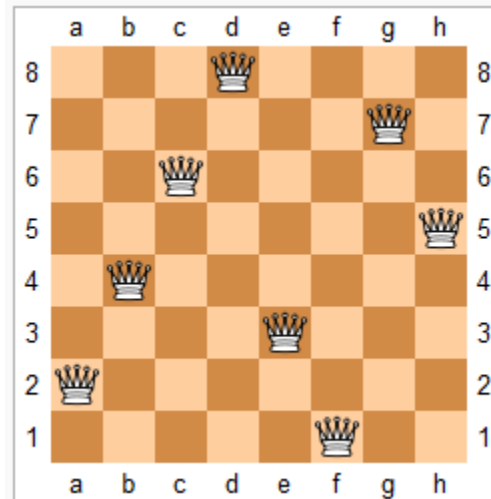


# Metoda de programare



## BACKTRACKING



1. Competențe . . . . .	3
2. Descrierea generală a metodei . . . . .	4
3. Generarea elementelor combinatoriale. . . . .	7
4. Probleme . . . . .	24
5. Aplicații . . . . .	28
6. Bibliografie și webografie . . . . .	29



## Competențe generale

- *elaborarea algoritmilor de rezolvare a problemelor*
- *implementarea algoritmilor într-un limbaj de programare*

## Competențe specifice

- *analiza problemei în scopul identificării metodei de programare adecvate pentru rezolvarea problemei*
- *aplicarea creativă a metodelor de programare pentru rezolvarea unor probleme intradisciplinare sau interdisciplinare, sau a unor probleme cu aplicabilitate practică*
- *analiza comparativă a eficienței diferitelor metode de rezolvare a aceleiași probleme și alegerea unui algoritm eficient de rezolvare a unei probleme*
- *elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării*
- *utilizarea tehnicilor moderne în implementarea aplicațiilor*



## 2. Descrierea generală a metodei

Rezolvarea unei probleme poate conduce la un obținerea unui număr foarte mare de soluții posibile. Nu întotdeauna ne interesează toate soluțiile, ci numai o parte dintre ele, care îndeplinesc anumite condiții. În astfel de cazuri este indicată folosirea metodei Backtracking (BKT).

Soluția unei probleme rezolvate cu metoda BKT se poate reprezenta sub forma unui vector:  $\mathbf{S} = (s_1, s_2, s_3, \dots, s_n)$ .

Fiecare componentă  $s_i$  a vectorului poate lua valori într-o anumită mulțime  $A_i$  unde  $i=1, 2, 3, \dots, n$ . Produsul cartezian  $A_1 \times A_2 \times A_3 \times \dots \times A_n$  se numește *spațiul soluțiilor posibile*.



## Descrierea generală a metodei

Problemele care se rezolvă prin metoda BKT generează soluții care îndeplinesc anumite condiții specifice problemei, denumite *condiții de validare (condiții interne)*.

Soluțiile posibile care îndeplinesc condițiile de validare sunt numite *soluții rezultat (soluții finale)*.

Unele probleme impun obținerea unei singure soluții, numită *soluția optimă*.



Principiul metodei BKT:

- soluția se construiește pas cu pas;
- 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.



### 3. Generarea elementelor combinatoriale

O serie de probleme rezolvate prin metoda BKT o reprezintă cea a problemelor de combinatorică:

- permutări;
- combinări;
- aranjamente;
- submulțimi;
- produs cartezian.



## a. Generarea permutărilor

### Enunț:

*Să se genereze toate permutările mulțimii  $\{1, 2, \dots, n\}$ .*

**Numărul de soluții:**  $n!$

Permutările de  $n$  elemente sunt mulțimi ordonate ce conțin elementele mulțimii  $\{1, 2, \dots, n\}$  în care fiecare element apare o singură dată. Orice permutare este alcătuită din toate elementele mulțimii, elementele fiind distincte.



## Exemplu:

$$n=3$$

$$A=\{1, 2, 3\}$$

Numărul de soluții:  $3!=6$

Soluțiile:  $\{1, 2, 3\}$   $\{1, 3, 2\}$   
 $\{2, 1, 3\}$   $\{2, 3, 1\}$   
 $\{3, 1, 2\}$   $\{3, 2, 1\}$

## Exerciții:

1. Scrieți a șasea soluție a permutării mulțimii  $\{1, 2, 3, 4\}$ .
2. Scrieți a câta soluție este permutarea  $\{2, 8, 6, 4\}$  obținută din mulțimea  $\{2, 4, 6, 8\}$ .
3. Scrieți toate soluțiile permutării mulțimii  $\{1, 2, 3, 4, 5\}$ , soluții care încep cu cifra 4 și se termină cu cifra 1.



## Implementare în C++:

```

1  #include<iostream>
2  using namespace std;
3  int st[10],k,n;
4
5  int bun()
6  {
7      int i;
8      for(i=1;i<k;i++)
9          if(st[k]==st[i])
10             return 0;
11     return 1;
12 }
13
14 int solutie()
15 {
16     if(n==k)
17         return 1;
18     else
19         return 0;
20 }
21
22 void afisare()
23 {
24     int i;
25     for(i=1;i<=n;i++)
26         cout<<st[i]<<' ';
27     cout<<endl;
28 }
29

```

```

30 void bkt()
31 {
32     k=1;
33     while(k>0)
34         if(st[k]<n)
35             {
36                 st[k]=st[k]+1;
37                 if(bun())
38                     if(solutie())
39                         afisare();
40                     else
41                         k++;
42             }
43         else
44             {
45                 st[k]=0;
46                 k--;
47             }
48     }
49
50 int main()
51 {
52     cout<<"n=";
53     cin>>n;
54     bkt();
55     return 0;
56 }
57

```



## b. Generarea combinărilor

### Enunț:

Să se genereze toate combinările de  $p$  elemente ale mulțimii  $\{1, 2, \dots, n\}$ .

Numărul de soluții:  $\frac{n!}{p!(n-p)!}$

Combinările de  $n$  elemente luate câte  $p$  reprezintă o modalitate de a alege  $p$  elemente din cele  $n$  date, fără a conta ordinea elementelor. Două mulțimi care au aceleași elemente, dar așezate în altă ordine se consideră egale.



## Exemplu:

$$n=3, \quad p=2$$

$$A=\{1, 2, 3\}$$

$$\text{Numărul de soluții: } \frac{n!}{p!(n-p)!} = \frac{3!}{2!(3-2)!} = \frac{6}{2} = 3$$

Soluțiile:  $\{1, 2\}$

$\{1, 3\}$

$\{2, 3\}$

## Exerciții:

1. Scrieți toate soluțiile combinărilor mulțimii  $\{1, 2, 3, 4\}$  luate câte 3.
2. Scrieți toate soluțiile combinărilor mulțimii  $\{2, 4, 6, 8\}$  luate câte 2.
3. Scrieți toate soluțiile combinărilor mulțimii  $\{1, 2, 3, 4, 5\}$  luate câte 3, soluții care încep cu cifra 1 și se termină cu 5.



## Implementare în C++:

```

1  #include<iostream>
2  using namespace std;
3  int st[10],k,n,p;
4
5  int bun()
6  {
7      if(k>1)
8          if(st[k]<=st[k-1])
9              return 0;
10     return 1;
11 }
12
13 int solutie()
14 {
15     if(p==k)
16         return 1;
17     else
18         return 0;
19 }
20
21 void afisare()
22 {
23     int i;
24     for(i=1;i<=p;i++)
25         cout<<st[i]<<' ';
26     cout<<endl;
27 }
28

```

```

29 void bkt()
30 {
31     k=1;
32     while(k>0)
33         if(st[k]<n)
34             {
35                 st[k]=st[k]+1;
36                 if(bun())
37                     if(solutie())
38                         afisare();
39                     else
40                         k++;
41             }
42         else
43             {
44                 st[k]=0;
45                 k--;
46             }
47     }
48
49 int main()
50 {
51     cout<<"n=";
52     cin>>n;
53     cout<<"p=";
54     cin>>p;
55     bkt();
56     return 0;
57 }
58

```



## c. Generarea aranjamentelor

### Enunț:

Să se genereze toate aranjamentele de  $p$  elemente ale mulțimii  $\{1, 2, \dots, n\}$ .

Numărul de soluții:  $\frac{n!}{(n-p)!}$

Aranjamentele de  $n$  elemente luate câte  $p$  reprezintă o modalitate de a selecta și aranja  $p$  elemente din cele  $n$  date. Două mulțimi care au aceleași elemente, dar așezate în altă ordine se consideră distincte.



## Exemplu:

$$n=3, \quad p=2$$

$$A=\{1, 2, 3\}$$

$$\text{Numărul de soluții: } \frac{n!}{(n-p)!} = \frac{3!}{(3-2)!} = \frac{6}{1} = 6$$

Soluțiile:  $\{1, 2\}$   $\{1, 3\}$   
 $\{2, 1\}$   $\{2, 3\}$   
 $\{3, 1\}$   $\{3, 2\}$

## Exerciții:

1. Scrieți toate soluțiile aranjamentelor mulțimii  $\{1, 2, 3, 4\}$  luate câte 2.
2. Scrieți ultimele 3 soluții ale aranjamentelor mulțimii  $\{2, 4, 6, 8\}$  luate câte 2.
3. Scrieți toate soluțiile aranjamentelor mulțimii  $\{1, 2, 3, 4, 5\}$  luate câte 3, soluții care încep cu cifra 4.



## Implementare în C++:

```

1  #include<iostream>
2  using namespace std;
3  int st[10],k,n,p;
4
5  int bun()
6  {
7      int i;
8      for(i=1;i<k;i++)
9          if(st[k]==st[i])
10             return 0;
11     return 1;
12 }
13
14 int solutie()
15 {
16     if(p==k)
17         return 1;
18     else
19         return 0;
20 }
21
22 void afisare()
23 {
24     int i;
25     for(i=1;i<=p;i++)
26         cout<<st[i]<<' ';
27     cout<<endl;
28 }
29

```

```

30 void bkt ()
31 {
32     k=1;
33     while(k>0)
34         if(st[k]<n)
35             {
36                 st[k]=st[k]+1;
37                 if(bun())
38                     if(solutie())
39                         afisare();
40                     else
41                         k++;
42             }
43         else
44             {
45                 st[k]=0;
46                 k--;
47             }
48     }
49
50 int main()
51 {
52     cout<<"n=";
53     cin>>n;
54     cout<<"p=";
55     cin>>p;
56     bkt();
57     return 0;
58 }
59

```



## d. Generarea submulțimilor

### Enunț:

*Să se genereze toate submulțimile mulțimii  $\{1, 2, \dots, n\}$ .*

**Numărul de soluții:**  $2^n - 1$

Submulțimile reprezintă o modalitate de a forma grupuri cu cel puțin un element și cel mult  $n$  elemente din cele  $n$  elemente ale unei mulțimi. Două submulțimi care au aceleași elemente, dar așezate în altă ordine se consideră egale.



## Exemplu:

$$n=3$$

$$A=\{1, 2, 3\}$$

$$\text{Numărul de soluții: } 2^3 - 1 = 7$$

Soluțiile:  $\{1\}$ ;  $\{1, 2\}$ ;  $\{1, 2, 3\}$ ;  $\{1, 3\}$ ;  $\{2\}$ ;  $\{2, 3\}$ ;  $\{3\}$

## Exerciții:

1. Scrieți toate submulțimile mulțimii  $\{1, 2, 3, 4\}$ .
2. Scrieți toate submulțimile mulțimii  $\{2, 4, 6, 8\}$ , submulțimi care conțin elementul 2.
3. Scrieți numărul submulțimilor de două elemente ale mulțimii  $\{1, 2, 3, 4, 5\}$ .



## Implementare în C++:

```

1  #include<iostream>
2  using namespace std;
3  int st[10],k,n;
4
5  int bun()
6  {
7      if(k>1)
8          if(st[k]<=st[k-1])
9              return 0;
10     return 1;
11 }
12
13 void afisare()
14 {
15     int i;
16     for(i=1;i<=k;i++)
17         cout<<st[i]<<' ';
18     cout<<endl;
19 }
20

```

```

21 void bkt()
22 {
23     k=1;
24     while(k>0)
25         if(st[k]<n)
26         {
27             st[k]=st[k]+1;
28             if(bun())
29             {
30                 afisare();
31                 k++;
32             }
33         }
34         else
35         {
36             st[k]=0;
37             k--;
38         }
39     }
40
41 int main()
42 {
43     cout<<"n=";
44     cin>>n;
45     bkt();
46     return 0;
47 }
48

```



## e. Generarea produsului cartezian

### Enunț:

Se dau  $n$  mulțimi  $A_1, A_2, \dots, A_n$ . Să se genereze toate elementele produsului cartezian  $A_1 \times A_2 \times \dots \times A_n$ .

### Numărul de soluții:

$$n \cdot n \cdot \dots \cdot n$$

Produsul cartezian a  $n$  mulțimi reprezintă o mulțime, numită și mulțimea produs, formată din ansamblul tuturor grupurilor de  $n$  elemente în care prima componentă aparține mulțimii  $A_1$ , a doua componentă aparține mulțimii  $A_2$ , ..., a  $n$ -a componentă aparține mulțimii  $A_n$ .



## Exemplu:

$$n=3$$

$$A_1=\{1, 2, 3\} \quad , \quad A_2=\{1, 2, 3\} \quad , \quad A_3=\{1, 2, 3\}$$

Numărul de soluții:  $n^n=3^3=27$

Soluțiile:  $\{1, 1, 1\}; \{1, 1, 2\}; \{1, 1, 3\}; \{1, 2, 1\}; \{1, 2, 2\}; \{1, 2, 3\};$   
 $\{1, 3, 1\}; \{1, 3, 2\}; \{1, 3, 3\};$   
 $\{2, 1, 1\}; \{2, 1, 2\}; \{2, 1, 3\}; \{2, 2, 1\}; \{2, 2, 2\}; \{2, 2, 3\};$   
 $\{2, 3, 1\} \quad \{2, 3, 2\}; \{2, 3, 3\};$   
 $\{3, 1, 1\}; \{3, 1, 2\}; \{3, 1, 3\}; \{3, 2, 1\}; \{3, 2, 2\}; \{3, 2, 3\};$   
 $\{3, 3, 1\}; \{3, 3, 2\}; \{3, 3, 3\}$

## Exerciții:

1. Scrieți numărul de soluții care încep cu 4, ale produsului cartezian a 4 mulțimi de forma  $\{1, 2, 3, 4\}$ .
2. Scrieți toate soluțiile produsului cartezian a mulțimilor  $\{2, 4, 6, 8\}$  și  $\{1, 3\}$ .
3. Scrieți ultima soluție a produsului cartezian a 3 mulțimi  $\{1, 3, 5\}$ ,  $\{2, 4, 6\}$  și  $\{7, 8, 9\}$ .



## Implementare în C++:

```
1  #include<iostream>
2  using namespace std;
3  int st[10],k,n;
4
5  int solutie()
6  {
7      if(k==n)
8          return 1;
9      else
10         return 0;
11 }
12
13 void afisare()
14 {
15     int i;
16     for(i=1;i<=n;i++)
17         cout<<st[i]<<' ';
18     cout<<endl;
19 }
20
```

```
21 void bkt()
22 {
23     k=1;
24     while(k>0)
25         if(st[k]<n)
26         {
27             st[k]=st[k]+1;
28             if(solutie())
29                 afisare();
30             else
31                 k++;
32         }
33         else
34         {
35             st[k]=0;
36             k--;
37         }
38     }
39
40 int main()
41 {
42     cout<<"n=";
43     cin>>n;
44     bkt();
45     return 0;
46 }
47
```



## BKT recursiv

```
void bkt(int k)
{
    int i;
    for(i=1;i<=n;i++)
    {
        st[k]=i;
        if(bun(k))
            if(solutie(k))
                afisare();
            else
                bkt(k+1);
    }
}
```

## Temă

Implementați în limbajul C++ problemele de combinatorică folosind algoritmi recursivi.



### Problema damelor

#### Enunț:

*Să se găsească toate modalitățile de a aranja  $n$  dame pe o tablă de șah de dimensiuni  $n \times n$ , astfel încât ele să nu se atace una pe cealaltă. Două dame se atacă dacă ele se află pe aceeași linie, pe aceeași coloană sau pe aceeași diagonală.*



## Exemplu:

Pentru  $n=4$  se obțin două soluții:

	D		
			D
D			
		D	

		D	
D			
			D
	D		

```
C:\MinGWStudio\Templates\bkt_dame\Debug\bkt_dame.exe
n=4
Solutie:
  linia 1 coloana 2
  linia 2 coloana 4
  linia 3 coloana 1
  linia 4 coloana 3
Solutie:
  linia 1 coloana 3
  linia 2 coloana 1
  linia 3 coloana 4
  linia 4 coloana 2

Terminated with return code 0
Press any key to continue ...
```

## Soluție:

- pe linia  $k$  nu trebuie să mai fie o altă damă: această condiție este satisfăcută datorită modului de reprezentare a soluției;
- pe coloana corespunzătoare, care este  $S[k]$  nu trebuie să mai fie o altă damă:  $S[k] \neq S[i]$ , pentru  $i \in [1, k-1]$ ;
- oricare dintre damele așezate nu trebuie să se afle pe o aceeași diagonală cu dama de pe linia  $k$ : pentru  $i \in [1, k-1]$  damele aflate pe pozițiile  $(i, S[i])$  respectiv  $(k, S[k])$  nu sunt pe aceeași diagonală dacă  $|i-k| \neq |S[i]-S[k]|$ ;



## Implementare în C++:

```

1  #include<iostream>
2  using namespace std;
3  int st[10],n,k;
4
5  int bun()
6  {
7      int i;
8      for(i=1;i<k;i++)
9          if( (st[k]==st[i]) || (abs(i-k)==abs(st[i]-st[k])) )
10             return 0;
11     return 1;
12 }
13
14 int solutie()
15 {
16     return k==n;
17 }
18
19 void afisare()
20 {
21     int i;
22     cout<<"Solutie:"<<endl;
23     for(i=1;i<=n;i++)
24         cout<<"    linia "<<i<<" coloana "<<st[i]<<endl;
25 }
26

```

```

27 void bkt ()
28 {
29     k=1;
30     while(k>0)
31         if(st[k]<n)
32             {
33                 st[k]++;
34                 if(bun())
35                     if(solutie())
36                         afisare();
37                     else
38                         k++;
39             }
40         else
41             {
42                 st[k]=0;
43                 k--;
44             }
45     }
46
47 int main()
48 {
49     cout<<"n=";
50     cin>>n;
51     bkt ();
52     return 0;
53 }
54

```



### Fișă de lucru

- Întrebări metoda de programare *Backtracking*
- Aplicații metoda de programare *Backtracking*



## 6. Bibliografie și webografie

1. Miloșescu M., *Informatică. Manual pentru clasa a XI*, Editura Didactică și Pedagogică, București, 2006
2. Țoca L., *Informatică. Manual pentru clasa a X*, Editura Niculescu, București, 2001
3. Popescu C., *Culegere de probleme de informatică*, Editura Donaris-Info, Sibiu, 2002
4. Ministerul Educației, Cercetării și Tineretului, Centrul Național pentru Curriculum și Evaluare în Învățământul Preuniversitar, *Proba scrisă la informatică. Examenul de bacalaureat – Variante (1-100)*, București 2008
5. <http://ro.wikipedia.org/wiki/Backtracking>
6. <http://en.wikipedia.org/wiki/Backtracking>

