LNMIIT
The LNM Institute of Information Technology

Digital Image Processing

(CSE2012)

Assignment - 2

Submit to: **Dr Anukriti Bansal**

Submitted By

VAIBHAV KHAMESRA - 21UCS224

SHREYASH ACHARYA - 21UCS200

DIVYANSH GARG - 21DCS003

Acknowledgment

It is a matter of great satisfaction and pleasure to present this project on **"Extracting a required element from an Image using Morphological image processing technique"** on the very outset of this report, we would like to extend our sincere & heartfelt obligation towards all the personages who have helped us in this endeavour. Without their active guidance, help, cooperation & encouragement, we would not have made headway in the project.

We take this opportunity to owe our thanks to our faculty member, Dr. Anukriti Bansal, for their encouragement and guidance at every stage of this project.

We express our gratitude to all those who have directly or indirectly helped us to make this project.

Collaborations

We are thankful to:

1. Mukul Verma

For helping us in this assignment

Problem Statement:

Given the attached image, write a program to implement a scheme that will extract only the elephant from the image. The output image will only have the elephant (in colour) and a white background.

Step by Step Procedure:

1. Importing the Image in colour and converting it to HSV using OpenCV ’s built in function.

2. After doing the same we find out the *Hue*, *Saturation* and *Intensity* range of the required object i.e., the elephant in this case

The Ranges were:

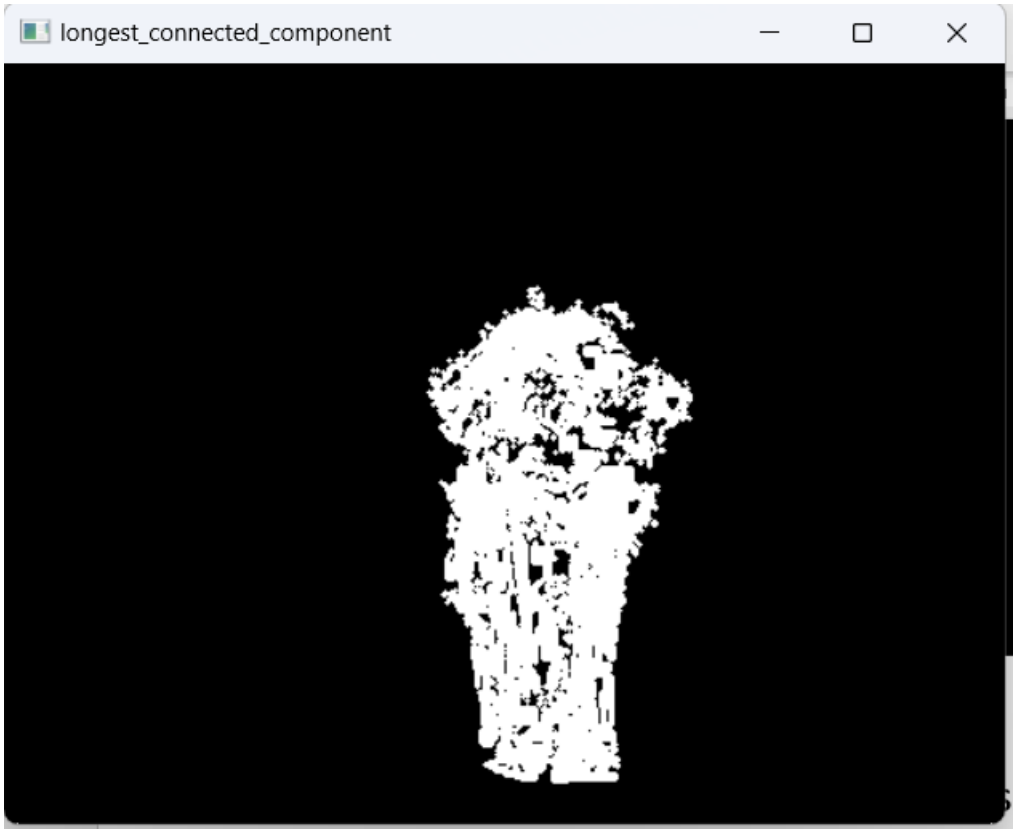
Hue: 0 to 31

Saturation: 26 to 100

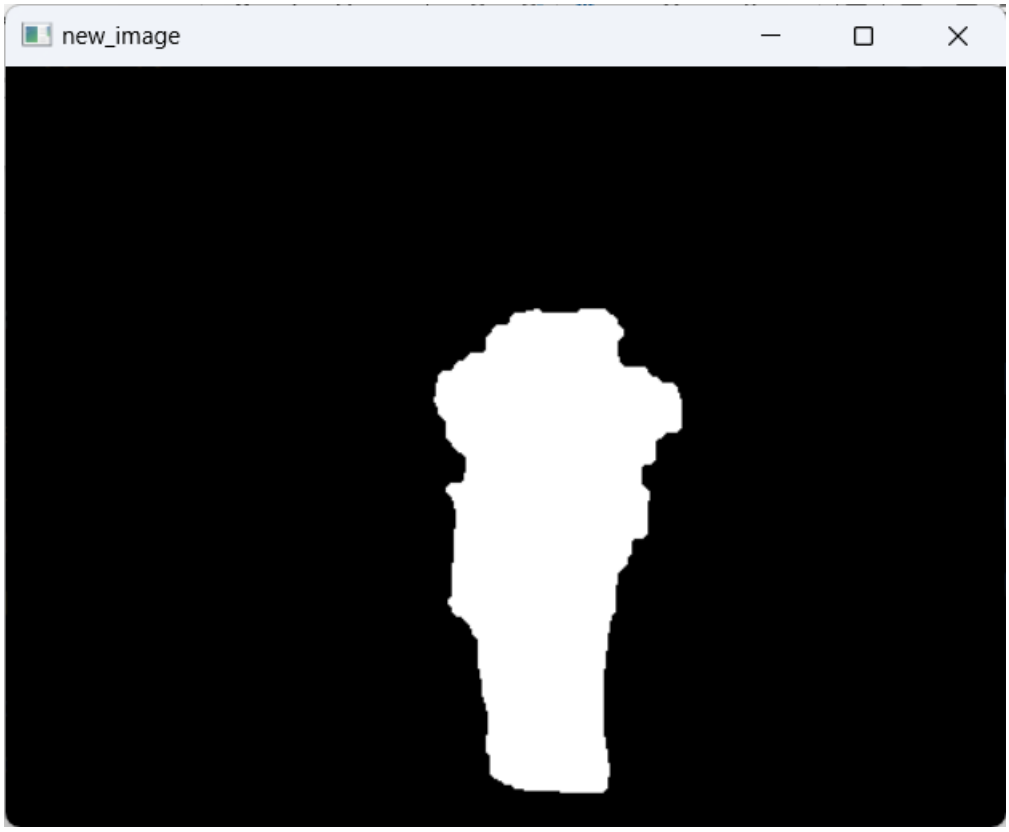
Intensity: 0 to 200

3. Now taking a blank Canvas (black) we changed all the pixels to *white* in the canvas whose HSV values were in the above range in the original image and *applied dilation* on the image.

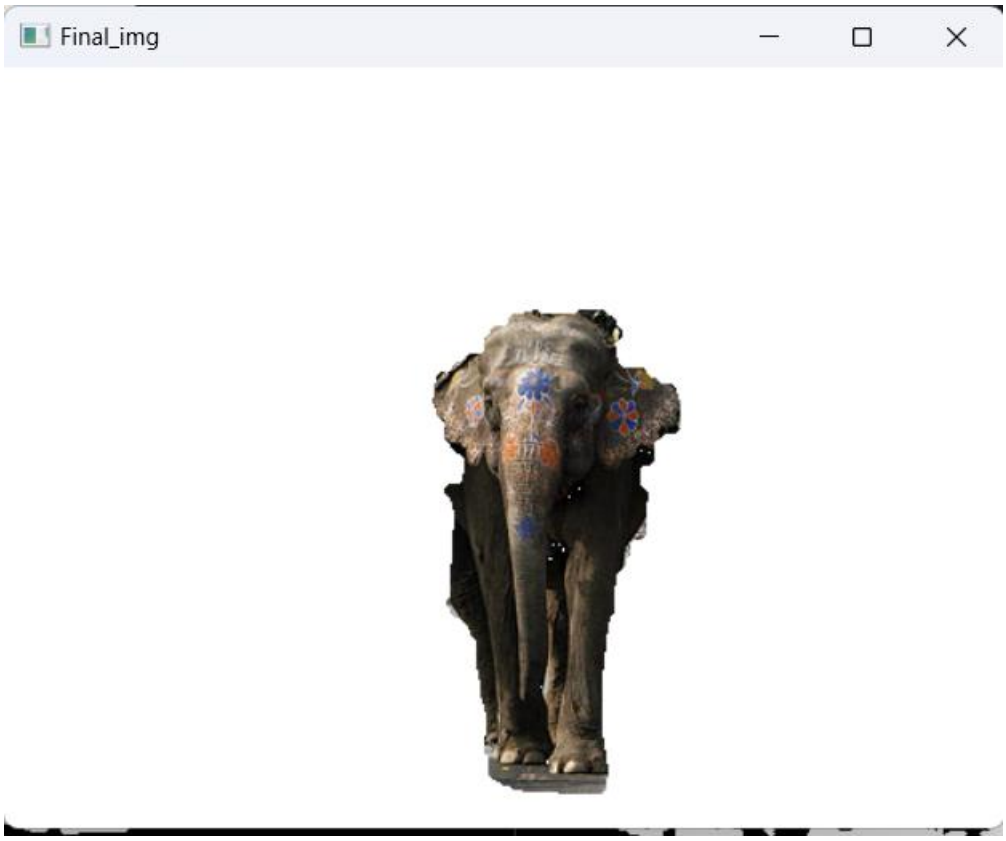
4. After this we wrote an algorithm to find out the longest connected component in the image using breadth-first-search to remove all the noise and hence the image looked like:



5. Now we needed to fill all the holes in the mask and make it smooth so for the same we used a combination of Dilation and Erosion that we found out using trial and error with different kernels and centres and our resulting mask came out to be:



6. Now we applied this Mask to get our element which came out as follows:

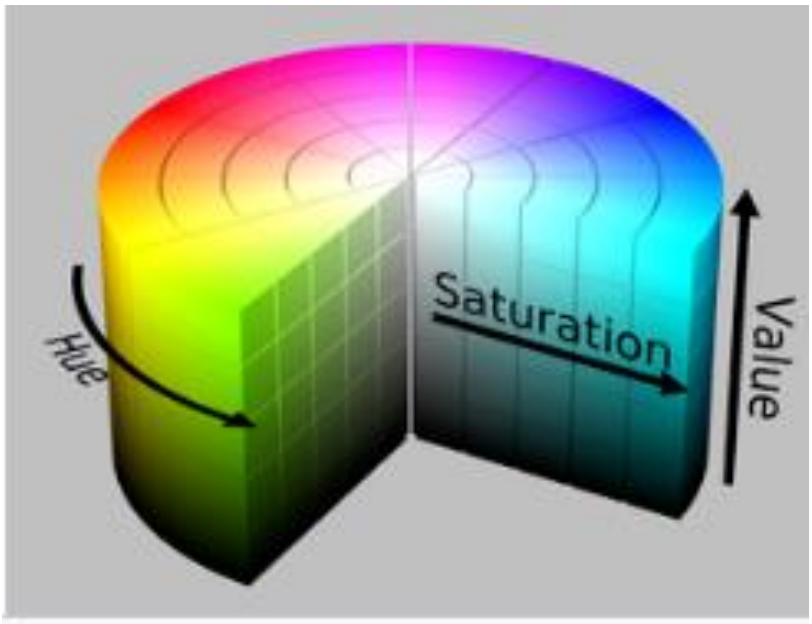


Concepts Used:

HSV Image:

HSV is cylindrical geometries with hue, their angular dimension, starting at the *Red at 0°*, passing through the *Green at 120°* and the *Blue at 240°*, and then wrapping back to *red at 360°*. In each geometry, the central vertical axis comprises the *neutral*, *achromatic*, or *Gray* colours ranging, from top to bottom, white at lightness 1 (value 1) to black at lightness 0 (value 0).

In both geometries, the primary and colours — Red, Yellow, Green, Cyan, Blue and Magenta —and linear mixtures between adjacent pairs of them, sometimes called *pure colours*, are arranged around the outside edge of the cylinder with saturation 1

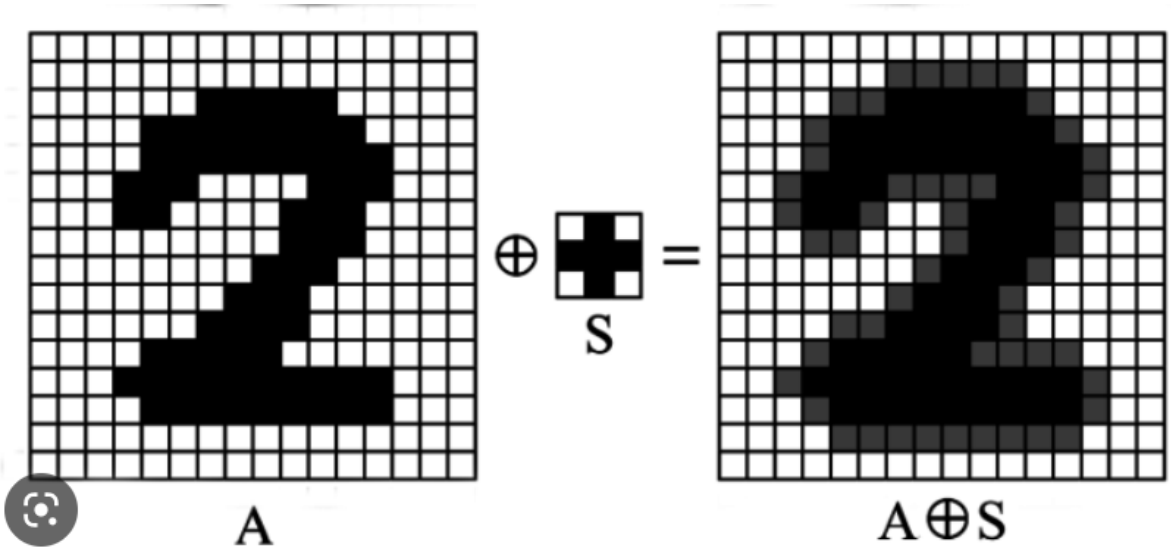


Dilation:

Dilation is a morphological operation in image processing that expands the boundaries of an object in an image. It involves convolving the image with a structuring element, which is a small, pre-defined shape such as a square or a circle. The structuring element is moved across the image, and for each pixel in the image, the maximum value of the overlapping pixels between the structuring element and the image is taken as the new value of that pixel.

Dilation can be used to achieve various effects on an image, such as filling small gaps, smoothing rough edges, or thickening object boundaries.

Example of Dilation:

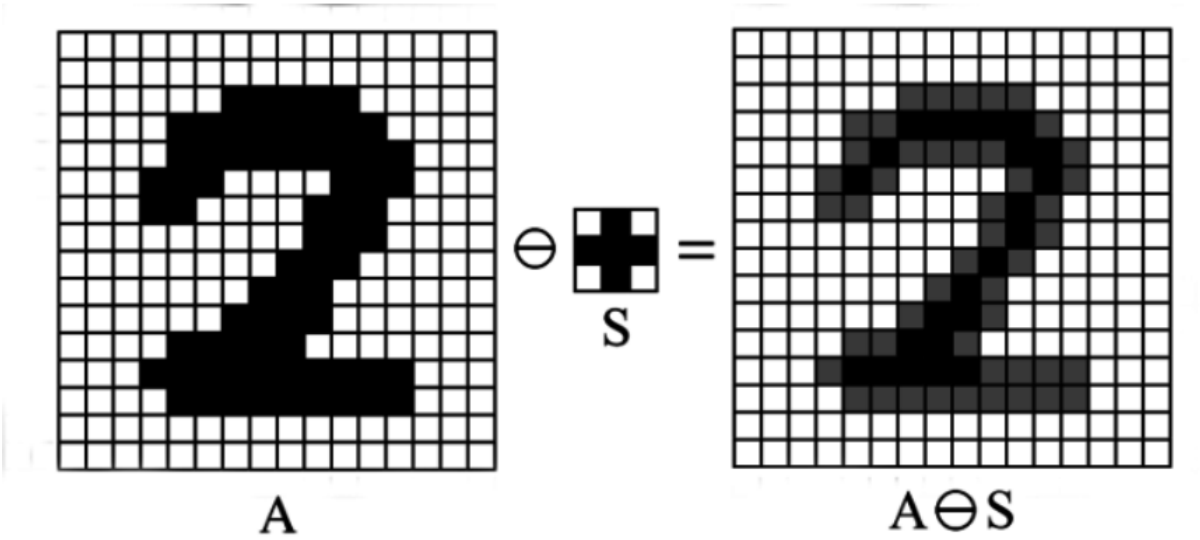


S is a the structuring Element

Erosion

Erosion is a morphological operation in image processing that shrinks or erodes the boundaries of an object in an image. It involves convolving the image with a structuring element, which is a small, pre-defined shape such as a square or a circle. The structuring element is moved across the image, and for each pixel in the image, the minimum value of the overlapping pixels between the structuring element and the image is taken as the new value of that pixel.

Erosion can be used to remove small protrusions or spikes, break thin connections between objects, or separate objects that are too close to each other.

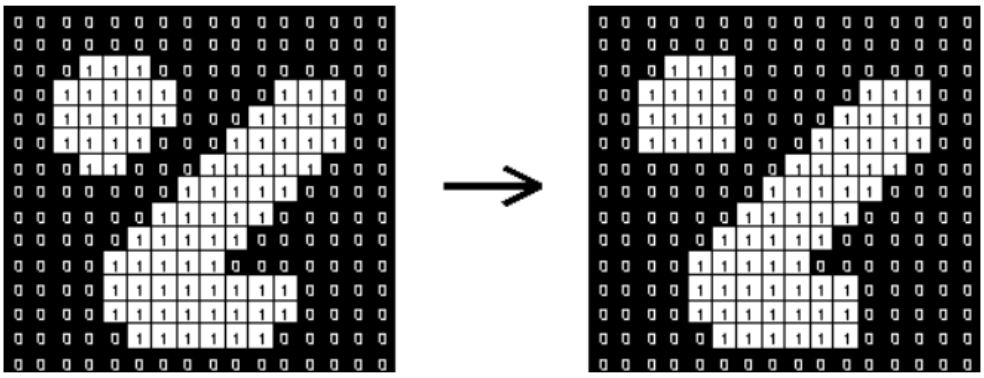


Where S is the structuring element

Opening

Opening is a morphological operation in image processing that is a combination of erosion followed by dilation. It involves eroding the image with a structuring element, which is a small, pre-defined shape such as a square or a circle. After the erosion operation, the resulting image is then convolved with the same structuring element using dilation.

The opening operation can be used to remove small objects or thin structures from an image, while preserving the larger objects or structures. It can also be used to smooth out the boundaries of objects and eliminate small gaps or holes within them.

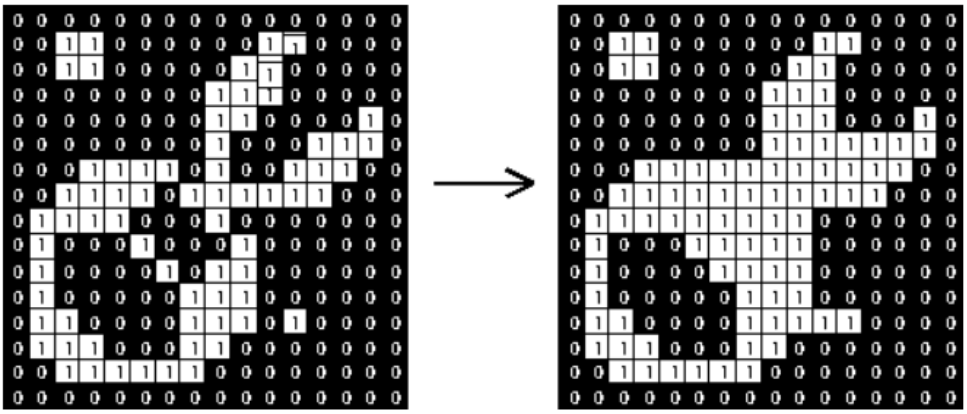


Closing

Closing is a morphological operation in image processing that is a combination of dilation followed by erosion. It involves dilating the image with a structuring element, which is a small, pre-defined shape such as a square or a circle.

After the dilation operation, the resulting image is then convolved with the same structuring element using erosion.

The closing operation can be used to fill small gaps or holes within objects, while preserving the overall shape and size of the objects. It can also be used to eliminate small protrusions or spikes from objects and smooth out their boundaries. Closing is often used in conjunction with opening to achieve more complex image processing tasks, such as noise reduction or segmentation.



Functions used and created:

1. convert_to_bw (img)
2. add_padding (img)
3. dilation (temp_img,kernel,center,itr = 1)
4. erosion (temp_img,kernel,center,itr = 1)
5. opening (img,kernel,center,itr = 1)
6. closing (img,kernel,center,itr = 1)
7. find_longest_component(image)

1. convert_to_bw(img):

Takes an grayscale image as an argument and converts it into a binary image and returns it (Thresholding)

```
def convert_to_bw(new_img):
    t = np.zeros(new_img.shape,np.uint8)
    for i in range(height):
        for j in range(breadth):
            if(new_img[i][j] <= 127):
                t[i][j] = 0
            else:
                t[i][j] = 255
    return t
```

2. add_padding(img):

Adds padding to a passed image using the nearest pixel value and returns the padded image

```
def add_padding(img):
    height,breadth = img.shape
    new_img = np.zeros((height + 2,breadth + 2))
    h,b = new_img.shape
    for i in range(height):
        for j in range(breadth):
            new_img[i+1][j+1] = img[i][j]
    #print(new_img.shape)
    new_img[0][0] = img[0][0]
    new_img[h-1][b-1] = img[height-1][breadth-1]
    new_img[0][b-1] = img[0][breadth-1]
    new_img[h-1][0] = img[height-1][0]
    for i in range(1,h-1):
        new_img[i][0] = img[i-1][0]
        new_img[i][h-1] = img[i-1][height-1]
    for j in range(1,b-1):
        new_img[0][j] = img[0][j-1]
        new_img[0][b-1] = img[0][breadth-1]
    return new_img
```

3. dilation (temp_img,kernel,center,itr = 1):

Dilates the passed image along with the specified kernel and the centre of the kernel no of iterations to be done can also be passed and returns the dilated image

```
def dilation(temp_img,kernel,center,itr = 1):
    height,breadth = temp_img.shape
    dilated_image = np.zeros((height,breadth))
    #print(eroded_image)
    k_height,k_breadth = kernel.shape
    for k in range(itr):
        for i in range(1,height-2):
            for j in range(1,breadth-2):
                flag = False
                for k in range(k_height):
                    for l in range(k_breadth):
                        if(temp_img[i+k][j+l] ==
255 and kernel[k][l] == 255):
                            flag = True;
                            dilated_image[i+center
[0]][j+center[1]] = 255;
                            break
                    if(flag):
                        break
        temp_img = np.copy(dilated_image)

    return dilated_image
```

4. erosion (temp_img,kernel,center,itr = 1):

erodes the passed image along with the specified kernel and the centre of the kernel no of iterations to be done can also be passed and returns the eroded image

```
def erosion(temp_img,kernel,center,itr = 1):
    height,breadth = temp_img.shape
    eroded_image = np.zeros((height,breadth))
    #print(eroded_image)
    k_height,k_breadth = kernel.shape
    for k in range(itr):
        for i in range(1,height-k_height-1):
            for j in range(1,breadth-k_breadth-1):
                flag = True
                for k in range(k_height):
                    for l in range(k_breadth):
                        if(temp_img[i+k][j+l] !=
255 and kernel[k][l] == 255):
                            flag = False;
                            break
                    if(not(flag)):
                        break
                if(flag):
                    eroded_image[i+center[0]][j+ce
nter[1]] = 255
        temp_img = np.copy(eroded_image)

    return eroded_image
```

5. opening (img,kernel,center,itr = 1):

Makes use of dilation and erosion function to facilitate the opening operation and returns the operated image

```
def opening(img,kernel,center,itr = 1):
    for i in range(itr):
        img = erosion(img,kernel,center,(1,1))
        img = dilation(img,kernel,center,(1,1))
    return img
```

6. closing (img,kernel,center,itr = 1):

Makes use of dilation and erosion function to facilitate the opening operation and returns the operated image.

```
def closing(img,kernel,center,itr = 1):
    for i in range(itr):
        img = dilation(img,kernel,(1,1))
        img = erosion(img,kernel,(1,1))
    return img
```

7. find_longest_component(image):

This function takes a binary image as an argument and finds the longest connected component in the image using breadth first search in a graph along with data structure queue and return a list of the connected pixels which can be used to make an image.

```
def find_longest_component(image):
    # Initialize variables
    max_size = 0
    max_index = None
    visited = set()
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0),
(1, 1), (1, -1), (-1, -1), (-1, 1)]

    # Iterate through every pixel in the image
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # If the pixel is black and has not been
visited before
            if image[i][j] == 255 and (i, j) not in
visited:
                # Initialize variables for BFS
                size = 0
                queue = [(i, j)]
                visited.add((i, j))

                # Perform BFS to find size of
connected component
                while queue:
                    i, j = queue.pop(0)
                    size += 1

                    for di, dj in directions:
                        ni, nj = i + di, j + dj
                        if 0 <= ni < image.shape[0]
and 0 <= nj < image.shape[1] and \
                                image[ni][nj] == 255
and (ni, nj) not in visited:
```

```
                            queue.append((ni, nj))
                            visited.add((ni, nj))

                # If current connected component is
bigger than previous ones, update
                if size > max_size:
                    max_size = size
                    max_index = (i, j)

    # Initialize variables for BFS on longest
component
    queue = [max_index]
    visited = set()
    visited.add(max_index)

    # Perform BFS on longest component and add to set
of pixels
    pixels = set()
    while queue:
        i, j = queue.pop(0)
        pixels.add((i, j))

        for di, dj in directions:
            ni, nj = i + di, j + dj
            if 0 <= ni < image.shape[0] and 0 <= nj <
image.shape[1] and \
                    image[ni][nj] == 255 and (ni, nj)
not in visited:
                queue.append((ni, nj))
                visited.add((ni, nj))

    return list(pixels)
```