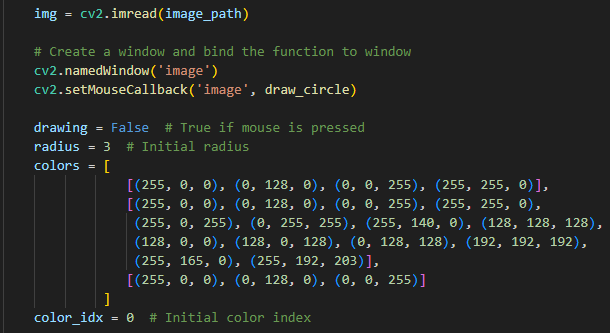
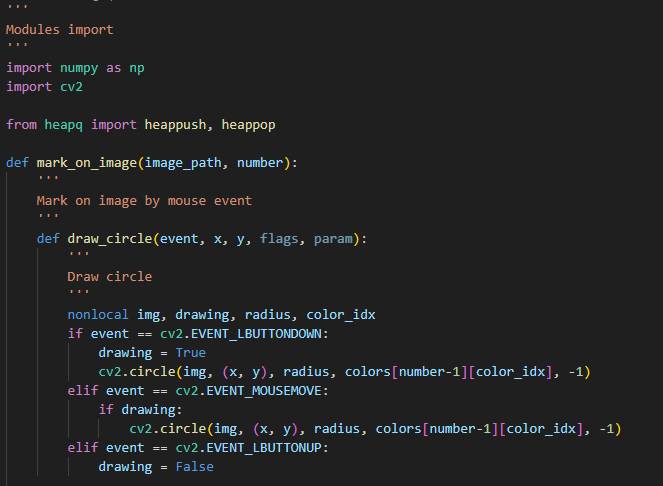
班級：資工三

學號：110590034

姓名：楊榮鈞



Function mark\_on\_image:

在此Function中有一個Function draw\_circle是要讓使用者用滑鼠畫畫時，畫出一個實心圓，其中利用cv2.EVENT\_LBUTTONDOWN、cv2.EVENT\_MOUSEMOVE和cv2.EVENT\_LBUTTONUP去偵測滑鼠的行為去決定現在是否再畫畫。

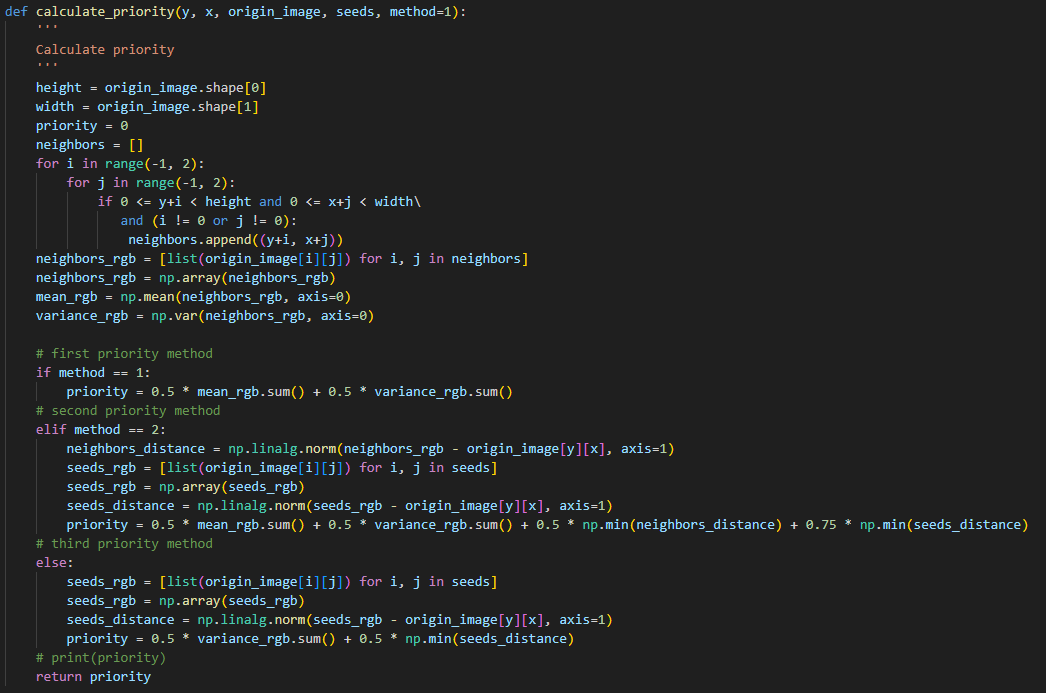
接著用cv2.imread讀取image\_path的圖片(原圖)，利用cv2.namedWindow和cv2.setMouseCallback建立畫畫的視窗。

初始化drawing(畫畫的狀態)、radius(畫出實心圓的半徑)、colors(三張原圖所需標記的顏色)和color\_idx(colors的index)。

接下來用while迴圈來執行畫畫的相關行為，首先用copy()複製一份原圖到img\_with\_text上，然後用cv2.putText顯示現在在使用的顏色(RGB)，用cv2.imshow顯示圖片。接著使用cv2.waitKey和ord去根據使用者所輸入的行為(keyboard input)做出不同的動作。

最後在使用cv2.destroyAllWindows關掉視窗。

按下q時，會關掉視窗；按下c時，會清空畫布(還原成沒有標記的狀態)；按下+時，會增加畫筆畫的實心圓半徑；按下-時，會減少畫筆畫的實心圓半徑；按下n時，切換到下一個顏色；按下s時，會儲存圖片(標記的圖片)。



Function calculate\_priority:

首先用shape找出origin\_image的height和width，然後初始化priority和建立neighbors，接著用for迴圈找到nieghbors。

然後我們找出neighbor的RGB的平均值和variance (mean\_rgb、variance\_rgb)，然後再根據method決定接下來的步驟。

當method為1，priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum()。

當method為2，會先計算當前pixel對neighbors的RGB value的distance到neighbors\_distance，然後再計算當前pixel對seeds的RGB value的distance到seeds\_distance，priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(neighbors\_distance) + 0.75 \* np.min(seeds\_distance)。

當method為其他值(ex: 3)，會計算當前pixel對seeds的RGB value的distance到seeds\_distance，priority = 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(seeds\_distance)。

最後回傳priority。



Function colorize\_watershed:

利用colors(標記用的顏色)和label\_map先產生出region\_image，在利用region\_image和origin\_image在watershed\_image上產生origin\_image加上region的薄霧顏色。

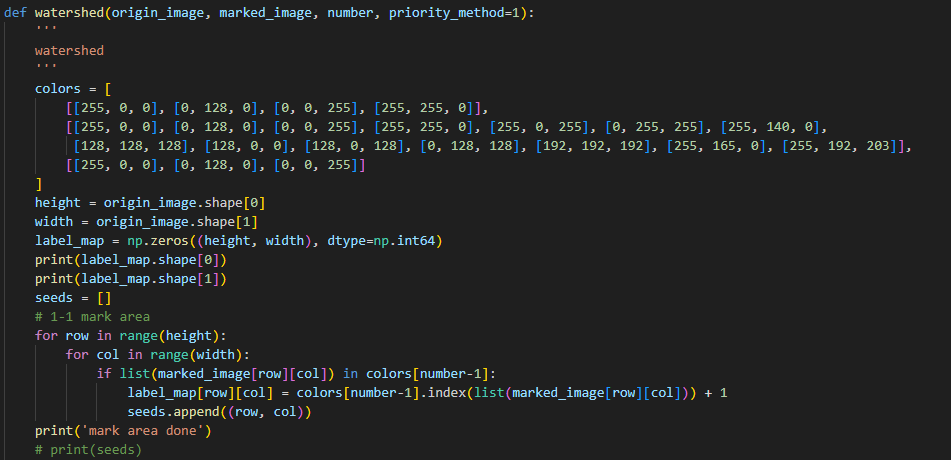
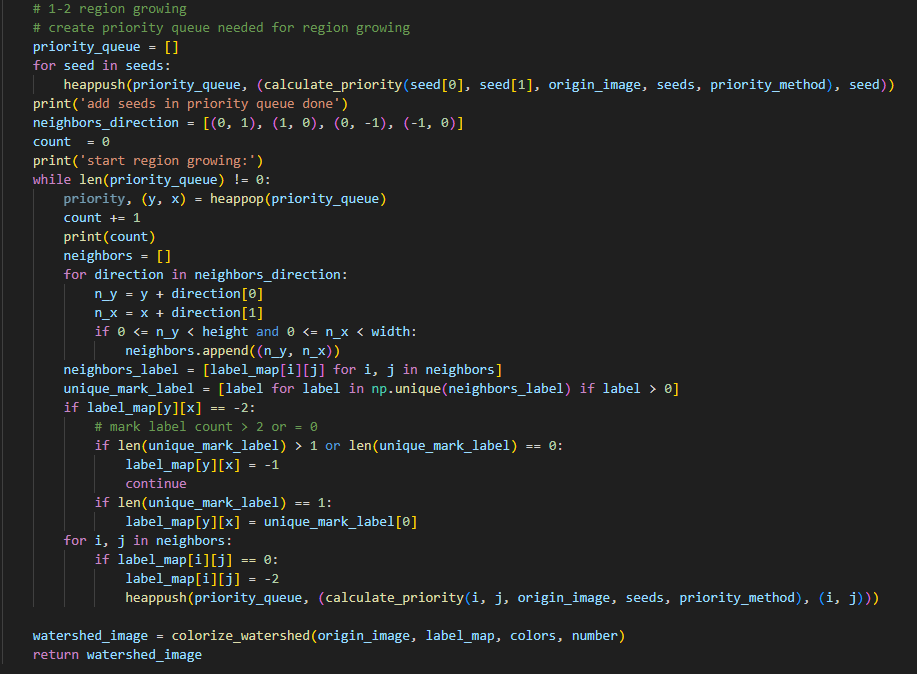
首先用shape找出origin\_image的height和width，在將origin\_image用astype轉成np.float64到o\_img上，然後將colors轉乘np.array並使用np.float64。

接著利用np.zeros建立region\_image的array，然後用for迴圈根據label\_map的值決定region\_image的顏色(當label\_map[row][col]>0時會將當前的pixel轉成label對應的colors中的顏色，若是小於等於0會直接設定為RGB(0,0,0))。

利用np.zeros建立watershed\_image的array，將region\_image用astype設定成np.float64，然後用for迴圈將label大於0(label\_map[row][col]>0)的pixel設定成o\_img對應的pixel\*0.5+region\_image對應的pixel\*0.5，使label小於等於0的值直接設成RGB(0,0,0)。

用cv2.imshow觀察region\_image，然後用cv2.waitKey(0)和cv2.destroyAllWindows()關閉視窗

最後回傳watershed\_image。

Function watershed:

初始化colors(建立標記的顏色)，利用shape找到origin\_image的height和width，使用np.zeros建立label\_map，建立seeds儲存一開始標記的位子。

用for迴圈找到marked\_image標記的位置，並將對應到此位置的label\_map設為對應colors的顏色的index+1，並將此位置存入seeds。

建立priority\_queue，然後先將seeds裡面seed的值用heapq的heappush將calculate\_priority的值和當前的pixel位置存入priority\_queue中。

建立neighbors\_direction(4 neighbor，所以是右上左下)，並初始化count(用來觀察priority\_queue處理了多少pixel)。

利用while迴圈label，當priority\_queue為empty時跳出迴圈。首先先使用heapq的heappop將priority最高(值最小)的pixel的位置取出，然後用for迴圈尋找這pixel的neighbor並將neighbor的位置存入neighbors中。

將neighbor對應到label\_map的label存入neighbors\_label中。

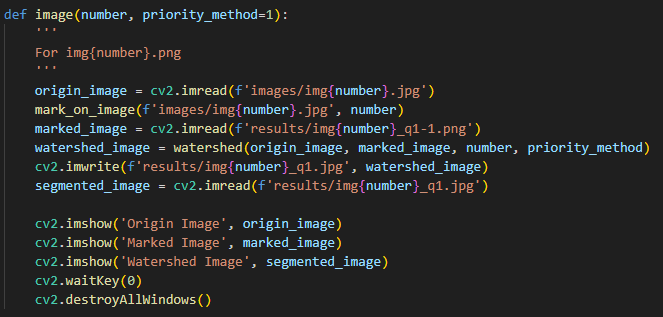
利用neighbors\_label找出大於0的unique label存入unique\_mark\_label中。

如果當前的pixel對應到label\_map的值為-2 (in queue)時，會判斷周圍是否有被標記的label(>0)，當被標記的label的數量超過1個或是沒有被標記的label時，當前pixel對應到label\_map的值會設為-1 (edge)，並進入下個迴圈；若是neighbor中被標記的label數量只有1個，那當前pixel對應到label\_map的值會設為那個label的值。

然後用for迴圈判斷neighbor對應到label\_map的值是否為0 (unmark)，如果是的話會將對應到label\_map的值設為-2 (in queue)，並使用heapq的heappush將calculate\_priority的值和neighbor的pixel位置存入priority\_queue中。

當priority\_queue為empty，然後跳出迴圈後，用colorize\_watershed將origin\_image、標記好的label\_map、colors和number(當前原圖的編號)轉換成watershed\_image。

最後回傳watershed\_image。



Function image:

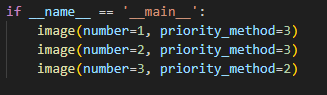
先用cv2的imread讀取圖片到origin\_image，然後傳入origin\_image的path到mark\_on\_image中。

接著用cv2的imread讀取marked\_image，然後將origin\_image和marked\_image、number還有priority\_method傳入watershed進行watershed segmentation，並將輸出結果存到watershed\_image。

接下來用cv2的imwrite儲存圖片，再用cv2.imread讀取segmented\_image(用watershed過後的image)。

利用cv2的imshow查看original image、marked image和watershed image。

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



執行image，產生所有助教給的圖片的marked image和watershed image。

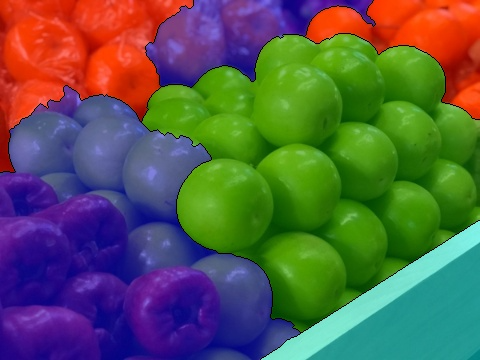
Result images

img1\_q1-1.png



img1\_q1.jpg (results裡面的是第三種priority算法)

priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum()



我預期的結果跟最後的輸出不太相同，藍色的region超出的範圍有點多。

priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(neighbors\_distance) + 0.75 \* np.min(seeds\_distance)



我預期的結果跟最後的輸出不太相同，相比於第一種的priority的圖上面藍色region的部分標示的範圍已經很漂亮了，但是又下的藍色region仍然有超出範圍的問題，且有一部份的region被綠色region給覆蓋掉了。

priority = 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(seeds\_distance)



我預期的結果跟最後的輸出大致相同，只有下面的藍色region有覆蓋到一點綠色region，綠色region也有覆蓋到一點藍色region。

img2\_q1-1.png



img2\_q1.jpg (results裡面的是第三種priority算法)

priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum()



我預期的結果跟最後的輸出大致上相同，只有中間紫色region的硬幣有點超出範圍。

priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(neighbors\_distance) + 0.75 \* np.min(seeds\_distance)



我預期的結果跟最後的輸出大致上相同，只有中間紫色region的硬幣有點超出範圍，上面粉紅region的硬幣的region有較完整一點。

priority = 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(seeds\_distance)



我預期的結果跟最後的輸出大致上相同，只有中間紫色region的硬幣有點超出範圍，上面粉紅region的硬幣的region有較完整一點，右上的硬幣region有完整覆蓋到(相對於前兩個priority算法)。

img3\_q1-1.png



img3\_q1.jpg (results裡面的是第二種priority算法)

priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum()



我預期的結果跟最後的輸出大致上相同，樹的只有樹葉較稀疏的有些部分被分到紅色的region。

priority = 0.5 \* mean\_rgb.sum() + 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(neighbors\_distance) + 0.75 \* np.min(seeds\_distance)



我預期的結果跟最後的輸出大致上相同，左邊的樹相比於第一個priority的算法有被藍色的region更完整的覆蓋到。

priority = 0.5 \* variance\_rgb.sum() + 0.5 \* np.min(seeds\_distance)



我預期的結果跟最後的輸出不太相同，左邊的樹的部分沒有像第二個priority method這麼好。

最後的結果總結：

這次hw4的第一個難點，我覺得是mark image的部分，因為不熟悉cv2畫圖的功能，所以花了很多時間在上面。第二個難點是q1-2的region growing的部分，其中最難的部分是priority queue的priority要如何設定。而這次作業我覺得最精華的部分也是priority的部分，首先我priority有先用過neighbor的RGB平均值\*0.5+neighbor的RGB的variance\*0.5，這樣的結果在作業的第一張圖上會看到有些region會超出範圍，第二張圖中桌面的另一邊會沒有標記到，而在第三張圖中region超出範圍的影響較小。

第二次我用的priority是neighbor的RGB平均值\*0.5+neighbor的RGB的variance\*0.5+min(當前pixel的RGB到neighbors的RGB的distance)\*0.5+min(當前pixel的RGB到seeds的RGB的distance)\*0.75，作業的第一張圖的結果的上半部分的藍色region有變的沒超出範圍，但是底下藍色region的部分仍有些許超出範圍，且有些被綠色region覆蓋到，第三張圖的左邊的樹的region有比較完整。

第三次我用的priority是neighbor的RGB的variance\*0.5 + min(當前pixel的RGB到seeds的RGB的distance)\*0.5，我覺得前兩張圖有region的比較好一點，但是第三張圖就沒有第二次用的priority這麼好了。

其中使用第二種和第三種方法花的時間會比較久，除非畫筆在image上mark的時候把圓變小一點，不然在priority算seeds distance時會算很久，因為畫筆比較粗的話，那seed就會變多，而priority\_method 2和3都會計算pixel到每個seed的RGB value的distance，目前img1\_q1-1.png、img2\_q1-1.png和img3\_q1-1.png產生每張個圖片的時間為40~50min。

我還有發現priority也可以配合老師剛剛教到的sebol來去做priority的計算。我的計算priority沒有使用到降階，如果使用降階，可以減少產生圖片的時間，然後mark image的部分也要注意轉成jpg還是png，轉成jpg時，會有些標記的label顏色會變淡，而png則沒有問題。

除了priority的部分外，後來也有發現當Mark的位置不同時，也會影響到watershed的結果(因為priority的算法會導致結果整個不一樣)。