HW2 Report

環境: python3, pillow, matplotlib, numpy

檔案:

------output------

------binarized.bmp → output of (1)

------histogram.jpg → output of (2)

------connected_marked.jpg → output of (3)

------lena.bmp

------report.pdf → report

------hw2-1.py → code of (1)

------hw2-2.py → code of (2)

------hw2-3.py → code of (3)

執行: 1. python3 hw2-1.py 2. python3 hw2-2.py 3. python3 hw2-3.py

(make sure there's a folder "output" and the image file "lena.bmp")

1. Binarize lena.bmp

   Modify all pixel values,

   If > 127 then put 255,

   else then 0.

```python
from PIL import Image
im = Image.open('lena.bmp')
width, height = im.size

#binarize with threshold 128 (0~127, 128~255)
for i in range(width):
    for j in range(height):
        im.putpixel((i,j), 0 if im.getpixel((i,j)) < 128 else 255)
im.save("output/binarized.bmp")
im.close()
```

2. Draw histogram of lena.bmp

   Make a array count with 256 zeros,

   then traverse the whole picture.

   Add 1 to count[i] if found a pixel value I,

   then we get the count array for histogram.

```python
from PIL import Image
import matplotlib.pyplot as plt

im = Image.open('lena.bmp')
width, height = im.size

#caculate the histogram
xaxis, count = [i for i in range(256)], [0]*256
for i in range(width):
    for j in range(height):
        count[im.getpixel((i,j))] += 1
#draw
plt.bar(xaxis, count, color='blue')
plt.savefig('output/histogram.jpg')
im.close()
```

3. Mark connected components: use 4-connected

   segments → consecutive 255s on a row,

   represented by a tuple(head_index, tail_index, row)

   group → list of connected segments

getsegments() inputs a row X col data array with only two types of value: 0 and 255, and outputs a list with row rows, each contain a list of segments at the row.

```python
def overlap(seg1, seg2, connect='4'):
    if connect == '4':  #4-connected
        return False if seg1[1] < seg2[0] or seg2[1] < seg1[0] else True
    else:               #8-connected
        return False if seg1[1] < (seg2[0]-1) or seg2[1] < (seg1[0]-1) else True

def getsegments(row, col, data):
    segments = [list() for i in range(row)]
    for i in range(row):
        j = 0
        while j<col:
            while j<col and data[i][j]==0: j+=1
            if j==col: break
            head = j
            while j<col and data[i][j]==255: j+=1
            segments[i] += [(head, j-1, i)]
    return segments
```

In the main program, first read the binarized lena image into a numpy array,
Then call groupup to get a list of groups, which all are lists of connected segments.

```
im = Image.open('output/binarized.bmp')
(width, height), data_array = im.size, numpy.array(im)
groups = groupup(getsegments(height, width, data_array), height, 500)
```

In groupup, we maintain a dictionary find_group for us to find the corresponding group of the segment. That is, find_group[seg] means the list of connected segments containing seg.

First we traverse from the top to the bottom, and at the first row we just simply create a new group for each segment.

Next, from the second to the last row, we check if there's any segment at it's previous row which overlaps with it, if yes then merge the two group, else then create a new group for it.

Last, we traverse back from the bottom to the top, if there's any vertically adjacent segments that overlap, merge the two. After all these steps, we get groups, a list of groups, and return it.

```
def mergeGroup(g1, g2, groups, findG):
    smallG, bigG = g1, g2
    if len(bigG) < len(smallG): bigG, smallG = smallG, bigG
    bigG += smallG
    for seg in smallG: findG[seg] = bigG
    groups.remove(smallG)
```

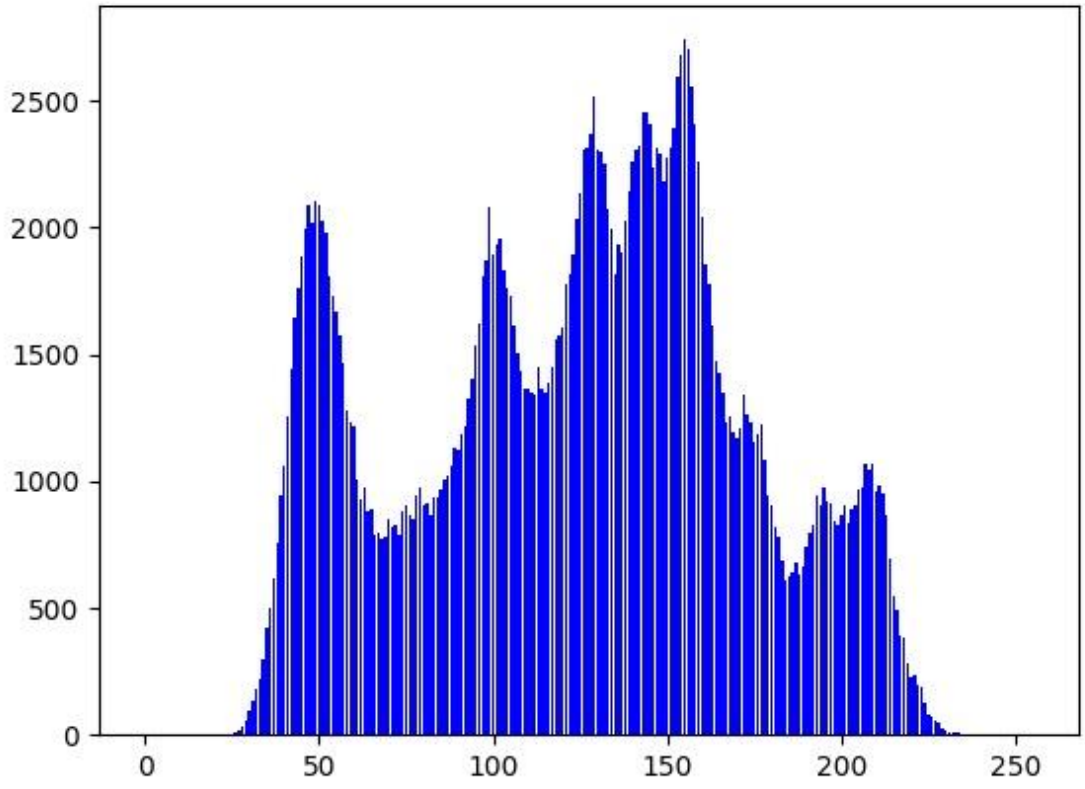
Note that when merging two groups, we always merge the smaller group to the bigger one, so that we could update the dictionary find_group quicker.

```
def groupup(segments, row, threshold):
    find_group, groups = dict(), []
    #to right-down
    for seg in segments[0]:
        #print(groups)
        newG = [seg]
        groups, find_group[seg] = groups + [newG], newG
    for i in range(1, row):
        for now in segments[i]:
            found_group = False
            for last in segments[i-1]:
                if overlap(now, last):
                    found_group = True
                    if now not in find_group:
                        lgroup = find_group[last]
                        find_group[now] = lgroup
                        lgroup.append(now)
                    else:
                        ngroup, lgroup = find_group[now], find_group[last]
                        if ngroup is not lgroup: mergeGroup(ngroup, lgroup, groups, find_group)
            if not found_group:
                newG = [now]
                find_group[now], groups = newG, groups + [newG]
    #to left-up
    for i in range(row-2, -1, -1):
        for now in reversed(segments[i]):
            for last in reversed(segments[i+1]):
                if overlap(now, last):
                    ngroup, lgroup = find_group[now], find_group[last]
                    if ngroup is not lgroup: mergeGroup(ngroup, lgroup, groups, find_group)
    #abandon groups by threshold
    retG = list()
    for group in groups:
        num = 0
        for seg in group: num += (seg[1] - seg[0] + 1)
        if num >= threshold: retG.append(group)
    return retG
```

結果:

1. binarized.bmp

2. histogram.jpg



3. connected_marked.jpg