ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÀI TẬP LỚN

MATHEMATICAL FOUNDATION FOR COMPUTER SCIENCE

Nhóm CSTH-ABC :

- Lê Phương Nam - 2170545
- Ngô Quốc Khánh - 1812593
- Lâm Phùng Phước Vinh - 2270093
- Nguyễn Quốc Anh - 2270074

TP.HCM, 2022

Connected-component labeling

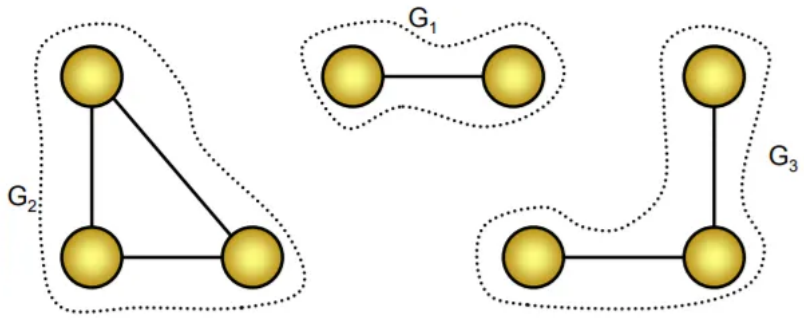
I. Một số khái niệm.

1. Khái niệm Connected-component (Thành phần liên thông)

a. Đối với đồ thị vô hướng G = (V,E)

G gọi là liên thông (connected) nếu luôn tồn tại đường đi giữa mọi cặp đỉnh phân biệt của đồ thị

Nếu G không liên thông thì chắc chắn nó sẽ là hợp của hai hay nhiều đồ thị con liên thông, các đồ con này đôi một không có đỉnh chung. Các đồ thị con liên thông rời nhau như vậy được gọi là các thành phần liên thông (connected-component) của đồ thị đang xét.



Hình 1. Đồ thị G và các đồ thị liên thông của nó.

Tương tự, những cạnh mà khi ta bỏ đi sẽ tạo ra một đồ thị có nhiều thành phần liên thông hơn so với đồ thị ban đầu gọi là một cạnh cắt hay một cầu.

b. Đối với đồ thị có hướng G = (V,E)

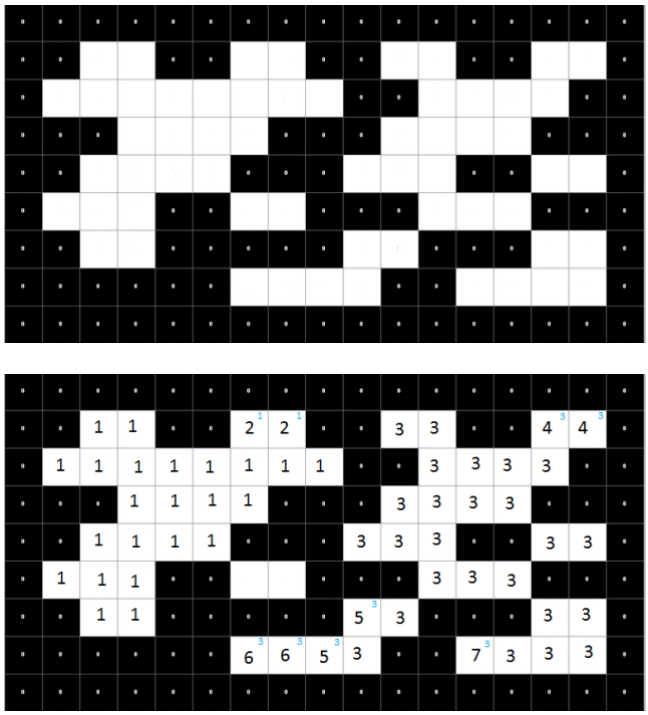
Có hai khái niệm về tính liên thông của đồ thị có hướng, tùy thuộc việc chúng ta có xem xét đếnhướng của các cạnh (cung) hay không :

- Đồ thị liên thông mạnh (Strongly connected) nếu luôn tồn tại các đường đi (theo hướng của cung) giữa hai điểm bất kỳ của đồ thị, G lúc này được gọi là đồ thị liên thông mạnh.
- Đồ thị liên thông yếu (Weakly connected) nếu luôn tồn tại các đường đi (không xét hướng của cung) giữa hai điểm bất kỳ của đồ thị, thì G là đồ thị liên thông yếu.

2. Connected-component labelling

Connected-component labelling (Gán nhãn thành phần liên thông) là một ứng dụng thuật toán của *lý thuyết đồ thị*, nơi tập hợp con của *các thành phần liên thông* được gắn nhãn duy nhất dựa trên *heuristic*. Gán nhãn thành phần liên thông không được nhầm lẫn với *phân vùng (phân đoạn) hình ảnh.*

Gán nhãn thành phần liên thông được sử dụng trong lĩnh vực thị giác máy tính để phát hiện các vùng liên thông trong ảnh nhị phân, các ảnh màu và độ phân giải cao vẫn có thể xử lý được.



Hình 2. Input và output mong muốn của một bài toán gán nhãn thành phần liên thông

II. Một số thuật toán gán nhãn thành phần liên thông.

1. Giải thuật cổ điển (The classical algorithm).

a. Giới thiệu.

Giải thuật được giới thiệu bởi **Rosenfeld** và **Pfaltz** vào năm 1966 sử dụng union-find data structure để giải quyết vấn đề và đạt hiệu suất khá cao. Gọi nó là giải thuật "cổ điển" vì giải thuật này sử dụng một kết quả từ một thuật toán cổ điển sử dụng cho sự liên thông trong lý thuyết đồ thị.

b. Thuật toán

Thuật toán bao gồm hai lần duyệt. Trong lần duyệt đầu tiên, thuật toán sẽ ghi lại các tương đương và đánh index tạm cho các điểm pixel đó. Trong lần duyệt thứ hai, thuật toán sẽ thay các index tạm bằng index của các lớp tương đương đó.

Union-find Data Structure: Cấu trúc Union-find sẽ theo dõi các tập hợp các phần tử được chia vào các tập con rời rạc (disjoint subsets), nhằm thực hiện các thao tác:

- Union: hợp hai tập con vào làm một.

- Find: Chỉ ra tập con mà phần tử thuộc về.

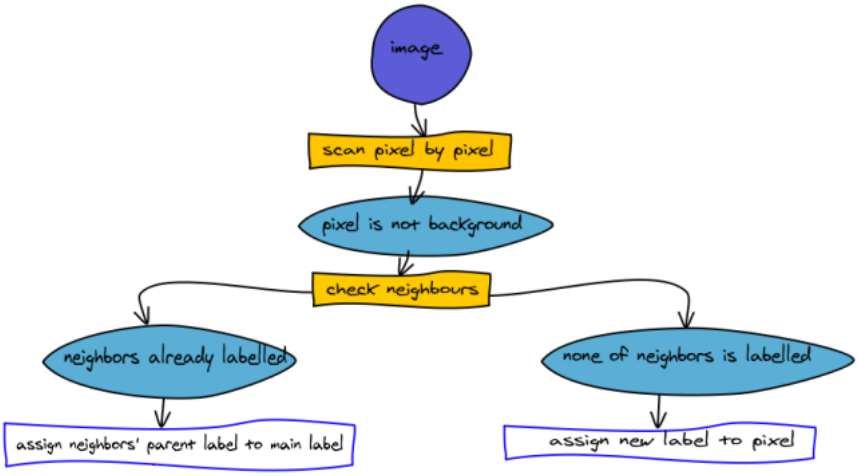
Lần duyệt đầu tiên (gán nhãn)

Đối với mỗi pixel khác 0, kiểm tra các điểm láng giềng của nó.

- Nếu nó không có láng giềng khác 0 - vậy nó là một thành phần mới - đặt cho nó một nhãn mới.
- Nếu nó có một lân cận khác 0 - các pixel này được kết nối với nhau - cho nó cùng nhãn với láng giềng.
- Nếu nó có nhiều hơn một láng giềng khác 0, có hai trường hợp:

1. Mọi láng giềng đều có cùng một nhãn . Vậy, gán cho pixel đang xét cùng nhãn.
2. Các láng giềng có nhãn khác nhau. Các pixel này đáng ra phải có nhãn giống nhau, chúng ta giải quyết như sau: Đầu tiên, đặt pixel hiện tại thành nhãn thấp nhất trong các láng giềng. Sau đó, lưu ý về sự tương đương của tất cả các nhãn được kết nối - nghĩa là các nhãn khác nhau phải giống nhau. Sự khác nhau này sẽ được giải quyết trong lần duyệt thứ hai.

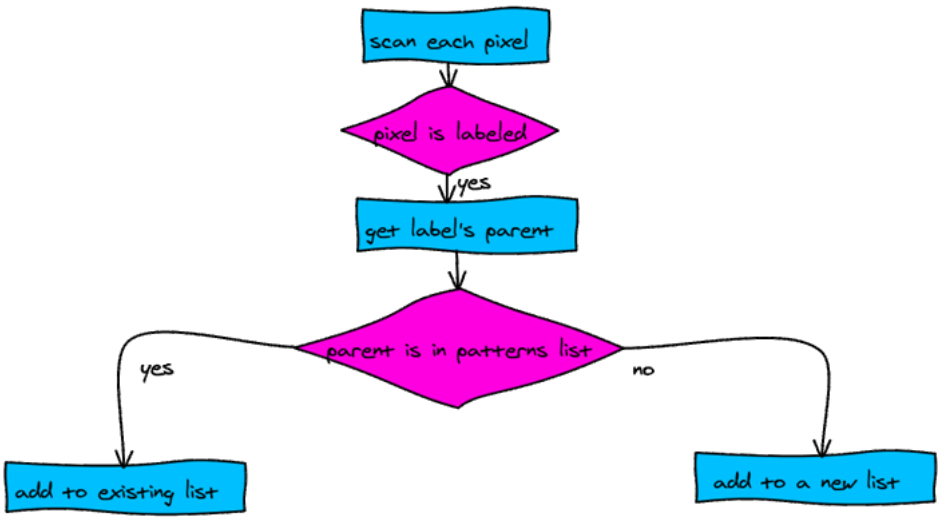
Mỗi pixel khác 0 bây giờ sẽ có một nhãn. Tuy nhiên, một số vùng được kết nối sẽ có các nhãn khác nhau. Vì vậy, cần xem lại hình ảnh một lần nữa để khắc phục điều này. Thực hiện điều này bằng cách sử dụng các bản ghi về các lớp tương đương: tất cả các nhãn tương đương (tức là tham chiếu đến cùng một đốm màu) sẽ nhận được cùng một nhãn.



Hình 3. Lưu đồ thuật toán cho lần duyệt đầu tiên.

Lần duyệt thứ hai (tổng hợp)

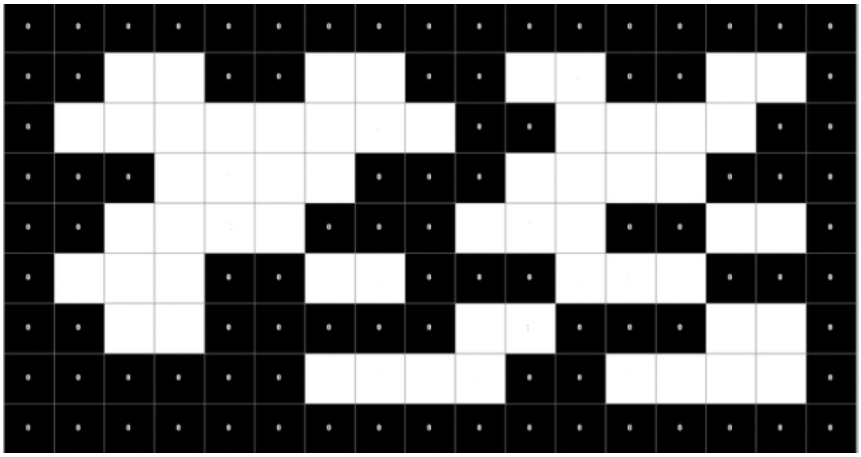
Kiểm tra xem nhãn này có các nhãn tương đương hay không và giải quyết theo các bước.



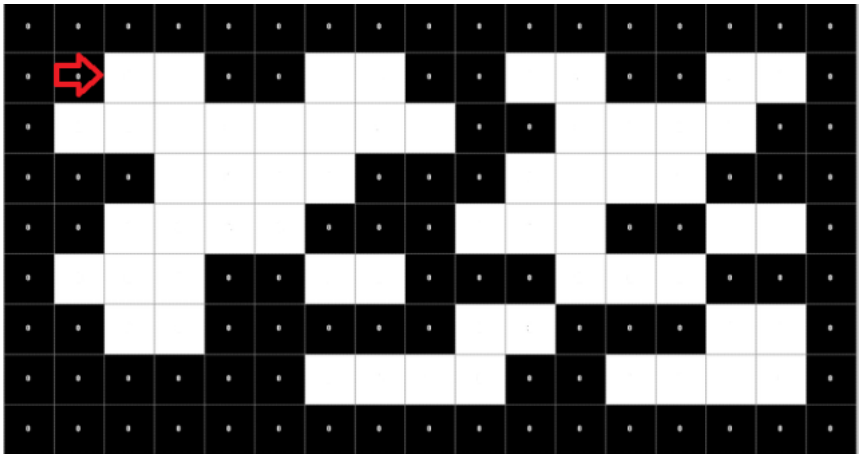
Hình 4. Lưu đồ thuật toán cho lần duyệt thứ hai.

Xét một ví dụ về lần duyệt thứ hai

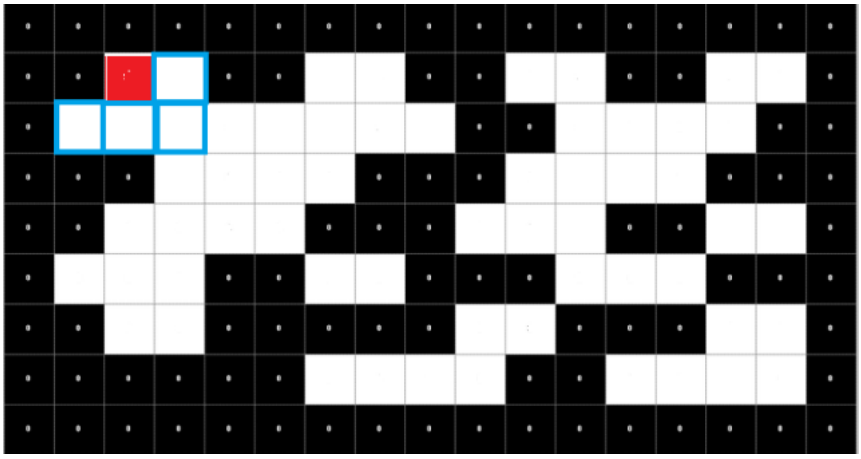
- Với input như bên dưới, bắt đầu với currentLabelCount = 1.



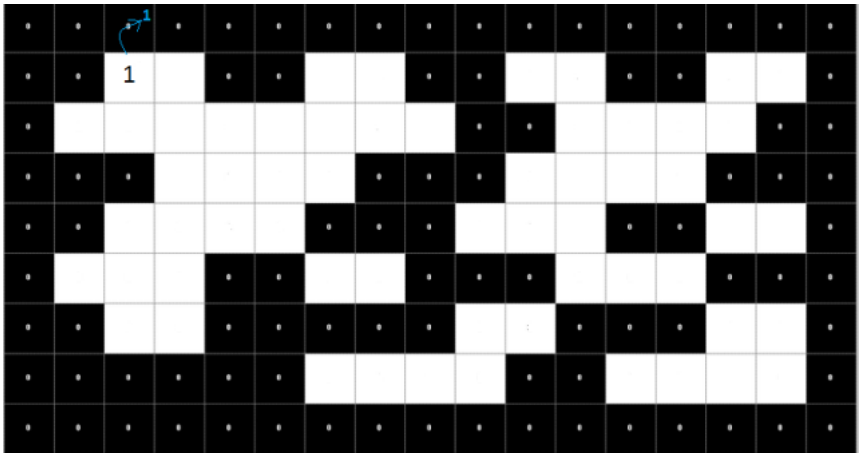
- Duyệt đến pixel đầu tiên không phải là nền (0)



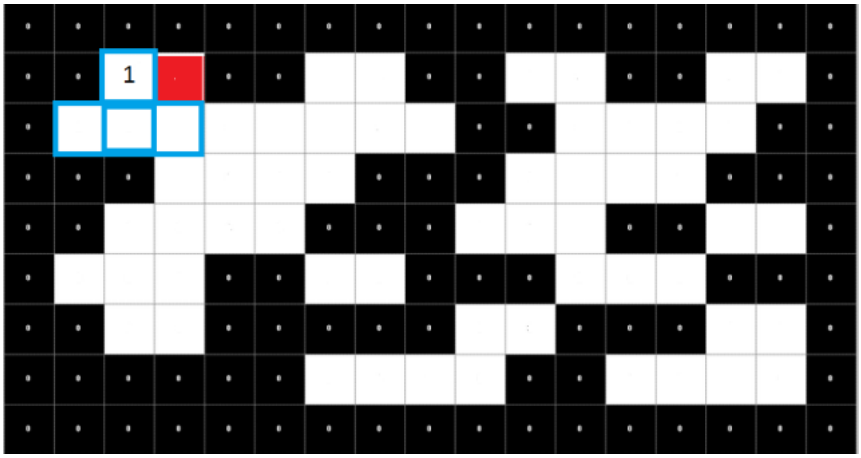
- Có được các láng giềng không phải nền:



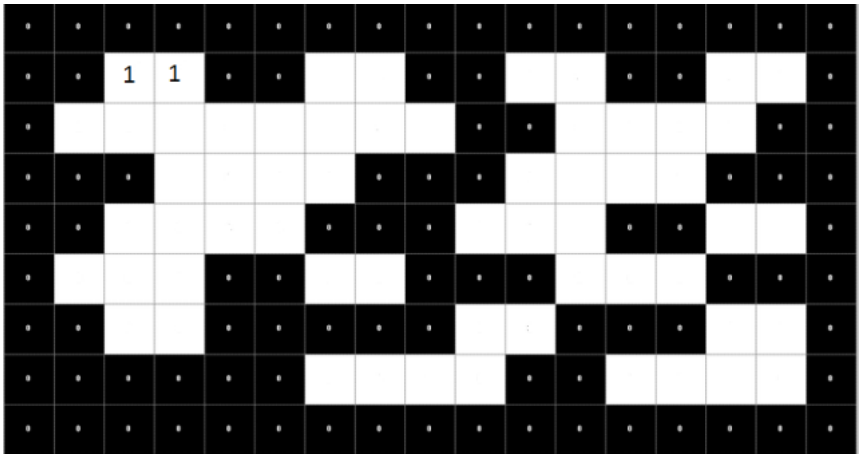
- Không có láng giềng nào được gắn nhãn, đặt pixel hiện tại thành currentLabelCount và đặt index gốc của nhãn thành chính nó :



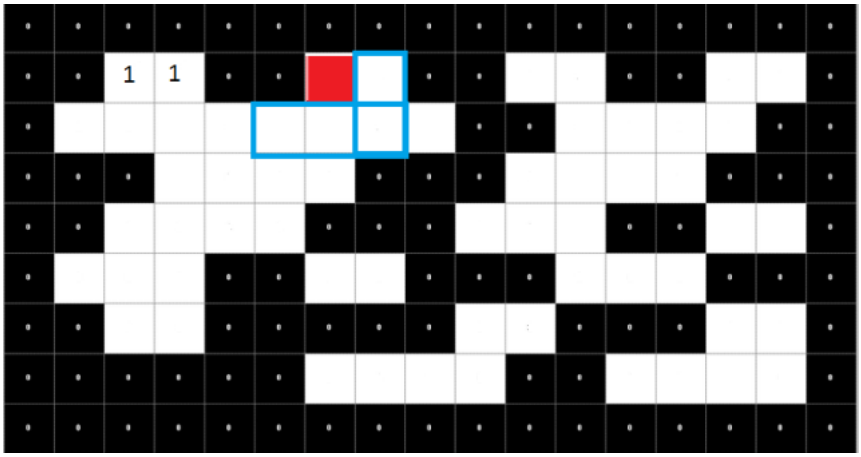
- Đến pixel tiếp theo, pixel này có một hàng xóm đã được gắn nhãn:



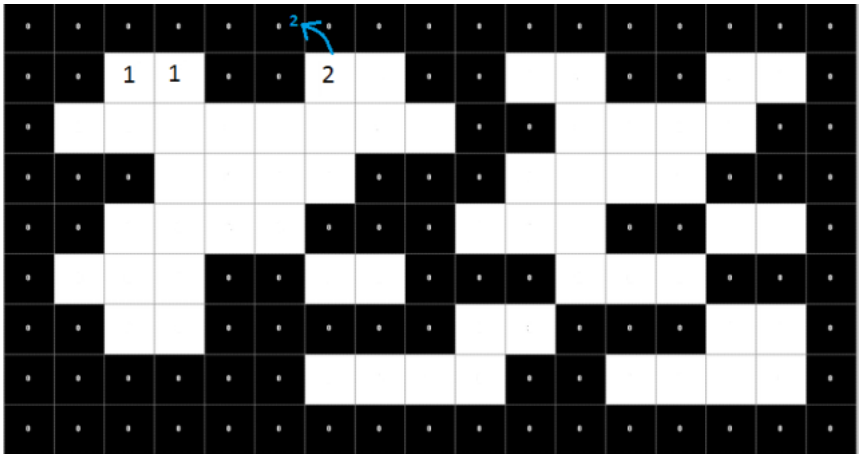
- Gán nhãn của pixel cho nhãn của láng giềng:



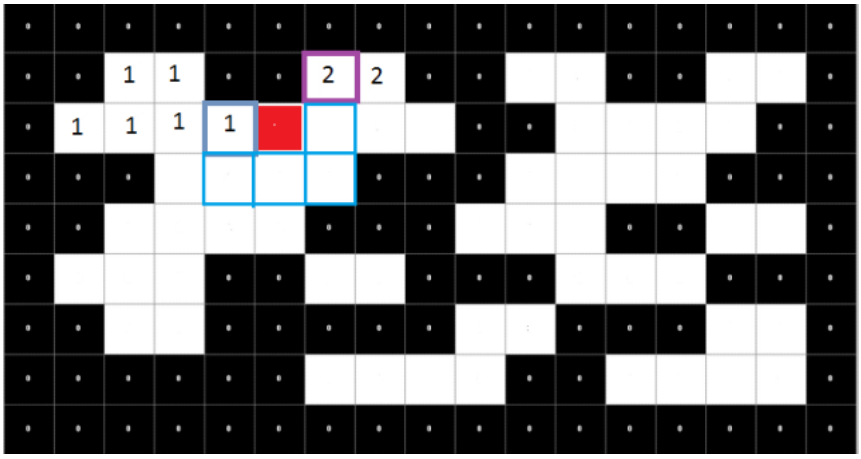
- Xét tiếp, không có láng giềng nào của pixel này được gắn nhãn:



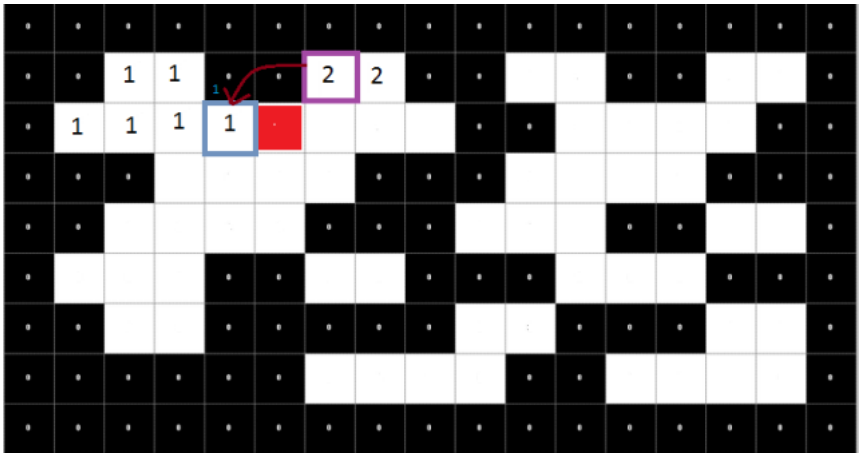
- Tăng currentLabelCount và gán nó cho pixel, một lần nữa nhãn của nó được đặt thành chính nó:



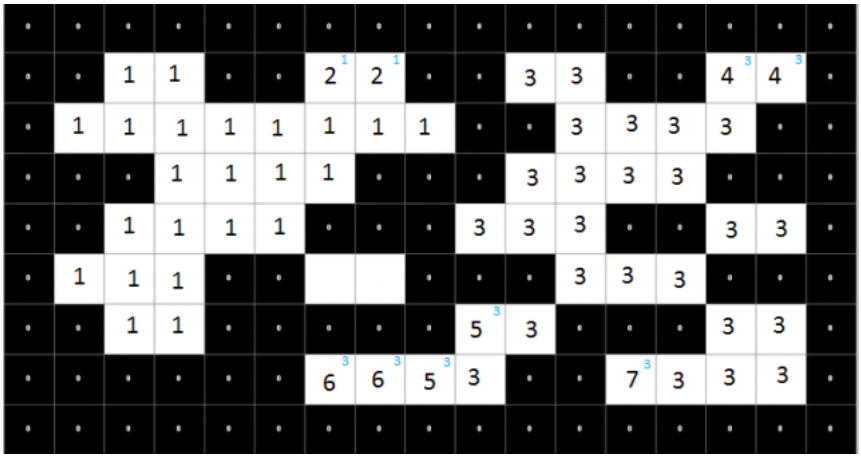
- Pixel đang xét này có láng giềng mang các nhãn khác nhau, sẽ thực hiện theo các bước đã nêu trên.



- Chọn nhãn chính, tức là: đó sẽ là nhãn nhỏ nhất trong danh sách đã gán (1).
- Đặt nó là nhãn gốc của các nhãn khác.



Lặp lại các bước trên sẽ có được kết quả như sau. Lưu ý số màu xanh lam ở góc trên bên phải, đó là nhãn chính.



- Thuật toán kết thúc, lưu trữ lại nhãn.

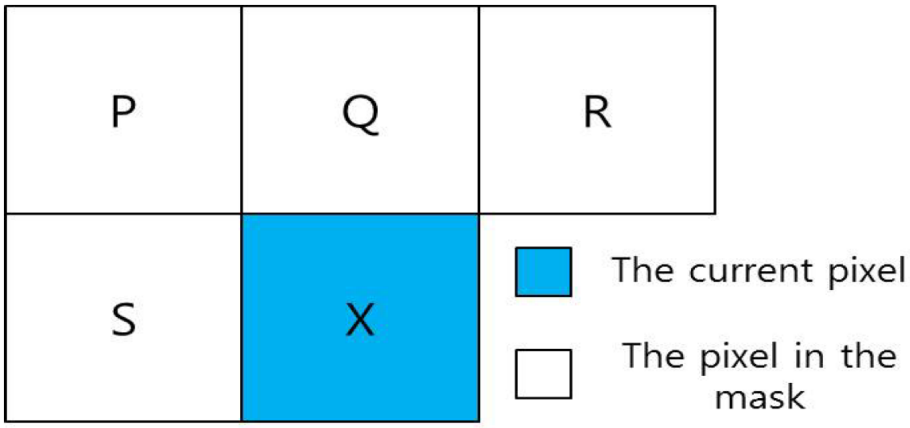
Độ phức tạp của thuật toán: O (N2)

2. Fast region expansion connected component labelling algorithm.

a. Giới thiệu

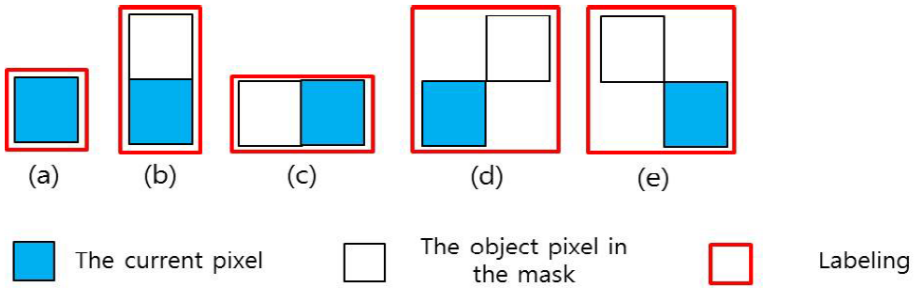
Thuật toán được giới thiệu ở ISCE năm 2014 bởi nhóm nghiên cứu tại Sungkyunkwan University, Sunwon, Hàn Quốc. Với quy trình quét một lần, pixel nền trước (đối tượng) được gán nhãn tạm thời và nhãn tương đương giữa các nhãn tạm thời được hợp nhất sau đó. Cách tiếp cận là hợp nhất vùng ghi nhãn của hình ảnh tiền cảnh trong quá trình gán nhãn trong một lần quét. So với các thuật toán gắn nhãn tại thời điểm đó, kết quả thử nghiệm thuật toán có độ phức tạp thấp hơn.

b. Thuật toán



Hoạt động cơ bản của mọi lần lặp như sau. Như thể hiện trong hình trên, chúng ta mang mặt nạ quét cho đến khi chúng tôi chạm vào pixel X của đối tượng không được gắn nhãn trong hình ảnh đầu vào. Sau đó, pixel đối tượng X được gắn nhãn cùng số nhãn với một trong các pixel lân cận P, Q, R, S. Nếu xảy ra xung đột nhãn, sự tương đương có thể được giải quyết bằng cách chọn một số thấp hơn, như là:

label (X) = min (label (P), label (Q), label (R), label (S).



Quy tắc mở rộng: Có bốn trường hợp mở rộng vùng khác nhau khi có cùng số nhãn, như thể hiện trong hình trên. Các pixel lân cận phù hợp với X được hợp nhất thành một hình chữ nhật mở rộng được đánh dấu bằng màu đỏ.

Chi tiết thuật toán như sau:

Algorithm: Iterative Region Expansion – Mở rộng vùng lặp lại

Bước 1: Xác định đối tượng pixel X chưa được gắn nhãn trong ảnh đầu vào

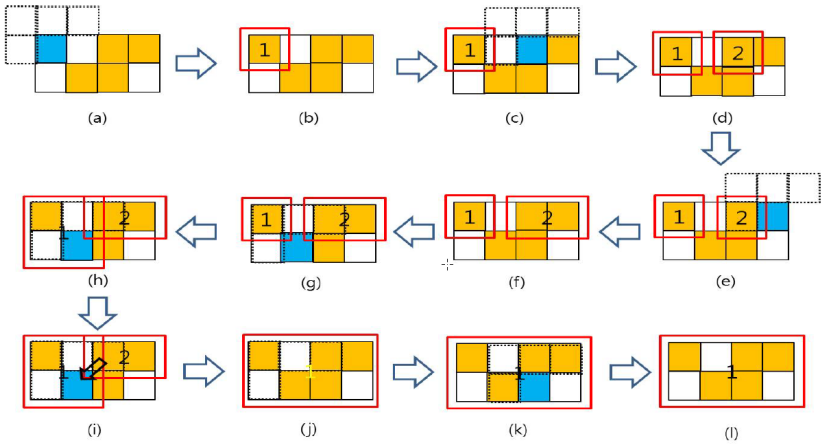
Bước 2: Nếu pixel X có một hoặc nhiều pixel đã được gắn nhãn lân cận thì:

- Nếu pixel X đã bao gồm nhãn được chỉ định thì ta sẽ bỏ qua và nhảy đến ô tiếp theo
- Nếu tồn tại nhãn P – Label(P) hoặc tồn tại nhãn Q – Label(Q) hoặc tồn tại nhãn R – Label(R) hoặc tồn tại nhãn S – Label(S), thì nhãn X là*Label(X) = min (Label(P), Label(Q),Label(R), Label(S))*

Bước 3: Nếu pixel X không có pixel lân cận được gắn nhãn, thì pixel X sẽ được gắn một nhãn mới.

Bước 4: Nếu vẫn còn pixel chưa được gắn nhãn, hãy chuyển sang **Bước 1**.

Ví dụ:



Hình 5 (a): mặt nạ quét được mô tả dưới dạng đường chấm chấm định vị một pixel đối tượng không được gắn nhãn với tọa độ (0,0) và pixel được gắn nhãn “1” như được hiển thị trong (b). Pixel không được gắn nhãn tiếp theo (2,0) được tìm thấy trong (c) và nó được gắn nhãn "2". Như được hiển thị trong (e), pixel không được gắn nhãn tiếp theo (3,0) được gắn nhãn "2" vì pixel lân cận của nó đã được gắn nhãn "2". Pixel không được gắn nhãn tiếp theo (1,1) nằm ở (g) và pixel được gắn nhãn "1" vì pixel lân cận theo đường chéo của nó (0,0) đã được gắn nhãn "1" như được hiển thị trong (h). Lưu ý rằng vùng được gắn nhãn được mở rộng thành hình vuông với bốn pixel chứa (0,0), (0,1), (1,0) và (1,1) bằng cách sử dụng quy tắc mở rộng được mô tả ở trên. Pixel không được gắn nhãn tiếp theo là (2,1) như được hiển thị tại (k) và nó được gắn nhãn bằng a vì pixel liền kề bên trái của nó đã được gắn nhãn "1". Ở đây, tất cả 8 pixel lân cận được kết nối của nó đang cập nhật nhãn của chúng với giá trị nhỏ nhất.

3. Line based connected component labelling algorithm

a. Giới thiệu

Thuật toán được giới thiệu năm 2010, bởi nhóm nghiên cứu tại Huazhong University of Science and Technology, Wuhan, Trung Quốc. Thuật toán dựa trên cấu trúc Union-find, tập trung vào vấn đề connected-line thay vì connected-pixel như các thuật toán khác đã đề cập. Có thể gói gọn thuật toán trong 3 cụm từ : trích xuất dòng , xác định các thành phần kết nối, gán nhãn. So sánh với các thuật toán khác thời điểm đó, thuật toán này có thời gian xử lý vượt trội, nhanh hơn từ 1.1 ~ 8 lần các thuật toán khác trong nhiều thử nghiệm khác nhau (dựa theo kết quả bài báo).

b. Thuật toán

Line Based Label Algorithm (LBLA)

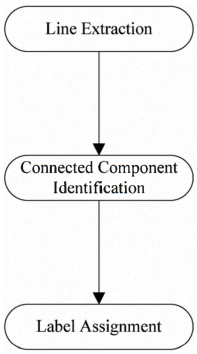
Để hiểu thuật toán, chúng ta có 3 định nghĩa được đưa ra như sau:

Định nghĩa 1: Dòng (line – L) được định nghĩa là một tập gồm 5 thành phần *L(LineID, x1, x2, y, LinkID)*. Với y là toạ độ dòng của ảnh, x1 là toạ độ của điểm bắt đầu, x2 là toạ độ của điểm kết thúc, LineID là chỉ số của dòng, LinkID là trường liên kết của điểm gốc tới các dòng trong cùng một thành phần liên kết.

Định nghĩa 2: Cho 2 dòng L1(LineIDL1, x1L1, x2L1, yL1, LinkIDL1) và L2(LineIDL2, x1L2, x2L2, yL2, LinkIDL2), nếu L2 thoả điều kiện dưới đây thì L2 được gọi là dòng kết nối của L1.

$$\begin{cases} |y_{L1} - y_{L2}| = 1 \\ !\left((x2_{L1} < (x1_{L2} - 1)) \,|\, (x1_{L1} > (x2_{L2} + 1))\right) \quad \text{is true} \end{cases}$$

Định nghĩa 3: Nếu LinkID của một dòng bằng với LineID của nỏ, thì được gọi là dòng vào.



Hình 6. Các bước của thuật toán

Thuật toán LBLA có 3 giai đoạn như hình trên.

- Giai đoạn đầu tiên được gọi là trích xuất dòng (Line Extraction), trích xuất các dòng từ các pixel của hình ảnh gốc.
- Giai đoạn thứ hai là xác định thành phần kết nối. Nếu hai dòng La, Lb có liên kết thì chúng là các dòng kết nối với nhau. Trong giai đoạn này, nếu các dòng cùng thuộc một thành phần kết nối thì nó sẽ được tổ chức như một cây với gốc dựa trên LinkID và mỗi dòng là một nút lá của cây gốc. LinkID được sử dụng để trỏ đến nút cha, nghĩa là LineID của dòng nút cha sẽ được gán cho LinkID của dòng hiện tại. Vì vậy, dòng cuối cùng có thể được tìm thấy bằng cách thực hiện tìm kiếm đệ quy trên LinkID của dòng. Điều kiện hội tụ của tìm kiếm theo dòng là LinkID của một dòng trong đường dẫn tìm kiếm bằng với LineID của chính nó.

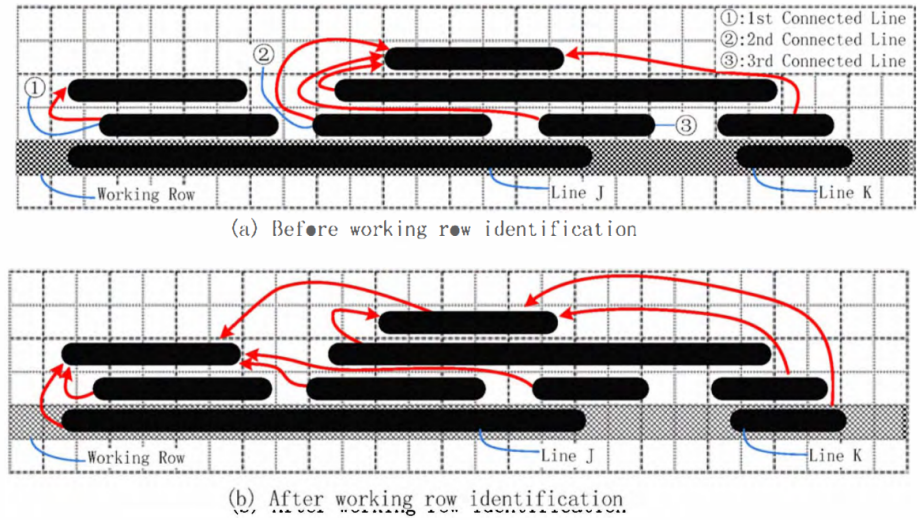
Theo **Định nghĩa 2**, mối quan hệ kết nối chỉ xảy ra giữa hai hàng lân cận. Vì vậy, quá trình nhận dạng kiểm tra hai hàng lân cận được thực hiện một lần một cách liên tục. Có ba tình huống đối với một dòng được kiểm tra với các dòng ở hàng lên:

1. Đã ngắt kết nối với tất cả các dòng ở hàng trên.
2. Được kết nối với chỉ một dòng ở hàng trên.
3. Được kết nối với nhiều hơn một dòng ở hàng trên.

Tình huống 1: Ngụ ý rằng một thành phần kết nối mới được phát hiện. Thì không nên làm gì cả.

Tình huống 2: Có nghĩa là một thành phần kết nối đã tồn tại nên được kéo dài. Dòng làm việc hiện tại và dòng được kết nối tương ứng ở hàng lên thuộc cùng một thành phần được kết nối. Dòng hiện tại nên được hợp nhất vào thành phần được kết nối này bằng cách sao chép LinkID của đường dây được kết nối, được minh họa bằng dòng K trong hình dưới.

Tình huống 3: Nó có nghĩa là một hoạt động hợp nhất sẽ được thực hiện. Quy trình của hoạt động hợp nhất như sau: Đầu tiên, tìm đường vào của thành phần được kết nối đầu tiên mà đường được kết nối đầu tiên thuộc về. Thứ hai, gán LinkID của dòng vào đó cho LinkID của dòng làm việc hiện tại (tức là, hợp nhất dòng làm việc hiện tại vào thành phần được kết nối gặp phải đầu tiên). Cuối cùng, tìm ra các đường vào tương ứng của tất cả các đường được kết nối khác và sửa đổi các LinkID của tất cả các dòng trong đường dẫn tìm kiếm thành LineID thu được đầu tiên. Tình huống này được minh họa bằng dòng J trong hình dưới.



- Giai đoạn thứ ba được gọi là gắn nhãn, gắn một nhãn duy nhất cho mỗi phần tử được kết nối và tất cả các thành phần trong cùng một nhóm sẽ được chia sẻ nhãn này. Số lượng các thành phần được kết nối trong hình ảnh được gắn nhãn có thể được thu thập bằng số dòng có LinkID bằng LineID. Tất cả các hình ảnh được gắn nhãn có thể được thu thập bằng các bước sau:
    1. Tìm nạp các dòng một cách tuần tự.
    2. Nếu đó là dòng vào, hãy kiểm tra xem nó đã được gán giá trị nhãn chưa. Nếu chưa, hãy gán một giá trị nhãn mới cho dòng này.
    3. Nếu đó không phải là dòng vào, hãy tìm dòng vào tương ứng bằng LinkID. Sau đó thực hiện quy trình 2 một cách đệ quy và gán giá trị nhãn của dòng lối vào tương ứng cho dòng hiện tại.

Sau ba giai đoạn nêu trên, tất cả các dòng trong cùng một thành phần được kết nối phải được gán cùng một giá trị nhãn duy nhất.

III. Implement

Trong các thuật toán đã trình bày, nhóm chúng tôi chọn thuật toán thứ nhất - The classical algorithm để thực hiện implement.

1. Các bước thực hiện.

- Lấy ma trận từ file text.

```
#-------------Using a 9x17 matrix to demo
test_array = loadtxt('test.txt')
im =  test_array.astype(int)
w = 17
h = 9
#----------------------------
```

a. Lần duyệt thứ nhất

- Duyệt từng phần tử (pixel), các phần tử có giá trị là 1 thì xét các phần tử láng giềng và nhãn của các phần tử láng giềng của nó. Trong implement sử dụng khai báo láng giềng - 8.

```
if im[row,col] == 1:
            # Neighbours-4
            #neighbours = [im[row-1,col], im[row+1,col], im[row,col-1], im[row,col+1]]
            #neighbours_labelled = [im_labelled[row-1,col], im_labelled[row+1,col], im_labelled[row,col-1], im_labelled[row,col+1]]
            #neighbours-8
            neighbours = [im[row-1,col], im[row+1,col], im[row,col-1], im[row,col+1],im[row-1,col-1], im[row+1,col+1], im[row+1,col-1]
            neighbours_labelled = [im_labelled[row-1,col], im_labelled[row+1,col], im_labelled[row,col-1], im_labelled[row,col+1],im_l
```

- Xét các trường hợp : phần tử không có láng giềng nào khác 0; phần tử chỉ có một láng giềng khác 0.

```
#If all neighbours are not zero
            if neighbours.count(0) == 8:
                tag = tag + 1
                im_labelled[row,col] = tag
            #If there is only 1 neighbour is non-Zero
            elif neighbours.count(0) == 7:
                index =  find_indices(neighbours, lambda e: e != 0 ) #get index of non-zero n
                #Two pixel are connect, so label curren pixel as its neighbour's label
                if neighbours_labelled[index[0]] == 0:
                    tag = tag + 1
                    im_labelled[row,col] = tag
                else:
                    im_labelled[row,col] = neighbours_labelled[index[0]]
```

- Trường hợp còn lại là phần tử có nhiều hơn 1 láng giềng khác 0. Khi đó ta cần thực hiện các bước đã nêu ở phần thuật toán, đồng thời lưu lại các lớp tương đương trong một ‘dictionary’ hỗ trợ bởi python.

```
else:
                #There are 2 situations: all neighbours have the same label ; neighbours have different labels
                #Get non-zero index
                indices = find_indices(neighbours, lambda e: e != 0 )
                list(map(int, indices))
```

```
                # Get all label of current pixel's neighbours
                list_neighbours = []
                for i in indices:
                    list_neighbours.append(neighbours_labelled[i])
                same_label = True
                #Check if neightbours have labels.
                if  not any(list_neighbours):
                    tag = tag+1
                    im_labelled[row,col] = tag
                else:
                    l = min(list(filter(lambda a: a != 0, list_neighbours)))
                    #Check if all neighbours have same labels
                    for label in list_neighbours:
                        if l != label:
                            same_label = False
                    #Situation 1: Same label, label current pixel as same as its neighbour
                    if same_label:
                        im_labelled[row,col] = l
                    #Situation 2: different labels, label current pixel as same as its neighbour.
                    else:
                        im_labelled[row,col] = l
                    # record equivalence classes
                        for i in indices:
                            temp1 = neighbours_labelled[i]
                            s = str(l)
                            if temp1 != l:
                                if s in dictionary:
                                # Get unique value with the key 's'
                                    dictionary[s] += str(temp1)
                                else:
                                    dictionary[s] = str(temp1)
                            for key in dictionary:
                                dictionary[key] = list(filter(lambda a: a != '0', dictionary[key]))
                                dictionary[key] = list(set(dictionary[key]))
```

b. Cấu trúc lại ‘dictionary’

Trong quá trình lưu trữ các lớp vào dictionary sẽ xuất hiện hiện tượng các cặp key-value bị trùng.

Ví dụ, đây là kết quả lưu trữ label vào dictionary khi chưa cấu trúc lại:

```
{'1': ['2'], '2': [], '3': ['4', '5', '7'], '4': [], '6': [], '5': ['6']}
```

Thấy được ‘3’ là value của key ‘2’, ‘4’ là value của key ‘3’, có thể nói ‘2’, ‘3’, ‘4’ là các label tương đương. Việc cần làm là đưa ‘4’ vào value của ‘2’ và xóa value của key ‘3’. Tương tự cho trường hợp nhiều key trùng lập.

```
dictionary_restructed = dictionary
for key in dictionary:
    for key2 in dictionary:
        if key in dictionary.get(key2):
             dictionary_restructed[key2] += dictionary[key]
             dictionary_restructed[key] = 'null'
for key in dictionary_restructed:
    dictionary[key] = list(filter(lambda a: a != 'null', dictionary[key]))
```

c. Lần duyệt thứ hai

Duyệt các phần tử lại từ đầu thêm một lần nữa, lúc này tất cả phần tử khác 0 đã được gán nhãn. Mục tiêu của lần duyệt này là gán tất cả nhãn tương đương về cùng một nhãn đại diện (key).

```
for row in range(0,h):
    for col in range(0, w):
        p = im_labelled[row,col]
        p_str = str(p)
        for key in dictionary:
            if p_str in dictionary.get(key):
            #Relabel all pixels have equivalence classes
                p_str = key
        q = int(p_str)
        im_labelled[row,col] = q
#Export file matrix
with open('labelled_array.txt', 'w') as f:
     csv.writer(f, delimiter=' ').writerows(im_labelled)
```

2. Kết quả thực nghiệm.

a. Input từ file text.txt.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0
0 0 0 1 1 1 1 0 0 0 1 1 1 1 0 0 0
0 0 1 1 1 1 0 0 0 1 1 1 0 0 1 1 0
0 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

b. Kết quả lần duyệt đầu

Output:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 2 2 0 0 3 3 0 0 4 4 0
0 1 1 1 1 1 1 1 1 0 0 3 3 3 3 0 0
0 0 0 1 1 1 1 0 0 0 3 3 3 3 0 0 0
0 0 1 1 1 1 0 0 0 3 3 3 0 0 3 3 0
0 1 1 1 0 0 1 1 0 0 0 3 3 3 0 0 0
0 0 1 1 0 0 0 0 0 5 3 0 0 0 3 3 0
0 0 0 0 0 0 6 6 5 3 0 0 7 3 3 3 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Các lớp tương đương:

```
{'1': ['2'], '2': [], '3': ['4', '5', '7'], '4': [], '6': [], '5': ['6']}
```

c. Cấu trúc lại ‘dictionary’

Các lớp tương đương sau khi cấu trúc lại:

```
{'1': ['2'], '2': [], '3': ['4', '5', '7', '6'], '4': [], '6': [], '5': []}
```

d. Lần duyệt thứ hai và kết quả sau cùng

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 1 0 0 3 3 0 0 3 3 0
0 1 1 1 1 1 1 1 1 0 0 3 3 3 3 0 0
0 0 0 1 1 1 1 0 0 0 3 3 3 3 0 0 0
0 0 1 1 1 1 0 0 0 3 3 3 0 0 3 3 0
0 1 1 1 0 0 1 1 0 0 0 3 3 3 0 0 0
0 0 1 1 0 0 0 0 0 3 3 0 0 0 3 3 0
0 0 0 0 0 0 3 3 3 3 0 0 3 3 3 3 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

IV. Reference.

https://algorist.com/problems/Connected_Components.html

https://www.codeproject.com/Articles/336915/Connected-Component-Labeling-Algorithm

https://towardsdatascience.com/implementing-a-connected-component-labeling-algorithm-from-scratch-94e1636554f

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6884436&tag=1

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5563571