Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0217 – Estructuras de Datos Abstractas y Algoritmos para Ingeniería
II ciclo 2020

Proyecto de Investigación

Connected Components Labeling

Sebastián Palacino Chacón B65242

Profesor: Juan Carlos Coto Ulate

Domingo 13 de Diciembre

Índice

Índice de figuras

1. Introducción

1.1. Resumen

El presente documente tiene como objetivo explicar el algoritmo de estructura de datos conocido como Connected Components Laveling. Para lograr esto se empezará explicando el contexto de este, así como algunos conceptos claves para lograr comprender a fondo este algoritmo. Una vez hecho esto, se verán ejemplos donde se puede usar este algoritmo. Teniendo en cuenta estos ejemplos, se procederá a entrar en detalle sobre el funcionamiento del algoritmo. Se hablará sobre la complejidad de este. Finalmente se mostrará un ejemplo de código en Python donde se aplica el algoritmo Connected Components Labeling.

1.2. Contexto

El algoritmo de Connected Components Labeling es un algoritmo que utiliza la teoría de grafos. A este algorimto se le conoce por algunos sinónimos como Connected Components Analysis, blob extraction, region labeling, entre otros. Este proceso ha sido muy relevant en el área de análisis de imagenes y de visión por computadora. El resultado de este algoritmo permite al humano interpretar distintas imagenes, esto se verá más en detalle cuando se hable de su utilidad. Al ser un algoritmo muy utilizado, la necesidad de optimizarlo está en el interes de las personas en las áreas pertinentes a los usos de este. Estas optimizaciones permiten aumentar la velocidad y la eficacia de este [1].

2. Conceptos Claves

Para lograr entender de la mejor forma el algoritmo de Connected Elements se debe entender primero algunos conceptos básicos.

2.1. Grafos

Los grafos son una estructrura de datos no lineales. Estos son un conjunto de nodos que están asociados entre sí mediante arcos. Estos arcos pueden o no tener peso y además puede o no que tengan dirección. Un grafo con dirección se le llama digrafo y uno con peso se le denomina pesado[2].

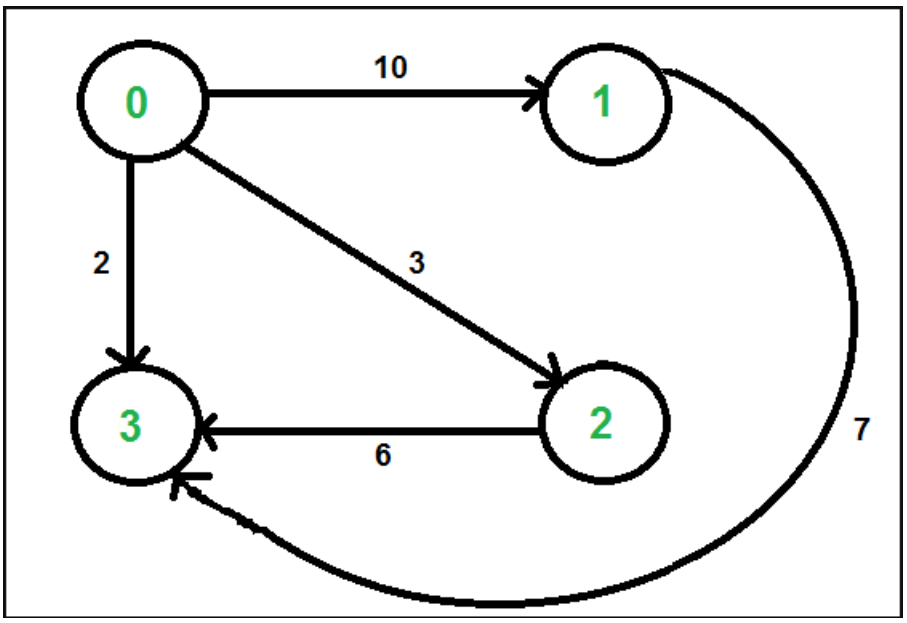

Figura 1: Grafo con dirección y peso

2.2. Algoritmos

Los algoritmos son una lista de instrucciones que se utiliza para resolver un determinado problema. Estos pueden ser representados en pseudocódigo o diagramas de flujo. Una forma muy práctica de entender que es un algoritmo es verlo como si fuera una receta de cocina [3].

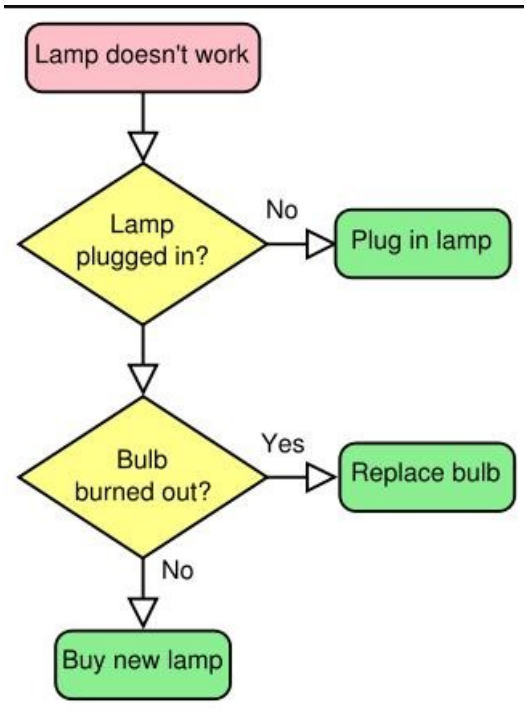

Figura 2: Ejemplo de un algoritmo

2.3. Conectividad de Pixeles

La conectividad de pixeles trata sobre la relación entre dos o más pixeles. Para lograr estar conectados se tienen que cumplir ciertas condiciones. Por ejemplo, tienen que compartir el

mismo valor de V. En una imagen en escala de grises V puede asumir cualquier tipo de valor, pero en el caso de imagenes binarias, solo puede asumir el valor de 1.
Los pixeles pueden tener vecindad entre ellos. Las vecindades que se hablaran son las de vecindad de 4 y de 8. Estas se verán representadas por las notaciones $N_4(p)$ y $N_8(p)$. Las formulas para representar vecindades entre pixeles con coordenadas x y y son las siguientes:

$$N_4(p) = (x+1, y), (x-1, y), (x, y+1), (x, y-1) \tag{1}$$

$$N_8(p) = N_4 \cup (x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1) \tag{2}$$

Otra forma de poder mostrar esta conectividad es la imagen 3.

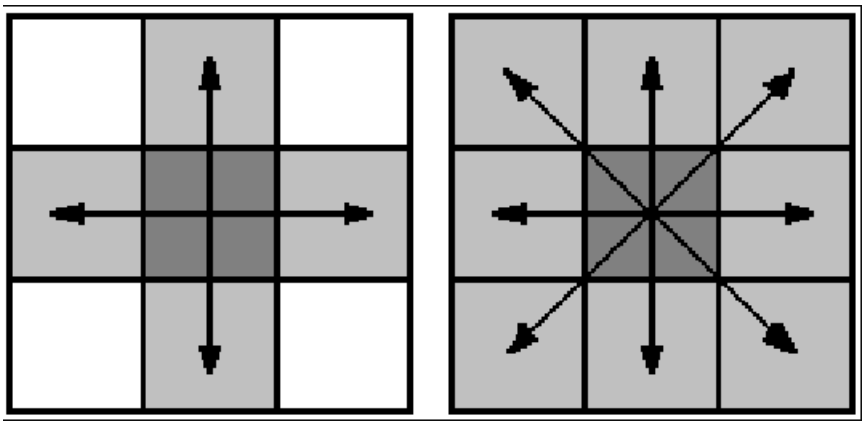

Figura 3: Conectividad de Pixeles. Izquierda: Conectividad de 4, Derecha: Conectividad de 8

3. Discusión

3.1. Funcionamiento del CCL

Como se vio anteriormente, existen dos posibles vías para este algoritmo. La primera era la de *One component at a time* y la segunda era *Two Pass* o también conocido como *Hoshen-Kopelman Algorithm*. El método que se va a explicar es el segundo. Se debe notar que aunque se explique el segundo, en la actualidad se utiliza el primero puesto que evita el problema de corrupción de datos[1].

3.1.1. Pasos del Algoritmo

CCL funciona más rápido cuando se tienen imagenes binarias o en la escala de grises. Asumiendo una conectividad de 8, se escanea fila por fila, y de arriba hacia abajo, los pixeles hasta llegar a algún punto p. Este punto significa un pixel donde se tiene un valor de intensidad V (V puede variar en valores dependiendo si es una imagen binaria o una en escala de grises, pero para este ejemplo se tiene una imagen binaria, entonces $V = 1$). Cuando se esta en p, se procede a ver su vecindad. Se revisa arriba, abajo, a la izquierda y a la derecha. Entonces dependiendo de los valores obtenidos en los vecinos de p se tiene que:

- *If* sus 4 vecinos son 0, asigna una nueva etiqueta a p *else*
- *If* uno de sus vecinos tiene $V = 1$, assigna la misma etiqueta a p *else*
- *If* más de uno de sus vecinos tiene $V = 1$, asigna una de las etiquetas de p y procede a hacer una lista de equivalencias entre las etiquetas

Una vez hecho esto, las etiquetas que comparten equivalencia son asignadas a una misma clase y se les asigna un valor único entre ellas. Se realiza una segunda escaneada con el fin de cambiar las etiquetas a un mismo valor dependiendo de sus clases [1].

A manera de lograr una mejor comprensión del tema, se presentan las figuras 4 y 5. En la figura 4 se puede observar que se realizo el *first pass*. En este paso se procede a asegurar que los pixeles de frente o los de relevancia lleven una etiqueta de 1. De esta forma nos aseguramos de que son materia relevante y posteriormente nos aseguraremos de proceder a distinguirlos para lograr ver su forma. En la figura 5 se tiene el *second pass*. En este paso procedemos a distinguir los objetos presentes en la imagen utilizando las etiquetas descritas en los pasos anteriores. Se puede notar que existen conflicto en las etiquetas, y es por esto que se utiliza una tabla de equivalencia de las etiquetas para lograr solucionar este problema. Esta tabla se logra apreciar en la tabla 1. Otro uso de esta tabla es que permite asignar la etiqueta de menor valor a todas aquellas a la que sea equivalente, y de esta forma podemos mostrar los objetos de la imagen. El resultado final se puede ver en la figura 6.

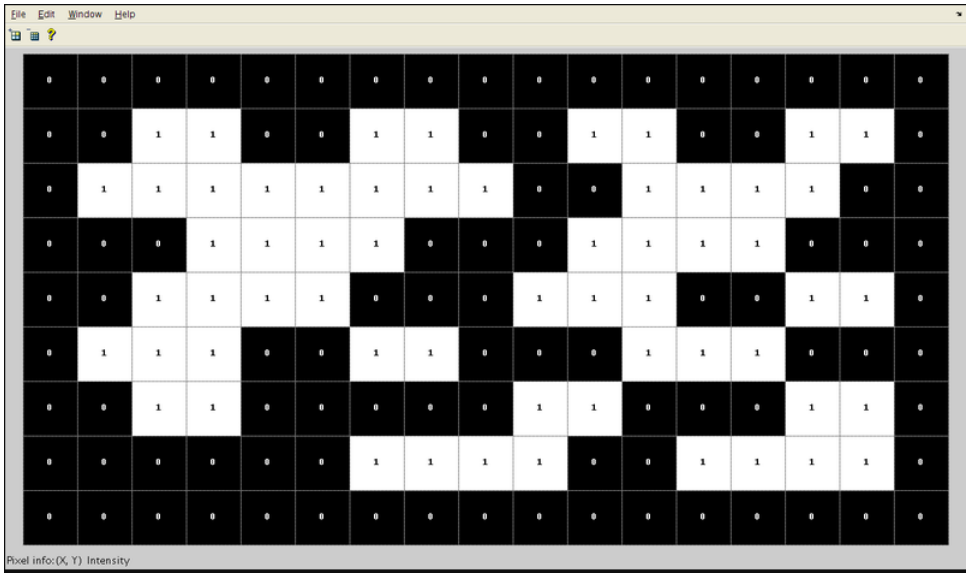

Figura 4: First pass

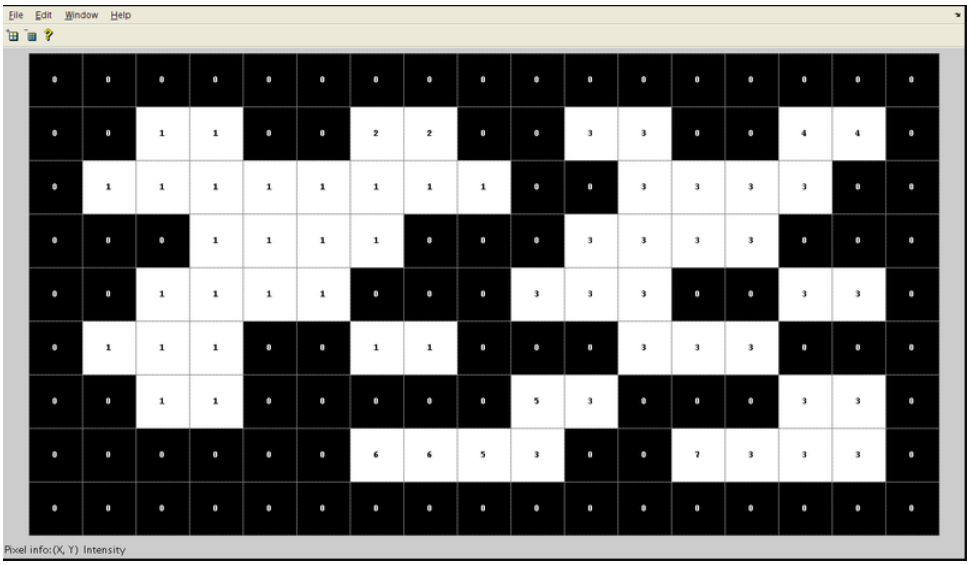

Figura 5: Second pass

Tabla 1: Tabla de Equivalencia de Etiquetas

Etiqueta	Equivalencia
1	1, 2
2	1, 2
3	3, 4, 5, 6, 7
4	3, 4, 5, 6, 7
5	3, 4, 5, 6, 7
6	3, 4, 5, 6, 7
7	3, 4, 5, 6, 7

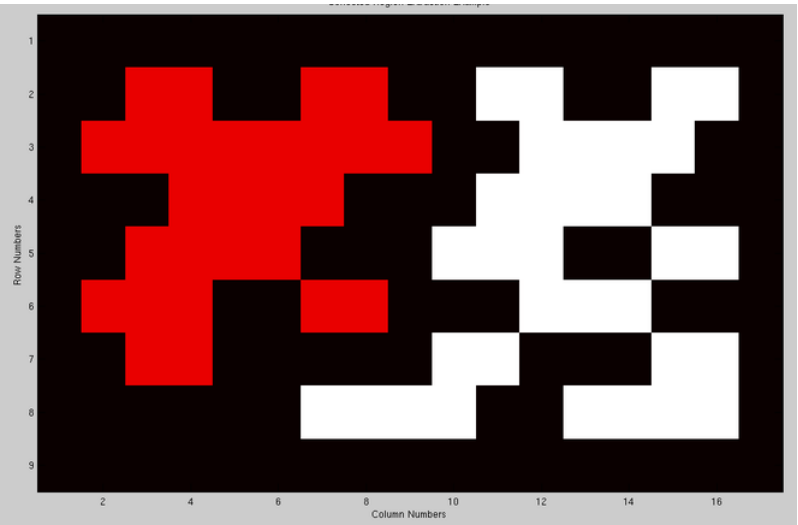

Figura 6: Resultado del CCL

3.2. Utilidad de CCL

El algoritmo de Connected Components Labeling ha sido de suma importancia para el área de procesamiento y análisis de imágenes. Al permitir mostrar donde hay objetos, o donde hay pixeles conectados, se logra determinar la existencia de una imagen que dependiendo de su fin, puede o no tener significado. Esto se puede apreciar en la figura 7. Por ejemplo, CCL ha sido de suma importancia para el área de la radiología puesto que permite a las máquinas mostrar donde hay órganos o objetos de importancia en las imágenes tomadas. Un ejemplo de esto se puede ver en la figura 8. En el área de robótica, ha permitido a los robots determinar donde existe un objeto basado en la conectividad de pixeles que la cámara de este detecta. Para los astrónomos también ha sido de gran uso puesto que les permite interpretar donde hay, o de que tamaño es, alguna sección que se desea estudiar de un planeta [1]. Esto se puede notar en la figura 9.

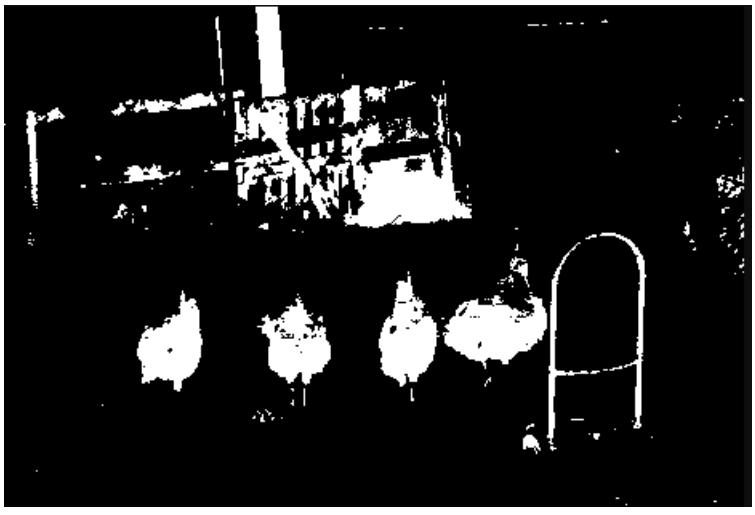

Figura 7: Uso de CCL para determinar cuantos pavos existen en la imagen

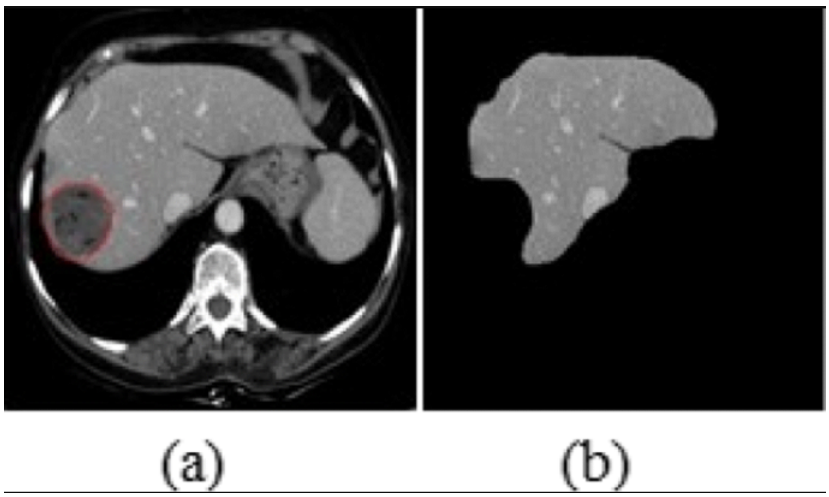

Figura 8: Uso de CCL para determinar un segmento del hígado

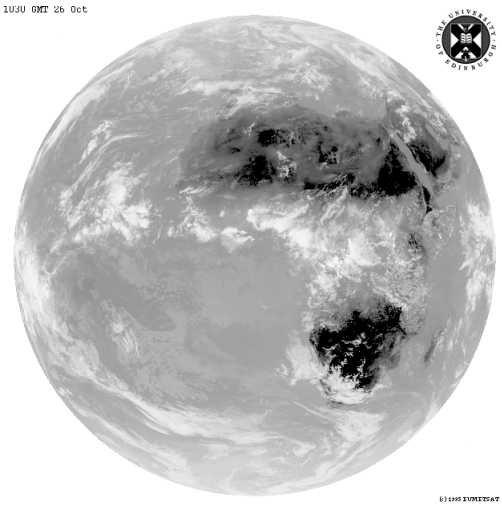

Figura 9: Uso de CCL para determinar una sección de la Tierra

3.3. Complejidad

Para analizar la complejidad del algoritmo se ira paso a paso según la etapa del algoritmo, además se usará la notación O(n). Se empieza asumiendo que se tiene N pixeles, entonces por ahora se tiene O(N).
En el *first pass* se analiza los vecinos dependiendo de la conectividad de pixeles. Dependiendo si se tiene que hacer operaciones *Union-Find*, se tendría un costo de O(1). Por lo tanto, tomando en cuenta la inicialización y el *first pass*, el costo se mantendría como O(N).
Al llegar al *second pass* se tiene que analizar las etiquetas existentes y ver si hay equivalencia entre ellas. Se utiliza la operación *Find()*. Con esto se tiene que su costo sería de O(1). Debido a que esto se implementa como una tabla hash en Python, su costo sigue siendo de O(1). Es por esto que en el *second pass* se sigue teniendo un costo de O(N).
En el código utilizado sucede un *third pass* que revisa las etiquetas y las cambia por sus equivalentes. Esto tiene un costo de O(N).
Analizando el resultado total se puede ver que en términos de complejidad de tiempo, se tiene una de O(N) [4].

3.4. Implementación del Algoritmo

Para la implementación del código se utilizó uno distinto al que se uso como referencia en la sección de Complejidad, sin embargo, el código presenta un comportamiento similar al descrito en la sección, por ende se asume una misma complejidad. Ahora bien, el código utilizado se puede observar en la figura 10.

```
1 import cv2
2 import numpy as np
3 import time
4
5 img = cv2.imread('shapes.jpg', 0)
6
7 #Nos aseguramos de que sea una imagen binaria
8 img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]
9 retval, labels = cv2.connectedComponents(img)
10
11 ################################################
12 ts = time.time()
13 num = labels.max()
14
15 #Asignamos Etiquetas
16 N = 50
17 for i in range(1, num+1):
18     pts =  np.where(labels == i)
19     if len(pts[0]) < N:
20         labels[pts] = 0
21 #Se imprime el tiempo transcurrido
22 print("Time passed: {:.3f} ms".format(1000*(time.time()-ts)))
23
24
25 ################################################
26 #Mapeamos las etiquetas de los componentes a los valores de hue
27
28 label_hue = np.uint8(179*labels/np.max(labels))
29 blank_ch = 255*np.ones_like(label_hue)
30 labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
31 #Pasamos de cvt a BGR para el display de la imagen
32
33 labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
34 #Seteamos la etiqueta bg a que sea negra
35
36 labeled_img[label_hue==0] = 0
37
38 cv2.imshow('labeled.png', labeled_img)
39 cv2.imwrite("labeled.png", labeled_img)
40 cv2.waitKey()
41
```

Figura 10: Código del algoritmo

Para este código se utilizan las librerías *OpenCV, numpy, y time* de Python. En la librería *cv2* obtenemos la función de Connected Components. Esta función nos permite computar los Connected Components de una imagen binaria [5]. Se utiliza la librería *time* para lograr calcular el tiempo que tarda el algoritmo en operar. Por último, la librería *numpy* es utilizada debido a su uso en matrices y vectores.

Al ingresar las imagenes binarias en la línea 5, se obtienen los resultados vistos en las figuras 11, 12, 13, respectivamente.

Figura 11: CCL para una imagen binaria de una Cara

Figura 12: CCL para una imagen binaria de varias figuras

Figura 13: CCL para una imagen binaria de varias cruces

4. Conclusión

Al haber investigado el algoritmo de Connected Components Labeling se logró llegar a distintas conclusiones. Estas fueron las siguientes:

- Se mostró la utilidad que tiene CCL en el área de reconocimiento de imagenes, ya sea en imageneología o en visón de computadres. Este algoritmo ha mostrado gran importancia y es por esta razón que sigue utilizando y además se continuan buscando formas de optimizarlo.
- Se observó que CCL muestra mayor facilidad a la hora de utilizar imagenes en las escalas de grises que en otras que tengan varios colores.
- Se probó el funcionamiento del algoritmo en un código en Python donde el estudiante ingreso distitnas imagenes binarias y obtuvo los Connected Components de ellas con distintos colores.

Referencias

[1] Anon. Image analysis - connected components labeling. https://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm.

[2] Anon. Graph data structure and algorithms, 2016. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/.

[3] Anon. What is an algorithm? https://www.bbc.co.uk/bitesize/guides/z22wwmn/revision/1#:~:text=An%20algorithm%20is%20a%20set,Programming.

[4] Anon. Connected components labeling. https://jacklj.github.io/ccl/.

[5] Anon. Opencv: Structural analysis and shape descriptions. https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#gaedef8c7340499ca391d459122e51bef5.