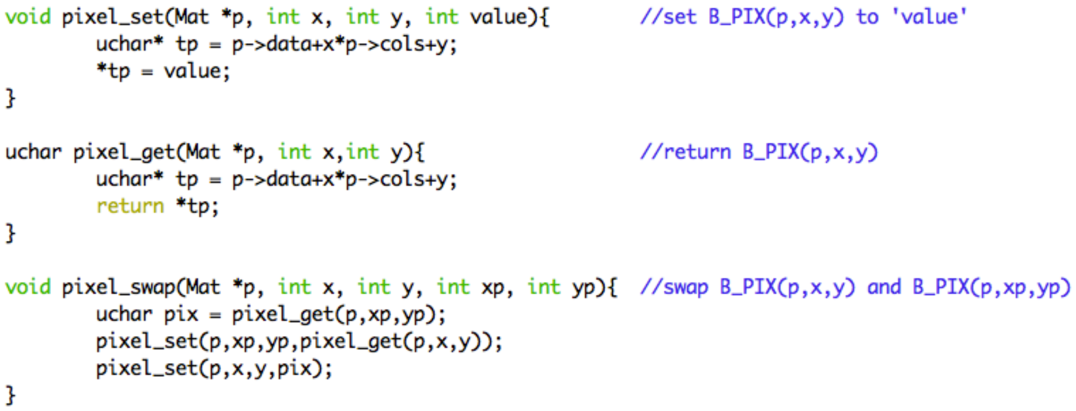
Computer Vision hw_2

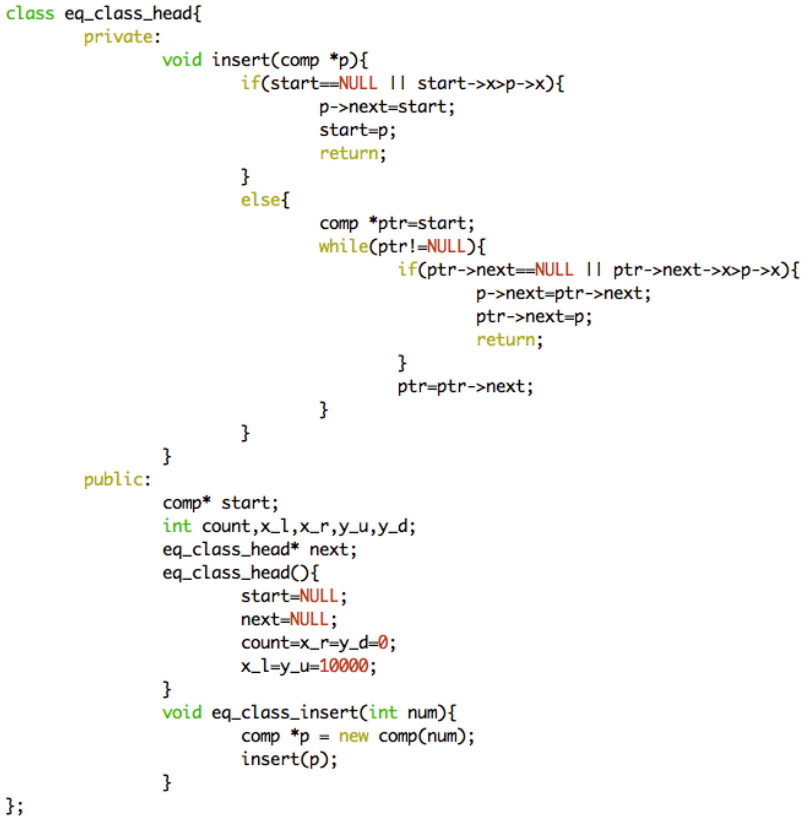
By R01922124 許彥彬

In this part the OpenCV-2.4.2 I/O function was included.

Three functions and two classes as follow will be use in the algorithms in this homework.



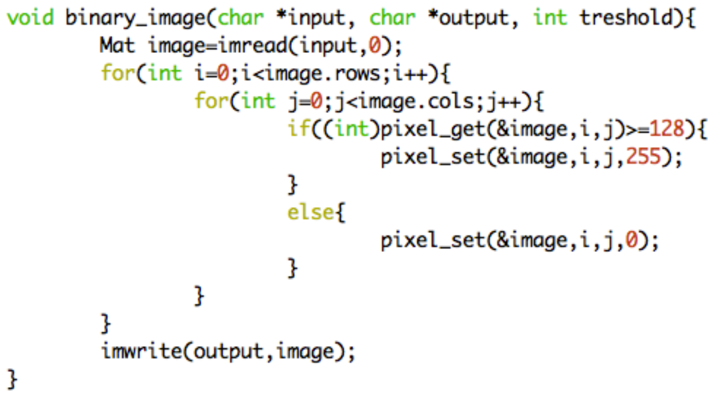
```
void pixel_set(Mat *p, int x, int y, int value){        //set B_PIX(p,x,y) to 'value'
        uchar* tp = p->data+x*p->cols+y;
        *tp = value;
}

uchar pixel_get(Mat *p, int x,int y){                   //return B_PIX(p,x,y)
        uchar* tp = p->data+x*p->cols+y;
        return *tp;
}

void pixel_swap(Mat *p, int x, int y, int xp, int yp){  //swap B_PIX(p,x,y) and B_PIX(p,xp,yp)
        uchar pix = pixel_get(p,xp,yp);
        pixel_set(p,xp,yp,pixel_get(p,x,y));
        pixel_set(p,x,y,pix);
}
```



```
class eq_class_head{
        private:
                void insert(comp *p){
                        if(start==NULL || start->x>p->x){
                                p->next=start;
                                start=p;
                                return;
                        }
                        else{
                                comp *ptr=start;
                                while(ptr!=NULL){
                                        if(ptr->next==NULL || ptr->next->x>p->x){
                                                p->next=ptr->next;
                                                ptr->next=p;
                                                return;
                                        }
                                        ptr=ptr->next;
                                }
                        }
                }
        public:
                comp* start;
                int count,x_l,x_r,y_u,y_d;
                eq_class_head* next;
                eq_class_head(){
                        start=NULL;
                        next=NULL;
                        count=x_r=y_d=0;
                        x_l=y_u=10000;
                }
                void eq_class_insert(int num){
                        comp *p = new comp(num);
                        insert(p);
                }
};
```

The two classes are used in making connected component. We will talk about them later in hw_2-3.

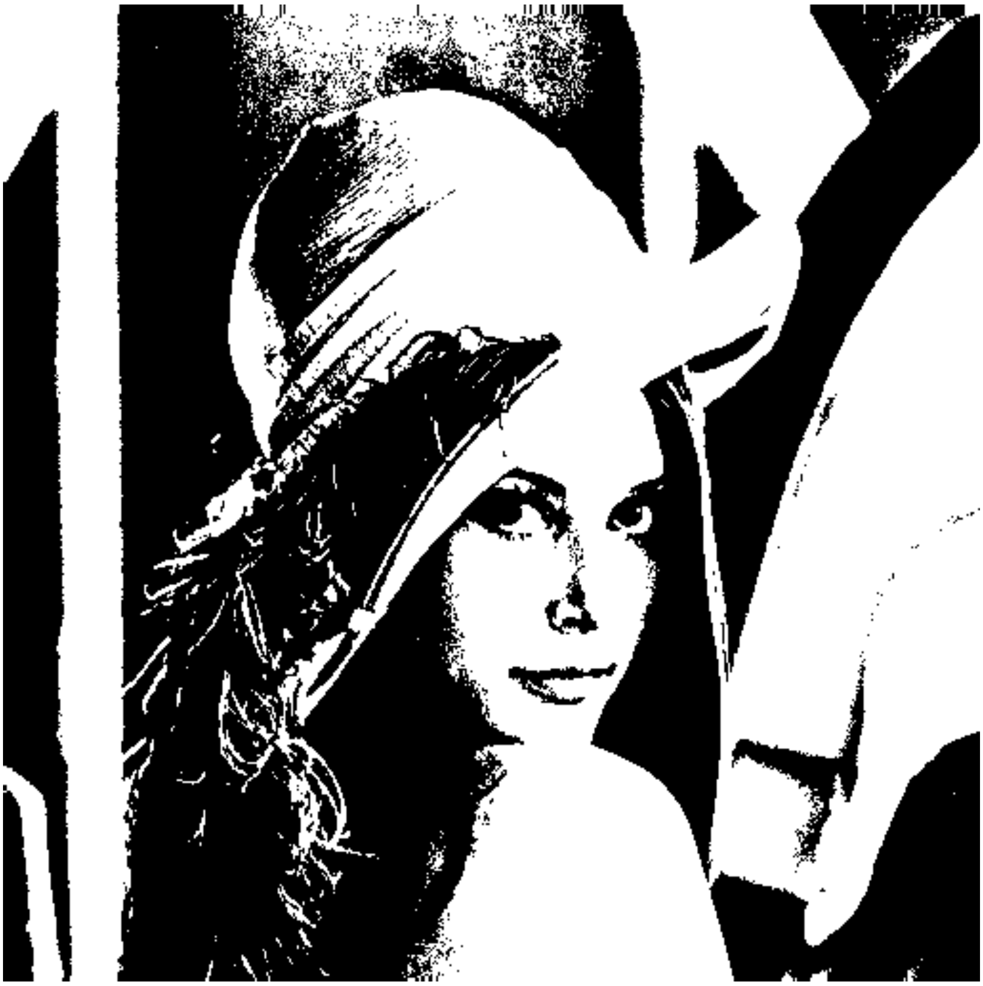
1. A binary image with threshold 128



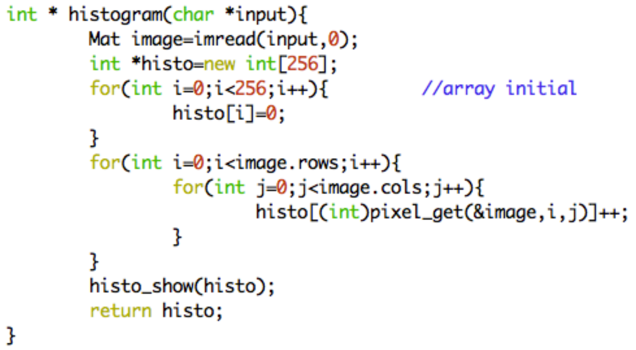
```
void binary_image(char *input, char *output, int treshold){
        Mat image=imread(input,0);
        for(int i=0;i<image.rows;i++){
                for(int j=0;j<image.cols;j++){
                        if((int)pixel_get(&image,i,j)>=128){
                                pixel_set(&image,i,j,255);
                        }
                        else{
                                pixel_set(&image,i,j,0);
                        }
                }
        }
        imwrite(output,image);
}
```

i. Search for the whole image lena.bmp. If the gray scale is great equal than 128, set the gray scale to 255. Otherwise set the gray scale to 0.

ii. The result is as follow.



2. A histogram



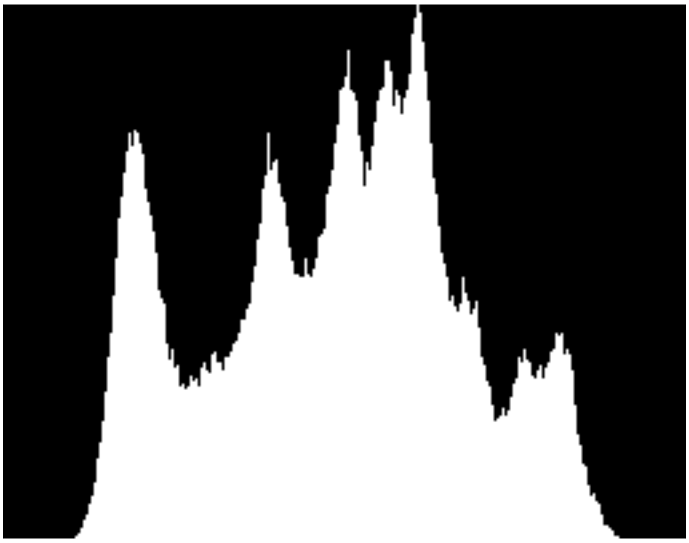
```
int * histogram(char *input){
        Mat image=imread(input,0);
        int *histo=new int[256];
        for(int i=0;i<256;i++){         //array initial
                histo[i]=0;
        }
        for(int i=0;i<image.rows;i++){
                for(int j=0;j<image.cols;j++){
                        histo[(int)pixel_get(&image,i,j)]++;
                }
        }
        histo_show(histo);
        return histo;
}
```

i. Create an integer array sized 256 with 0 as initial.

ii. Get the gray scale for every pixel in the image lena.bmp, and add 1 to corresponding bin.

For example, if the gray scale of pixel(0,0) is 15, them we will add 1 to the 15th bin of array "histo".

iii. Plot the histogram after this process done. The result is as follow.

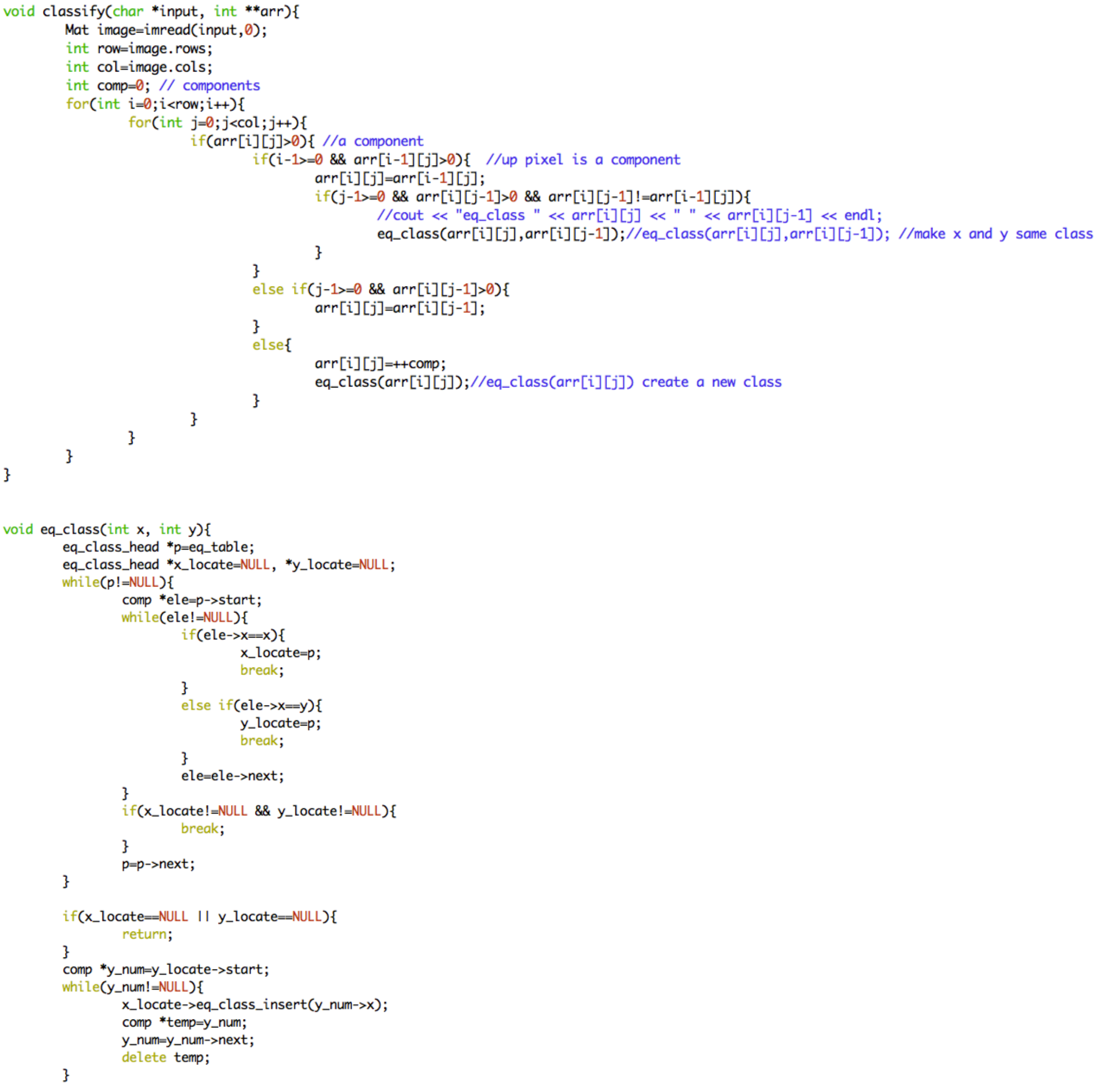


3. Connected components (use 4 – connected)

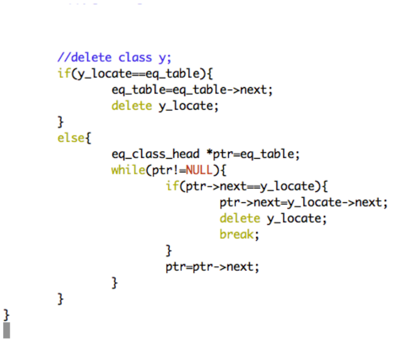
In this part, I used the classical algorithm which showed in 2.3.4 of the book but adjust a little stuff. The two classes mentioned before are used in here. I use "4 – connected" in this part.

i. For the fist step, simply search the binary image and write the information into a matrix (two dimension array). Value 1 for gray scale 0 (black) and 0 for scale 255 (white).

ii. Top-down search the matrix, when matrix(a, b) is 1, if matrix(a-1, b) is k with k larger than 0. Assign matrix(a,b) as k. If matrix(a,b-1) is k' also larger than 0. Then Label k and k’ same class. If matrix(a.b-1) is k’ and and matrix(a-1,b) is 0, assign matrix(a,b) as k’. If both matrix(a-1,b) and matrix(a,b-1) is 0, then assign matrix(a,b) as a new label.

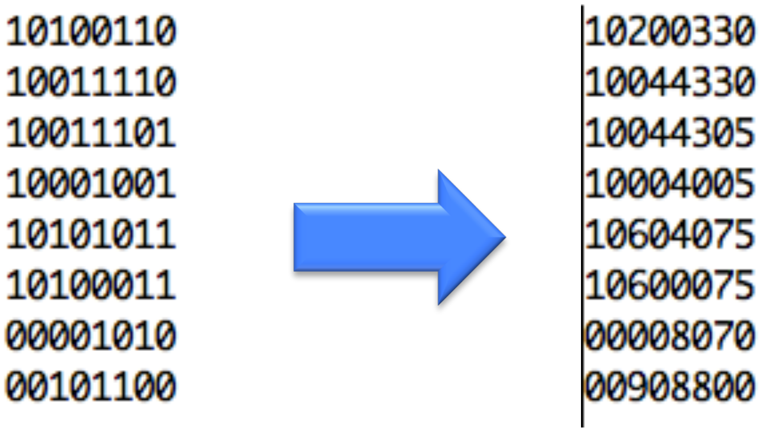


```
void classify(char *input, int **arr){
        Mat image=imread(input,0);
        int row=image.rows;
        int col=image.cols;
        int comp=0; // components
        for(int i=0;i<row;i++){
                for(int j=0;j<col;j++){
                        if(arr[i][j]>0){ //a component
                                if(i-1>=0 && arr[i-1][j]>0){  //up pixel is a component
                                        arr[i][j]=arr[i-1][j];
                                        if(j-1>=0 && arr[i][j-1]>0 && arr[i][j-1]!=arr[i-1][j]){
                                                //cout << "eq_class " << arr[i][j] << " " << arr[i][j-1] << endl;
                                                eq_class(arr[i][j],arr[i][j-1]);//eq_class(arr[i][j],arr[i][j-1]); //make x and y same class
                                        }
                                }
                                else if(j-1>=0 && arr[i][j-1]>0){
                                        arr[i][j]=arr[i][j-1];
                                }
                                else{
                                        arr[i][j]=++comp;
                                        eq_class(arr[i][j]);//eq_class(arr[i][j]) create a new class
                                }
                        }
                }
        }
}


void eq_class(int x, int y){
        eq_class_head *p=eq_table;
        eq_class_head *x_locate=NULL, *y_locate=NULL;
        while(p!=NULL){
                comp *ele=p->start;
                while(ele!=NULL){
                        if(ele->x==x){
                                x_locate=p;
                                break;
                        }
                        else if(ele->x==y){
                                y_locate=p;
                                break;
                        }
                        ele=ele->next;
                }
                if(x_locate!=NULL && y_locate!=NULL){
                        break;
                }
                p=p->next;
        }

        if(x_locate==NULL || y_locate==NULL){
                return;
        }
        comp *y_num=y_locate->start;
        while(y_num!=NULL){
                x_locate->eq_class_insert(y_num->x);
                comp *temp=y_num;
                y_num=y_num->next;
                delete temp;
        }
```

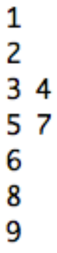


```
        //delete class y;
        if(y_locate==eq_table){
                eq_table=eq_table->next;
                delete y_locate;
        }
        else{
                eq_class_head *ptr=eq_table;
                while(ptr!=NULL){
                        if(ptr->next==y_locate){
                                ptr->next=y_locate->next;
                                delete y_locate;
                                break;
                        }
                        ptr=ptr->next;
                }
        }
}
```

An easy example using 8x8 lena.bmp is shown below.



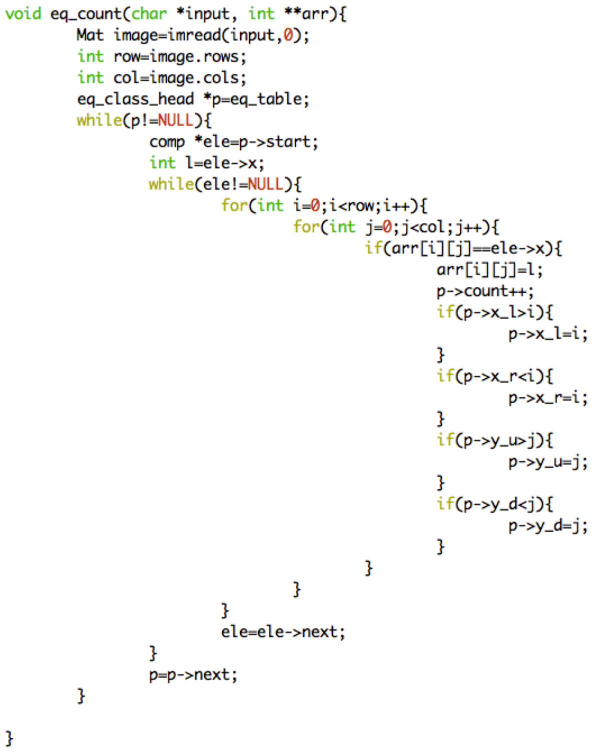
```
10100110        10200330
10011110        10044330
10011101        10044305
10001001        10004005
10101011        10604075
10100011        10600075
00001010        00008070
00101100        00908800
```



```
1
2
3 4
5 7
6
8
9
```

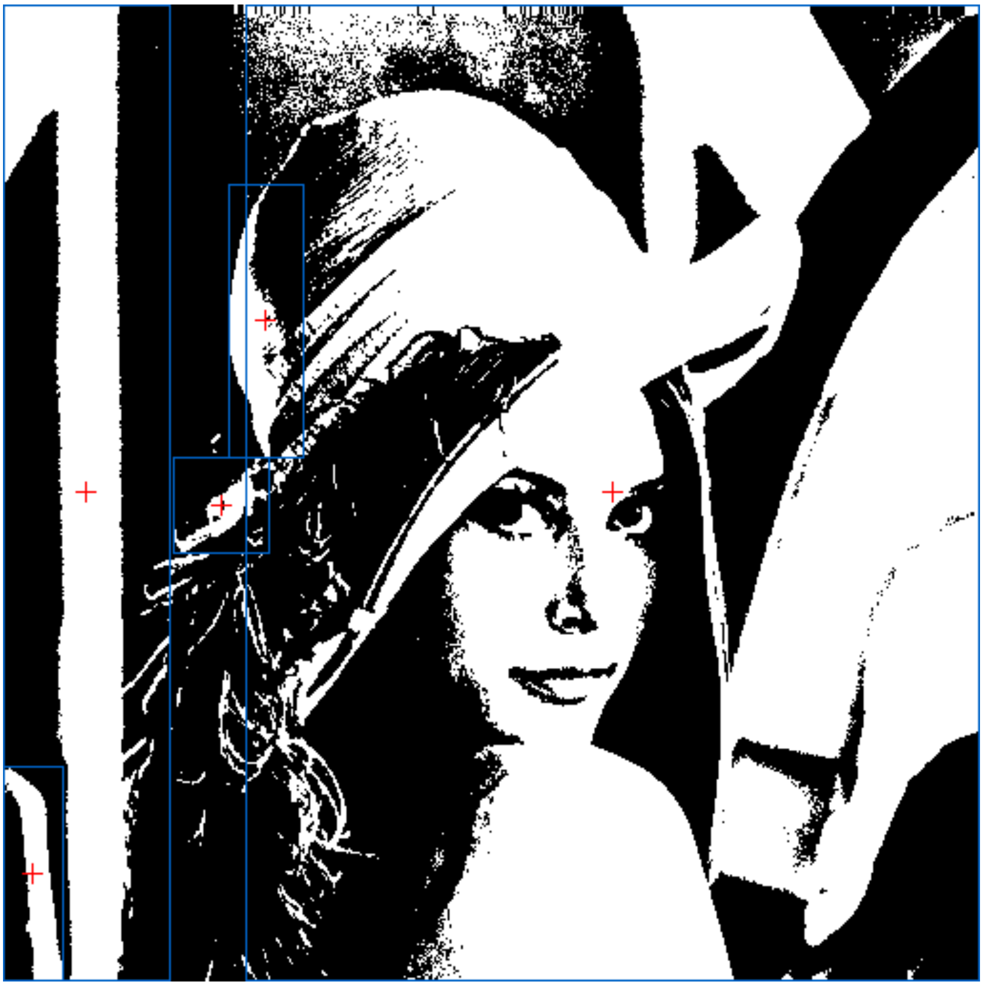
Equivalence classes.

iii. Use the equivalence class information and the matrix after top – down pass to recognize the connected components and centroids. Only connected component, which has more than 500 pixels, will be recognize.



```
void eq_count(char *input, int **arr){
    Mat image=imread(input,0);
    int row=image.rows;
    int col=image.cols;
    eq_class_head *p=eq_table;
    while(p!=NULL){
        comp *ele=p->start;
        int l=ele->x;
        while(ele!=NULL){
            for(int i=0;i<row;i++){
                for(int j=0;j<col;j++){
                    if(arr[i][j]==ele->x){
                        arr[i][j]=l;
                        p->count++;
                        if(p->x_l>i){
                            p->x_l=i;
                        }
                        if(p->x_r<i){
                            p->x_r=i;
                        }
                        if(p->y_u>j){
                            p->y_u=j;
                        }
                        if(p->y_d<j){
                            p->y_d=j;
                        }
                    }
                }
            }
            ele=ele->next;
        }
        p=p->next;
    }
}
```

iv. Result is as follow.



4. Appendix

   i. build_all.sh

   command: “sh build_all.sh” will automatically compile the code

   ii. hw_2.cpp

   source code

   iii. bi_lena.bmp, histogram.bmp, bi_lena_4connected.bmp

   results for this homework

   iv. R01922124_HW2.pdf