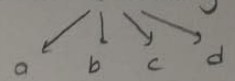
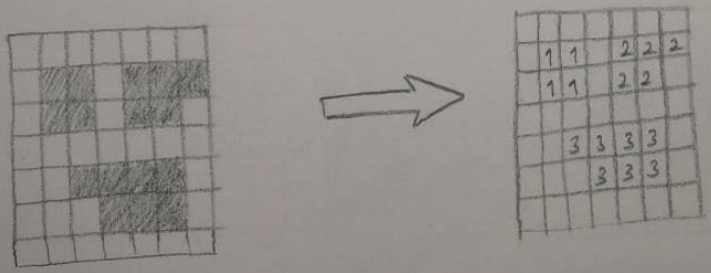
1-)

① There are $2^4 = 16$ possible states for the 4 binary variables

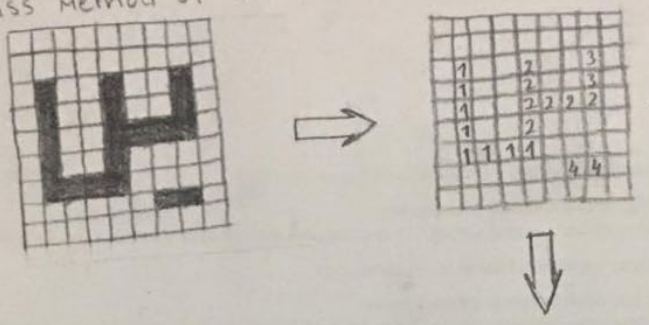


a	b	c	d	Label Assignments	Equivalences
0	0	0	0	–	–
0	0	0	1	$\angle(p) = \angle(d)$	–
0	0	1	0	$\angle(p) = \angle(c)$	–
0	0	1	1	$\angle(p) = \angle(c)$	$\angle(d) = \angle(c)$
0	1	0	0	$\angle(p) = \angle(b)$	–
0	1	0	1	$\angle(p) = \angle(b)$	$\angle(d) = \angle(b)$
0	1	1	0	$\angle(p) = \angle(b)$	$\angle(c) = \angle(b)$
0	1	1	1	$\angle(p) = \angle(b)$	$\angle(d) = \angle(c) = \angle(b)$
1	0	0	0	$\angle(p) = \angle(a)$	–
1	0	0	1	$\angle(p) = \angle(a)$	$\angle(d) = \angle(a)$
1	0	1	0	$\angle(p) = \angle(a)$	$\angle(c) = \angle(a)$
1	0	1	1	$\angle(p) = \angle(a)$	$\angle(d) = \angle(c) = \angle(a)$
1	1	0	0	$\angle(p) = \angle(a)$	$\angle(b) = \angle(a)$
1	1	0	1	$\angle(p) = \angle(a)$	$\angle(d) = \angle(b) = \angle(a)$
1	1	1	0	$\angle(p) = \angle(a)$	$\angle(c) = \angle(b) = \angle(a)$
1	1	1	1	$\angle(p) = \angle(a)$	$\angle(d) = \angle(c) = \angle(b) = \angle(a)$



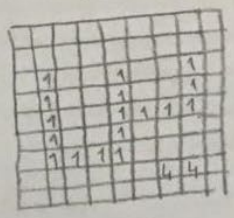
2nd example

2 Pass Method of Connected Component Labeling



Label mask after first pass

- center point of the mask gets the lowest label of pixels in the mask connected with it.
- If there is no pixel that is connected with center point new label is assigned.



Label mask after second pass

- When we start our analysis by assuming that p is ⊥, one of the parameters, a, b, c or d must be ⊥ thus, we can talk about label assignment. After that, we check in order from a to d, the parameter that gets ⊥ first, label assignment is assigned there.
  - If we examine the 0000 status, there is no label assignment or equivalance because it has no value of ⊥.
  - If we examine the 1001 status, the label assignment becomes $L(p) = L(a)$ since a is the first one that takes the value ⊥, but since d is equal to ⊥ it is specified as $L(d) = L(a)$ in the equivalences section

2-)

```
// gcc hw1.c -o hw1 img_pro.c -lm
// ./ hw1 cathedral.pgm

#include "img_pro.h"

int main(int argc , char **argv)
{
unsigned char **img , **img2 , **img3;
char *pgm_file;
int i , j , NC , NR , count = 0;

if(argc!=2)
{
  printf("\nUsage: hw1 [Image file (*.pgm)]\n");
  printf("\nE.g.   hw1 cathedral.pgm \n");
  exit(-1);
}

pgm_file = argv[1]; // 2rd input for which image should be corrected

img = pgm_file_to_img(pgm_file , &NC , &NR); // to save uncorrected pixel values
show_pgm_file(pgm_file); // shows uncorrected image

img2 = alloc_img(NC , NR); // to allocate space for corrected pixel values (same size with img)
img3 = alloc_img(NC , NR); // to allocate space for corrected pixel values (same size with img)

//--------------PART - 1----------------

for(i = 0 ; i < NR ; i++) // for threshold "128" processed image (It can be maximum of 255)(BW)
{
            for(j = 0 ; j<NC ; j++)
```

```
	{
		if(127 > img[i][j])
		{
			img2[i][j] = 0;
		}
		else if(255 > img[i][j] && 127 <= img[i][j])
		{
			img2[i][j] = 255;
		}
	}
}

img_to_pgm_file(img2,"hw1.pgm",NC,NR); // Converting image(matrix) to Grayscale(actual image)
show_pgm_file("hw1.pgm"); // Displays BW image

//--------------PART - REST--------------

int  l = 1; // label counter
long int label[NR][NC];

// CCL check for a , b , c , d , p
for(i = 1 ; i < NR - 1 ; i++)
{
	for(j = 1 ; j < NC - 1 ; j++)
	{
		if(img2[i][j] == 255) // the case of, "p" == 1 on CCL
		{
			if(img2[i-1][j-1] == 255) // checking CCL for "a"
			{
				label[i][j] = l;
			}
			else if(img2[i-1][j] == 255) // checking CCL for "b"
			{
```

```
                        label[i][j] = l;
                    }
                    else if(img2[i-1][j+1] == 255) // checking CCL for "c"
                    {
                        label[i][j] = l;
                    }

                    else if(img2[i][j-1] == 255) // checking CCL for "d"
                    {
                        label[i][j] = l;
                    }
                    else // if there is no neighbor, then increment the label counter
                    {
                        l++;
                    }
                }
            }
}

// 8 neighbor connection check
/*
for(i = 1 ; i < NR - 1 ; i++)
{
            for(j = 1 ; j < NC - 1 ; j++)
            {
                if(img2[i][j] == 255) // the case of, "p" == 1 on CCL
                {
                    if(img2[i-1][j-1] == 255)
                    {
                        label[i][j] = l;
                    }
                    else if(img2[i-1][j] == 255)
                    {
```

```
            label[i][j] = l;
}
else if(img2[i-1][j+1] == 255)
{
            label[i][j] = l;
}

else if(img2[i][j-1] == 255)
{
            label[i][j] = l;
}
else if(img2[i][j+1] == 255)
{
            label[i][j] = l;
}

else if(img2[i+1][j-1] == 255)
{
            label[i][j] = l;
}
else if(img2[i+1][j] == 255)
{
            label[i][j] = l;
}
else if(img2[i+1][j+1] == 255)
{
            label[i][j] = l;
}

else // if there is no neighbor, then increment the label counter
{
            l++;
}
```

```
                        }
            }
}
*/
int counter = 0;
counter = 255 / l;
int incrementation = counter;
int val = counter;

for(i = 0 ; i < NR ; i++)
{
            for(j = 0 ; j<NC ; j++)
            {
                        img3[i][j] = label[i][j]; // assigning the labels on the image
            }
}

img_to_pgm_file(img3,"hw2.pgm",NC,NR); // Converting image(matrix) to Grayscale(actual image)
show_pgm_file("hw2.pgm"); // Displays the labels

printf("Total number of labels: %d\n\n", l);
free_img(img);
free_img(img2);

return(1);
}
```