Instituția Publică Liceul Teoretic “SpiruHaret”

PROIECT
LA INFORMATICĂ

TEMA:”Tehnici de elaborare a algoritmilor.

Metoda reluării.(BackTracking) “

Profesor: Guțu Maria

Efectuat de: Plamadeala Vladislav

Chișinău, 2019

Cuprins

Introducere

Pentru acei care rezolva problem cu ajutorul calculatorului exista mai multe metode. Din acestea cele mai des utiliate sunt:

-Metoda Greedy

-Metoda Divite et impera

-Metoda Branch and Bound

-Metoda Backtracking(reluarii)

Metoda Reluarii(Backtracking) se utilieaza la rezolvarea problemelor de cautare, generand toate solutiile posibile. Se construieste Solutia pas cu pas. Daca constata ca pentru o anumita valoare nu se ajunge la solutie, se renunta la acea valoare si se reia cautarea din punctul unde am ramas.

Metoda se aplică problemelor în care soluția poate fi reprezentată sub forma unui vector:

$X = (x_1, x_2, x_3, \ldots, x_k, \ldots, x_n)$.

Cerința problemeieste, de obicei, găsirea tuturor soluțiilor posibile sau găsirea numărului de soluții care satisfac anumite condiții specific problemei.

Fiecare component x_k a vectorului X poate lua valori dintr-o anumită mulțime A_k, k=1,2,…, n.Se consider că cele m_k elemente ale fiecărei mulțimi A_k sunt **ordonate** conform unui criteriu bine stabilit. Deci, în metoda reluării componentele vectorului X primesc valori pe rând, în sensul că lui x_ki se atribuie o valoare numai dacă au fost déjà atribuite valori lui$x_1, x_2, \ldots, x_{k-1}$.

Anume micșorarea lui k dă nume metodei, cuvântul *"reluare"*, semnificând revenirea la alte variante de alegere a variabilelor. Aceeaşi semnificație o are și denumirea engleză a metodei–*backtracking*

(back-înapoi, track - urmă).

Schema generală

Schema generală a unui algoritm recursive bazat pe metoda reluării este redată cu ajutorul procedurii ce urmează:

```
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;
```

Procedura Reluare comunică cu programul appellant şi subprogramele apelate
Prin variabilele globale ce reprezintă vectorul X şi multimile A_1, A_2,...,A_n. Sub programele apelate execută următoarele operații:

PrimulElement (k)–returnează primul element din mulțimeaA_k;

Continuare (k) – returnează valoare *true* dacă elementele înscrise în primele k componenteale vectorului X satisfac condițiile de continuare şi *false* în caz contrar;

ExistaSuccesor (k) – returnează valoarea *true* dacă elemental memorat în component x_k are un successor în multimea A_k şi *false* în caz contrar;

Succesor (k)–returnează succesorul elementului memorat în component x_k;

PrelucrareaSolutiei– de obicei, în această procedură soluți are ținută în vectorul X este afişată la ecran;

Probleme rezolvate

Problema celor n regine Fiind dată o tablă de şah de dimensiune n x n, se cer toate soluţiile de aranjare a n regine astfel încât să nu atace (să nu se afle două regine pe aceeaşi linie, coloană sau diagonală).

- Reprezentarea soluţiei ?
- Condiţii interne ?
- Condiţii de continuare ?

- ✓ $x = (x_1, x_2, \ldots, x_n)$
- ✓ Condiţii interne: pentru orice $i \neq j$, $x_i \neq x_j$
- ✓ Condiţii de continuare: pentru orice $\mathbf{i<k}$, $x_i \neq x_k$

```
function cont(x:stiva; k:integer):boolean;
 var i:integer;
 begin
      cont:=true;
       for i:=1 to k-1 do
            if (x[i]=x[k]) or
           (abs(x[k]-x[i])=abs(k-i))
       then
            cont:=false;
end;
```

Generarea produsului cartezian a n multimi

Se consideră n mulțimi finite S_1, S_2, …S_n, de forma {1,2..,S_n}. Să se genereze produsul cartezian al acestor mulțimi.

Indicatii de rezolvare:

Am considerat mulțimile de forma {1,2..,S_n}pentru a simplifica problema, în special la partea de citire și afișare, algoritmul de generare ramânând nemodificat.

Ldentificăm următoarele particularități şi condiții:

- Vectorul soluție: X=(x_1, x_2,... x_n) ∈ S1 x S2x...Sn
- Fiecare element X_k∈ S_k
- Nu există condiții interne pentru vectorul soluție. Nu are funcție de validare.
- Obținem soluția când s-au generat n valori.
- Avem soluție dacă k=n+1

```
//generare prod cartezian
#include <iostream>
#include <fstream>
using namespace std;
int x[50], n,k,s[50][100], card[50];
ifstream f("fis.in");
void citeste()
{  int i,j; f>>n;
   for(i=1;i<=n;i++)
   {   f>>card[i];
       for(j=1;j<=card[i];j++) f>>s[i][j];}}
void scrie()
{   int i;cout<<endl;
    for(i=1;i<=n;i++) cout<<s[i][x[i]]<<" ";}
int solutie(int k) { return(k==n+1);}

void back(int k)
{   if(solutie(k))scrie();
    else
     for(int i=1;i<=card[k];i++)
       {x[k]=i;
         // if(valid(k)) nu este cazul
        back(k+1);}
}
int main()
{citeste(); back(1); return 0;}
```

Generarea aranjamentelor

Generarea aranjamentelor-Se citesc două numere naturale n şi p. Să se genereze toate submulţimile mulţimii {1,2, ..., n} de p elemente. Două mulţimi cu aceleaşi elemente, la care ordinea acestora diferă, sunt considerate diferite.

- Reprezentarea soluţiei ?
- Condiţii interne ?
- Condiţii de continuare ?

- ✓ $x = (x_1, x_2, \ldots, x_n)$
- ✓ Condiţii interne: pentru orice $i \neq j$, $x_i \neq x_j$
- ✓ Condiţii de continuare: pentru orice $\mathbf{i<k}$, $x_i \neq x_k$

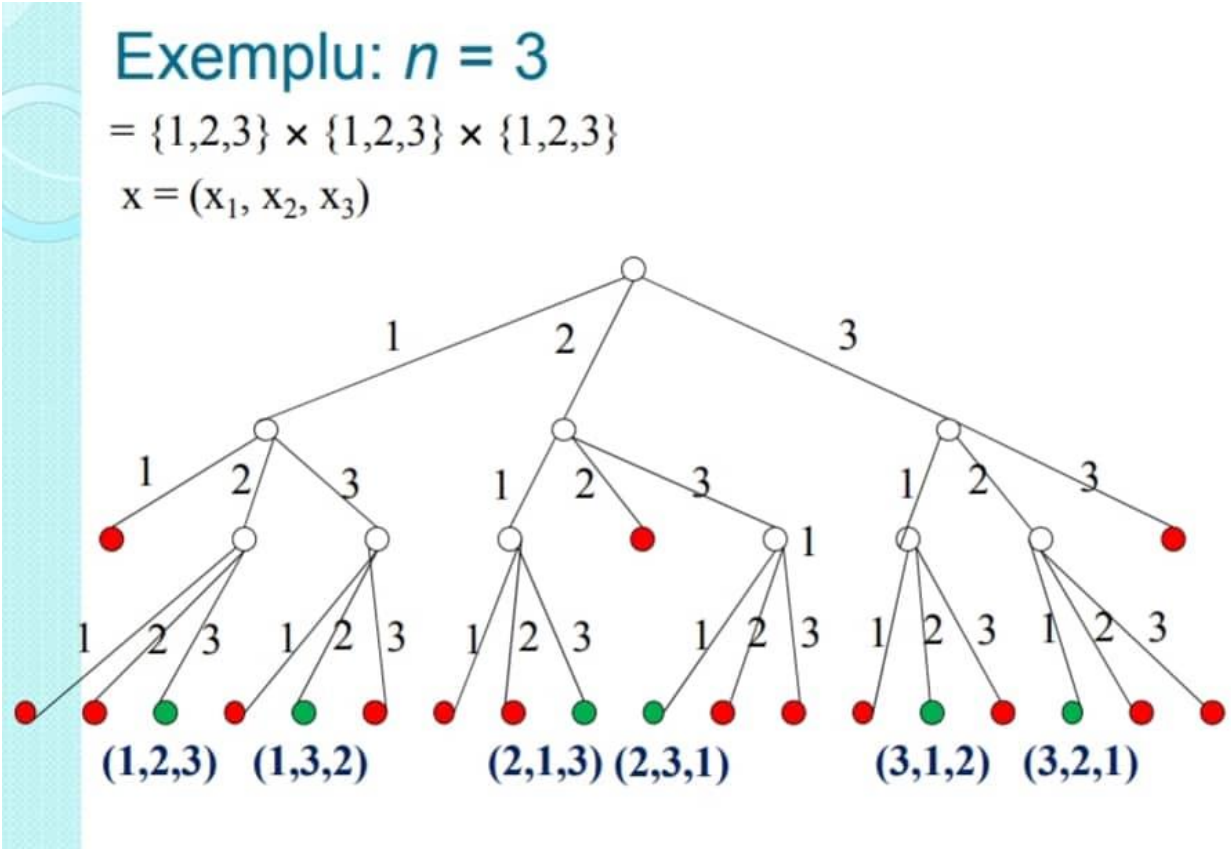
```
procedure tipar(x:stiva);
var i:integer;
begin
      nrsol := nrsol+1;
      write('Solutia ',nrsol,': ');
      for i:=1 to p do
            write(x[i],' ');
      writeln;
end;
```

Generarea permutărilor

Se citește un număr natural n. Să se generezetoatepermutărilemulțimii {1,2,…,n}.

- Reprezentarea soluţiei ?
- Condiţii interne ?
- Condiţii de continuare ?

- ✓ $x = (x_1, x_2, \ldots, x_n)$
- ✓ Condiţii interne: pentru orice $i \neq j$, $x_i \neq x_j$
- ✓ Condiţii de continuare: pentru orice $\mathbf{i<k}$, $x_i \neq x_k$



Colorarea Hartilor

Problema matematică: orice hartă poate fi colorată utilizând cel mult 4 culori (rezultat demonstrat în 1976 de către Appel and Haken – unul dintre primele rezultate obţinute folosind tehnica demonstrării asistate de calculator)

Colorarea hărţilor

- **Reprezentarea soluţiei**: $x = (x_1, x_2, \ldots, x_n)$, unde x_k = culoarea curentă cu care este colorată ţara k, $x_k \in \{1,2,3,4\}$
- **Condiţiile interne**: $x_i \neq x_j$ pentru orice două ţări vecine i şi j.
- **Condiţii de continuare**: $x_i \neq x_k$ pentru orice ţară $i \in \{1,2,\ldots,k-1\}$ vecină cu ţara k.
- Folosim o matrice **a** pentru a memora relaţia de vecinătate dintre ţări:

$$a_{ij} = \begin{cases} 1, \text{ dacă ţările i şi j sunt vecine} \\ 0, \text{ altfel} \end{cases}$$

```
function cont(x:vector;
k:integer):boolean;
 var i:integer;
 begin
 cont:=true;
 for i:=1 to k do
  if((x[i]=x[k]) and
     a[k,i]=1)) then
       cont:=false;
end;
```

Concluzie

Metoda backtracking se aplică problemelor în care soluția se poate prezenta sub forma unui vector $x=\{x_1,x_2,\ldots, x_n\}$ unde x1 aparține unei mulțimi S1, x2 aparține mulțimii S2 s.a.m.d. Cerința problemei este, de obicei, găsirea tuturor soluțiilor posibile sau găsirea numărului de soluții care satisfac anumite conditii specifice problemei. De multe ori metoda se foloseste si pentru gasirea unei singure solutii (dupa gasirea acesteia se intrerupe executia programului), a unei solutii maxime/minime.

Metoda de generare a tuturor solutiilor posibile si apoi de determinare a solutiilor rezultat prin verificarea îndeplinirii conditiilor interne necesita foarte mult timp.Metoda backtracking evita aceasta generare si este mai eficienta. Metoda backtracking urmăreşte evitarea generării tuturor solutiilor posibile, micşorându-se astfel complexitatea algoritmului şi scurtându-se timpul de executie. Trebuie precizat de la bun început că la nici un caz (exceptând eventual câteva cazuri particulare) nu vom reduce complexitatea algoritmului de la una exponentială la una polinomială, ci doar la una mai putin exponentială (de exemplu, cum am spus şi mai sus, de la un 3^n la un 2^n).

Tipuri de probleme ce pot fi rezolvate cu această metodă:

1. Generarea permutărilor, aranjamentelor, combinarilor
2. Problema celor n regine
3. Colorarea hărtilor
4. Siruri de paranteze ce se închid corect
5. Partitiile unui număr natural
6. Generarea unor numere cu proprietăti specificate
7. Generarea produsului cartezian
8. Generarea tuturor solutiilor unei probleme, pentru a alege dintre acestea o solutie care minimizează sau maximizeaza o expresie

Bibliografie

- http://www.scritub.com/stiinta/informatica/METODA-BACKTRACKING1055131414.php
- https://www.slideshare.net/BalanVeronica/catalinametoda-relurii
- https://sites.google.com/site/matriciinformaticapascal/home/metoda-backtracking
- http://www.lbi.ro/~carmen/vineri/teorie_backtraking.pdf?fbclid=IwAR1iEmemm7YAir0wLwbQQXHjUEVOdMWoTM5Wo8hTq5lTSt6qNrQTlxZUXxA