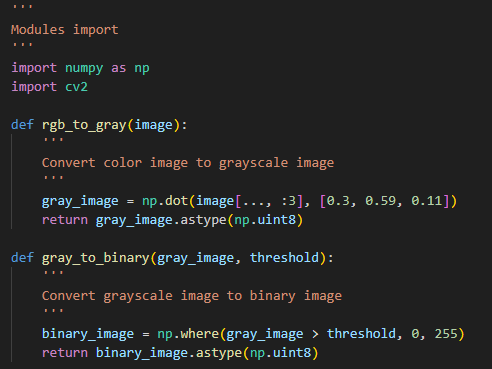
班級：資工三

學號：110590034

姓名：楊榮鈞



Function rgb\_to\_gray:

使用numpy的dot去實現。

rgb轉灰階是利用hw1的公式(0.3×𝑅)+(0.59×𝐺)+(0.11×𝐵)去轉成灰階。

最後用astype(np.uint8)是因為現在的的範圍是0~255。

回傳gray\_image。

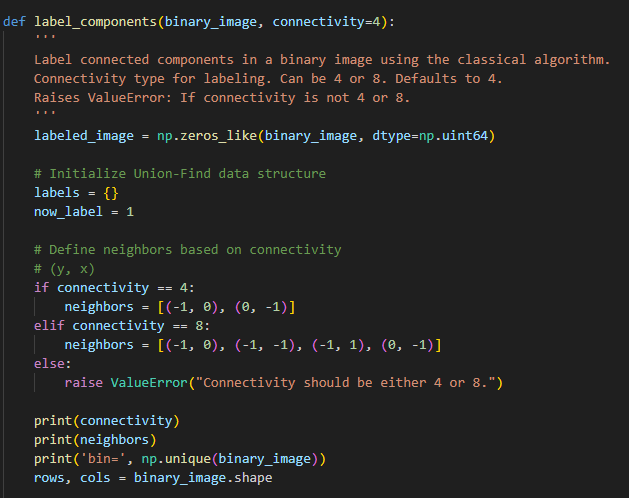
Function gray\_to\_binary:

使用numpy的where去實現。

灰階轉binary是當大於threshold時，會設成0，要是小於等於threshold，則是設成255，也就是背景會設為黑色。

最後用astype(np.uint8)是因為現在的的範圍是0~255。

回傳binary\_image。



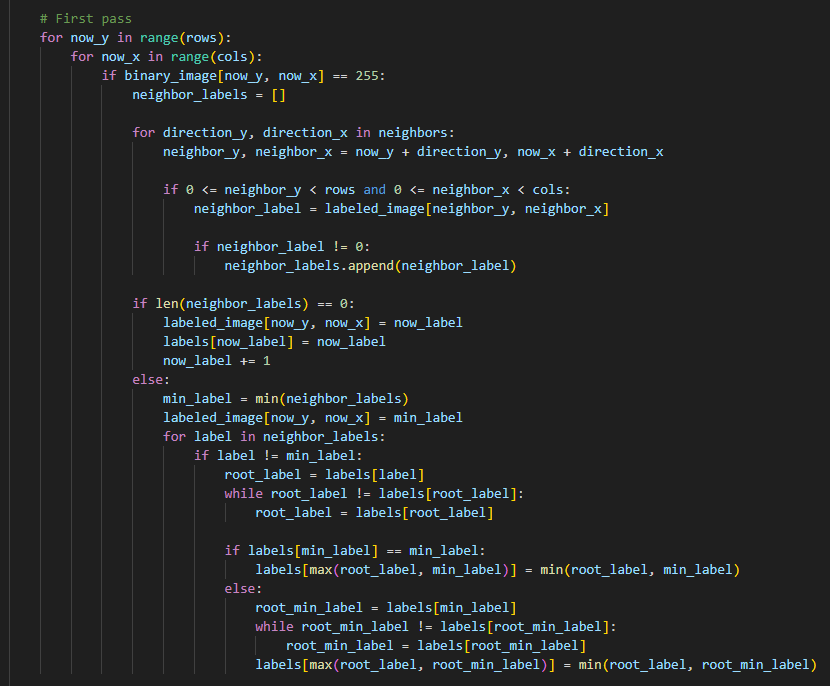
Function label\_components (分成三個截圖，這是第一個截圖):

首先利用numpy的zeros\_like建立labeled\_image的array，其中dtype=np.uint64是因為label的數量可能會很多，但是不確定會有多少，所以先設定np.uint64。

接著初始化labels的dictionary和now\_label，以及根據connectivity的值判斷我們需要判斷的neighbors有哪些，當connectivity不是4或是8時，會raise ValueError。

其中connectivity為4的話，要判斷的方向是上方(-1, 0)和左方(0, -1)；connectivity為8的話，要判斷的方向是上方(-1, 0)、左上方(-1, -1)、右上方(-1, 1)和左方(0, -1)。

根據binary\_image的shape得到labeled的rows(高)和cols(寬)。



Function label\_components (分成三個截圖，這是第二個截圖):

首先會先pass第一次，去標記所有pixel的label，並儲存label到labels裡面，其中labels的value會儲存每個key的parent(相鄰neighbor的最小label)，當labels的key等於label時，代表這個key是root。

當binary\_image的pixel為255的時候，會初始化neighbor\_labels的list。

接著會找出neighbor的方位以及pixel的位置，如果位置超過圖片範圍會跳過，如果沒超出範圍，則會把這個neighbor的位置上的label存到neighbor\_labels中。

當儲存完所有neighbor的label之後，會開始判斷現在這個pixel的label要是什麼。

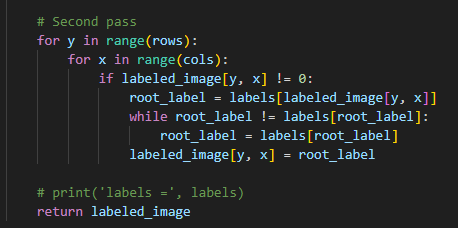
如果neighbor\_labels的長度是0，就會把現在這個位置填入now\_label的值，然後把這個label放到labels中，並將key和value設成now\_label，接著now\_label加上一。

如果neighbor\_labels有存在label時，會先找出neighbor\_labels的min\_label(裡面最小的label)，然後將現在的位置設定成min\_label。

在設定完現在位置的label後，我會讓neighbor\_labels的label在labels中設定他們的parent，所以會先找到label的root\_label是誰(label的root)。接下來根據min\_label是否為root決定他們的value要怎麼給。

如果min\_label是root代表我只需要min\_label跟root\_label比較誰是parent，然後把parent放到比較大的label中。

如果min\_label不是root，那我需要先找到root\_min\_label(min\_label的root)，然後比較root\_min\_label和root\_label誰是parent，然後把parent放到比較大的label中。

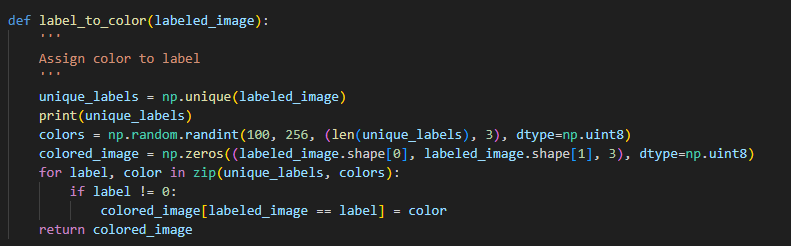


Function label\_components (分成三個截圖，這是第三個截圖):

在pass完第一次之後，會在pass第二次，這次主要是檢查每個pixel的label是否為root，並把不是root的label全部換成root(把不同label但是連在一起的pixel的label全部設成root的label)。

首先判斷每個有label的pixel的label是否為root。當label是root的話，就會檢查下一個pixel；當label不是root的話，根據第一次儲存的labels的所有label找到root，然後把現在pixel的label換成root\_label，當換完pixel後，會再檢查下一個pixel。

最後回傳labeled\_image。



Function label\_to\_color:

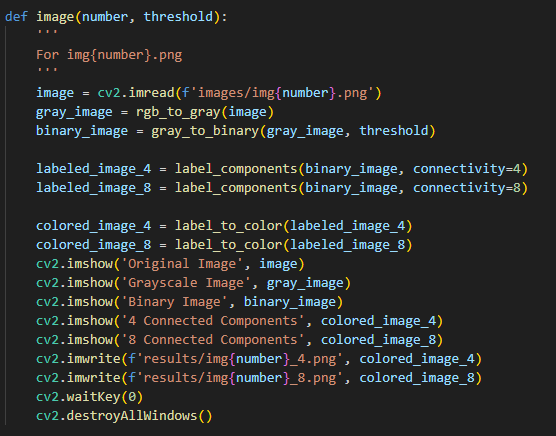
首先會用numpy的unique找出所有label，並存在unique\_labels中。

根據unique\_labels的長度給出相同長度的隨機顏色存入colors中，並將colors設定dtype=np.uint8，因為顏色的value範圍是100~255，然後將顏色的value的最小值設為100是防止太小的value會讓顏色太暗，跟背景的黑色不容易辨識。

將colored\_image初始化，利用numpy的zeros讓他的長和寬跟labeled\_image一樣，然後分成RGB三個value，dtype=np.uint8是因為RGB value的範圍為0~255。

接著把所有labeled\_image不為0的label的pixel根據unique\_labels和colors把colored\_image的RGB填上隨機顏色。

最後回傳colored\_image。



Function image:

先用cv2的imread讀取圖片到image，然後利用rgb\_to\_gray把image轉成gray\_image，再利用gray\_to\_binary把gray\_image轉成binary\_image。

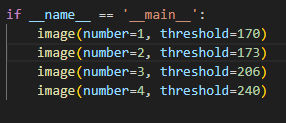
接下來用label\_components把binary\_image轉成labeled\_image\_4和labeled\_image\_8 (在參數設定connectivity=4，會使用4-connected，在參數設定connectivity=8，會使用8-connected)。

接著利用label\_to\_color將labeled\_image的label填上隨機顏色，把labeled\_image\_4轉成colored\_image\_4還有labeled\_image\_8轉成colored\_image\_8。

利用cv2的imshow查看original image、gray image、binary image、4-connected的labeled\_image加上隨機顏色的結果和8-connected的labeled\_image加上隨機顏色的結果。

利用cv2的imwrite將4-connected的labeled\_image加上隨機顏色的結果和8-connected的labeled\_image加上隨機顏色的結果存到results裡面。

利用cv2的waitKey(0)和destroyAllWindows()將imshow用顯示出來的圖片關閉。



執行image，產生所有助教給的圖片的4-connected的labeled\_image加上隨機顏色的結果和8-connected的labeled\_image加上隨機顏色的結果。

Result images

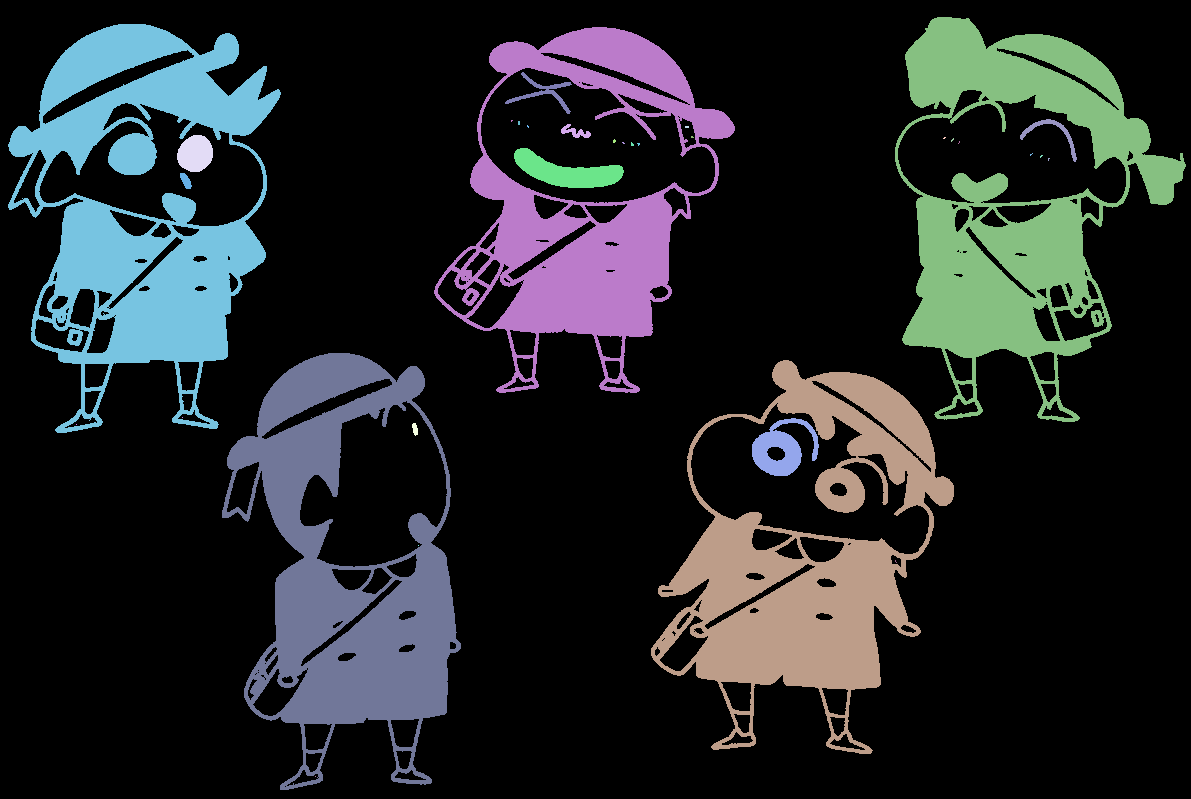
這次的Result images有很多縮小的時候看起來像是分開的圖案但是顏色相同，在放大檢視後，可以發現pixel是連在一起的，所以圖案的顏色才會是相同的。

img1\_4.png



我預期的結果跟最後的輸出相同，跟8-connected相差比較明顯的是書包的扣子和旁邊的方塊，4-connected的結果是分開的。

img1\_8.png



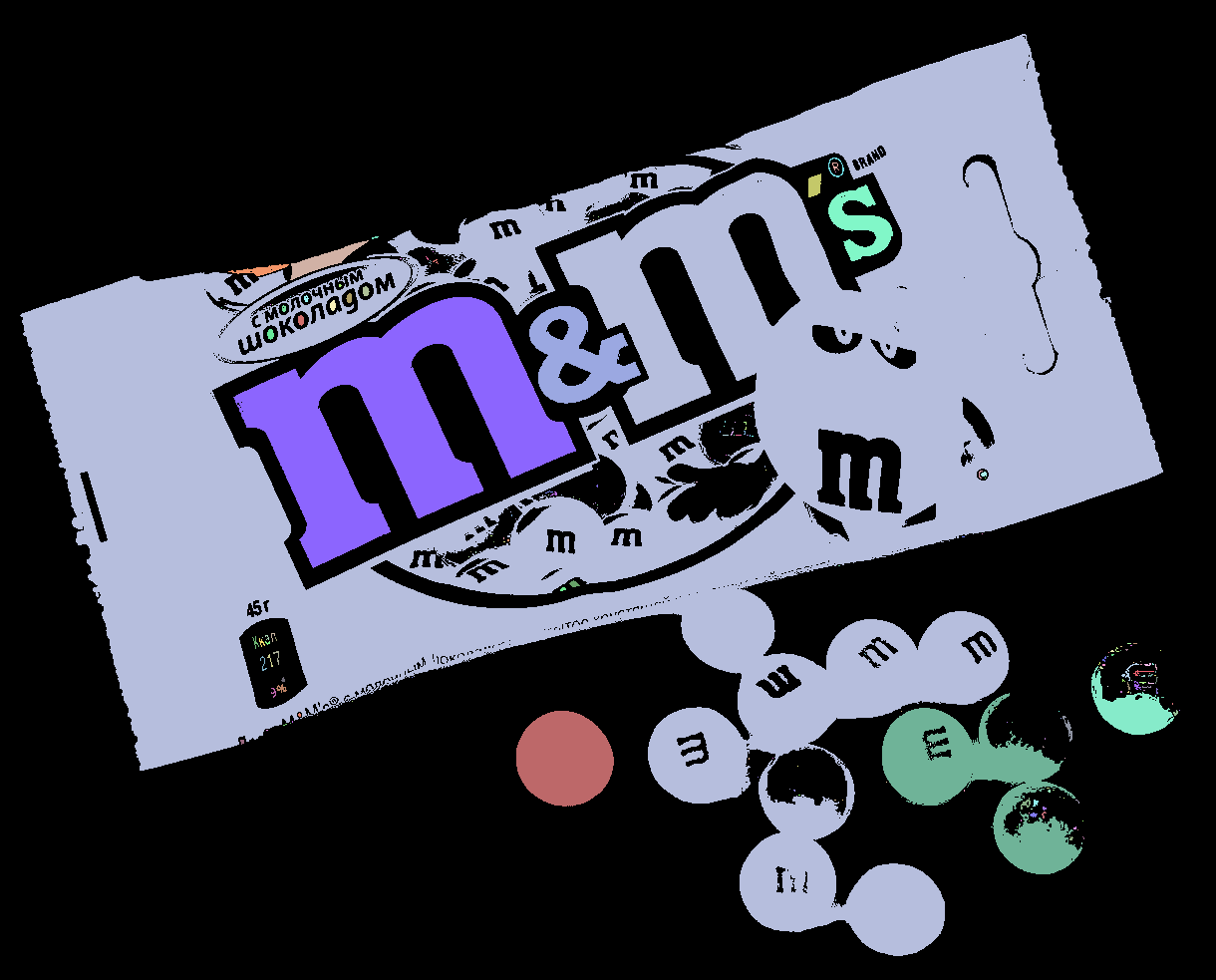
我預期的結果跟最後的輸出相同，跟4-connected相差比較明顯的地方是書包的扣子和旁邊的方塊，8-connected的結果是連在一起。

img2\_4.png



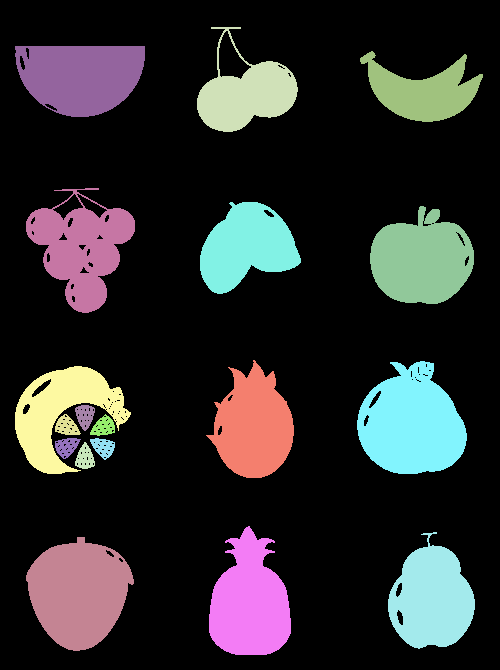
我預期的結果跟最後的輸出相同，放大看mm巧克力的袋子的左下方的數字(217)可以很明顯的看出4-connected和8-connected的差別，左上的mm巧克力球的邊邊可以看到有點藍灰的顏色，4-connected是分開的，而8-connected的部分是連起來的。

img2\_8.png



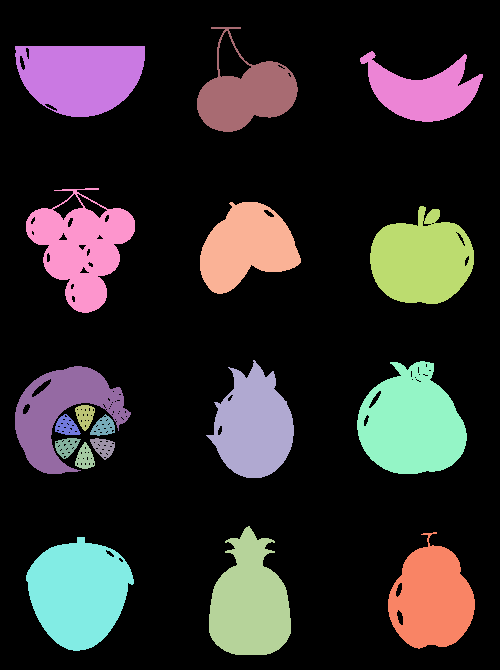
我預期的結果跟最後的輸出相同，放大看mm巧克力的袋子的左下方的數字(217)可以很明顯的看出4-connected和8-connected的差別，左上的mm巧克力球的邊邊可以看到跟4-connected的差別在8-connected的顏色跟包裝的顏色一樣。

img3\_4.png



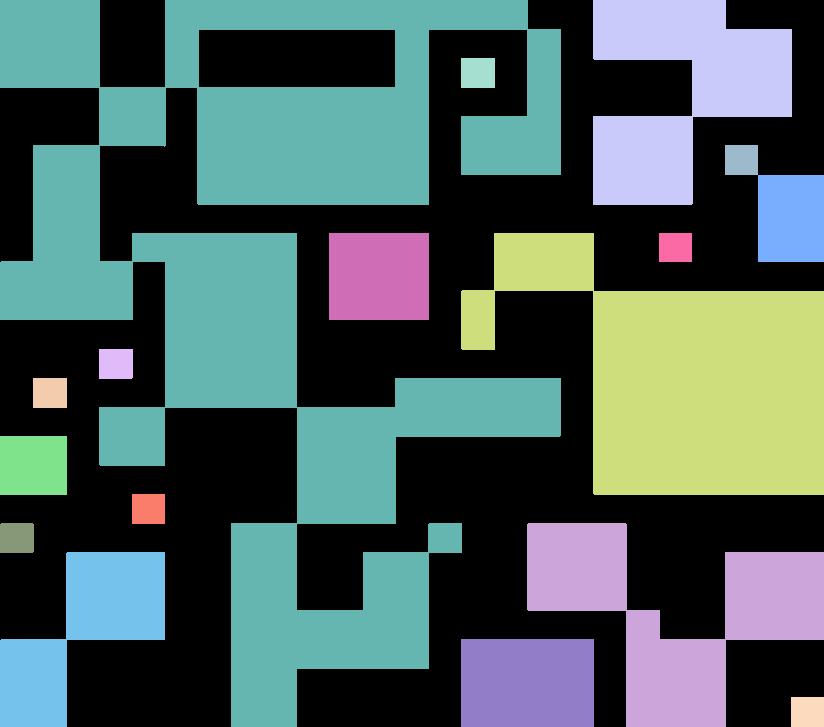
我預期的結果跟最後的輸出相同，跟8-connected的label數量一樣，所以只有顏色不同。

img3\_8.png



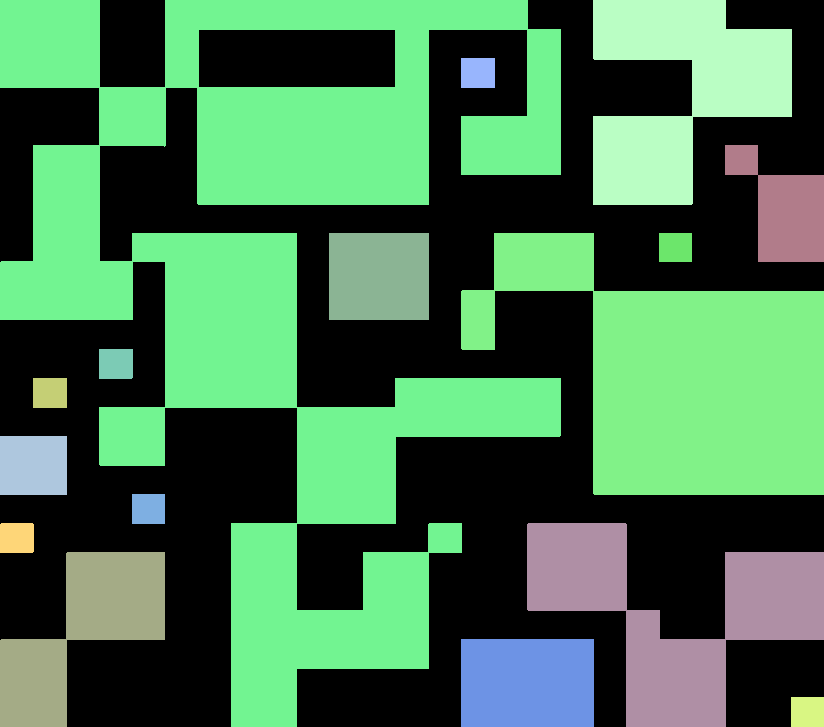
我預期的結果跟最後的輸出相同，跟4-connected的label數量一樣，所以只有顏色不同。

img4\_4.png



我預期的結果跟最後的輸出相同，雖然有很多方塊是連在一起的但是從右上方塊中，可以明顯看出8-connected跟4-connected的差別，4-connected的方塊是分開的。

img4\_8.png



我預期的結果跟最後的輸出相同，雖然有很多方塊是連在一起的但是從右上方塊中，可以明顯看出8-connected跟4-connected的差別，8-connected的方塊是連在一起的。

最後的結果總結：

在一開始寫hw2的時候有兩個難點，第一個是threshold的調整，沒調好threshold會讓圖片的結果變得很奇怪，像是有些圖片會有很pixel連在一起，會讓4-connected和8-connected的結果看起來差不多。

第二個是在寫label component的function的時候，我一開始在pixel有neighbor的時候，是直接將pixel設定為min\_label，然後把所有在labels中非min\_label的key的parent全部設為min\_label，這個寫法導致我loss掉許多原本非min\_label的key的parent，因此圖片會出現很多pixel連在一起但是不同的label。

而出現很多pixel連在一起但是不同的label的這個問題的解決方法是，我開始找非min\_label的label的root(root\_label)還有min\_label的root(min\_root\_label)，然後去比較這兩個root誰比較小，比較小的label設定為比較大的label的parent。

這次hw2中，我有更深刻的解到label component的實作方法，也發現binary image的threshold對label component的重要性。