Homework 7

```
def mark_pair_relationship(img_ib):

    def h(a, m):
        if a == m:
            return 1
        return 0

    img_pair = np.zeros(img_ib.shape)
    for i in range(img_ib.shape[0]):
        for j in range(img_ib.shape[1]):
            if img_ib[i][j] > 0:
                # not background pixel
                x1, x2, x3, x4 = 0, 0, 0, 0
                if i == 0:
                    if j == 0:
                        x1, x4 = img_ib[i][j + 1], img_ib[i + 1][j]
                    elif j == img_ib.shape[1] - 1:
                        x3, x4 = img_ib[i][j - 1], img_ib[i + 1][j]
                    else:
                        x1, x3, x4 = img_ib[i][j + 1], img_ib[i][j - 1],
img_ib[i + 1][j]
                elif i == img_ib.shape[0] - 1:
                    if j == 0:
                        x1, x2 = img_ib[i][j + 1], img_ib[i - 1][j]
                    elif j == img_ib.shape[1] - 1:
                        x2, x3 = img_ib[i - 1][j], img_ib[i][j - 1]
                    else:
                        x1, x2, x3 = img_ib[i][j + 1], img_ib[i - 1][j],
img_ib[i][j - 1]
                else:
                    if j == 0:
                        x1, x2, x4 = img_ib[i][j + 1], img_ib[i - 1][j],
img_ib[i + 1][j]
                    elif j == img_ib.shape[1] - 1:
                        x2, x3, x4 = img_ib[i - 1][j], img_ib[i][j - 1],
img_ib[i + 1][j]
                    else:
                        x1, x2, x3, x4 = img_ib[i][j + 1], img_ib[i - 1][j],
img_ib[i][j - 1], img_ib[i + 1][j]
                if h(x1, 1) + h(x2, 1) + h(x3, 1) + h(x4, 1) >= 1 and img_ib[i]
[j] == 1:
                    img_pair[i][j] = 1 # p
                else:
                    img_pair[i][j] = 2 # q
```

```
    return img_pair
```

- 先根據簡報中Pair Relationship Operator的定義以及計算方式，寫好函數；為了方便起見，將p設為 1 、q設為2。

```
img = cv2.imread('lena.bmp')
img = binarize(img, 128)

p = np.zeros((64, 64))
for i in range(p.shape[0]):
    for j in range(p.shape[1]):
        p[i][j] = img[8 * i][8 * j][0]
```

- open lena.bmp -> binarize lena.bmp -> downsample lena.bmp with the topmost-left pixel as the downsampled data

```
p_thin = copy.deepcopy(p)

while True:

    p_thin_old = copy.deepcopy(p_thin)

    # Step 1:  Yokoi Operator
    p_yokoi = compute_yokoi_number(p_thin_old)

    # Step 2:  Pair Relationship Operator
    p_mark = mark_pair_relationship(p_yokoi)

    # Step 3:  Connected Shrink Operator

    def h(b, c, d, e):
        if b == c and (d != b or e != b):
            return 1
        else:
            return 0

    for i in range(p_mark.shape[0]):
        for j in range(p_mark.shape[1]):
            if p_mark[i][j] == 1: # p
                if i == 0:
                    if j == 0:
                        x7, x2, x6 = 0, 0, 0
                        x3, x0, x1 = 0, p_thin[i][j], p_thin[i][j + 1]
                        x8, x4, x5 = 0, p_thin[i + 1][j], p_thin[i + 1][j + 1]
                    elif j == p_thin.shape[1] - 1:
                        x7, x2, x6 = 0, 0, 0
                        x3, x0, x1 = p_thin[i][j - 1], p_thin[i][j], 0
                        x8, x4, x5 = p_thin[i + 1][j - 1], p_thin[i + 1][j], 0
```

```
                    else:
                        x7, x2, x6 = 0, 0, 0
                        x3, x0, x1 = p_thin[i][j - 1], p_thin[i][j], p_thin[i]
[j + 1]
                        x8, x4, x5 = p_thin[i + 1][j - 1], p_thin[i + 1][j],
p_thin[i + 1][j + 1]
                elif i == p_mark.shape[0] - 1:
                    if j == 0:
                        x7, x2, x6 = 0, p_thin[i - 1][j], p_thin[i - 1][j + 1]
                        x3, x0, x1 = 0, p_thin[i][j], p_thin[i][j + 1]
                        x8, x4, x5 = 0, 0, 0
                    elif j == p_mark.shape[1] - 1:
                        x7, x2, x6 = p_thin[i - 1][j - 1], p_thin[i - 1][j], 0
                        x3, x0, x1 = p_thin[i][j - 1], p_thin[i][j], 0
                        x8, x4, x5 = 0, 0, 0
                    else:
                        x7, x2, x6 = p_thin[i - 1][j - 1], p_thin[i - 1][j],
p_thin[i - 1][j + 1]
                        x3, x0, x1 = p_thin[i][j - 1], p_thin[i][j], p_thin[i]
[j + 1]
                        x8, x4, x5 = 0, 0, 0
                else:
                    if j == 0:
                        x7, x2, x6 = 0, p_thin[i - 1][j], p_thin[i - 1][j + 1]
                        x3, x0, x1 = 0, p_thin[i][j], p_thin[i][j + 1]
                        x8, x4, x5 = 0, p_thin[i + 1][j], p_thin[i + 1][j + 1]
                    elif j == p_mark.shape[1] - 1:
                        x7, x2, x6 = p_thin[i - 1][j - 1], p_thin[i - 1][j], 0
                        x3, x0, x1 = p_thin[i][j - 1], p_thin[i][j], 0
                        x8, x4, x5 = p_thin[i + 1][j - 1], p_thin[i + 1][j], 0
                    else:
                        x7, x2, x6 = p_thin[i - 1][j - 1], p_thin[i - 1][j],
p_thin[i - 1][j + 1]
                        x3, x0, x1 = p_thin[i][j - 1], p_thin[i][j], p_thin[i]
[j + 1]
                        x8, x4, x5 = p_thin[i + 1][j - 1], p_thin[i + 1][j],
p_thin[i + 1][j + 1]

                a1 = h(x0, x1, x6, x2)
                a2 = h(x0, x2, x7, x3)
                a3 = h(x0, x3, x8, x4)
                a4 = h(x0, x4, x5, x1)

                if (a1 + a2 + a3 + a4) == 1:
                    p_thin[i][j] = 0

    # ckeck if the last output stops changing
    if np.sum(p_thin == p_thin_old) == p_thin.shape[0] * p_thin.shape[1]:
        break
```

```
cv2.imwrite('lena.thinned.bmp', p_thin)
```

- 先複製一張原圖作為thinning後的結果
- 進入iteration：先計算yokoi number -> 將計算完的yokoi number結果進行Pair Relationship計算 -> 將Pair Relationship計算後的結果進行Connected Shrink -> 對比舊的thinning和新的thinning是否不同
    - Connected Shrink Operator：

      同時使用預先做好的thinning結果圖跟經過Pair Relationship計算的marked image

      從左上到右下依序操作：如果該pixel上的value＝1(即為p) -> 計算a1、a2、a3、a4，如果其中恰好一個等於1 -> 將thinning結果圖上相對應pixel的value更新為0
- 每次iteration的input都是前一次iteration計算後的結果（更新後的thinning）
- Result：