METODA BACKTRACKING

Este o metoda folosită în cazul problemelor în care se cere afişarea tuturor soluţiilor, iar o soluţie poate fi dată sub forma unui vector.

Fiecare elemnt al vectorului aparţine la rândul său unei mulţimi finite de elemente intregi, aflate într-o ordine bine stabilită.

De reţinut!!!:

- Soluţia se construieşte pas cu pas, pornind de la primul element şi adăugând la vector următoarele elemente, cu revenire la elementul anterior în caz de insucces;
- Elementul care trebuie adăugat se caută în mulţime printre elementele care respectă restricţiile specifice problemei (condiţiile interne)

Exemple de probleme care pot fi rezolvate cu ajutorul metodei backtracking:

- Generarea permutărilor multimii care contine n elemente {1,2,3...,n};.
  Expl: pentru n=3 se vor obţine următoarele soluţii:
  1 2 3
  1 3 2
  2 1 3
  2 3 1
  3 1 2
  3 2 1
- Generarea aranjamentelor multimii care contine n elemente luate cite p;.
  Expl: pentru n=3 si p=2 se vor obţine următoarele soluţii:
  1 2
  1 3
  2 1
  2 3
  3 1
  3 2

MECANISMUL METODEI BACKTRACKING:

Pentru implementarea acestei metode se foloseşte o rutina principală care conţine mai multe funcţii, cu rol bine stabilit. În general rutina principală rămâne neschimbată în toate tipurile de probleme, schimbându-se doar funcţiile din cadrul rutinei.

Vom prezenta în continuare implementarea metodei backtracking pentru cazul *permutorilor de n elemente*.

Rutina generală:

.....

```
void back()
{
  k=1; // ma pozitionez pe prima poziţie din vectorul solutie
  Init(k); // iniţializez prima poziţie din vectorul soluţie cu o valoare de start
  while (k>0) // cât timp mai sunt în vectorul soluţie
        {
              if (Maisuntvalori(k)==1) //verific daca mai sunt valori netestate incă
              {
                    sol[k]++; // înaintez în vectorul în care ne construim soluţia
                    if (E_valid(k)) // testez dacă elementul este valid
                          if (Solutie(k)==1) //testez dacă noua valoare este soluţie
                                Tipar(k); // tipăresc soluţia
                          else
                             {
                                k++; // cresc nivelul în vectorul soluţie
                                Init(k);   // iniţializez noua poziţie
                             }
              }
              else k--; // cobor nivelul(ma intorc) în vectorul soluţie
        }
}
```

.....

Observaţii:

În vectorul sol[] ne vom construi soluţia, pas cu pas;

Indicele k indică poziţia curentă în vectorul soluţie (la început k=1 construirea soluţiei pornindu-se de pe prima poziţie);

Funcţia Init(k) – iniţializează orice nouă poziţie în vectorul soluţie cu o valoare de start(FOARTE IMPORTANT: poziţia va fi iniţializată cu prima valoare aflată înaintea primei valori posibile dintr-o eventuală soluţie. În cazul permutărilor cea mai mica valoare posibila este 1 deci iniţializarea se va face cu 0);

Funcţia E_valid(k) – verifică dacă o nouă valoare pusă pe poziţia k poate face parte dintr-o eventuală soluţie;

Funcţia Soluţie(k) – verifică dacă noua valoare validă poate forma deja o soluţie;

Funcţia MaiSuntValori(k) – verifică dacă pe poziţia k mai sunt valori netestate încă(pe fiecare poziţie a vectorului soluţie există un domeniu maxim de valori care de multe ori e acelaşi pentru fiecare poziţie a vectorului)

Funcţia Tipar(k) – tipăreşte vectrul soluţie până la poziţia k

Implementarea funcţiilor pentru problema permutărilor:

Funcţia Init(k):

.....

```
void Init(int k)
{
  sol[k]=0;
}
```

.....
Funcţia E_valid(k):
.....
```
int E_valid(int k)
{
  for(int i=1;i<k;i++)
     if (sol[i]==sol[k])
            return 0;
  return 1;
}
```
.....
Funcţia Soluţie(k)
.....
```
int Solutie(int k)
  { if(k==n)
      return 1;
    else return 0;
            //   return (k==n);
  }
```
.....
Funcţia MaiSuntValori(k)
.....
```
int Maisuntvalori(int k)
  { if (sol[k]<n) return 1;
    else return 0;}      //  return (sol[k]<n);
```
.....
Funcţia Tipar(k)
.....
```
void Tipar(int k)
   { for(int i=1;i<=k;i++)
       cout<<sol[i]<<"  ";
     cout<<endl;
   }
```
.....

Probleme:

1. **Să se afișeze aranjamente de n luate cite p.**
   Rezolvare:

```
#include<iostream.h>
int sol[20],n,k,p,nrsol;

void Init(int k)
{
  sol[k]=0;
}

int Maisuntvalori(int k)
```

```
  { if (sol[k]<n) return 1;
    else return 0;}       //  return (sol[k]<n);

int E_valid(int k)
{
  for(int i=1;i<k;i++)
     if (sol[i]==sol[k])
       return 0;
  return 1;
}
////////////////////////////////////
int Solutie(int k)
  { if(k==p)
      return 1;
    else return 0;
        //   return (k==n);
  }
/////////////////////////////////////////
 void Tipar(int k)
   { for(int i=1;i<=k;i++)
       cout<<sol[i]<<"  ";
     cout<<endl;
   }
//////////////////////////////////////////
void back()
{
 k=1;
   Init(k); nrsol=0;
 while (k>0)
 { if (Maisuntvalori(k)==1)
      {           sol[k]++;
       if (E_valid(k))
           if (Solutie(k)==1)
                       {Tipar(k);
                       nrsol++;}
           else {k++;
                    Init(k);   }
       }
  else k=k-1;
} }
////////////////////////
void main()
{ cout<<"n=";
  cin>>n;
  cout<<"p=";
  cin>>p;
  back();
  cout<<"Sunt "<<nrsol<<" solutii!"<<endl;
}
```

2. **Să se afișeze toate posibilitățile în care se pot aranja corect n paranteze**
Rezolvare:

```
#include<iostream.h>
int sol[20],n,k,p,nrsol;

void Init(int k)
{
  sol[k]=-1;
}

int Maisuntvalori(int k)
 { if (sol[k]<1) return 1;
   else return 0;}      //  return (sol[k]<n);

int E_valid(int k)
{  int nr0=0,nr1=0;
 for(int i=1;i<=k;i++)
    if (sol[i]==0) nr0++;
      else nr1++;
if((nr1>nr0)||(nr0>n/2)||(nr1>n/2))
      return 0;
 return 1;
}
////////////////////////////////////
int Solutie(int k)
 { if(k==n)
     return 1;
   else return 0;
       //   return (k==n);
 }
/////////////////////////////////////////
 void Tipar(int k)
  { for(int i=1;i<=k;i++)
     if(sol[i]==1) cout<<")";
     else cout<<"(";
    cout<<endl;
  }
/////////////////////////////////////////////
void back()
{
 k=1;// ma poyitioney pe prima poy din vectoru solutie
  Init(k); nrsol=0;
 while (k>0)
 { if (Maisuntvalori(k)==1)
     {          sol[k]++;
      if (E_valid(k))
          if (Solutie(k)==1)
```

```
                    {Tipar(k);
                    nrsol++;}
          else {k++;
                Init(k);   }
      }
  else k=k-1;
} }
//////////////////////
void main()
{ cout<<"n=";
  cin>>n;
  back();
  cout<<"Sunt "<<nrsol<<" solutii!"<<endl;
}
```