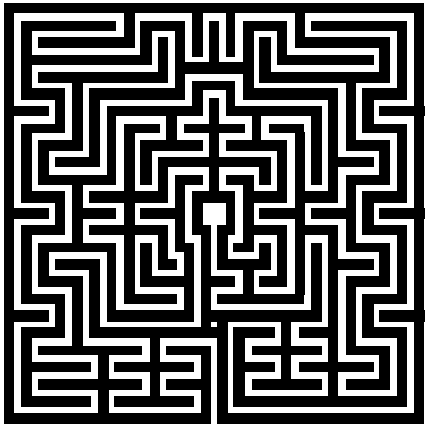
Tehnici de programare

Aplicații ale recursivității

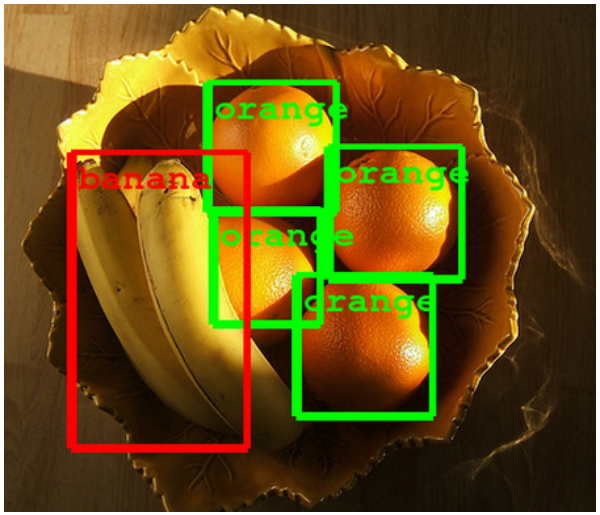
ovidiu.banias@aut.upt.ro

Recursivitate

Backtracking Recursiv



Iesire din labirint



Procesare de imagini

Backtracking recursiv

Recursiv

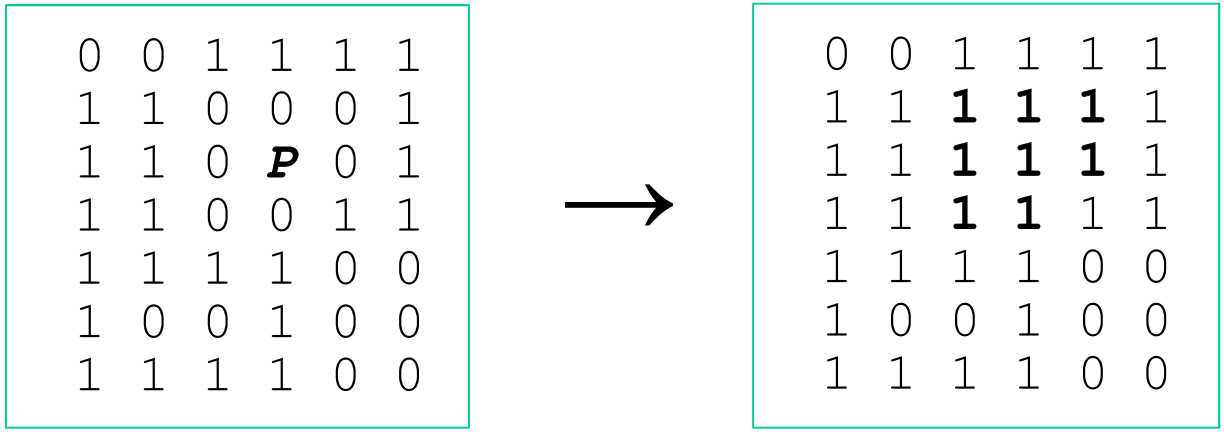
```
void Back(int k){ // B. recursiv

  if (Solution(k))
     Print();
  else{
     Init(k);
     while (Successor(k))
        if Valid(k)
           Back(k+1);
   }
}
```

Iterativ

```
void Back(int n){ // B. interativ

 k=1; Init(k);

 while (k>0){
  isS=0;isV=0;
  if (k<=n)
   do{
    isS=Successor(k);
    if (isS) isV=Valid(k);
   }while (isS && !isV);
   if (isS)
    if (Solution(k))
     Print();
    else{
     k++;
     Init(k);
    }
   else
    k--;
  }
}
```

Recursivitate. Tehnica FILL

Problemă: Se dă o matrice binară şi un punct **P** în matrice de coordonate (i,j). Valorile **1** delimitează o suprafață închisă, iar valorile **0** se află fie în interiorul suprafeței închise, fie în exteriorul ei. Ştiind că punctul **P** se află în interiorul unei suprafețe închise (împrejmuită de valori **1**), să se umple toată suprafața cu **1**.



Recursivitate. Tehnica FILL

Observaţii:

- Pornind dintr-un punct **P(i,j)**, se pot face maxim 4 deplasări :

  Sus: **P(i-1,j)**,

  Jos: **P(i+1,j)**,

  Stânga: **P(i,j-1)**,

  Dreapta: **P(i, j+1)**.

- Face sens deplasarea doar într-un punct cu valoarea 0
- Umplerea suprafeței se poate face numai în paşi succesivi
- Umplerea punctelor adiacente lui P se face recursiv

Recursivitate. Tehnica FILL

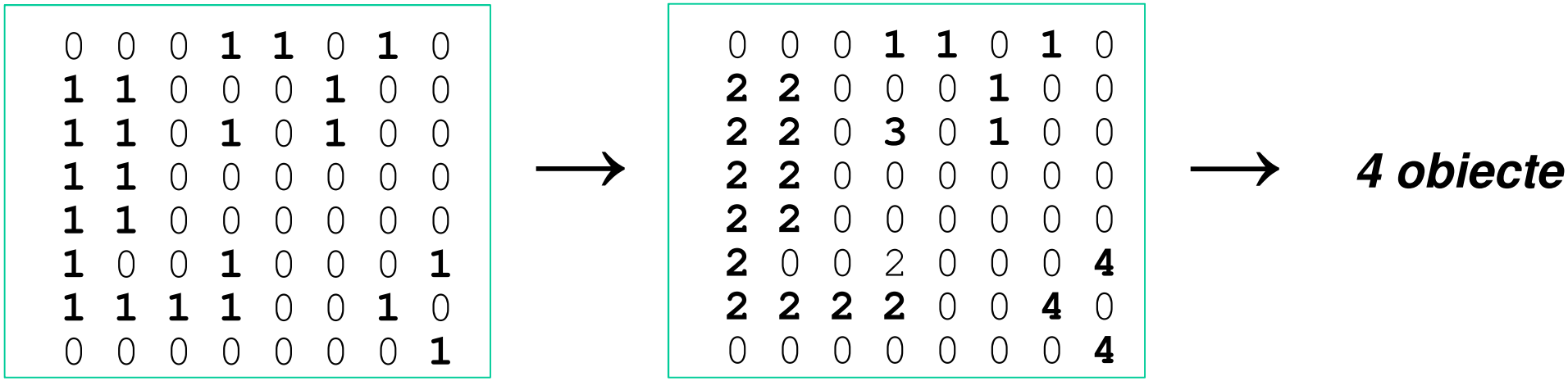
```
void FILL(int i, int j){ // FILL/umplere recursiva

     if (!a[i][j]){
          a[i][j]=1; //umple

          FILL(i-1,j); //sus
          FILL(i,j+1); //dreapta
          FILL(i+1,j); //jos
          FILL(i,j-1); //stanga
     }
}
```

- Poate fi utilă bordarea matricii cu valori 1 – pentru a nu ieşii din matrice
- Perechi (i,j) reprezentând punctul curent expandabil, sunt puse/scoase din stivă

Recursivitate. Problema fotografiei

Problemă: Se dă o fotografie alb-negru reprezentată printr-o matrice binară. Se cere numărarea obiectelor din fotografie, ştiind că un obiect este format din valori de 1 adiacente (8 poziții de adiacență N,NE,E,SE,S,SW,W,NW).

NW	N	NE
W	·	E
SW	S	SE



Recursivitate. Problema fotografiei

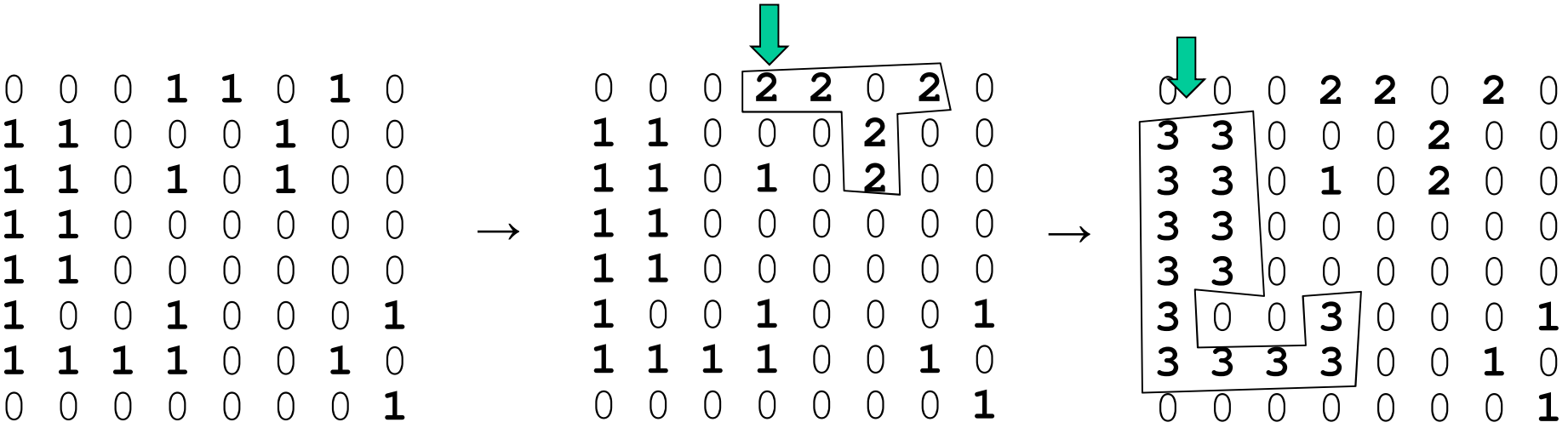
Rezolvare:

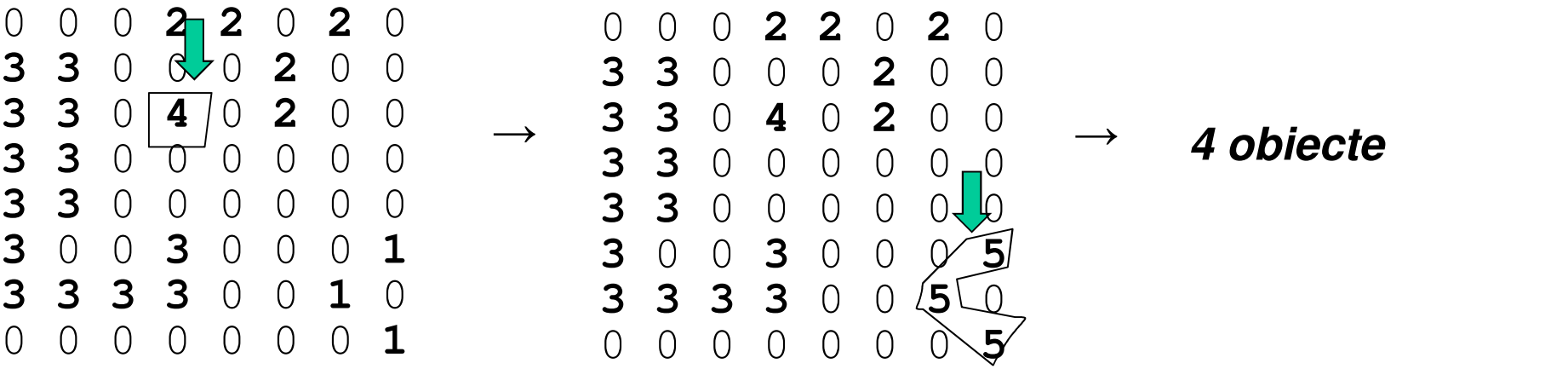
- Se parcurge matricea până se găseşte un element cu valoarea 1 ce nu aparține nici unui obiect (este nemarcat)
- Pornind de la elementul găsit se expandează căutarea pe cele 8 direcții – similar metodei FILL şi se marchează elementele cu valoarea 1
- Înainte de a repeta algoritmul, toate elementele marcate la pasul anterior se consideră a fi componente unui singur obiect; se repetă primul pas

NW	N	NE
W	·	E
SW	S	SE

Recursivitate. Problema fotografiei

NW	N	NE
W	·	E
SW	S	SE





4 obiecte

Recursivitate. Problema fotografiei

```
int x[8]={-1,-1,-1,0,1,1,1,0};
int y[8]={-1,0,1,1,1,0,-1,-1};
.
.
.
void FillObj(int i, int j){ //marcare recursiva
    int k; //local !

    if (a[i][j]==1){
        a[i][j]=nrObj; //numarul curent al obiectului
        for (k=0;k<8;k++) //cele 8 directii
           FillObj(i+x[k],j+y[k]);
    }
}

int main(){
{
  ...
  nrObj++;
  FillObj(i,j);
  ...
}
```

Recursivitate. Ieșiere din labirint

Problemă: Se dă un labirint reprezentat printr-o matrice binară n x m. A[i][j]={0,1}|0<=i<n, 0<=j<m. A[i][j]=1 – la punctul de coordinate i şi j se află un zid, A[i][j]=0 – la punctul de coordinate i şi j nu există zid. Având în vedere că o cameră este reprezentată de un element în matrice să se afişeze toate posibilitățile de ieşire din labirint pornind dintr-un punct inițiat de coordonate (x,y).

0	0	0	**1**	**1**	0	**1**	0
1	**1**	0	0	0	**1**	0	0
1	**1**	0	**1**	0	**1**	0	0
1	**1**	0	0	0	0	0	0
1	**1**	0	0	0	0	0	0
1	0	**P**	**1**	0	0	0	**1**
1	**1**	**1**	**1**	0	0	**1**	0
0	0	0	0	0	0	0	**1**

- Câte posibilități de ieşire din labirint?
- Cum implementăm?
- Care e diferența majoră între algoritmul de ieşire din labirint şi algoritmul FILL?

Recursivitate. Labirint

```
void Search(int i, int j){

    int k;

    if (i==0 || i==n-1 || j==0 || j==m-1){
        Print(i,j);
    }else

    for (k=0;k<8;k++){
        if (a[i+dx[k]][j+dy[k]]==0){// se poate
            a[i+dx[k]][j+dy[k]]=a[i][j]+1;
            Search(i+dx[k],j+dy[k]);
            a[i+dx[k]][j+dy[k]]=0; // !!!
        }
    }
}
```