PROIECTAREA ALGORITMILOR

Lect. univ. dr. Adrian Runceanu

Curs 10

Metoda Backtracking

Conţinutul 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 foloseşte în rezolvarea problemelor care îndeplinesc simultan următoarele condiţii:

- soluţia lor poate fi pusă sub forma unui vector S ={ x_1, x_2, ..., x_n }, cu $x_1 \in A_1$, $x_2 \in A_2$,…, $x_n \in A_n$
- mulţimile A_1, A_2 ,…., A_n sunt mulţimi finite, iar elementele lor se consideră că se află într-o relaţie de ordine bine stabilită
- nu se dispune de o altă metodă de rezolvare, mai rapidă
- x_1, x_2 ,…, x_n pot fi la rândul lor vectori
- A_1, A_2 ,…, 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 $A_1xA_2x…xA_n$ 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 mulţimi finite A, nu are rost să generăm produsul cartezian AxAx.....xA, pentru ca apoi, să testăm, pentru fiecare element al acestuia, dacă este sau nu permutare (nu are rost să generăm 1,1,1,...,1, pentru ca apoi să constatăm că nu am obţinut 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 construieşte soluţia pas cu pas: x_1, x_2, …, x_n*
- *dacă se constată că, pentru o valoare aleasă, nu avem cum să ajungem la soluţie, se renunţă la acea valoare şi se reia căutarea din punctul în care am rămas*

10.1 Prezentarea generala a metodei

Concret:

- se alege primul element x_1, ce aparţine lui A_1
- presupunând generate elementele $x_1,x_2,\ldots,x_k$, aparţinând mulţimilor A_1, $A_2,\ldots$, A_k, se alege (dacă există) x_{k+1}, primul element disponibil din mulţimea A_{k+1}, apar două posibilităţi:

  1) ***Nu s-a găsit un astfel de element***, caz în care se reia căutarea considerând generate elementele $x_1,x_2,\ldots,x_{k+1}$, iar aceasta se reia de la următorul element al mulţimii A_k rămas netestat

2) **A fost găsit**, caz în care se testează dacă acesta îndeplineşte anumite condiţii de continuare apărând astfel două posibilităţi:

– îndeplineşte, caz în care se testează dacă s-a ajuns la soluţie si apar din nou două posibilităţi:

- s-a ajuns la soluţie, se tipăreşte soluţia si se reia algoritmul considerând generate elementele $x_1,x_2,\ldots,x_k$, (se caută în continuare, un alt element al mulţimii A_k, rămas netestat)
- nu s-a ajuns la soluţie, caz în care se reia algoritmul considerând generate elementele $x_1,x_2,\ldots,x_{k+1}$, si se caută un prim element $x_{k+2} \in A_k$

- nu le îndeplineşte, caz în care se reia algoritmul considerând generate elementele $x_1,x_2,\ldots,x_k$, iar elementul x_{k+1} se caută între elementele mulţimii A_k, rămase netestate

- Algoritmii se termină atunci când nu mai există nici un element $x_1 \in A_1$ netestat.

Observaţie:

Tehnica ***Backtracking*** are ca rezultat obţinerea tuturor soluţiilor problemei.

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

- Am arătat că orice soluţie se generează sub formă de vector.

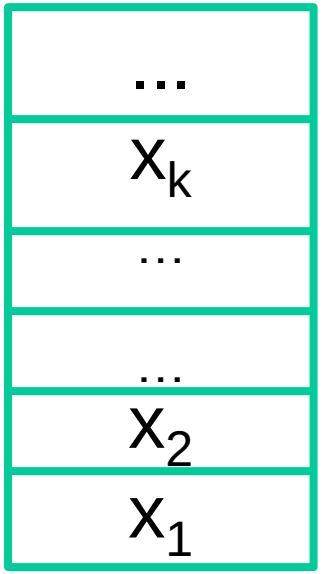
- Vom considera că *generarea soluţiilor se face intr-o stivă*.

- Metoda backtracking se poate implementa şi recursiv, dar implementarea iterativă foloseşte o stivă, iar principiul de lucrul 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 iniţializat (pentru a alege, în ordine, elementele mulţimii k+1).

- Iniţializarea trebuie făcută cu *o valoare aflată* (în relaţia de ordine considerată, pentru mulţimea A_{k+1}) *înaintea tuturor valorilor posibile din mulţime*.

Conţinutul cursului

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

Generarea permutărilor

Enunţ:

Se citeşte un număr natural n. Se cere să se genereze toate permutările mulţimii {1, 2,…,n}

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

Soluţiile ar fi:

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Observaţie:

Se observă, că elementele mulţimilor sunt în ordine strict crescătoare şi consecutive.

Aşadar modul de alegere din mulţime nu este aleatoriu ci se face în funcţie de ordinea elementelor.

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

1. **vectorul soluţie** – câte componente are, ce conţine fiecare componentă.
2. **mulţimea de valori posibile** pentru fiecare componentă (sunt foarte importante limitele acestei mulţimi).
3. **condiţiile de continuare** (condiţiile ca o valoare x_k să fie acceptată).
4. **condiţia ca ansamblul de valori generat să fie soluţie.**

- pentru generarea permutărilor mulţimii {1,2.....n}, orice nivel al stivei va lua valori de la 1 la n.

- Iniţializarea unui nivel (oarecare) se face cu valoarea 0.

- Procedura de iniţializare o vom numi **INIT** şi va avea doi parametri: k (nivelul care trebuie iniţializat si ST (stiva).

- Găsirea următorului element al mulţimii A_k (element care a fost netestat) se face cu ajutorul procedurii **SUCCESOR** (AS,ST,K).

- Parametrul AS (am succesor) este o variabilă booleană.

- În situaţia în care am găsit elementul, acesta este pus în stivă şi 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 îndeplineşte condiţiile 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 situaţia în care elementul îndeplineşte condiţiile de continuare, EV ia valoarea **TRUE**, in caz contrar (nu îndeplineşte condiţiile de continuare) EV ia valoarea **FALSE**.

- Testul dacă s-a ajuns sau nu la soluţia finală se face cu ajutorul funcţiei **SOLUTIE**(k)

- O soluţie se tipăreşte 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 condiţiile ce trebuiesc îndeplinite de
elementele aflate în stiva până la nivelul dat k.
Condiţiile se stabilesc astfel încât în cazul îndeplinirii lor
variabila ev să primească de fiecare dată valoarea false
(condiţiile se neagă!!!)
end;
functia solutie();
begin
        daca stiva este plina  atunci solutie<-true
                               altfel  solutie<-false;
end;
procedura tipar;
begin
        Se parcurge stiva şi 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
                succesor;
                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.
```

Pentru enunţul de la exemplul anterior, generăm, cu ajutorul stivei, soluţiile prin metoda backtracking.

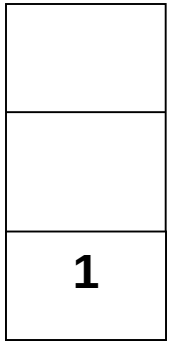
Pentru n=3:

pas 1:

• Se ia primul element din mulţimea A_1 şi se pune pe nivelul 1 al stivei.

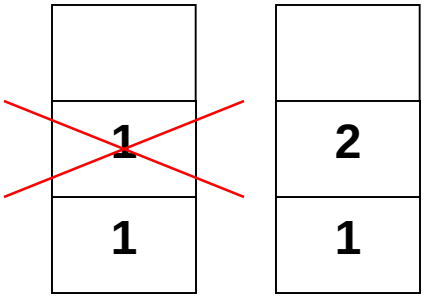
• Nu există deocamdată nici o restricţie deoarece există permutări ce încep cu 1.

• Se trece la completarea următorului nivel al stivei.



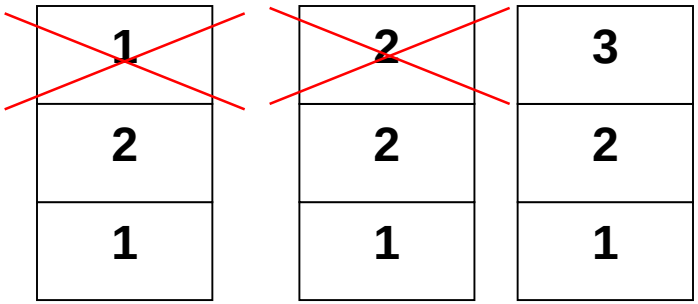
pas 2:

- Se ia primul element din mulţimea A_2 şi se pune pe nivelul 2 al stivei.
- Se observă că valoarea 1 a mai fost introdusă în vectorul soluţie, pe nivelul anterior, şi, în acest caz valoarea 1 nu poate fi ataşată la vectorul soluţie.
- Aşadar, se ia următorul element din mulţimea A_2, adică 2.
- Se constată că poate fi introdusă în vectorul soluţie.
- Se trece la completarea următorului nivel al stivei.



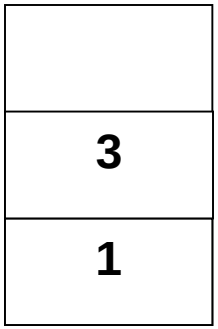
pas 3:

- Se ia primul element din mulţimea A_3 şi se pune pe nivelul 3 al stivei.
- Se observă că valoarea 1 a mai fost introdusă în vectorul soluţie, pe nivelul anterior, şi, în acest caz valoarea 1 nu poate fi ataşată la vectorul soluţie.
- Aşadar, se ia următorul element din mulţimea A_3, adică 2.
- La fel ca valoarea 1, 2 nu poate fi introdus în stivă.
- Se trece la următorul element din mulţime, 3.
- Se constată că poate fi introdusă în vectorul soluţie.
- S-au completat toate nivelele stivei şi deci soluţia este 1 2 3.



pas 4:

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



10.2. Implementarea metodei backtracking

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

Implementarea programului pentru generarea permutarilor:

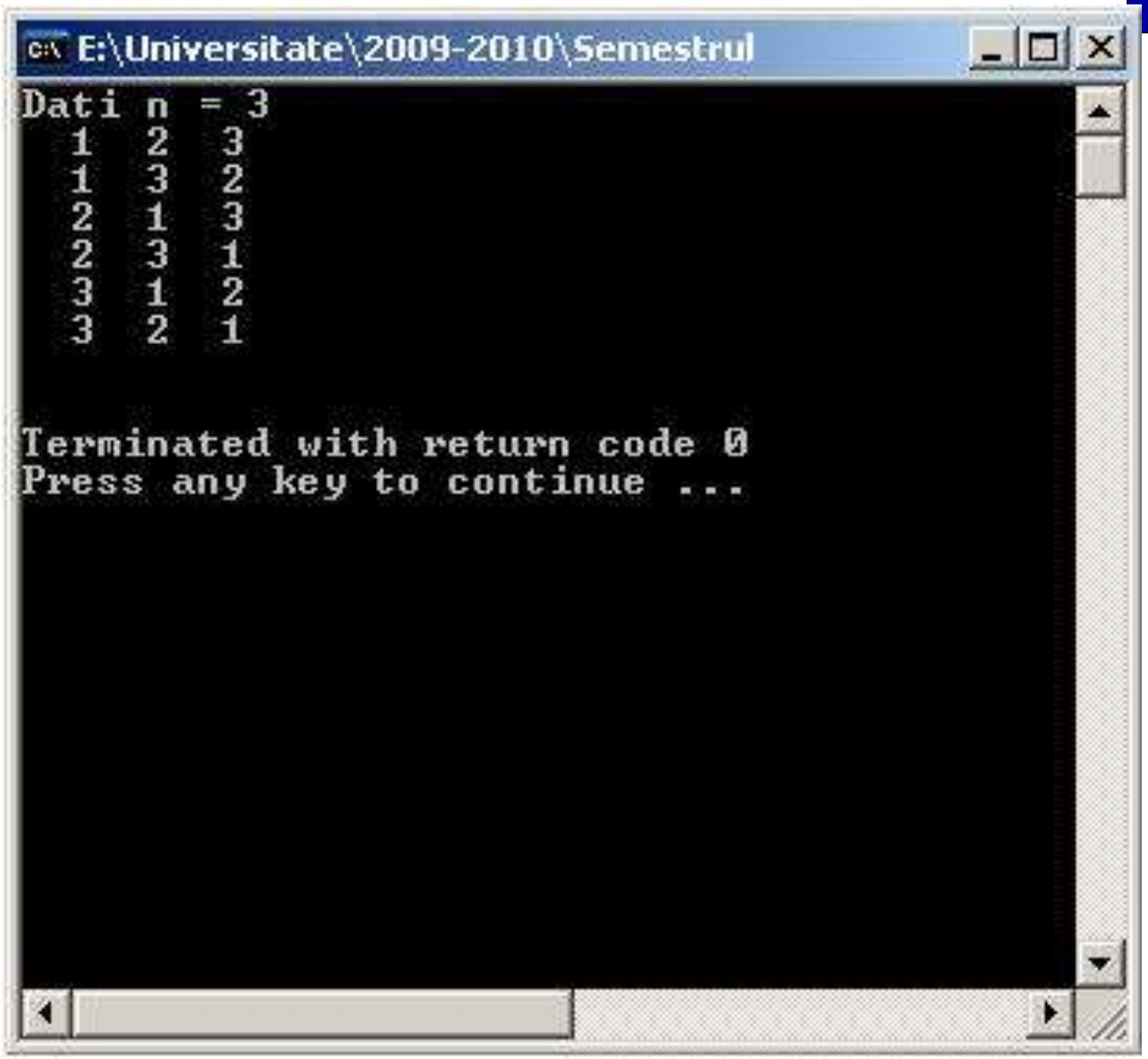
```
#include<iostream.h>
int st[20],i,k,p,q,m,n,as,ev;
void init(int k,int st[ ])
{
  st[k]=0;
}
int succesor(int k,int st[ ])
{
   if ( st[k]<n )
          {
           st[k]=st[k]+1;
           as=1;
           }
      else as=0;
     return as;
}
```

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)
{
   cout<<"Dati n = "; cin>>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--;
  }
}
```


E:\Universitate\2009-2010\Semestrul
Dati n = 3
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
Terminated with return code 0
Press any key to continue ...

Problema 2:

Fiind dată o tablă de şah, de dimensiune n*n, se cer toate soluţiile de aranjare a n regine, astfel încât să nu se afle două regine pe aceeaşi linie, coloană sau diagonală (reginele să nu se atace reciproc).

Afişarea să se facă astfel:

Exemplu: Pentru n=4 se va afişa:

Soluţia 1:

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

Soluţia 2:

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

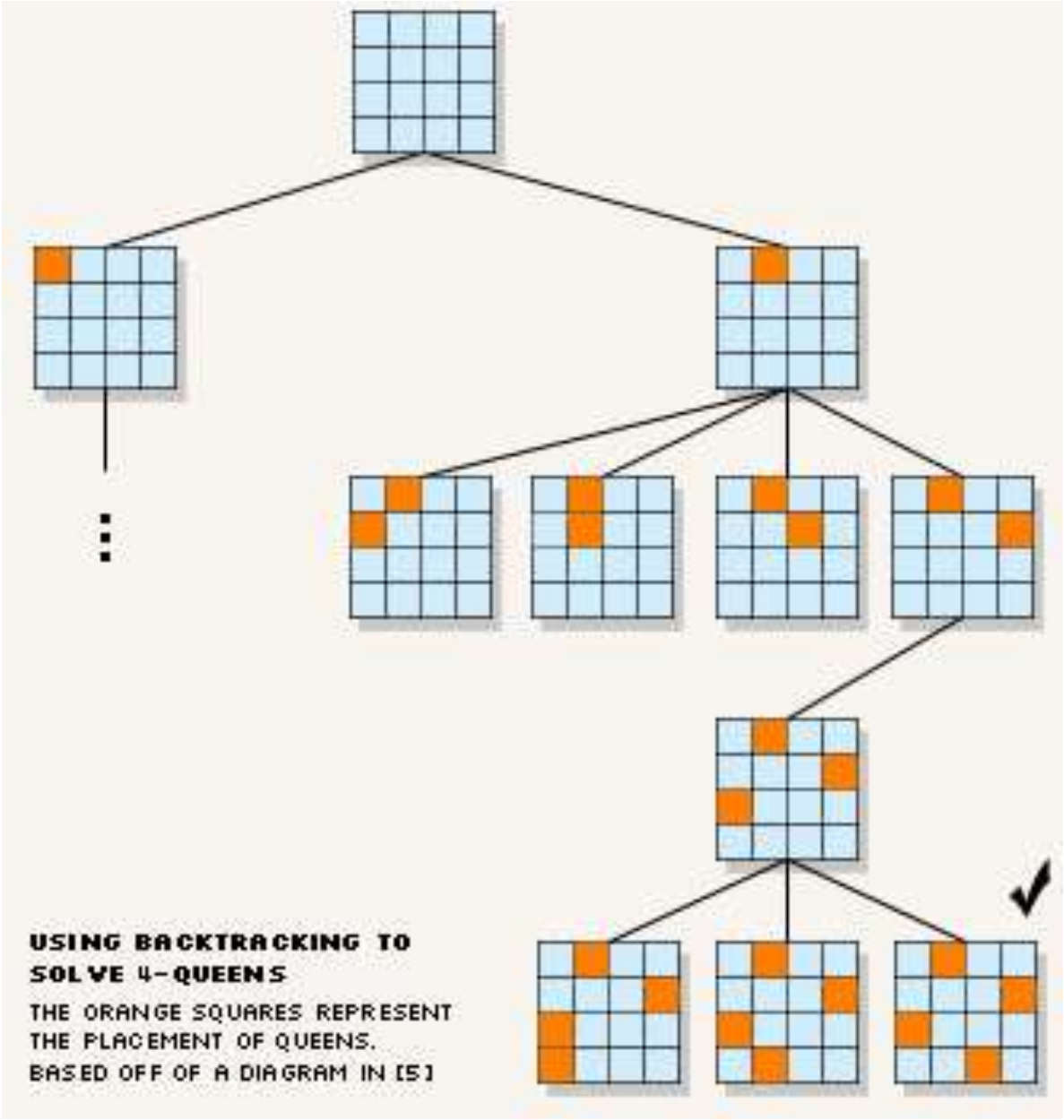
```
# include <iostream.h>
# include <math.h>

 int st[100];
 int k,n,i,a;
 int ev,as;

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

int succesor(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;
   }
}
```

```
int main(void)
{
   cout<<“Dati numarul de regine = “; cin>>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--;
  }
}
```


E:\Universitate\2009-2010\Semestrul
Dati numarul n= 4
Solutia nr. : 1
* R * *
* * * R
R * * *
* * R *
Solutia nr. : 2
* * R *
R * * *
* * * R
* R * *
Terminated with return code 0
Press any key to continue ...


USING BACKTRACKING TO
SOLVE 4-QUEENS
THE ORANGE SQUARES REPRESENT
THE PLACEMENT OF QUEENS.
BASED OFF OF A DIAGRAM IN [5]

Problema 3:

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

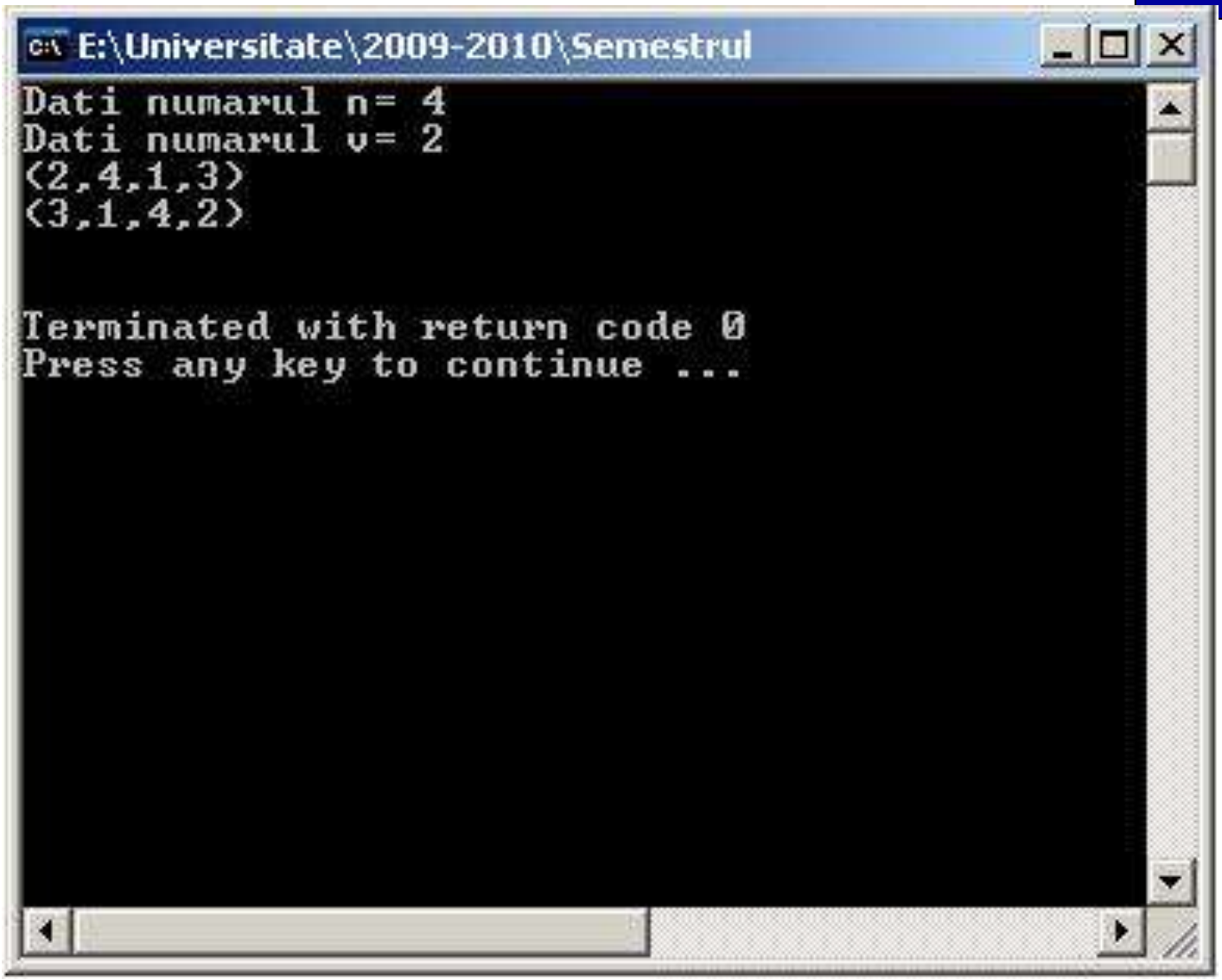
Exemplu:

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

```
# include <iostream.h>
# include <math.h>

 int st[100];
 int k,n,i,v;
 int ev,as;

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

int succesor(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()
{
      cout<<"Dati numarul n= ";cin>>n;
      cout<<"Dati numarul v= ";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--;
       }
}
```


E:\Universitate\2009-2010\Semestrul
Dati numarul n= 4
Dati numarul v= 2
(2,4,1,3)
(3,1,4,2)
Terminated with return code 0
Press any key to continue ...

Conţinutul 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, …, 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, …, 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 numar n natural nenul si se cere sa se afiseze toate descompunerile numarului n in suma de numere naturale.

Exemplu:
Pentru n=5 se vor afisa 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
5

Problema 4

Să se afişeze toate posibilităţile de aşezare a n ture pe o tablă de şah de n*n astfel încât să nu se atace una pe celalaltă.

Exemplu: pentru n=3

Soluţia 1 :

*** T ***
*** * T**
T* *

Soluţia 2 :

*** T ***
T * *
*** * T**

s.a.m.d.

Problema 5

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

Exemplu:
Pentru n=5 si p=3 se obtin solutiile:

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

Problema 6

Avem la dispoziţie 6 culori: alb, galben, roşu, verde, albastru şi negru. Să se precizeze toate drapelele tricolore care se pot proiecta, ştiind 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 roşu", "roşu verde galben"

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

1 – alb

2 – galben

3 – roşu

4 – verde

5 – albastru

6 - negru

Întrebări?