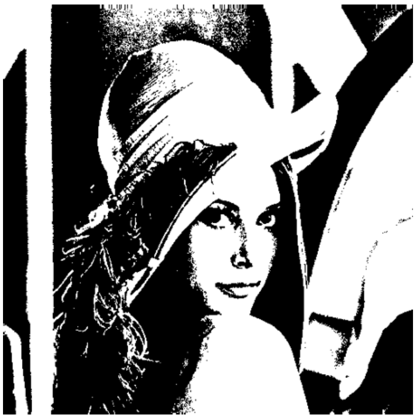
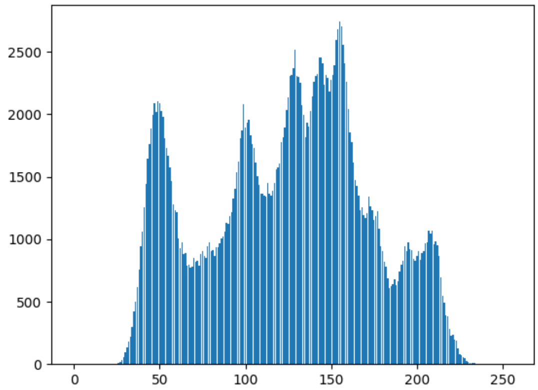
Computer Vision Homework 2　D07922015　謝銘峰

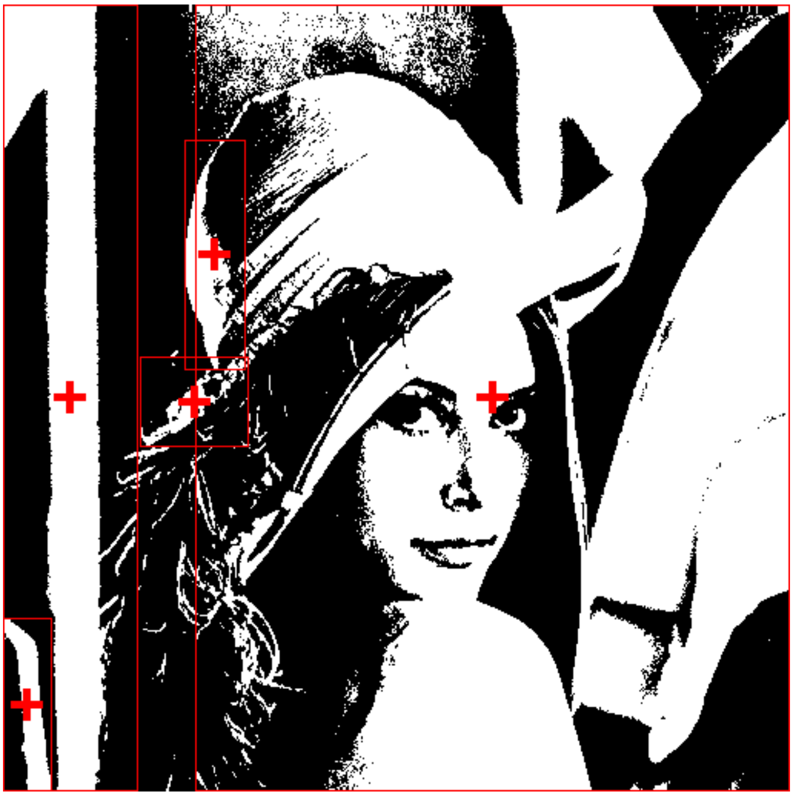
Write a program to generate (Using bmp):



A binary image (threshold at 128)



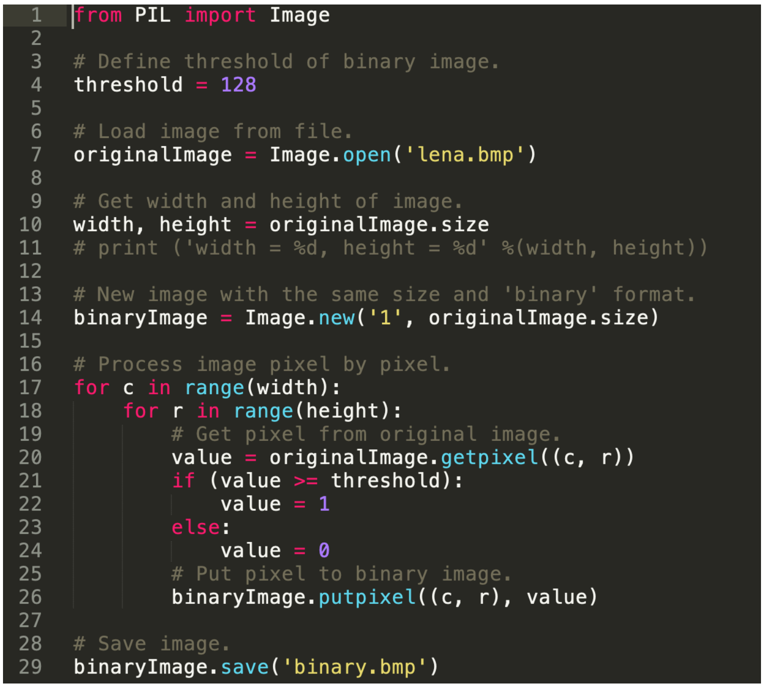
A histogram



Connected components (regions with + at centroid, bounding box)

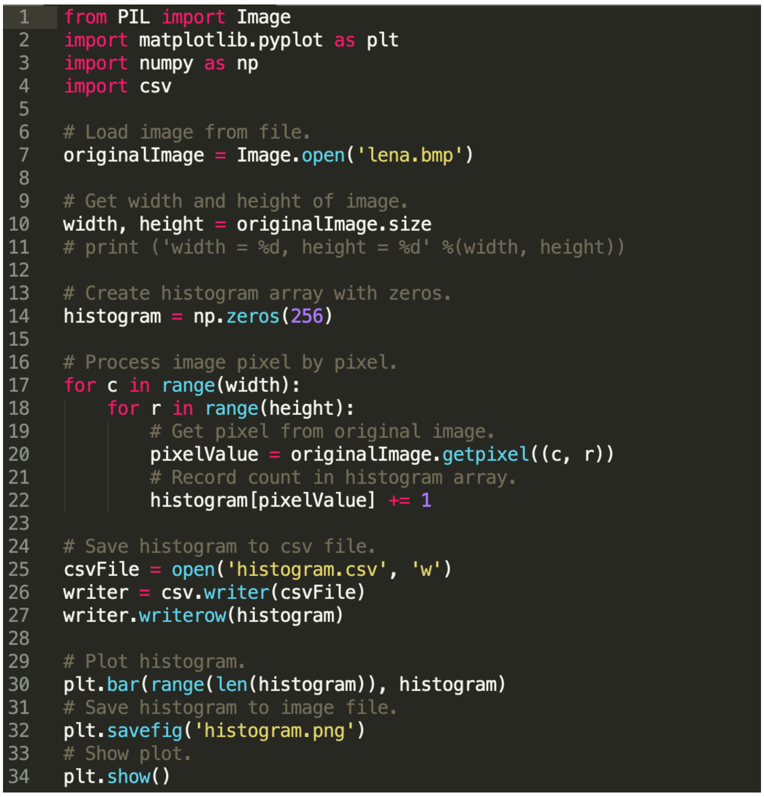
Write a program to generate Threshold and Histogram:

(a) a binary image (threshold at 128)



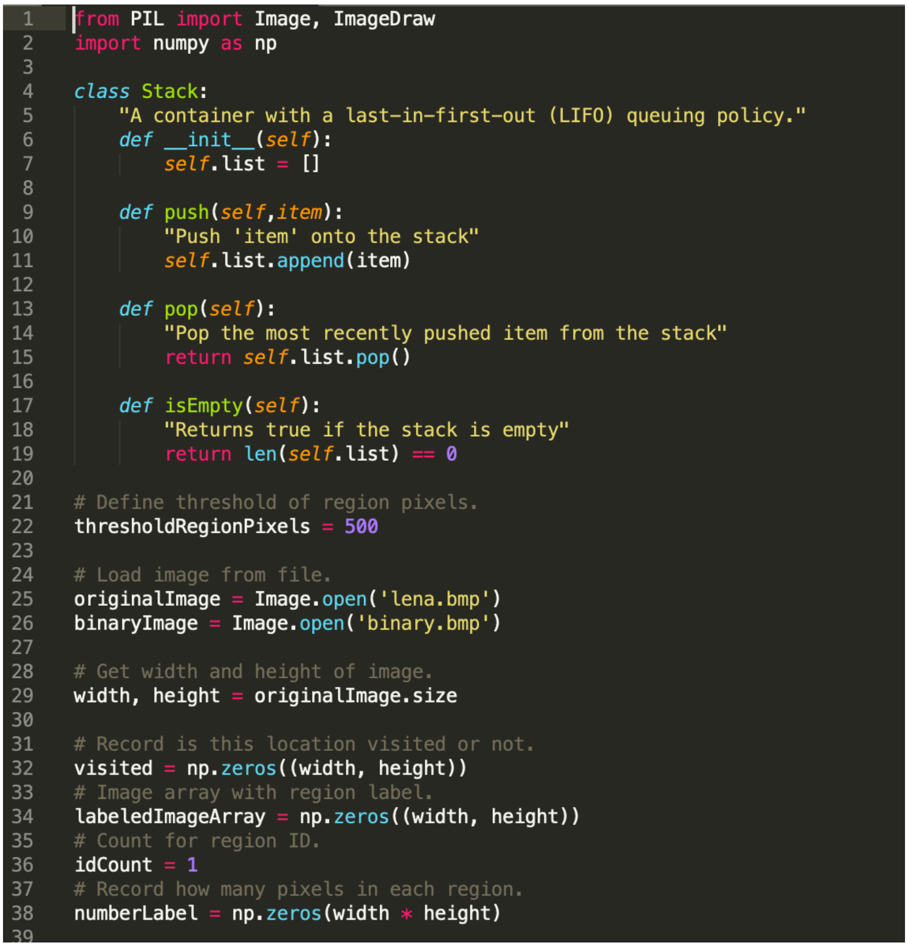
```python
from PIL import Image

# Define threshold of binary image.
threshold = 128

# Load image from file.
originalImage = Image.open('lena.bmp')

# Get width and height of image.
width, height = originalImage.size
# print ('width = %d, height = %d' %(width, height))

# New image with the same size and 'binary' format.
binaryImage = Image.new('1', originalImage.size)

# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # Get pixel from original image.
        value = originalImage.getpixel((c, r))
        if (value >= threshold):
            value = 1
        else:
            value = 0
        # Put pixel to binary image.
        binaryImage.putpixel((c, r), value)

# Save image.
binaryImage.save('binary.bmp')
```

(b) a histogram

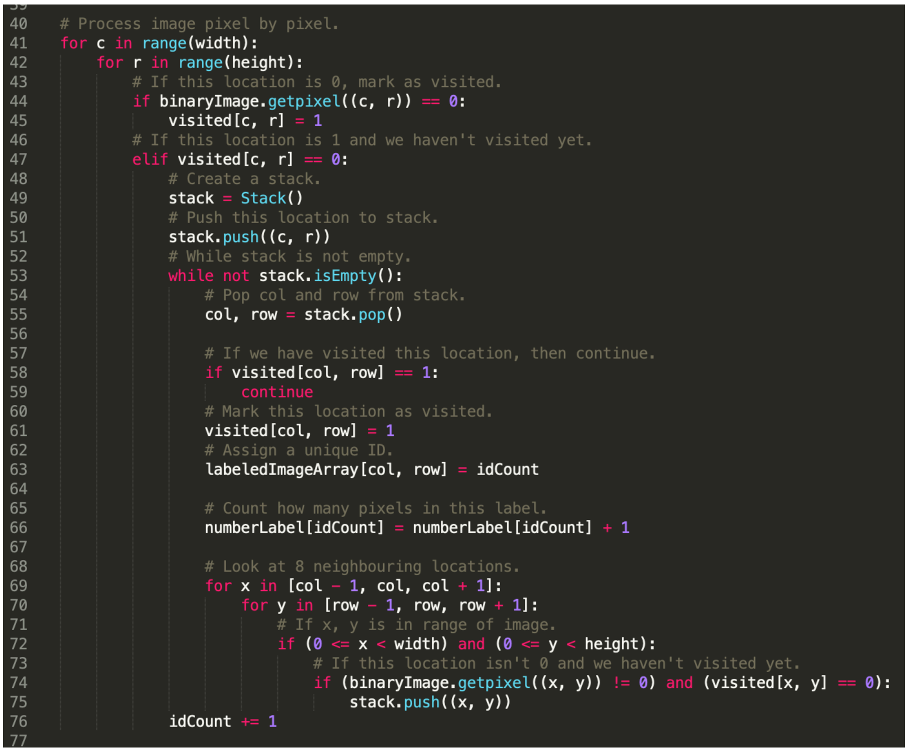


```python
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import csv

# Load image from file.
originalImage = Image.open('lena.bmp')

# Get width and height of image.
width, height = originalImage.size
# print ('width = %d, height = %d' %(width, height))

# Create histogram array with zeros.
histogram = np.zeros(256)

# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # Get pixel from original image.
        pixelValue = originalImage.getpixel((c, r))
        # Record count in histogram array.
        histogram[pixelValue] += 1

# Save histogram to csv file.
csvFile = open('histogram.csv', 'w')
writer = csv.writer(csvFile)
writer.writerow(histogram)

# Plot histogram.
plt.bar(range(len(histogram)), histogram)
# Save histogram to image file.
plt.savefig('histogram.png')
# Show plot.
plt.show()
```

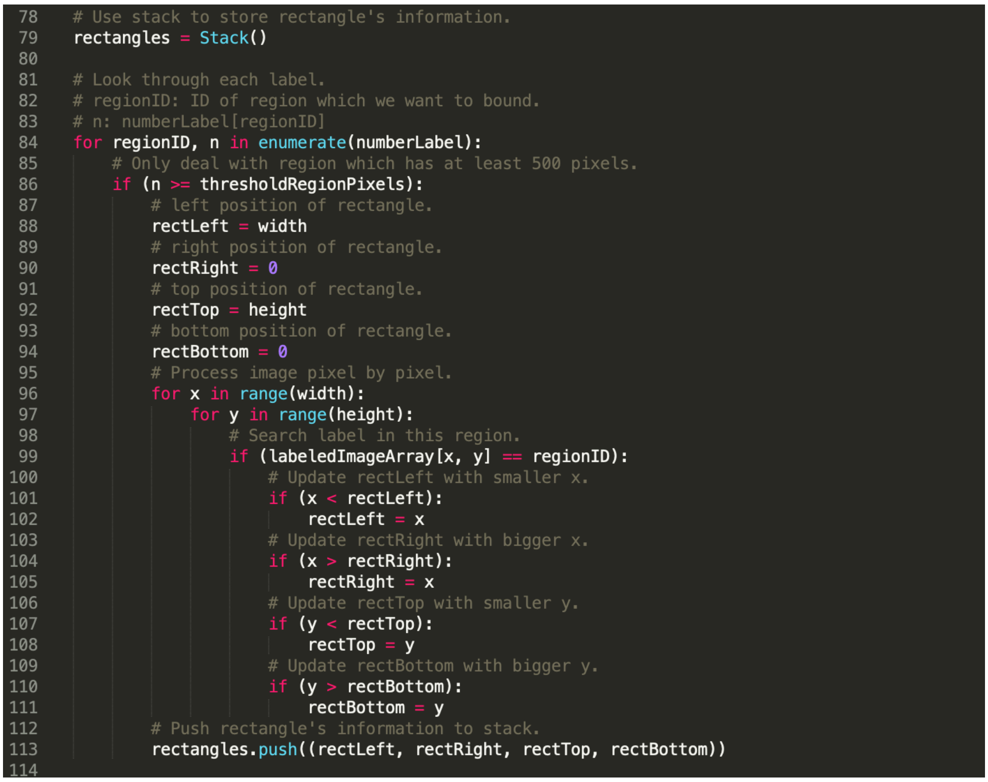
(c) connected components (regions with + at centroid, bounding box)



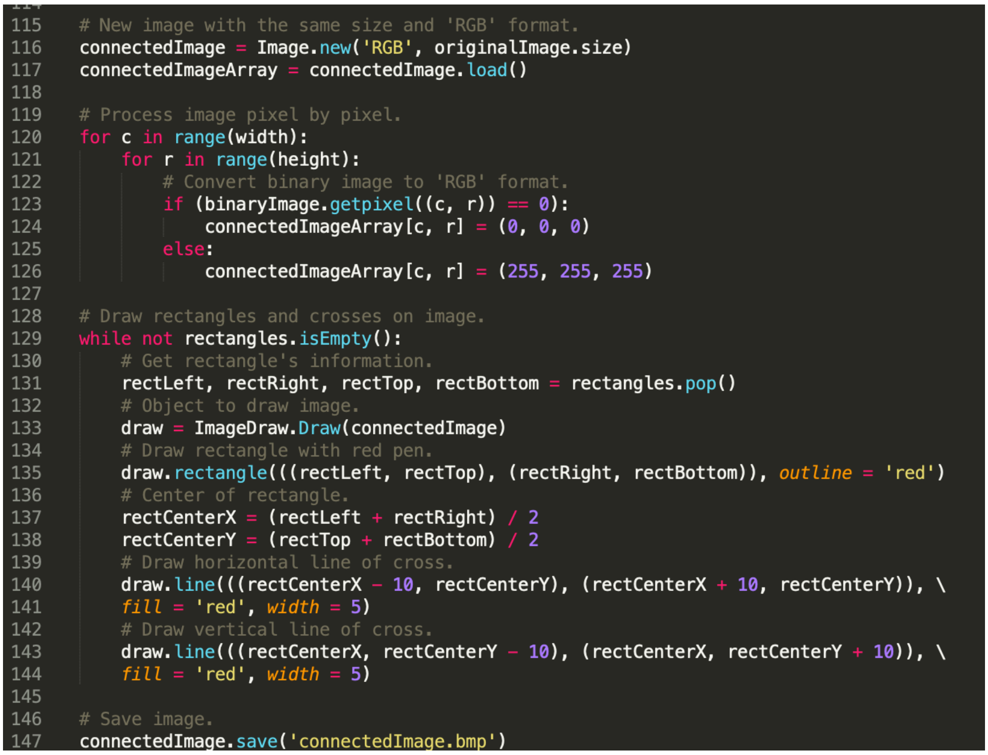
```python
from PIL import Image, ImageDraw
import numpy as np

class Stack:
    "A container with a last-in-first-out (LIFO) queuing policy."
    def __init__(self):
        self.list = []

    def push(self,item):
        "Push 'item' onto the stack"
        self.list.append(item)

    def pop(self):
        "Pop the most recently pushed item from the stack"
        return self.list.pop()

    def isEmpty(self):
        "Returns true if the stack is empty"
        return len(self.list) == 0

# Define threshold of region pixels.
thresholdRegionPixels = 500

# Load image from file.
originalImage = Image.open('lena.bmp')
binaryImage = Image.open('binary.bmp')

# Get width and height of image.
width, height = originalImage.size

# Record is this location visited or not.
visited = np.zeros((width, height))
# Image array with region label.
labeledImageArray = np.zeros((width, height))
# Count for region ID.
idCount = 1
# Record how many pixels in each region.
numberLabel = np.zeros(width * height)

```



```python
# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # If this location is 0, mark as visited.
        if binaryImage.getpixel((c, r)) == 0:
            visited[c, r] = 1
        # If this location is 1 and we haven't visited yet.
        elif visited[c, r] == 0:
            # Create a stack.
            stack = Stack()
            # Push this location to stack.
            stack.push((c, r))
            # While stack is not empty.
            while not stack.isEmpty():
                # Pop col and row from stack.
                col, row = stack.pop()

                # If we have visited this location, then continue.
                if visited[col, row] == 1:
                    continue
                # Mark this location as visited.
                visited[col, row] = 1
                # Assign a unique ID.
                labeledImageArray[col, row] = idCount

                # Count how many pixels in this label.
                numberLabel[idCount] = numberLabel[idCount] + 1

                # Look at 8 neighbouring locations.
                for x in [col - 1, col, col + 1]:
                    for y in [row - 1, row, row + 1]:
                        # If x, y is in range of image.
                        if (0 <= x < width) and (0 <= y < height):
                            # If this location isn't 0 and we haven't visited yet.
                            if (binaryImage.getpixel((x, y)) != 0) and (visited[x, y] == 0):
                                stack.push((x, y))
            idCount += 1

```



```
# Use stack to store rectangle's information.
rectangles = Stack()

# Look through each label.
# regionID: ID of region which we want to bound.
# n: numberLabel[regionID]
for regionID, n in enumerate(numberLabel):
    # Only deal with region which has at least 500 pixels.
    if (n >= thresholdRegionPixels):
        # left position of rectangle.
        rectLeft = width
        # right position of rectangle.
        rectRight = 0
        # top position of rectangle.
        rectTop = height
        # bottom position of rectangle.
        rectBottom = 0
        # Process image pixel by pixel.
        for x in range(width):
            for y in range(height):
                # Search label in this region.
                if (labeledImageArray[x, y] == regionID):
                    # Update rectLeft with smaller x.
                    if (x < rectLeft):
                        rectLeft = x
                    # Update rectRight with bigger x.
                    if (x > rectRight):
                        rectRight = x
                    # Update rectTop with smaller y.
                    if (y < rectTop):
                        rectTop = y
                    # Update rectBottom with bigger y.
                    if (y > rectBottom):
                        rectBottom = y
        # Push rectangle's information to stack.
        rectangles.push((rectLeft, rectRight, rectTop, rectBottom))

```



```
# New image with the same size and 'RGB' format.
connectedImage = Image.new('RGB', originalImage.size)
connectedImageArray = connectedImage.load()

# Process image pixel by pixel.
for c in range(width):
    for r in range(height):
        # Convert binary image to 'RGB' format.
        if (binaryImage.getpixel((c, r)) == 0):
            connectedImageArray[c, r] = (0, 0, 0)
        else:
            connectedImageArray[c, r] = (255, 255, 255)

# Draw rectangles and crosses on image.
while not rectangles.isEmpty():
    # Get rectangle's information.
    rectLeft, rectRight, rectTop, rectBottom = rectangles.pop()
    # Object to draw image.
    draw = ImageDraw.Draw(connectedImage)
    # Draw rectangle with red pen.
    draw.rectangle(((rectLeft, rectTop), (rectRight, rectBottom)), outline = 'red')
    # Center of rectangle.
    rectCenterX = (rectLeft + rectRight) / 2
    rectCenterY = (rectTop + rectBottom) / 2
    # Draw horizontal line of cross.
    draw.line(((rectCenterX - 10, rectCenterY), (rectCenterX + 10, rectCenterY)), \
    fill = 'red', width = 5)
    # Draw vertical line of cross.
    draw.line(((rectCenterX, rectCenterY - 10), (rectCenterX, rectCenterY + 10)), \
    fill = 'red', width = 5)

# Save image.
connectedImage.save('connectedImage.bmp')
```