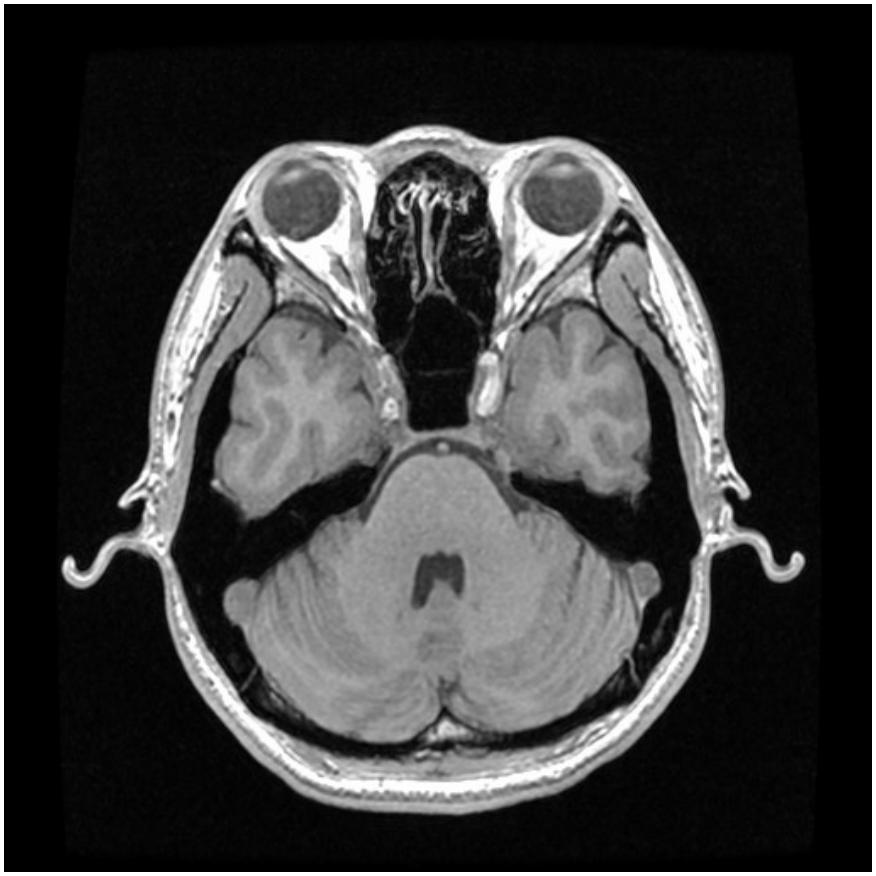
二値化画像処理（輪郭追跡など）

準備：Pgmlib.hに関数を追加

```
/* 階調画像をコピーする関数*/
void copy_image( int n1, int n2 )
/* 画像No.n1を画像No.n2にコピーする */
{
   int x, y; /* ループ変数 */

   /* 横幅，縦幅の代入 */
   width[n2] = width[n1];    height[n2] = height[n1];
   /* 階調データのコピー */
   for(y=0;y<height[n1];y++)
      for(x=0;x<width[n1];x++)
         image[n2][x][y] = image[n1][x][y];
}
```

Before



二値化処理

After

二値化画像処理

二値化することにより，輪郭の抽出等が容易になる

演習内容

- 輪郭線追跡アルゴリズム
- 膨張・収縮処理

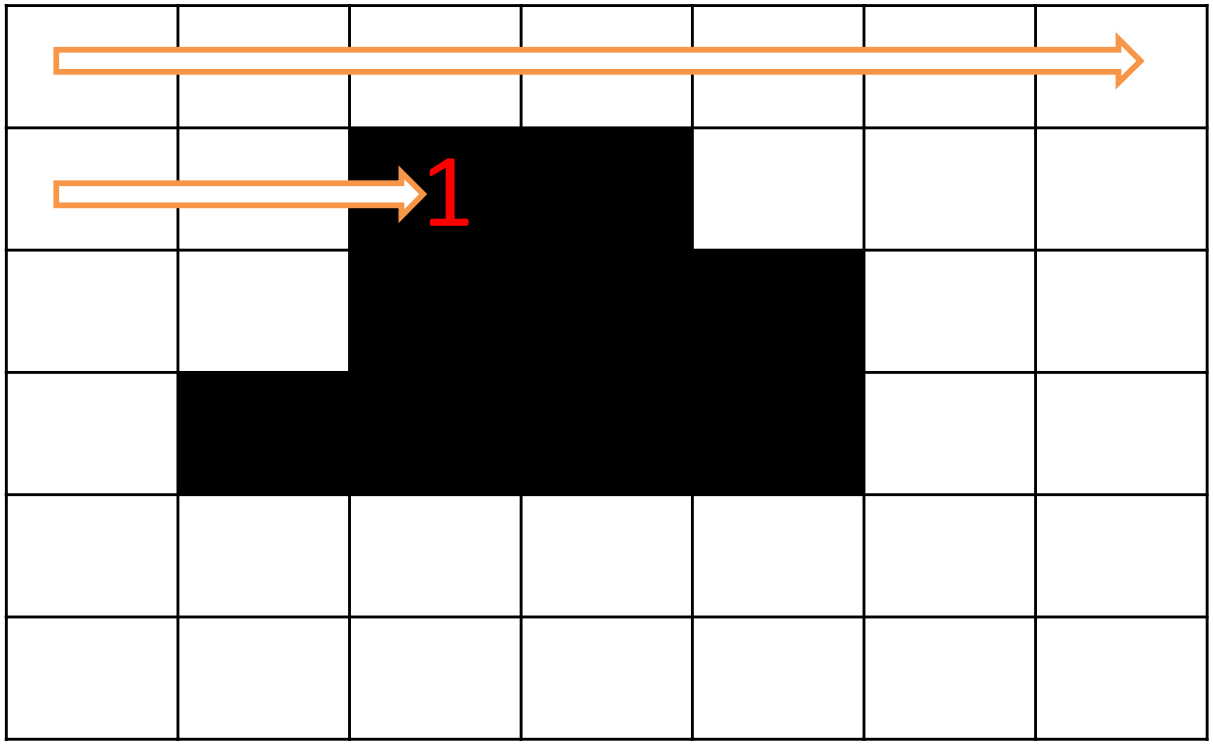
演習内容

- 輪郭線追跡アルゴリズム

- 膨張･収縮処理

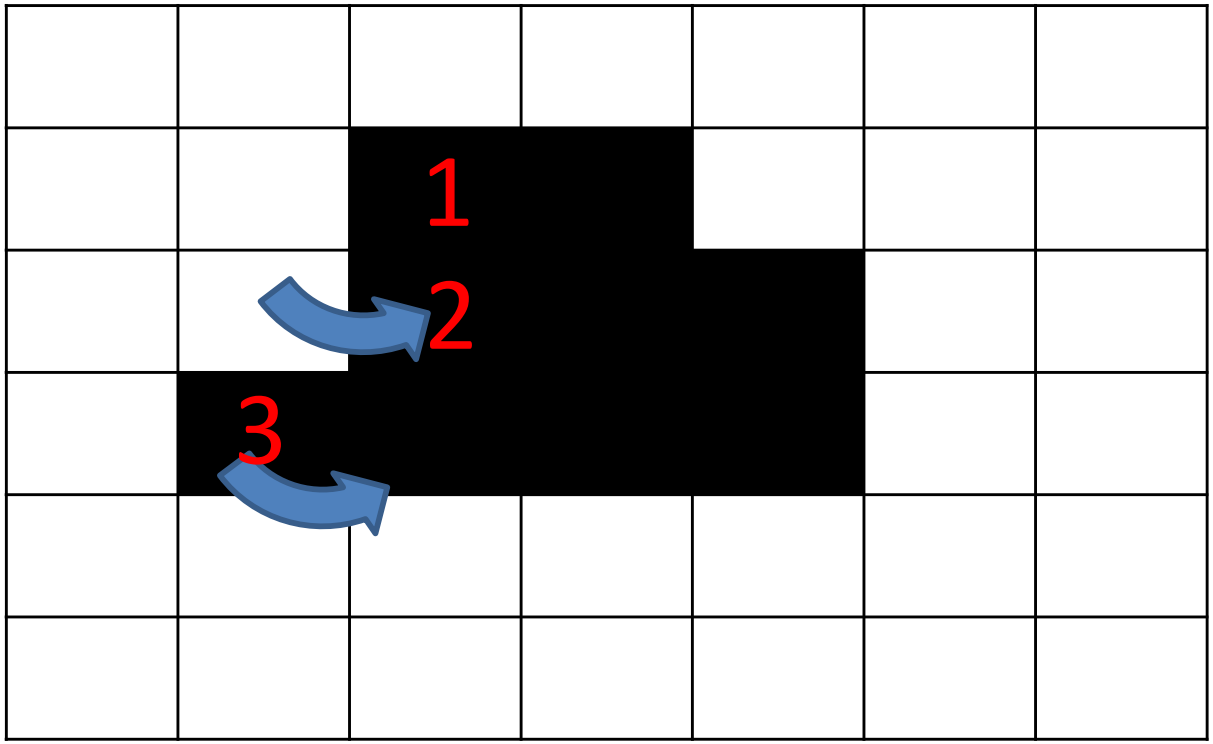
輪郭線追跡アルゴリズム

1. 原画像をラスタ走査して最初に検出した図形画素を開始点(No.1)とする.



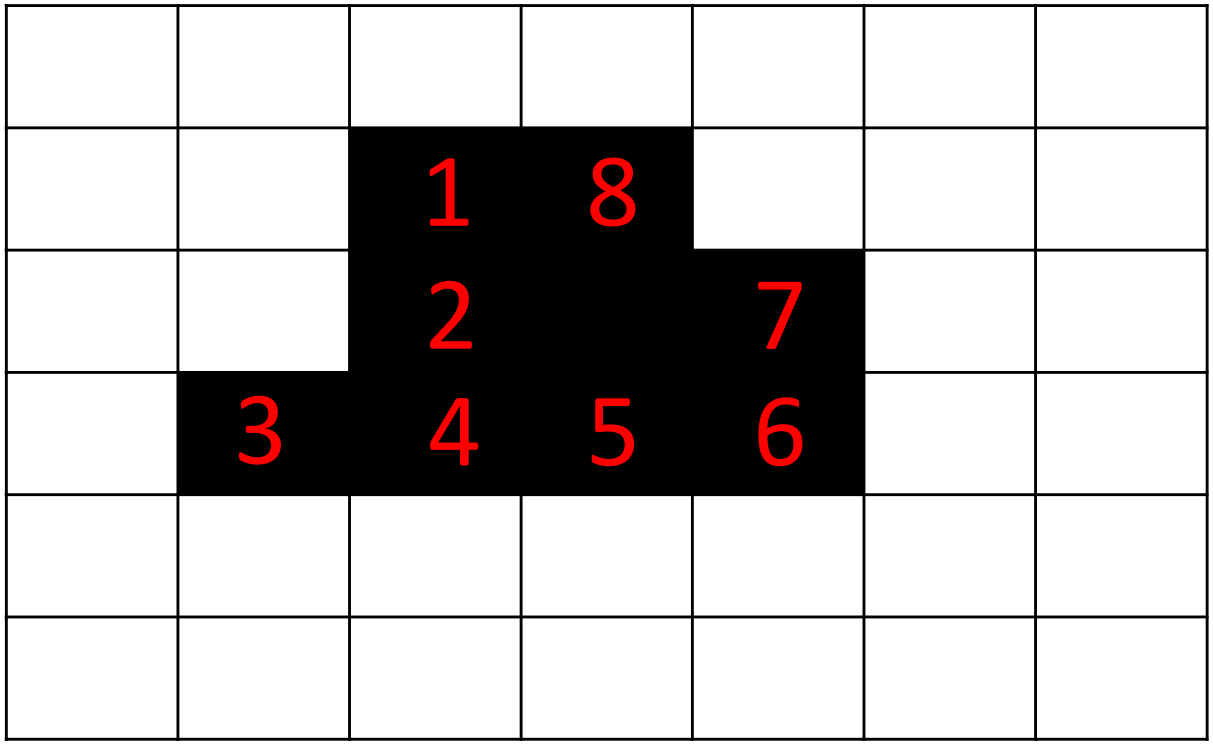
輪郭線追跡アルゴリズム

2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



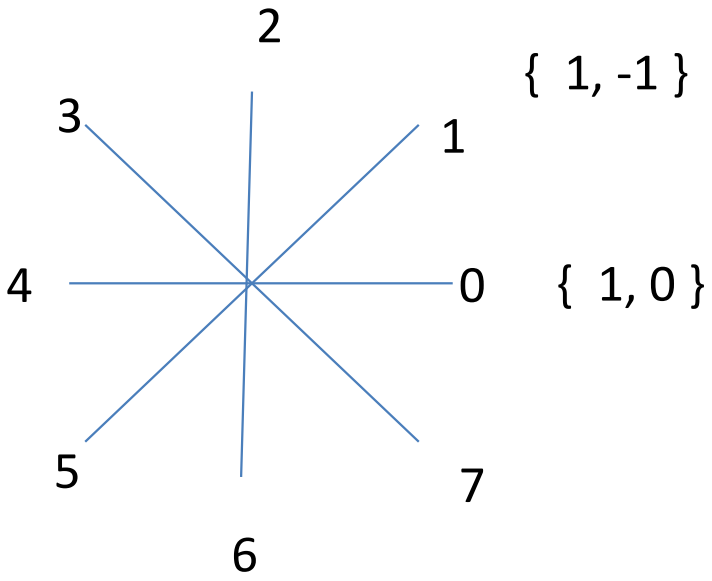
輪郭線追跡アルゴリズム

2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



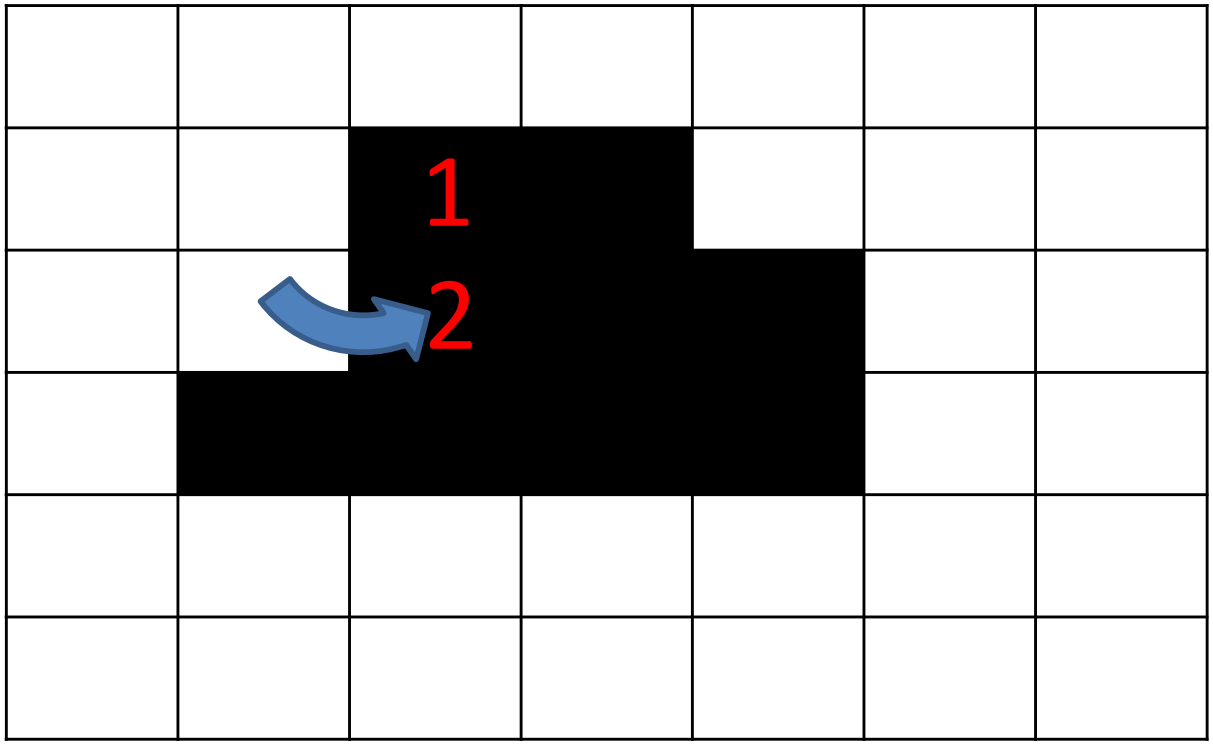
Freeman の Chain code

```
int c_code[8][2] = {  /* Freeman の Chain code */
    {  1, 0 }, { 1, -1 }, { 0, -1 }, { -1, -1 },
    { -1, 0 }, {-1,  1 }, { 0,  1 }, {  1,  1 }};
int next_code[8] = { 7, 7, 1, 1, 3, 3, 5, 5 };
```



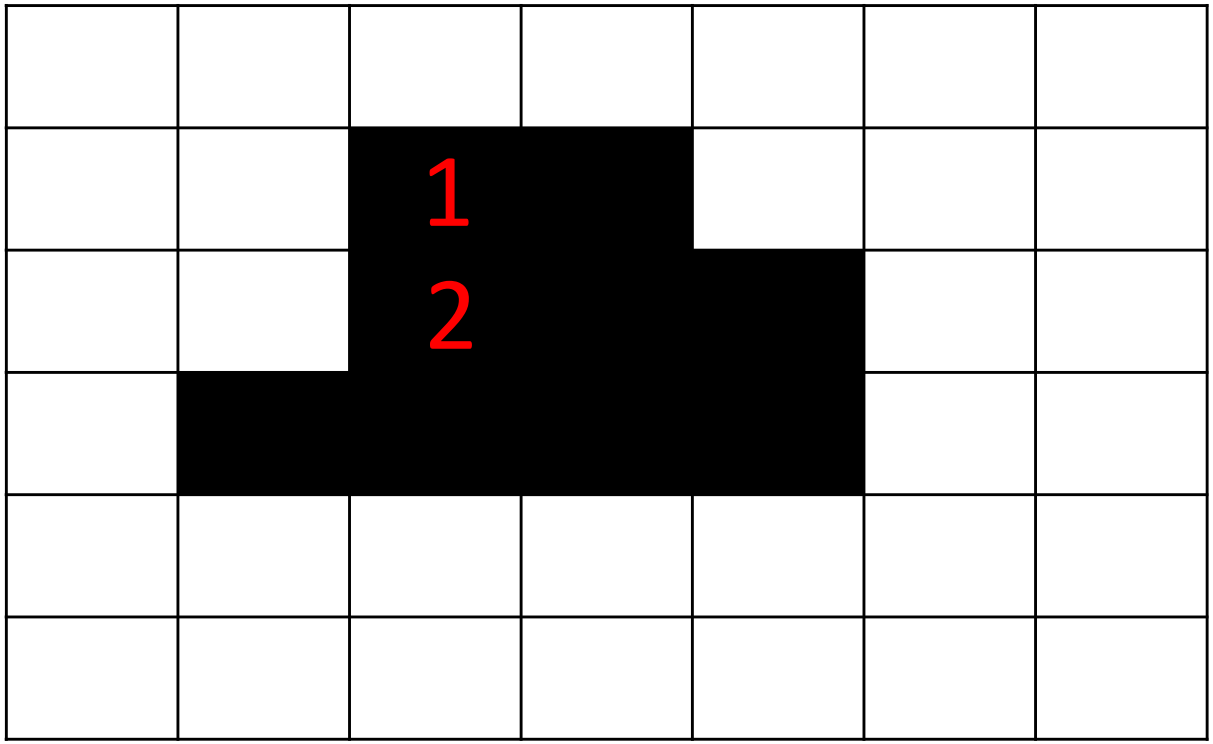
輪郭線追跡アルゴリズム

2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



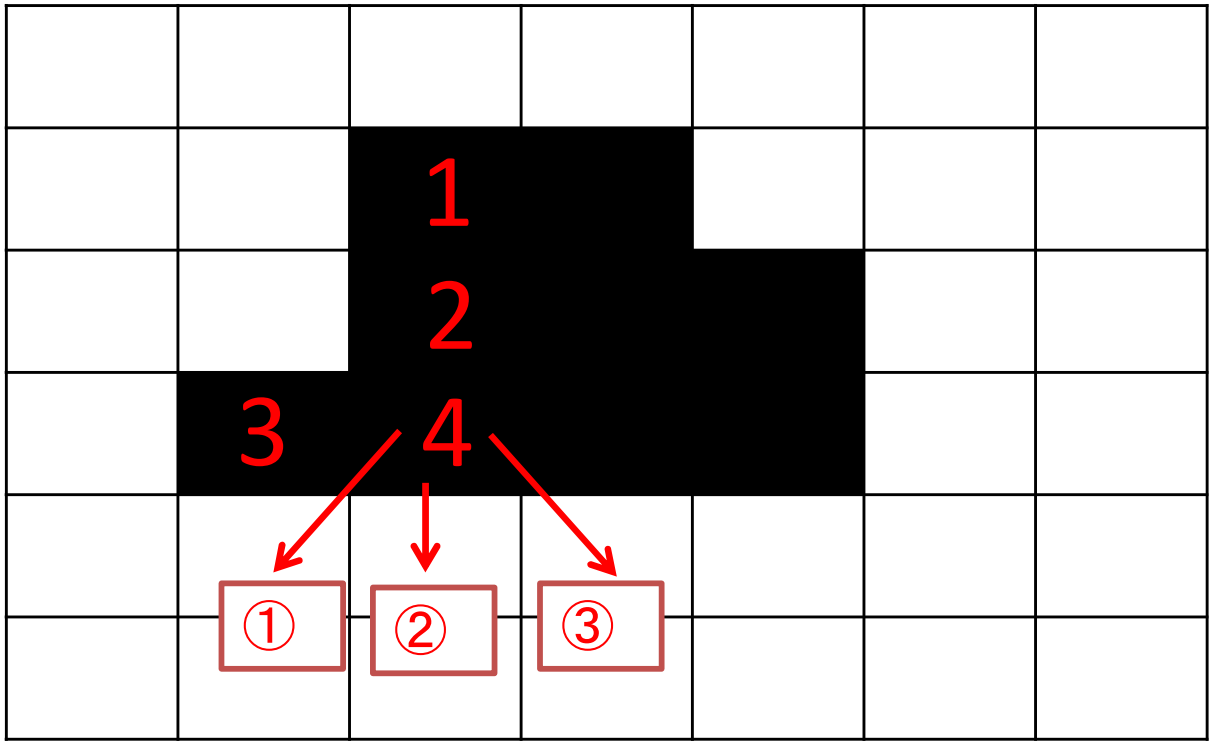
輪郭線追跡アルゴリズム

2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



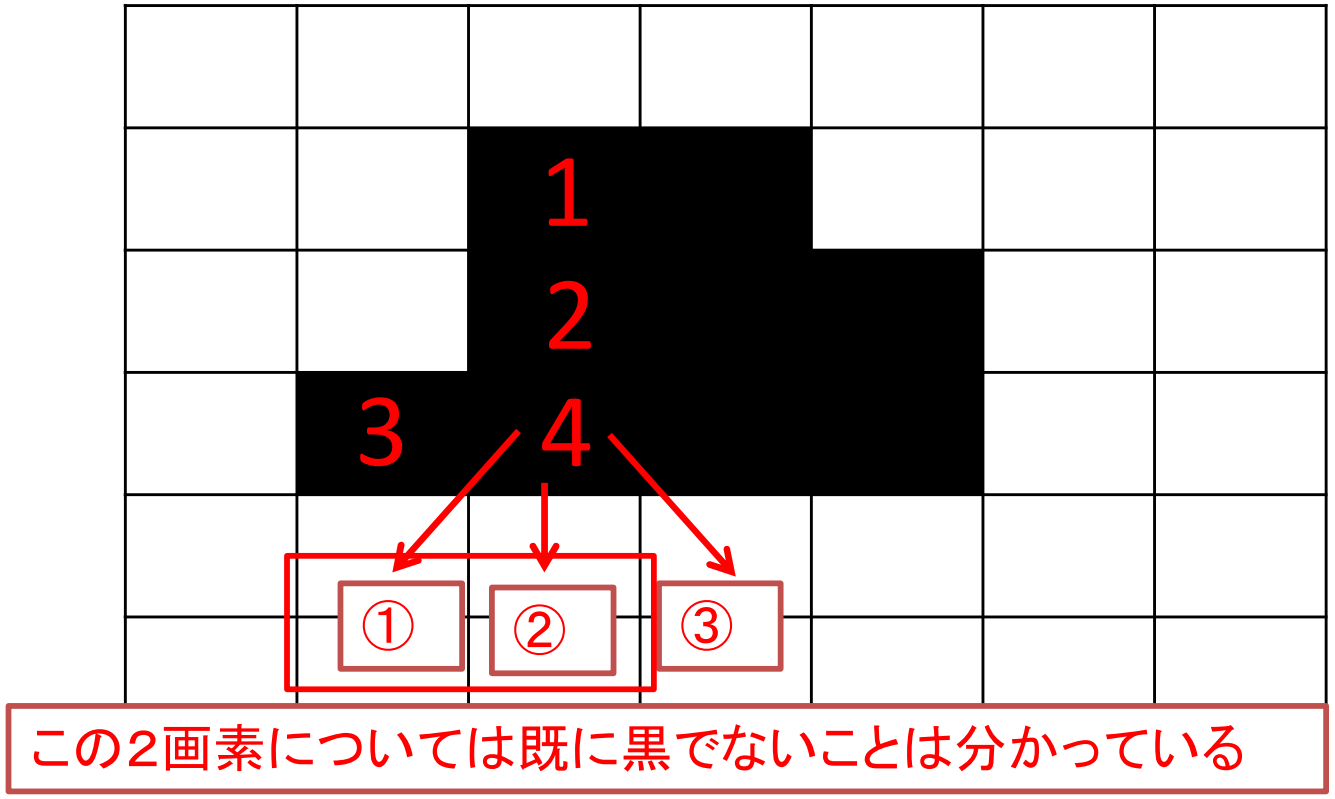
輪郭線追跡アルゴリズム

2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



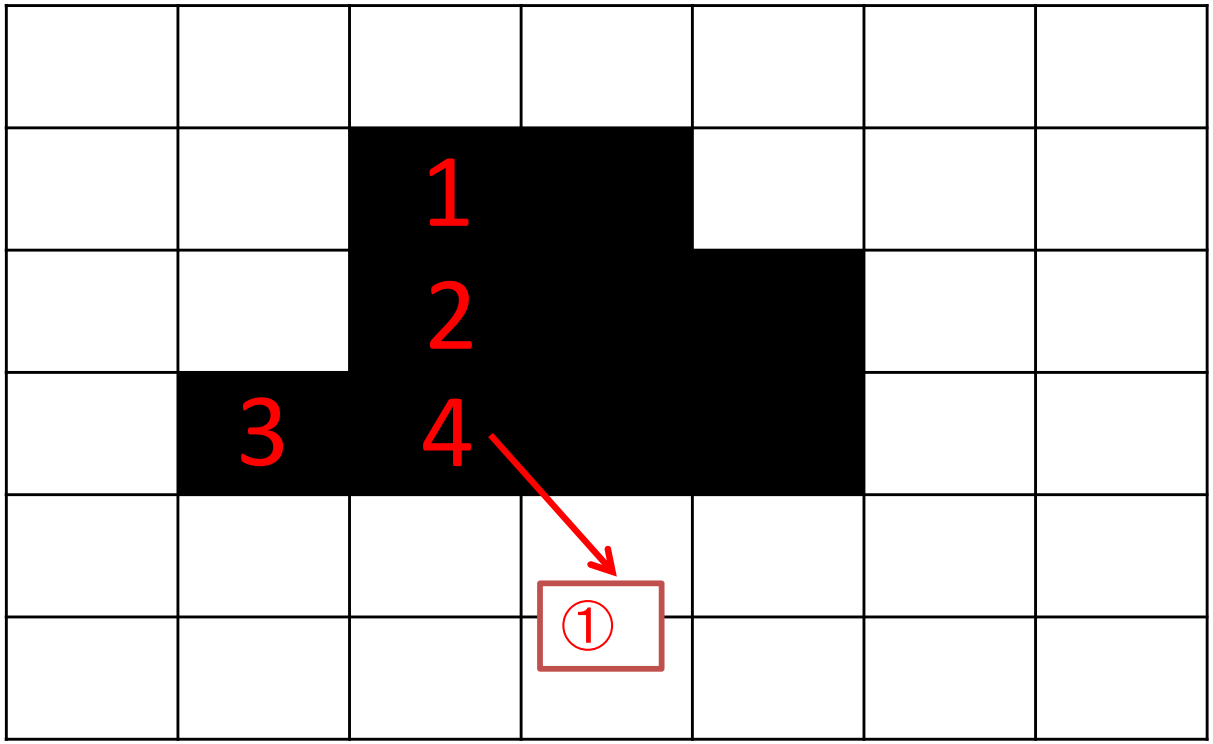
輪郭線追跡アルゴリズム

2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



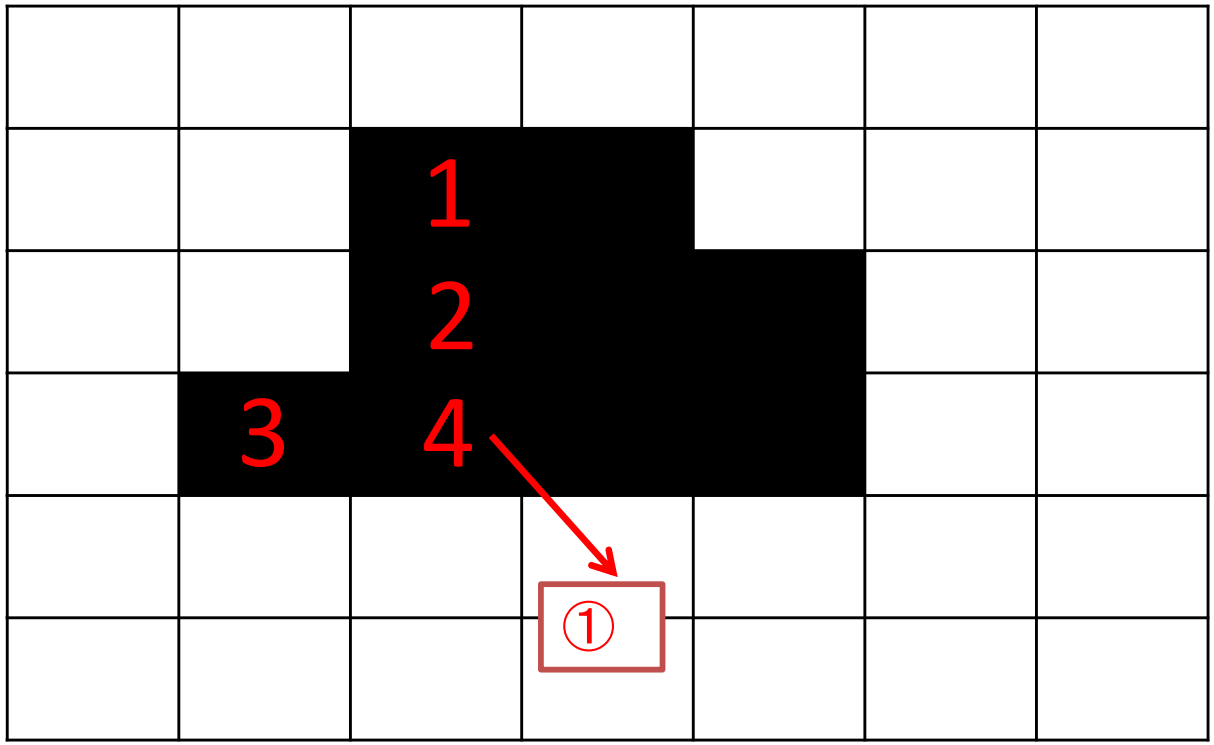
輪郭線追跡アルゴリズム

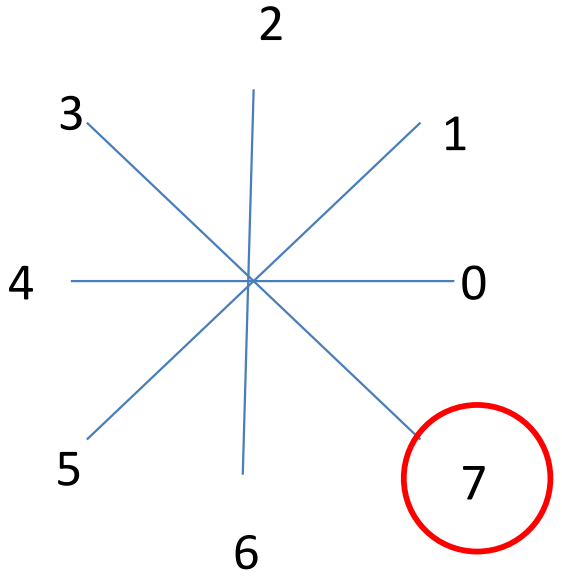
2. 開始点の画素を中心とし，左下の画素から反時計周りに8近傍内を調べて図形画素があればNo. 2とする．以下同様に辿る



Freeman の Chain code

```
int c_code[8][2] = {  /* Freeman の Chain code */
    {  1, 0 }, { 1, -1 }, { 0, -1 }, { -1, -1 },
    { -1, 0 }, {-1,  1 }, { 0,  1 }, {  1,  1 }};
int next_code[8] = { 7, 7, 1, 1, 3, 3, 5, 5 };
```

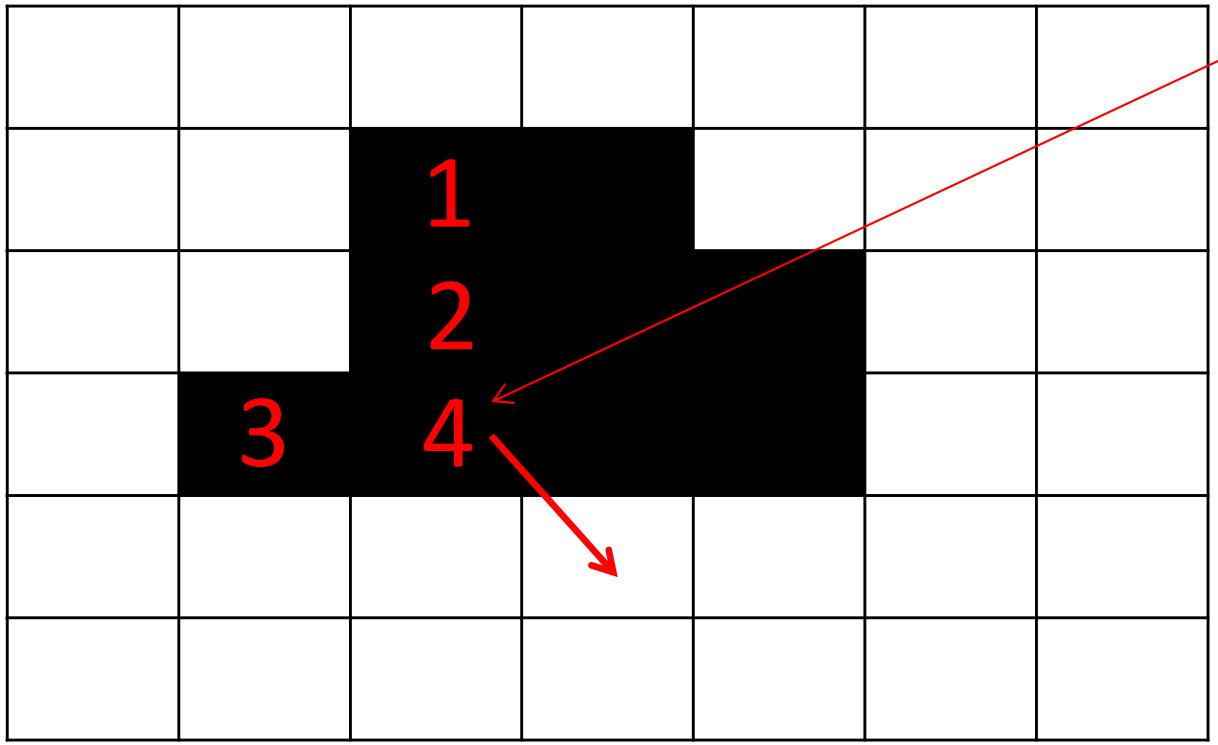




7から反時計回りに回転させればよい

Freeman の Chain code

```
int c_code[8][2] = {  /* Freeman の Chain code */
    {  1, 0 }, { 1, -1 }, { 0, -1 }, { -1, -1 },
    { -1, 0 }, {-1,  1 }, { 0,  1 }, {  1,  1 }};
int next_code[8] = { 7, 7, 1, 1, 3, 3, 5, 5 };
```



c_code[0]が黒

次の探索開始は
c_code[7]からで良い

c = next_code[c];
/* 次の探索方向を代入 */

サンプルプログラム

注意：白(脳）領域の輪郭線

main関数

```
char file[256]=“”;
printf("二値化を行う画像を指定:¥n");
load_image( 0, file );  /* ファイル → 画像No.0(原画像)  */
binarize(0);  //以前作成済み
detect_boundary( 0,1 ); /* 輪郭線を求めて保存する */
return 0;
```

detect_boundary関数

```
void detect_boundary( int n,int n2)
/* 画像No.n中の白い図形の輪郭線を求めて保存する */
{
   int c_code[8][2] = {  /* Freeman の Chain code */
    {  1, 0 }, { 1, -1 }, { 0, -1 }, { -1, -1 },
    { -1, 0 }, {-1,  1 }, { 0,  1 }, {  1,  1 }};
    int next_code[8] = { 7, 7, 1, 1, 3, 3, 5, 5 };

    int x=0,y=0,xstart,ystart,xs,ys,xp,yp;
    FILE *fp;
```

```
int c,cs,found=0,count=0;
width[n2] = width[n];
height[n2] = height[n];
/* 図形要素の輪郭線を1つだけ求めている */
do{
   if ( image[n][x][y] == □□□ ){  /* 白画素を検出 */
      count=1; /* 輪郭線の画素総数 */
      xstart=x;  ystart=y;  /* 探索開始点 */
      xs=xstart;  ys=ystart;  c=□; /* ＝初期探索方向 */
      do{
         found=0;  cs=c;
         do{
            xp = xs + c_code[c][0];
            yp = ys + c_code[c][1];
```

```
        if ( inside(n,xp,yp) && image[n][xp][yp]==□□□ )
            found=1;
        else { c++;   if ( c > □ ) c=0; }
    }while( found == 0 && c != cs );
    /* 次の画素を検出した場合，コードを出力 */
    if ( xp != xstart || yp != ystart ){
        image[n2][xp][yp] = □□□;
        count++; /* 総点数 */
    }
    xs = □;  ys = □;  /* 現在の点を移動 */
    c = next_code[c];   /* 次の探索方向を代入 */
  }while( (xp!=xstart || yp!=ystart) && found==1 );
}
```

```
        x++;  if ( x == □□□□){ x=0; y++; }
    }while( y < height[n] && found==0 );
            if ( count > 0 ){
                    printf("輪郭線の構成点数＝%d¥n",count);
                    printf("求めた輪郭線を保存するときのファイル
名 : ¥n");
                    save_image(n2,"");
                    printf("輪郭線は保存されました¥n",str);
    } else printf("画像中に黒画素は検出されませんでした¥n");
}
```

int inside(int n, int x, int y)関数

```
int inside( int n, int x, int y )
/* 内外部判定 */
{
   if ( x >= 0 && x < □□□□ && y >= 0 && y < □□□□ )
   return □;  else  return 0;
}
```

演習1

- 上記のアルゴリズムに従って，輪郭線追跡を行うプログラムを作成せよ
- ただしmain関数は以下の通りとせよ

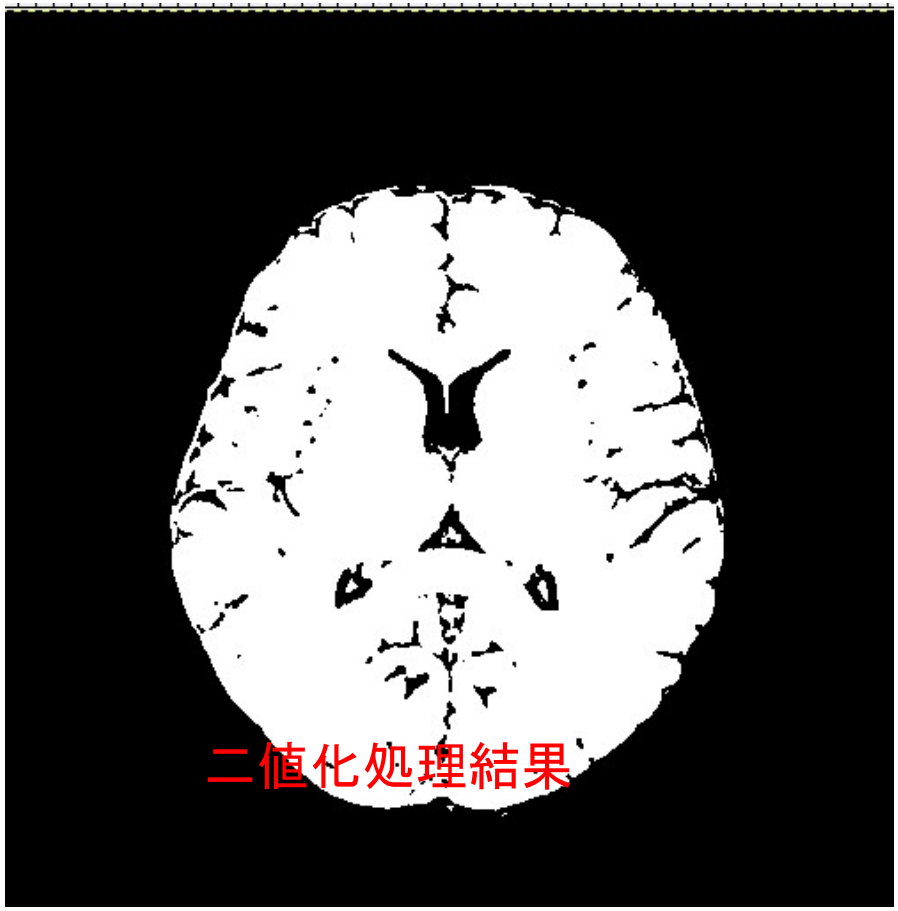
```
printf("二値化を行う画像を指定:¥n");
load_image(0,"");
binarize(0);  //以前作成済み
save_image(0,"");
detect_boundary(0,1);
//輪郭線を求めて1番目のデータに保存
return 0;
```

実行結果

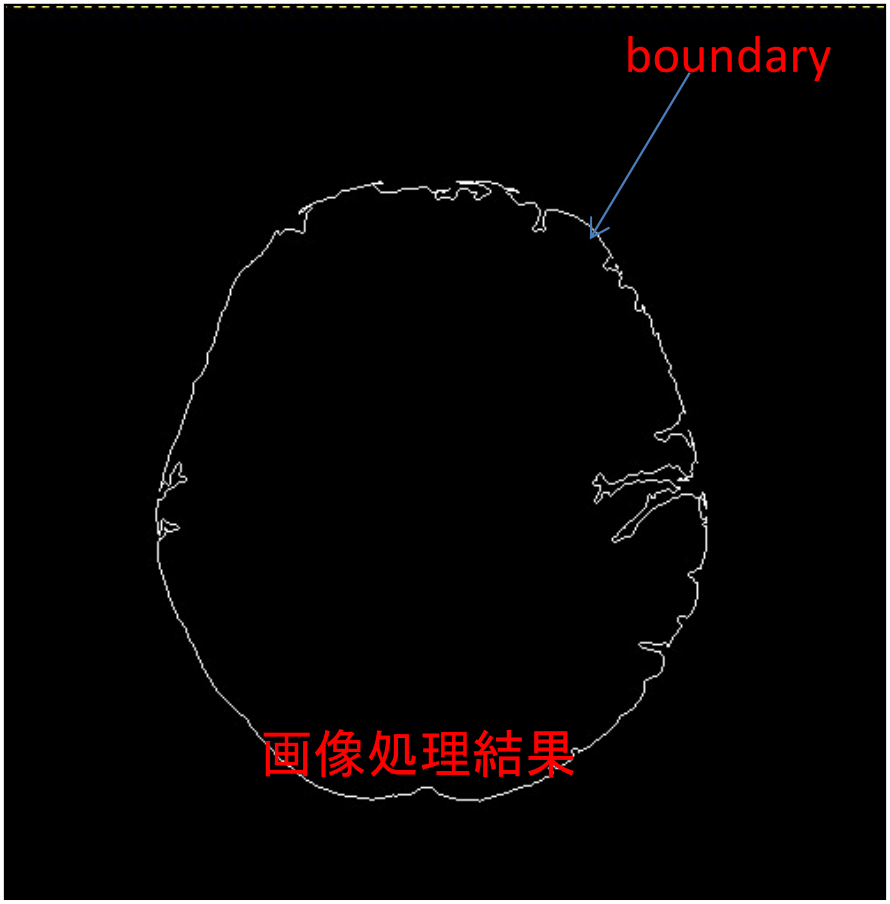
二値化を行う画像を指定:
入力ファイル名 (*.pgm) : mri4.pgm
横の画素数 = 512, 縦の画素数 = 512
最大階調値 = 255
画像は正常に読み込まれました.
２値化のしきい値（0-255）: 128
出力ファイル名 (*.pgm) : mri4-out1.pgm
画像は正常に出力されました.
輪郭線の構成点数＝1552
求めた輪郭線を保存するときのファイル名 :
出力ファイル名 (*.pgm) : mri4-out2.pgm

実行結果

二値化画像



輪郭線画像

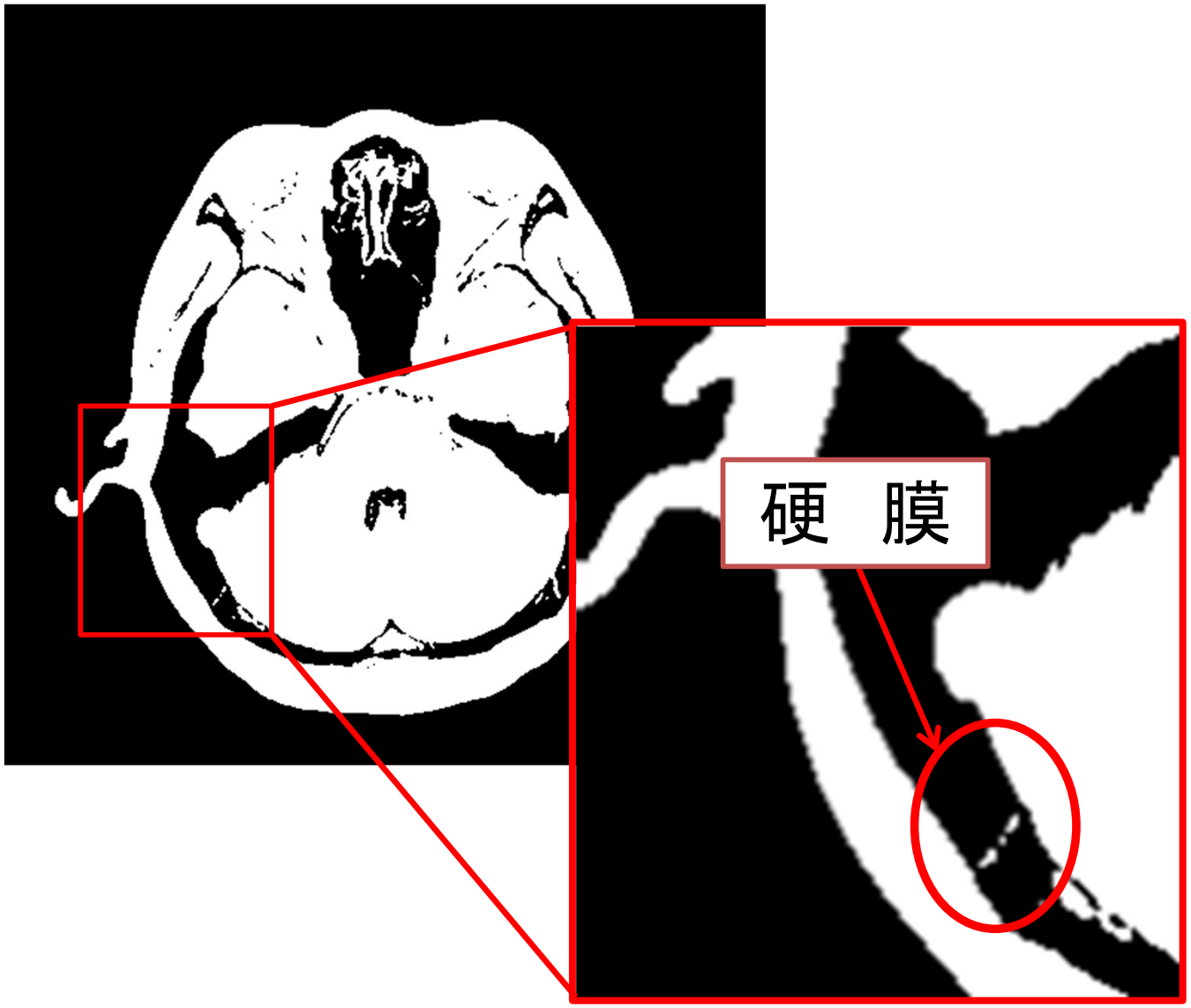


演習内容

- 輪郭線追跡アルゴリズム

- 膨張・収縮処理

二値化画像処理



膨張/縮小処理

脳領域でも頭蓋でもない

↓

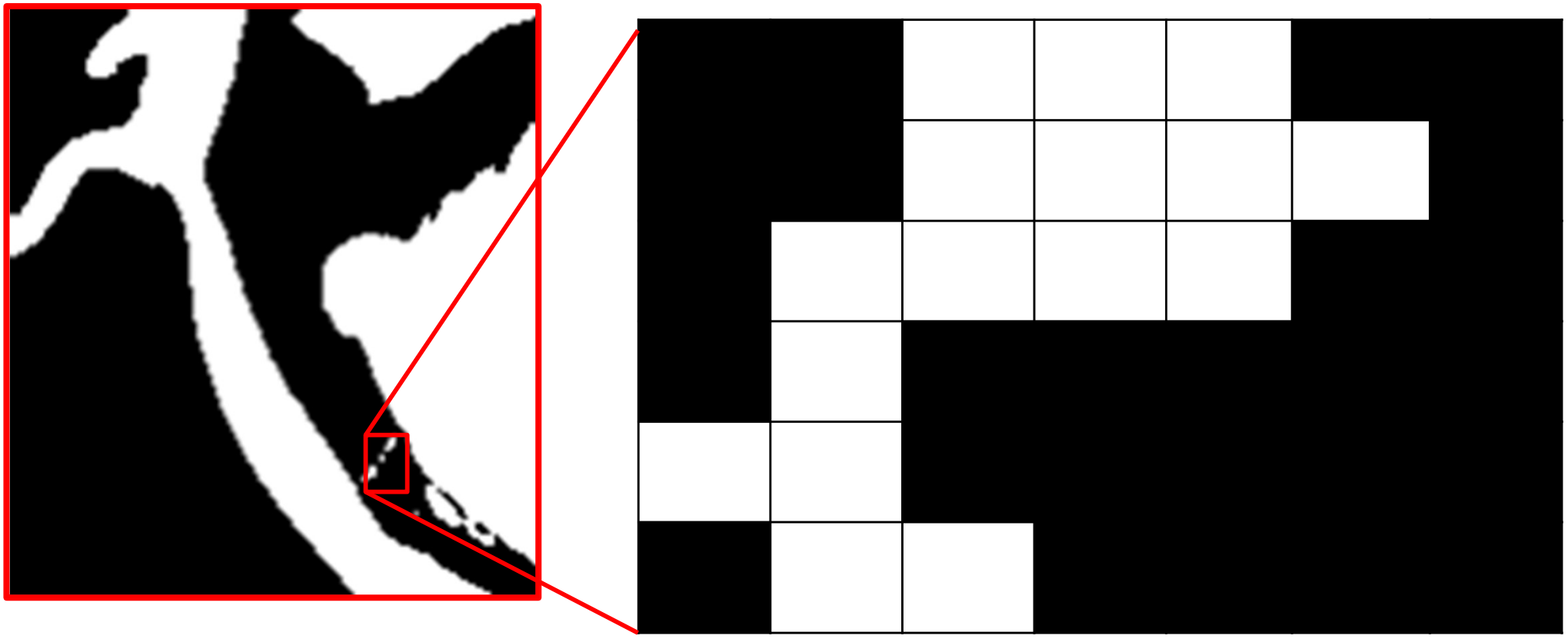
領域判別を行うとき誤判別を行う可能性が高い

↓

膨張縮小処理

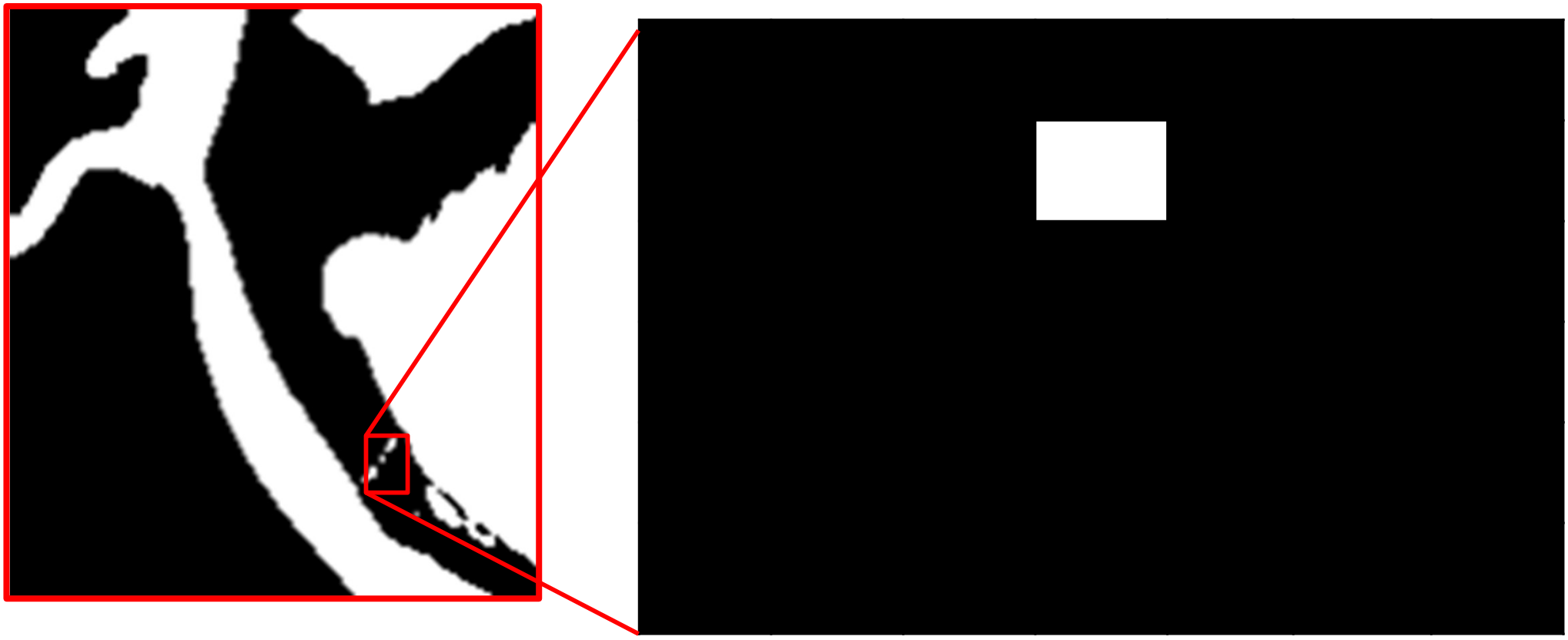
縮小処理

- 8近傍中に1つでも背景画素（黒）を含む図形画素（白）を背景画素（黒）にする処理



縮小処理

- 再度，収縮処理を実行する



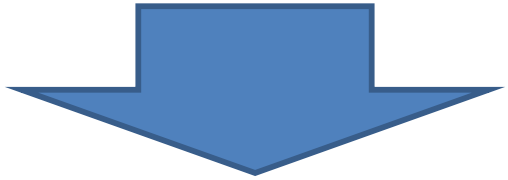
縮小処理

- 再度，収縮処理を実行する
- 収縮処理の回数は任意

硬膜が除去できる

膨張処理

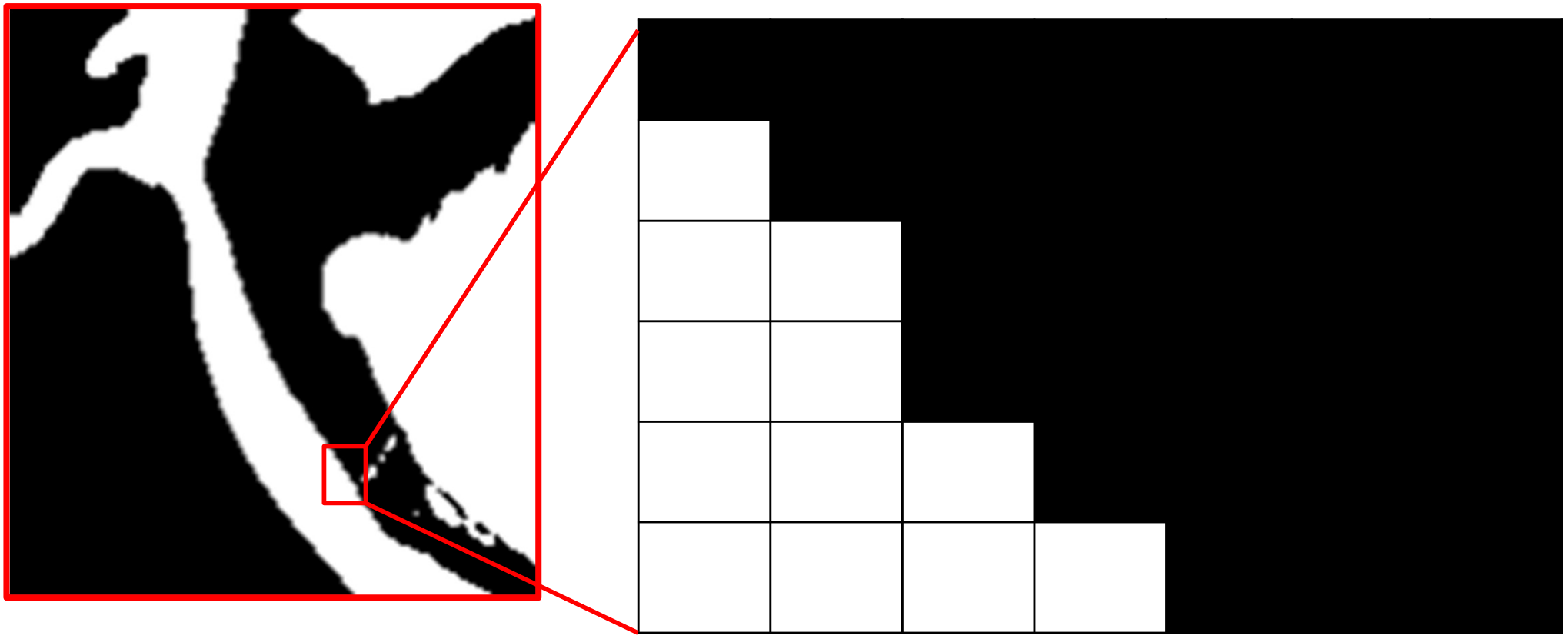
- 全ての画素に適用しているため，脳や頭蓋等の領域も縮小してしまっている



- 膨張処理を行い元の画像に近づける

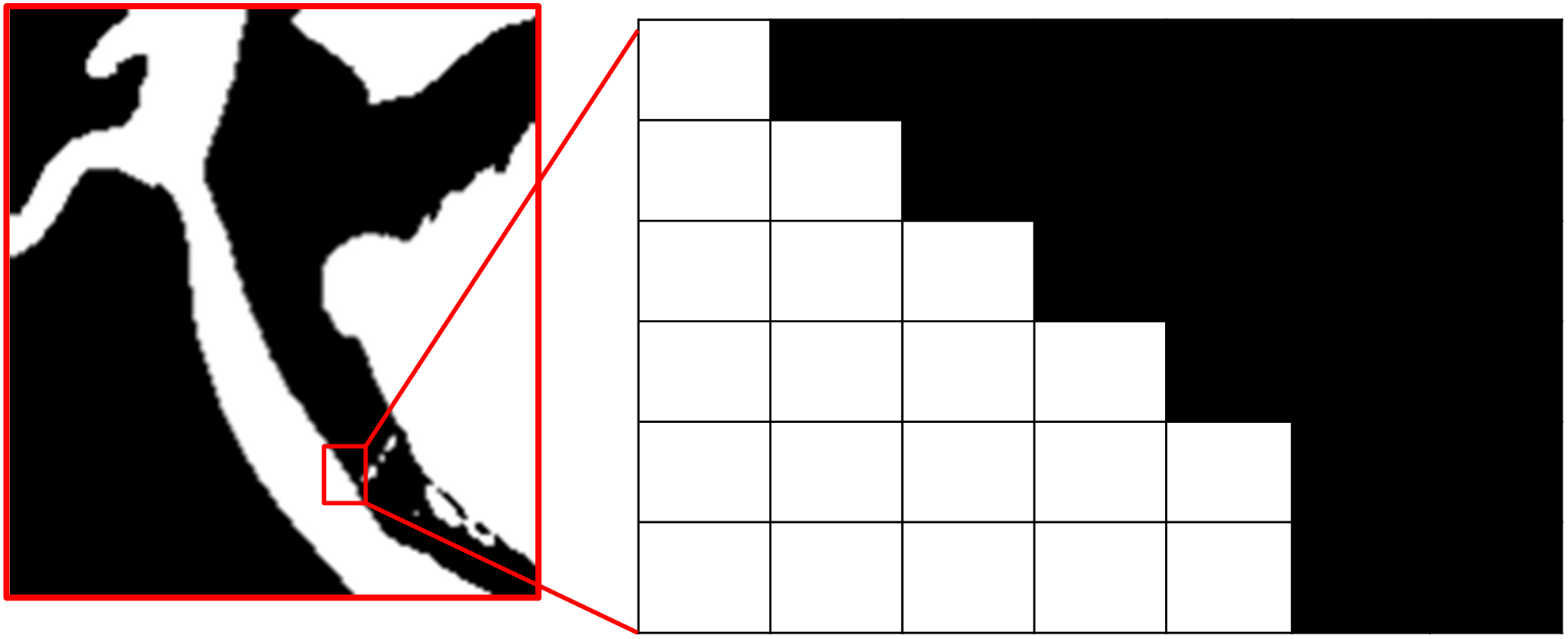
膨張処理

- 8近傍中に1つでも図形画素（白）を含む背景画素（黒）を図形画素（白）にする処理



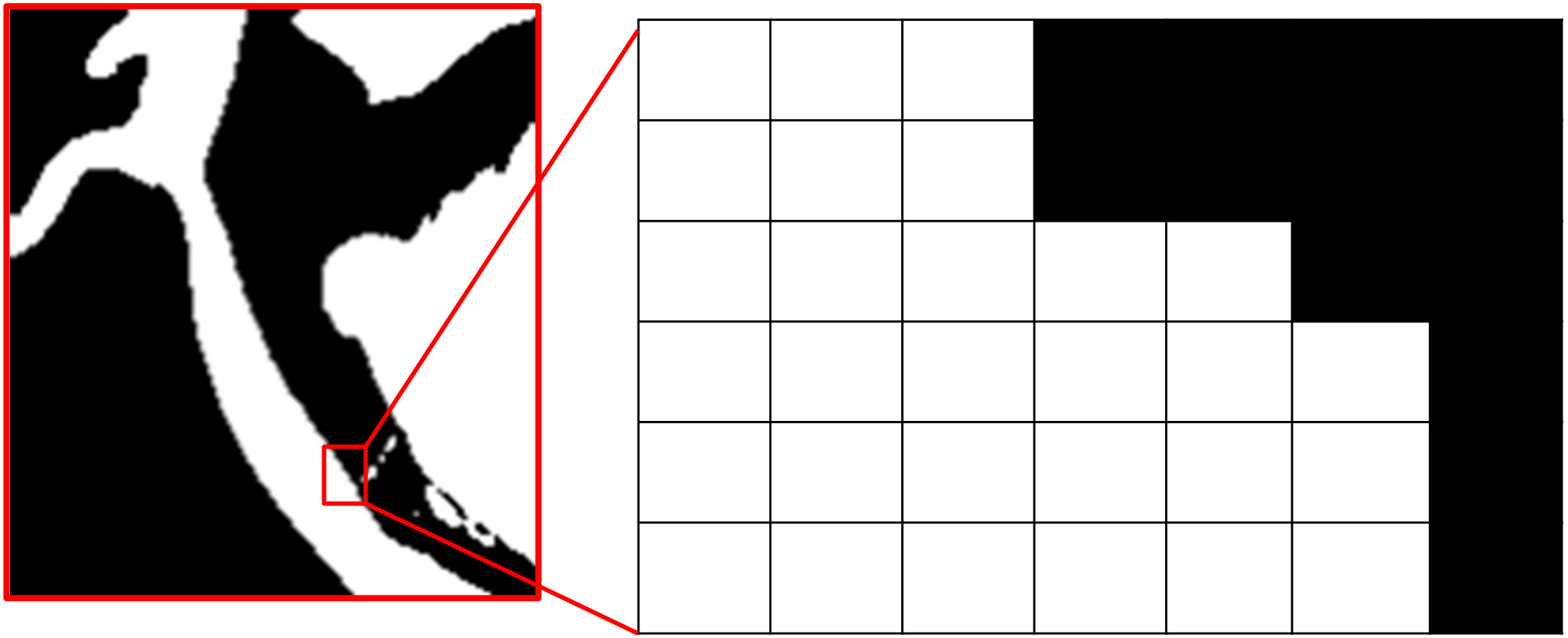
膨張処理

- 8近傍中に1つでも図形画素（白）を含む背景画素（黒）を図形画素（白）にする処理



膨張処理

- ２回収縮処理をしたので２回膨張処理を行う



演習2

上記のアルゴリズムに従って，膨張・縮小を行うプログラムを作成せよ

ただしmain関数は以下の通りとせよ

```
printf("二値化を行う画像を指定:¥n");
load_image(0,"");
binarize(0);  //以前作成済み
save_image(0,"");

dilatation(0, 0, 1);
dilatation(1, 1, 2);

save_image(2,"");
```

演習2

上記のアルゴリズムに従って，膨張・縮小を行うプログラムを作成せよ

ただしmain関数は以下の通りとせよ

```
printf("二値化を行う画像を指定:¥n");
load_image(0,"");
binarize(0);  //以前作成済み
save_image(0,"");

dilatation(0, 0, 1);   //0は黒画素を膨張：脳を縮小
dilatation(1, 1, 2);   //1は白画素を膨張：脳を膨張

save_image(2,"");
```

```
int count( int n, int x, int y, int col )
/* No.nの画像の座標(x,y)の8近傍中のcol色の画素数を返す */
{
   int s,t,xp,yp,counter=0;

   for(t=-1;t<2;t++)
      for(s=-1;s<2;s++){
         xp = x + s;  yp = y + t;
         if ( xp >= 0 && xp < width[n]
           && yp >= 0 && yp < height[n]
           && image[n][xp][yp] == col )
             counter++;
      }
   return counter;
}
```

```
void dilatation( int col, int n1, int n2 )
/* ２値画像(No.n1)の階調colの図形を膨張させてNo.n2へ */
{
   int x,y,i,n,col1,col2;
   if (col==0) { col1=255;  col2=0; } else { col1=0;  col2=255; };
   /* 膨張させる回数を指定する */
   printf("縮小/膨張処理を行う回数 : ");  scanf("%d",&n);
   /* 膨張処理を施した画像をNo.n2に作る */
   copy_image( n1, n2 );  /* ← pgmlib.h 内の関数 */
   for(i=0;i<n;i++){
      /* 膨張処理後の画像をNo.n2へ */
      for(y=0;y<height[n1];y++)
         for(x=0;x<width[n1];x++)
            if ( image[n1][x][y] == col1 && count(n1,x,y,col2) > 0 )
               image[n2][x][y] = col2;
      /* No.n2 を No.n1 へコピーする */
      copy_image( n2, n1 );  /* pgmlib.h 内の関数 */
   }
}
```