# Fișa 02 – backtracking 1

1. Concepeți un program care citește un număr natural ***n***>1 și afișează pe ecran permutările mulțimii {1, 2, 3, …, ***n***}.
2. Concepeți un program care citește două numere naturale ***n***>1 și ***p***<=***n*** și afișează pe ecran aranjamentele elementelor mulțimii {1, 2, 3, …, ***n***} luate câte ***p***.
3. Concepeți un program care citește două numere naturale ***n***>1 și ***p***<=***n*** și afișează pe ecran combinările elementelor mulțimii {1, 2, 3, …, ***n***} luate câte ***p***.
4. Concepeți un program care citește un număr natural ***n***>1 și cele ***n*** elemente ale unei mulțimi de numere naturale și afișează pe ecran toate permutările mulțimii.
5. Concepeți un program care citește două numere naturale ***n***>1, ***p***<=***n*** și cele ***n*** elemente ale unei mulțimi de numere naturale și afișează pe ecran toate aranjamentele elementelor mulțimii luate câte ***p***.
6. Concepeți un program care citește două numere naturale ***n***>1, ***p***<=***n*** și cele ***n*** elemente ale unei mulțimi de numere naturale și afișează pe ecran toate combinările elementelor mulțimii luate câte ***p***. Observație: se vor afișa toate submulțimile cu ***p*** elemente ale mulțimii date.
7. Modificați programul 6 astfel încât să se citească o mulțime cu ***n*** elemente de la tastatură și să se afișeze toate submulțimile acesteia indiferent câte elemente au.
8. Concepeți un program care afișează pe ecran toate cuvintele de 4 litere care se pot forma folosind litere din mulțimea A={a, b, c, d, e}. Primele cuvinte afișate vor fi: *aaaa*, *aaab*, *aaac*, …
9. Concepeți un program care afișează pe ecran toate cuvintele de 4 litere care se pot forma folosind litere din mulțimea A={a, b, c, d, e} astfel încât să nu fie două vocale sau două consoane alăturate. Primele cuvinte afișate vor fi: *abab*, *abac*, *abad*, …
10. Concepeți un program care afișează toate posibilitățile de a aranja ***n*** nebuni pe o tablă de șah de dimensiuni ***n*** x ***n*** astfel încât să nu se atace între ei. Nebunii se aranjează în așa fel încât să fie câte un singur nebun pe fiecare coloană a tablei. De exemplu:

|  |  |
| --- | --- |
| **Intrare** | **Ieșire** |
| n = 3 | N N N  - - -  - - -  N - N  - - -  - N -  - - -  N N N  - - -  - N -  - - -  N - N  - - -  - - -  N N N |

1. Concepeți un program care afișează toate posibilitățile de a aranja ***n*** ture pe o tablă de șah de dimensiuni ***n*** x ***n*** astfel încât să nu se atace între ele. Realizați programul în așa fel încât să se afișeze și numărul soluției, ca în exemplu:

|  |  |
| --- | --- |
| **Intrare** | **Ieșire** |
| n = 2 | Soluția 1  T –  - T  Soluția 2  - T  T - |

1. *(punct în plus)* Concepeți un program care afișează toate posibilitățile de a aranja ***n*** regine pe o tablă de șah de dimensiuni ***n*** x ***n*** astfel încât să nu se atace între ele. Realizați programul în așa fel încât să se afișeze și numărul soluției.

*Rezolvarea primului program este dată mai jos pentru a fi folosită ca model (****verificați-o!****):*

//afisare permutari multime {1, 2, 3, ..., n}

// 1. Concepeți un program care citește un număr natural n>1 și afișează pe ecran permutările mulțimii {1, 2, 3, …, n}.

// OK

#include <iostream>

using namespace std;

int s[100]; //vectorul solutie

int n; //numarul de elemente din multime

int nrsol = 0;

//afisare solutie completa

void afis(int k)

{

cout << "\nSol. " << (++nrsol) << " : ";

for( int i=1 ; i<=k-1 ; i++ )

cout << s[i] << " ";

}

/\*

//genereaza urmatoarea valoare pe poz. k

int succesor(int k)

{

if(s[k]==n+1) return 0;

s[k]++;

return 1;

}

\*/

//genereaza urmatoarea valoare pe poz. k

void succesor(int k)

{

s[k]++;

}

//verifica daca pana la pozitia k avem solutie completa

int solutie(int k)

{

if(k==n+1) return 1;

else return 0;

}

//verifica daca elementul de pe poz. k este "bun"

int OK(int k)

{

for( int i=1 ; i<k ; i++ )

if(s[k]==s[i])

return 0;

return 1;

}

//verifica daca pana la pozitia k-1 avem solutie completa

//se asigura ca incepem cautarea de la primul element al multimii

void init(int k)

{

s[k]=0;

}

//prelucreaza valorile de pe pozitia k a solutiei

void back(int k)

{

if( solutie(k) )

afis(k);

else

{

init(k);

while( s[k]<n )

{

succesor(k);

if( OK(k) )

back(k+1);

}

}

}

int main()

{

//citim numarul de elemnte al multimii {1, 2, ..., n}

cout << "\nIntroduceti numarul de elemente : ";

cin >> n;

//pornim algoritmul de afisare a permutarilor

cout << "\nPermutarile sunt : \n";

back(1);

//gata

cout << "\n\n";

return 0;

}