HW4_Extraction of Connected Components

一、 Extraction of Connected Components(連通分量)

連通分量是指圖片中的一組相鄰的黑色像素，它們形成一個區塊，也就是說若兩個點之間存在一條路徑，則這兩個點屬於同一個連通分量。

提取連通分量是在圖片處理中一個非常常見的操作，可以用來對圖片中的物體進行計數或辨識，通常在提取連通分量時，我們會使用演算法來找出圖片中所有連通分量的像素位置，例如使用深度優先搜尋(Depth-First Search)或廣度優先搜尋(Breadth-First Search)來實踐連通分量的提取。

二、 深度優先搜尋演算法(DFS)

DFS 的核心精神就是先遇到的頂點就先 Visiting，並且以遇到的頂點做為新的搜尋起點，直到所有相連的頂點都被探索過。

在提取連通分量的問題中，DFS 可以透過搜尋所有相鄰的像素，來尋找與起始位置有相連的所有像素，透過不斷遞迴 DFS 函數，可以把圖片中所有連通分量都提取出來。

具體來說，首先將起始位置的像素標記為已拜訪，並把它加入到連通分量的陣列中，然後搜尋與起始位置相鄰的所有像素，如果該像素值為 0(黑色)且沒有被拜訪過，就對該像素再使用 DFS 函數，進一步搜尋

與它相連的像素，這樣就能夠找出整張圖片中所有與起始位置相連的像素。

1. 將起始位置的像素標記為已拜訪，並將它加入道連通分量的陣列中
2. 搜尋起始位置的相鄰像素，如果該像素值為 0 且沒有被拜訪過，則調用 DFS 函數搜尋與該像素相連的像素
3. 對於每個相鄰像素，重複步驟 1 和 2，直到找出整個連通分量。

三、程式實作

首先，我們進入 draw_img 函數，為我們畫一張大小為(100,300)的全白(像素值為 255)的灰階圖片，並使用 puText 在這張全白的灰階圖片上寫字，寫完字後回傳。

```
img = draw_img()
```

圖 一　程式起始點

```
def draw_img():
    img = 255 * np.ones((100,300) , dtype=np.uint8)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img , 'NCHU SIP Lab' , (0,25) , font , 1 , 0 ,2)
    cv2.putText(img , 'Happy New Year' , (0,50) , font , 1 , 0 ,2)
    cv2.putText(img , '2023' , (0,90) , font , 1 , 0 ,2)
    return img
```

圖 二　draw_img 函數內容

圖 三　輸出的圖片

第二步我們將圖三的圖片傳入 ConnectedComponents 函數，函數會先讀取圖片的 row 和 col，並且製作一張與圖片大小相同、全為 False 的 bool 圖

片 visited，用於紀錄是否拜訪過，再建立一個空陣列 components 用於儲存所有的連通分量，之後用雙層迴圈讀取圖片中每一個像素值，若像素值尚未被拜訪過且該像素值為黑色(0)，就建立一個空陣列 component，用於儲存此像素的連通分量，之後將圖片、此像素的位置以及 visited、component 傳入 DFS 函數，開始針對此像素值進行 DFS 搜尋，搜尋後的結果再 append 進 components 陣列。

```
components = ConnectedComponents(img.copy())
```

圖 四　　呼叫 ConnectedComponents 函數

```
def ConnectedComponents(img):
    row , col = img.shape[0] , img.shape[1]
    visited = [[False for i in range(col)] for j in range(row)] #建立一個跟圖片一樣大小的booling，用於紀錄是否拜訪過
    components = [] #儲存所有components的list
    for i in range(row):
        for j in range(col):
            if not visited[i][j] and img[i][j]==0:
                component = []
                DFS(img , i , j , visited , component) #對pixel做DFS搜尋
                components.append(component)
    return components
```

圖 五　　ConnectedComponents 函數內容

進入 DFS 函數後，要再確認一次像素值是否為 0 以及是否沒有拜訪過，因為剛剛我們所確認的是原始點，原始點往外擴散的其他點也同樣會繼續當作原始點，所以要再確認一次。確認過後，我們將該點設為 True，表示我們拜訪過這次的 pixel，之後不再拜訪，之後將該點位置 append 進 component 陣列內，代表此原始點的連通分量，之後使用遞迴的方式搜尋該點的上下左右四個像素點，全部的遞迴完成後即可得到此次的連通分量。

```python
def DFS(img , row , col , visited , component):
    if img[row][col] == 0 and not visited[row][col]:
        visited[row][col] = True #標記為拜訪過的pixel
        component.append((row,col))
        #對上下左右的像素進行DFS
        DFS(img, row - 1, col, visited, component)
        DFS(img, row + 1, col, visited, component)
        DFS(img, row, col - 1, visited, component)
        DFS(img, row, col + 1, visited, component)
```

圖 六　DFS 函數內容

待每一個像素點都順過一輪後，就能提取我們的連通分量，components 內有幾個 component 代表此圖有幾個區塊。得到結果後，我們再將結果傳入 print_table 函數，將我們的結果印出來。

```python
print_table(components)
```

圖 七　呼叫 print_table

```python
def print_table(components):
    print('-'*30)
    print(f'{"Connected":<8}          {"No. of pixels in":<15}')
    print(f'{"conponent":<8}          {"connected component":<15}')
    print('-'*30)
    for i , component in enumerate(components):
        if i<9:
            print(f'0{i+1:<7}              {len(component):<15}')
        else:
            print(f'{i+1:<8}             {len(component):<15}')
```

圖 八　print_table 內容

```
----------------------------------------
Connected          No. of pixels in
conponent          connected component
----------------------------------------
01                   198
02                   150
03                   169
04                   157
05                   174
06                   68
07                   163
08                   103
09                   159
10                   138
11                   169
12                   198
13                   121
14                   138
15                   159
16                   159
17                   109
18                   147
19                   155
20                   147
21                   138
22                   72
23                   165
24                   170
25                   165
26                   158
```

圖 九　　Extraction of Connected Components 結果

圖片	Extraction of Connected Components結果
	---------- Connected conponent / No. of pixels in connected component 01 198 02 150 03 169 04 157 05 174 06 68 07 163 08 103 09 159 10 138 11 169 12 198 13 121 14 138 15 159 16 159 17 109 18 147 19 155 20 147 21 138 22 72 23 165 24 170 25 165 26 158

圖 十　　對照圖

從圖十來看，可以發現確實原始圖片有 26 個區塊，且第 6 塊區塊(I)及第

22 塊區塊(r)的 pixel 值較少，這是合理的，因此程式邏輯無誤。

完整程式碼如下：

```
import cv2
import numpy as np
def ConnectedComponents(img):
    row , col = img.shape[0] , img.shape[1]
    visited = [[False for i in range(col)] for j in range(row)] #建立一個跟圖片一樣大小的booling，用於紀錄是否拜訪過
    components = [] #儲存所有components的list
    for i in range(row):
        for j in range(col):
            if not visited[i][j] and img[i][j]==0:
                component = []
                DFS(img , i , j , visited , component) #對pixel做DFS搜尋
                components.append(component)
    return components

def DFS(img , row , col , visited , component):
    if img[row][col] == 0 and not visited[row][col]:
        visited[row][col] = True #標記為拜訪過的pixel
        component.append((row,col))
        #對上下左右的像素進行DFS
        DFS(img, row - 1, col, visited, component)
        DFS(img, row + 1, col, visited, component)
        DFS(img, row, col - 1, visited, component)
        DFS(img, row, col + 1, visited, component)

def draw_img():
    img = 255 * np.ones((100,300) , dtype=np.uint8)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img , 'NCHU SIP Lab' , (0,25) , font , 1 , 0 ,2)
    cv2.putText(img , 'Happy New Year' , (0,50) , font , 1 , 0 ,2)
    cv2.putText(img , '2023' , (0,90) , font , 1 , 0 ,2)
    return img

def print_table(components):
    print('-'*40)
    print(f'{"Connected":<8}          {"No. of pixels in":<15}')
    print(f'{"conponent":<8}          {"connected component":<15}')
    print('-'*40)
    for i , component in enumerate(components):
        if i<9:
            print(f'0{i+1:<7}             {len(component):<15}')
        else:
            print(f'{i+1:<8}             {len(component):<15}')

img = draw_img()
components = ConnectedComponents(img.copy())
print_table(components)
cv2.imwrite('nchu.png',img)
cv2.imshow('img',img)
cv2.waitKey(0)
```

圖 十一　完整程式碼

四、心得

從這次作業當中可以學到何謂 Connected_Components，以及了解到其中的應用，也因此學到了 DFS 的應用，可以說是一舉兩得，此次作業雖然比前三次作業的程式碼都要來的短，但觀念也不少，需要蒐集許多資料才能寫出來。