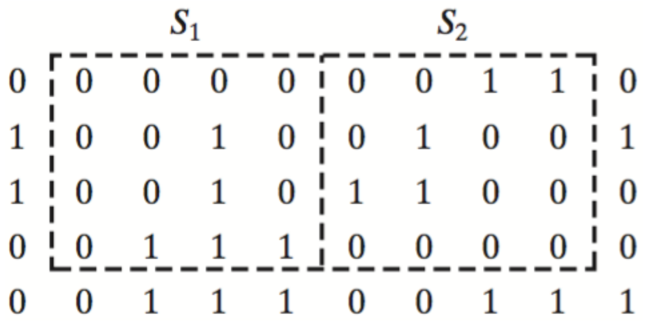
ECE558 Homework 02 (100 points in total)

Due 10/03/2018

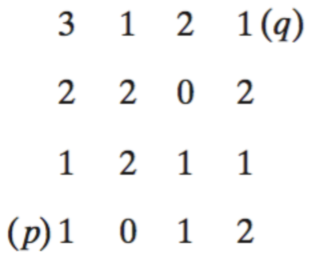
How to submit your solutions: put your report (word or pdf) and results images (.png) in a folder named [your_unityid]_hw02 (e.g., twu19_hw02), and then compress it as a zip file (e.g., twu19_hw02.zip). Submit the zip file through **moodle**.

Problem 1 (10 points): Consider the two image subsets, S_1 and S_2, shown in the following figure. For V = {1}, determine whether these two subsets are (a) 4-adjacent, (b) 8-adjacent, and (c) m-adjacent. Show the justification in detail.



	S_1				S_2				
0	0	0	0	0	0	0	1	1	0
1	0	0	1	0	0	1	0	0	1
1	0	0	1	0	1	1	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1

Problem 2 (30 points): Consider the image segment shown.



	3	1	2	1	(q)
	2	2	0	2	
	1	2	1	1	
(p)	1	0	1	2	

(a) (8 point) Let V = $\{0, 1\}$ and compute the lengths of the shortest 4-, 8-, and m-path between p and q. Show the corresponding paths. If a particular path does not exist between these two points, explain why.

(b) (2 point) Repeat for V = $\{1,2\}$.

(c) (20 point) Write matlab or python code to implement the function.
Input arguments: an image segment, a predefined set V, two pixel locations p and q, the path type
Outputs: the length of the shortest path, and the path

Test your code: you should test your code thoroughly to make sure it can handle different situations, e.g., a particular path does not exist between the given p and q, as well as invalid input arguments; create at least 3 more different testing examples beside the given one for (a) and (b); record the run time of your code.
Requirement: Built-in functions in Matlab or Python for finding the shortest path cannot be used. You need to implement a self-contained code from scratch. Submit your code together with your results.
Optional: you can think about how to exploit the idea of representing image as graph, and then you can write a general function for finding the shortest path in the graph.

Problem 3 (60 points): Write matlab or python code to implement 2D convolution for gray images.

Input arguments: an input gray image, a predefined filter kernel, output shape options (full, same, or valid), padding options (zero, wrap around, copy edge, or reflect across edge)
Outputs: the filtered image
Test your code: test your code by computing the filtered image for the provided lena-gray.bmp image. You are welcome to include a few more testing results on other images if you want.

Requirements: Built-in conv. Functions cannot be used. You need to implement a self-contained 2D convolution function from scratch. Organize your code nicely. Submit your code together with testing results.

Optional: you can also think about how to implement 2D convolution for both color and gray images.