第９回：二値化画像処理
（ラベリングなど）

前回の復習

- 二値化することにより輪郭線の追跡などが容易になることがわかった

- また，膨張や縮小アルゴリズムを適用することで硬膜等の除去が行えることがわかった

目　的

- 二値化画像処理についてさらに理解を深める
  - Opening/closing処理を行う
  - ラベリングアルゴリズムを実装する

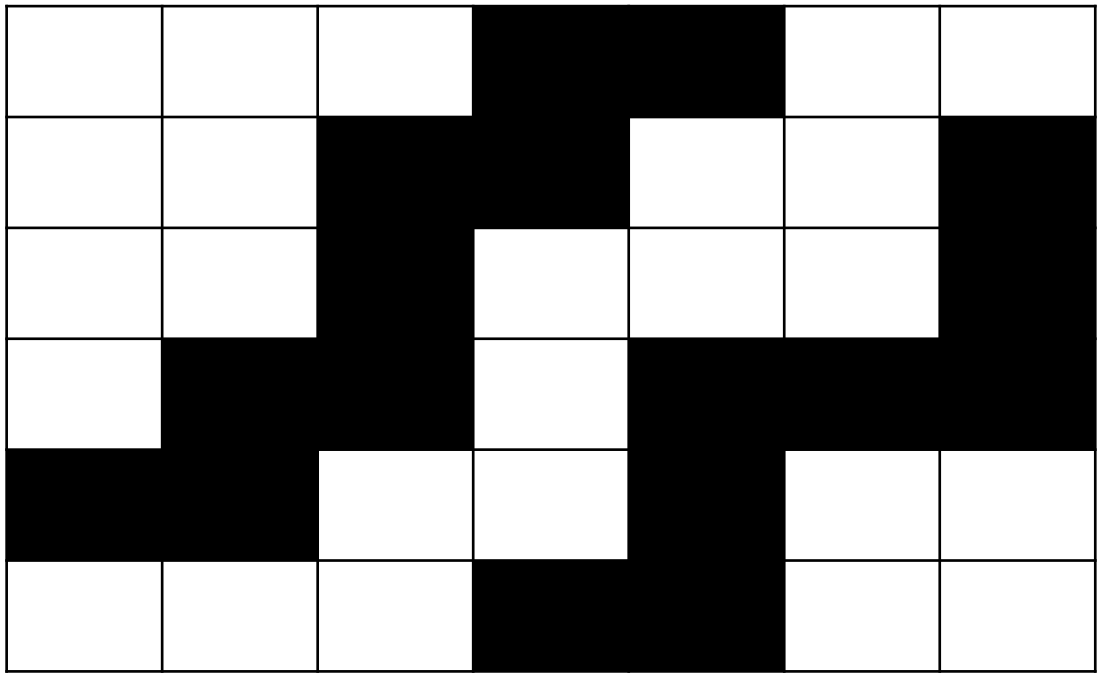
Opening/Closing処理とは

- 膨張/縮小を繰り返すことにより画像に空いた隙間を埋めること･もしくはその逆の処理

前回作成したプログラムをそのまま利用できる

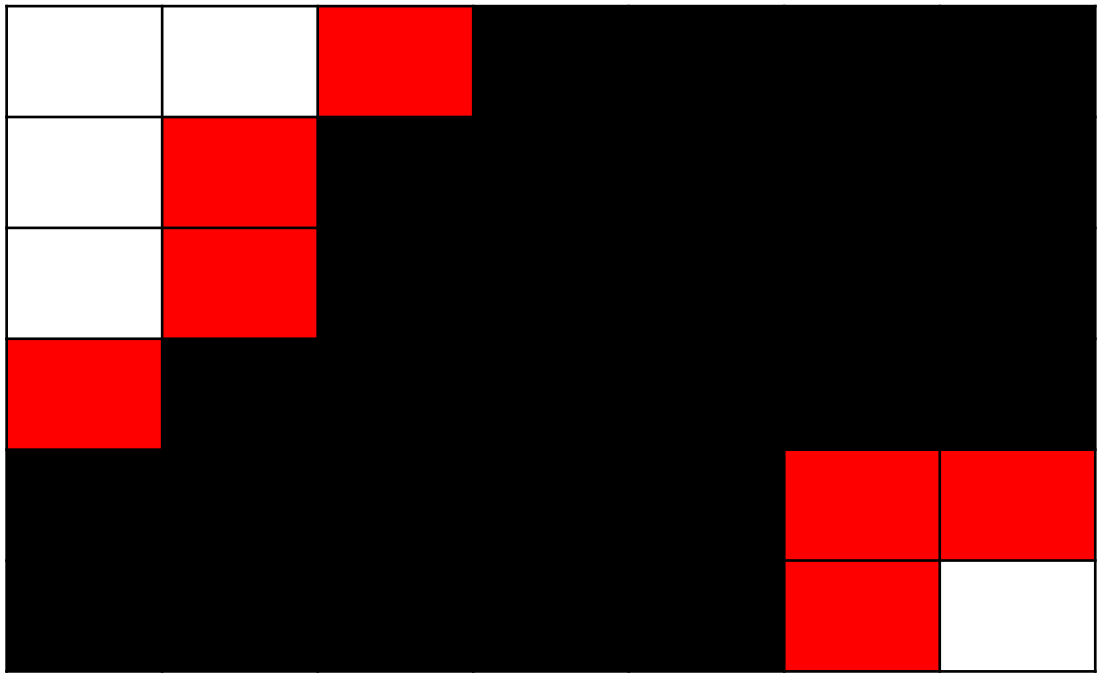
Closing処理のイメージ

原画像

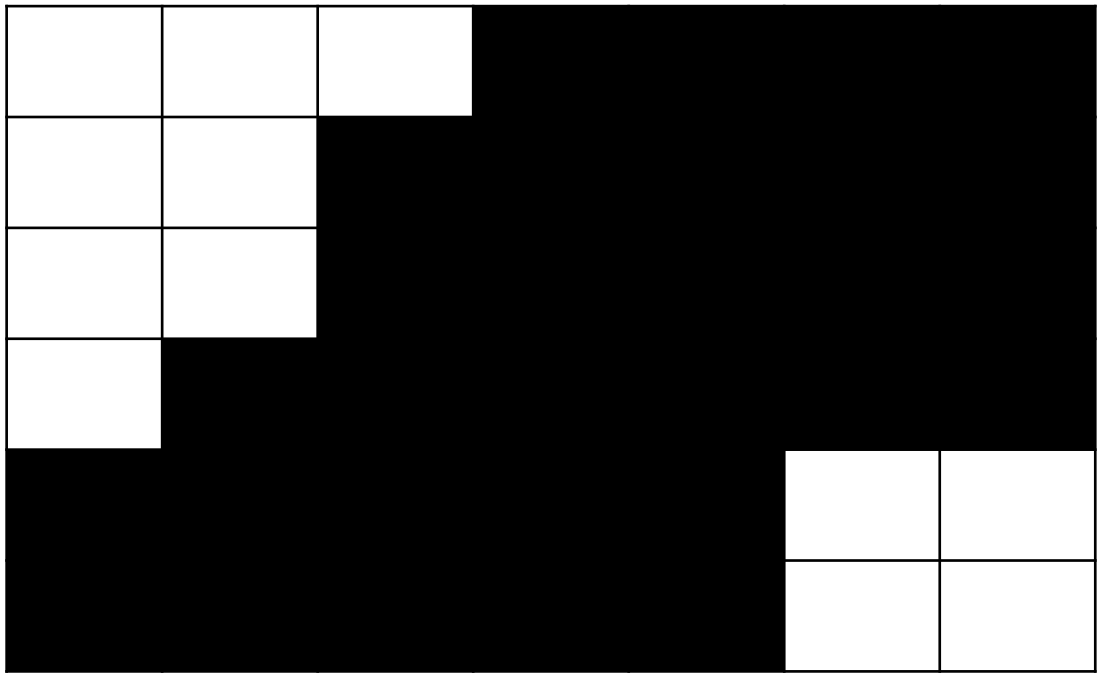


Closing処理のイメージ

Closing処理のイメージ

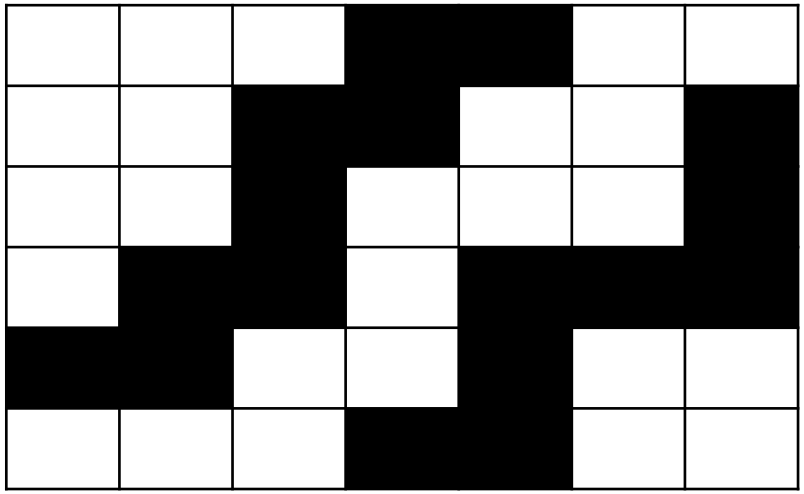


Closing処理のイメージ

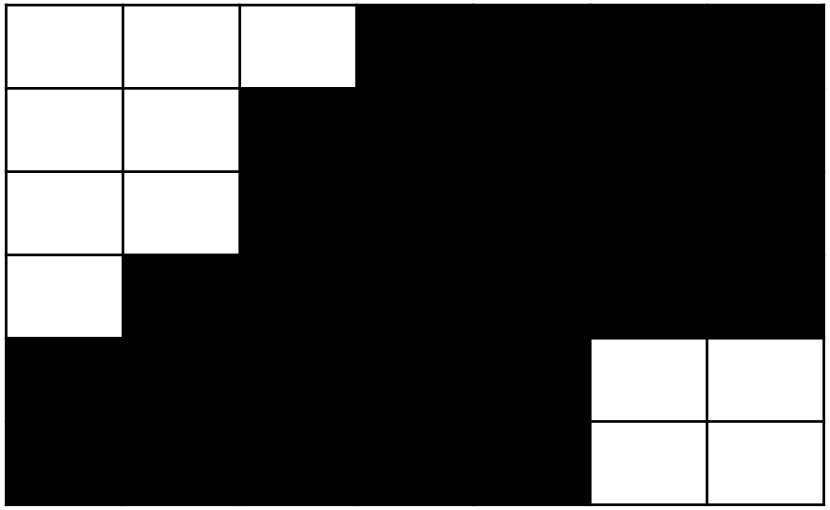


処理前後の比較

原画像



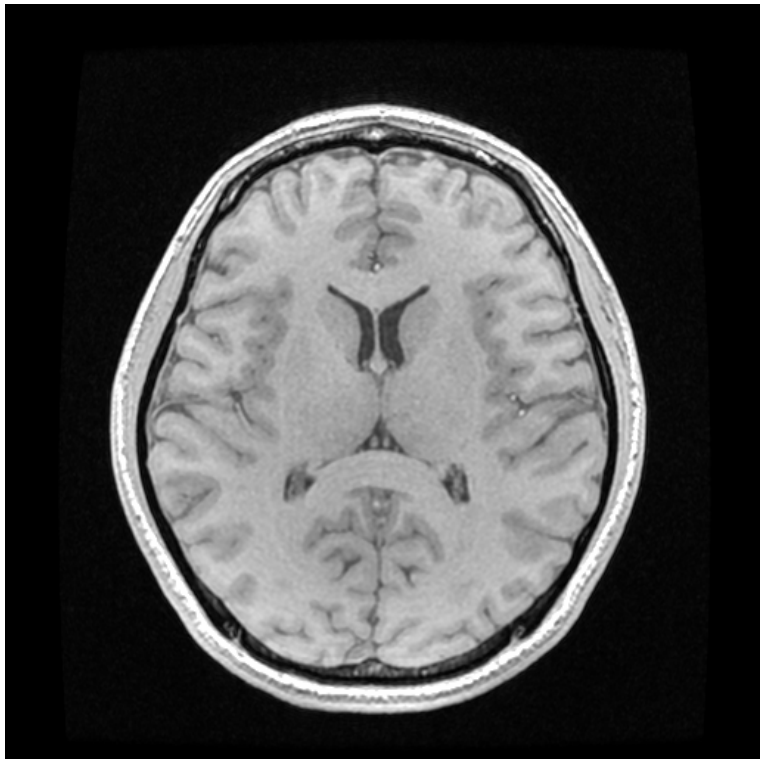
Closing後

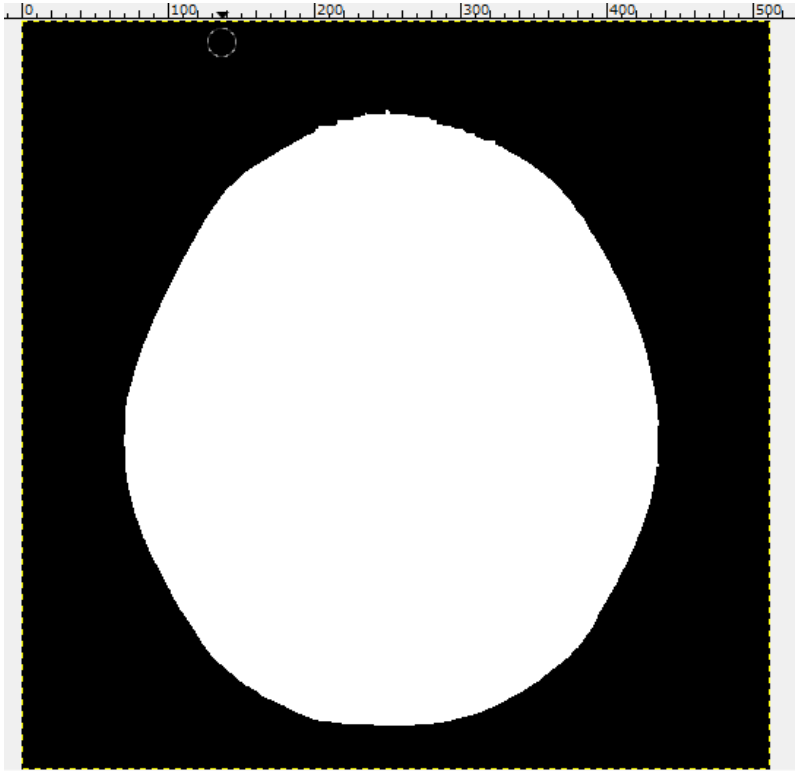


実行例

二値化を行う画像を指定:
入力ファイル名 (*.pgm) : mri3.pgm
横の画素数 = 512, 縦の画素数 = 512
最大階調値 = 255
画像は正常に読み込まれました.
２値化のしきい値（0-255）: 30
出力ファイル名 (*.pgm) : mri3-1.pgm
画像は正常に出力されました.
膨張・収縮させる回数: 5
出力ファイル名 (*.pgm) : mri3-2.pgm
画像は正常に出力されました.

画像処理結果の例





演習1

- 前回作成したプログラムを利用してclosing処理を実行せよ

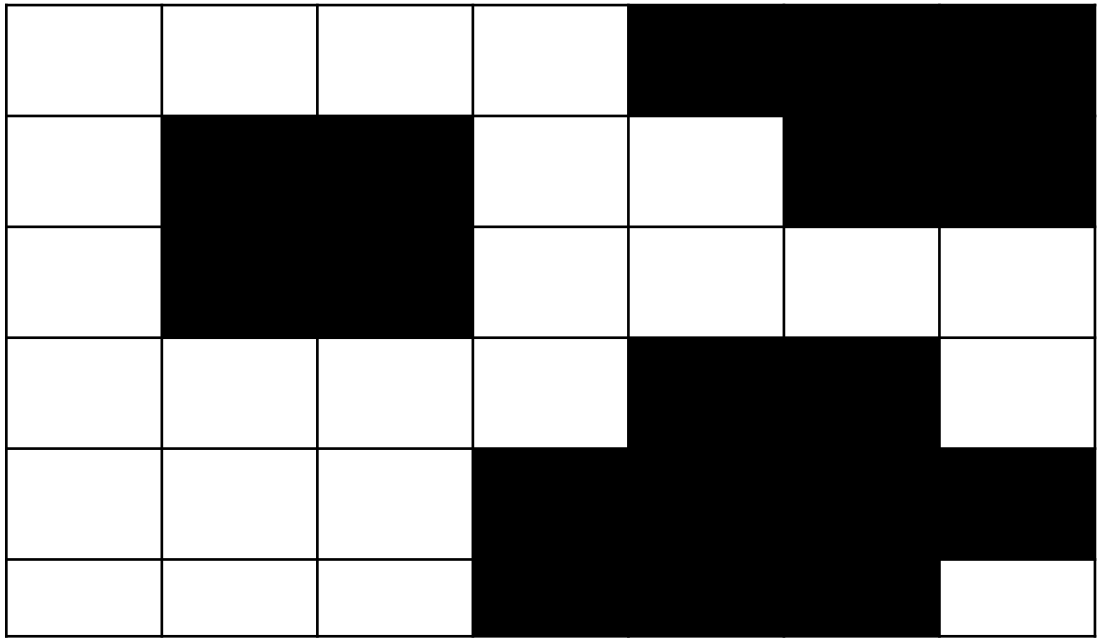
- Opening処理とはどのような処理か考えておくこと

目　的

- 二値化画像処理についてさらに理解を深める
  - Opening/closing処理を行う
  - ラベリングアルゴリズムを実装する

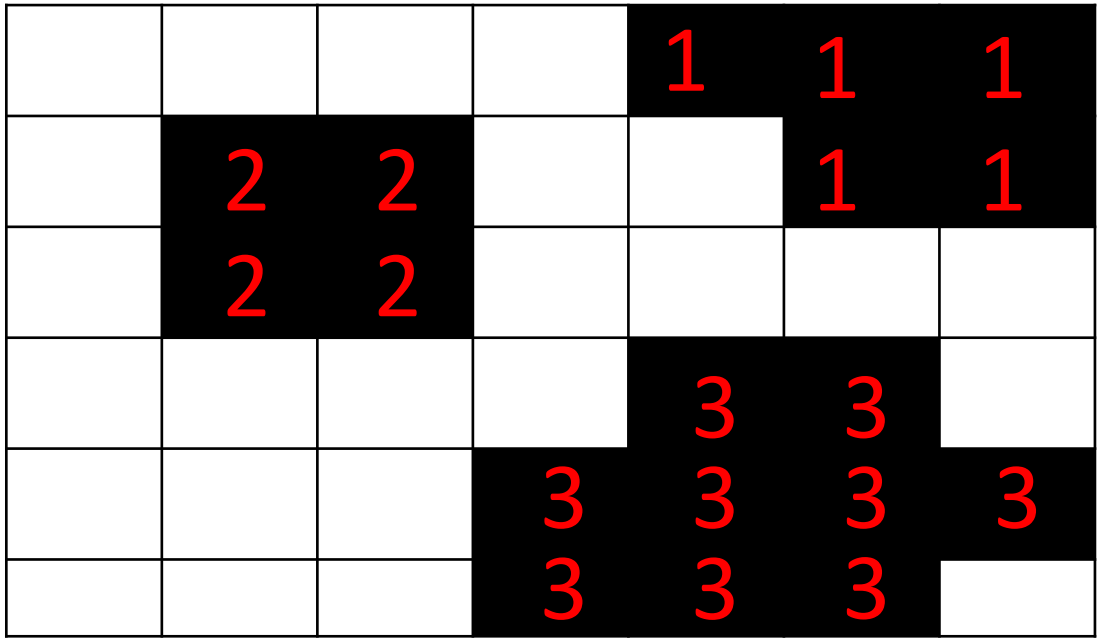
ラベリング処理とは

原画像



ラベリング処理とは

原画像



main関数

```
char file[256]=“”;
load_image(0,file);
labeling(0,1);
save_image(1,file);
```

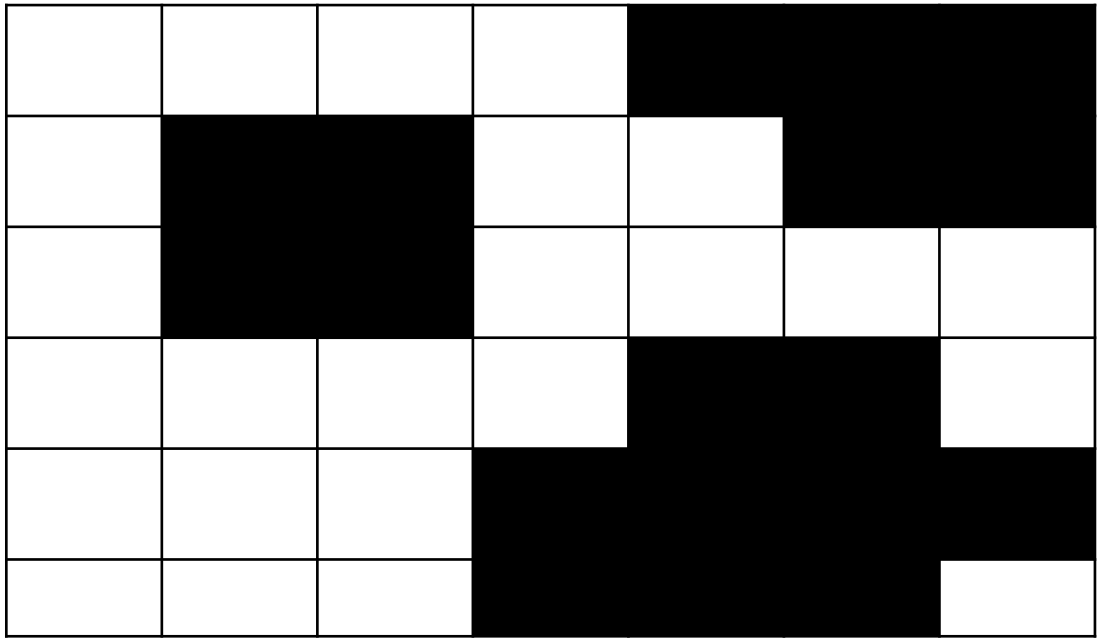
Labeling関数

```
void labeling( int n1, int n2 )
{
   int x,y;
   int count=0; /* 白い背景中の黒い孤立図形の総数 */

   /* ラベルと画像No.n2を初期化 */
   width[n2]=width[n1];  height[n2]=height[n1];
   for(y=0;y<height[n1];y++)
      for(x=0;x<width[n1];x++){
         label[x][y] = 0;  image[n2][x][y] = 0;
      }
```

ラベリング処理とは

原画像



配列変数 label

0	0	0	0	1	1	1
0	2	2	0	0	1	1
0	2	2	0	0	0	0
0	0	0	0	3	3	0
0	0	0	3	3	3	3
0	0	0	3	3	3	0

```
/* 画像No.n1をスキャン */
   for(y=0;y<height[n1];y++)
      for(x=0;x<width[n1];x++){
         if ( image[n1][x][y] == 0 && label[x][y] == 0 ){
                  count++;
                  search( n1, x, y, count );
         }
      }
```

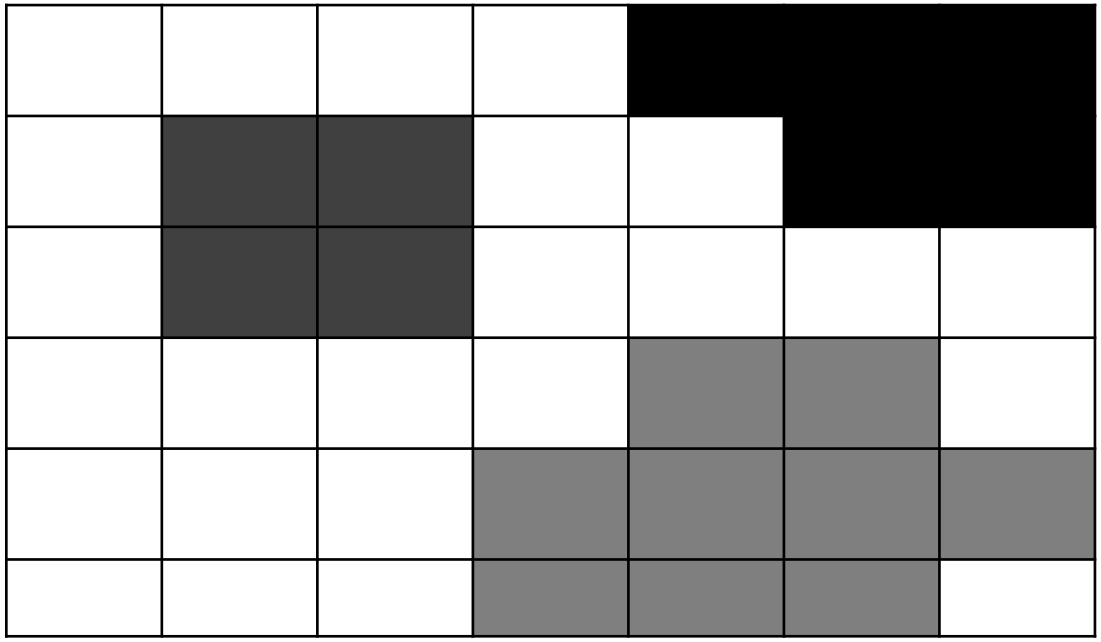
```
/* 画像No.n2を作る */
   printf("孤立図形総数(ラベル最大値) = %d¥n",count);
   if ( count > 0 )
      for(y=0;y<height[n2];y++)
         for(x=0;x<width[n2];x++)
            image[n2][x][y] = (label[x][y] * 50) % 255;
}
```

剰余を画素値に設定している.

例えば,（ラベル１×５０）％２５５＝５０

ラベリング処理とは

原画像



復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

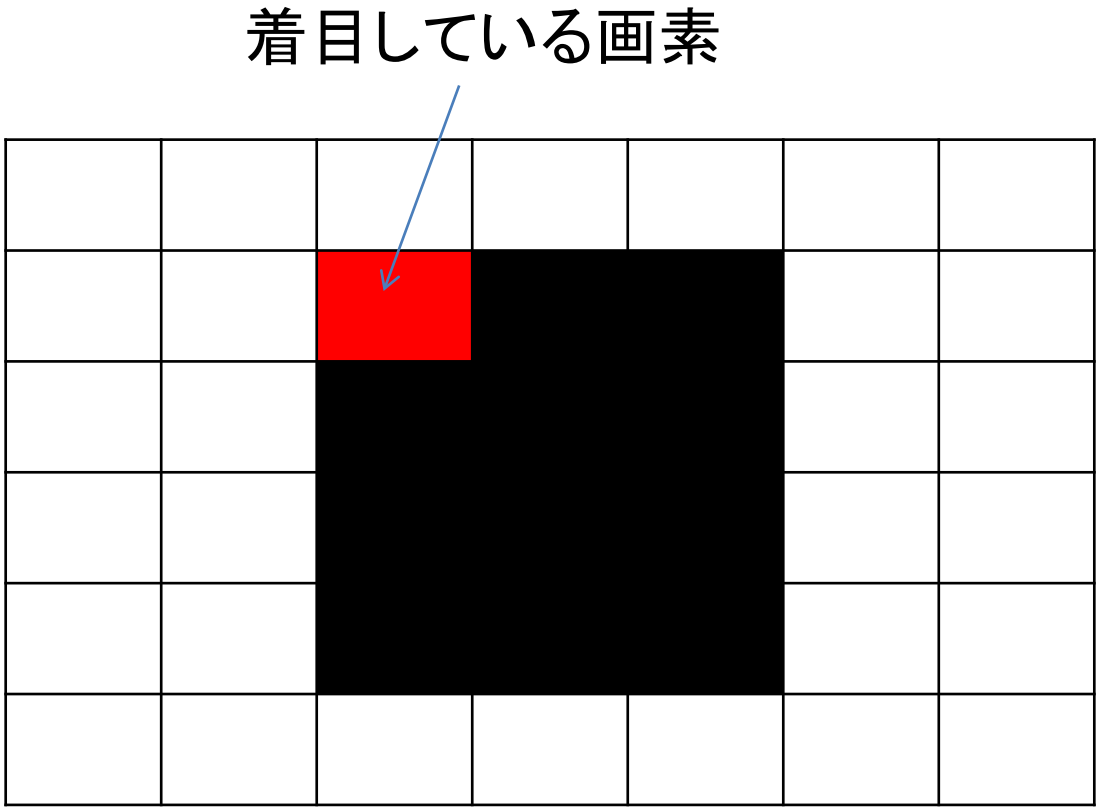
復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

void search(int n, int x, int y, int cnum)関数

search(画像番号, 着目している画素のX座標,着目している画素のY座標, ラベル番号)



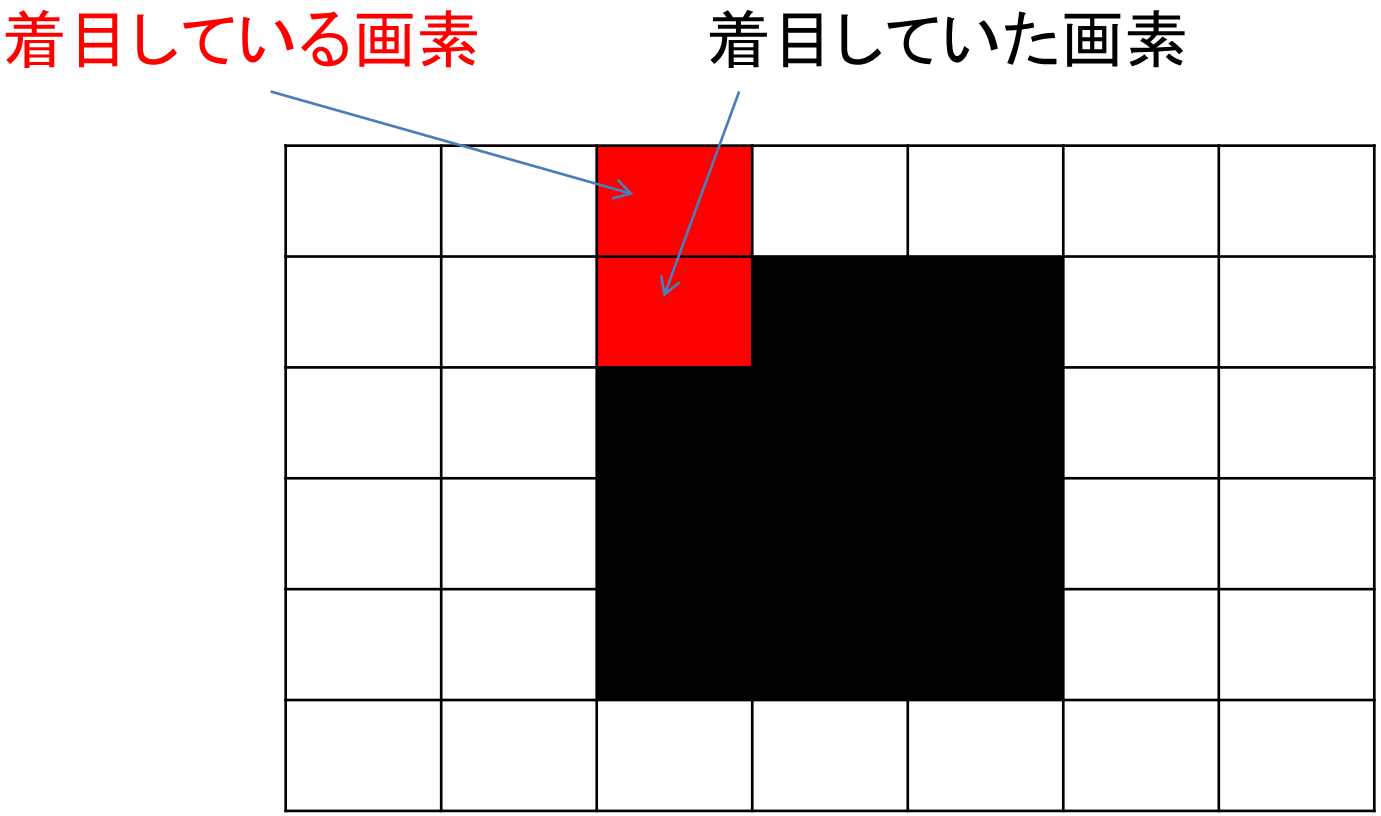
復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

void search(int n, int x, int y, int cnum)関数

```
search( n, x, y-1, cnum );
```



復習：再帰呼び出し

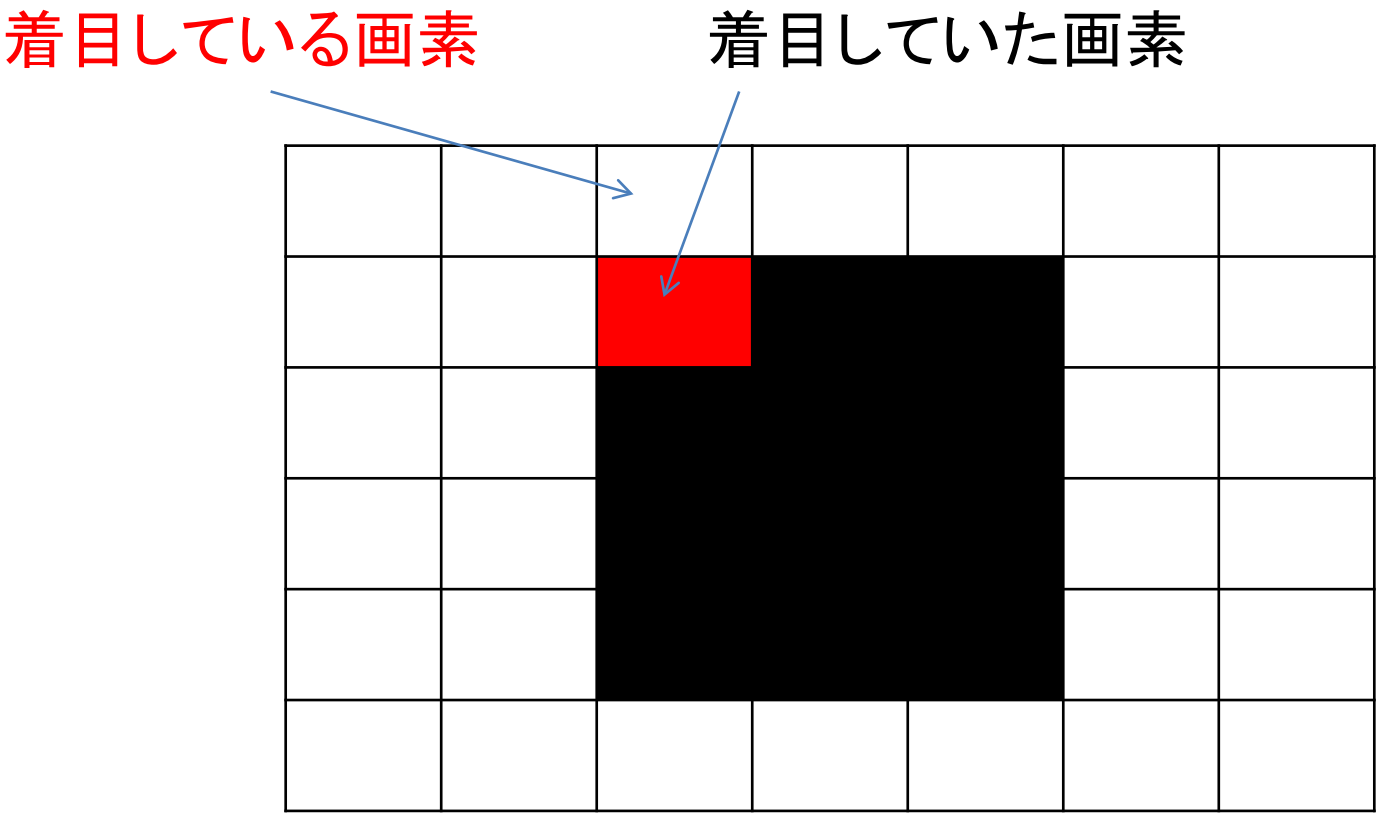
- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

黒でないので条件にかからない

void search(int n, int x, int y, int cnum)関数

```
search( n, x, y-1, cnum );
```



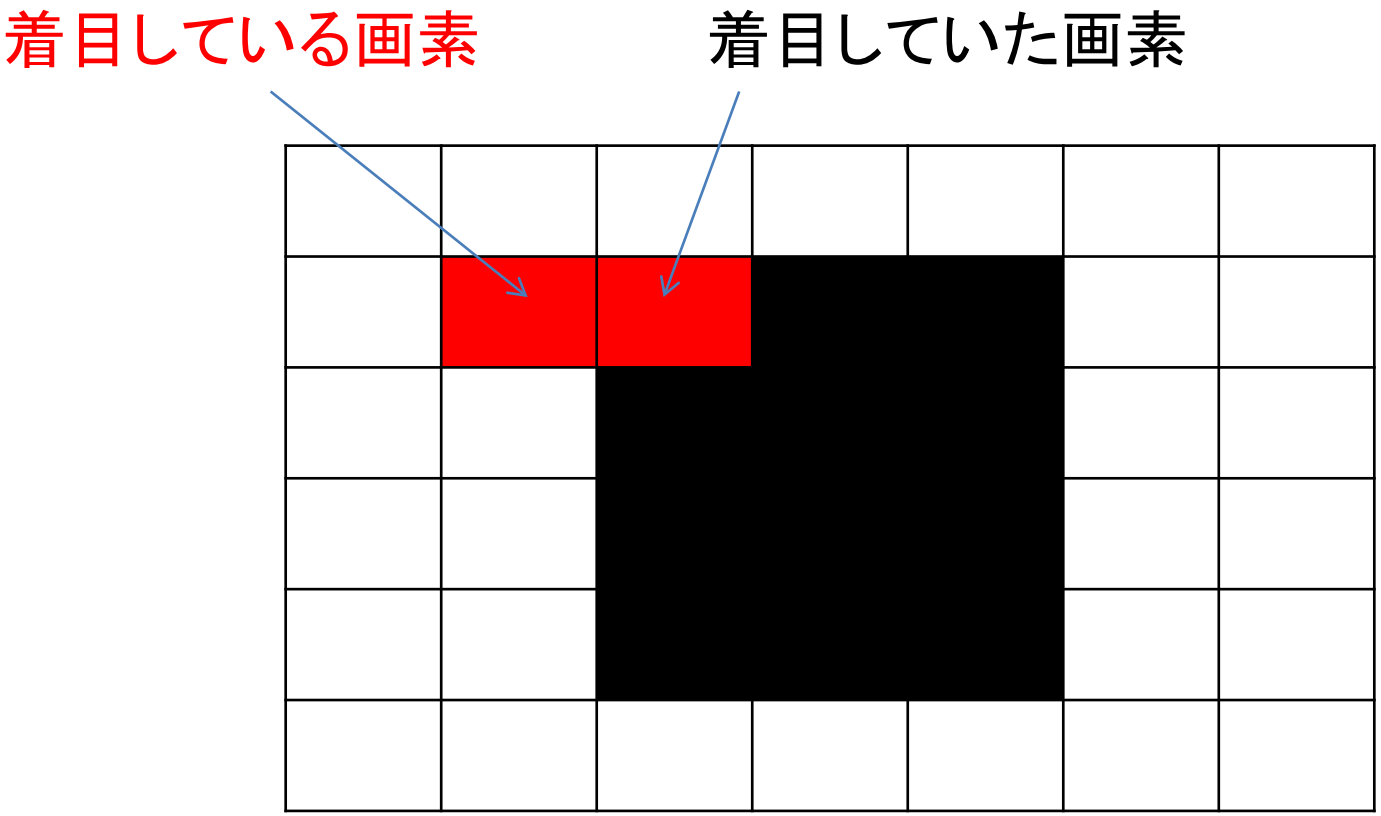
復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

void search(int n, int x, int y, int cnum)関数

```
search( n, x-1, y, cnum );
```



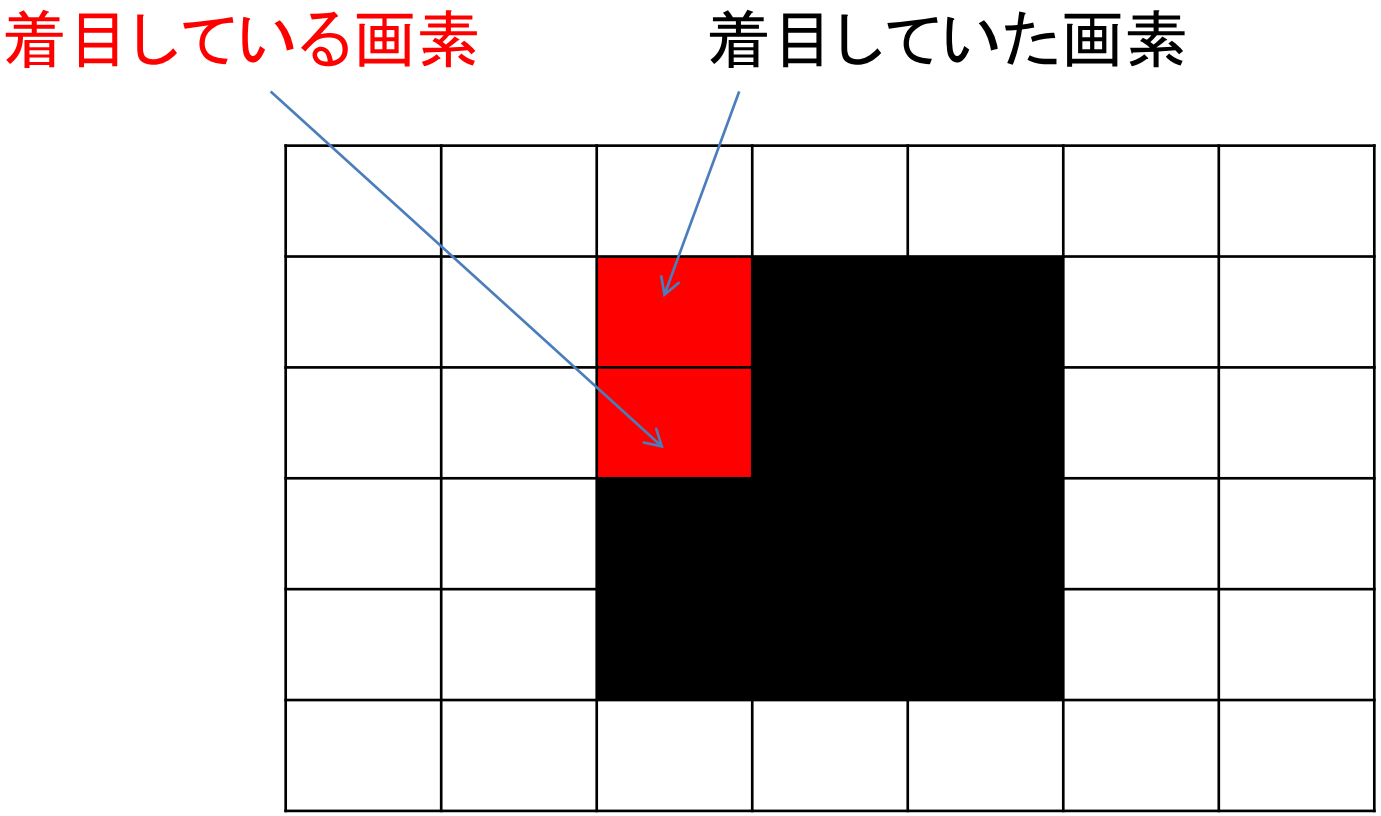
復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

void search(int n, int x, int y, int cnum)関数

```
search( n, x, y+1, cnum );
```



復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

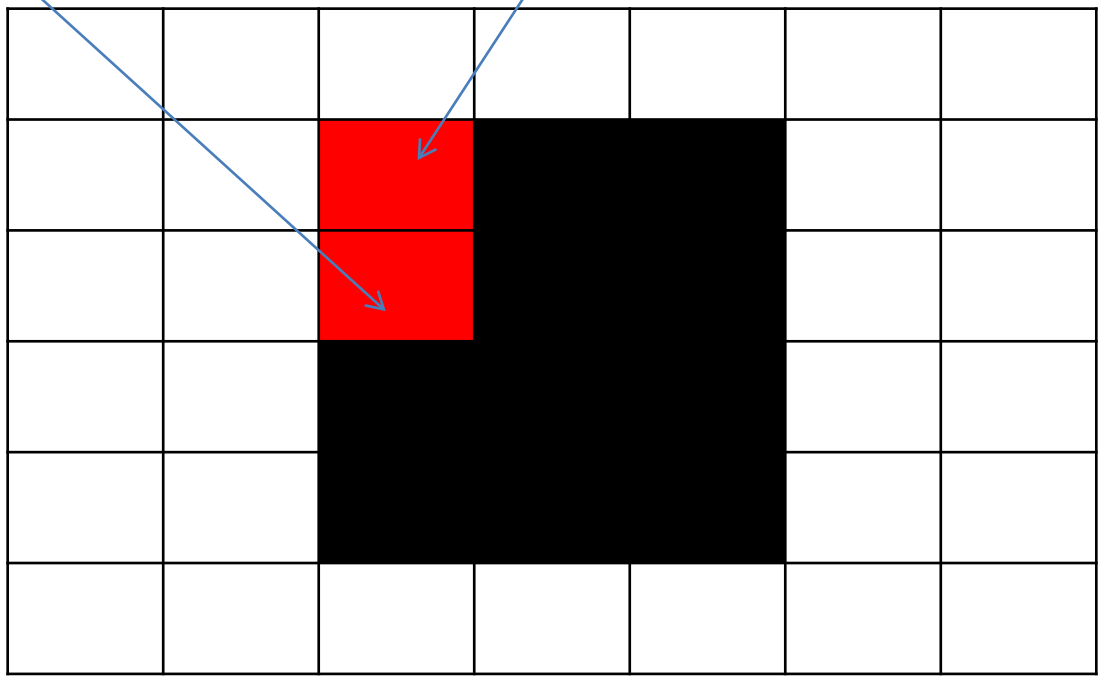
```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 && label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

void search(int n, int x, int y, int cnum)関数

```
search( n, x, y-1, cnum );
```

着目していた画素

着目している画素



復習：再帰呼び出し

- 関数内から自身の関数を呼ぶこと

```
void search( int n, int x, int y, int cnum )
/* 画像No.nの座標(x,y)の周囲を探索．cnum:現在のラベル */
{
   if ( inside(n,x,y) && image[n][x][y]==0 &&
label[x][y]==0 ){
      label[x][y] = cnum;
      search( n, x, y-1, cnum ); /* 上の画素 */
      search( n, x-1, y, cnum ); /* 左の画素 */
      search( n, x, y+1, cnum ); /* 下の画素 */
      search( n, x+1, y, cnum ); /* 右の画素 */
   }
}
```

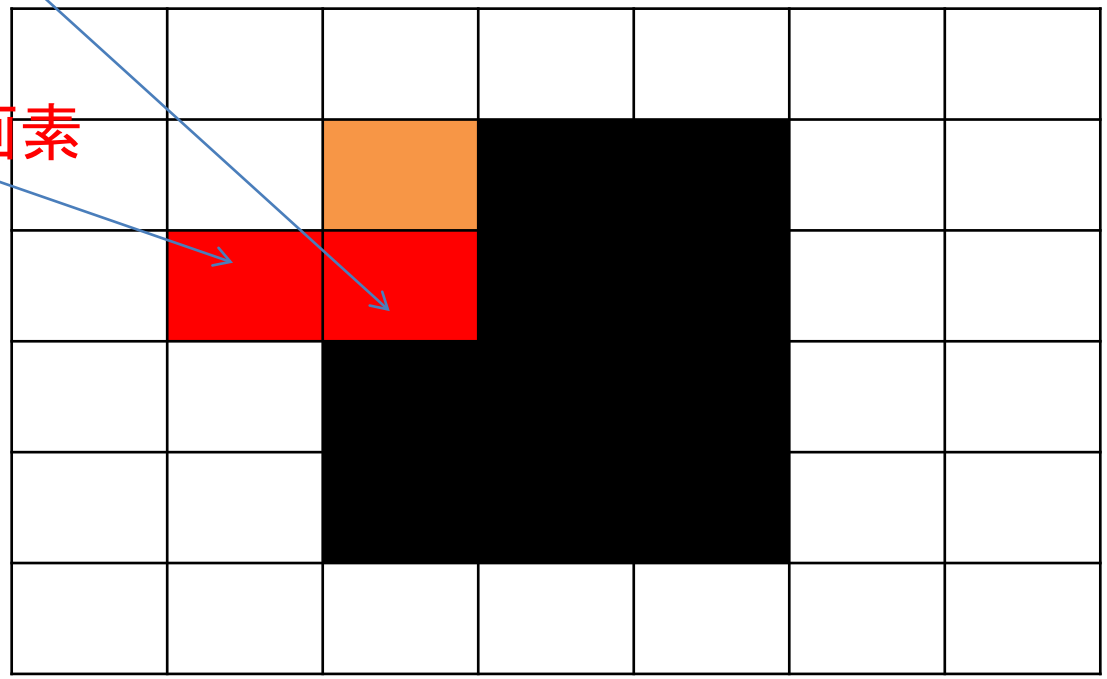
既にラベル番号が振られているので条件に掛からない

void search(int n, int x, int y, int cnum)関数

```
search( n, x, y-1, cnum );
```

着目していた画素

着目している画素

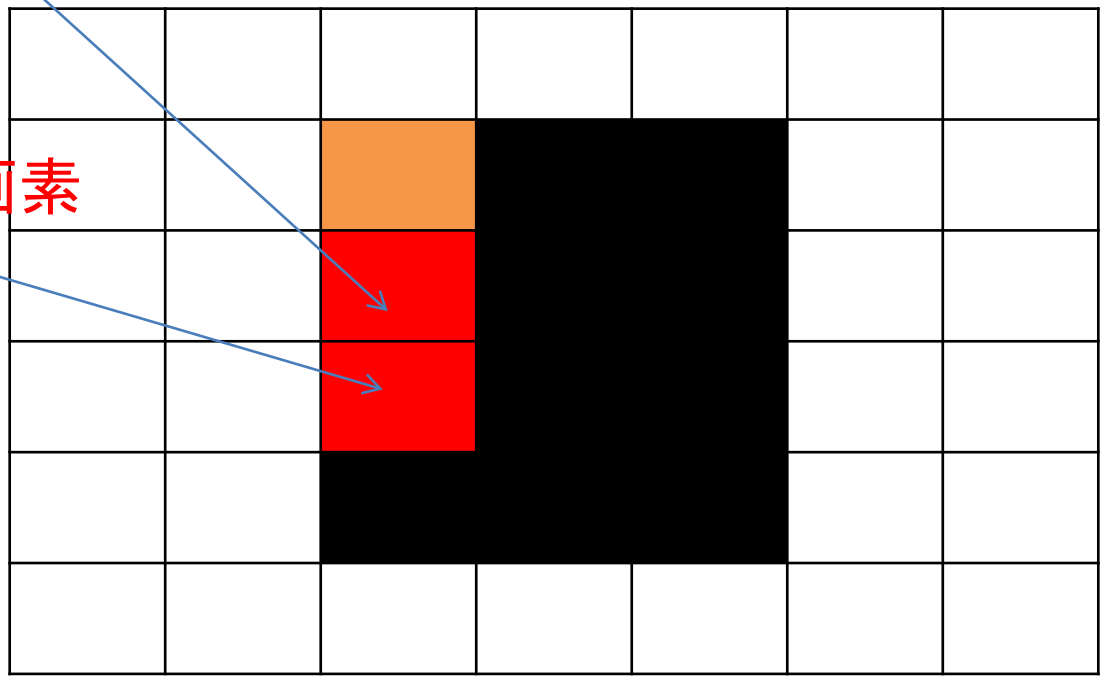


void search(int n, int x, int y, int cnum)関数

```
search( n, x, y-1, cnum );
```

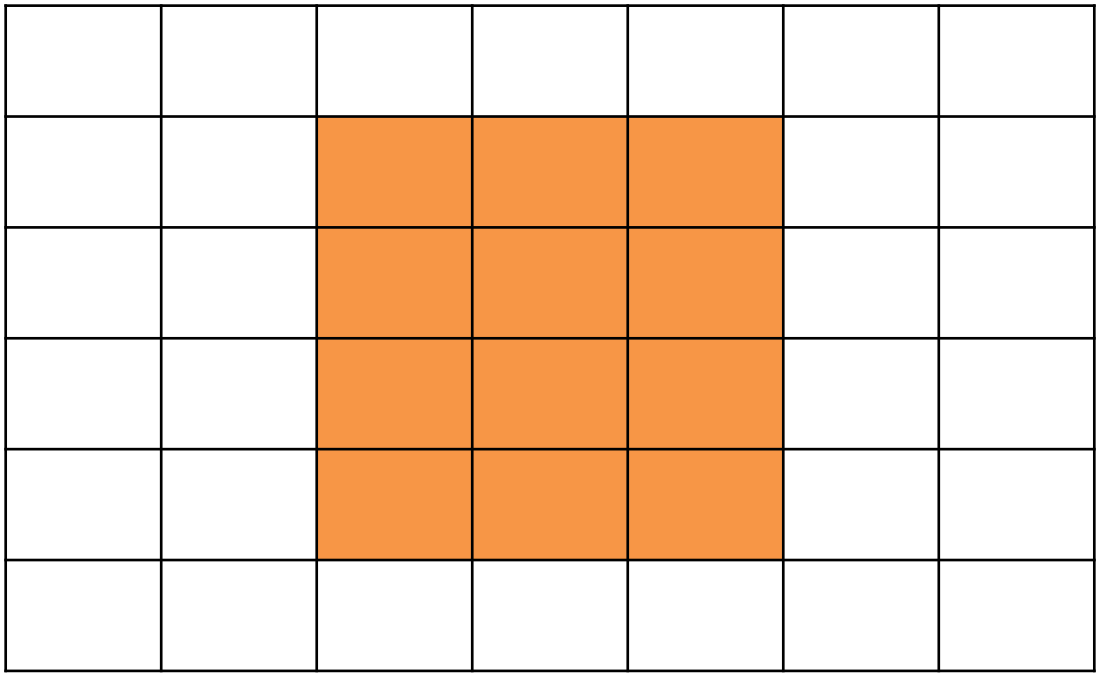
着目していた画素

着目している画素



void search(int n, int x, int y, int cnum)関数

- 繰り返すことにより，閉じられた領域についてラベリングが完了する



レポート課題

1. ヒストグラム画像を出力するプログラムを作成し実行する（既に作成済み）

   例えば，画像の明るさを変更して，処理前後のヒストグラムの様子を比較する

2. 平滑化を行うプログラムを作成し実行する（既に作成済み）

   例えば，意図的にノイズを含ませたデータを用いて平滑化前後の画像を比較する

レポート課題

3.　ラプラシアンフィルタを作成し処理前後の画像を比較する（既に作成済み）

4. 輪郭抽出のプログラムを作成し，処理前後の画像を比較する（既に作成済み）

5. 膨張縮小処理のプログラムを作成し，動作を確認する（既に作成済み）

6. Closing処理を行い，動作を確認する（既に作成済み）

7. ラベリング処理を行い，動作を確認する（既に作成済み）

うまく実行できない場合は，考えられる原因等を述べること

加点対象

- 授業で扱っていない画像処理フィルタを作成し実行する
  - アルゴリズム
  - プログラムリスト
  - 実行結果
  - 考察