Capitolul 10. Generarea elementelor combinatoriale

10.1 Noţiuni teoretice

În practică se ajunge uneori la problema de a alege, dintr-o mulţime oarecare de obiecte, submulţimi de elemente caracterizate prin anumite proprietăţi, de a aşeza elementele uneia sau ale mai multor submulţimi într-o anumită ordine etc. Poate apărea, de asemenea, problema determinării numărului tuturor submulţimilor unei mulţimi, constituite după anumite reguli, sau a numărului tuturor funcţiilor definite pe o mulţime formată din *n* elemente, cu valori într-o mulţime cu *m* elemente etc.

Domeniul matematicii care studiază astfel de probleme se numește **combinatorică** și are importanță pentru teoria probabilităților, logica matematică, teoria numerelor, precum și pentru alte ramuri ale științei și tehnicii.

Combinatorica este deci o ramură a matematicii care studiază mulțimi de obiecte și modalitățile de a le combina. Analiza combinatorică include:

- numărarea unor obiecte sau anumite clase de echivalenţă de obiecte care se identifică pe baza unei anumite relații (*combinatorica enumerativă*);
- verificarea dacă un criteriu poate fi satisfăcut şi construirea sau analizarea obiectelor ce satisfac criteriul respectiv (*modele combinatoriale*);
- găsirea obiectului „celui mai mare”, „celui mai mic” sau „optimal” (*combinatorica extremală* sau *optimizare combinatorială*);
- găsirea unor structuri algebrice pe care o formează anumite obiecte (*combinatorica algebrică*).

Problemele de combinatorică se pot rezolva folosind metoda backtracking. Algoritmii de tip backtracking prezentaţi în capitolul anterior vor fi folosiţi pentru a genera permutări, aranjamente etc.

Reguli generale ale combinatoricii

- **Regula sumei**: Dacă un anumit obiect A poate fi ales în m moduri, iar alt obiect B poate fi ales în n moduri, atunci **alegerea lui A sau B** poate fi realizată în **(m + n) moduri**.
- **Regula produsului**: Dacă un anumit obiect A poate fi ales în m moduri și dacă după fiecare astfel de alegere, un obiect B se poate alege în n moduri, atunci **alegerea perechii (A;B)** în aceasta ordine poate fi realizata în **(m· n) moduri**.
- O mulţime împreună cu o ordine bine determinată de dispunere a elementelor sale este o **combinație** sau **o mulțime ordonată.** Astfel, dacă *A* este o mulţime finită având *n* elemente şi dacă fiecărui element al său i se asociază un număr de la 1 la *n* numit rangul elementului astfel încât la elemente diferite ale lui *A* să corespundă numere de ordine diferite, mulţimea *A* devine o mulţime ordonată. Prin urmare, orice mulţime finită poate deveni o mulţime ordonată, ordinea elementelor stabilindu-se prin numerotarea elementelor mulţimii respective. Mulţimea ordonată

obţinută se va nota cu $\{a_1, a_2, ..., a_n\}$, unde ordinea elementelor este dată de indici.

Algoritmii şi programele prezentate mai jos folosesc mulţimi de forma {1,2,…,n} pentru a simplifica lucrurile. De asemenea, este bine să numărăm câte soluţii generează fiecare program şi să verificăm aceste numere cu ajutorul formulelor cunoscute de la matematică.

10. 1.1 Generarea permutărilor

Fie *A* o mulţime finită cu *n* elemente. Această mulţime se poate ordona în mai multe moduri, obţinându-se mulţimi ordonate, diferite, care se deosebesc între ele numai prin ordinea elementelor. Fiecare din mulţimile ordonate care se formează cu cele *n* elemente ale mulţimii *A*, se numeşte permutare a acestei mulţimi sau o permutare de *n* elemente.

Enunț: ***(Generarea permutărilor)*** Dându-se ***n*** să se afişeze toate permutările mulțimii numerelor de la 1 la n în ordine lexicografică.

O permutare a mulțimii A este o succesiune de n elemente în care fiecare element apare exact o dată.

Generăm permutările lui A folosind vectorul x = {x_1, x_2, … x_n}, unde x_i este un element din A.

Condiții interne: $\forall\, i, j \in A,\ x_i \neq x_j,$ pentru $i \neq j$

Condiții de continuare: $x_k \neq x_i, \forall\, i \in \{1, 2, \ldots k-1\}$

Varianta nerecursivă

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n,k;
int init(int k)
{
 x[k]=0;
}
int succesor(int k)
{
 if (x[k]<n) return 1;
 else return 0;
}
int continuare(int k)
{
 int i;
 for (i=1;i<=k-1;i++)
  if (x[k]==x[i]) return 0;
 return 1;
}
int solutie(int k)
{
 if (k==n+1) return 1;
 else return 0;
}
int afisare()
```

Varianta recursivă

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n;

int folosit[NR];

void backr(int k) {
  int i;

  if ( k > n ) {
    for ( i = 1; i <= n; i++ )
      cout<<x[i]<<" ";
    cout<< "\n";
  }
  else
    for (i=1; i<=n; i++)
    {
        x[k]=i;

      if ( !folosit[x[k]] ) { // -
//daca elementul curent nu e deja
//folosit
        folosit[x[k]] = 1;    //
//marcheaza-l ca folosit
        backr(k + 1 );     //
```

```
{
 int i;
 for (i=1;i<=n;i++)
  cout<<x[i]<<" ";
 cout<<"\n";
}
int backtracking()
{
 int k,i;
 k=1; init(1);
 while (k!=0)
  if (solutie(k))
  {afisare(); k--;}
  else
  if (succesor(k))
  {
   x[k]++;
   if (continuare(k)) k++;
  }
  else {init(k); k--;}
}
int main()
{
 cout<<"n= "; cin>>n;
 backtracking();
}
```

```
//genereaza restul permutarii
        folosit[x[k]] = 0;    //
//demarcheaza elementul
      }
}
}

int main() {
  cout<<"Dati n:";
  cin>>n;
  backr(1);
  return 0;
}
```

Folosim pentru generare mulţimea {1,2,…,n}.

Generăm soluțiile element cu element, în vectorul x, pentru fiecare valoare pusă pe poziția k generăm toate soluțiile posibile, o valoare este pusă în x[k] numai dacă este validă, adică nu apare pe pozițiile anterioare, altfel continuarea secvenței nu are șanse să ducă la o soluție.

Pentru a verifica dacă x_k a fost plasat deja în vectorul x am procedat astfel:

- la varianta nerecursivă: am parcurs vectorul pentru a verifica dacă x_k apare sau nu printre elementele generate pe nivelele anterioare;
- la varianta recursivă: folosirea unui vector folosit cu n elemente – vectorul caracteristic ce are doar elemente de 0 și 1. Valoarea 1 va preciza faptul că elementul de pe poziţia corespunzătoare a fost plasat anterior în vectorul soluţie, iar valoarea 0 că nu.

Vrem să determinăm numărul de soluții. Avem:

1. O mulţime cu un singur element $A_1 = \{a_1\}$ poate fi ordonată într-un singur mod, deci $P_1 = 1$.

2. O mulţime cu două elemente $A_2 = \{a_1, a_2\}$ poate fi ordonată în două moduri, obținându-se două permutări: $\{a_1, a_2\}$ și $\{a_2, a_1\}$. Deci $P_2 = 2 = 1 \cdot 2$.

3. O mulţime cu trei elemente $A_3 = \{a_1, a_2, a_3\}$ poate fi ordonată în 6 moduri, permutările acestei mulţimi fiind următoarele: $\{a_1, a_2, a_3\}$, $\{a_1, a_3, a_2\}$, $\{a_2, a_1, a_3\}$, $\{a_2, a_3, a_1\}$, $\{a_3, a_1, a_2\}$, $\{a_3, a_2, a_1\}$. Deci, $P_3 = 6 = 1 \cdot 2 \cdot 3$.

Numărul permutărilor de ordin n se notează P_n și este 1*2*3*...*n = n!

10. 1.2 Generarea aranjamentelor

Considerăm *n* obiecte distincte $a_1,a_2,\ldots,a_n$. Dacă $1 \le m \le n$ un *aranjament* (*simplu*) *a celor n obiecte luate câte m* este o mulţime (grupare) ordonată $\{a_{i_1},...,a_{i_m}\}$ de *m* obiecte distincte din cele *n* date. Două aranjamente $\{a_{i_1},...,a_{i_m}\}$, $\{a_{j_1},...,a_{j_m}\}$ a celor *n* obiecte luate câte *m* se consideră *distincte* dacă ele diferă prin ordinea elementelor sau prin natura lor, adică există $k \le m$ astfel încât obiectele a_{i_k} şi a_{j_k} sunt distincte. În matematică, numărul de aranjamente (fără repetiție) a ***n*** $(n \in \mathbb{N})$ elemente luate câte ***m,*** $(m \in \mathbb{N}, m \le n)$ se notează cu A_n^m și se calculează cu formula:

$$A_n^m = n * (n-1) * \ldots\ldots * (n-m+1) = \frac{n!}{(n-m)!}$$

Ne propunem că găsim o formulă pentru calculul numărului A_n^m. Să observăm că două aranjamente de *n* elemente luate câte *m* diferă între ele atât prin *natura* elementelor, cât şi prin *ordinea* lor. De exemplu, dacă $A=\{a_1,a_2,a_3\}$, atunci din elementele sale se pot constitui:

1) 3 submulţimi având fiecare câte un element: $\{a_1\}$, $\{a_2\}$, $\{a_3\}$; acestea se deosebesc între ele numai prin natura elementelor, iar numărul lor este $A_3^1 = 3$;

2) 6 submulţimi ordonate având fiecare câte 2 elemente: $\{a_1,a_2\}$, $\{a_1,a_3\}$, $\{a_2,a_1\}$, $\{a_2,a_3\}$, $\{a_3,a_1\}$, $\{a_3,a_2\}$; acestea se deosebesc între ele fie prin natura elementelor, fie prin ordinea elementelor, iar numărul lor este $A_3^2 = 6 = 3 \cdot 2$;

3) 6 submulţimi ordonate având fiecare câte 3 elemente: $\{a_1,a_2,a_3\}$, $\{a_1,a_2,a_3\}$, $\{a_2,a_1,a_3\}$, $\{a_2,a_3,a_1\}$, $\{a_3,a_1,a_2\}$, $\{a_3,a_2,a_1\}$.

Acestea se deosebesc între ele numai prin ordinea lor, iar numărul lor este $A_3^3 = 6 = 3 \cdot 2 \cdot 1$. Mai observăm că în acest caz, în care numărul *m* al elementelor fiecărei submulţimi coincide cu numărul *n* al elementelor mulţimii *A*, aranjamentele de *n* elemente luate câte *n* coincid cu permutările de *n* elemente ale lui *A*.

Raţionamentul inductiv făcut pe exemplul anterior ne sugerează să considerăm că numărul A_n^m poate fi calculat cu formula următoare:

$$A_n^m = n * (n-1) * \ldots\ldots * (n-m+1) = \frac{n!}{(n-m)!}$$

Putem spune că permutările sunt un caz particular al aranjamentelor (m = n). Diferența dintre problema permutărilor este că din elemente posibile, la permutări le folosim pe toate, la aranjamente doar un număr de m.

Enunț: ***(Generarea aranjamentelor/funcţiilor injective[1]).*** Se citesc **n** şi **p** numere naturale. Se cere să se genereze toate submulţimile mulţimii **{1,2,...,n}** de **p** elemente. Două mulţimi cu aceleaşi elemente, la care ordinea acestora diferă, sunt considerate distincte.

Soluţie. Programul este asemănător cu cel de generare a permutărilor, doar condiţia pentru găsirea unei soluţii este **k=p+1**.

Corespunzător celor 2 variante de mai sus, putem scrie următoarele 2 programe:

Varianta nerecursivă

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n,p,k;
int init(int k)
{
 x[k]=0;
}
int succesor(int k)
{
 if (x[k]<n) return 1;
 else return 0;
}
int continuare(int k)
{
 int i;
 for (i=1;i<=k-1;i++)
  if (x[k]==x[i]) return 0;
 return 1;
}
int solutie(int k)
{
 if (k==p+1) return 1;
 else return 0;
}
int afisare()
{
 int i;
 for (i=1;i<=p;i++)
  cout<<x[i]<<" ";
 cout<<"\n";
}
int backtracking()
{
 int k,i;
 k=1; init(1);
 while (k!=0)
  if (solutie(k))
  {afisare(); k--;}
  else
```

Varianta recursivă

```
   #include <iostream>
   using namespace std;

   #define NR 10
   int x[NR],n,p;

   int folosit[NR];

   void backr(int k) {
     int i;

     if ( k > p ) { // s-a format
o solutie din p elemente
       for ( i = 1; i <= p; i++ )
         cout<<x[i]<<" ";
       cout<< "\n";
     }
     else
       for (i=1; i<=n; i++)
       {
           x[k]=i;

         if ( !folosit[x[k]] ) {
// daca elementul curent nu e deja
folosit
           folosit[x[k]] = 1;
// marcheaza-l ca folosit
           backr(k + 1 );     //
genereaza restul aranjamentului
           folosit[x[k]] = 0;
// demarcheaza elementul
         }
   }
   }

   int main() {

     cout<<"Dati n:";
     cin>>n;
     cout<<"Dati p:";
     cin>>p;
```

[1] Matematic, prin aranjamente de n luate câte p se înţelege numărul aplicaţiilor injective de la mulţimea {1,2,...,p} la mulţimea {1,2,...,n}.

```
   if (succesor(k))
  {
   x[k]++;
   if (continuare(k)) k++;
  }
  else {init(k); k--;}
}
int main()
{
 cout<<"n= "; cin>>n;
 cout<<"p= "; cin>>p;
 backtracking();
}
```

```
   backr(1);
   return 0;
}
```

10.1.3 Generarea combinărilor

Fie din nou *n* obiecte distincte $a_1,a_2,\ldots,a_n$. Dacă $1 \le m \le n$ o *combinare* (*simplă*) *a celor n obiecte luate câte m* este o mulţime (grupare) $\{a_{i_1},\ldots,a_{i_m}\}$ de *m* obiecte distincte din cele *n* date. Două combinări $\{a_{i_1},...,a_{i_m}\}$, $\{a_{j_1},...,a_{j_m}\}$ a celor *n* obiecte luate câte *m* se consideră *distincte* dacă ele diferă prin natura lor, adică există $s \le m$ astfel încât obiectul a_{i_s} este diferit de toate obiectele a_{j_t}, t=1,2,…,*m*.

Fie mulţimea $A=\{a_1,a_2,a_3\}$ şi să considerăm toate *submulţimile* sale, care sunt următoarele:

1) mulţimea vidă: $\varnothing$;

2) 3 submulţimi formate din câte un singur element: $\{a_1\}$, $\{a_2\}$, $\{a_3\}$;

3) 3 submulţimi formate din câte două elemente fiecare: $\{a_1,a_2\}$, $\{a_1,a_3\}$, $\{a_3,a_2\}$;

4) mulţimea totală: $\{a_1,a_2,a_3\}$.

Fie obiectele distincte *a*,*b*,*c*. Combinările (simple) ce se pot forma cu aceste trei obiecte luate câte două sunt: $\{a,b\}$, $\{a,c\}$, $\{c,b\}$.

Date cele *n* obiecte distincte a_1, a_2, … , a_n, vom nota numărul tuturor combinărilor celor *n* obiecte luate câte *m* cu C_n^m sau $\binom{n}{m}$. Formula de calcul al acestui număr este dată în rezultatul ce urmează:

$$C_n^m = \frac{n(n-1)...(n-m+1)}{m!}.$$

Enunţ: ***(Generarea combinărilor).*** Se citesc **n** şi **p** numere naturale, cu **n** mai mare sau egal cu **p**. Se cere să se genereze toate submulţimile cu **p** elemente ale mulţimii **{1,2,...,n}**. Două mulţimi se consideră egale dacă şi numai dacă au aceleaşi elemente, indiferent de ordinea în care acestea apar.

Soluţie. Programul este asemănător cu cel de generare a permutărilor, cu următoarele deosebiri:

- condiţiile de continuare verifică în plus dacă $\mathbf{x_k<x_{k-1}}$ (evitând în acest fel generarea a două soluţii cu aceleaşi elemente dar în altă ordine);
- condiţia ca mulţimea valorilor pentru $\mathbf{x_k}$ să nu fie epuizată este $\mathbf{x_k<n-p+k}$;
- condiţia care garantează obţinerea unei soluţii este **k=p+1.**

Comparativ cu problema aranjamentelor și permutărilor, ordinea elementelor în soluțiile generate nu este importantă.

De exemplu pentru o soluție generată: {1,2,3} nu va mai trebui să generăm și soluția {1,3,2}.

Astfel, condiția de continuare, sau de validare a unui element este aceea că el trebuie să fie strict mai mare decât cel plasat anterior. În acest mod asigurăm faptul că elementele nu se vor repeta şi că vor fi generate în ordine strict crescătoare. Trebuie, însă, să avem grijă să nu punem această condiţie și asupra primului element din vectorul soluţie, deoarece el nu are cu cine să fie comparat.

Varianta nerecursivă

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n,p,k;
int init(int k)
{
 x[k]=0;
}
int succesor(int k)
{
 if (x[k]<n-p+k) return 1;
 else return 0;
}
int continuare(int k)
{
 int i;
 if (k>1)
  if (x[k]<x[k-1]) return 0;
 for (i=1;i<=k-1;i++)
  if (x[k]==x[i]) return 0;
 return 1;
}
int solutie(int k)
{
 if (k==p+1) return 1;
 else return 0;
}
int afisare()
{
 int i;
 cout<<"{ ";
 for (i=1;i<=p;i++)
  cout<<x[i]<<" ";
 cout<<"}\n";
}
int backtracking()
{
```

Varianta recursivă

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n, p;
// generam combinari de n luate
cate p


void comb( int n, int k) {
  int i;

  if ( k > p ) {
      //avem deja o solutie pe
care o afisam
    for ( i = 1; i <= p; i++ )
      cout<<x[i]<<" ";
    cout<< "\n";
  }
  else
    for (i=x[k-1]+1; i<=n; i++)
    {
        x[k]=i;
        comb( n, k + 1 );     //
mergem la pasul urmator, completam
x[k+1]
      }
}


int main() {
  cout<<"Dati n:";
  cin>>n;
  cout<<"Dati p:";
  cin>>p;
  comb(n, 1);
  return 0;
}
```

```
 int k,i;
 k=1; init(1);
 while (k!=0)
  if (solutie(k))
  {afisare(); k--;}
  else
  if (succesor(k))
  {
   x[k]++;
   if (continuare(k)) k++;
  }
  else {init(k); k--;}
}
int main()
{
 cout<<"n= "; cin>>n;
 cout<<"Multimea vida\n";
 for (p=1;p<=n-1;p++)
  backtracking();
 cout<<"{ ";
 for (p=1;p<=n;p++)
  cout<<p<<" ";
 cout<<"}\n";
}
```

10. 1.4 Generarea produsului cartezian

Enunț: ***(Generarea elementelor unui produs cartezian).*** Fie date **m** mulţimi $A_1, A_2, ..., A_m$ unde pentru fiecare $A_i=\{1,2,...,n_i\}$. Se pune problema generării tuturor celor $n_1 x n_2 x ... x_m$ elemente ale produsului cartezian $A_1 x A_2 x ... x A_m$.

Soluţie. Programul este asemănător cu cel de generare a permutărilor, cu următoarele deosebiri:

- condiţia de continuare este **k<m+1**;
- condiţia ca mulţimea valorilor pentru x_k să nu fie epuizată este $x_k<nr_k$; unde nr_k reprezintă numărul de elemente ale mulţimii A_k;
- condiţia care garantează obţinerea unei soluţii este **k=m+1**, unde **m** este numărul de mulţimi.

Am considerat mulţimile de forma $\{1,2..,a_n\}$ pentru a simplifica problema, în special la partea de citire si afişare, algoritmul de generare rămânând nemodificat.

Varianta nerecursivă

```
#include <iostream>
using namespace std;

#define MAX 10
int x[MAX],nr[MAX],m,i,k;
int init(int k)
{
 x[k]=0;
}
int succesor(int k)
```

Varianta recursivă

```
#include <iostream>
using namespace std;

#define MAX 10
int x[MAX],m,nr[MAX];

void afisare()
{
  int i;
   for ( i = 1; i <= m; i++ )
```

```
{
 if (x[k]<nr[k]) return 1;
 else return 0;
}

int continuare(int k)
{
 int i;
 if (k<m+1) return 1;
 return 0;
}

int solutie(int k)
{
 if (k==m+1) return 1;
 else return 0;
}

int afisare()
{
 int i;
 for (i=1;i<=m;i++)
  cout<<x[i]<<" ";
 cout<<"\n";
}

int backtracking()
{
 int k,i;
 k=1; init(1);
 while (k!=0)
  if (solutie(k))
  {afisare(); k--;}
  else
  if (succesor(k))
  {
   x[k]++;
   if (continuare(k)) k++;
  }
  else {init(k); k--;}
}

int main()
{
 cout<<"m= "; cin>>m;
 cout<<"Numarul de elemente ";
 for (i=1;i<=m;i++)
 {
  cout<<"ale multimii "<<i<<" = ";
  cin>>nr[i];
 }
 backtracking();
return 0;
}
```

```
      cout<<x[i]<<" ";
    cout<< "\n";
}

void backr(int k) {
  int i;

  if (k == m +1) { // s-a format o
solutie din m elemente
   afisare();
  }
  else
    for (i=1; i<=nr[k]; i++)
    {
        x[k] = i; // plasam
elementul i pe pozitia k
        backr(k+1); // mergem la
pasul k+1
    }
}

int main()
{
int i;
 cout<<"m= "; cin>>m;
 cout<<"Numarul de elemente ";
 for (i=1;i<=m;i++)
 {
  cout<<"ale multimii "<<i<<" = ";
  cin>>nr[i];
 }

  backr(1);
  return 0;
}
```

10. 1.5 Generarea submulţimilor unei mulţimi

Enunț: ***(Generarea tuturor submulţimilor unei mulţimi).*** Se consideră mulţimea **{1,2,...,n}**. Se cer toate submulţimile acestei mulţimi.

Soluţie. Observăm că pentru a obţine toate submulţimile unei mulţimi, este suficient să generăm pe rând, pentru mulţimea **S={1,2,...,n}** combinări de **n** luate câte **p**, cu **p=1,2,…,n-1**, la care trebuie să adăugăm mulţimea vidă şi mulţimea **S**.

Numărul tuturor submulţimilor unei mulţimi formate din *n* elemente este egal cu 2^n. Pentru orice număr natural $n \geq 0$ are loc egalitatea:

$$C_n^0 + C_n^1 + C_n^2 + ... + C_n^n = 2^n.$$

Suma din membrul stâng al acestei egalităţi reprezintă numărul tuturor submulţimilor unei mulţimi cu *n* elemente.

Pentru varianta recursivă generăm toate combinaţiile de lungime n cu valorile 0 şi 1. Pentru fiecare combinaţie parcurgem soluţia x şi afişăm elementele din mulţimea A cărora le corespund valori 1 în x.

Varianta nerecursivă

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n,p,k;
int init(int k)
{
 x[k]=0;
}
int succesor(int k)
{
 if (x[k]<n-p+k) return 1;
 else return 0;
}
int continuare(int k)
{
 int i;
 if (k>1)
  if (x[k]<x[k-1]) return 0;
 for (i=1;i<=k-1;i++)
  if (x[k]==x[i]) return 0;
 return 1;
}
int solutie(int k)
{
 if (k==p+1) return 1;
 else return 0;
}
int afisare()
{
 int i;
 cout<<"{ ";
 for (i=1;i<=p;i++)
```

Varianta recursivă

```
#include <iostream>
using namespace std;

#define MAX 10
int x[MAX],n;

void afisare()
{
 int i;
 cout<<"{";
 for (i=1;i<=n;i++)
   if(x[i]==1)
  cout<<i<<" ";
 cout<<"}\n";
}


void backr(int k)
{
  int i;
   if (k == n +1) { // s-a format
o solutie din n elemente
   afisare();
  }
  else
  for(i=0;i<=1;i++)
   {
       x[k]=i;  // plasam
elementul 0 sau 1 pe pozitia k
      backr(k+1);// mergem la
pasul k+1
   }
}
```

```
  cout<<x[i]<<" ";
 cout<<"}\n";
}
int backtracking()
{
 int k,i;
 k=1; init(1);
 while (k!=0)
  if (solutie(k))
  {afisare(); k--;}
  else
  if (succesor(k))
  {
   x[k]++;
   if (continuare(k)) k++;
  }
  else {init(k); k--;}
}

int main()
{
 cout<<"n= "; cin>>n;
 cout<<"Multimea vida\n";
 for (p=1;p<=n-1;p++)
  backtracking();
 cout<<"{ ";
 for (p=1;p<=n;p++)
  cout<<p<<" ";
 cout<<"}\n";
}
```

```
int main()
{
int i;
 cout<<"n= "; cin>>n;

  backr(1);
  return 0;
}
```

Generarea partiţiilor

Enunț: ***Generarea tuturor partițiilor unei mulţimi***

Se citeşte un număr natural n. Să se genereze partiţiile mulţimii {1,2,...,n}. O partiţie a unei mulţimi A se defineşte ca fiind un set de mulţimi A1, A2, ..., Ap nevide, distincte două câte două, iar reuniunea celor p mulţimi este A. De exemplu, mulţimea {1,2,3,4,5} are ca partiţie setul {1,3,4}{2,5} sau setul {1,4} {2} {3,5}.

Utilizăm în rezolvare o stivă de lungime n, în care x[i] = j are semnificaţia că elementul i aparţine mulţimii Aj. De exemplu, pentru mulţimea {1,2,3,4,5}, partiţia {1,3,4}{2,5} se memorează astfel: x = (1,2,1,1,2).

Enunț: ***Generarea tuturor partițiilor unui număr natural***

Fie n>0, natural. Să se scrie un program care să afişeze toate partiţiile unui număr natural n. Numim partiţie a unui număr natural nenul n o mulţime de numere naturale nenule {p1, p2, …, pk} care îndeplinesc condiţia p1+p2+ …+pk = n. Ex: pt n = 4 programul va afişa:

4=1+1+1+14=1+1+2
4=1+3
4=2+2
4=4

Observaţii:

- lungimea vectorului soluţie cel mult n;
- există posibilitatea ca soluţiile să se repete;
- condiţia de final este îndeplinită atunci când suma elementelor vectorului soluţie este n.

Am menţionat mai sus că vom folosi doi parametri, unul pentru poziţia în vectorul soluţie **x** şi un al doilea în care avem sumele parţiale la fiecare moment. Avem determinată o soluţie atunci când valoarea celui de-al doilea parametru este egală cu n. În această situaţie la fiecare plasare a unei valori în vectorul x valoarea celui de al doilea parametru se măreşte cu elementul ce se plasează în vectorul soluţie. Apelul procedurii back din programul principal va fi back(1,0).

Partițiile unei mulțimi

```
#include<iostream>
using namespace std;

# define MAX 10
int n,nc,x[MAX];

void afis()
{ int i,j;
  for(j=1;j<=nc;j++)
    {
      cout<<'{';
      for(i=1;i<=n;i++)
       if (x[i]==j) cout<<i<<' ';
      cout<<'}';
    }
  cout<<endl;
}

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

void main()
{
  cin>>n;
  back(1);
}
```

Partițiile unui număr

```
#include <iostream>
using namespace std;

# define MAX 10
int n, nr,x[MAX];

void afis(int k)
{
int i;
nr++;
cout<<"Solutia nr. "<< nr<<": ";
for(i=1;i<=k;i++)
    cout<<x[i]<<" ";
cout<<endl;
}

void back(int k, int sum)
{ int j;
if (sum==n)
    afis(k-1);
else for(j=1;j<=n-sum;j++)
if (j>=x[k-1])
{
x[k]=j;
back(k+1, sum+j);
}
}

int main()
{
cout<<"Dati n:";
cin>>n;
nr=0;
back(1,0);
cout<<nr<<" solutii";
}
```

10. 2 Teste grilă

1. O companie are 12 birouri și 2 angajate noi. În câte feluri se pot aloca birouri diferite celor 2 angajate?

a. 24 b. 132 c. 100 d. 23

2. Dacă se utilizează metoda backtracking pentru a genera toate permutările mulţimii **{a,b,c,d}** şi primele soluţii afişate sunt **dcba, dcab, dbca**, atunci penultima soluţie este:

a. acdb b. dcab c. abcd d. abdc

(Bacalaureat 2007, varianta 59, I, 4)

3. Într-o sală de laborator, scaunele sunt numerotate cu o literă mare urmată de un număr între 1 și 50. Câte scaune pot fi numerotate diferit în felul acesta? Se presupune că sunt 26 litere mari.

a. 1000 b. 76 c. 1300 d. 50

4. Un program generează şi scrie într-un fişier toate permutările unei mulţimi cu n elemente, câte o permutare pe un rând. Pentru n=5 câte rânduri va avea fişierul rezultat?

a. 25 b. 120 c. 100 d. 125

5. Dacă se utilizează metoda backtracking pentru a genera toate numerele naturale**,** în ordine strict crescătoare, formate din 4 cifre pare distincte, care dintre numerele de mai jos trebuie eliminate astfel încât cele rămase să reprezinte o succesiune de numere corect generată? 1)2068 2) 2084 3) 2088 4) 2468 5) 2086 6) 2406

a. numai 3 b. atât 3 cât şi 5 c. atât 3 cât şi 4 d. numai 4

(Bacalaureat 2007, varianta 61, I, 5)

6. Câte șiruri diferite de 7 biți există?

a. 128 b. 256 c. 100 d. 49

7. Folosind metoda backtracking, se generează toate numerele de 4 cifre distincte, cu proprietatea că cifrele aparţin mulțimii {7,8,3,2,5}. Primele 10 soluţii generate sunt: 7832, 7835, 7823, 7825, 7853, 7852, 7382, 7385, 7328, 7325. Indicaţi ce număr urmează după 2538:

a. 5783 b. 5782 c. 2537 d. 5738

(Bacalaureat 2007, varianta 71, I, 4)

8. Folosind numai cifrele {0,5,3,8}, se construiesc, prin metoda backtracking, toate numerele cu 3 cifre în care oricare două cifre alăturate nu au aceeaşi paritate. Se obţin, în ordine numerele: 505, 503, 585, 583, 305, 303, 385, 383, 850, 858, 830,838. Utilizând acelaşi algoritm pentru a obţine numere cu patru cifre din mulţimea {0,3,6,2,9}, în care oricare două cifre alăturate nu au aceeaşi paritate, al şaselea număr care se obţine este:

a. 3092 b. 3690 c. 6309 d. 3096

(Bacalaureat 2007, varianta 88, I, 6)

9. Un student trebuie să aleagă un proiect de programare din 3 liste. Prima listă conține 9 proiecte, a doua 8, iar a treia 12. Câte posibilități are:

a. 3 b. 10 c. 29 d. 30

10. În câte feluri putem alege 2 cărți scrise în limbi diferite dintr-o colecție de 5 cărți scrise în română, 9 scrise în engleză și 10 în germană?

a. 45 b. 185 c. 48 d. 100

11. Echipa națională de fotbal poate alege un jucător de la una din cele 3 echipe de fotbal clasate pe primele trei locuri în clasament. Cele 3 echipe au 20, 24 și respectiv 22 de jucători. Câți jucători diferiți pot fi aleși:

a. 60 b. 66 c. 10560 d. 10000

12. Câte permutări ale mulțimii {A, B, C, D, E, F, G, H} conțin literele A,B,C alăturate?

a. 60 b. 40320 c. 1000 d.720

13. Fiind date numele a n elevi care pot participa la un proiect, ce algoritm vom alege pentru a lista toate grupele formate din câte k participanți?

Se știe că într-o grupă ordinea este importantă.

a. Generarea permutărilor b. Generarea aranjamentelor c. Generarea combinărilor d. Generarea tuturor partițiilor

14. Utilizând metoda backtracking se generează toate permutările mulţimii {1,2,3,4}. Dacă primele trei permutări generate sunt, în această ordine: 1234, 1243, 1324 precizaţi care este permutarea generată imediat după 3412.

a. 3421 b. 3413 c. 4123 d. 3214

(Bacalaureat 2008, varianta 42, III, 1)

15. Pentru a genera toate numerele naturale cu exact 4 cifre şi care au cifrele în ordine strict descrescătoare, se poate utiliza un algoritm echivalent cu cel pentru generarea:?

a) aranjamente de 4 obiecte luate câte 10

b) combinări de 10 elemente luate câte 4

c) permutarea a 10 obiecte

d) produs cartezian

(Bacalaureat 2008, varianta 56, III, 2)

16. Folosind metoda backtracking, s-au generat toate secvenţele formate din 3 cifre, fiecare secvenţă generată având numai cifre din mulţimea {1,2,3,4}, oricare două cifre alăturate din secvenţă fiind fie ambele pare, fie ambele impare. Scrieţi secvenţa care lipseşte din şir :

111, 113 ,131, 133, 313, 331, 333, 222, 224, 242 , 244, 422, 424, 442, 444.

a. 311 b. 433 c. 222 d. nu lipsește

(Bacalaureat 2008, varianta 68, III, 1)

17. Se utilizează metoda backtracking pentru a genera toate submulţimile cu p elemente ale unei mulţimi cu m elemente.

Dacă m=7 şi p=4 atunci numărul de submulţimi generate este:

a. 60 b. 35 c. 5 d. 15

(Bacalaureat 2008, varianta 68, III, 1)

18. La un concurs sportiv sunt 5 echipe, iar în fiecare echipă sunt câte 10 elevi. Problema determinării tuturor grupelor de câte 5 elevi, câte unul din fiecare echipă, este similară cu generarea tuturor:

a. elementelor produsului cartezian AxAxAxAxA, unde A={1,2,…,10}

b. submulţimilor cu 5 elemente ale mulţimii {1,2,…,10}

c. permutărilor mulţimii {1,2,3,4,5}

d. partiţiilor mulţimii {1,2,…,10}

(Bacalaureat 2008, varianta 77, III, 1)

19. Un program construieşte elementele produsului cartezian AxBxC pentru mulţimile A={1,2,3,4}, B={1,2,3}, C={1,2}. Care dintre următoarele triplete NU va fi afişat?

a. (3,2,1) b. (1,3,2) c. (1,2,3) d. (2,2,2)

(Bacalaureat 2008, varianta 78, III, 1)

20. Problema generării tuturor codurilor formate din exact 4 cifre nenule, cu toate cifrele distincte două câte două, este similară cu generarea tuturor:

a. aranjamentelor de 9 elemente luate câte 4

b. permutărilor elementelor unei mulţimi cu 4 elemente

c. elementelor produsului cartezian AxAxAxA unde A este o mulţime cu 9 elemente

d. submulțimilor cu 4 elemente ale mulţimii {1,2,3,4,5,6,7,8,9}

(Bacalaureat 2008, varianta 79, III, 1)

21. Un comitet este alcătuit din membri astfel încât fiecare din cele 41 județe ale României contribuie cu un senator sau cu un deputat. Se presupun că există 2 senatori și 3 deputați din fiecare județ. Câte astfel de comitete se pot alcătui?

a. 100 b. 5^{40} c. 5^{41} d.1000

22. Se generează toate șirurile strict crescătoare de numere naturale nenule mai mici sau egale cu 4, având primul termen 1 sau 2, ultimul termen 4 și cu diferența dintre oricare doi termeni aflați pe poziții consecutive cel mult 2, obținându-se soluțiile (1, 2, 3,4), (1, 2, 4), (1, 3, 4), (2, 3, 4), (2, 4). Folosind aceeași metodă generăm toate șirurile strict crescătoare de numere naturale nenule mai mici sau egale cu 6, având primul termen 1 sau 2, ultimul termen 6 și diferența dintre oricare doi termeni aflați pe poziții consecutive cel mult 2, care dintre afirmațiile următoare este adevărată:

a. imediat după soluția (1, 3, 4, 5, 6) se generează soluția (2, 3, 4, 5, 6)

b. penultima soluție generată este (2,3,5,6)

c. imediat după soluția (1,2,4,6) se generează soluția (1,3,4,6)

d. în total sunt 13 soluții generate

23. Pentru a determina toate modalitățile de a scrie numărul 16 ca sumă de numere naturale nenule consecutive se folosește metoda backtracking. Câte soluții avem?

a. 2 b. 4 c. 0 d. 1

24. Se consideră mulțimile A = {1, 2, 3}, B = {1}, C = {2, 3, 4}. Elementele produsului cartezian AxBxC se generează, folosind metoda backtracking, în ordinea (1, 1, 2), (1, 1, 3), (1, 1, 4), (2, 1, 2), (2, 1, 3), (2, 1, 4), (3, 1, 2), (3, 1, 3), (3, 1, 4). Dacă prin același algoritm se generează produsul cartezian al mulțimilor AxBxC unde A = {x, y}, B = {x, u}, c = {x, y, z}, atunci cel de-al șaptelea element generat este:

a. (y,u,x) b. (y,x,x) c. (x,x,x) d. (y,y,x)

25. Un chestionar conține 10 întrebări. Fiecare întrebare are 4 răspunsuri posibile. Câte posibilități există de a completa acest chestionar?

a. 4^{20} b. 4^{10} c. 4^5 d. 40

26. Un chestionar conține 10 întrebări. Fiecare întrebare are 4 răspunsuri posibile. Câte posibilități există de a completa acest chestionar, dacă este permis să nu se dea răspunsuri la întrebări?

a. 4^{20} b. 5^{10} c. 4^5 d. 50

27. Câte șiruri de 8 biți încep cu 3 zerouri sau se termină cu 2 zerouri?

a. 48 b. 100 c. 1000 d. $2^5+2^6-2^3$

28. Se consideră algoritmul care generează în ordine strict crescătoare toate numerele formate cu 5 cifre distincte alese din mulțimea {1, 0, 5, 7, 9} în care cifra din mijloc este 0. Selectați numărul care precede și numărul care urmează secvenței de numere generate: 19075; 51079; 51097

a. 19057, 57019 b. 15079, 71059 c. 19057, 59071 d. 15097, 71095

29. Dacă pentru generarea tuturor submulțimilor unei mulțimi A = {1, 2, ..., n} cu $1 \leq n \leq 10$, se utilizează un algoritm backtracking astfel încât se afișează în ordine, pentru n=3, submulțimile {}, {1}, {2}, {3},{1, 2}, {1,3}, {2,3}, {1, 2, 3}, atunci, utilizând exact același algoritm pentru n = 4, în șirul submulțimilor generate, soluția a 7-a va fi:

a. {1,3} b. {1,4} c. {2,3} d. {2,4}

30. Pentru a determina toate modalitățile de a scrie numărul 8 ca sumă de numere naturale nenule distincte (abstracție făcând de ordinea termenilor) se folosește metoda backtracking obținându-se, în ordine, toate soluțiile 1+2+5, 1+3+4, 1+7, 2+6, 3+5. Aplicând exact același procedeu, se determină soluțiile pentru scrierea numărului 10. Câte soluții de forma 1+ ... există?

a. 6 b. 3 c. 4 d. 5

10.3 Probleme propuse

1. Folosind metoda backtracking să se genereze în ordine lexicografică cuvintele de câte patru litere din mulțimea A={a,b,c,d,e}, cuvinte care nu conțin două vocale alăturate. Primele 8 cuvinte generate sunt, în ordine: abab, abac, abad, abba, abbb, abbc, abbd, abbe.

2. Să se genereze și să numere toate permutările mulțimii {1,2,3,...,n} care încep cu valoarea 1.

3. Să se genereze toate delegațiile formate din 5 persoane, obligatoriu minim 2 femei să fie participante. Avem la dispoziție b bărbați și f femeii.

4. Se citește un număr. Să se genereze toate numerele având aceleași cifre ca el. Care este cel mai mare?

5. Să se genereze anagramele unui cuvânt dat de la tastatură.

6. Să se genereze toate numerele de lungime n formate doar cu cifre pare/impare.

7. Scrieți un program care să afișeze toate numerele de n (n<=10) cifre, formate numai din cifre distincte și care sunt divizibile cu 4.

8. Să se genereze numerele mai mici decât n citit care trecute în baza 2 au în componenta lor exact p cifre de 1.

9. Să se genereze toate secvențele în cod binar de lungime n. Pentru fiecare secvența se va genera numărul asociat în baza 10. Să se genereze toate codurile posibile.

10. Se citește un număr natural n. Să se afișeze toate modalitățile de a-l descompune ca sumă de numere naturale consecutive. Daca acest lucru nu este posibil, se va afișa mesajul „Imposibil”.
    Exemplu: Numărul 6 se poate scrie ca următoarele sume: 1+2+3.
    Numărul 16 nu poate fi scris ca sumă de numere consecutive.

10.4 Răspunsuri şi rezolvări

10.4.1 Teste grilă

1.b	2.a	3.c	4.b	5.c	6.a
7.a	8.a	9.c	10.b	11.c	12.d
13.b	14.a	15.b	16.a	17.b	18.a
19.c	20.a	21.c	22.c	23.c	24.b
25.b	26.b	27.d	28.a	29.1	30.d

10.4.2 Probleme

1. Rezolvarea problemei se face similar cu generarea aranjamentelor unei mulţimi, numerotăm literele cu 1,2,3,4,5 (a este 1, e este 5).

```
   #include <iostream>
   using namespace std;

   #define NR 10
   int x[NR],n, nc;

   int folosit[NR];
   char a[]={' ', 'a',
'b','c','d','e'};
   //vom lucra cu indicii
elementelor a are indicele 1

   int bun(int k)
   {
       if(k>=2)
        if (x[k]==1 && x[k-1] ==1
|| x[k]==5 && x[k-1] ==1 ||x[k]==1
&& x[k-1] ==1 || x[k]==1 && x[k-1]
==5)
          return 0;
       return 1;
   }
   void backr(int k) {
     int i;

     if ( k > 4 ) {
           nc++;
       cout<<"Solutia "<<nc<<":";
       for ( i = 1; i <= 4; i++ )
         cout<<a[x[i]];
       cout<< "\n";
     }
     else
   for (i=1; i<=n; i++)
       {
           x[k]=i;

         if (bun(k)) { // -//daca
elementul curent nu e deja folosit
si e bun (nu sunt doua vocale
alaturate

           backr(k + 1 );     //
//genereaza restul permutarii

         }
   }
   }

   int main() {
     n=5;
     backr(1);
     cout<<"Sunt "<<nc<<"
solutii";
     return 0;
}
```

2. Plasăm valoarea 1 pe prima poziţie şi începem generarea de la cea de-a 2-a, nu mai punem 1 în x[k].

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n, nc;

int folosit[NR];

void backr( int n, int k) {
  int i;

  if ( k > n ) {
    nc++;
  cout<<"Solutia "<<nc<<":";
    for ( i = 1; i <= n; i++ )
      cout<<x[i]<<" ";
    cout<< "\n";
  }
else
    for (i=2; i<=n; i++)
{ x[k]=i;
      if ( !folosit[x[k]] ) { // -
//daca elementul curent nu e deja
//folosit
        folosit[x[k]] = 1;    //
//marcheaza-l ca folosit
        backr( n, k + 1 );     //
//genereaza restul permutarii
        folosit[x[k]] = 0;    //
//demarcheaza elementul
      }
} }
int main() {
  cout<<"Dati n:";
  cin>>n;
  x[1]=1;
  backr( n, 2);
  cout<<"Sunt "<<nc<<" solutii";
  return 0; }
```

3. Notez femeile cu 1,2 …f, iar bărbaţii îi notez în continuare cu f+1,….f+b.

```
#include <iostream>
using namespace std;

#define NR 6
int x[NR], nc,f,b,nf;

int folosit[NR];

void backr(int k) {
  int j;

  if (k>5)
    {
    if(nf>=4)
    {
    nc++;
  cout<<"Solutia "<<nc<<":";
    for ( j = 1; j <= 5; j++ )
      cout<<x[j]<<" ";
    cout<< "\n";
    }
  }
  else
    for (int i=1; i<=f+b; i++)
    {
        x[k]=i;
if (!folosit[x[k]]) { // -//daca
elementul curent nu e deja
//folosit
        folosit[x[k]] = 1;
        if(x[k]<=f)
            nf++;// //marcheaza-l
ca folosit
        backr(k + 1 );     //
//genereaza restul permutarii
        folosit[x[k]] = 0;
        // //demarcheaza
elementulf(x[k]<=f)
          if(x[k]<=f)
            nf--;
      }
} }
int main() {
  cout<<"Dati numarul de femei:";
  cin>>f;
  cout<<"Dati numarul de
barbati:";
  cin>>b;
  backr(1);
  cout<<"Sunt "<<nc<<" solutii";
  return 0; }
```

4. Descompun numărul și îi rețin cifrele într-un vector.

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n,mx,nr,a;

int folosit[NR],cifre[NR];

void backr( int n, int k) {
  int i;

  if ( k > n ) {
        nr = 0;
    for ( i = 1; i <= n; i++ ){
      cout<<x[i];
      nr = nr * 10 + x[i];
    }
    if(nr > mx)
        mx = nr;
    cout<< "\n";
  }
else
    for (i=1; i<=n; i++)
    {
        x[k]=cifre[i];

      if ( !folosit[i] ) { // -
//daca elementul curent nu e deja
//folosit
        folosit[i] = 1;    //
//marcheaza-l ca folosit
        backr( n, k + 1 );     //
//genereaza restul permutarii
        folosit[i] = 0;    //
//demarcheaza elementul
      }
}}
int main() {
    int i = 0;
    cout<<"Dati a:";
    cin>>a;
    while(a)
    {
        i++;
        cifre[i] = a % 10;
        a /= 10;    }
backr( i, 1);
    cout<<"Cea mai mare valoare
este: " << mx;
    return 0;}
```

5. Citim un șir, calculăm lungimea lui și punem în stivă direct literele lui.

```
#include <iostream>
using namespace std;

#define NR 10
int n;
char x[NR];

int folosit[NR];
string s;

void backr( int n, int k) {
  int i;

  if ( k > n - 1) {
    for ( i = 0; i < n; i++ )
      cout<<x[i];
    cout<< "\n";
  }
  else
for (i=0; i<n; i++)
    {
        x[k]=s[i];

      if ( !folosit[i] ) { // -
//daca elementul curent nu e deja
//folosit
        folosit[i] = 1;    //
//marcheaza-l ca folosit
        backr( n, k + 1 );     //
//genereaza restul permutarii
        folosit[i] = 0;    //
//demarcheaza elementul
      }
}
}

int main() {
  cout<<"Dati s:";
  cin>>s;
  n = s.size();
  backr( n, 0);
  return 0;
}
```

6. În stivă putem pune toate cifrele, condiția de continuitate este ca elementul pus în x[k] să fie de aceeași paritate cu cel din x[k-1];

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n;

int bun(int k)
{
    if(k == 1 && x[1] == 0)
        return 0;
    if(k > 1){
        if(x[k-1] % 2 == 0 && x[k]
% 2 == 0)
            return 1;
        else
            if(x[k-1] % 2 != 0 &&
x[k] % 2 != 0)
                return 1;
                else
                    return 0;
    }
    return 1;
}
```

```
void backr( int n, int k) {
  int i;
  if ( k > n ) {
    for ( i = 1; i <= n; i++ )
      cout<<x[i];
    cout<< "\n";
  }
  else
    for (i=0; i<=9; i++)
    {
        x[k]=i;
      if ( bun(k) ) { // -//daca
elementul curent nu e deja
//folosit
        backr( n, k + 1 );     //
//genereaza restul permutarii
      }
}
}
int main() {
  cout<<"Dati n:";
  cin>>n;
  backr( n, 1);
  return 0;}
```

7. Punem în stivă cifrele, dar fără 0 pe prima poziție. La completarea vectorului x, calculăm numărul care are aceste cifre.

```
#include <iostream>
using namespace std;

#define NR 10
int x[NR],n;
long long num;

int folosit[NR];

int bun(int n)
{
    int i;
    num = 0;
    if(x[1] == 0)
        return 0;
    for(i = 1; i <= n; i++){
        num = num * 10 + x[i];
    }
    if(num % 4 == 0)
        return 1;
    return 0;
}
void backr( int n, int k) {
  int i;
```

```
else
    for (i=0; i<=9; i++)
    {
        x[k]=i;

      if ( !folosit[x[k]] ) { // -
//daca elementul curent nu e deja
//folosit
        folosit[x[k]] = 1;    //
//marcheaza-l ca folosit
        backr( n, k + 1 );     //
//genereaza restul permutarii
        folosit[x[k]] = 0;    //
//demarcheaza elementul
      }
}
}

int main() {
  cout<<"Dati n:";
  cin>>n;
  backr( n, 1);
  return 0;
}
```

```
if ( k > n ) {
    if(bun(n))
    {
        for ( i = 1; i <= n; i++ )
          cout<<x[i];
        cout<< "\n";
    }
  }
```

8. Generăm toți vectorii formați doar din 0 și 1 și verificăm la fiecare pas să nu punem mai mult de p de 1. Tipărim o soluție dacă are exact p de 1.

```
#include <iostream>
using namespace std;

#define MAX 10
int x[MAX],n,sum, p;
int bun(int k)
{
  if(sum<=p)
        return 1;
  else
       return 0;
}


void backr(int k)
{
  int i;
   if (k == n +1) { // s-a format
o solutie din n elemente
   afisare();
  }
  else
  for(i=0;i<=1;i++)
   {
       x[k]=i;
       if (bun(k)){
       if(i == 1)
         sum = sum+1;   // plasam
elementul 0 sau 1 pe pozitia k
       backr(k+1);// mergem la
pasul k+1
       if(i==1)
         sum = sum-1;}
   }
}
```

```
void afisare()
{
 int i;
 if(sum==p)
 {
 cout<<"{ ";
 for (i=1;i<=n;i++)
  cout<<x[i]<<" ";

 cout<<"}\n";
 }
}


int main()
{
int i;
 cout<<"n= ";
 cin>>n;
cout<<"p= ";
 cin>>p;
  backr(1);
  return 0;
}
```

9. Generăm toți vectorii formați doar din 0 și 1. La afișarea unei soluții calculăm numărul din baza 10 care are această descompunere în baza 2.

```
#include <iostream>
using namespace std;
```

```
void backr(int k)
{
  int i;
```

```
#define MAX 10
int x[MAX],n,sum, p;

void afisare()
{
 int i;
 cout<<"{ ";
 int sum=0;
 for (i=1;i<=n;i++)
  {
    cout<<x[i]<<" ";
    sum = sum*2 + x[i];
  }

 cout<<"} "<<" este descompunerea
numarului "<<sum <<"\n";
}
```

```
   if (k == n +1) { // s-a format
o solutie din n elemente
   afisare();
  }
  else
  for(i=0;i<=1;i++)
   {
       x[k]=i;
       backr(k+1);// mergem la
pasul k+1

   }
}

int main()
{
int i;
 cout<<"n= ";
 cin>>n;
  backr(1);
  return 0;
}
```

10. Calculăm suma la fiecare pas.

```
#include<iostream>

using namespace std;
#define MAX 10;
int n, nr,x[20];

void afis(int k)
{ int i;
  nr++;
  cout<<"Solutia:"<<nr<<":";
  for(i=1;i<=k;i++)
     cout<<x[i]<<" ";
  cout<<endl;
}

void back(int k, int sum)
{ int j;
  if (sum==n && k>2)
    afis(k-1);
  else
```

```
for(j=x[k-1]+1;j<=n-sum;j++)
          if (j==x[k-1]+1 || k==1)
             {
               x[k]=j;
               back(k+1, sum+j);
             }
}

int main()
{
  cout<<"Dati n:";
  cin>>n;
  nr=0;
  back(1,0);
  if (nr==0)
    cout<<"Imposibil";
  return 0;
}
```

Bibliografie

1. *Programarea în limbajul C/C++ pentru liceu,* Emanuela Cerchez, Marinel Şerban, Editura Polirom, 2005, Iaşi
2. *Informatică – Manual pentru clasa a XI-a, varianta Pascal şi C++,* Tudor Sorin, Vlad Huţanu, Editura L&S Infomat, 2006, Bucureşti
3. *Fundamentele programării – culegere de probleme pentru clasele IX-XI. Bacalaureat la informatică,* Dana Lica, D. P. Anastasiu şi alţii, Editura L&S Soft, Bucureşti
4. *Bac 2009: subiecte posibile*, Vlad Giorgie Daniel, Marcu Daniela ș. a., Editura CYGNUS, 2009, Suceava;
5. *Colecția revistei Gazeta de informatică.*
6. *Tehnici de programare,* Octavian Aspru, Editura Adias, 1997, Rm. Vâlcea.