

Language : python 3.7

Library :

```
import numpy as np
from PIL import Image
```

Execution way: python3 hw6.py

(please put lena.bmp at the same directory as hw6.py)

Problem: Calculate Yokoi Connectivity number

Description :

首先，先將原圖做downsampling(以最左上角的點代表整個8*8的block)後，對圖像進行前幾次作業已經做過很多次的binarization。這邊的順序有稍微調換一下因為我覺得先做downsampling可以節省一些計算量，binarization只要做downsample過的。

接著，就可以進行yokoi connectivity number 的計算了，我用兩個for迴圈，去看每個pixel，根據ppt上的 corner neighborhood 找出他的corner neighbor，若遇到邊界沒有值得時候設為0。

Corner Neighborhood (for corresponding x_i)

	x_2	x_6
		x_1

x_7	x_2	
x_3		

x_3		
x_8	x_4	

		x_1
	x_4	x_5

然後再利用ppt給的公式

$$h(b, c, d, e) = \begin{cases} q & \text{if } b = c \text{ and } (d \neq b \vee e \neq b) \\ r & \text{if } b = c \text{ and } (d = b \wedge e = b) \\ s & \text{if } b \neq c \end{cases}$$

待每個pixel的b,c,d,e計算出來後，就可以利用h equation，計算出q,r,s，透過結果便可以計算出該pixel的yoke connectivity number

Code

```
from PIL import Image
import numpy as np

class Solution:
    def __init__(self):
        pass

    def binarization(self, image): =

    def down_sampling(self, image, origin_size, target_size):
        pixels = image.load()
        new_image = Image.new(image.mode, (target_size, target_size))
        new_pixels = new_image.load()
        for i in range(target_size):
            for j in range(target_size):
                new_pixels[i, j] = pixels[i * int(origin_size/target_size), j * int(origin_size/target_size)]
        return new_image

    def h(self, b, c, d, e):
        if b != c: return 's'
        if b == d and b == e: return 'r'
        return 'q'

    def valid(self, new_x, new_y, h, w):
        return new_x >= 0 and new_x < h and new_y >= 0 and new_y < w

    def check(self, image, cor, x, y):
        h, w = image.size
        pixels = image.load()
        new_x, new_y = x + cor[0], y + cor[1]
        if self.valid(new_x, new_y, h, w): return pixels[new_x, new_y]
        return 0

    def Yokoi_num(self, image, corners):
        pixels = image.load()
        res = []
        h, w = image.size
        for y in range(w):
            row = []
            for x in range(h):
                if pixels[x, y] == 0:
                    row.append(' ')
                    continue
                else:
                    temp = []
                    for cor in corners:
                        b = pixels[x, y]
                        c, d, e = 0, 0, 0
                        c = self.check(image, cor[0], x, y)
                        d = self.check(image, cor[1], x, y)
                        e = self.check(image, cor[2], x, y)
                        temp.append(self.h(b,c,d,e))
                    if temp.count('r') == 4:
                        row.append('5')
                    else:
                        if temp.count('q') == 0:
                            row.append(' ')
                        else:
                            row.append(str(temp.count('q')))
            res.append(row)
        return np.array(res)
```

```
def save(self, save_name, res):
    file = open(save_name, "w")
    with open(save_name, "w") as file:
        for i in range(res.shape[0]):
            for j in range(res.shape[1]):
                file.write(res[i, j])
            file.write('\n')

def problem1(self, path, out_file, corners):
    image = Image.open(path)
    down_sampling_image = self.down_sampling(image, 512, 64)
    # down_sampling_image.save('down_sampling.bmp')
    bin_image = self.binarization(down_sampling_image)
    # bin_image.save('bin_image.bmp')
    yokoi = self.Yokoi_num(bin_image, corners)
    self.save(out_file, yokoi)
```

Result:

11111111	12111111111112232221	111111111111
15555551	115555555511 2 11 11	115555555551
15555551	1 2115555112 21112221	15555555551 21
15555551	1 2 155112 2221511	155555555511 1
15555551	22 2112 22 121	1555555555511
15555551	1 2 21 2 1 1	155555555551
15555551	12 1 121111 1321	15555555555511
15111551	1322 1155551111	15555555555551
111 1551	1 1215555551	15555555555511
11 1551	2115555551	1551155555511
21 1551	2 155555511	1551 1155511
1 1551	2 15555555551	1551 115551 1
1551	11211555555551	1551 15511 12
1551	15555555555551	1551 1111 111
1551	1 2221155555555551	1151 11 1151
1551	2 22 1 15555555555551	151 1111 1551
1551	2 1 1155555555555551	151 115551 11551
1551	2 11555555555555511	1511155511 115551
1551	12 11555555555555555551	155551
1551	11 221555555555555555555112	1155551
1551	111 22 1555555555555555555551 1	1555551
1551	1511 1 1251121111121115555555511	1155551
1551	15521 1 121 1 11 1 155555511	1555551
1551	1151 132 2 115555511	11555551
1551	151 322 115555111 121	155555551
1551	1221 2 1555551 131	115555551
1551	2 1 11555551 1	115555551
1551	2 1155555551	1 155555551
1551	2 11555555551	2115555551
1551	1 11555555551	1555555551
1551	1 11511115555521	1 11555555551
1551	1 1 1111 115551 2	155555555551
1551	131 111 15111 2	155555555551
1551	121 1121 1 111 1 2	115555555551
1551	11 111 1 221 11 1 2	155555555551
1551	12 1 21 121 11 1111 2	1555555555551
1551	1 12 22 151111111551 2	1155555555551
1551	1 2 155555115551 1	1555555555551
1551	2 22 12555551 15551 1	1555555555551
1551	1 1 1555511 15511 2	11555555555551
1551	21 155551 1 151 2	155555555555551
1551	2 15555112 151 2	155555555555551
1551	1 1 1155555551111 2	155555555555551
1551	2 22 11511111212 211555555555555551	1555555555555551
1551	1 12 151 2 1 1555555111555551	1555555551 1555551
1551	1111 121 155555551 1555551	155555551 1555551
1551	11111111 155555551 1555551	155555551 1555551
1551	115551 15555551 1555511	211111111 155511
1551	15551 211111111 155511	2 11 155511
11521	1 12 122155511 2	11 15511
1 151	1 1 15555511 2111	15511
22 1511	1 1555555511 155111	1511
22 1511	1 1555555551 15551	1151
2 151	1 1155555555551 155511	1511
2 1521	1 15555555555551 15551 12151	
2 151	121 15555555555551 155511 1551	
2 1511	155555555555551 115551 1511	
21 1511	11 155555555555551 111111151	
11 151	11555555555555511 11511	
11 151	15555555555555551 151	
11 151	11555555555555551 211	
11 151	115555555555555511 1	
11 151	155555555555555551	
11 111	121111111111111111	