Aplicații - metoda Backtracking

1) Generarea permutarilor multimii {1,2,…n} – 123, 124, 125, 1327, 202, 3150, 3161, 3158, 3156

```
int x[100], n; // variabile globale

void afis (int k)  // afiseaza solutia curenta, adică vectorul x cu k elemente
{    int i;
     for (i=1; i<=k; i++) cout <<x[i]<<" ";
     cout <<endl;
 }

 int valid (int k) // verifica conditiile de continuare pentru x[k]
{ int i;
   for (i=1; i<k; i++)
       if (x[k]==x[i]) return 0;
    return 1;
  }

 void bkt (int k)    // k = indicele elementului din vârful stivei
 {   int i;
     for (i=1; i<=n; i++)
       { x[k]=i;  // i se da lui x[k] urmatoarea valoare din multimea de valori posibila
         if (valid(k))  // daca x[k] satisface conditiile de continuare
              if (k==n) afis(k);  // daca s-a ajuns la o solutie se afiseaza
              else bkt(k+1);  // altfel, se trece la alegerea lui x[k+1] prin autoapel
        }
   }
```

Apel: bkt(1);

- **Modelarea problemei**

I) x[k] ∈ {1,2,…,n} , oricare ar fi k=1,n
// for (i=1; i<=n; i++) din functia **bkt()**

II) Conditiile de continuare pentru x[k]   // functia **valid()**
elemente distincte:
x[k] ≠ x[i], oricare ar fi i=1,k-1

III) S-a ajuns la o solutie daca k=n   // conditia din functia **bkt()** pe care apelez **afis()**

Obs.
1) Permutari pe un vector oarecare a=(a1, a2, ….an)
X=(1,2,3) -> a1, a2, a3
X=(1,3,2) -> a1, a3, a2
….
X=(3,2,1) -> a3, a2, a1
In afis() : cout <<a[x[i]]; // se afiseaza elementele cu indicii permutati

2) Numarare solutii – in variabila nr globala

nr++; // in functia afis()

***) Solutie eficienta ca timp de executare**

```
int use[10];  // variabila globala
void bkt (int k)
{
   int i;
   for(i=1;i<=n;i++)
     if (!use[i])
   {
      x[k]=i;
      use[i]=1;
      if(k==n) afis(k);
      else bkt (k+1);
      use[i]=0;
   }
}
```

2) Generarea aranjamentelor din n luate cate m, pe multimea {1,2,…n} - 196

Exemplu: n=5, m=3

1, 2, 3
1, 2, 4
1, 2, 5
1, 3, 2
1, 3, 4
1, 3, 5
1, 4, 2

5, 4, 3

- **Modelarea problemei**

I) **x[k] ∈ {1,2,…,n} , oricare ar fi k=1,m**
II) **Conditii de continuare pentru x[k] - elemente distincte**
x[k] ≠ x[i], oricare ar fi i=1,k-1
III) **S-a ajuns la o solutie daca k=m**

3) Generarea combinarilor din n luate cate m, pe multimea {1,2,…n} – 197, 204, 3152
Exemplu: n=5, m=3

1, 2, 3
1, 2, 4
1, 2, 5
1, 3, 4
1, 3, 5
1, 4, 5
2, 3, 4
2, 3, 5

2, 4, 5
3, 4, 5

Sol 1)

- **Modelarea problemei**
  I) x[k] ∈ {1,2,…,n} , oricare ar fi k=1,m
  II) Conditii de continuare pentru x[k] - vectorul x este strict crescator
  x[k]>x[k-1], oricare ar fi k>1
  III) S-a ajuns la o solutie daca k=m

```
int valid (int k)
{ int i;
  if( k>1 && x[k]<=x[k-1]) return 0;
  return 1;
 }
```

Sol 2) // optimizata ca timp

- **Modelarea problemei**
  I) x[k] ∈ {x[k-1]+1, … ,n-m+k} , oricare ar fi k=1,m //x[k]>x[k-1] sir strict crescător
  I) nu există *valid()*
  II) S-a ajuns la o solutie daca k=m

```
void bkt(int k)
{
   int i;
   for(i=x[k-1]+1; i<=n-m+k; i++)
   {
      x[k]=i;
      if(k==m) afis(k);
      else bkt(k+1);
   }
}
```

4) Generarea submultimilor multimii {1,2,…n} - 198, 1286, 3247

Exemplu:
n=3
1
1 2
1 2 3
1 3
2
2 3
3

Solutia 1

- **Modelarea problemei**

  I) x[k] ∈ {1,2,…,n} , oricare ar fi k
  II) Conditiile de continuare pentru x[k]
  Vectorul x este strict crescator
  x[k]>x[k-1], oricare ar fi k>1
  III) Oricare ar fi k<=n avem solutie

```
int valid (int k)
{ int i;
  if( k>1 && x[k]<=x[k-1]) return 0;
  return 1;
 }

void bkt(int k)
{
   int i;
   for(i=1; i<=n; i++)
   {
      x[k]=i;
      if(valid(k))
         { afis(k);
          if (k<n)  bkt(k+1);
         }
   }
}
```

Solutia 2

- **Modelarea problemei**

  II) x[k] apartine {x[k-1]+1,…,n} , oricare ar fi k //x[k]>x[k-1]
  III) Nu exista *valid()*
  IV) Oricare ar fi k<=n avem solutie

```
void bkt(int k)
{
   int i;
   for(i=x[k-1]+1; i<=n; i++)
   {
       x[k]=i;
       afis(k);
       bkt(k+1);
   }
}
```

Tema: 198, 1286, 3247

5) Problema celor n dame – 1281

Modelarea problemei – pe fiecare linie se asaza exact o dama -> x[k] = coloana pe care se asaza dama din linia k

I) **x[k]** ∈ **{1,2,…,n}** , oricare ar fi k=1,n

II) Conditiile de continuare pentru x[k]

a) Dama din linia k nu trebuie sa se atace **pe coloana** cu nicio dama de pe liniile anterioare (1, k-1) -> **x[k]≠x[i], oricare ar fi i=1,k-1**

b) Dama din linia k nu trebuie sa se atace **pe diagonala** cu nicio dama de pe liniile anterioare (1, k-1) -> **k-i≠|x[k]-x[i]| , oricare ar fi i=1,k-1**

III) S-a ajuns la o solutie daca: **k=n**

Exemplu:

N=4 **x=(3, 1, 4, 2) // alta solutie x=(2, 4, 1, 3)**

		D	
D			
			D
	D		

```
int valid(int k)
{
   int i;
   for(i=1;i<k;i++)
      if(x[k]==x[i]  || abs(x[k]-x[i]) ==k-i) return 0;
   return 1;
}

void afis()
{ int i, j;
for(i=1;i<=n;i++)
  {for (j=1; j<=n; j++)
      if(x[i]==j ) cout<<"* ";
      else cout<<"- ";
    cout<<'\n';
}
exit(0); // <cstdlib>
}
```

6) Produs cartezian de multimi – 1278, 1277

Ex: n=3, v=(2,1,3) // vectorul de cardinale

A1: 1,2
A2: 1
A3: 1, 2, 3

1 1 1
1 1 2
1 1 3
2 1 1
2 1 2
2 1 3

1278)

Modelarea problemei

I) **x[k]** ∈ **{1,2,..v[k]}** , oricare ar fi k=1,n

II) Nu exista *valid()*

III) S-a ajuns la o solutie daca: **k=n**