Computer Vision HW2, Basic Image Manipulation Report

tags: `NTU CS` `Computer Vision` `Writeup` `Report`

NTU CSIE, R08922024, Alfons Hwu

Prequisites and env as the following

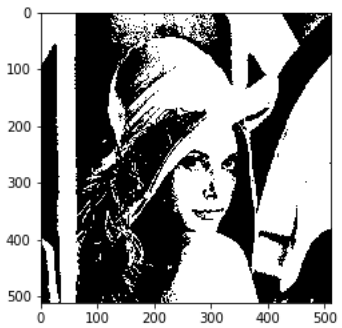
```
Ubuntu WSL for windows with jupyter notebook
Python3.6.7
OpenCV for image IO
Matplotlib for displaying image
```

Simply execute the program with `python3 hw2.py`

a, generate a binary image

```
def img_binarize(img_in):
    return (img_in > 0x7f) * 0xff
```

Simply binarize the image with simple python code.



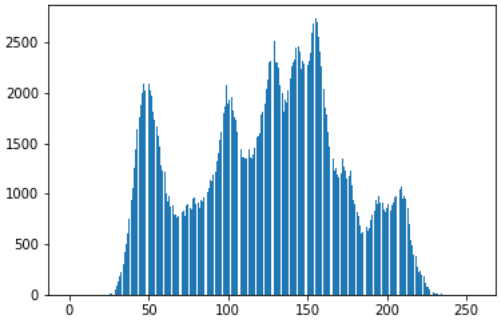
Time complexity: $O(MN)$

b, image histogram

```
def img_hist(img_in):
    hist = [0 for i in range(256)]

    row, col= img_in.shape
    for i in range(0, row):
        for j in range(0, col):
            hist[img_in[i, j]] += 1
```

Iterate through the image, map `[0, 255]` into list(same as the function of C++ map) to store the statistical data of image.



Time complexity: $O(MN)$

c, connected components

In this problem, we hope to segment the image into different connected components(or disjoint sets).

The idea and algorithm is the same as this problem --> https://www.cnblogs.com/hfc-xx/p/4666223.html (https://www.cnblogs.com/hfc-xx/p/4666223.html) in the UVA online judge, i.e. try to count the occurrances of fully connected '@', yet he uses the recursive dfs algorithm which is **not applicable here** due to the depth of recursion causes **stack overflow**.

In this problem, I switched to the iterative solution. Iterating through all the image pixel one by one and setting the child -> parent relation for grouping the CCs simultaneously.

Moreover, to reduce the time complexity, **path compression algorithm** is used to flatten the search tree, reducing the worst case scenario from $O(N)$ to search from leaf to root to directly the root(parent), amortized time complexity can even close up to $O(1)$

Check this --> https://www.geeksforgeeks.org/union-find-algorithm-set-2-union-by-rank/ (https://www.geeksforgeeks.org/union-find-algorithm-set-2-union-by-rank/) and search **path compression** for more details about this algorithm.

Path + union find comporession part

```
def union_find(label):
    original_label = label
    cnt = 0
    row, col = cc_img.shape
    while label != parent_label[label] and cnt < row * col:
        label = parent_label[parent_label[label]]
        cnt += 1

    parent_label[original_label] = label # path compression to avoid TLE
    return label
```

Pixel iteration and build child->parent relationship

```
 # set parent label
    row, col = cc_img.shape
    for i in range(row * col):
        parent_label.append(i)

    # do connected components
    label = 2
    for i in range(row):
        for j in range(col):
            ok1 = 0
            ok2 = 0
            if cc_img[i, j] == 1:
                if j - 1 >= 0 and cc_img[i, j - 1] > 1: # left has already labeled
                    cc_img[i, j] = union_find(cc_img[i, j - 1])
                    ok1 = 1

                if i - 1 >= 0 and cc_img[i - 1, j] > 1: # up has already labeled
                    if ok1: # set the connected component to make left = up as the same
                        parent_label[cc_img[i, j]] = union_find(cc_img[i - 1, j])
                    else:
                        cc_img[i, j] = cc_img[i - 1, j]

                    ok2 = 1

                if ok2 == 0 and ok1 == 0:
                    cc_img[i, j] = label
                    label += 1

    # union and find merging
    for i in range(row):
        for j in range(col):
            if cc_img[i, j] > 1:
                cc_img[i, j] = union_find(cc_img[i, j])
```

Geometric pattern drawing with cv2 libraries

```
def draw_rect(u, d, l, r, color):
    cv2.rectangle(rgb_img, (l, u), (r, d), color, 2)

def draw_cent(cen_i, cen_j, color):
    cv2.line(rgb_img, (cen_j - SHIFT, cen_i), (cen_j + SHIFT, cen_i), color, 2)
    cv2.line(rgb_img, (cen_j, cen_i - SHIFT), (cen_j, cen_i + SHIFT), color, 2)
```

The iteration counts accumulates each operation in union find and iterations pixelwise for time complexity. With $663388/(512^2) = 2.5$
Time complexity: Average $O(MN)$