HW7

- **A discription of your homework**

  Programming language used: Python 2.7

  Library used: Numpy, PIL

- **Your parameters**

  i: row

  j: column

  tem: 用於儲存每個像素的灰階數值 0~255

  I,b: 計算 marked-interior/border-pixel operator 時的 border/interior label

  p,q: 計算 pair relationship operator 時的 label

  a: 計算 marked-pixel connected shrink operator 時的a_1, a_2, a_3, a_4

- **Functions**

  bi: binarize Lena

  unit: downsample Lena from 512x512 to 64x64

  bor: marked-interior/border-pixel operator

  pair: pair relationship operator

  thin: marked-pixel connected shrink operator

  chk: check if output result changes under repetition of bor, pair ,thin

  out: output result to .txt file

- **The algorithm you used**

1. Binarize Lena (512x512)

   Threshold: 128

   gray scale >= 128 → white(1)

   gray scale < 128 → black(0)

2. Using 8x8 blocks as a unit, take the topmost-left pixel as the downsampled data

   → Downsampling Lena from 512x512 to 64x64

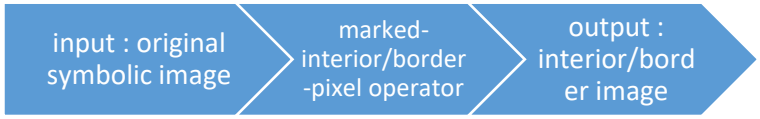
   取原圖 i,j 為 8 的倍數的像素

   & add one pixel of background to the edge of the 64x64 image

   → for convenience of calculating the yokoi number

   i,j 為 0 或 65 時，灰階值給 0

   lena
   64x64

3. 

$$h(c,d) = \begin{cases} c & if\ c = d \\ b & if\ c \neq d \end{cases}$$

$$f(c) = \begin{cases} b & if\ c = b \\ i & if\ c \neq b \end{cases}, where\ i: interior\ label, b: border\ label$$

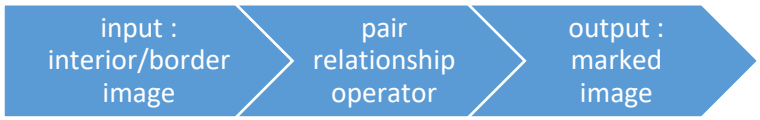
$$a_0 = x_0$$

$a_n = h(a_{n-1}, x_n)$

- $for\ 4\text{-}connected$

$a_n = h(a_{n-1}, x_n)\ ,\ n = 1, \dots, 4$

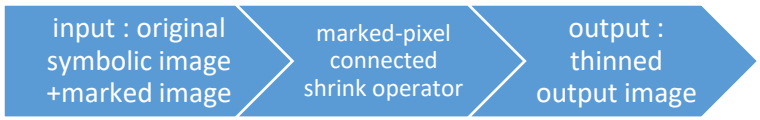
$output = f(a_4)$

4. 

- for 4-connectivity

$$h(a,i) = \begin{cases} 1 & if\ a = i \\ 0 & otherwise \end{cases}$$

$$output = \begin{cases} q & if \sum_{n=1}^{4} h(x_n, i) < 1 \vee x_0 \neq b \\ p & if \sum_{n=1}^{4} h(x_n, i) \geq 1 \wedge x_0 = b \end{cases}$$

5. 

marked-pixel connected shrink operator:

removable(by connected shrink operator on original symbolic image)

marked(by marked image)

delete those pixels satisfied the two conditions mentioned above

- for 4-connectivity

$$h(b,c,d,e) = \begin{cases} 1 & if\ b = c \wedge (b \neq d \ \vee \ b \neq e) \\ 0 & otherwise \end{cases}$$

- $a_1 = h(x_0, x_1, x_6, x_2)$
- $a_1 = h(x_0, x_2, x_7, x_3)$
- $a_1 = h(x_0, x_3, x_8, x_4)$
- $a_1 = h(x_0, x_4, x_5, x_1)$

$$output = f(a_1, a_2, a_3, a_4, x_0) = \begin{cases} g & if\ exactly\ one\ of\ a_1, a_2, a_3, a_4 = 1 \\ x_0 & otherwise \end{cases}$$

6. use thinned output image as next original symbolic image

   repeat step1, step2, step3 until the last output never changed

7. output result

- **Principal code fragment**

```
def bi(x):
    tem = np.zeros(x.shape)
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i][j]<128:
                tem[i][j] = 0
            else:
                tem[i][j] = 1
    return tem

def unit(x):
    tem = np.zeros([64+2,64+2])
    for i in range(64+2):# (64+2)*(64+2) array
        for j in range(64+2):
            if i==0 or j==0 or i==65 or j==65:
                tem[i][j]=0
            else:
                tem[i][j]=x[(i-1)*8][(j-1)*8]
    return tem

I = 2
b = 3
def bor(x):
    tem = np.zeros(x.shape)
    for i in range(1,x.shape[0]-1):
        for j in range(1,x.shape[1]-1):
            if x[i][j]==1:
                tem[i][j]=x[i][j]
                if tem[i][j]!=x[i-1][j]:
                    tem[i][j] = b #
                if tem[i][j]!=x[i][j-1]:
                    tem[i][j] = b #
                if tem[i][j]!=x[i+1][j]:
                    tem[i][j] = b #
                if tem[i][j]!=x[i][j+1]:
                    tem[i][j] = b #
                if tem[i][j]!=b:
                    tem[i][j] = I #
    return tem
```

```
p = 4
q = 5
def pair(x):
    tem = np.zeros(x.shape)
    for i in range(1,x.shape[0]-1):
        for j in range(1,x.shape[1]-1):
            if x[i][j]==b:
                if x[i-1][j]==I or x[i][j-1]==I or x[i+1][j]==I or x[i][j+1]==I:
                    tem[i][j] = p #
                else:
                    tem[i][j] = q #
            elif x[i][j]==I: #
                tem[i][j] = q #
    return tem

def thin(x):
    y = pair(bor(x)) #marked by pair
    for i in range(1,x.shape[0]-1):
        for j in range(1,x.shape[1]-1):
            if x[i][j]==1:
                a=[]
                if x[i][j]==x[i][j+1]:
                    if x[i][j]!=x[i-1][j+1] or x[i][j]!=x[i-1][j]:
                        a.append(1)
                if x[i][j]==x[i-1][j]:
                    if x[i][j]!=x[i-1][j-1] or x[i][j]!=x[i][j-1]:
                        a.append(1)
                if x[i][j]==x[i][j-1]:
                    if x[i][j]!=x[i+1][j-1] or x[i][j]!=x[i+1][j]:
                        a.append(1)
                if x[i][j]==x[i+1][j]:
                    if x[i][j]!=x[i+1][j+1] or x[i][j]!=x[i][j+1]:
                        a.append(1)
                if a.count(1)==1 and y[i][j]==p:
                    x[i][j]=0
    return x

def chk(x):
    count=0
    while(True):
        count +=1
        x1=np.copy(x)
        x2=thin(x)
        if np.array_equal(x1, x2) == True: break

    print(count)
    return x2


def out(x):
    f = open('thin.txt','w')
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i][j]==1:
                f.write('*')
            else:
                f.write(' ')
        f.write('\n')
    f.close
```

- **Resulting images**