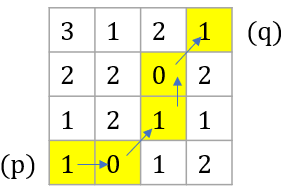
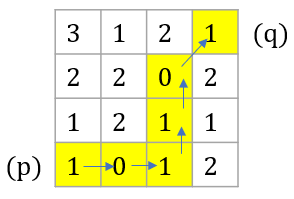
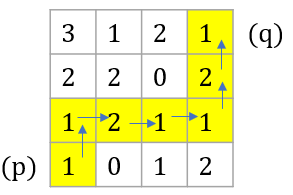
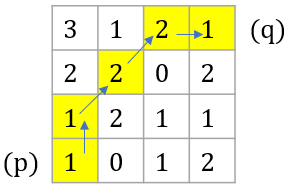
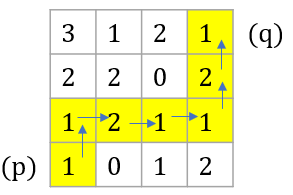
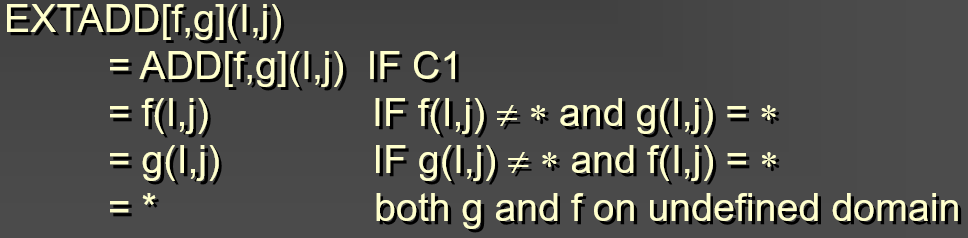
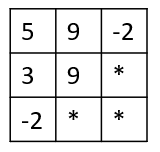
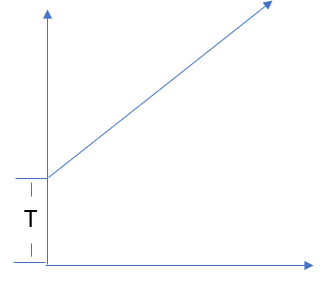
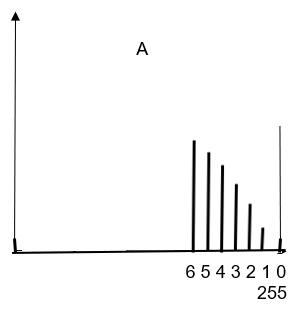
* 1. The two image subsets are not 4 adjacent because none of the pixels in either of the two image subsets where V={1} are 4 adjacent.
  2. The two image subsets are 8 adjacent because a pixel in S1 is 8 adjacent with a pixel in S2 by a diagonal.
  3. The two image subsets are M adjacent because the two pixels in the subsets are 8 adjacent, and the intersection of their 4 adjacent neighbors where V={1} is empty.
     1. The shortest 4 path between p and q does not exist because q is 4 adjacent to only two pixels of value 2, so there is no way to get from p to q through {0,1}
     2. The shortest 8 path (shown below) has length 4 
     3. The shortest M path (shown below) has length 5 
  4. The 4, 8, and M paths are shown below from left to right for V = {1,2} 

1. F
   1. From the slides this is the block diagram for EXTADD
   2. EXTADD(f,g) = 
   3. 
   4. 

Writeup:

Connor Tremblay

[ctrembl1@binghmaton.edu](mailto:ctrembl1@binghmaton.edu)

Purpose:

The purpose of this project was to implement a negative image function, histogram equalization function, image thresholding function, and to label the largest regions of an image after performing a thresholding function.

Method:

For my negative image function, I set each pixel to 255-value which is a very well known and common way of implementing a negative function.

For my histogram equalization function, I redistributed the CDF of the pixel value probabilities over the entire domain of gray values, creating an image which is enhanced visually.

For my thresholding function, I used Balanced histogram Thresholding, and was inspired by the algorithm linked in the following page: <https://en.wikipedia.org/wiki/Balanced_histogram_thresholding>

For my region detection algorithm, I first create a thresholded image then iterate through the pixels of the array and check if 4 adjacent pixels have the same white value, and store all values in a region in a vector. I then store that vector in another vector. I store checked pixels in a boolean array to prevent redundant checking. Lastly I sort the vector of vectors based on size of the regions and color the regions appropriately.

Results:

Everything that I implemented works as expected and I am very pleased with the results. The image thresholding algorithm causes some edge defects around the shapes, however I think that this is just my choice of algorithm and is not really an error.

Bug report:

None to mention