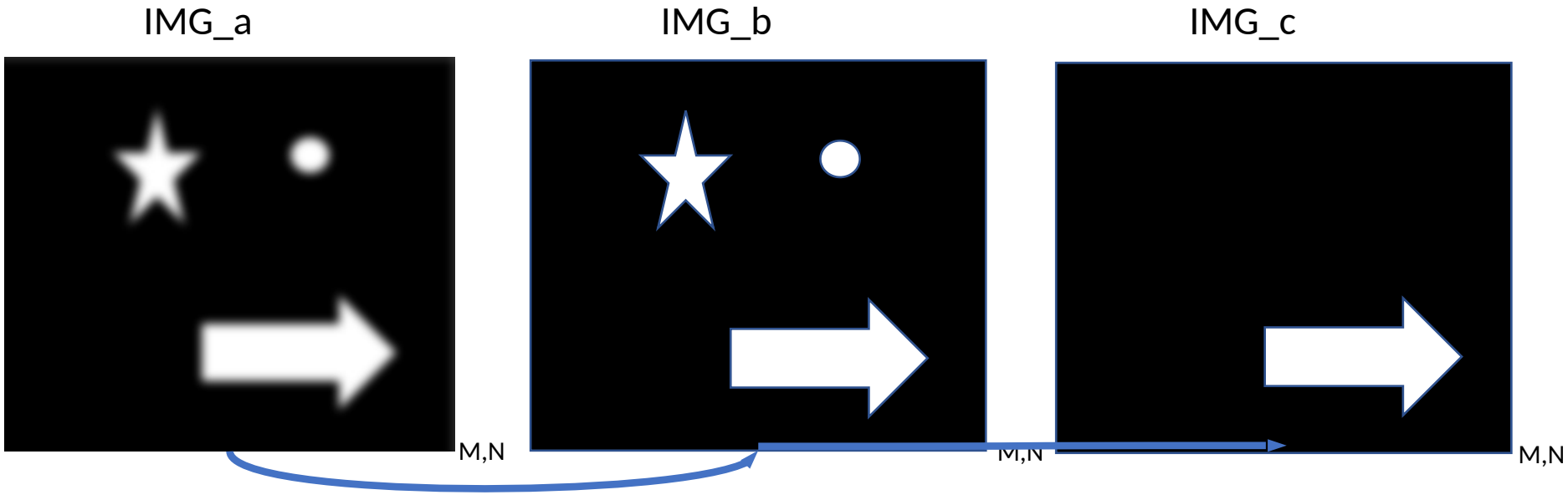
Vizinhança/ Conectividade Componentes conexos

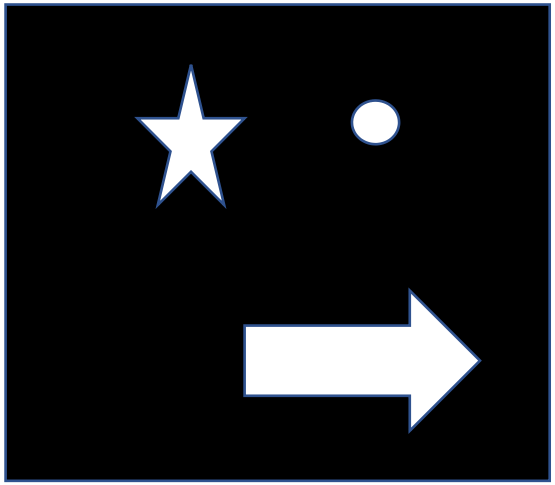


“limiarização” rotulação/seleção por área

Dada uma imagem binária (exemplo: IMG_b), deseja-se selecionar certo objeto de interesse pela sua área relativa (exemplo: IMG_c).

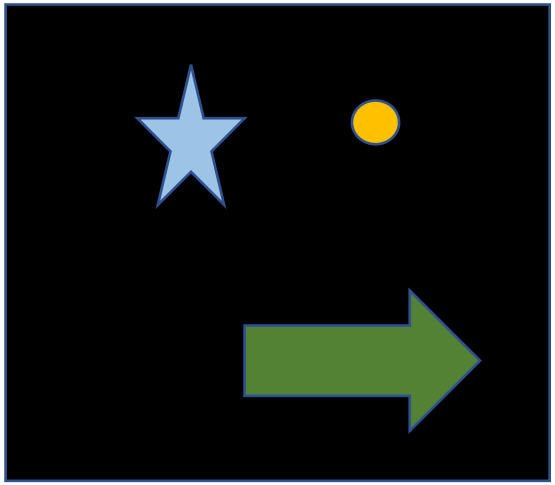
Solução: rotulação de pixels conectados, contabilização da área em pixels do objeto de interesse e seleção do rótulo conveniente.

IMG_b



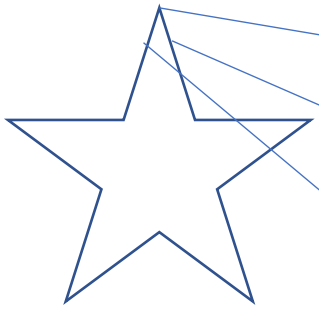
M,N

rotulação

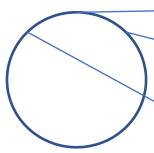


M,N

Atenção: As cores aqui estão sendo usadas apenas para ilustrar rótulos diferentes (exemplo ao lado). Não são informações RGB.

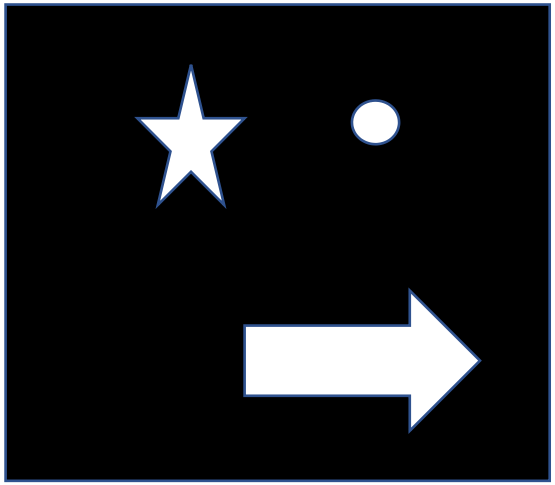


10
10 10 10
10 10 10 10 10
10 10 10 10 10 10
10 10 10 10 10 10 10

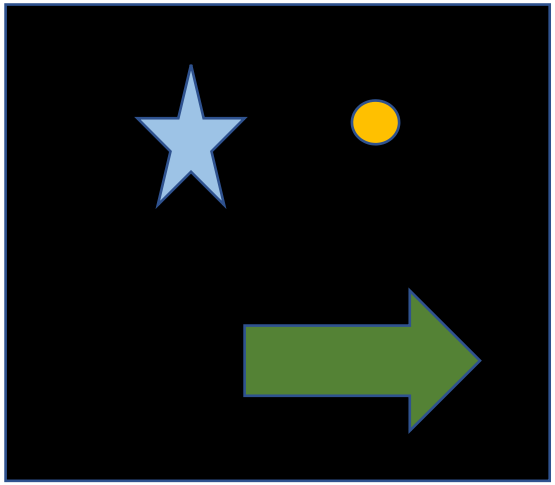


20 20
20 20 20 20 20 20
20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20

IMG_b

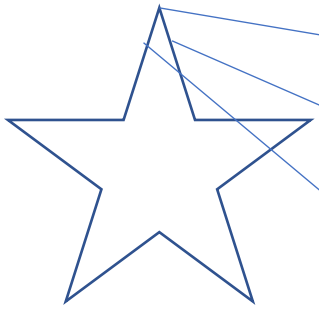


M,N

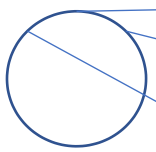


M,N

Atenção: As corres aqui estão sendo usadas apenas para ilustrar rótulos diferentes (exemplo ao lado). Não são informações RGB.

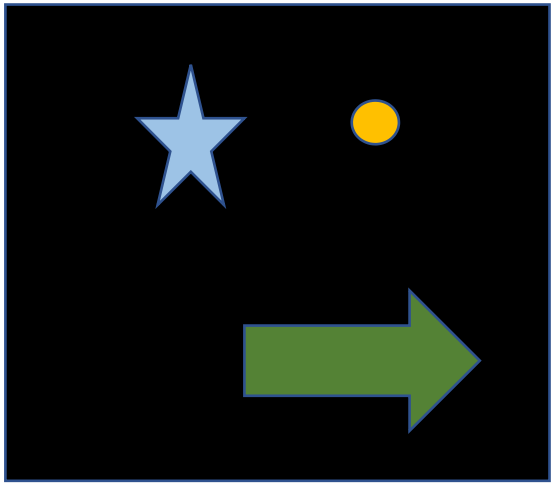


10
10 10 10
10 10 10 10 10
10 10 10 10 10 10
10 10 10 10 10 10 10



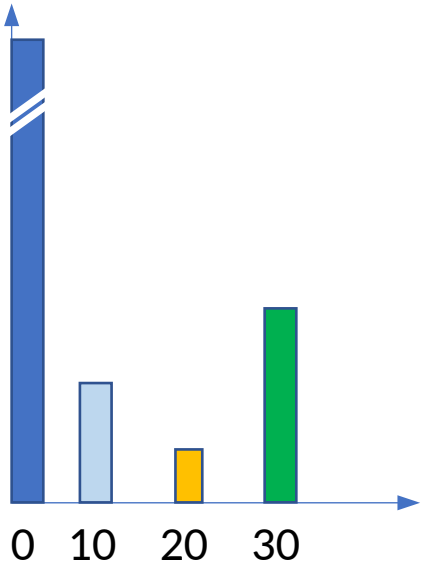
20 20
20 20 20 20 20 20
20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20

IMG_b_rotulada

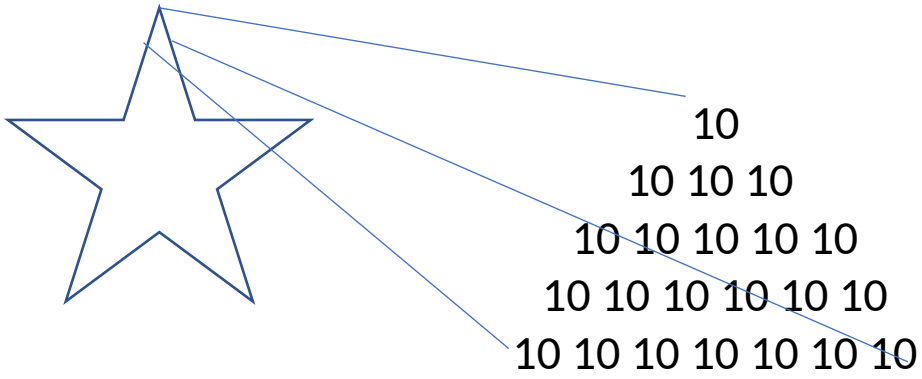


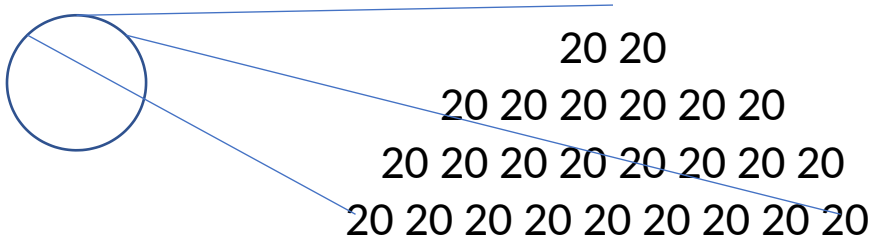
M,N

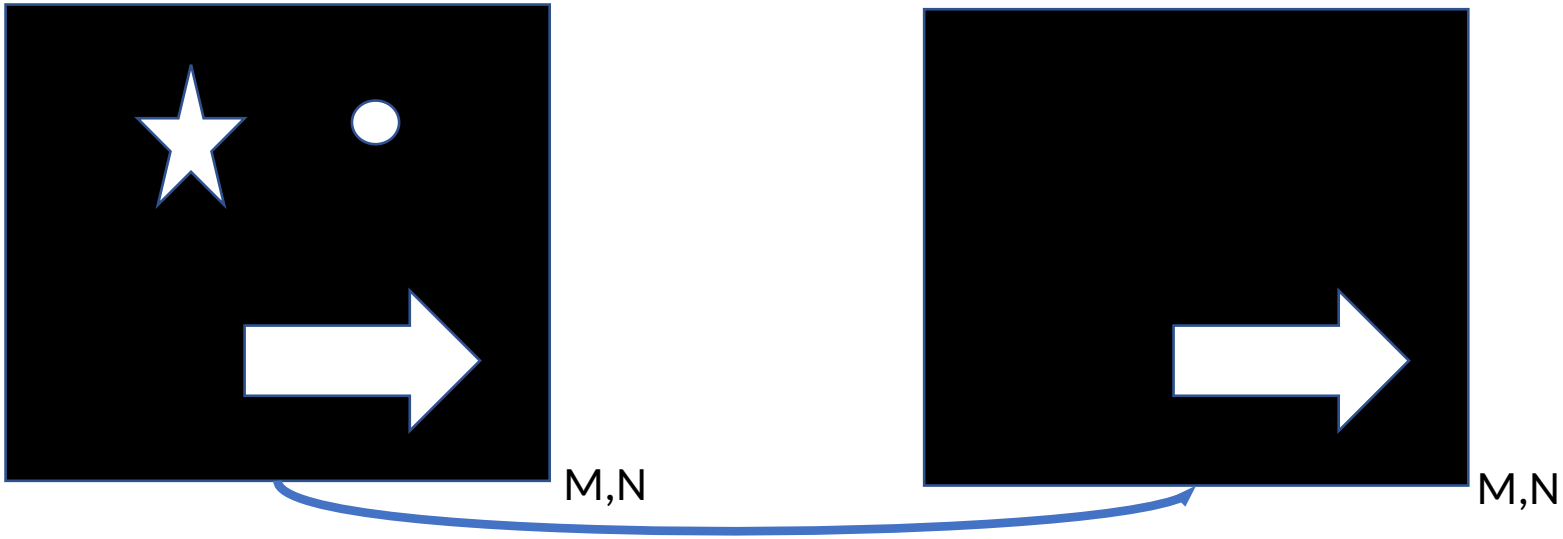
histograma

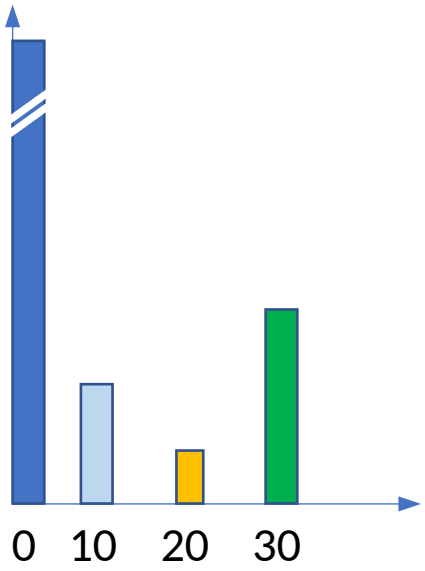


Atenção: As cores aqui estão sendo usadas apenas para ilustrar rótulos diferentes (exemplo ao lado). Não são informações RGB.

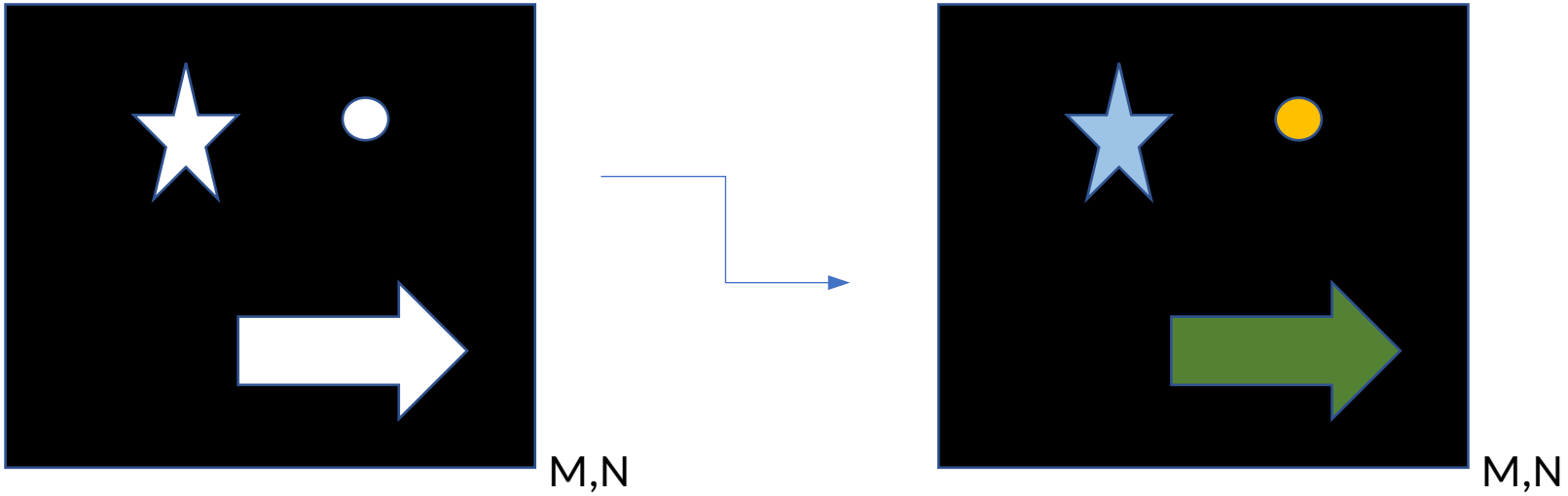








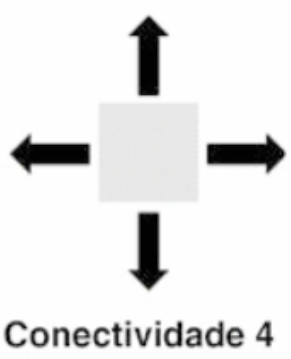
Seleção de pixels com base no rótulo e área do objeto de interesse.



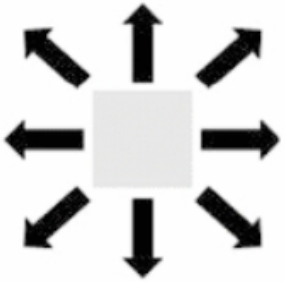
Para executar a rotulação é preciso:

1. Entender o conceito de conectividade de pixels;
2. Tratar elementos de borda na imagem.

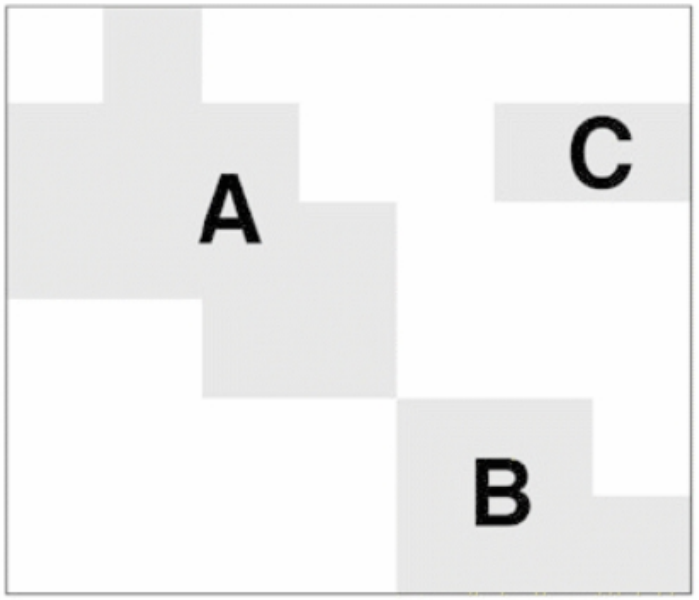
Vizinhança/Conectividade



Conectividade 4



Conectividade 8



Conectividade 4: 3 objetos – A, B e C são distintos
Conectividade 8: 2 objetos –A e B formam um objeto

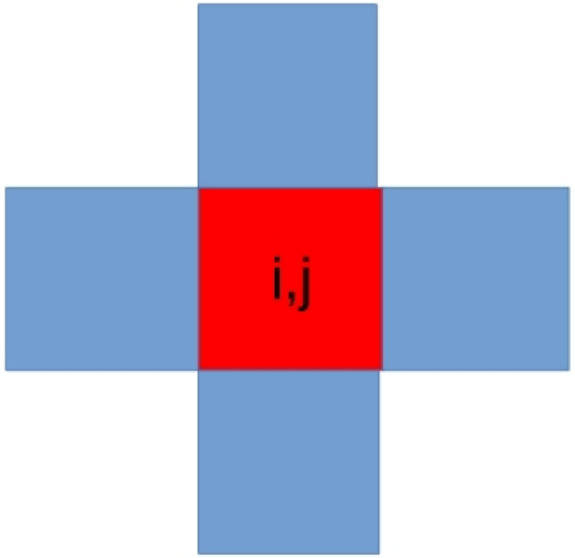
Figura 8.1 A imagem binária ilustrada contém dois objetos (grupos de pixels conectados) de conectividade 8 e três objetos de conectividade 4.

Fonte: Solomon & Breckon, 2016.

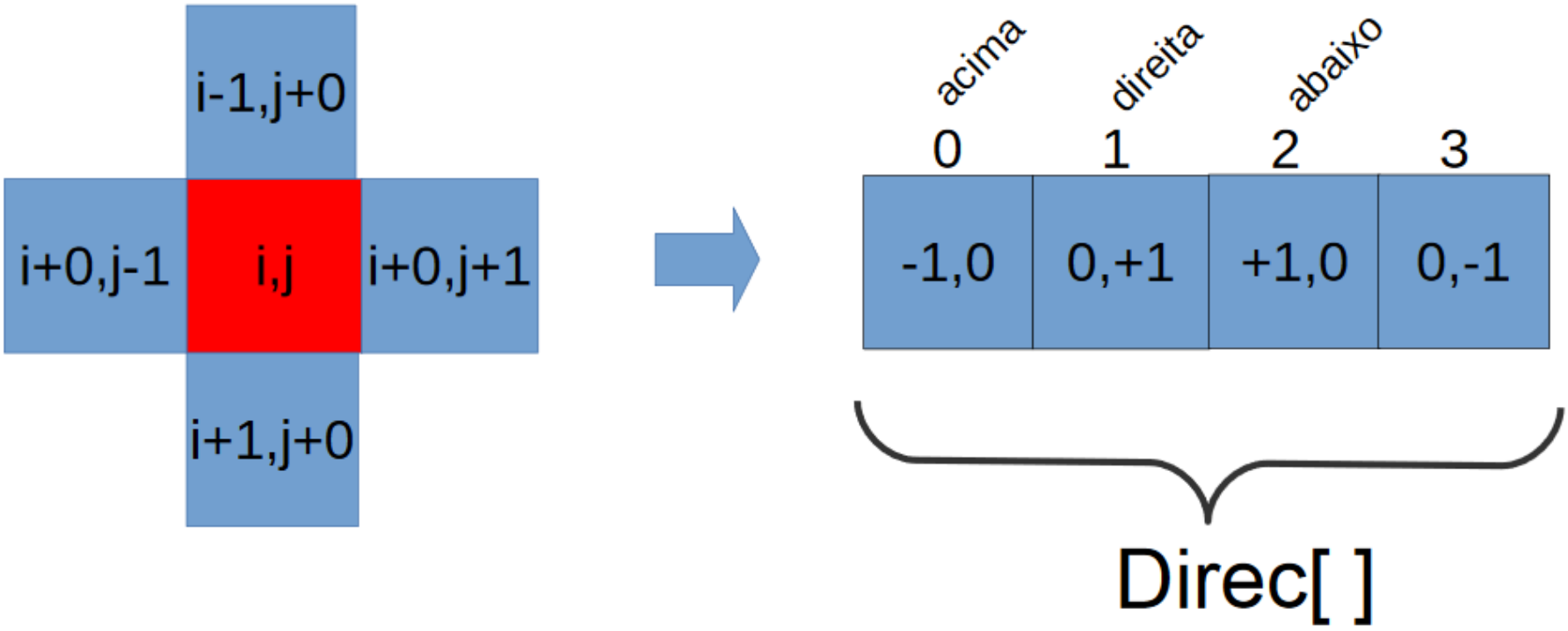
Pixels conexos são vizinhos adjacentes por algum conceito de vizinhança e princípio de similaridade;

Vizinhança/Conectividade

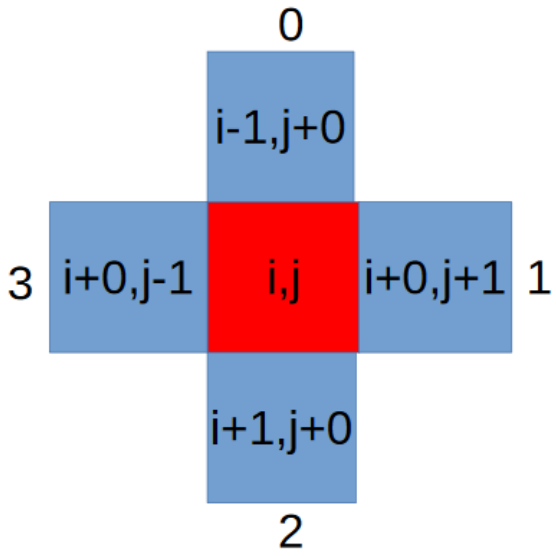
REPRESENTAÇÃO DA VIZINHANÇA-4
Vizinhos Norte, Sul, Oeste e Leste

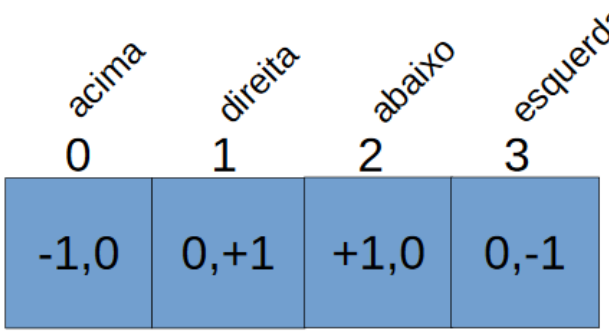


Vizinhança/Conectividade



Vizinhança/Conectividade






```
typedef struct{
    int linha;
    int coluna;
}ponto;
```

Acima de (i,j): Nova_posicao=ponto+direc[acima]
Direita de (i,j): Nova_posicao=ponto+direc[direita]
Abaixo de (i,j): Nova_posicao=ponto+direc[abaixo]
Esquerda de (i,j): Nova_posicao=ponto+direc[esquerda]

Sentido horário

Rotulação

- Pixels de borda: é importante lembrar que os pixels de borda são coordenadas fora da imagem;

- Para controlar essa situação, é comum utilizar uma moldura de 1 pixel de espessura no entorno da imagem;

- São desconsiderados os pixels de coordenadas contidas nessa moldura;

Exercício: implemente uma função que a partir da localização (i,j) do centro de uma janela NxN (N ímpar), retorna as coordenadas dos vizinhos de (i,j). Inicialmente, faça para N=3.

Rotulação recursiva

```
connected_component(IM_binaria)
{   label:= 0;
    maxRow, maxCol = size(IM_binária);
    IM_saida = new zeroed-image(maxRow, maxCol )
    IM_saida = findComponents(IM_binaria , IM_saida, maxRow, maxCol, label);
    display(IM_saida);

}


procedure findComponents(IM_binaria , IM_saida, maxRow, maxCol, label)
{   for R:=0 to maxRow
            for C:= 0 to maxCol
            if (Im_ binaria[R,C] == 1 &&  IM_saída[R,C] == 0)
              {    label:=label+10;
                   labeler(Im_Binaria,IM_saida,label,R,C);
              }

}


procedure labeler(IM_Binaria, IM_Saida, label,R,C)
{   IM_saida[R,C]:=label;
    Nset:= neighbours(R,C);
      for each N in Nset
      {    if(IM_binaria[N.Row,N.Col] == 1 &&  IM_saída[N.Row,N.Col] == 0)
                    labeler(IM_binaria,IM_saida,label,N.Row,N.Col);
      }
}
```

Rotulação não recursiva

Implementações recursivas podem ser limitadas pelo tamanho da pilha de recursão

Em Python você pode incrementar a profundidade de pilha permitida - com isso, serão possíveis chamadas recursivas mais profundas:

```
import sys
sys.setrecursionlimit(10000)
```

Mas o aconselhável é você tentar otimizar seu código, por exemplo, usando iteração ao invés de recursão. É o que esse discute a seguir.

Rotulação não recursiva
Demanda o uso explícito de uma pilha

```
deltas direc[ ]={{-1,0},{0,1},{1,0},{0,-1}};
rot=0
pilha=CriaPILHA();
for i=1;i<=M;i++
    for j=1;j<=N;j++
        if IM[i,j]==1
            marcador(i,j,direc,rot, pilha)
```

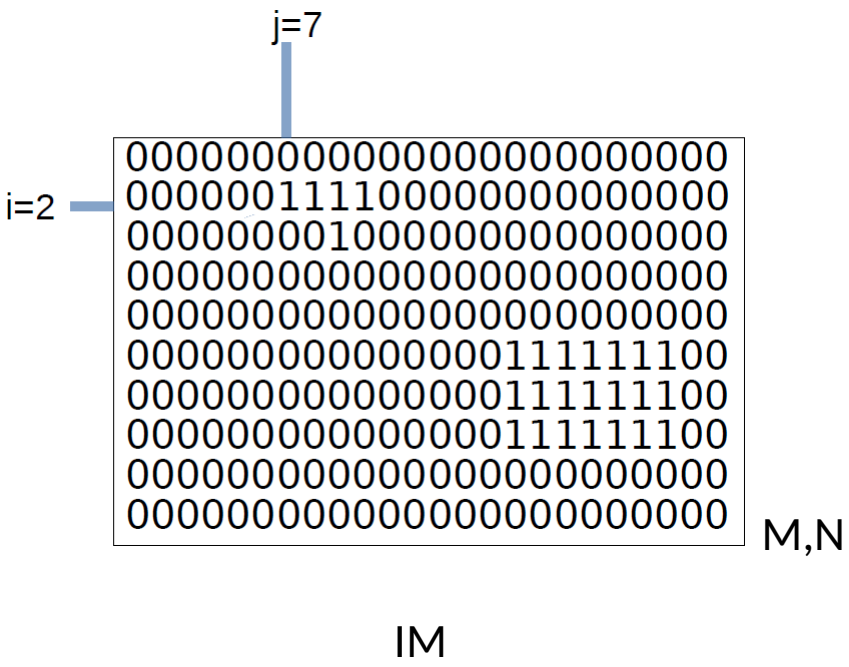
Rotulação não recursiva

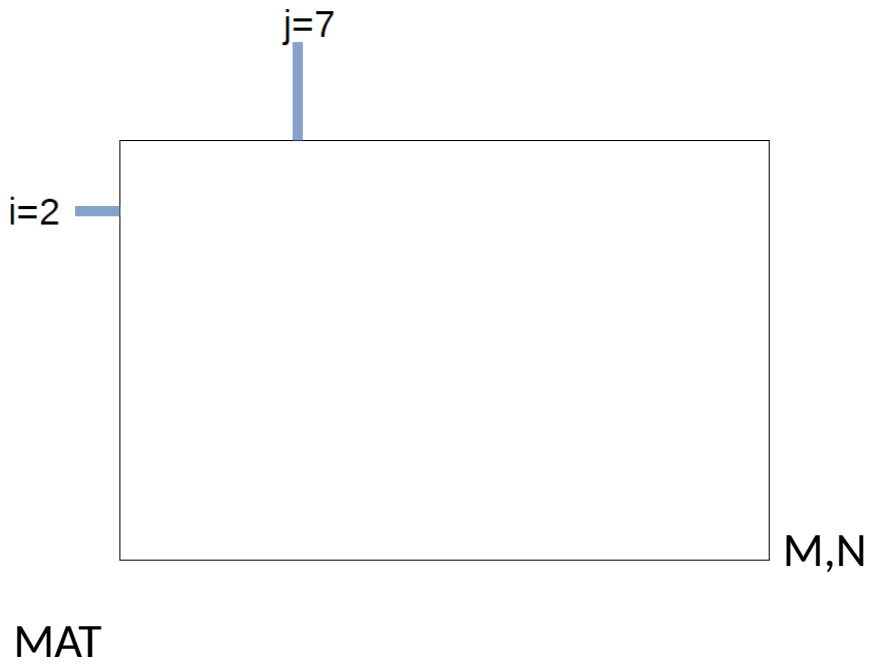
IM = imagem binária
MAT = matriz rotulada

```
marcador(i,j,direc,rot, pilha)
        flag=-1;
        SE (IM[i,j] ==1)
            empilha(ponto(i,j));
            marcaCelula(  pilha, i,j,rot,direc)
            ENQUANTO(PILHA Ñ VAZIA)
                FOR (mov=0;mov <=3;mov++)
                        g=i+direc[mov].deltaLin; h=j+direc[mov].deltaCol;
                        SE(IM[g,h] == 1)
                                empilha(pilha,ponto(g,h));
                                marcaCelula(pilha, i,j,rot,direc);
                                i=g; j=h; flag=1; break;
                SE (flag==-1)
                        (i,j) = desempilha(pilha); //vizinhança toda marcada
marcaCelula(pilha, i,j,rot,direc)
        FOR (mov=0;mov <=3;mov++)
            g=i+direc[mov].deltaLin; h=j+direc[mov].deltaCol;
            SE(mat[g,h] != 0 && mat[g,h] !=1)
                    mat[i,j]=mat[g,h]; //estende o rótulo existente
                    break;
            SE (mat[i,j] ==1) // não há rotulo na vizinhança
                    rot+=20 // novo rótulo
                    mat[i,j]=rot;
```

Rotulação não recursiva

```
deltas direc[ ]={{-1,0},{0,1},{1,0},{0,-1}};
rot=0
pilha=CriaPILHA();
for i=1;i<=M;i++
        for j=1;j<=N;j++
          if IM[i,j]==1
                    marcador(i,j,direc,rot, pilha)
```

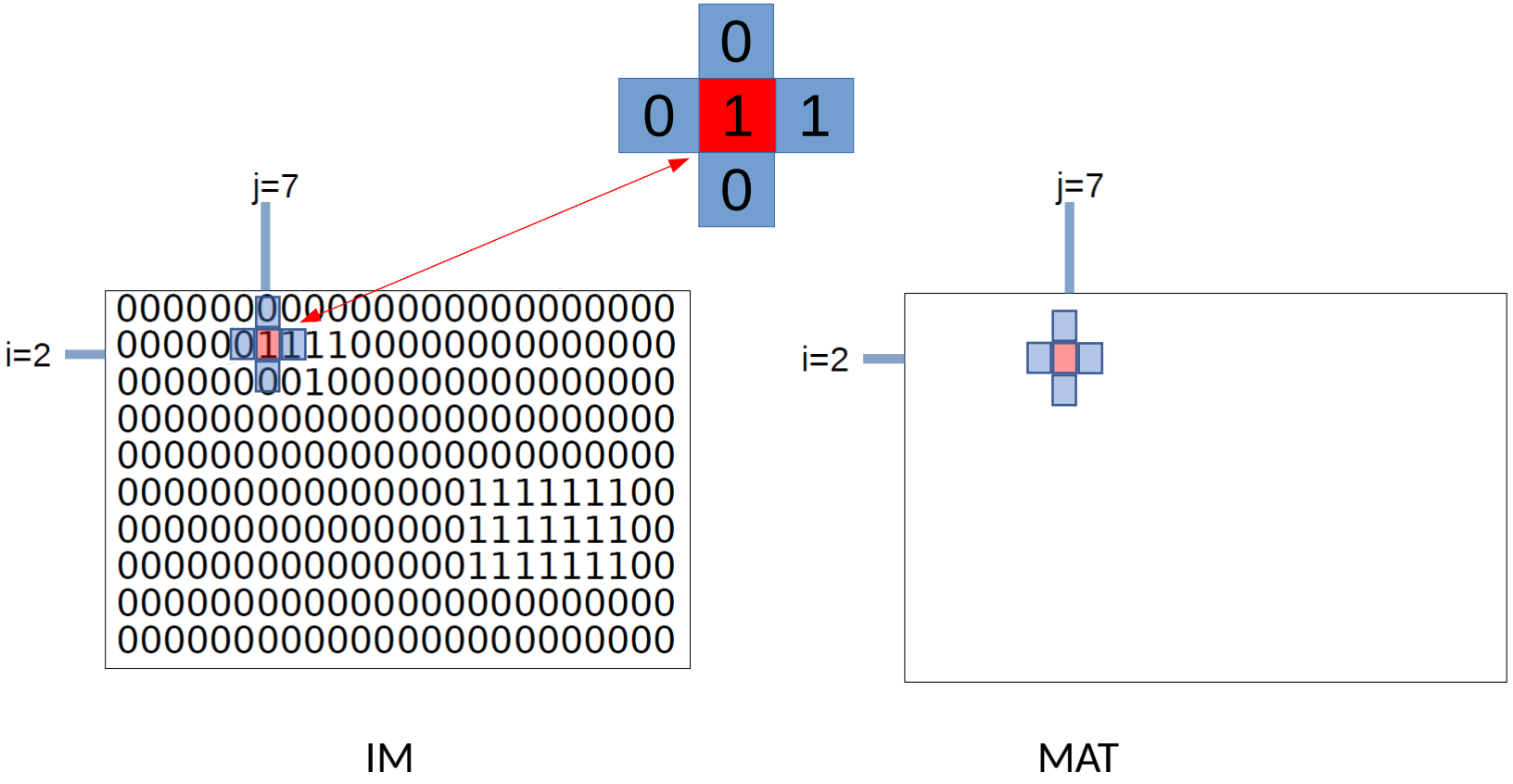




Rotulação não recursiva

Não existe rótulo (valor diferente de zero e 1) na vizinhança de (2,7)

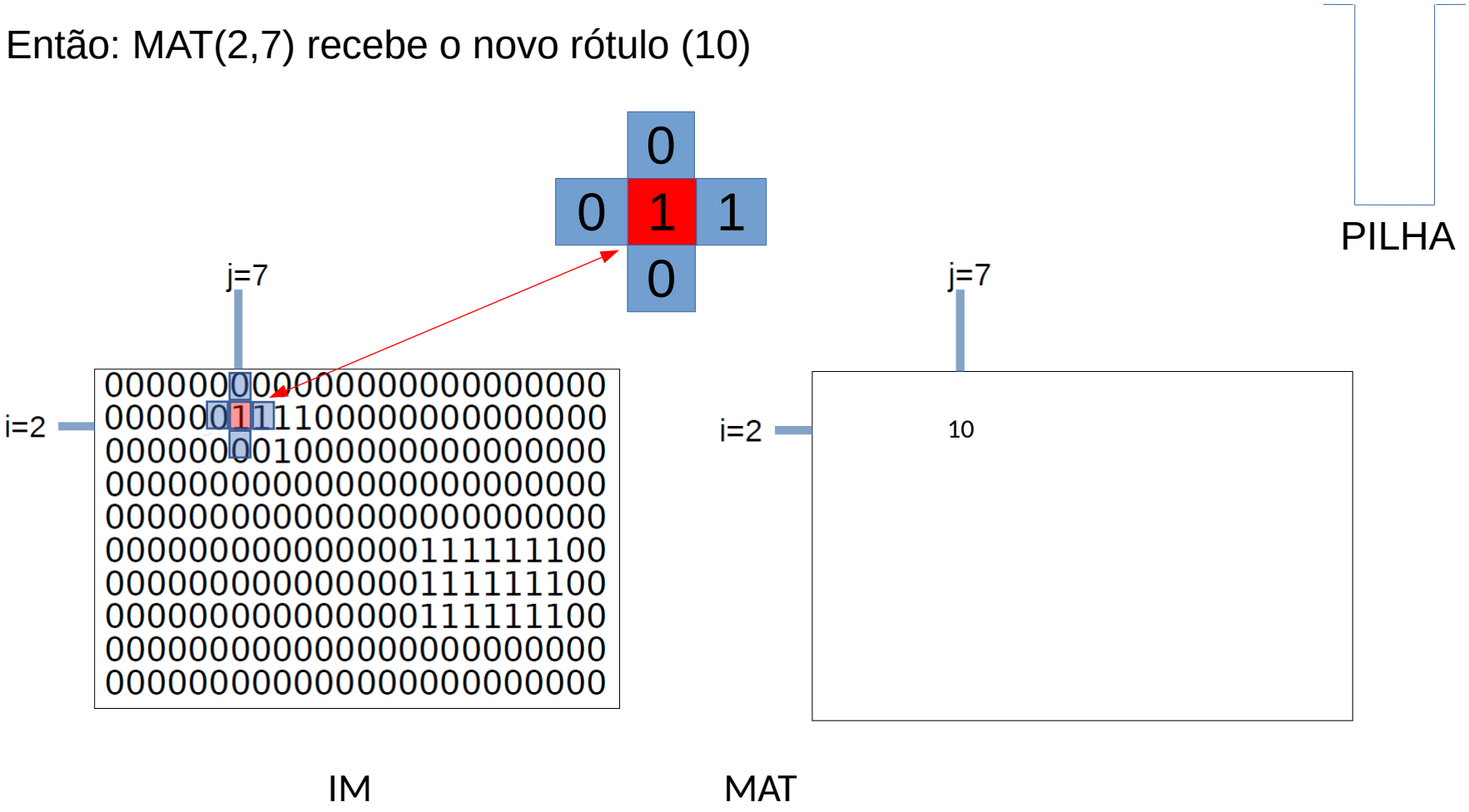
Então: MAT(2,7) recebe o novo rótulo (10)



Rotulação não recursiva

Não existe rótulo (valor diferente de zero e 1) na vizinhança de (2,7)

Então: MAT(2,7) recebe o novo rótulo (10)



Rotulação não recursiva

Uma vez rotulado MAT(2,7), procura-se um vizinho de (2,7) não rotulado e igual a 1;
Este ocorre à direita de (2,7):

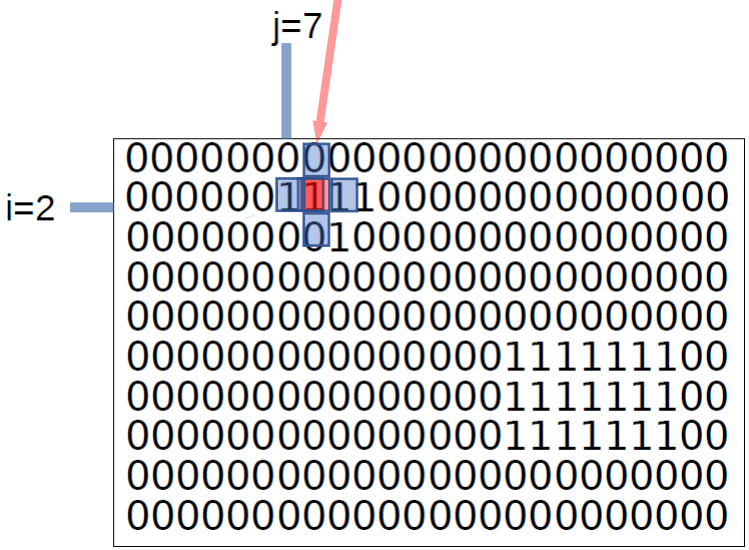
Empilha (2,7)

Atualiza (i,j) para a direita (2,8):

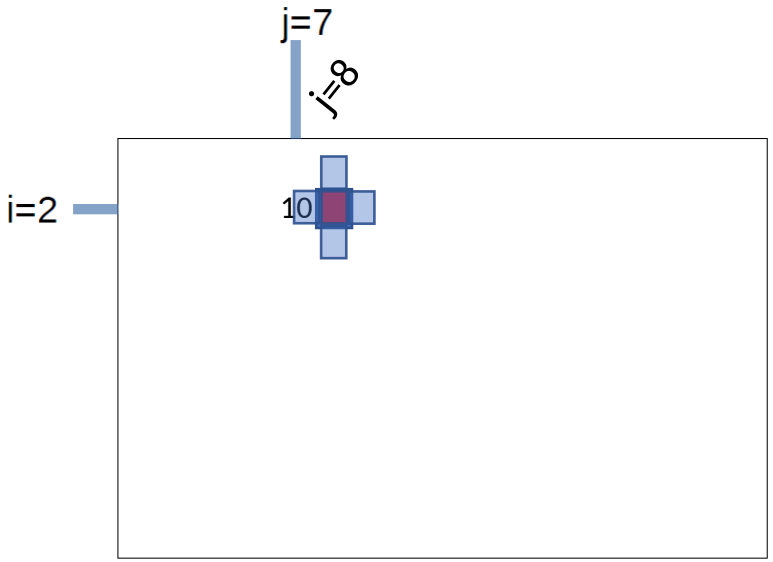
i=i+direc[direita].deltaLin,

j=j+direc[direita].deltaCol

2,7



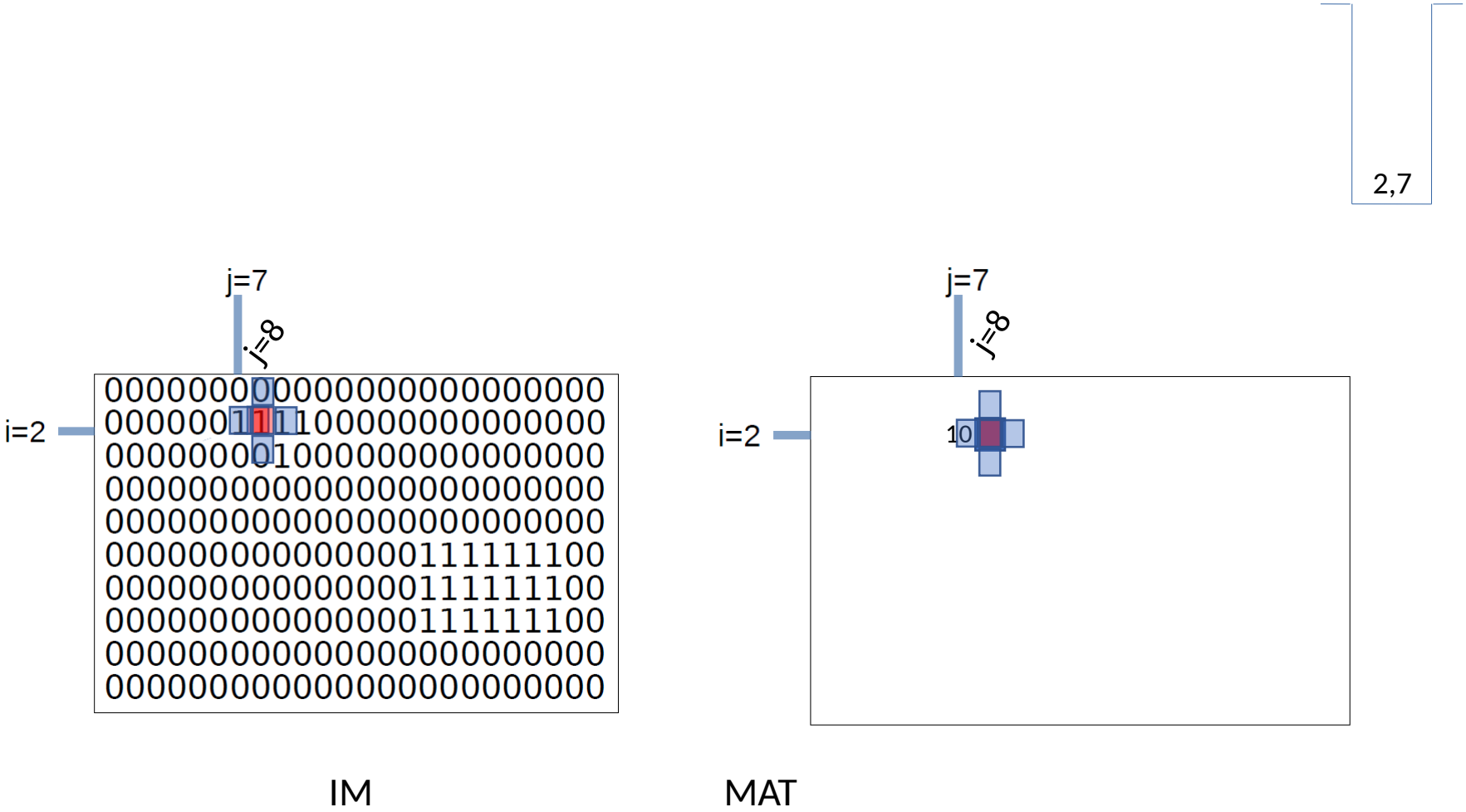
IM



MAT

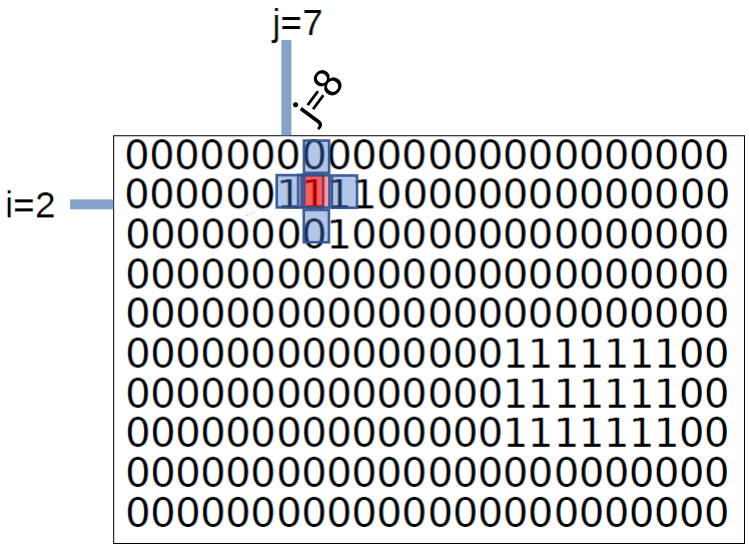
Rotulação não recursiva

Existe rótulo na vizinhança de MAT(2,8)?

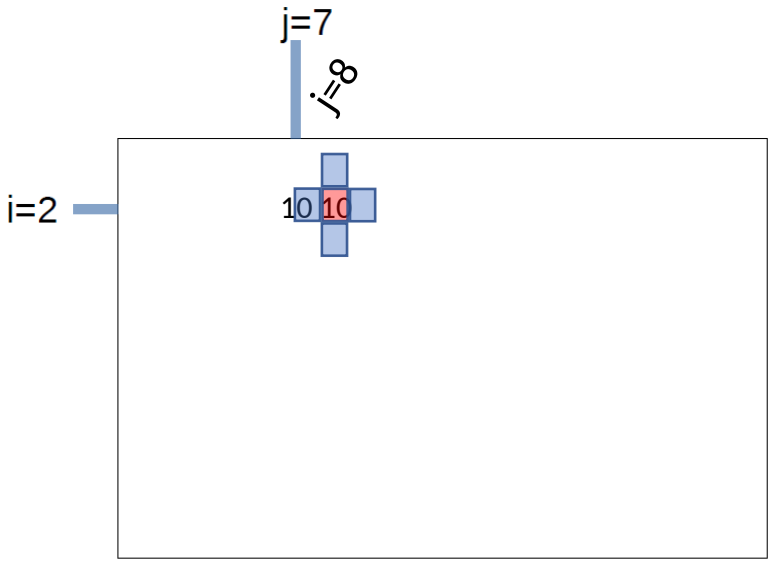


Rotulação não recursiva

Existe rótulo na vizinhança de MAT(2,8)?
SIM: então MAT(2,8) recebe o mesmo rótulo desse vizinho



IM



MAT

2,7

Rotulação não recursiva

Uma vez rotulado MAT(2,8), procura-se um vizinho de (2,8) não rotulado e igual a 1;
Este ocorre à direita de (2,8):

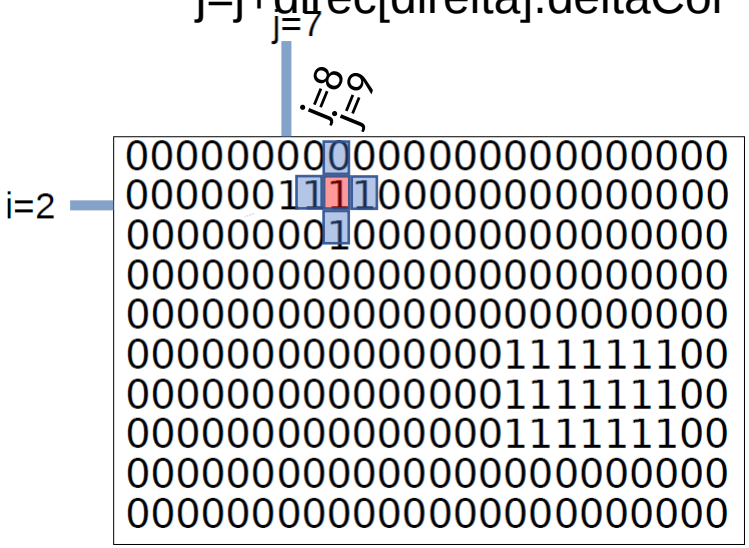
Empilha (2,8)

Atualiza (i,j) para a direita (2,9):

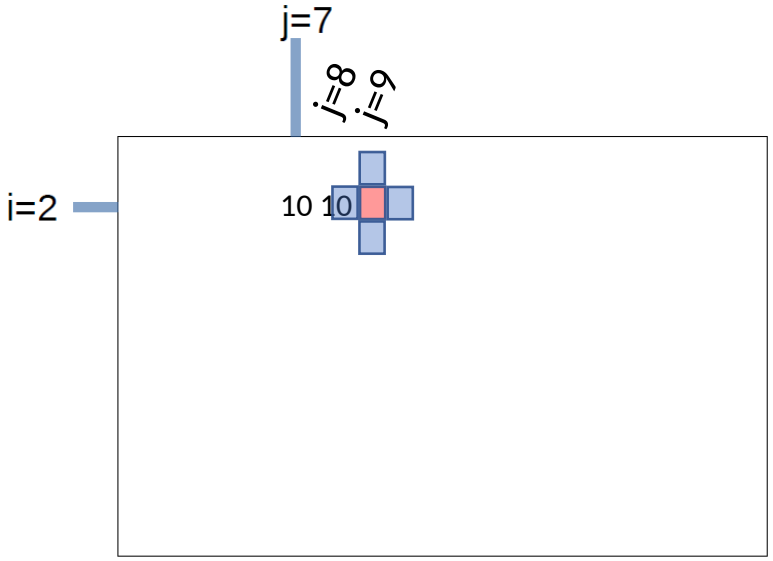
i=i+direc[direita].deltaLin,

j=j+direc[direita].deltaCol

2,8
2,7



IM

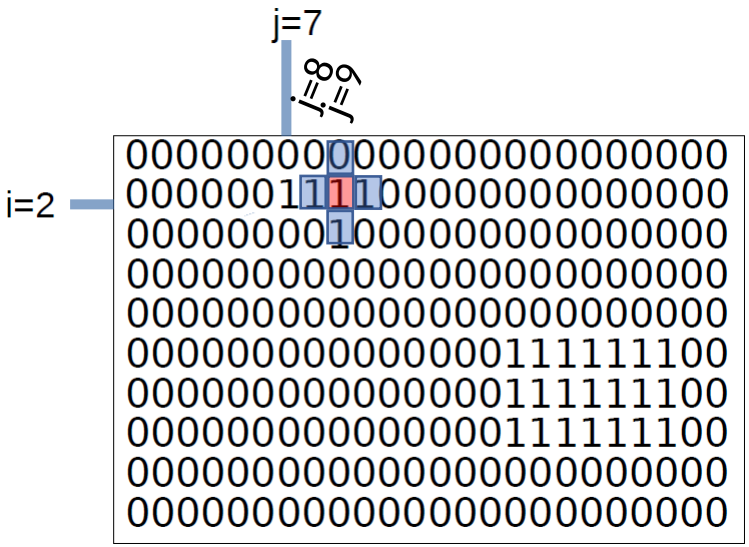


MAT

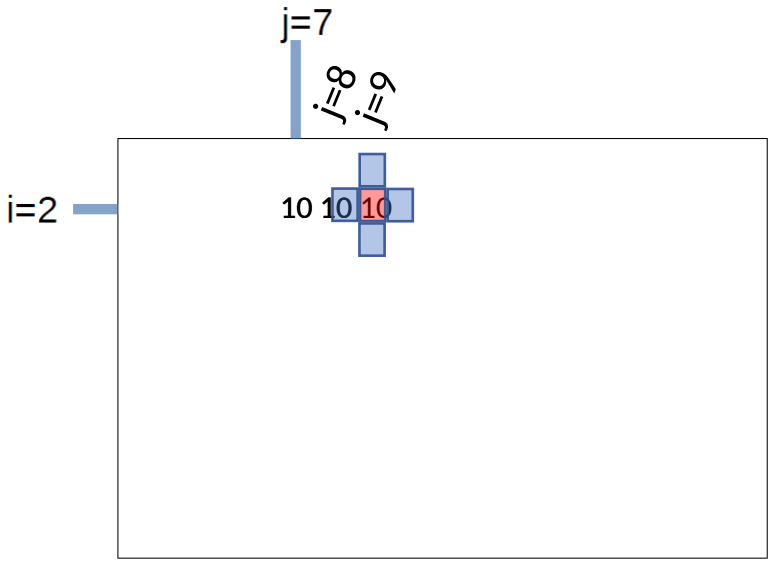
Rotulação não recursiva

Existe rótulo na vizinhança de MAT(2,9)?
SIM: então MAT(2,9) recebe o mesmo rótulo desse vizinho

2,8
2,7



IM



MAT

Rotulação não recursiva

Uma vez rotulado MAT(2,9), procura-se um vizinho de (2,9) não rotulado e igual a 1;
Este ocorre à direita de IM(2,9):

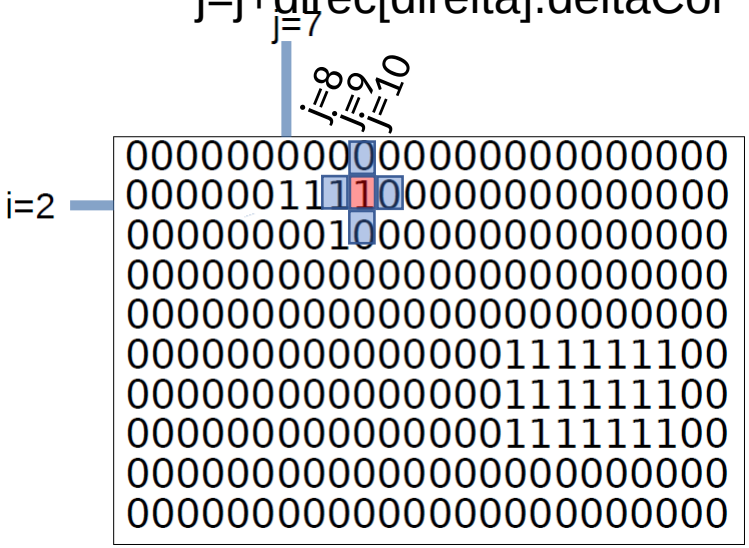
Empilha (2,9)

Atualiza (i,j) para a direita (2,10):

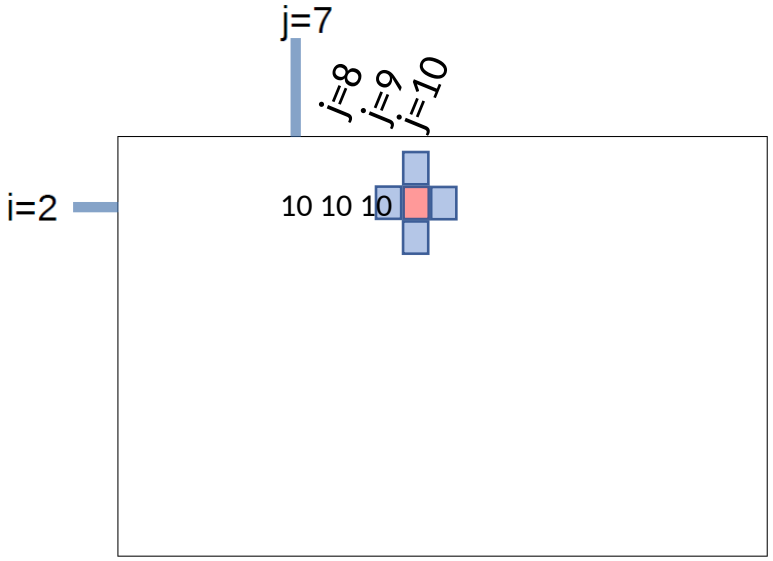
i=i+direc[direita].deltaLin,

j=j+direc[direita].deltaCol

2,9
2,8
2,7



IM

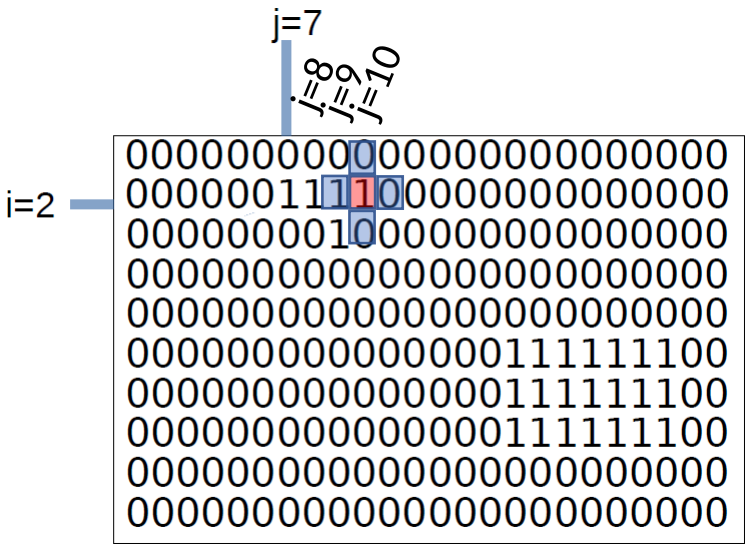


MAT

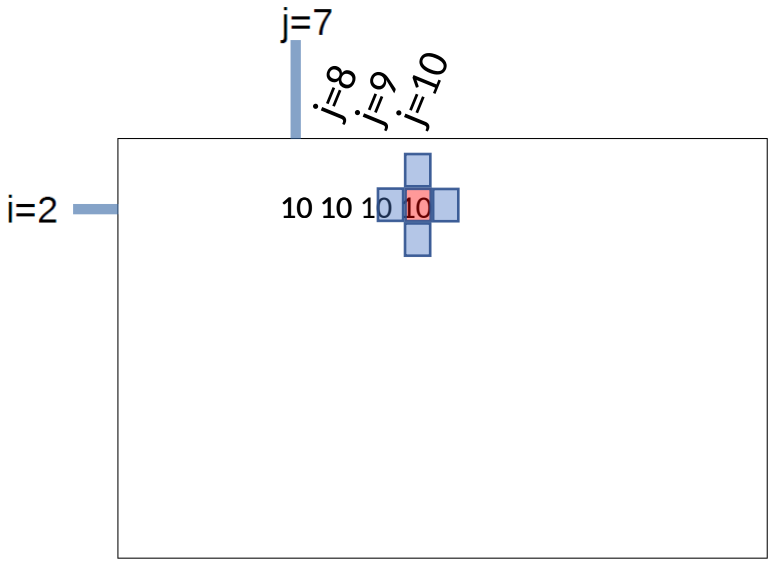
Rotulação não recursiva

Existe rótulo na vizinhança de MAT(2,10)?
SIM: então MAT(2,10) recebe o mesmo rótulo desse vizinho

2,9
2,8
2,7



IM

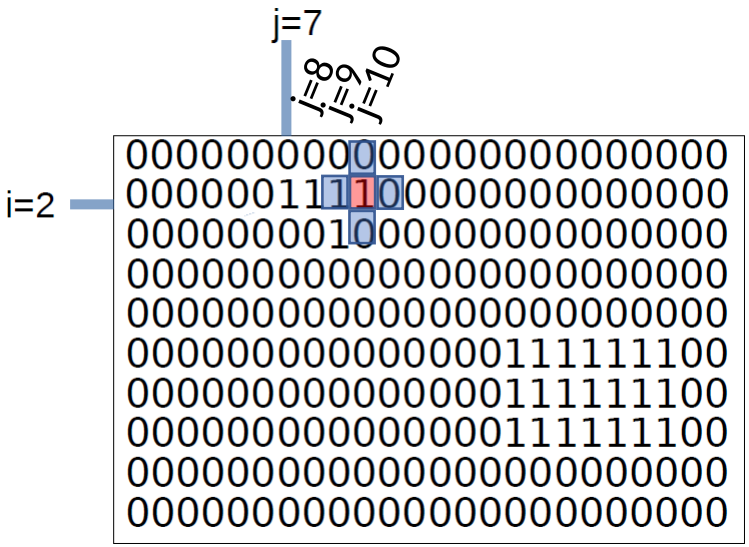


MAT

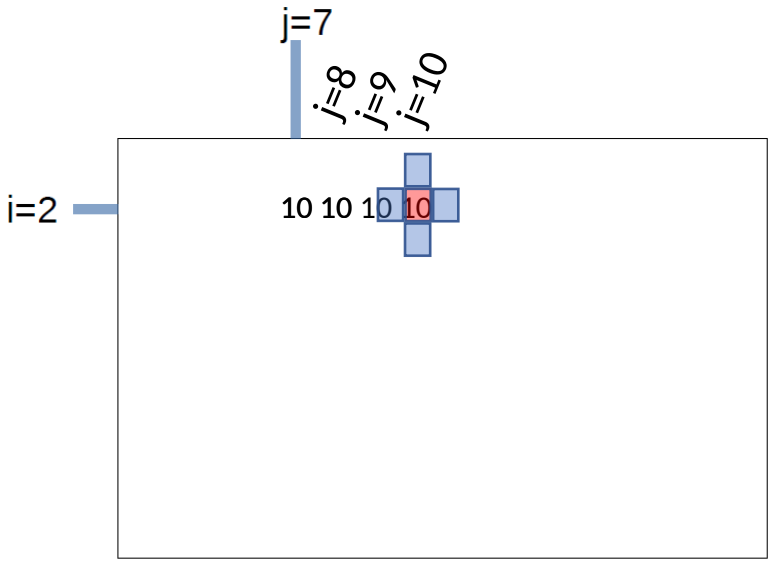
Rotulação não recursiva

Uma vez rotulado MAT(2,10), procura-se um vizinho de (2,10) não rotulado e igual a 1;
Todos os vizinhos são zeros ou são rotulados (marcados):

2,9
2,8
2,7



IM

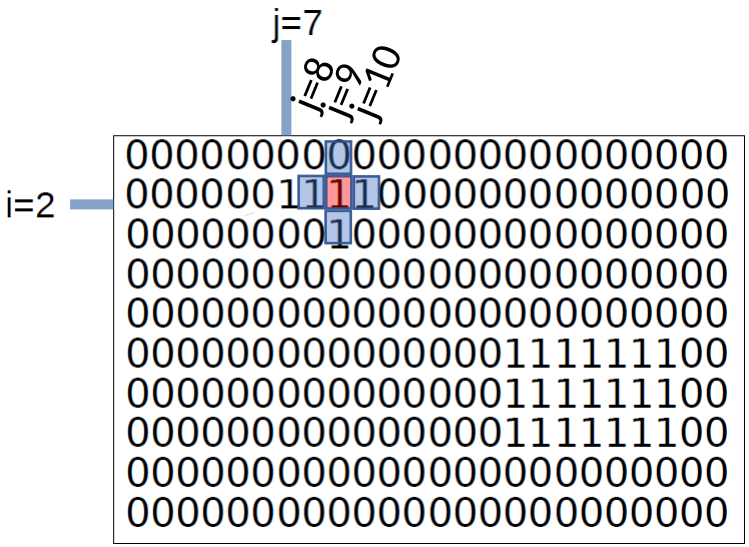


MAT

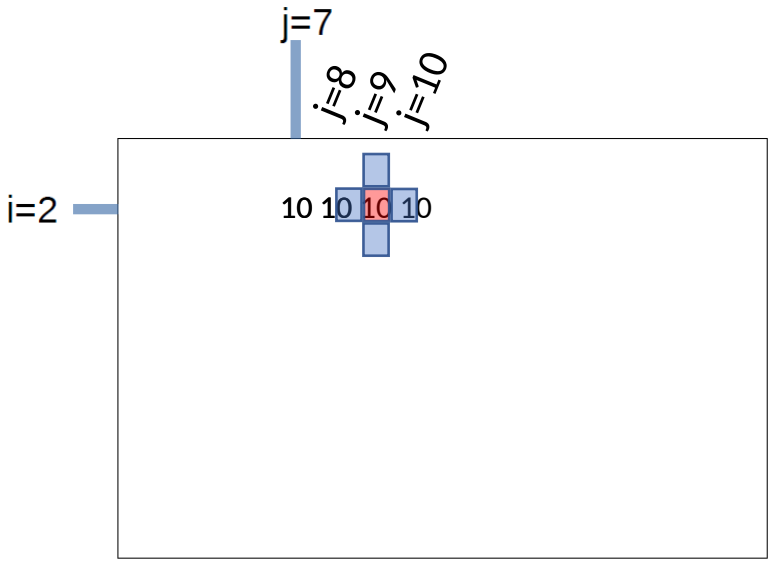
Rotulação não recursiva

Uma vez rotulado MAT(2,10), procura-se um vizinho de (2,10) não rotulado e igual a 1;
Não há tal vizinho!!!
Recorremos a pilha:
desempilha(piha) → i=2, j=9

~~2,9~~
2,8
2,7



IM

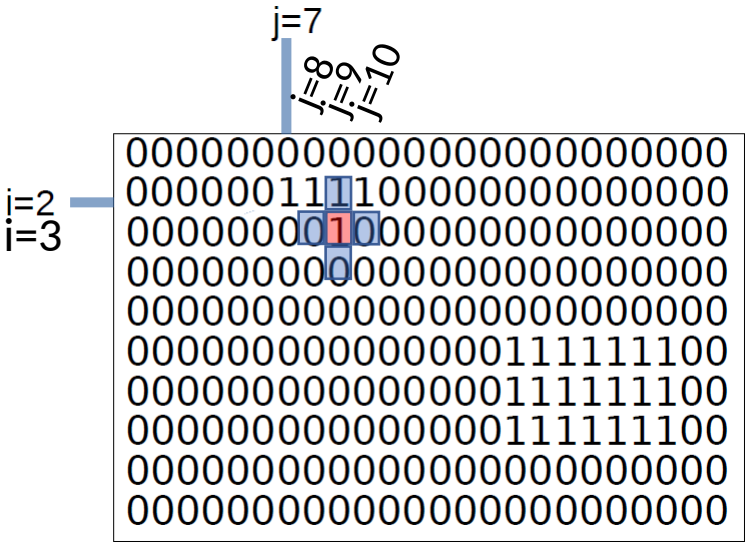


MAT

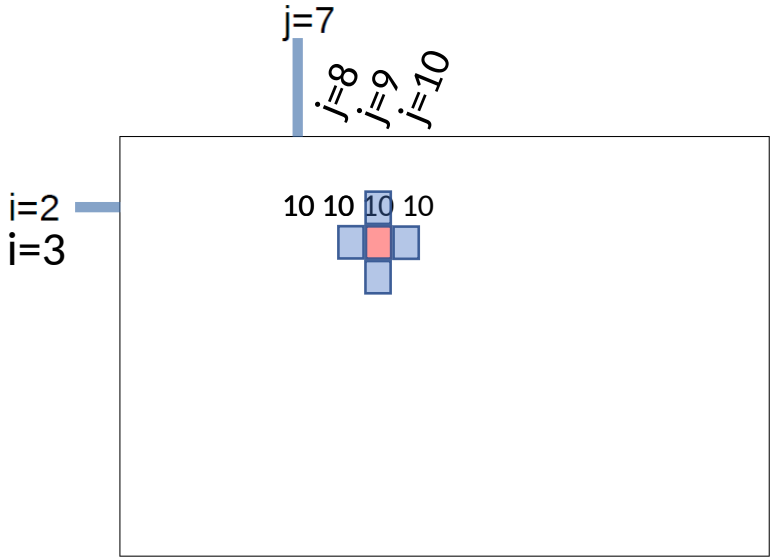
Rotulação não recursiva

desempilha(piha) → i=2, j=9
Há um vizinho sem rótulo abaixo de (2,9):
Empilha (2,9)
Atualiza (i,j) para abaixo de (2,9), i=3,j=9:
i=i+direc[abaixo].deltaLin,
j=j+direc[abaixo].deltaCol

2,9
2,8
2,7



IM

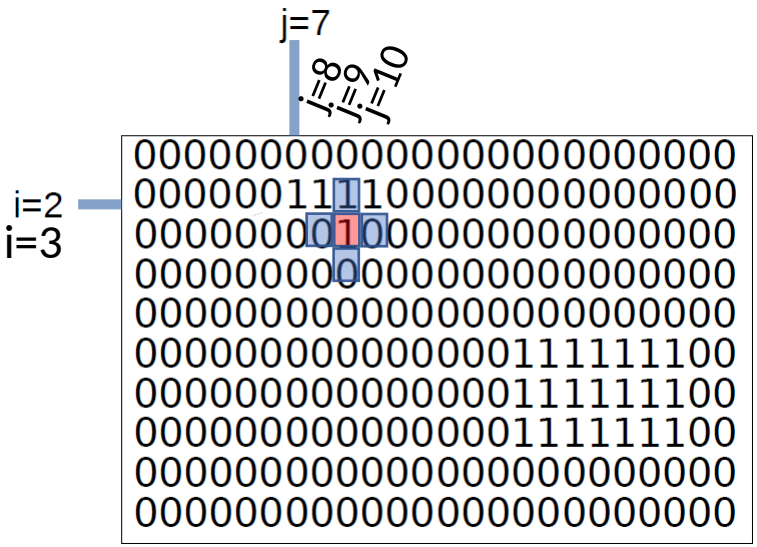


MAT

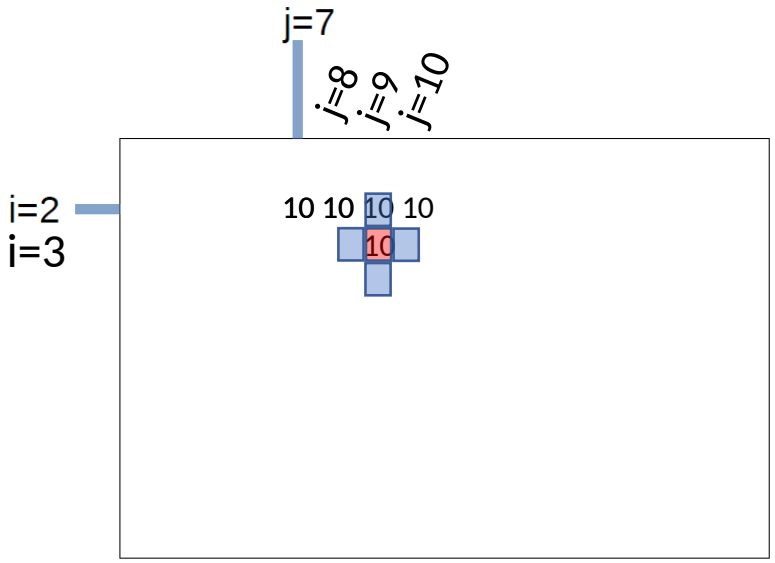
Rotulação não recursiva

Existe rótulo na vizinhança de MAT(3,9)?
SIM: então MAT(3,9) recebe o mesmo rótulo desse vizinho

2,9
2,8
2,7



IM

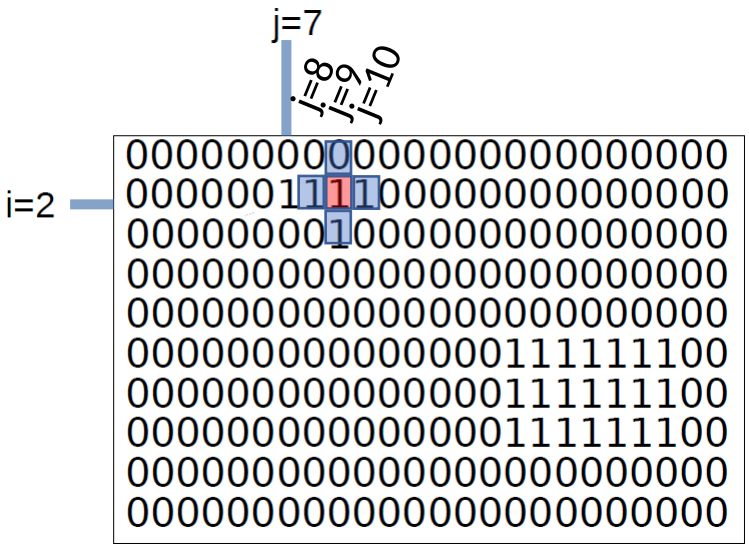


MAT

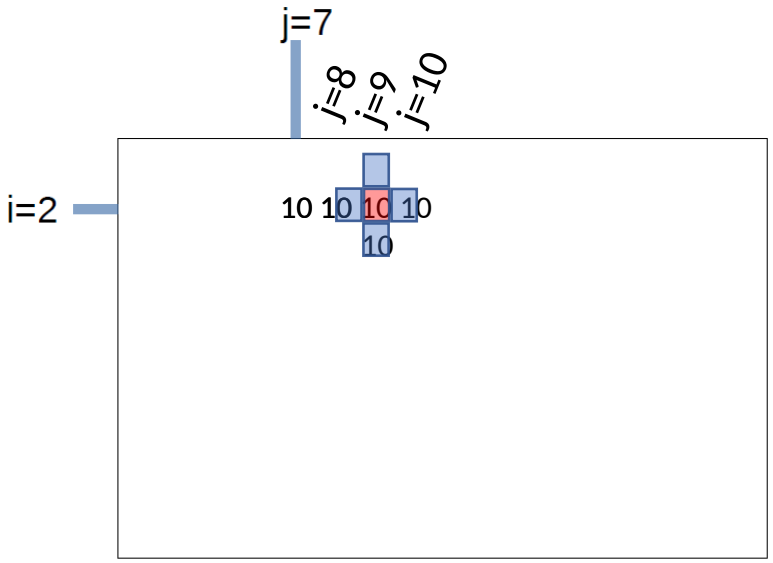
Rotulação não recursiva

Uma vez rotulado MAT(3,9), procura-se um vizinho não rotulado e igual a 1;
Não há tal vizinho!!!
Recorremos a pilha:
desempilha(piha) → i=2, j=9

~~2,9~~
2,8
2,7



IM

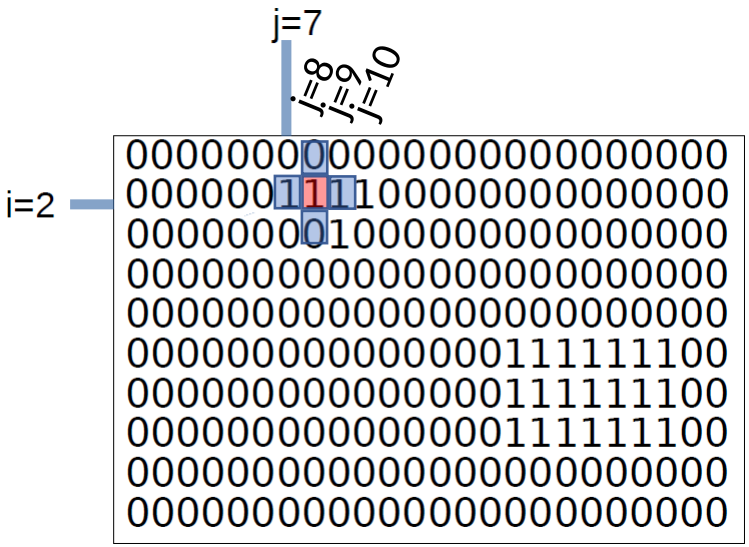


MAT

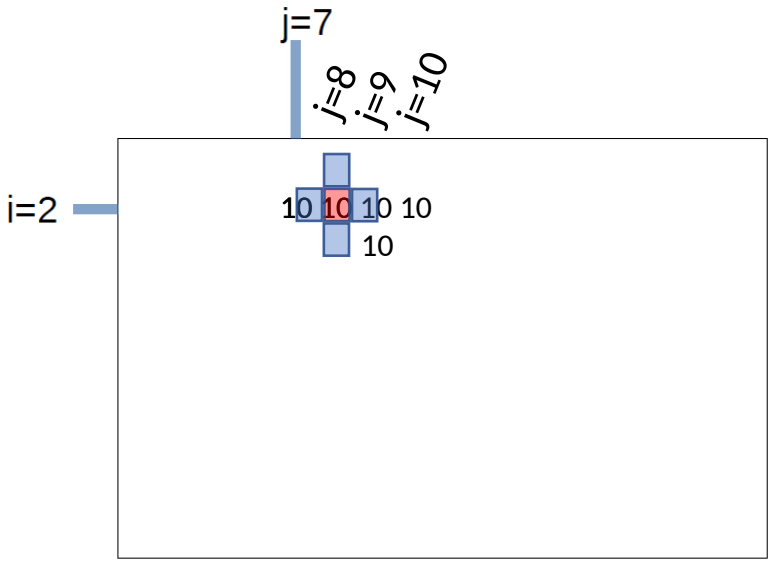
Rotulação não recursiva

Para (2,9), procura-se um vizinho não rotulado e igual a 1;
Não há tal vizinho!!!
Recorremos a pilha:
desempilha(piha) → i=2, j=8

~~2,9~~
~~2,8~~
2,7



IM

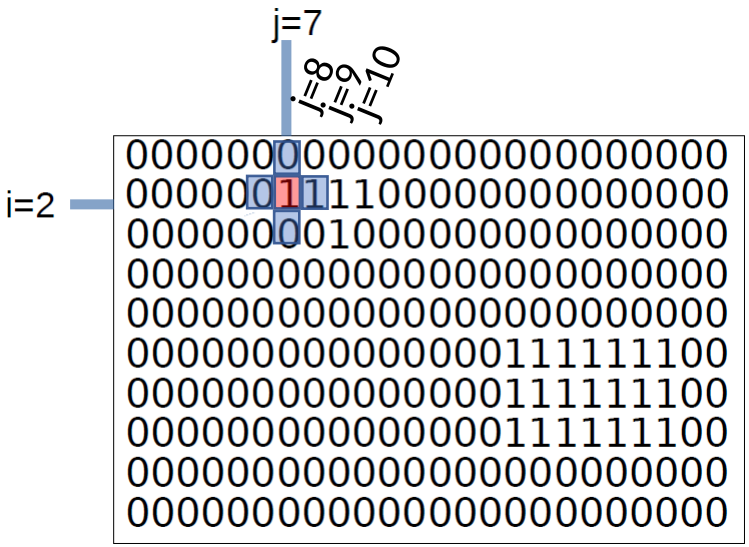


MAT

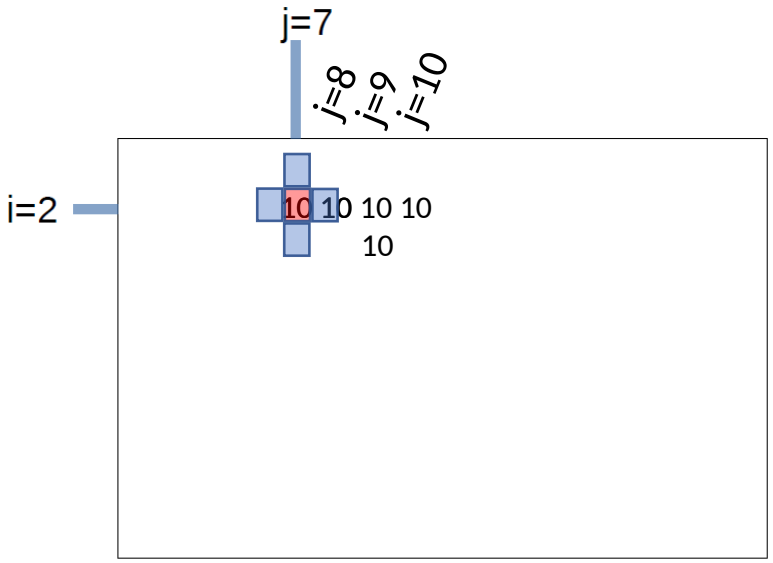
Rotulação não recursiva

Para (2,8), procura-se um vizinho não marcado/rotulado e igual a 1;
Não há tal vizinho!!!
Recorremos a pilha:
desempilha(piha) → i=2, j=7

~~2,9~~
~~2,8~~
~~2,7~~



IM

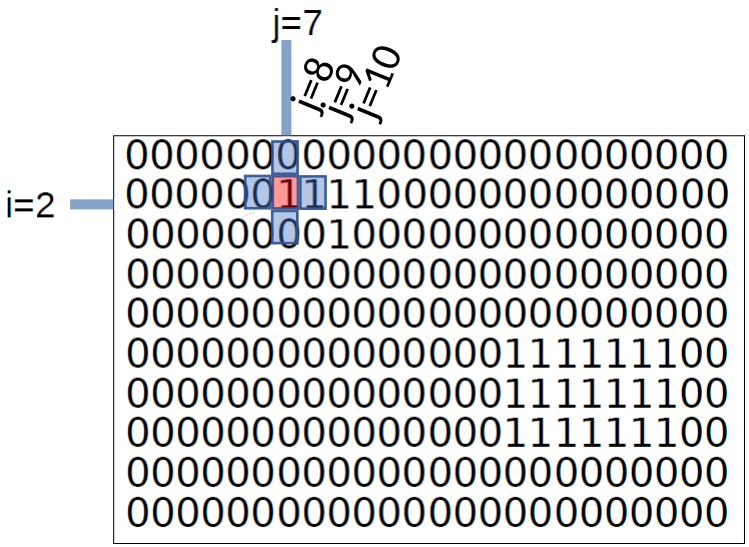


MAT

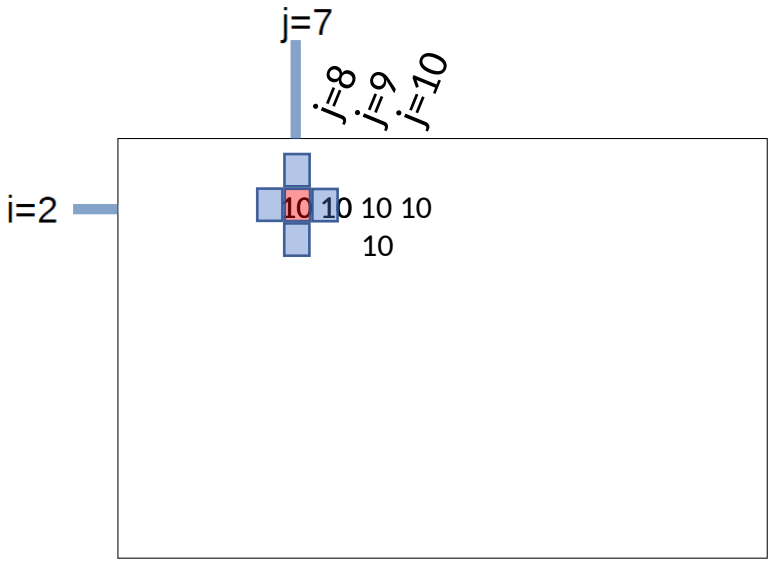
Rotulação não recursiva

Para (2,7), procura-se um vizinho não marcado/rotulado e igual a 1;
Não há tal vizinho e a pilha está vazia!!!
Indica que um componente conexo foi completamente rotulado.

~~2,9~~
~~2,8~~
~~2,7~~



IM

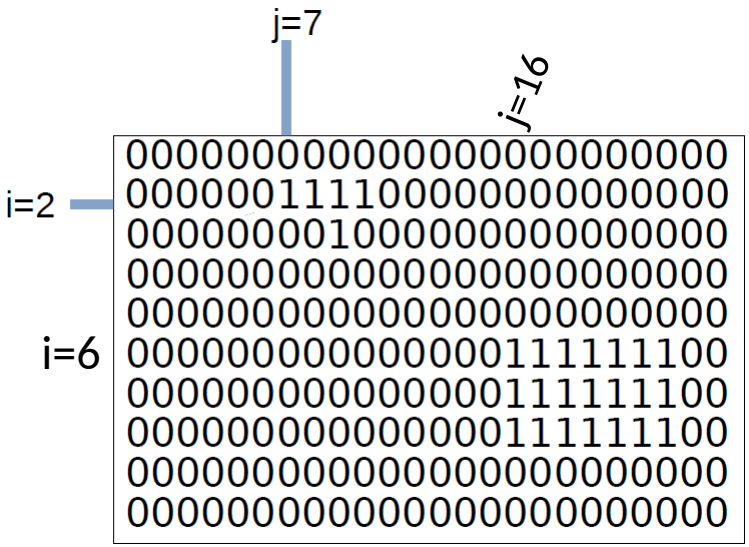


MAT

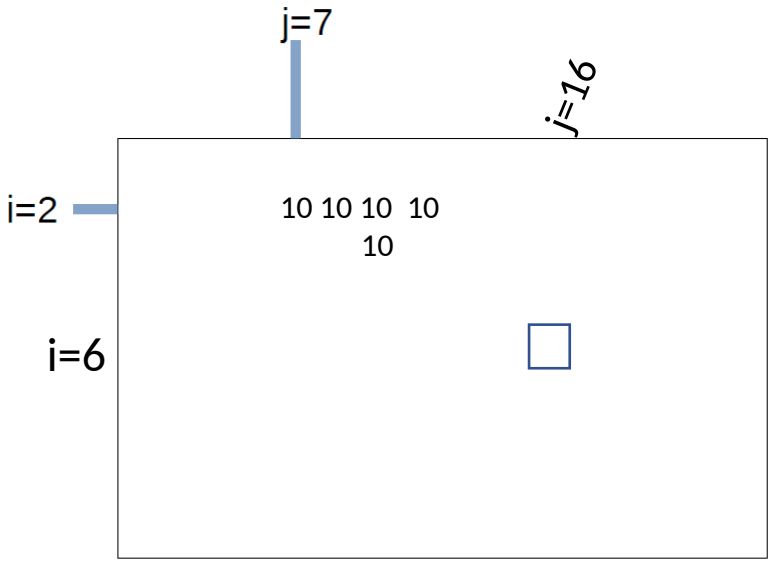
Rotulação não recursiva

Retorna ao laço para i,j até encontra outro componente a ser rotulado... O que vai acontecer para i = 6, j =16...

```
for i=1;i<=M;i++
        for j=1;j<=N;j++
          if IM[i,j]==1
                    marcador(i,j,direc,rot, pilha)
```



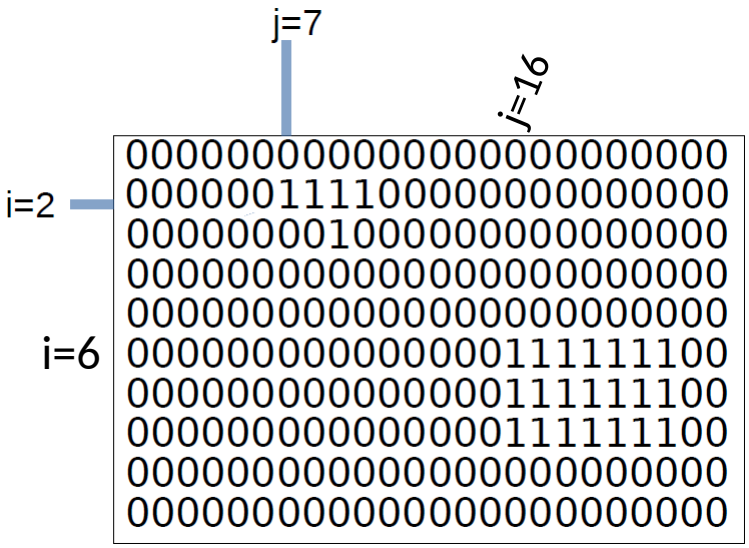
IM



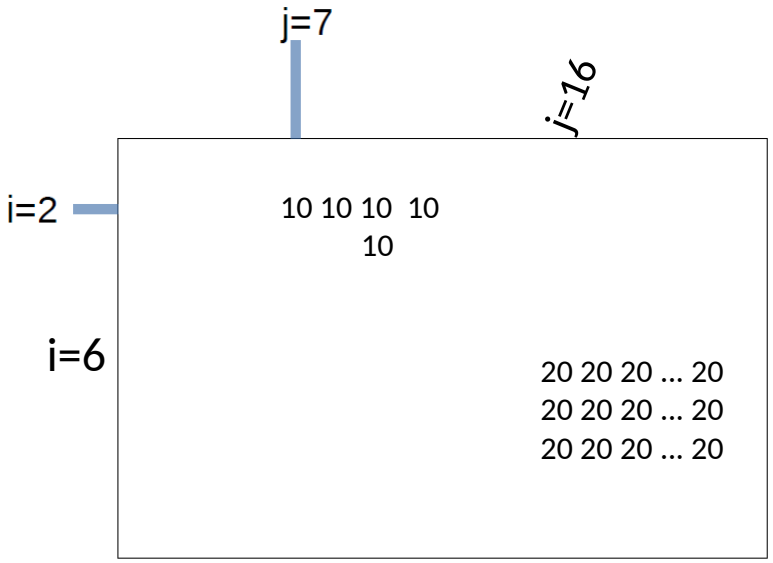
MAT

Rotulação não recursiva

O procedimento já descrito se repete e a rotulação é executada...



IM



MAT

Outras aplicações do conceito de janela em uma imagem serão vistos no decorrer do curso...