

# PROIECTAREA ALGORITMILOR

Adrian Runceanu

## Curs 10

# Metoda Backtracking

## Conținutul cursului

- 10.1. Prezentarea generala a metodei**
- 10.2. Implementarea metodei backtracking**
- 10.3. Probleme propuse spre rezolvare**

## 10.1 Prezentarea generala a metodei

Această tehnică se folosește în rezolvarea problemelor care îndeplinesc simultan următoarele condiții:

- soluția lor poate fi pusă sub forma unui vector  $S = \{ x_1, x_2, \dots, x_n \}$ , cu  $x_1 \in A_1, x_2 \in A_2, \dots, x_n \in A_n$
- mulțimile  $A_1, A_2, \dots, A_n$  sunt mulțimi finite, iar elementele lor se consideră că se află într-o relație de ordine bine stabilită
- nu se dispune de o altă metodă de rezolvare, mai rapidă
- $x_1, x_2, \dots, x_n$  pot fi la rândul lor vectori
- $A_1, A_2, \dots, A_n$  pot coincide

## 10.1 Prezentarea generala a metodei

- La întâlnirea unei astfel de probleme, dacă nu cunoaştem această tehnică, suntem tentaţi să generăm toate elementele produsului cartezian  $A_1 \times A_2 \times \dots \times A_n$  si fiecare element să fie testat dacă este soluţie.
- Rezolvând problema în acest mod, timpul de execuţie este atât de mare, încât poate fi considerat infinit, algoritmul neavând nici o valoare practică.

## 10.1 Prezentarea generala a metodei

- De exemplu, dacă dorim să generăm toate permutările unei mulțimi finite  $A$ , nu are rost să generăm produsul cartezian  $A \times A \times \dots \times A$ , pentru ca apoi, să testăm, pentru fiecare element al acestuia, dacă este sau nu permutare (nu are rost să generăm  $1, 1, 1, \dots, 1$ , pentru ca apoi să constatăm că nu am obținut o permutare, când de la a doua cifră 1 ne puteam da seama că cifrele nu sunt distincte).

## 10.1 Prezentarea generala a metodei

Tehnica ***Backtracking*** are la bază un principiu extrem de simplu:

- *se construiește soluția pas cu pas:  $x_1, x_2, \dots, x_n$*
- *dacă se constată că, pentru o valoare aleasă, nu avem cum să ajungem la soluție, se renunță la acea valoare și se reia căutarea din punctul în care am rămas*

## 10.1 Prezentarea generala a metodei

### **Concret:**

- se alege primul element  $x_1$ , ce aparține lui  $A_1$
- presupunând generate elementele  $x_1, x_2, \dots, x_k$ , aparținând mulțimilor  $A_1, A_2, \dots, A_k$ , se alege (dacă există)  $x_{k+1}$ , primul element disponibil din mulțimea  $A_{k+1}$ , apar două posibilități:
  - 1) ***Nu s-a găsit un astfel de element***, caz în care se reia căutarea considerând generate elementele  $x_1, x_2, \dots, x_{k+1}$ , iar aceasta se reia de la următorul element al mulțimii  $A_k$  rămas netestat

2) **A fost găsit**, caz în care se testează dacă acesta îndeplinește anumite **condiții de continuare** apărând astfel două posibilități:

- **îndeplinește**, caz în care se testează dacă s-a ajuns la soluție și apar din nou două posibilități:
  - **s-a ajuns la soluție**, se tipărește soluția și se reia algoritmul considerând generate elementele  $x_1, x_2, \dots, x_k$ , (se caută în continuare, un alt element al mulțimii  $A_k$ , rămas netestat)
  - **nu s-a ajuns la soluție**, caz în care se reia algoritmul considerând generate elementele  $x_1, x_2, \dots, x_{k+1}$ , și se caută un prim element  $x_{k+2} \in A_k$

- **nu le îndeplinește**, caz în care se reia algoritmul considerând generate elementele  $x_1, x_2, \dots, x_k$ , iar elementul  $x_{k+1}$  se caută între elementele mulțimii  $A_k$ , rămase netestate
- Algoritmii se termină atunci când nu mai există nici un element  $x_1 \in A_1$  netestat.

### **Observație:**

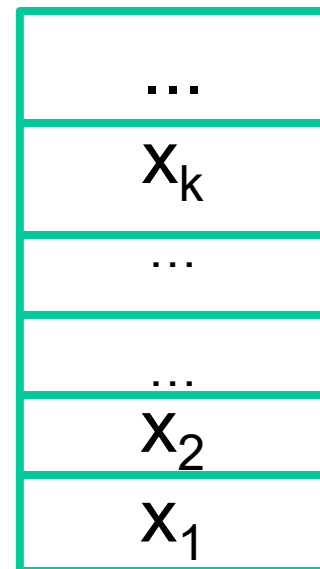
Tehnica **Backtracking** are ca rezultat obținerea tuturor soluțiilor problemei.

În cazul în care se cere o singură soluție se poate forța oprirea, atunci când a fost găsită.

- Am arătat că orice soluție se generează sub formă de vector.
- Vom considera că *generarea soluțiilor se face într-o stivă*.
- Metoda backtracking se poate implementa și recursiv, dar implementarea iterativă folosește o stivă, iar principiul de lucru este cel specific stivei – **LIFO (Last In First Out) – Ultimul Intrat Primul Iesit**

## 10.1 Prezentarea generala a metodei

- Astfel,  $x_1 \in A_1$ , se va găsi pe primul nivel al stivei,  $x_2 \in A_2$  se va găsi pe al doilea nivel al stivei, ...,  $x_k \in A_k$  se va găsi pe nivelul k al stivei.
- În acest fel, stiva (notată **ST**) va arăta astfel:



## 10.1 Prezentarea generala a metodei

- Nivelul  $k+1$  al stivei trebuie inițializat (pentru a alege, în ordine, elementele mulțimii  $k+1$ ).
- Inițializarea trebuie făcută cu *o valoare aflată* (în relația de ordine considerată, pentru mulțimea  $A_{k+1}$ ) *înaintea tuturor valorilor posibile din mulțime*.

## Conținutul cursului

**10.1. Prezentarea generala a metodei**

**10.2. Implementarea metodei backtracking**

**10.3. Probleme propuse spre rezolvare**

## Generarea permutărilor

Enunț:

Se citește un număr natural  $n$ . Se cere să se genereze toate permutările mulțimii  $\{1, 2, \dots, n\}$

Dacă  $n=3$ , mulțimile  $A_1 = A_2 = A_3 = \{1, 2, 3\}$

Soluțiile ar fi:

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

*Observație:*

Se observă, că elementele mulțimilor sunt în ordine strict crescătoare și consecutive.

*Așadar modul de alegere din mulțime nu este aleatoriu ci se face în funcție de ordinea elementelor.*

Înainte de a scrie programul care ne va obține soluțiile, trebuie să stabilim unele detalii cu privire la:

1. **vectorul soluție** – câte componente are, ce conține fiecare componentă.
2. **mulțimea de valori posibile** pentru fiecare componentă (sunt foarte importante limitele acestei mulțimi).
3. **condițiile de continuare** (condițiile ca o valoare  $x_k$  să fie acceptată).
4. **condiția ca ansamblul de valori generat să fie soluție.**

- pentru generarea permutărilor mulțimii  $\{1,2,\dots,n\}$ , orice nivel al stivei va lua valori de la 1 la  $n$ .
- Inițializarea unui nivel (oarecare) se face cu valoarea 0.
- Procedura de inițializare o vom numi **INIT** și va avea doi parametri: **k** (nivelul care trebuie inițializat și **ST** (stiva).

- Găsirea următorului element al mulțimii  $A_k$  (element care a fost netestat) se face cu ajutorul procedurii **SUCCESSOR (AS,ST,K)**.
- Parametrul **AS (am succesor)** este o variabilă booleană.
- În situația în care am găsit elementul, acesta este pus în stivă și **AS** ia valoarea **TRUE**, in caz contrar (nu a rămas un element netestat) **AS** ia valoarea **FALSE**.

- Odată ales un element, trebuie văzut dacă acesta îndeplinește **condițiile de continuare** (altfel spus, dacă elementul este valid).
- Acest test se face cu ajutorul procedurii **VALID (EV,ST,K)**.
- Parametrul **EV (este valid)** este o variabilă booleană.
- În situația în care elementul îndeplinește **condițiile de continuare**, **EV** ia valoarea **TRUE**, în caz contrar (nu îndeplinește **condițiile de continuare**) **EV** ia valoarea **FALSE**.

- Testul dacă s-a ajuns sau nu la soluția finală se face cu ajutorul funcției **SOLUTIE(k)**
- O soluție se tipărește cu ajutorul procedurii **TIPAR.**

*Schema  
generală a  
algoritmului  
în  
pseudocod  
este:*

```

procedura init;
begin
    st[k]<-valoare initiala de pornire
end;
procedura succesor;
begin
    daca st[k]<valoarea ultimului element din S atunci
    begin
        st[k]<-st[k]+1;
        as<-true
    end
    altfel      as<-false;
end;
procedura valid;
begin
    ev<-true;
În continuare se verifică condițiile ce trebuie să îndeplinite de
elementele aflate în stiva până la nivelul dat k.
Condițiile se stabilesc astfel încât în cazul îndeplinirii lor
variabila ev să primească de fiecare dată valoarea false
(condițiile se neagă!!!)
end;
functia solutie();
begin
    daca stiva este plina atunci solutie<-true
    altfel solutie<-false;
end;
procedura tipar;
begin
    Se parcurge stiva și se tipăresc valorile aflate acolo;
end;

```

# Programul principal este de fapt **algoritmul cu revenire backtracking**

```

begin
    se citesc date de intrare
    k<-1;
    init;
    cat timp k > 0 executa
    begin
        repeta
            sucesor;
            daca as atunci valid
        pana cand (as si ev) sau (not as);
        daca as atunci
            daca solutie atunci tipar
                altfel
                    begin
                        k<-k+1;
                        init
                    end
                altfel
                    k<-k-1;
        end;
    end;
end.

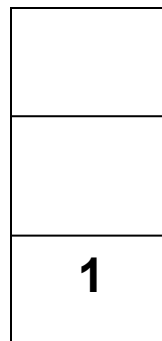
```

Pentru enunțul de la exemplul anterior, generăm, cu ajutorul stivei, soluțiile prin metoda backtracking.

Pentru  $n=3$ :

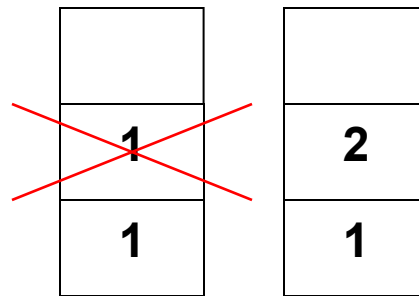
pas 1:

- Se ia primul element din mulțimea  $A_1$  și se pune pe nivelul 1 al stivei.
- Nu există deocamdată nici o restricție deoarece există permutări ce încep cu 1.
- Se trece la completarea următorului nivel al stivei.



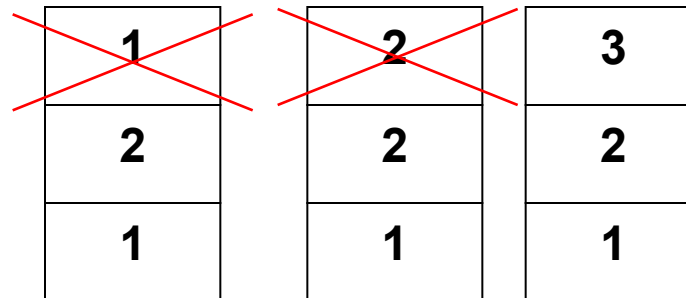
## pas 2:

- Se ia primul element din mulțimea  $A_2$  și se pune pe nivelul 2 al stivei.
- Se observă că valoarea 1 a mai fost introdusă în vectorul soluție, pe nivelul anterior, și, în acest caz valoarea 1 nu poate fi atașată la vectorul soluție.
- Așadar, se ia următorul element din mulțimea  $A_2$ , adică 2.
- Se constată că poate fi introdusă în vectorul soluție.
- Se trece la completarea următorului nivel al stivei.



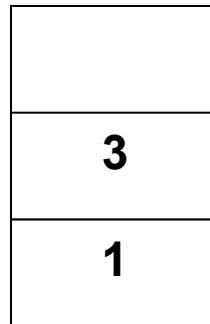
## pas 3:

- Se ia primul element din mulțimea  $A_3$  și se pune pe nivelul 3 al stivei.
- Se observă că valoarea 1 a mai fost introdusă în vectorul soluție, pe nivelul anterior, și, în acest caz valoarea 1 nu poate fi atașată la vectorul soluție.
- Așadar, se ia următorul element din mulțimea  $A_3$ , adică 2.
- La fel ca valoarea 1, 2 nu poate fi introdus în stivă.
- Se trece la următorul element din mulțime, 3.
- Se constată că poate fi introdusă în vectorul soluție.
- S-au completat toate nivelele stivei și deci soluția este 1 2 3.



## pas 4:

- Pentru nivelul 3 al stivei nu mai există o altă valoare din mulțimea  $A_3$  ce poate fi adăugată.
- Din acest motiv, se revine la nivelul 2 al stivei (înapoi) și se completează cu următoarea valoare din mulțimea  $A_2$  ce nu a fost utilizată, adică 3.
- Se trece la completarea următorului nivel al stivei.



## 10.2. Implementarea metodei backtracking

- Proceddeul se repetă până când nu se mai poate introduce o valoare pe nivelul 1 al stivei.
- În acel moment s-au generat toate soluțiile problemei.

Implementarea programului pentru generarea permutarilor:

```
#include<iostream>
using namespace std;
int st[20],i,k,p,q,m,n,as,ev;
void init(int k,int st[ ])
{
    st[k]=0;
}
int sucesor(int k,int st[ ])
{
    if ( st[k]<n )
        {
            st[k]=st[k]+1;
            as=1;
        }
    else as=0;
    return as;
}
```

## 10.2. Implementarea metodei backtracking

```
int valid(int k,int st[ ])  
{  
    ev=1;  
    for (i=1;i<=k-1;i++)  
        if (st[i]==st[k]) ev=0;  
    return ev;  
}
```

```
int solutie(int k)
{
    if ( k==n ) return 1;
        else return 0;
}
void tipar(void)
{
    for(i=1;i<=n;i++)
        cout<<" "<<st[i];
    cout<<"\n";
}
```

```
int main(void)
{
    cin>>n;
    cout<<"Permutarile multimii de numere de la 1 la "<<n<<" sunt:\n";
    k=1; init(k,st);
    while (k>0)
    {
        do{
            as=succesor(k,st);
            if (as) ev=valid(k,st);
        }while (!( !as || (as && ev)));
        if (as)
            if (solutie(k)) tipar();
            else
            {
                k++;
                init(k,st);
            }
        else
            k--;
    }
}
```

```

1  #include <iostream>
2  using namespace std;
3
4  int st[20],i,k,p,q,m,n,as,ev;
5
6  void init(int k,int st[ ])
7  {
8      st[k]=0;
9  }
10 int sucesor(int k,int st[ ])
11 {
12     if ( st[k]<n )
13     {
14         st[k]=st[k]+1;
15         as=1;
16     }
17     else as=0;
18     return as;
19 }
20 int valid(int k,int st[ ])
21 {
22     ev=1;
23     for (i=1;i<=k-1;i++)
24         if (st[i]==st[k]) ev=0;
25     return ev;
26 }
27 int solutie(int k)
28 {
29     if ( k==n ) return 1;
30     else return 0;
31 }
32 void tipar(void)
33 {
34     for(i=1;i<=n;i++)
35         cout<<" "<<st[i];
36     cout<<"\n";
37 }

```

```

38
39 int main(void)
40 {
41     cin>>n;
42     cout<<"Permutarile multimii de numere de la 1 la "<<n<<" sunt:\n";
43     k=1; init(k,st);
44     while (k>0)
45     {
46         do{
47             as=sucesor(k,st);
48             if (as) ev=valid(k,st);
49         }while (!( !as || (as && ev)));
50         if (as)
51             if (solutie(k)) tipar();
52         else
53         {
54             k++;
55             init(k,st);
56         }
57         else
58             k--;
59     }
60     return 0;
61 }
62

```

Execute Mode, Version, Inputs & Arguments

GCC 11.1.0  Interactive Stdin Inputs

4

CommandLine Arguments

Execute

## Result

CPU Time: 0.00 sec(s), Memory: 3432 kilobyte(s)

```
Permutarile multimii de numere de la 1 la 4 sunt:
```

```
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
```

## Problema 2:

Fiind dată o tablă de șah, de dimensiune  $n \times n$ , se cer toate soluțiile de aranjare a  $n$  regine, astfel încât să nu se afle două regine pe aceeași linie, coloană sau diagonală (reginele să nu se atace reciproc).

Afișarea să se facă astfel:

*Exemplu:* Pentru  $n=4$  se va afișa:

### Soluția 1:

```
* R * *
* * * R
R * * *
* * R *
```

### Soluția 2:

```
* * R *
R * * *
* * * R
* R * *
```

```
#include <iostream>
# include <math.h>
using namespace std;
int st[100];
  int k,n,i,a;
  int ev,as;

void init(void)
{  st[k]=0;    }

int sucesor(void)
{
  if(st[k]<n)
  {
    st[k]=st[k]+1;
    as=1;
  }
  else as=0;
  return as;
}
```

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

```
int solutie(void)
{
    if(k==n) return 1;
    else return 0;
}
```

```
void tipar(void)
{
    int i,j;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
            if(st[i]==j) cout<<" R ";
            else cout<<" * ";
        cout<<endl;
    }
    cout<<"\n";
}
```

```
int main(void)
{
    cin>>n;
    cout<<"Variantele de asezare a "<<n<<" regine pe o tabla de
        "<<n<<"x"<<n<<" sunt:\n";
    k=1; init(k,st);
    while (k>0)
    {
        do{
            as=succesor(k,st);
            if (as) ev=valid(k,st);
        }while (!( !as || (as && ev)));
        if (as)
            if (solutie(k)) tipar();
            else
            {
                k++;
                init(k,st);
            }
        else
            k--;
    }
}
```

```

1 #include <iostream>
2 using namespace std;
3
4 int st[20],i,k,p,q,m,n,as,ev;
5
6 void init(int k,int st[ ])
7 {
8     st[k]=0;
9 }
10 int sucesor(int k,int st[ ])
11 {
12     if ( st[k]<n )
13     {
14         st[k]=st[k]+1;
15         as=1;
16     }
17     else as=0;
18     return as;
19 }
20 int valid(int k,int st[ ])
21 {
22     ev=1;
23     for (i=1;i<=k-1;i++)
24         if (st[i]==st[k]) ev=0;
25     for(i=0;i<=k-1;i++)
26         if(abs(k-i)==abs(st[k]-st[i])) ev=0;
27
28     return ev;
29 }
30 int solutie(int k)
31 {
32     if ( k==n ) return 1;
33     else return 0;
34 }
35 void tipar(void)
36 {
37     int i,j;
38     for(i=1;i<=n;i++)
39     {
40         for(j=1;j<=n;j++)
41             if(st[i]==j) cout<<" R ";
42             else cout<<" * ";
43         cout<<endl;
44     }
45     cout<<"\n";
46 }


```

```

47
48
49 int main(void)
50 {
51     cin>>n;
52     cout<<"Variantele de asezare a "<<n<<" regine pe o tabla de "<<n<<"x"<<n<<" sunt:\n";
53     k=1; init(k,st);
54     while (k>0)
55     {
56         do{
57             as=sucesor(k,st);
58             if (as) ev=valid(k,st);
59             }while (!( !as || (as && ev)));
60         if (as)
61             if (solutie(k)) tipar();
62             else
63             {
64                 k++;
65                 init(k,st);
66             }
67         else
68             k--;
69     }
70     return 0;
71 }
72

```

Execute Mode, Version, Inputs & Arguments





GCC 11.1.0 

Interactive

Stdin Inputs

4

CommandLine Arguments

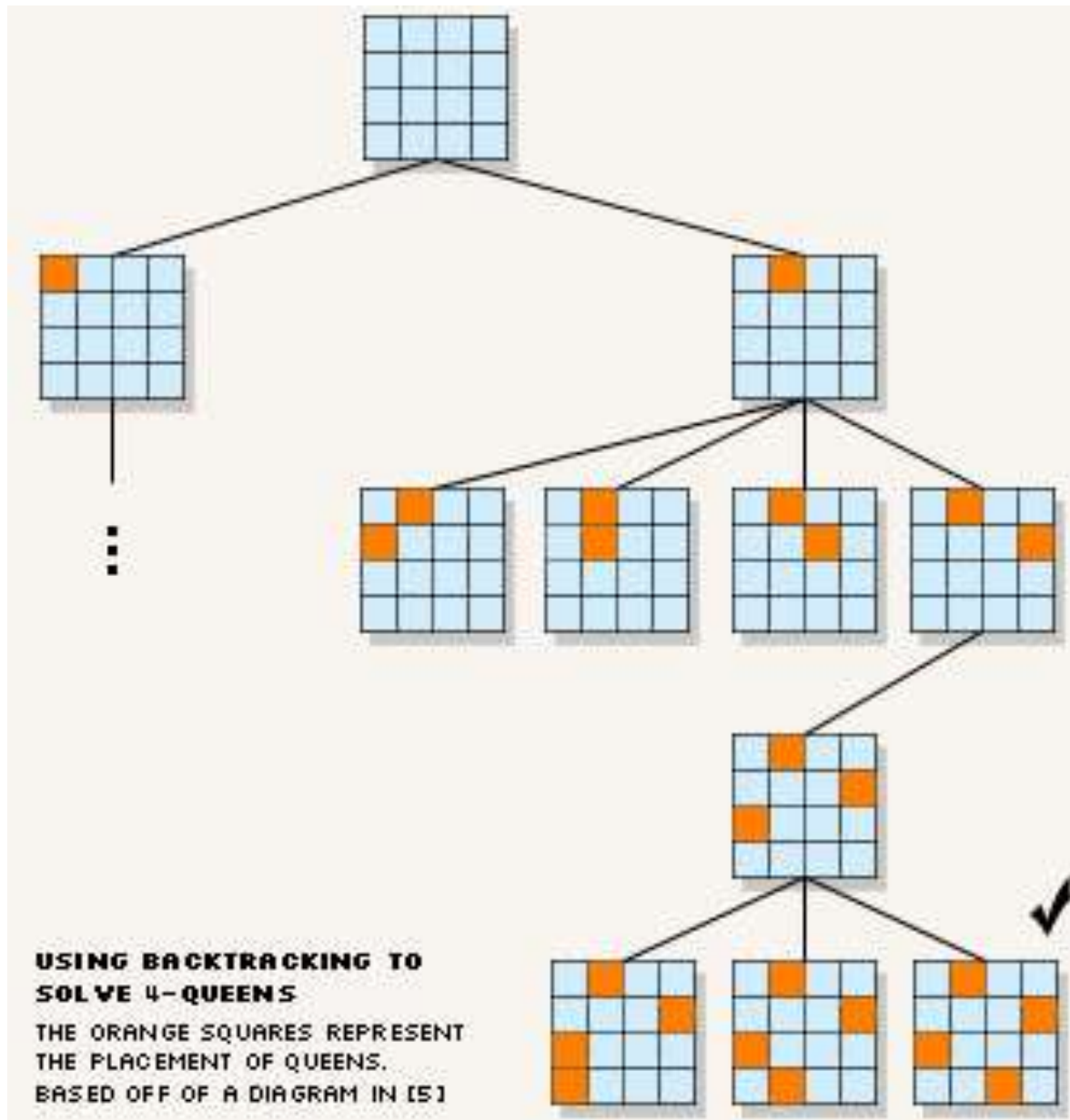
 Execute   

### Result

CPU Time: 0.00 sec(s), Memory: 3496 kilobyte(s)

```
Variantele de asezare a 4 regine pe o tabla de 4x4 sunt:
```

```
* R * *  
* * * R  
R * * *  
* * R *  
  
* * R *  
R * * *  
* * * R  
* R * *
```



## Problema 3:

Sa se genereze toate permutarile multimii  $\{1,2,\dots,n\}$  cu proprietatea ca *diferenta in modul dintre oricare doua numere consecutive este cel putin egala cu valoarea  $v$*  citita de la tastatura.

Exemplu:

Pentru  $n=4$  si  $v=2$  singurele solutii sunt:  
 $(2,4,1,3)$  si  $(3,1,4,2)$

```
# include <iostream>
# include <math.h>
using namespace std;
int st[100];
int k,n,i,v;
int ev,as;

void init(void)
{ st[k]=0; }

int sucesor(void)
{
    if(st[k]<n)
    {
        st[k]=st[k]+1;
        as=1;
    }
    else as=0;
    return as;
}
```

```
int valid(void)
{
    int i;
    ev=1;
    for(i=0;i<k;i++)
        if(st[k]==st[i]) ev=0;
    if(k>1)
        if(abs(st[k]-st[k-1])<v) ev=0;
    return ev;
}
int solutie(void)
{
    if(k==n) return 1;
    else return 0;
}
void tipar(void)
{
    int i;
    cout<<"(";
    for(i=1;i<=n;i++)
        if(i<n) cout<<st[i]<<" ";
        else cout<<st[i];
    cout<<")"<<endl;
}
```

```
int main()
{
    cin>>n;
    cin>>v;
    k=1; init();
    while(k>0)
    {
        do
        {
            as=succesor();
            if(as==1) ev=valid();
        }while( !(!as) || (as && ev)) );
        if(as==1)
            if(solutie()==1) tipar();
            else
            {
                k++;    init();
            }
        else k--;
    }
}
```

```

1  #include <iostream>
2  using namespace std;
3
4  int st[100],k,n,i,v,ev,as;
5
6  void init(int k,int st[ ])
7  {
8      st[k]=0;
9  }
10 int sucesor(int k,int st[ ])
11 {
12     if ( st[k]<n )
13     {
14         st[k]=st[k]+1;
15         as=1;
16     }
17     else as=0;
18     return as;
19 }
20 int valid(int k,int st[ ])
21 {
22     ev=1;
23     for(i=0;i<k;i++)
24         if(st[k]==st[i]) ev=0;
25     if(k>1)
26         if(abs(st[k]-st[k-1])<v) ev=0;
27
28     return ev;
29 }
30 int solutie(int k)
31 {
32     if ( k==n ) return 1;
33     else return 0;
34 }
35 void tipar(void)
36 {
37     int i;
38     cout<<" ";
39     for(i=1;i<=n;i++)
40         if(i<n) cout<<st[i]<<" ";
41         else cout<<st[i];
42     cout<<" "<<endl;
43 }

```

```

44
45 int main(void)
46 {
47     cin>>n;
48     cin>>v;
49     cout<<"Permutarile multimii de numere de la 1 la "<<n<<" cu diferenta "<<v<<" sunt:\n";
50     k=1; init(k,st);
51     while (k>0)
52     {
53         do{
54             as=sucesor(k,st);
55             if (as) ev=valid(k,st);
56         }while (!( !as || (as && ev)));
57         if (as)
58             if (solutie(k)) tipar();
59         else
60         {
61             k++;
62             init(k,st);
63         }
64         else
65             k--;
66     }
67     return 0;
68 }
69

```

Execute Mode, Version, Inputs & Arguments

GCC11.1.0 

Interactive

Stdin Inputs

4  
2

CommandLine Arguments

 Execute   

### Result

CPU Time: 0.00 sec(s), Memory: 3496 kilobyte(s)

```
Permutarile multimii de numere de la 1 la 4 cu diferenta 2 sunt:  
(2,4,1,3)  
(3,1,4,2)
```

## Conținutul cursului

**10.1. Prezentarea generala a metodei**

**10.2. Implementarea metodei backtracking**

**10.3. Probleme propuse spre rezolvare**

## Problema 1

Sa se genereze toate aranjamentele multimii  $\{1, 2, \dots, n\}$ , cu cate  $p$  elemente.

Exemplu:

Pentru  $n = 5$  si  $p = 3$  se vor afisa multimile:

(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1);

(1,2,4), (1,4,2), (2,1,4), (2,4,1), (4,1,2), (4,2,1);

(1,2,5), (1,5,2), (2,1,5), (2,5,1), (5,1,2), (5,2,1);

(1,3,4), (1,4,3), (3,1,4), (3,4,1), (4,1,3), (4,3,1);

(1,3,5), (1,5,3), (3,1,5), (3,5,1), (5,1,3), (5,3,1);

(1,4,5), (1,5,4), (4,1,5), (4,5,1), (5,1,4), (5,4,1),

s.a.m.d.

## Problema 2

Sa se genereze toate combinarile multimii  $\{1, 2, \dots, n\}$ , cu cate  $p$  elemente.

Exemplu:

Pentru  $n = 5$  si  $p = 3$  se vor afisa multimile:  
 $(1,2,3)$ ,  $(1,2,4)$ ,  $(1,2,5)$ ,  $(1,3,4)$ ,  $(1,3,5)$ ,  $(1,4,5)$ ,  
 $(2,3,4)$ ,  $(2,3,5)$ ,  $(3,4,5)$ .

## Problema 3

Se considera un număr  $n$  natural nenul și se cere să se afișeze toate descompunerile numărului  $n$  în suma de numere naturale.

Exemplu:

Pentru  $n=5$  se vor afișa valorile:

$1+1+1+1+1$

$1+1+1+2$ ;  $1+1+2+1$ ;  $1+2+1+1$ ;  $2+1+1+1$

$1+2+2$ ;  $2+1+2$ ;  $2+2+1$

$1+4$ ;  $4+1$

$2+3$ ;  $3+2$

## Problema 4

Să se afișeze toate posibilitățile de așezare a  $n$  ture pe o tablă de șah de  $n \times n$  astfel încât să nu se atace una pe celalaltă.

Exemplu: pentru  $n=3$

**Soluția 1 :**

\* T \*

\* \* T

T\* \*

s.a.m.d.

**Soluția 2 :**

\* T \*

T \* \*

\* \* T

## Problema 5

Să se descompună un număr natural  $N$ , în toate modurile posibile, ca sumă de  $P$  numere naturale ( $P \leq N$ ).

Exemplu:

Pentru  $n=5$  și  $p=3$  se obțin soluțiile:

1+1+3;      1+3+1;      3+1+1;  
1+2+2;      2+1+2;      2+2+1;

## Problema 6

Avem la dispoziție 6 culori: alb, galben, roșu, verde, albastru și negru. Să se precizeze toate drapelele tricolore care se pot proiecta, știind că trebuie respectate următoarele reguli:  
orice drapel are culoarea din mijloc galben sau verde;  
cele trei culori de pe drapel sunt distincte.

Exemple: "alb galben roșu", "roșu verde galben"

**Indicatii:** Folosim o stivă cu 3 nivele, reprezentând cele trei culori de pe drapel și codificăm culorile prin numere:

- 1 – alb
- 2 – galben
- 3 – roșu
- 4 – verde
- 5 – albastru
- 6 - negru

# Întrebări?