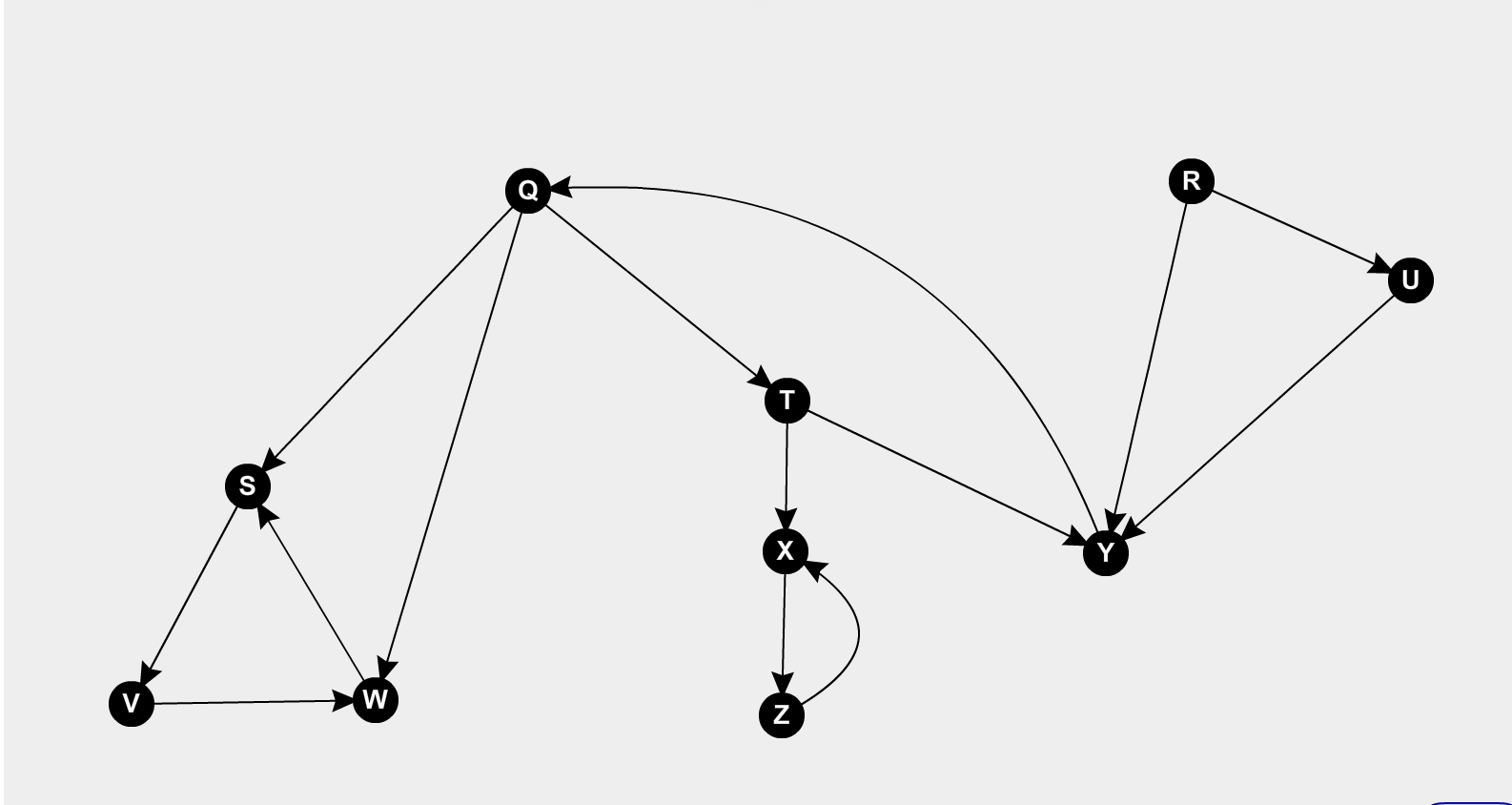
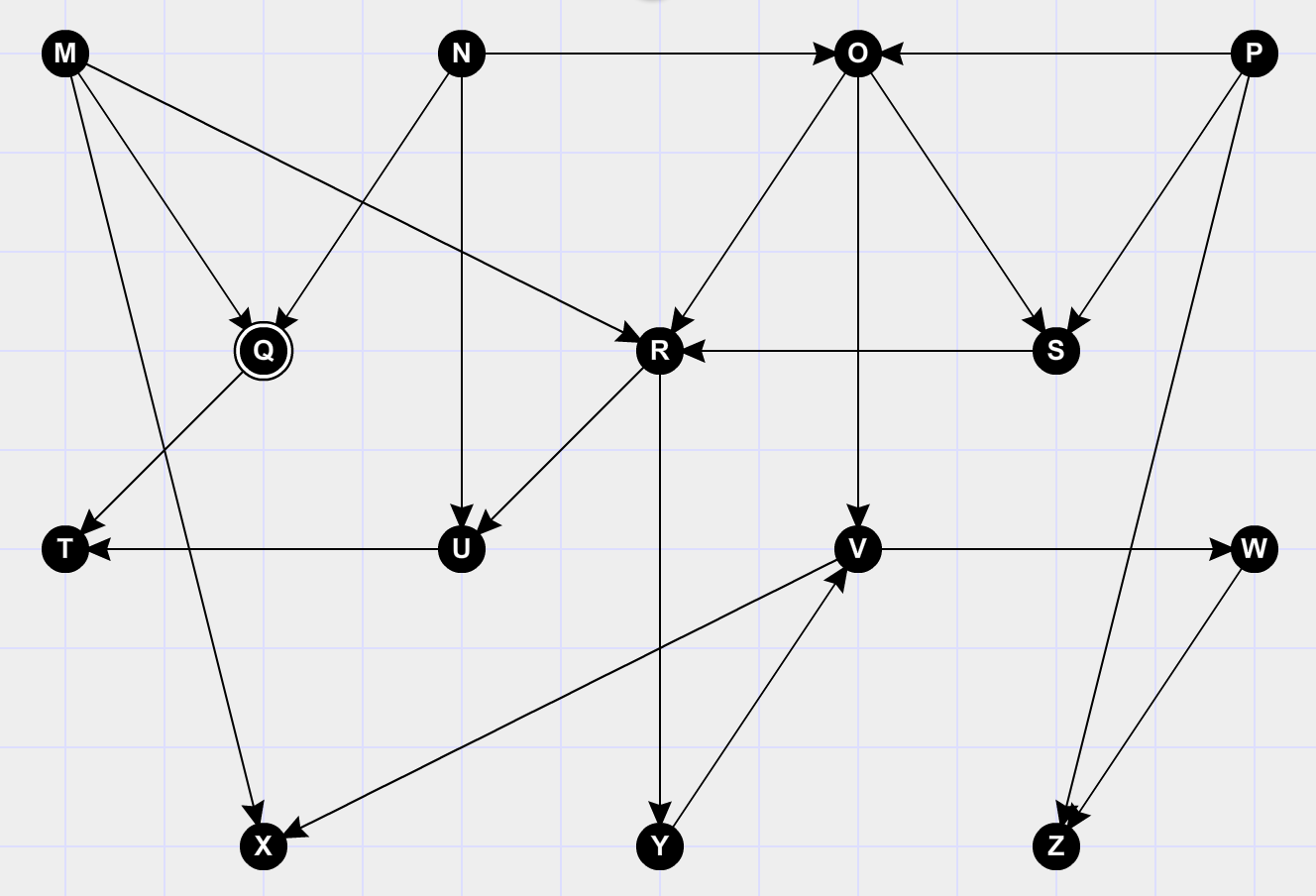
Franklin SmithCIS 315 Intermediate Algorithms Spring 2018Assignment 11. In that graph that’s given to us in the adjacency matrix representation, there cannot be more than one sink. This is because there would be another edge from the first sink to the second sink which would mean that the first sink would need an out degree which would violate the property of a sink in graph theory.Assuming we are given a NxN matrix M

i = 0j = 0 Here we start at position M[0][0] which is the first position in position = M[i][j] the matrix. We then iterate though the matrix, if the indexedwhile( i <= n and j <= n): position is a 0 we move right by one item in the matrix. If the if position == 0: indexed position is a 1 then we move down by one item in the   
 j = j+1 matrix. After this when either the row or column indexes are if position == 1: exhausted in the while loop we end up at a point in the matrix. i = i + 1 which would be M[i][j]. After this we iterate through the rowresult = True of the ith index in the matrix. We check to make sure that all for k in range(len(M)): of the items in this row are 0. After this we iterate thorough if M[k][j] != 0: the column j checking to see that all of the items in this row result = False are 1. Except for the item at the intersection between the rowfor p in range(0,j,1): and the column, this item is 0. The two for loops that split if M[j][p] != 1 : from (0,j)U(j+1,n) skips the item at the intersection. The result = False complexity of this algorithm is θ(n) time. This is because for t in range(j+1, n, 1) the first while loop is O(2n) (worst case). The first for loop if M[j][t] != 1 following the while loop is O(n). We drop all of the constants result = False so the algorithm runs in θ(n)return result

22.2-7To solve this problem I would initialize a graph of which every vertex on the graph symbolizes a wrestler and each edge represents a rivalry. That being said the graph will contain r edges and n vertices. I would do the number of Breadth First Searches required to visit all vertices. After this I would group the wrestlers that have even distance in the category “baby faces”. I would group the wrestlers that have odd distance in the category “heels”. After this I would analyze every edge to double check that it goes from a heel to a baby face. This specific algorithm would have O(n+r) time for the breadth first searches, and it would take O(n) time to analyze each distance as odd or even and categorize them into baby faced, or heels, it would take O(r) time to check edges, which in total is O(n+r) time overall.

22.3-2 Showing a depth-first search works on the below graph

22.4-1 Show the ordering of the vertices produced by Topological-Sort when its run on the below DAG.



5. Give a linear time algorithm to find the length of the longest path from node 1 to node n.

To start I would find the topological sorting of the graph by doing the topological sort. Which would represent a linear representation of the graph. After this I would process each of the vertices in the graph one by one in topological order. Upon each vertex that I am processing, I would update the distance of its adjacent with the distance of the current vertex. The time complexity of this algorithm is O(V+E). The complexity of topological sort is O(V+E). After we are done with the first step of topological sort the algorithm processes all of the vertices, every time the algorithm arrives upon a vertex, it iterates over all of the adjacent vertices. The amount of adjacent vertices in the graph is O(E), thus the inner loop is ran O(V+E) times. Thus, the overall complexity of the algorithm is O(V+E).