CS 111 Programing Assignment 5: Geometric Transformation & Morphological Operators

Submission instructions:

Please **submit your code, output images and a PDF file (containing the output images) in a single zip file** to Canvas. You must also **submit the same PDF file** to Gradescope.

BOTH submissions are required for full points.
Your work is due by **11:59 p.m. on Saturday, the 1st of June**.

Introduction:

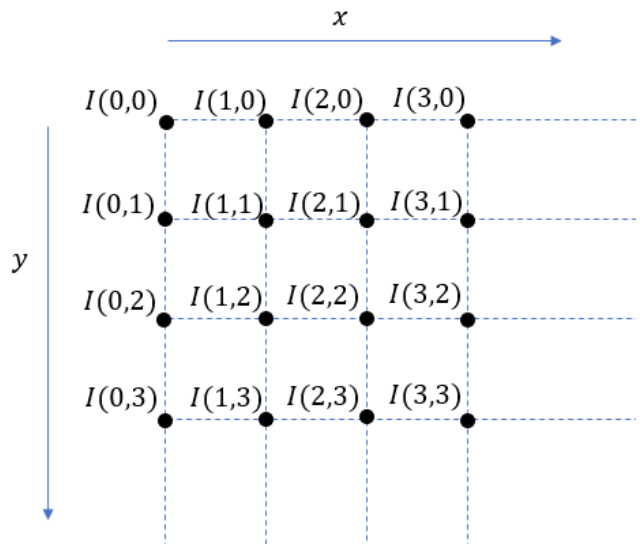
This programming assignment is focused on geometric transformation and morphological operators.

You will **NOT** be allowed to use any OpenCV functions unless told otherwise. In this assignment, you will work with gray and binary images only. So, use CV_LOAD_IMAGE_GRAYSCALE when reading your images.

Geometric Transformation:

I. Bilinear Interpolation:

a. In geometric transformation, we assume the image is a 2D function that has a value for each point on the 2D grid.



b. The transformation maps a 2D point to another 2D point.

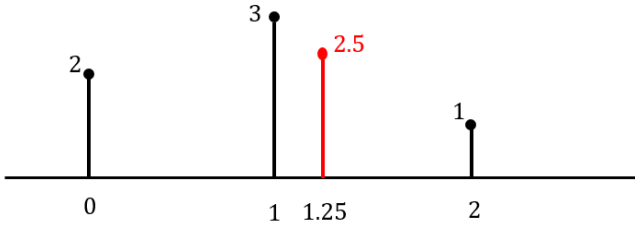
$$T(x,y) \rightarrow (x',y')$$

If (x',y') is on the grid, we can easily find its value, but it needs a general solution for real numbers. The (x',y') can be anywhere on the 2D space not limited to the integers (grid numbers). If the query point (x',y') is inside the range of the image (x: 0 to N-1, y: 0 to M – 1, for a MxN image) then the operation of finding the value for query point is "interpolation". If the query point is outside the coordinates of the image the operation for finding the value is called "extrapolation.

c. One simple method of interpolation is called "nearest neighbor". In this method you will assign the value of the closest grid point to the query point. Because of poor result and blockiness nature of this interpolation method, it is used very rarely.

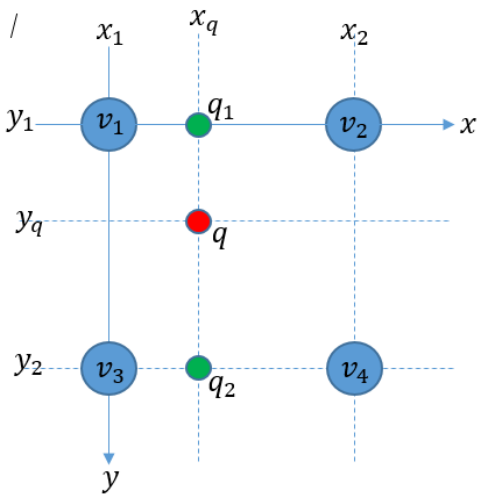
d. One important and widely used method for interpolation is called bilinear interpolation. In this method it is assumed that each grid block creates a 2D linear function and we can find the value for the query points in each block of the grid.

e. In 1D linear interpolation you can estimate the query value between known values. In this example we want to linearly interpolate the value at 1.25, which is between 1 and 2.



$$V(1.25) = (1.25 - 1)1 + (2 - 1.25)3 = 0.25 + 2.25 = 2.5$$

f. In 2D, we calculate the linear interpolation in one direction first and use its result to perform another linear interpolation in the other direction , as follows:



$$v(q_1) = (x_q - x_1)v_2 + (x_2 - x_q)v_1$$

$$v(q_2) = (x_q - x_1)v_4 + (x_2 - x_q)v_3$$

Then,

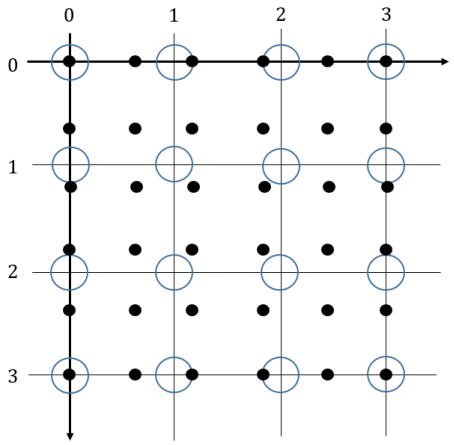
$$v(q) = (y_q - y_1)v(q_2) + (y_2 - y_q)v(q_1)$$

II. Image Resizing

a. Suppose we want to resize an image to twice of its current size. Since we will have more pixels in the output, we should interpolate the values between the pixels. In this work we will perform a bilinear interpolation. To resize a image we place the four corners of the output image on the same coordinates of the four corners of the input image and then interpolate all the output pixels.

b. Assume you have 4x4 image and now you want to resize it to 6x6. The four corners of the output image(6x6) is placed on the four corners of the input image (4x4) and all output pixels (black dots) are within the

range of the input image pixels (circles). In the following image, you can see how the query coordinates are placed.



c. In this assignment you should write a function that scales the input image into *s* times of its input size. The function Mat Resize(Mat I, float s) should first fill in the X and Y Mat's that contained the query point coordinates. Then you should calculate the query value by using bilinear interpolation.

Mat Resize(Mat I, float s)

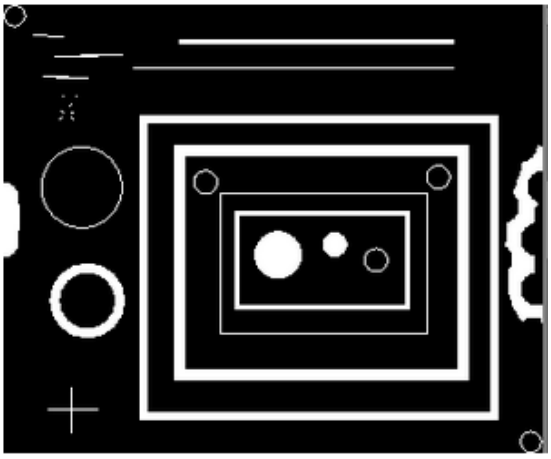
Complete the function *Resize(Mat I, float s)* in the "pa5.cpp", and apply on the input image "aerial.png" and write it in "aerial_640.png". Submit the code and the output image in the zip file.

Morphological Operators:

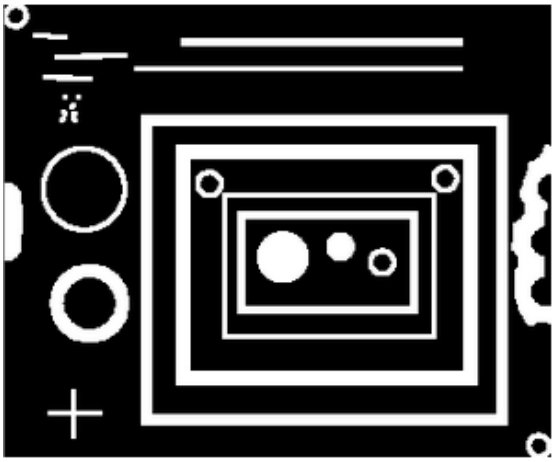
Morphology operations are a broad set of image processing operations which process the input binary images based on their shape. In a morphological operation, each pixel in the image is adjusted based on the value of other pixels in its neighborhood. By choosing the size and shape of the neighborhood, you can perform a morphological operation that is sensitive to specific shapes in the input image.

I. Dilation and Erosion:

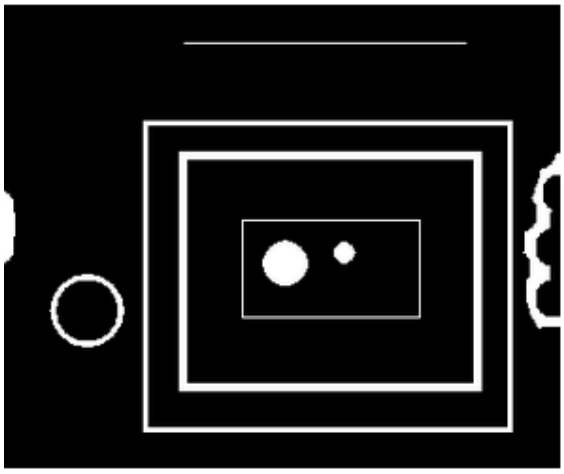
The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels from object boundaries. The three figures below show the effect of erosion and dilation on a binary image. The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image.



Input binary image



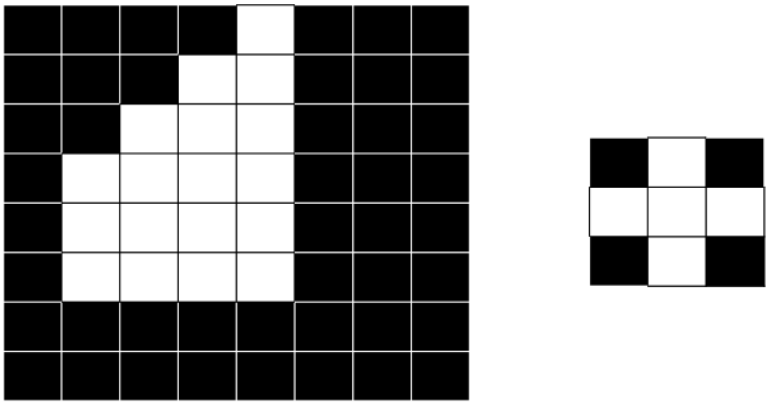
Dilated image



Eroded image

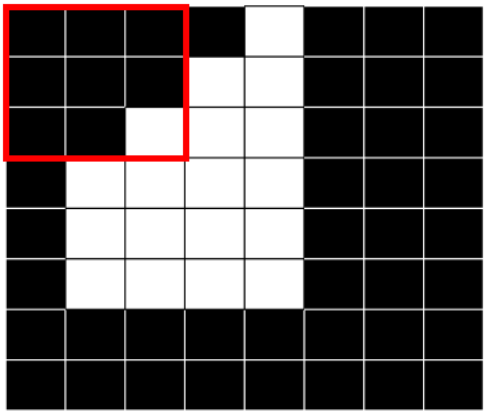
The structuring element acts as a mask: the neighborhood of pixels in the input binary image to be considered during morphological operations will be those that correspond to the pixel values of 1 in the structuring element. Consequently, the output of dilation and erosion will change depending on the shape of the structuring element.

The following diagrams explain how dilation and erosion work. Assume we start off with the following binary image and structuring element:

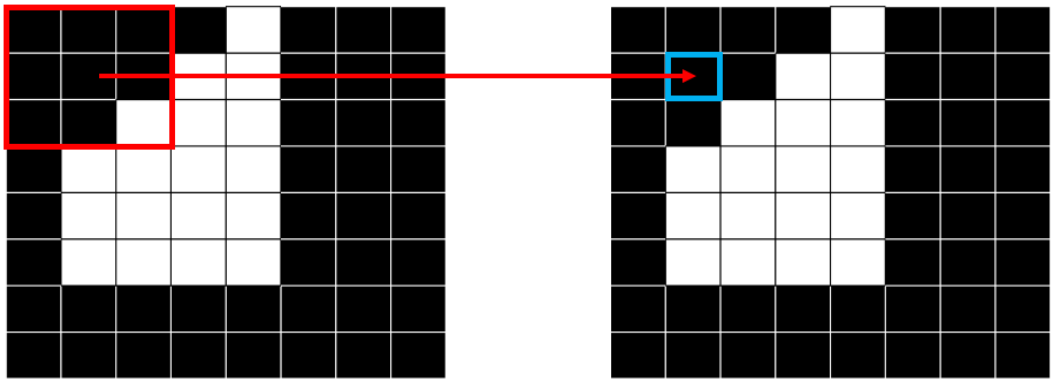


Left - Binary image, Right - Structuring element

As in convolution, the structuring element is centered on each pixel of the input image, including the border pixels.

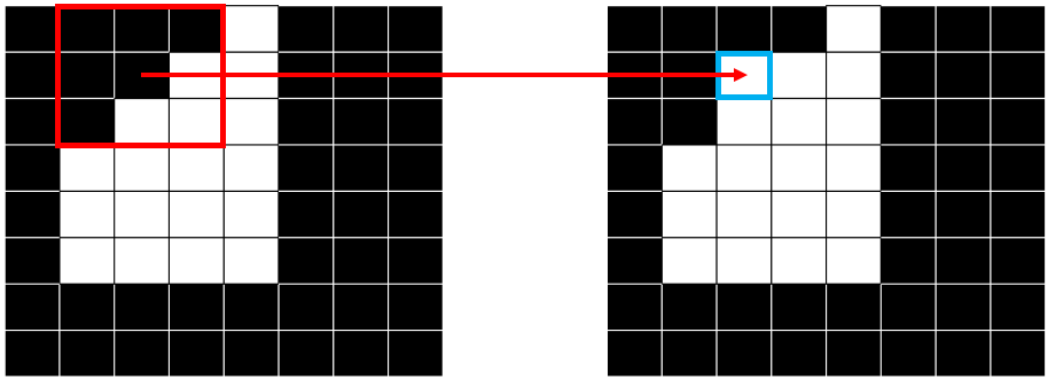


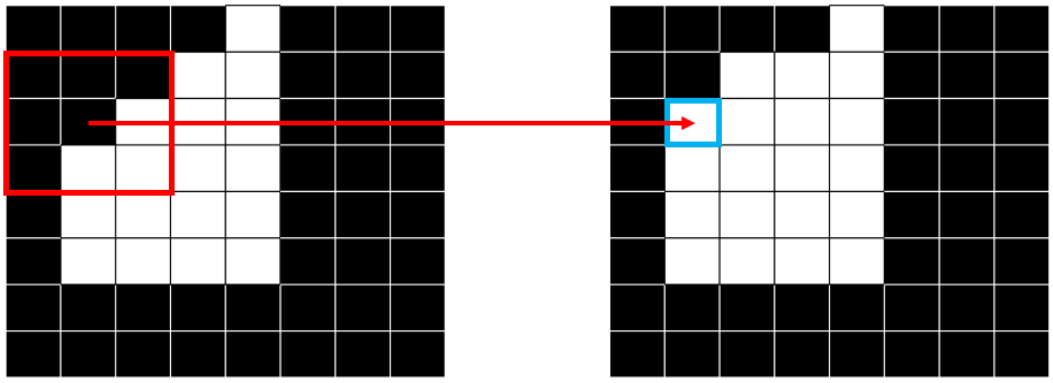
In the figure below, all the input pixels that correspond to 1 in the structuring element are 0. Therefore, for both erosion and dilation, the new pixel value will not change:



Since all overlapping input pixels are 0, there is no change in the output pixel value.

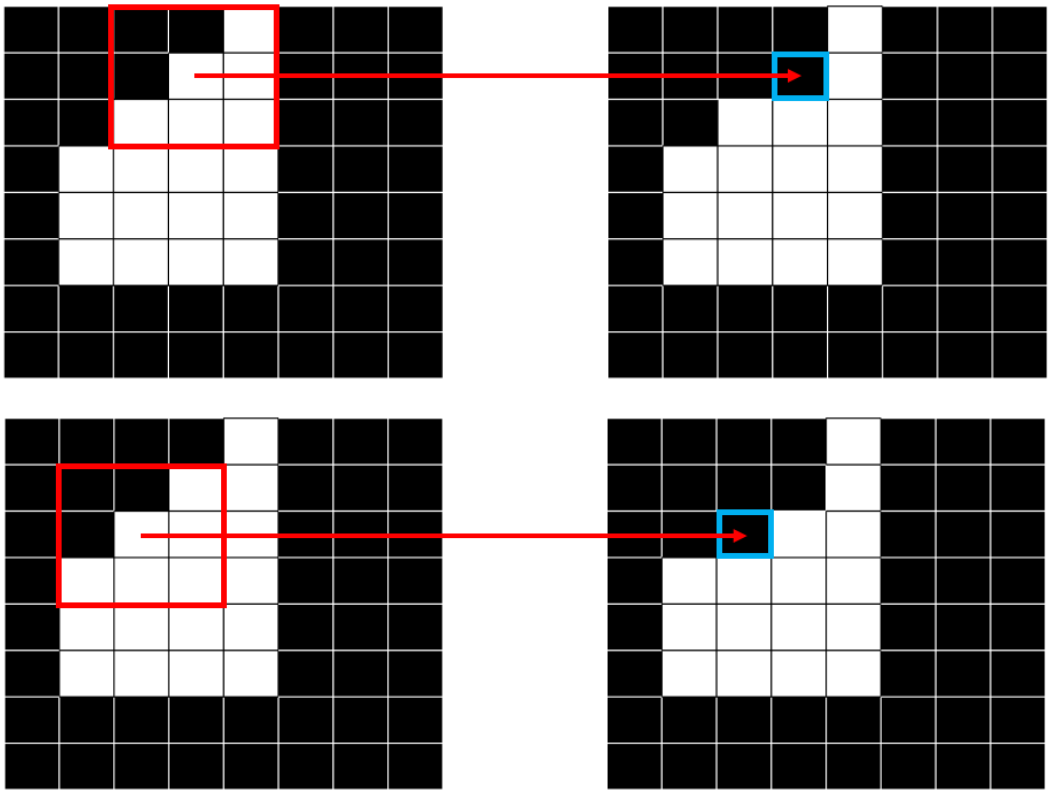
For dilation, the new center pixel value is 1 if *any* of the overlapping input pixel values is 1. In both the diagrams below, two input pixels with a value of 1 overlap with the structuring element. Therefore, the new pixel value at the corresponding center pixels will be 1.





Dilation sets pixels on the border to 1 since at least one of the overlapping input pixels is 1.

For erosion, the new center pixel value is 0 if *any* of the overlapping input pixel values is 0. In both the diagrams below, two input pixels with a value of 0 overlap with the structuring element. Therefore, the new pixel value at the corresponding center pixels will be 0.



Erosion sets pixels on the border to 0 since at least one of the overlapping input pixels is 0.

II. Opening and Closing:

Dilation and erosion are often used in combination to implement image processing operations. Two such operations are known as morphological *opening* and morphological *closing*. Morphological *opening* of an image is an erosion followed by a dilation, using the same structuring element

for both operations. On the other hand, morphological *closing* of an image is a dilation followed by an erosion, using the same structuring element for both operations.

Morphological opening is useful for removing small objects from an image while preserving the shape and size of larger objects in the image. Morphological closing is useful for filling small holes from an image while preserving the shape and size of the objects in the image.

For this assignment, you will be modifying **pa5.cpp** to implement the above four morphological operations on the provided binary image "**binary.bmp**". You will generate results using two different structuring elements: a square of size 11 and a circle of diameter 11. Both these structuring elements will be *cv::Mat* objects of size 11x11, however they will have different shapes: the square will have ones in every location whereas the circle will not. An example is shown in the figures below.

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0	0

Note: While you may fix the size of your structuring elements, the functions you write must be able to handle 2-D structuring elements of any dimension and shape.

Note: You must make sure that the image is binary after reading it i.e. it can take only two values: 0 & 1 (or 255). Similarly, the image you write must be of BMP format so that image compression artifacts do not introduce other values.

III. Dilation:

a. Write a function *Mat Dilate(Mat I, Mat elem)*, that performs dilation on an input binary image *I* using the structuring element *elem*.
b. Perform dilation on the provided image "**binary.bmp**" using both structural elements.
c. Save your results as "**dilation_square.bmp**" and "**dilation_circle.bmp**".
d. Submit your results and document them in your PDF file.

IV. Erosion:

a. Write a function *Mat Erode(Mat I, Mat elem)*, that performs erosion on an input binary image *I* using the structuring element *elem*.
b. Perform erosion on the provided image "**binary.bmp**" using both structural elements.
c. Save your results as "**erosion_square.bmp**" and "**erosion_circle.bmp**".
d. Submit your results and document them in your PDF file.

V. Morphological Opening:

a. Write a function *Mat Open(Mat I, Mat elem)*, that performs morphological opening on an input binary image *I* using the structuring element *elem*.
b. Perform opening on the provided image "**binary.bmp**" using both structural elements.
c. Save your results as "**opening_square.bmp**" and "**opening_circle.bmp**".
d. Submit your results and document them in your PDF file.

VI. Morphological Closing:

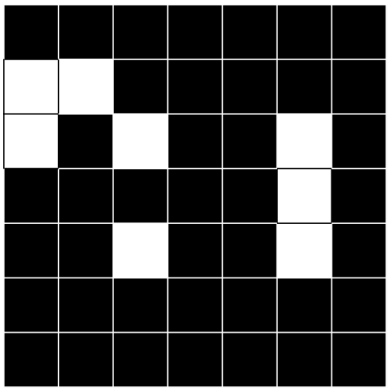
a. Write a function *Mat Close(Mat I, Mat elem)*, that performs morphological closing on an input binary image *I* using the structuring element *elem*.
b. Perform opening on the provided image "**binary.bmp**" using both structural elements.
c. Save your results as "**closing_square.bmp**" and "**closing_circle.bmp**".
d. Submit your results and document them in your PDF file.

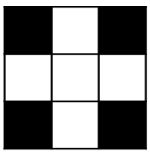
VII. Connected Components:

In class, you learnt how morphological operators can be used to compute the connected components of a binary image. In this part of the assignment, you will write a function to compute the connected components of a binary image using morphological operators.

Computing connected components using morphological operators is an iterative process that involves dilation. Below we will run through an example binary image and how connected components will be computed for that.

Let's assume we have the following binary image, denoted by *I*, and the structural element, denoted by *S*:





The first step is to select **any** pixel from I which is white and put that in a new image. Let's call that new image X_0:

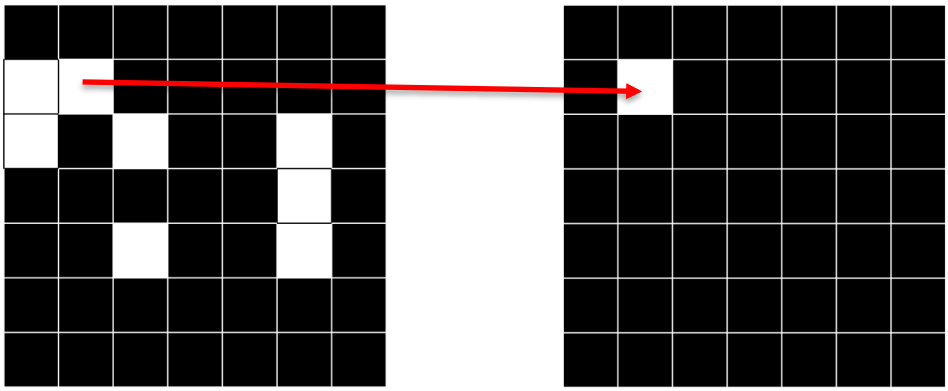
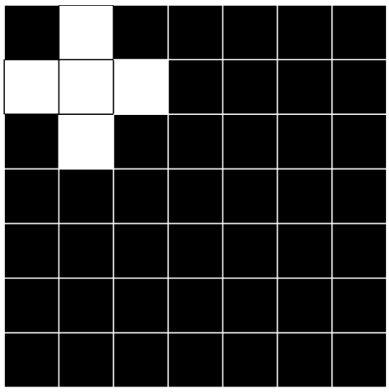
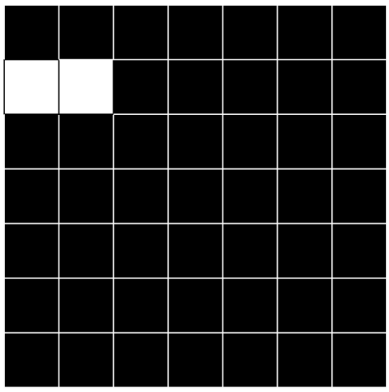


Image I (left) and X_0 (right)

We must now compute the image X_1. X_1 is the dilation of X_0 with the structural element S, followed by an intersection with I. The image on the left below shows the dilation of X_0 with S. The image on the right shows the intersection of the dilated image with I:

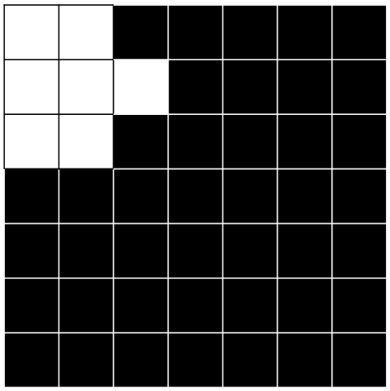


$(X_0 \oplus S)$

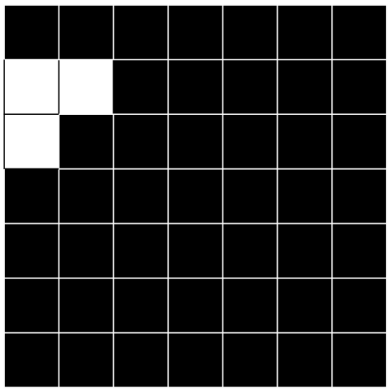


$X_1 = (X_0 \oplus S) \cap I$

Next, we must compare X_1 and X_0. Are they equal? If so, we have our connected component. If not, we repeat the dilation and intersection steps above. In our case, X_0 is not equal to X_1, therefore, we must continue the dilation and intersection operation on X_1 to get X_2.

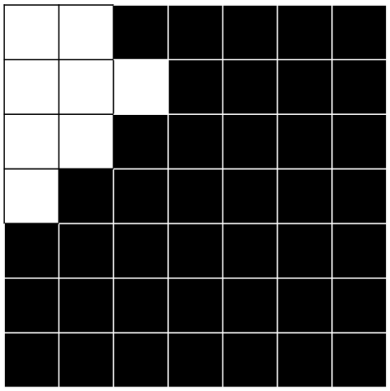


$(X_1 \oplus S)$

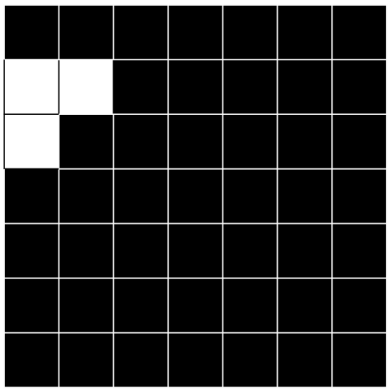


$X_2 = (X_1 \oplus S) \cap I$

Again, we compare X_2 and X_1 and find that they are not equal. So, we repeat dilation and intersection on X_2 to get X_3:

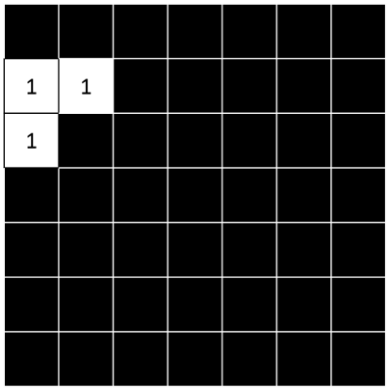


$(X_2 \oplus S)$

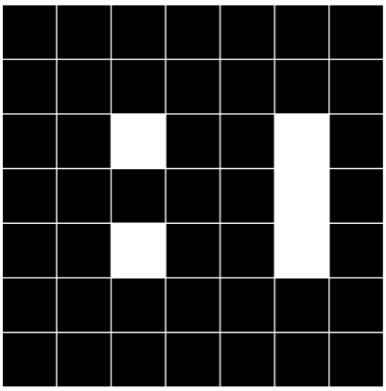


$X_3 = (X_2 \oplus S) \cap I$

Notice that now, X_3 is equal to X_2. Hence, our first connected component is X_3. We will copy these pixels over to a new image, called a *label image*, and assign these pixels a *label*. A label is simply a number assigned to a pixel. In this case, we will assign the pixels a label of 1, since they are the first connected component we found. The label image will look like this:



We are not done yet! There are still other connected components in our image I. However, every time we find a component, we must remove it from our original image and repeat. Therefore, removing X_3 from I, we get the updated I:

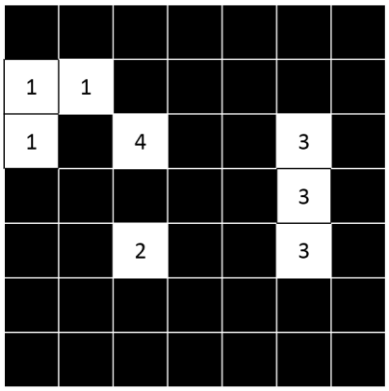


Now, we will repeat the same process on the updated I, i.e. select any random white pixel, dilate and intersect, until we have removed all connected components from it, i.e. image I is all zeros. That is when there are no more components left in the image.

The equation that represents finding a single connected component is:

$$X_k = (X_{k-1} \oplus S) \cap I \quad \text{while } X_k \neq X_{k-1}$$

Depending on the order of the pixel selection, the final label image may look something like the image below. Note that **each pixel will have a label**, and that **pixels belonging to the same connected component will have the same label**. The numbering of the labels might be different.



a. Write a function *Mat ConnectedComponents(Mat I, Mat elem)*, that computes the connected components on an input binary image *I* using the structuring element *elem*.
b. The function should return a label image of the same size as the input image, containing the connected component label assignments of each pixel in the input image.
c. Compute the connected components of the provided image "**binary.bmp**" using a **3x3 square structural element** (i.e. all one's).
d. Save your result as "**connected_components.bmp**". You might need to rescale the label image between 0 and 255 to make the label assignments visible.
e. Submit your result and document them in your PDF file.