare;
 var i, j: Indice;
 begin
 inc(NrSol); writeln('Solutia nr. ', NrSol);
 for i := 1 to n do
   begin
   for j := 1 to n do
      if   j = C[i] then write(' * ')
                 else write(' o ');
   writeln
end;
writeln; readln;
end;
```

```
procedure Plaseaza_Regina(k: Indice);
{cand apelam procedura Plaseaza__Regina cu parametrul k am
plasat deja regine pe liniile 1, 2, ...,k-l}
var i, j: Indice; ok: boolean;
begin
if k-1 = n then                  {am obtinut o solutie}
Afisare          {prelucrarea solutiei consta in afisare}
else
{trebuie sa mai plasam regine pe liniile k,k+l,...,n}
 for i := 1 to n do          {determin multimea MC a candidatilor pentru
          pozitia k}
  begin ok := true;          {verific daca pot plasa regina de pe linia k in
          coloana i}
       for j := 1 to k-1 do
          if (C[j]=i) or (abs(C[j]-i)=(k-j)) then ok := false;
                        {regina s-ar gasi pe aceeasi coloana sau aceeasi
          diagonala cu o regina deja plasata}
          if ok then      {valoarea i respecta conditiile interne} begin
           C[k] := i;{i este un candidat, il extrag imediat}
           Plaseaza_Regina(k+1) ;
end;  end;   end;
begin                            {program principal}
write('n= '); readln(n);
Plaseaza_Regina(1);    end.
```

Dintr-un nr. de 6 cursuri optionale un elev trebuie sa aleaga 3. Sa se afiseze toate posibilitatile de alegere precum si nr. lor.

```
program cursuri;
const n=6;
        p=3;
type stiva=array [1..10] of integer;
var st:stiva;
    ev,as:boolean;
    k:integer;
procedure init(k:integer;var st:stiva);
begin
if k>1 then st[k]:=st[k-1]
        else if k=1 then st[k]:=0;
end;
procedure succesor(var as:boolean;var
st:stiva;k:integer);
begin
if st[k]<n-p+k then
begin
st[k]:=st[k]+1;
 as:=true;
 end;
else as:=false;
end;
procedure valid(var ev:boolean;var
st:stiva;k:integer);
var i:integer;
begin
ev:=true;
for i:=1 to k-1 do if st[i]=st[k] then
ev:=false;
end;
function solutie(k:integer):boolean;
begin
solutie:=(k=p);
end;
procedure tipar;
var i:integer;
begin
for i:=1 to p do write (st[i]);
writeln;
end;
begin;
k:=1;init(k,st);
while k>0 do
   begin
      repeat
         succesor (as,st,k);
          if as then valid(ev,st,k);
      until (not as) or (as and ev);
   if as then
         if solutie(k) then tipar
        else begin
              k:=k+1;
              init(k,st)
              end
         else k:=k-1;
   end;
  readln;
end.
```

Generarea permutărilor mulţimii {1,2,….n} :

Permutarile de ordin n reprezinta toate posibilitatile de a aranja elementele unei multimi de n elemente.
Astfel o condiție necesara este ca x[i] să fie diferit de x[k].
Variabila k este un indice ce ne arata nr. de permutari, variabila i indica nr. fiecărei permutări. Se verifică daca i=k-1, dacă nu, soluția este incompletă, se verifică daca x[i]=x[k] (x[k] reprezintă permutarea anterioara) , dacă nu, verificarea a avut loc cu succes și se mărește indicele i cu 1, dacă x[i]=x[k] atunci se intoarce la permutarea anterioara și se înlocuieste unul din elementele permutarii cu altul, are loc verificarea.

```
program permutari;
var x:array[1..200] of integer;
nr,i,n,k:integer;
function col(k:integer):boolean;
begin
col:=true;
for i:=1 to k-1 do
if x[i]=x[k] then
begin col:=false;
exit;
end;
end;
begin
write('n='); readln(n);
nr:=0;
 k:=1;
end.
```

Generarea Partițiilor unei mulțimi:

Fie n numar natural nenul. Scrieti un program de generare a tuturor partitiilor multimii {1, 2, 3,..., n}.

Prin partitie se intelege o descompunere a multimii initiale intr-o reuniune de multimi nevide si disjuncte. Pentru o multime cu n elemente vor exista maxim n multimi in cadrul unei partitii.

Solutia este o stivă de lungimi variabile x = (x1, x2,…, xk), unde xi ∈ {1,2,…,n}

Condiții: x1+ x2 +…+ xk = n

xk-1 ≤ xk (avem xk ∈ {xk-1,…,n})

x1 + x2 +... + xk ≤ n

```
begin
k:=1; s:=0;
while(k>=1) do
begin
if(x[k]<n) then
begin
x[k]:=x[k]+1; s:=s+1;
if (s<=n)
then begin
if(s=n) then begin
tipar(x,k);
s:=s-x[k];
k:=k-1;
end
else begin
k:=k+1;
x[k]:=x[k-1]-1;
s:=s+x[k];
end;
end
else begin
s:=s-x[k];
k:=k-1;
end;
end;
end;
```

```
void back(){
int k=0, s=0;
while(k>=0){
if(x[k]<n){
x[k]++; s++;
if(s<=n) { //conditia continua
if(s=n){ // solutie
tipar(x,k);
s=s-x[k]; k--;
//revenire
}
else
{ //avansare
k++;
x[k]=x[k-1]-1;
s+=x[k];
}
}
else {//revenire
s=s-x[k]; k--;
}
}
}
}
```

Turnuri de cuburi:

Se dau n cuburi numerotate 1,2,...,n, de laturi Li si culori Ci, i=1,2,...,n (fiecare culoare este codificata printr-un caracter). Sa se afiseze toate turnurile care se pot forma luând k cuburi din cele n disponibile, astfel încât:

-laturile cuburilor din turn sa fie in ordine crescatoare;

-culorile a oricare doua cuburi alaturate din turn sa fie diferite.

```
Program cuburi;
type stiva=array [1..100] of integer;
var st:stiva;
    i,n,p,k:integer;
    as,ev:boolean;
    L:array [1..10] of integer;
    C:array [1..10] of char;
procedure init(k:integer;var st:stiva);
begin
st[k]:=0;
end;
procedure succesor(var as:boolean;var
st:stiva;k:integer);
begin
if st[k]<n then
            begin
            st[k]:=st[k]+1;
            as:=true;
            end
            else as:=false;
end;
procedure valid(var
ev:boolean;st:stiva;k:integer);
var i:integer;
begin
ev:=true;
for i:=1 to k-1 do if L[st[k]]<=L[st[i]] then
ev:=false;
if C[st[k]]=C[st[k-1]] then ev:=false;
end;
function solutie(k:integer):boolean;
for i:=1 to n do
x[i]:=0;
while k>0 do
if k=n+1 then
 begin k:=k-1;
 nr:=nr+1;
writeln(' solutia nr ',nr);
for i:=1 to n do
write(x[i]:2);
readln;
begin
solutie:=(k=p);
end;
procedure tipar;
var i:integer;
begin
for i:=1 to p do write(st[i],' ');
writeln;
end;
begin
write('n= ');read(n);
write('p= ');read(p);
for i:=1 to n do
begin
write('L[',i,']=');readln(L[i]);
write('C[',i,']=');readln(C[i]);
end;
k:=1;init(k,st);
while k>0 do
      begin
        repeat
          succesor(as,st,k);
          if as then valid(ev,st,k);
        until (not as) or (as and ev);
 if as then if solutie(k) then tipar
 else begin
 k:=k+1;
 init(k,st);
  end
   else k:=k-1;
   end; end.
    end;
  else if x[k]<n then
  begin x[k]:=x[k]+1;
  if col(k) then k:=k+1;
   end;
   else begin x[k]:=0; k:=k-1; end;end.
```

Date Bibliografice.

- https://www.slideshare.net/BalanVeronica/metoda-backtracking?ref=http://informatica-clasa-11b.blogspot.com/p/metoda-reluarii.html

- https://www.slideshare.net/BalanVeronica/mada-34540933?ref=http://informatica-clasa-11b.blogspot.com/p/metoda-reluarii.html

- https://www.slideshare.net/BalanVeronica/metoda-relurii-34569777?ref=http%3A%2F%2Finformatica-clasa-11b.blogspot.com%2Fp%2Fmetoda-reluarii.html&fbclid=IwAR08cVQGhrC89VmpS4hGwSXuQUY-Z4RaAj9e823fqJursAsbdNSq7DAkhtA

- http://www.scritub.com/stiinta/informatica/METODA-BACKTRACKING1055131414.php?fbclid=IwAR1ydXGW1YKAkWTOJ3eTIBzDg7gfkpxSL5cxzJGV8ay14ySVhKqbDQ-l7J8