

PROIECTAREA ALGORITMILOR

Adrian Runceanu

Curs 11

Metoda Backtracking (continuare)

Conținutul cursului

11.1. Exemple de probleme rezolvate cu ajutorul metodei backtracking

11.2. Metoda backtracking – varianta recursiva

11.3. Backtracking generalizat in plan

Problema 1

Sa se genereze toate aranjamentele multimii $\{1, 2, \dots, n\}$, 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),

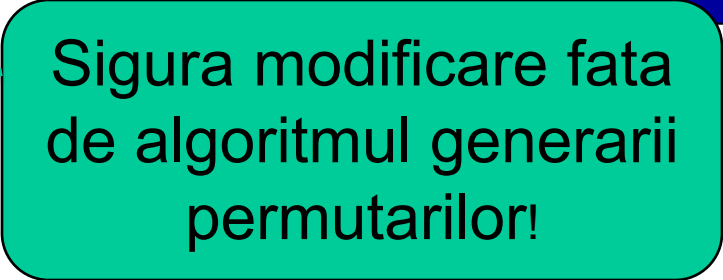
s.a.m.d.

Implementarea programului pentru generarea aranjamentelor

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

```
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==p ) return 1;
    else return 0;
}
void tipar(void)
{
for(i=1;i<=k;i++)
    cout<<" "<<st[i];
cout<<"\n";
}
```



Sigura modificare fata
de algoritmul generarii
permutarilor!

```
int main(void)
{
    cout<<"Dati n = "; cin>>n;
    cout<<"Dati p = "; cin>>p;
    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--;
    }
}
```

Problema 2

Sa se genereze toate combinarile multimii $\{1, 2, \dots, n\}$, cu cate p elemente.

Exemplu:

Pentru $n = 5$ si $p = 3$ se vor afisa multimile:

$(1,2,3)$, $(1,2,4)$, $(1,2,5)$, $(1,3,4)$, $(1,3,5)$, $(1,4,5)$,
 $(2,3,4)$, $(2,3,5)$, $(3,4,5)$.

Implementarea programului pentru generarea combinarilor:

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

```
int valid(int k,int st[ ]  
{  
    ev=1;  
    for (i=1;i<=k-1;i++)  
        if (st[i]==st[k]) ev=0;  
    if(k>1)  
        if(st[k]<st[k-1]) ev=0;  
    return ev;  
}
```

- Modificare fata de algoritmul generarii permutarilor!
- Elementele trebuie sa fie in ordine crescatoare

```
int solutie(int k)
{
if ( k==p ) return 1;
    else return 0;
}
void tipar(void)
{
    for(i=1;i<=k;i++)
        cout<<" "<<st[i];
    cout<<"\n";
}
```

Generam p combinari !



```
int main(void)
{
    cout<<"Dati n = "; cin>>n;
    cout<<"Dati p = "; cin>>p;
    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--;
    }
}
```



The screenshot shows a Windows command prompt window with the following text:

```
C:\E:\Universitate\2009-2010\Semestrul  
Dati n = 5  
Dati p = 3  
1 2 3  
1 2 4  
1 2 5  
1 3 4  
1 3 5  
1 4 5  
2 3 4  
2 3 5  
2 4 5  
3 4 5  
  
Terminated with return code 0  
Press any key to continue ...  
-
```

Problema 3

Se considera un număr n natural nenul și se cere să se afișeze toate descompunerile numărului n în suma de numere naturale.

Exemplu:

Pentru $n=5$ se vor afișa valorile:

$1+1+1+1+1$

$1+1+1+2;$ $1+1+2+1;$ $1+2+1+1;$ $2+1+1+1$

$1+2+2;$ $2+1+2;$ $2+2+1$

$1+1+3;$ $1+3+1;$ $3+1+1$

$1+4;$ $4+1$

$2+3;$ $3+2$

5

Implementarea programului:

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

```
int valid(int k,int st[ ])  
{  
    int s=0;  
    ev=1;  
    for(i=1;i<=k;i++)  
        s+=st[i];  
    if( s > n ) ev=0;  
    return ev;  
}
```

- Trebuie calculata suma elementelor aflate la un moment dat in stiva
- Daca suma depaseste pe n, atunci ev=0!

```
int solutie(int k)
{
    int s=0;
    for(i=1;i<=k;i++)
        s+=st[i];
    if( s == n ) return 1;
    else return 0;
}

void tipar(void)
{
    for(i=1;i<=k;i++)
        cout<<" "<<st[i];
    cout<<"\n";
}
```

- Trebuie calculata suma elementelor aflate la un moment dat in stiva
- Daca suma este egala cu n, atunci am gasit o solutie!

```
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--;
    }
}
```

```
c:\E:\Universitate\2009-2010\Semestrul
Dati n = 5
1 1 1 1 1
1 1 1 2
1 1 2 1
1 1 3
1 2 1 1
1 2 2
1 3 1
1 4
2 1 1 1
2 1 2
2 2 1
2 3
3 1 1
3 2
4 1
5

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

Problema 4

Să se descompună un număr natural n , în toate modurile posibile, ca sumă de p numere naturale ($p \leq n$).

Exemplu:

Pentru $n=5$ și $p=3$ se obțin soluțiile:

$1+1+3;$ $1+3+1;$ $3+1+1;$

$1+2+2;$ $2+1+2;$ $2+2+1;$

Implementarea programului:

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

```
int valid(int k,int st[ ]  
{  
    int s=0;  
    ev=1;  
    for(i=1;i<=k;i++)  
        s+=st[i];  
    if( s > n ) ev=0;  
    return ev;  
}
```

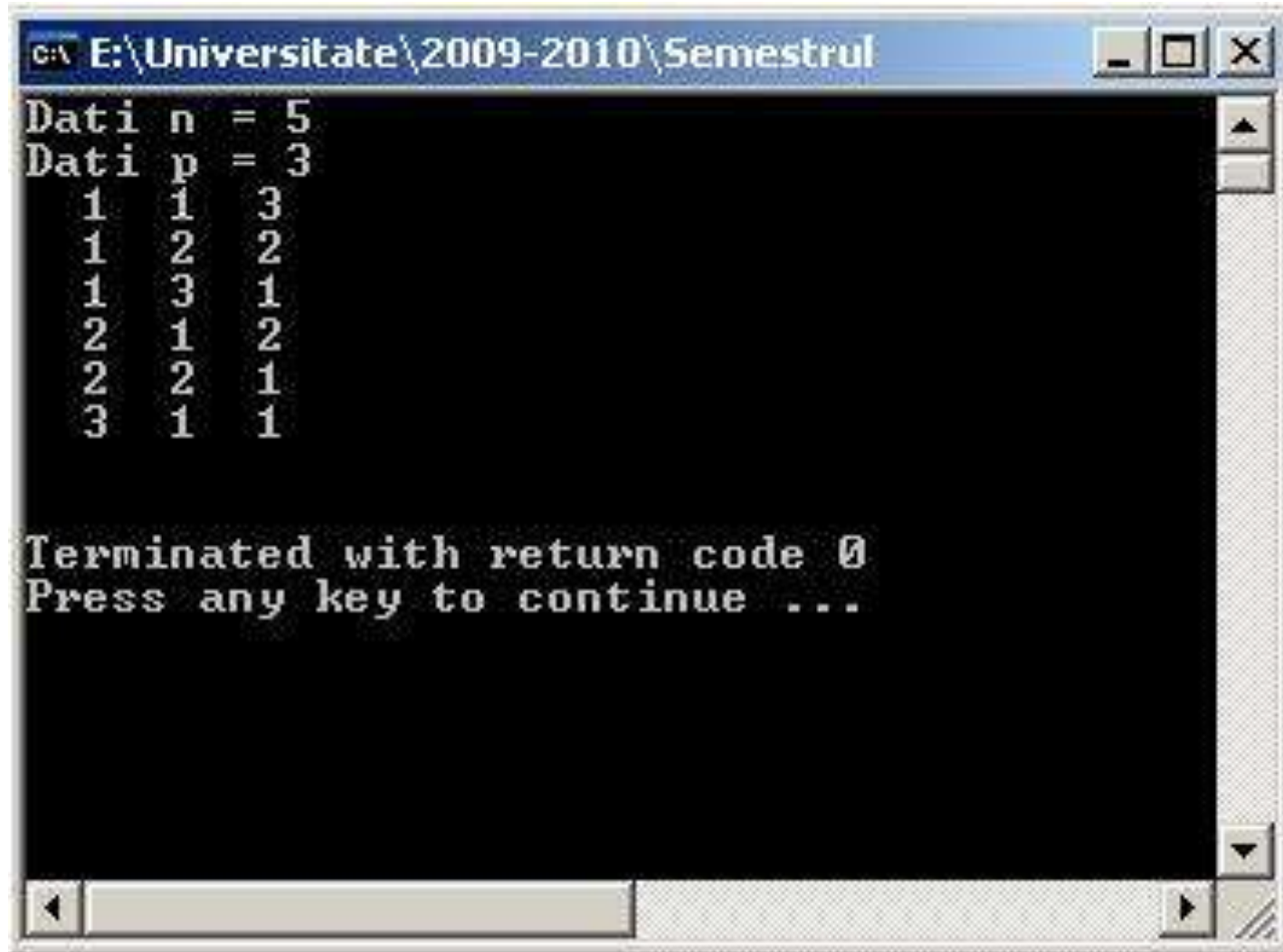
- Trebuie calculata suma elementelor aflate la un moment dat in stiva
- Daca suma depaseste pe n, atunci ev=0!

```
int solutie(int k)
{
    int s=0;
    for(i=1;i<=k;i++)
        s+=st[i];
    if( s == n && k == p ) return 1;
    else return 0;
}
```

```
void tipar(void)
{
    for(i=1;i<=k;i++)
        cout<<" "<<st[i];
    cout<<"\n";
}
```

- Trebuie calculata suma elementelor aflate la un moment dat in stiva
- Daca suma este egala cu n, atunci am gasit o solutie!
- In plus trebuie sa avem fix p elemente in stiva!

```
int main(void)
{
    cout<<"Dati n = "; cin>>n;
    cout<<"Dati p = "; cin>>p;
    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--;
    }
}
```



The screenshot shows a Windows command prompt window with the following text:

```
c:\ E:\Universitate\2009-2010\Semestrul  
Dat i n = 5  
Dat i p = 3  
 1 1 3  
 1 2 2  
 1 3 1  
 2 1 2  
 2 2 1  
 3 1 1  
  
Terminated with return code 0  
Press any key to continue ...
```

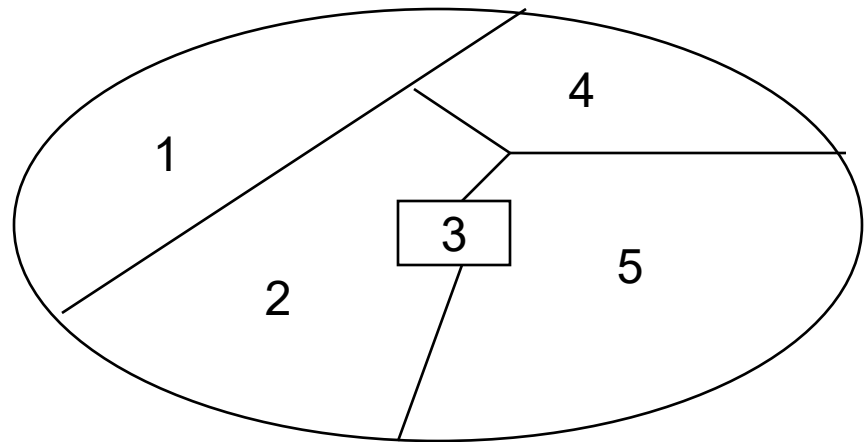
Problema 5

Problema colorării hărților

Fiind dată o hartă cu n țări, se cere **o soluție** de colorare a hărții, utilizând cel mult 4 culori, astfel încât două țări cu frontiera comună să fie colorate diferit.

O soluție este:

- Țara 1 – culoarea 1
- Țara 2 – culoarea 2
- Țara 3 – culoarea 1
- Țara 4 – culoarea 3
- Țara 5 – culoarea 4



Pentru a specifica harta utilizam o matrice patratica cu valori binare:

$$A(i,j) = \begin{cases} 1, & \text{daca țara } i \text{ are frontieră comună cu țara } j \\ 0, & \text{altfel} \end{cases}$$

- Se va utiliza stiva st, unde **nivelul k al stivei simbolizează țara k**, iar **st[k] culoarea atașată țării k**.
- Stiva are înălțimea n și pe fiecare nivel ia valori între 1 și 4, adică numărul culorilor este maxim 4.
- Condiția ca două țări vecine să aibă aceeași culoare este:

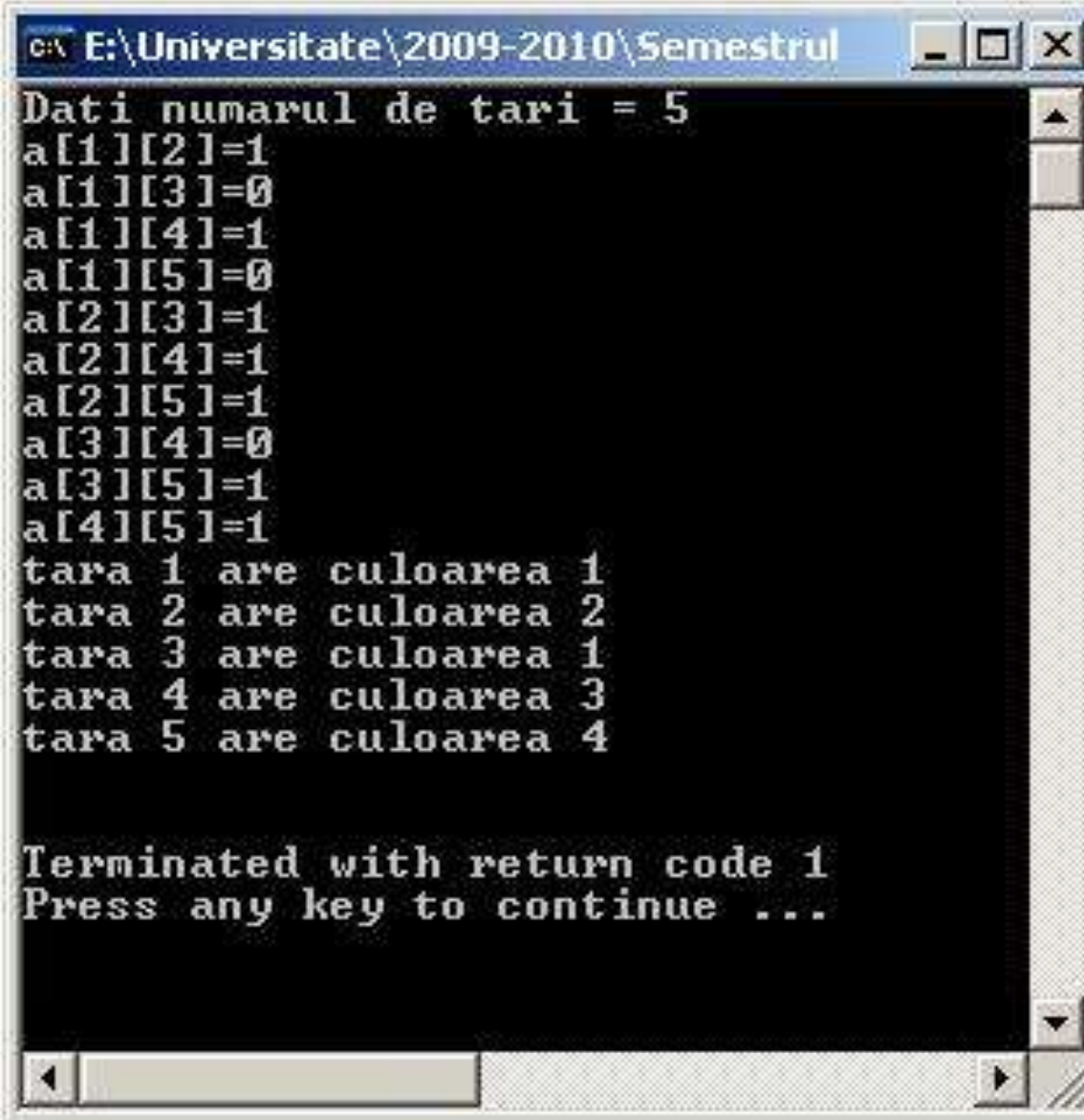
$$(st[k]==st[i]) \ \&\& \ (a[i][k]==1)$$

```
#include<iostream.h>
#include<stdlib.h>
int st[20],k,as,ev,n,i,j,a[20][20];
void init(int k,int st[ ])
{
    st[k]=0;
}
int sucesor(int k,int st[ ])
{
    if ( st[k]<4 )
        {
            st[k]=st[k]+1;
            as=1;
        }
    else as=0;
    return as;
}
```

```
int valid(int k,int st[ ]  
{  
    ev=1;  
    for(i=1;i<=k-1;i++)  
        if(st[k] == st[i] && a[i][k] == 1) ev=0;  
    return ev;  
}
```

```
int solutie(int k)
{
    if( k == n ) return 1;
        else return 0;
}
void tipar(void)
{
    for(i=1;i<=k;i++)
        cout<<"tara "<<i<<" are culoarea "<<st[i]<<"\n";
    exit(1);
}
```

```
int main(void)
{
    cout<<"Dati numarul de tari = "; cin>>n;
    for(i=1;i<=n-1;i++)
        for(j=i+1;j<=n;j++)
            {
                cout<<"a["<<i<<"]["<<j<<"]="";
                cin>>a[i][j]; a[j][i]=a[i][j];
            }
    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--;
    }
}
```



```
c:\E:\Universitate\2009-2010\Semestrul
Dati numarul de tari = 5
a[1][2]=1
a[1][3]=0
a[1][4]=1
a[1][5]=0
a[2][3]=1
a[2][4]=1
a[2][5]=1
a[3][4]=0
a[3][5]=1
a[4][5]=1
tara 1 are culoarea 1
tara 2 are culoarea 2
tara 3 are culoarea 1
tara 4 are culoarea 3
tara 5 are culoarea 4

Terminated with return code 1
Press any key to continue ...
```

Problema 6

Problema comis-voiajorului

Un comis-voiajor trebuie sa viziteze n orase etichetate $1, \dots, n$. Va pleca din orasul 1 si se va intoarce tot in 1, trecand prin fiecare oras (in afara de oraşul 1, celelalte orase trebuie vizitate o singura data).

Cunoscand legaturile existente intre orase, sa se tipareasca toate drumurile posibile.

Rezolvare:

Datele referitoare la drumurile dintre orase vor fi reprezentate intr-o matrice $A_{n \times n}$ cu:

$A_{ij} = 1$ daca drum direct intre orasul i si orasul j

0 nu exista drum direct intre orasul i si orasul j

$A_{ii} = 0$

Implementarea solutiei
ramane ca tema!

Problema 6

Vectorul solutiilor rezultat va fi $x=(x_1, \dots, x_n, x_{n+1})$ cu $x_1=x_{n+1}=1$ si cu semnificatia:

pe poz. i ($i=1, n+1$) se viziteaza orasul x care apartine S .

La pasul k ($2 \leq k \leq n$) se incearca vizitarea orasului $x[k]+1$ daca:

(1) $1 \leq x[k] < n$ (initial $x[k]=1$)

si $a[x[k]-1, x[k]+1]=1$ (pt. $k=n$ trebuie si $a[1, x[k]+1]=1$)

Daca (1) este adevarata atunci $x[k] < x[k]+1$ si trebuie sa fie satisfacute conditiile interne:

(2) $x[k] \neq x[i]$ cu $i=1, \dots, k-1$

Daca (1) si (2) sunt adevarate se poate trece la vizitarea orasului $k+1$.

In caz contrar cu orasul de pe pozitia x_k nu ajungem la o solutie rezultat si va trebui sa ne intoarcem la o alta alegere pentru orasul de pe pozitia $k-1$.

Conținutul cursului

11.1. Exemple de probleme rezolvate cu ajutorul metodei backtracking

11.2. Metoda backtracking – varianta recursiva

11.3. Backtracking generalizat in plan

11.2. Backtracking recursiv

Funcțiile folosite sunt în general aceleși, cu două mici excepții:

1. În funcția **SOLUTIE** condiția este $n+1$;
2. rutina **backtracking** se transformă în funcție, care se apelează prin **BACK(1)**

Principiul de funcționare al funcției **BACK**, corespunzător unui nivel k este următorul:

- în situația în care avem o soluție, o tipărim și revenim pe nivelul anterior
- în caz contrar se initializează nivelul și se caută un succesori
- când am găsit unul verificăm dacă este valid; funcția se autoapelează pentru $(k+1)$, în caz contrar urmând să se continue căutarea succesoriului;
- dacă nu avem succesori, se trece pe nivel inferior $(k-1)$ prin ieșirea din funcția **BACK**

Implementarea algoritmului pentru generarea permutarilor –
varianta recursiva:

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

```
int valid(int k,int st[ ])
{
    ev=1;
    for (int i=1;i<=k-1;i++)
        if (st[i]==st[k]) ev=0;
    return ev;
}
int solutie(int k)
{
    if ( k==n+1 ) return 1;
    else return 0;
}
void tipar(void)
{
    for(int i=1;i<=n;i++)
        cout<<" "<<st[i];
    cout<<"\n";
}
```

```
void back(int k)
{
    if(solutie(k)) tipar();
    else{
        init(k,st);
        while(succesor(k,st))
        {
            ev=valid(k,st);
            if(ev) back(k+1);
        }
    }
}

int main(void)
{
    cout<<"Dati n = "; cin>>n;
    back(1);
}
```

Conținutul cursului

11.1. Exemple de probleme rezolvate cu ajutorul metodei backtracking

11.2. Metoda backtracking – varianta recursiva

11.3. Backtracking generalizat in plan

11.3. Backtracking generalizat in plan

Backtraking generalizat în plan

Metoda Backtracking în plan are câteva modificări:

- stiva conține mai multe coloane (este dublă, triplă, ...);
- trebuiesc codificate oarecum direcțiile prin numere, litere, elemente, etc.

Problema labirintului se poate rezolva după un algoritm de backtracking generalizat în plan.

11.3. Backtracking generalizat in plan

Problema labirintului

Se dă un labirint sub formă de matrice de m linii și n coloane.

Fiecare element al matricii reprezintă o cameră. Într-una din camerele labirintului se găsește un om.

Se cere să se afle toate soluțiile ca acel om să iasă din labirint, fără să treacă de două ori prin aceeași cameră.

11.3. Backtracking generalizat in plan

Generalizare

- Această variantă a problemei este varianta în care fiecare cameră are pereții proprii în părțile laterale.
- Există o altă variantă în care fiecare element al matricii este fie un culoar, fie un perete, putându-se trece doar dintr-un culoar în altul.
- Aici, se poate trece dintr-o cameră în alta, doar dacă între cele două camere nu există perete (camerele sunt imediat apropiate).
- Prin labirint, putem trece dintr-o cameră în alta doar mergând în sus, în jos, la stânga sau la dreapta, nu și în diagonală.

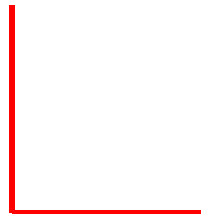
11.3. Backtracking generalizat in plan

Codificare

- Principiul backtracking generalizat spune că trebuie codificate direcțiile.
- În aceste caz vor fi codificate și combinațiile de pereți ai fiecărei camere.
- Astfel, un element al camerei va fi un element al unei matrici cu n linii și n coloane, având valori de la 0 la 15.

- În sistemul binar, numerele 0..15 sunt reprezentate ca 0..1111, fiind memorate pe 4 biți consecutivi.
- Vom lua în considerare toți cei 4 biți, astfel numerele vor fi 0000..1111.
- Fiecare din cei 4 biți reprezintă o direcție, iar valoarea lui ne spune dacă în acea direcție a camerei există sau nu un perete.
- Vom reprezenta numărul astfel:
$$nr = b1 \ b2 \ b3 \ b4 \quad (b = \text{bit})$$
- Astfel, $b1$ indică direcția nord (**sus**), $b2$ indică direcția est (**dreapta**), $b3$ indică direcția sud (**jos**) iar $b4$ indică direcția vest (**stânga**).

- Valorile unui bit sunt, firește, **0** și **1**.
0 înseamnă că în direcția respectivă *nu există un perete*,
iar **1** înseamnă că în direcția respectivă *există un perete și pe acolo nu se poate trece*.
- De exemplu: **0111** - camera aceasta are pereți în dreapta, în jos și în stânga, iar în sus este drum liber spre camera vecină.



- Acest număr este de fapt 7, așa fiind notat în matricea labirintului.

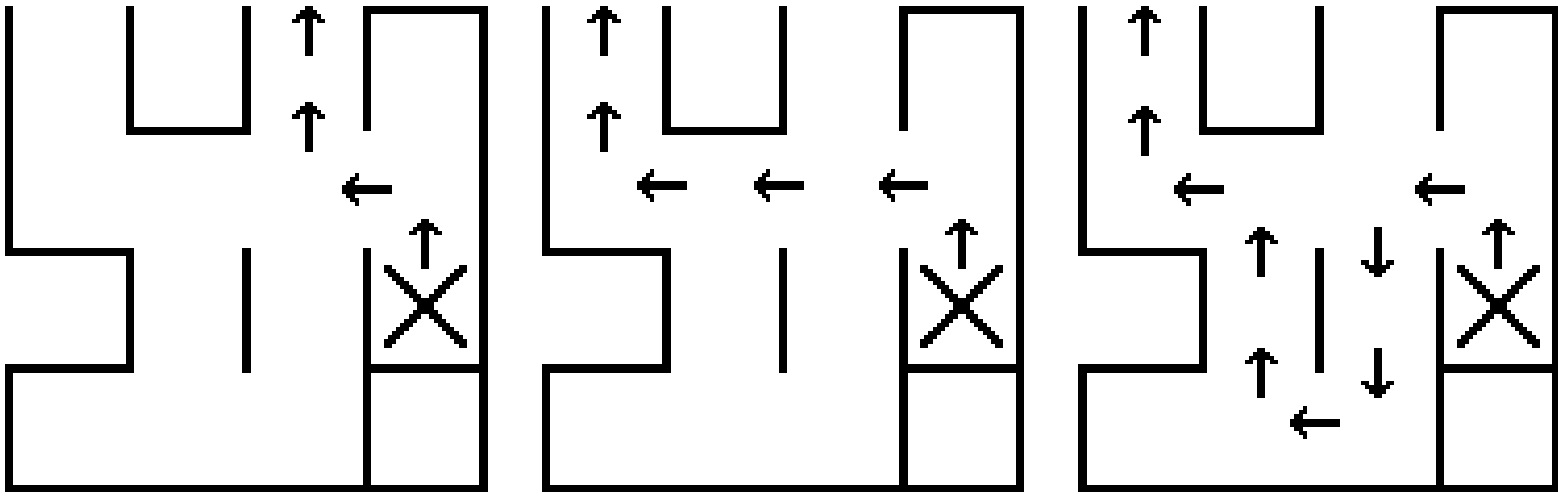
- În ceea ce privește direcțiile, vom reține doar coordonatele unde se află omul din labirint, acestea fiind schimbate în funcție de drumul pe care-l urmează.
- Vom avea o **stivă triplă** astfel:
 - **coloana 1 va indica direcția**. Aceasta va fi de la 1 la 4, 1 reprezentând sus, 2 reprezentând dreapta, 3 reprezentând jos, iar 4, stânga.
 - **coloana 2 va reține indicele de linie** al camerelor parcurse în labirint
 - **coloane 3 va reține indicele de coloană** al camerelor parcurse în labirint
 - fiecare linie va însemna o cameră, cu indicele de linie și de coloană

11.3. Backtracking generalizat in plan

Astfel:

- dacă direcția este 1 (sus), linia se va micșora cu 1,
- dacă direcția este 2 (dreapta), coloana se va mări cu 1,
- dacă direcția este 3 (jos), linia se va mări cu 1,
- dacă direcția este 4 (stânga), coloana se va micșora cu 1.

- Punctul de pornire din acest labirint este **linia 3, coloana 4**.
- Astfel, avea 3 soluții de a ieși din labirint, fără a trece de două ori prin aceeași cameră:
- $(3,4)-(2,4)-(2,3)-(1,3)$ -ieșire
- $(3,4)-(2,4)-(2,3)-(2,2)-(2,1)-(1,1)$ -ieșire
- $(3,4)-(2,4)-(2,3)-(3,3)-(4,3)-(4,2)-(3,2)-(2,2)-(2,1)-(1,1)$ -ieșire



11.3. Backtracking generalizat in plan

Rezolvare

- După metoda backtracking, trebuie să găsim toate posibilitățile de a ieși din labirint.
- S-a ieșit din labirint când linia este 0, coloana este 0, linia este $m+1$ sau când coloana este $n+1$.
- Trebuie să trecem din cameră în cameră, să verificăm toate posibilitățile de ieșire.

11.3. Backtracking generalizat in plan

- Când trecem dintr-o cameră în alta, reținem în matricea triplă, direcția, linia și coloana respectivă.
- Va trebui să respectăm regulile preblemei, adică să nu trecem de două ori prin aceeași cameră.
- Această regulă a fost impusă deoarece dacă am rezolva problema fără ea, atunci ne-am învârti în gol !
- Această verificare se face comparând coordonatele camerei în care suntem cu coordonatele camerelor prin care am trecut, cele reținute de matricea triplă pe coloanele 2 și 3.

- Pentru a verifica dacă între două camere există sau nu un perete, va trebui să respectăm un lucru: dacă o cameră are perete în sus, atunci obligatoriu și camera de sus trebuie să aibă perete jos, iar dacă o cameră nu are perete sus, atunci nici camera de sus nu trebuie să aibă perete jos.
- Aceasta deoarece dacă ar fi așa, am trece dintr-o cameră în alta, dar invers nu, ceea ce este imposibil.
- Imaginați-vă că sunteți într-o cameră. Vreți să ieșiți în hol. Din cameră se vede holul și ușa, dar după ce ieșiți în hol și vreți să reveniți în cameră, vă întoarceți și constatați că în spatele vostru nu există nici o ușă pentru a intra înapoi în cameră, în locul ei fiind doar un perete.

Astfel vom verifica după relațiile:

- $a[\text{st}[k][2], \text{st}[k][3]] \&\& 8 = 0$ - nu putem merge în sus
- $a[\text{st}[k][2], \text{st}[k][3]] \&\& 4 = 0$ - nu putem merge la dreapta
- $a[\text{st}[k][2], \text{st}[k][3]] \&\& 2 = 0$ - nu putem merge în jos
- $a[\text{st}[k][2], \text{st}[k][3]] \&\& 1 = 0$ - nu putem merge la stânga

A – matricea labirintului

ST – stiva triplă

- Operatorul logic $\&\&$ verifică biții corespunzători ai 2 numere întregi, returnând 1 dacă ambii sunt 1 și 0 în rest.
- Acesta este rezultatul final. 8 înseamnă 1000, adică dacă camera respectivă are perete sus, rezultatul va fi tot 8, dacă nu, va fi 0.

- Când mergem prin labirint, înaintăm în camera respectivă iar apoi verificăm dacă între cele două camere există sau nu perete pentru a putea trece pe acolo.
- Apoi verificăm să nu fi ieșit din labirint ($l1 < 1 \parallel l1 > m \parallel c1 < 1 \parallel c1 > n$) și dacă prin această cameră am mai trecut odată.

Afișare

- Când am găsit o soluție, afișăm pe rând toate liniile din matricea triplă, dar fără coloana 1, ci numai coloanele 2 și 3.

Întrebări?