**Metoda Backtracking – partea II**

**Aplicații**

1. **Generarea permutărilor**

Prin permutare a unei mulțimi înțelegem o aranjare a elementelor sale, într-o anumită ordine. Este cunoscut, printre altele, faptul că numărul de permutări ale unei mulțimi cu n elemente este Pn=n!=1⋅2⋅⋅⋯⋅n. Prin convenție, P0=0!=1.

Fie un număr natural n. Să se afișeze, în ordine lexicografică, permutările mulțimii {1,2,⋯,n}.

Ex. Pentru n=3, se va afișa:

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

Care sunt proprietățile unei permutări, pe care le va respecta și vectorul x[]:

* elementele sunt numere naturale cuprinse între 1 și n;
* elementele nu se repetă;
* vectorul x[] se construiește pas cu pas, element cu element. El va conține o permutare validă când va conține n elemente, desigur corecte.

Cele observate mai sus ne permit să precizăm condițiile specifice algoritmului backtracking, într-un mod mai formal:

* **condiții externe:** x[k]∈{1,2,⋯,n};
* **condiții interne:** x[k]∉{x[1],x[2],⋯,x[k−1]}, pentru k∈{2,3,⋯,n}
* **condiții de existență a soluției:** k=n

### **Sursă C++**

Următorul program afișează pe ecran permutările, folosind un algoritm recursiv:

#include <iostream>

using namespace std;

int x[10] ,n;

void Afis()

{

for( int j=1;j<=n;j++)

cout<<x[j]<<" ";

cout<<endl;

}

bool OK(int k){

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

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

return false;

return true;

}

bool Solutie(int k)

{

return k == n;

}

void back(int k){

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

{

x[k]=i;

if( OK(k) )

if(Solutie(k))

Afis();

else

back(k+1);

}

}

int main()

{

cin>>n;

back(1);

return 0;

}

#include <iostream>

using namespace std;

int n,st[20];

void afiseaza()

{

for (int j=1;j<=n;j++)

cout<<st[j]<<" ";

cout<<"\n";

}

int valid(int k)

{

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

if (st[j]==st[k])

return 0;

return 1;

}

void back(int k)

{

int i;

for(i=1;i<=n;i++)

{

st[k]=i;

if (valid(k))

if (k==n)

afiseaza();

else back(k+1);

}

}

int main()

{

cout<<"n=";cin>>n;

back(1);

return 0;

}

### **Semnificația funcțiilor**

* void Afis(); afișează soluția curentă. Când se apelează, vectorul soluție x are n elemente, reprezentând o permutare completă;
* bool OK(int k); verifică condițiile interne. La apel, x[k] tocmai a primit o valoare conform condițiilor externe. Prin funcția OK() se va verifica dacă această valoare este validă;
* bool Solutie(int k); verifică dacă avem o soluție completă. Acest lucru se întâmplă când permutarea este completă – am dat o valoare corectă ultimului element al tabloului, x[n], adică atunci când k=n;
* void back(int k); – apelul acestei funcții dă valori posibile elementului x[x] al vectorului soluție și le verifică:
  + se parcurg valorile pe care le pot lua elementele vectorului, conform condițiilor externe (în acest caz, 1..n);
    - se memorează în x[k] valoarea curentă;
    - dacă valoarea lui x[k] este corectă, conform condițiilor interne, se verifică dacă avem o soluție completă. În caz afirmativ se afișează această soluție, în caz contrar se trece la următorul element, prin apelul recursiv;
  + la finalul parcurgerii, se revine la elementul anterior al vectorului x, prin revenirea din apelul recursiv.

### **Observații**

* generarea valorilor din vectorul soluție începe cu primul element al acestuia, x[ 1 ]; în consecință, apelul principal al funcției back() este back(1);
* generarea permutărilor în ordine lexicografică se obține datorită faptului că, în funcția back() valorile posibile pe care le primește x[k] sunt parcurse în ordine crescătoare (for(int i=1 ; i<=n ; ++i)....). Dacă am fi parcurs valorile de la n la 1, s-ar fi generat permutările în ordine invers lexicografică;
* algoritmul este exponențial și poate fi folosit numai pentru valori mici ale lui n .

## O variantă (puțin) mai bună

Algoritmul de generarea a permutărilor este unul exponențial, deci lent. Totuși, poate fi ușor îmbunățit în ceea ce privește verificarea condițiilor interne. Acestea cer ca valoarea curentă a lui x[k] (elementul care se generează) să nu se repete. În varianta anterioară am parcurs elementele care îl preced și le-am comparat cu x[k].

Această parcurgere poate fi evitată dacă folosim un vector caracteristic, uz[], cu următorul înțeles:

uz[v]= 1dacă valoarea v a fost plasată deja în vectorul soluție,

0 dacă valoarea v nu a fost plasată încă în vectorul soluție

Următoarele programe folosesc această idee. Primul respectă îndeaproape schema anterioară, în timp ce al doilea este mai scurt – verificarea condițiilor interne și a celor de existență a soluției făcându-se în în funcția back(), fără a mai scrie funcții de sine stătătoare:

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  int x[10] , n , p, uz[10];  void Afis(int k)  {  for(int j = 1 ; j <= k ; j ++)  cout << x[j] << " ";  cout << endl;  }  bool OK(int k)  {  return uz[x[k]] == 0;  }  bool Solutie(int k)  {  return k == n;  }  void back(int k)  { for(int i = 1 ; i <= n ; ++ i)  {  x[k] = i;  if(OK(k))  {  uz[i] = 1;  if(Solutie(k))  Afis(k);  else  back(k + 1);  uz[i] = 0;  }  }  }  int main()  {  cin >> n;  back(1);  return 0;  } | #include <iostream>  using namespace std;  int x[10] , n , p, uz[10];  void Afis(int k)  {  for(int j = 1 ; j <= k ; j ++)  cout << x[j] << " ";  cout << endl;  }  void back(int k){  for(int i = 1 ; i <= n ; ++ i)  if(uz[i] == 0)  {  x[k] = i;  uz[i] = 1;  if(k == n)  Afis(k);  else  back(k + 1);  uz[i] = 0;  }  }  int main(){  cin >> n;  back(1);  return 0;  } |

Aplicatii:

* + - 1. generarea tuturor anagramelor unui cuvant
      2. generarea anagramelor cu coditia sa nu avem 2 consoane (vocale) alaturate

GENERAREA ARANJAMENTELOR

#include <iostream>

using namespace std;

//generarea aranjamentelor de n elemente - luate cate p

int x[10] , n , p, uz[10];

void Afis(int k)

{

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

cout << x[j] << " ";

cout << endl;

}

void back(int k){

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

if(uz[i] == 0)

{

x[k] = i;

uz[i] = 1;

if(k == p)

Afis(k);

else

back(k + 1);

uz[i] = 0;

}

}

int main()

{

cout<<"Dati n si p (p<=n):";cin >> n>>p;

back(1);

return 0;

}

PROBLEME DIN CATEGORIA ARANJAMENTELOR

* 1. GENERAREA TUTUTOR FUNCTIILOR INJECTIVE

f : A🡪B . Pentru simplificare putem considera ca domeniul de definiție este mulțimea A={1,2,...,m}, iar codomeniul mulțimea B={1,2,..., n} . Se obs. că pentru a avea soluții trebuie ca **m<=n.**

O soluție va avea **m** elemente și un element al soluției de pe nivelul k al stivei va reprezenta valoarea f(k) a funcției.

Problema se reduce la generarea aranjamentelor de **n** elemente luate cate **m**.

**#include <iostream>**

**using namespace std;**

**//generarea functiilor injective = generarea aranjamentelor**

**//de n elemente - luate cate m**

**int st[10] , n , m, uz[10];**

**void Afis(int k)**

**{ int i;**

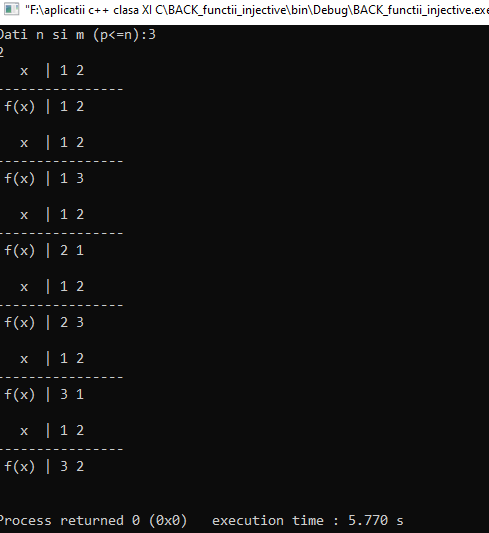
**cout<<" x | ";**

**for(i=1;i<=m;i++) cout<<i<<' ';cout<<endl;**

**for(i=1;i<=m;i++) cout<<"--------";**

**cout<<endl<<" f(x) | ";**

**for(i=1;i<=m;i++) cout<<st[i]<<" ";cout<<endl<<endl;**

**}**

**void back(int k){**

**for(int i = 1 ; i <= n ; ++ i)**

**if(uz[i] == 0)**

**{**

**st[k] = i;**

**uz[i] = 1;**

**if(k == m)**

**Afis(k);**

**else**

**back(k + 1);**

**uz[i] = 0;**

**}**

**}**

**int main(){**

**cout<<"Dati n si m (m<=n):";**

**cin >> n>>m;**

**back(1);**

**return 0;**

**}**

**2.Pb. 4 pag 21 manual clasa XI**

**Se citesc de la tastatura n caractere distincte(n<=10). Se cere sa se formeze toate cuvintele de m caractere (m<=n) ce se pot forma cu m caractere din cele n citite.**

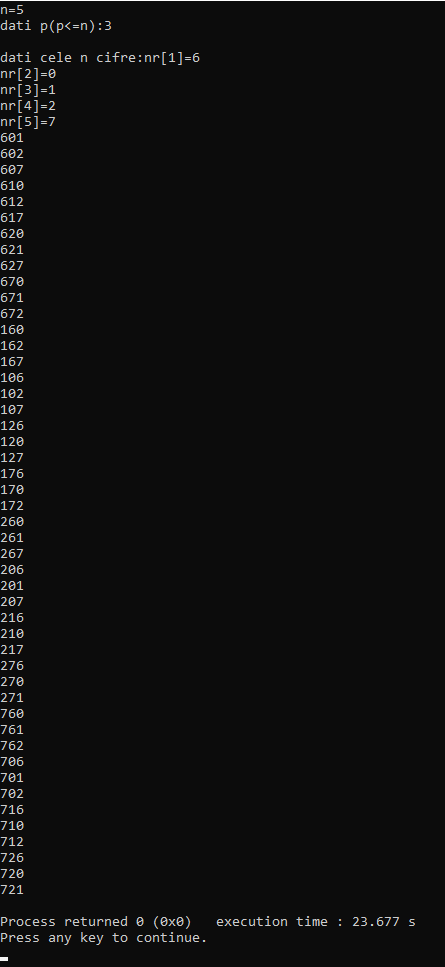
**3.Să se genereze toate numerele formate din m cifre distincte din n cifre posibile citite de la tastatură.**

Obs. că în șirul de cifre citite poate fi și cifra 0 , de aceea trebuie avut grije ca numerele generate să nu înceapă cu 0.

#include <iostream>

using namespace std;

int n,p,k,st[20],nr[10];

void afiseaza(int k) 

{

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

cout<<nr[st[j]];

cout<<"\n";

}

{

int valid(int k)

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

if (st[j]==st[k])

return 0;

if(k==1&& nr[st[k]]==0) return 0;

return 1;

}

void back(int k)

{

int i;

for(i=1;i<=n;i++)

{

st[k]=i;

if (valid(k))

if (k==p)

afiseaza(k);

else back(k+1);

}

}

int main()

{

cout<<"n=";cin>>n;

cout<<"dati p(p<=n):";cin>>p;

cout <<endl<<"dati cele n cifre:";

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

{cout<<"nr["<<i<<"]=";

cin>>nr[i];

}

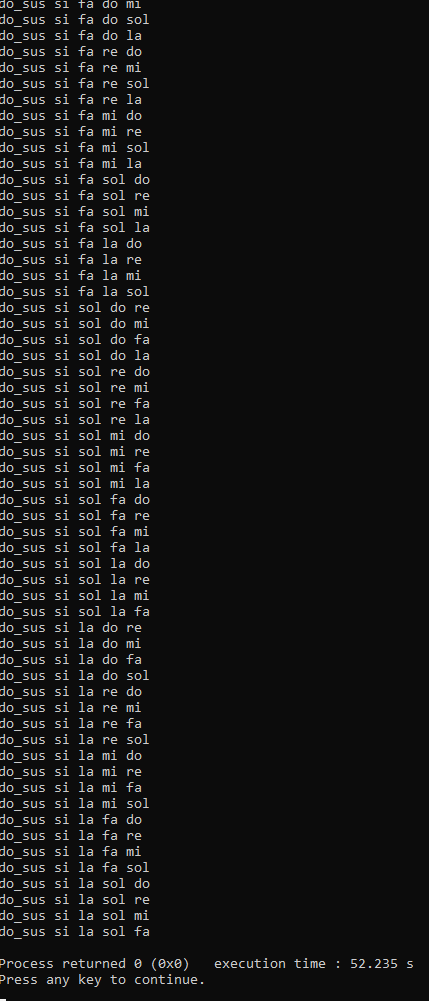
back(1);

return 0;

}

4.Se citesc de la tastatură cele 8 note muzicale . Se cere să se pună în evidentă toate posibilitățile de alcătui melodii în care sa fie alese în orice ordine doar p(p<8) dintre cele 8 note distincte .

#include <iostream>

using namespace std;

int n,p,k,st[10];char note[9][5];

void afiseaza(int k)

{

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

cout<<note[st[j]]<<" ";

cout<<"\n";

}

int valid(int k)

{for(int j=1;j<k;j++)

if (st[j]==st[k])

return 0;

return 1;

}

void back(int k)

{

int i;

for(i=1;i<=n;i++)

{

st[k]=i;

if (valid(k))

if (k==p)

afiseaza(k);

else back(k+1);

}

}

int main()

{

n=8;

cout<<"dati p(p<=n):";cin>>p;

cout <<endl<<"dati notele muzicale";

for(int i=1;i<=8;i++)

{cout<<"note["<<i<<"]=";

cin>>note[i];

}

back(1);

return 0;

}

5. Pb 8 manual cls. XI pag 21

Să se genereze toate drapelele tricolore ce se pot forma din 6 culori : alb, galben, roșu, verde , albastru și negru. Într-un drapel culorile sunt distincte și ordinea culorilor contează. Se vor genera doar drapelele care au în mijloc culoarea alb, verde sau roșu. (aranjamente cu condiție).

#include <iostream>

using namespace std;

int n,p,k,st[10];char culori[6][10];

void afiseaza(int k)

{

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

cout<<culori[st[j]]<<" ";

cout<<"\n";

}

int valid(int k)

{for(int j=1;j<k;j++)

if (st[j]==st[k])

return 0;

if(k==2)

if (culori[st[k]]=="galben"|| culori[st[k]]=="albastru"||culori[st[k]]=="negru")

return 0;

return 1;

}

void back(int k)

{

int i;

for(i=1;i<=n;i++)

{

st[k]=i;

if (valid(k))

if (k==3)

afiseaza(k);

else back(k+1);

}

}

int main()

{

cout<<"dati nr. de culori:";cin>>n;

cout <<endl<<"dati culorile :";

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

{cout<<"culori["<<i<<"]=";

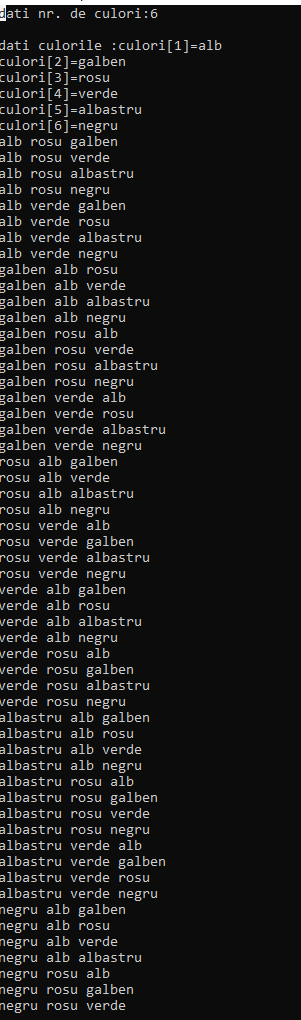
cin>>culori[i];

}

back(1);

return 0;

}



6. Să se genereze toate drapelele tricolore ce se pot forma din 9 culori : alb, turcoaz, rosu, galben, negru, albastru, verde , mov, maro. Într-un drapel culorile sunt distincte și ordinea culorilor contează. Se vor genera doar drapelele care au în mijloc culoarea alb, sau negru; prima culoare sa fie una deschisa , iar cea de-a treia sa fie o culoare inchisa. (aranjamente cu condiție).

#include <iostream>

using namespace std;

int n,p,k,st[10];char culori[9][10];

void afiseaza(int k)

{

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

cout<<culori[st[j]]<<" ";

cout<<"\n";

}

int valid(int k)

{int sw=1;

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

if (st[j]==st[k])sw=0;

if(sw) switch(k)

{case 1:if(!(st[k]<=4&&st[k]>=2))

sw=0;break;

case 2:if((st[k]!=1&&st[k]!=5))

sw=0;break;

case 3:if(!(st[k]<=9&&st[k]>=6))

sw=0;break;

}

return sw;

}

void back(int k)

{ int i;

for(i=1;i<=n;i++)

{ st[k]=i;

if (valid(k))

if (k==3)

afiseaza(k);

else back(k+1);

}

}

int main()

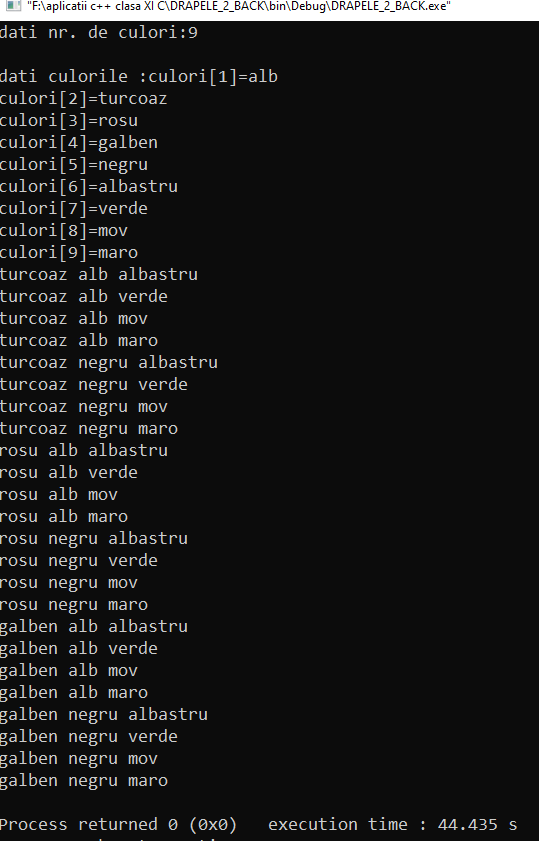
{ cout<<"dati nr. de culori:";cin>>n;

cout <<endl<<"dati culorile :";

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

{cout<<"culori["<<i<<"]=";

cin>>culori[i];

 }

back(1);

return 0;

}