***METODA RELUĂRII***

**Slide nr1:**DEFINITIE

Aceasta metoda reprezintă mai mulți vectori ce formează o matrice.

Ea constă în efectuarea unei căutări în vederea găsirii unei soluții și revenirea

La alte soluții posibile.

***Metoda Backtracking***

Backtracking = parcurgerea limitată (conform

condiţiilor de continuare) în adâncime a unui

arbore.

Spaţiul soluţiilor este organizat ca un arbore

◦ un vârf este viabil dacă sunt şanse să se

găsească o soluţie explorând subarborele cu

rădăcina în acel vârf

◦ sunt exploraţi numai subarborii cu rădăcini

viabile

**Slide nr2:** ALGORITM

În metodele reluării se presupune ca soluția problemei pe care

trebuie să o rezolvăm poate fi reprezentată printr-un vector X=(X1,X2,...,Xk,...,Xn).

Fiecare componentă xk a vectorului X poate lua valori dintr-o anumită mulțime Ak,k=1,2,...,n.

Se consideră ca mk elemente ale fiecărei mulțimi Ak sunt ordonate conform unui criteriu stabilit.

Cînd valoarea pentru Xk este stabilă,se verifică anumite condiții de continuare referitoare la x1,x2,...,xk.

**Slide nr3:**AVANTAJUL METODEI DATE

Avantajul principal al algoritmilor bazați pe metoda reluării constăin faptul

ca respective sint relativ simple,iar depanarea lor nu necesită teste complicate.Complexitatea

temporală a acestor algoritmi este determinată de numărul de elemente k din mulțimea soluțiilor posibile S.

În majoritatea problemelor de o reală importanță practică metoda reluării conduce la algoritmi exponențiali.

DEZAVANTAJE:

Restricţiile listate aici se datorează în cea mai mare măsură trecerii de la scrierea de

mână a unui program, la proiectarea lui asistată. Aceeaşi problemă, apare spre exemplu la

orice mediu vizual, deoarece componentele furnizate nu pot acoperi infinita diversitate

necesară elaborării unei aplicaţii complexe. Oricum, ca şi în cazul acestor medii vizuale,

există, şi în aplicaţia mea, posibilitatea inserării de cod sursă, printre codul sursă generat.

Să analizăm câteva din limitările existente:

La ieşire se pot afişa doar părţi din vectorul soluţie, sau doar câteva din soluţii

(eventual toate dacă există), dar nu se poate afişa ceva combinaţie între elementele vectorului

soluţie. Pentru aceasta este nevoie ca programatorul să intervină în cadrul codului sursă

generat, mai precis trebuie să intervină în cadrul procedurii Afiseaza.

**Slide nr4:**PROGRAM GENERAL

Begin

ifk<=n then

Begin

X[k]:=PrimulElement (k);

IfContinuare(k) then Reluare (k+1);

While ExistaSuccesor(k) do

Begin

X[k]:=Succesor(k);

IfContinuare(k) then Reluare (k+1);

End; {while}

End {then}

Else PreluareaSolutiei;

End; {Reluare}

End.

**SLIDE NR5:**PROBLEME

**1. Generarea permutărilor**

Pentru această problemă trebuie specificate următoarele lucruri:

Numele fişierului de intrare este perm.in care conţine o singură valoare ce va fi citită în

variabila de tip byte n.

Tipul de bază al vectorului soluţie este byte.

Domeniul unui element este 1..n

Se doresc toate soluţiile

Condiţiile de continuare sunt de tip condiţionare de toate cele de dinainte, şi anume diferit

de toate cele de dinainte.

La final vectorul are lungimea n

Construirea vectorului se face pe baza domeniului

Codul sursă generat este următorul:

**Program Backtracking;**

**uses crt;**

**type TSolutie=array[0..20] of Byte;**

**var v:TSolutie;**

**solutii:array[1..100] of TSolutie;**

**nrsol:integer;**

**n:Byte;**

**procedure Citire;**

**var f:text;**

**begin**

**assign(f,'perm.in');**

**reset(f);**

**readln(f,n);**

**Close(f);**

**end;**

**procedure Afisare;**

**var i,k:byte;**

**begin**

**for k:=1 to nrsol do**

**begin**

**for i:=1 to n do**

**begin**

**write(solutii[k,i],',');**

**end;**

**writeln;**

**end**

**end;**

**function LungimeData(k:byte):boolean;**

**begin**

**LungimeData:=k=n;**

**end;**

**function final(k:byte):boolean;**

**begin**

**final:=LungimeData(k);**

**end;**

**function Continuare(pozitie:integer; element:Byte):boolean;**

**var c:byte;**

**begin**

**Continuare:=true;**

**for c:=1 to pozitie-1 do**

**if v[c]=element**

**then begin**

**Continuare:=false;**

**exit;**

**end;**

**end;**

**procedure back(k:integer);**

**var element:Byte;**

**y0:byte;**

**begin**

**if final(k-1)**

**then**

**begin**

**inc(nrsol);**

**solutii[nrsol]:=v;**

**end**

**else begin**

**for y0:=1 to n do**

**begin**

**element:=y0;**

**if Continuare(k,y0)**

**then begin**

**v[k]:=element;**

**back(k+1);**

**end;**

**procedure Initializeaza;**

**begin**

**nrsol:=0;**

**end;**

**begin**

**Citire;**

**Initializeaza;**

**back(1);**

**Afisare;**

**end.**

**2.** ***Generarea combinărilor***

Pentru această problemă trebuie specificate următoarele lucruri:

Numele fişierului de intrare este comb.in care conţine două valoari ce vor fi citite în

variabilele de tip byte n şi m.

Tipul de bază al vectorului soluţie este byte.

Domeniul unui element este v[k]+1..n

Se doresc toate soluţiile

La final vectorul are lungimea m

Construirea vectorului se face pe baza domeniului

Se iniţializează poziţia 0 cu 0.

**Program Backtracking;**

uses crt;

type TSolutie=array[0..20] of Byte;

var v:TSolutie;

solutii:array[1..100] of TSolutie;

n:Byte;

m:Byte;

nrsol:word;

procedure Citire;

var f:text;

begin

assign(f,'comb.in');

reset(f);

readln(f,n);

readln(f,m);

Close(f);

end;

procedure Afisare;

var i,k:byte;

begin

for k:=1 to nrsol do

begin

for i:=1 to m do

begin

write(solutii[k,i],',');

end;

writeln;

end

end;

function LungimeData(k:byte):boolean;

begin

LungimeData:=k=m;

end;

function final(k:byte):boolean;

begin

final:=LungimeData(k);

end;

function Continuare(pozitie:integer; element:Byte):boolean;

begin

Continuare:=true;

end;

procedure back(k:integer);

var element:Byte;

y0:byte;

begin

if final(k-1)

then

begin

inc(nrsol);

solutii[nrsol]:=v;

end

else begin

for y0:=v[k]+1 to n do

begin

element:=y0;

if Continuare(k,y0)

then begin

v[k]:=element;

back(k+1);

end;

procedure Initializeaza;

begin

nrsol:=0;

v[0]:=0;

end;

begin

Citire;

Initializeaza;

back(1);

Afisare;

end.

**3**. ***Problema labirintului:*** se cere să se ajungă din camera A în camera B printr-un drum de

lungime minima

**Program Backtracking;**

uses crt;

type poz=record x:byte; y:byte; end;

TSolutie=array[0..20] of pozitie;

var v:TSolutie;

solutie:TSolutie;

lungime:byte;

n:Byte;

m:Byte;

a:array[1..n,1..m] of byte;

ix:Byte;

iy:Byte;

fx:Byte;

fy:Byte;

min:integer;

procedure Citire;

var f:text;

i1, i2:byte;

begin

assign(f,'labirint.in');

reset(f);

readln(f,n);

readln(f,m);

for i1:=1 to n do

for i2:=1 to m do

readln(f,a[i1,i2]);

readln(f,ix);

readln(f,iy);

readln(f,fx);

readln(f,fy);

Close(f);

end;

procedure Afisare;

var i,k:byte;

begin

for i:=1 to min do

begin

write('(');

write(solutie[i].x,',');

write(solutie[i].y,',');

writeln(')');

end;

end;

function UltimaPozitie(k:byte):boolean;

begin

UltimaPozitie:=(v[k].x=fx) and(v[k].y=fy);

end;

function final(k:byte):boolean;

begin

final:= (a[v[k].x,v[k].y]=0) and UltimaPozitie(k);

end;

function Continuare(pozitie:integer; element:pozitie):boolean;

var c:byte;

begin

Continuare:=true;

for c:=1 to pozitie-1 do

if (v[c].x=element.x) and(v[c].y=element.y)

then begin

Continuare:=false;

exit;

end;

end;

procedure back(k:integer);

var element:pozitie;

begin

if final(k-1)

then

if k<min

then begin

min:=k-1;

solutie:=v;

end

else

else begin

element.x:=v[k-1].x+1;

element.y:=v[k-1].y+1;

if Continuare(k,element)

then begin

v[k]:=element;

back(k+1);

end;

element.x:=v[k-1].x+1;

element.y:=v[k-1].y-1;

if Continuare(k,element)

then begin

v[k]:=element;

back(k+1);

end;

element.x:=v[k-1].x-1;

element.y:=v[k-1].y+1;

if Continuare(k,element)

then begin

v[k]:=element;

back(k+1);

end;

element.x:=v[k-1].x-1;

element.y:=v[k-1].y-1;

if Continuare(k,element)

then begin

v[k]:=element;

back(k+1);

end;

end

end;

procedure Initializeaza;

begin

v[1].x:=ix;

v[1].y:=iy;

min:=10000;

end;

begin

Citire;

Initializeaza;

back(2);

Afisare;

end.

**4.** .Generarea partitiilor unei multimi

ENUNT : Fie n numar natural nenul. Scrieti un program de generare a

tuturor partitiilor multimii {1, 2, 3,..., n}.

EXEMPLU: Pentru n=3 programul va genera

{1} {2} {3}

{1} {2,3}

{2} {1,3}

{3} {1,2}

{1,2,3}

SOLUTIE: 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.

I. Solutia x[1]......x[n]. Fiecare multime din cadrul partitiei va fi

codificata cu numere de la 1 la n. Elementul x[i] va avea ca valoare

numarul asociat multimii din care face parte.

Solutia Codificare

{1,2,3} 1 1 1

{1,2} {3} 1 1 2

{1,3} {2} 1 2 1

{1} {2,3} 1 2 2

{1} {2} {3} 1 2 3

Implementare

#include<iostream>

using namespace std;

char sp[]=" ";

int x[20], n, nrsol=0, max[20], maxim;

int DeterminareMaxim(int k)

{int maxim=0,i;

for(i=1;i<=k;i++)

if (x[i]>maxim) maxim=x[i];

return maxim;

}

void Afisare()

{int i,j;

cout<<sp;

maxim=DeterminareMaxim(n);

for(j=1;j<=maxim;j++)

{cout<<"{";

for(i=1;i<=n;i++)

if (x[i]==j) cout<<i<<" ";

cout<<"} ";

}

cout<<endl;

nrsol++;

if (nrsol%22==0) cin.get();

}

void BackRec(int k)

{int i;

for(i=1;i<=DeterminareMaxim(k-1)+1;i++)

{ x[k]=i;

if (k==n) Afisare();

else BackRec(k+1);

}

}

int main()

{cout<<endl<<endl<<sp<<"Partiile multimii {1,2,3.....,n}"<<endl;

cout<<endl<<sp<<" Dati valoarea lui n: "; cin>>n;

cout<<endl;

BackRec(1);

cout<<endl<<sp<<"Numar solutii: "<<nrsol;

return 0;

}

5. .Generarea produsului cartezian

ENUNT : Fie n numar natural nenul si A1, A2, ... An n multimi cu L1,

L2, .... Ln elemente. Scrieti un program de generare a elementelor

produsului cartezian A1 x A2 x A3 x ... x An

EXEMPLU: Pentru n=3 si L=(1,2,3) programul va genera

{1 1 1}

{1 1 2}

{1 1 3}

{1 2 1}

{1 2 2}

{1 2 3}

SOLUTIE:

I. Solutia x[1]......x[n]. Elementul x[i] va avea ca valoare un element

al multimii Ai.

II. x[k] poate avea ca valoare orice valoare din intervalul [1,Lk].

III. Valid - nu este necesar

Implementare

#include<iostream>

using namespace std;

char sp[]=" ";

int x[20], n, nrsol=0, a[20];

void Afisare()

{int i,j;

cout<<sp;

for(i=1;i<=n;i++) cout<<x[i]<<" ";

cout<<endl;

nrsol++;

if (nrsol%24==0) cin.get();

}

void BackRec(int k)

{int i;

for(i=1;i<=a[k];i++)

{x[k]=i;

if (k==n) Afisare();

else BackRec(k+1);

}

}

int main()

{int i;

cout<<endl<<endl<<sp<<"Produsul cartezian a n multimi de cardinal a1, a2, ... an"<<endl;

cout<<endl<<sp<<" Dati valoarea lui n: "; cin>>n;

cout<<endl<<sp<<" Dati cardinalul fiecarei multimi: "<<endl<<endl;

for (i=1;i<=n;i++)

{ cout<<sp<<" Multimea "<<i<<": "; cin>>a[i];

}

BackRec(1);

cout<<endl<<sp<<"Numar solutii: "<<nrsol;

return 0;

Slide6:

Bibliografie:

<https://sites.google.com/site/metodabacktracking/probleme-rezolvate>

<https://www.slideshare.net/BalanVeronica/metoda-relurii>

<https://mihaioltean.github.io/generarea_automata_a_programelor_de_calculator.pdf>