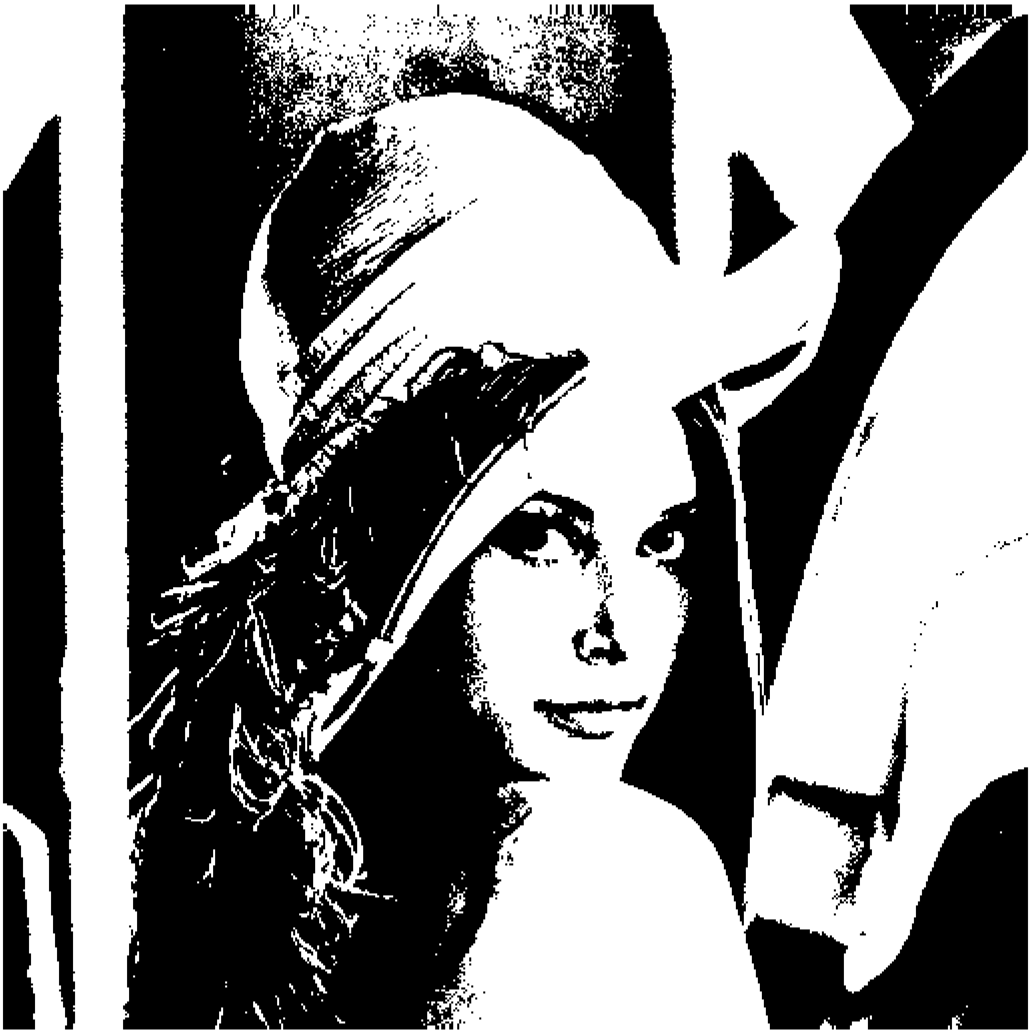
資工所碩一 林群崴 R07922164

1. Binarize Lena with the threshold 128



在main function中：

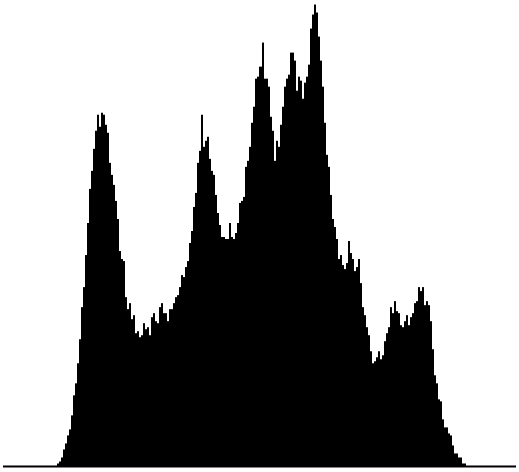
```
    b_img = binary(b_img,t, widthLimit, heightLimit);
```

其中b_img為一個openCV的mat

```
Mat binary(Mat img, uchar t, int widthLimit, int heightLimit) {
    for (int height = 0; height < heightLimit; height++) {
        uchar *data = img.ptr<uchar>(height);
        for (int width = 0; width < widthLimit; width++) {
            data[width] = data[width] < t ? 0 : 255;
        }
    }

    return img;
}
```

解釋：利用迴圈讀取每一個像素，若小於threshold的將其設為0，大於則設為255

2. Histogram



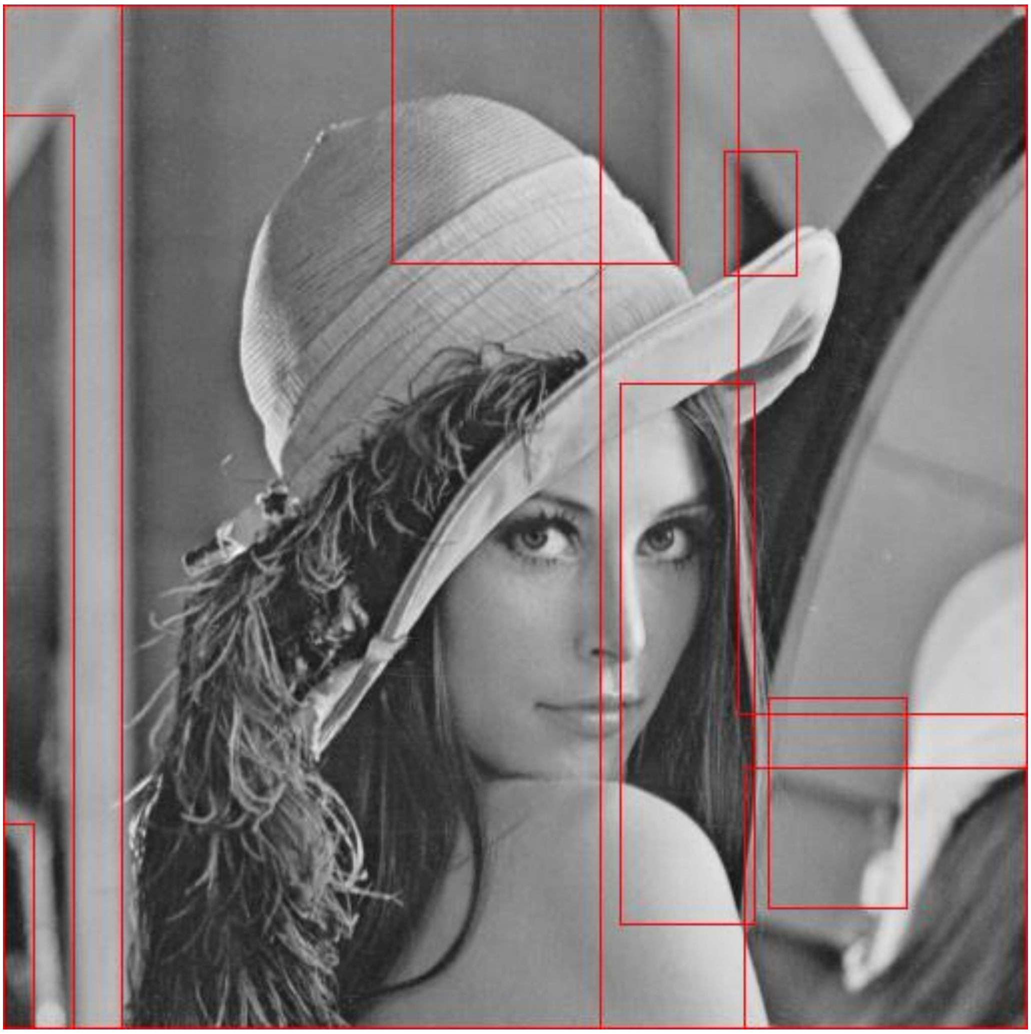
在main function中：

```
    vector <int>v(256,0);     //v = {0,0,...0}
    Mat his = histogram(img, v, widthLimit, heightLimit);
```

```
Mat histogram(Mat img, vector <int>&v, int widthLimit, int heightLimit) {

    int grayscale;

    for (int height = 0; height < heightLimit; height++) {
        uchar *data = img.ptr<uchar>(height);
        for (int width = 0; width < widthLimit; width++) {
            grayscale = data[width];
            v.at(grayscale)++;
        }
    }

    Mat showHistImg(256, 256, CV_8UC1, Scalar(255));  //把直方圖秀在一個256*256大的影像上
    drawHistImg(showHistImg,v);
    return showHistImg;
}
```

解釋：利用迴圈讀取每個pixel，並利用一個256維的向量v紀錄0~255的grayscale分別有多少，每當讀取一個pixel，將其對應的grayscale加上1。

3. Connected Components（採用 8-connected）



```
//bounding box
    Mat c_img = connected(b_img, widthLimit, heightLimit);
    vector<int> labelSet;
    int max = 1;//找出最大的標籤幾號
    for (int height = 0; height < heightLimit; height++) {
        for (int width = 0; width < widthLimit; width++) {
            max = max < c_img.at<int>(height, width) ? c_img.at<int>(height, width) : max;
            labelSet.push_back(c_img.at<int>(height, width));
        }
    }
    sort(labelSet.begin(), labelSet.end());
    labelSet.erase(unique(labelSet.begin(), labelSet.end()), labelSet.end());
    cout << "最大值是" << max << endl;

    //開始畫邊界
    img = imread("lena.bmp", CV_LOAD_IMAGE_COLOR);
    int com = 0;
    int rec[4] = {heightLimit, widthLimit, 0, 0}; //[0]:左 [1]:上 [2]:右 [3]:下

    while(!labelSet.empty()) {
        int label = labelSet.back();
        labelSet.pop_back();
        for (int height = 0; height < heightLimit; height++) {
            for (int width = 0; width < widthLimit; width++) {
                if (c_img.at<int>(height, width) == label) {
                    com++;
                    if (width < rec[0])rec[0] = width;
                    if (height < rec[1])rec[1] = height;
                    if (width > rec[2])rec[2] = width;
                    if (height > rec[3])rec[3] = height;
                }
            }
        }
        if (com > 500)rectangle(img, Point(rec[0], rec[1]), Point(rec[2], rec[3]), Scalar(0, 0, 255), 1);
        com = 0;
        rec[0] = heightLimit;
        rec[1] = widthLimit;
        rec[2] = 0;
        rec[3] = 0;
    }
```

```
Mat connected(Mat img, int widthLimit, int heightLimit) {
    Mat labImg(heightLimit, widthLimit, CV_32S, Scalar(0));

    //initialization of each forebackground pixel in a ascending order
    int i = 1;
    for (int height = 0; height < heightLimit; height++) {
        for (int width = 0; width < widthLimit; width++) {
            labImg.at<int>(height, width) = img.at<uchar>(height, width) == 255 ? 0 : i++;
        }
    }
    //repeat until flag "change" is false
    bool change;
    do {
        change = false;
        //Top-down
        for (int height = 0; height < heightLimit; height++) {
            for (int width = 0; width < widthLimit; width++) {

                int temp = labImg.at<int>(height, width);
                if (temp == 0)continue;

                //若是最靠近左邊的pixel則不跟左邊比
                if ((width > 0) && (temp > labImg.at<int>(height, width - 1)) &&
(labImg.at<int>(height, width - 1) > 0)) {
                    temp = labImg.at<int>(height, width - 1);
                }

                if ((width > 0) && (height > 0) && (temp > labImg.at<int>(height - 1, width
- 1)) && (labImg.at<int>(height - 1, width - 1) > 0)) {
                    temp = labImg.at<int>(height - 1, width - 1);
                }

                //跟上面比
                if ((height > 0) && (temp > labImg.at<int>(height - 1, width)) &&
(labImg.at<int>(height - 1, width) > 0))
                    temp = labImg.at<int>(height - 1, width);

                //跟右上比
                if ((height > 0) && (width < widthLimit - 1) && (temp >
labImg.at<int>(height - 1, width + 1)) && (labImg.at<int>(height - 1, width + 1) > 0)) {
                    temp = labImg.at<int>(height - 1, width + 1);
                }
                if (labImg.at<int>(height, width) != temp) {
                    labImg.at<int>(height, width) = temp;
                    change = true;
                }
            }
        }
        for (int height = heightLimit - 1; height >= 0; height--) {
            for (int width = widthLimit - 1; width >=0; width--) {
                int temp = labImg.at<int>(height, width);
                if (temp == 0)continue;

                //若是最靠近右邊的pixel則不跟右邊比
                if ((width < widthLimit - 1) && (temp > labImg.at<int>(height, width + 1))
&& (labImg.at<int>(height, width + 1) > 0)) {
                    temp = labImg.at<int>(height, width + 1);
                }

                //跟右下比
                if ((width < widthLimit - 1) && (height < heightLimit - 1) && (temp >
labImg.at<int>(height + 1, width + 1)) && (labImg.at<int>(height + 1, width + 1) > 0)) {
                    temp = labImg.at<int>(height + 1, width + 1);
                }

                //跟下比
                if ((height < heightLimit - 1)&&(temp > labImg.at<int>(height + 1,
width))&&(labImg.at<int>(height + 1, width) > 0)) {
                    temp = labImg.at<int>(height + 1, width);
                }

                if (width > 0) {    //跟左下比
                    if ((width > 0) && (height < heightLimit - 1) && (temp >
labImg.at<int>(height + 1, width - 1)) && (labImg.at<int>(height + 1, width - 1) > 0)) {
                        temp = labImg.at<int>(height + 1, width - 1);
                    }
                }

                if (labImg.at<int>(height, width) != temp) {
                    labImg.at<int>(height, width) = temp;
                    change = true;
                }
            }
        }
    } while (change);
    cout << "完成！" << endl;
    return labImg;
```

```
}
```

解釋：

在Component的function中採用8-connected，第一步驟先將每個非背景pixel（grayscale非0）由小到大（從1開始）將每個pixel的label，第二步驟利用迴圈從左上至右下傳遞，再從右下至左上傳遞；每當發生一次傳遞，將變數change設置為true，不斷地top-down和buttom-up，直到change為fales，代表不再有pixel的傳遞發生。

其中畫bounding box的時候，利用rec的整數array，其中rec[0]代表左上角的點x軸落在哪個位置，rec[1]代表左上角的點y軸落在哪個位置，rec[2]代表右下角的點x軸落在哪個位置，rec[3]代表右下角的點y軸落在哪個位置。

利用巢狀迴圈從整張圖的邊界開始縮小，利用類似冠軍問題的解法將此component最左、最上、最右、最下分別記錄進對應的變數，可由rec[0]和rec[1]決定一點p1、rec[2]和rec[3]決定一點p2，又此兩點在二維平面上決定一個矩形，此即bounding box，應題目要求，component的pixel總數小於500者不予顯示。