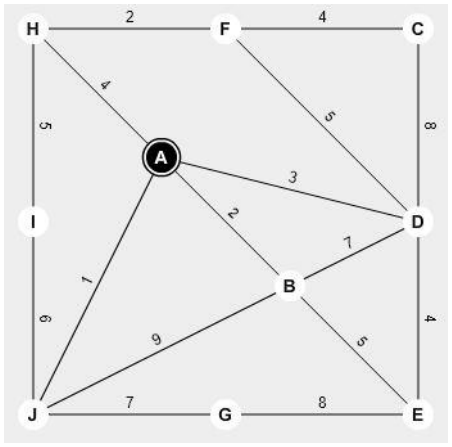
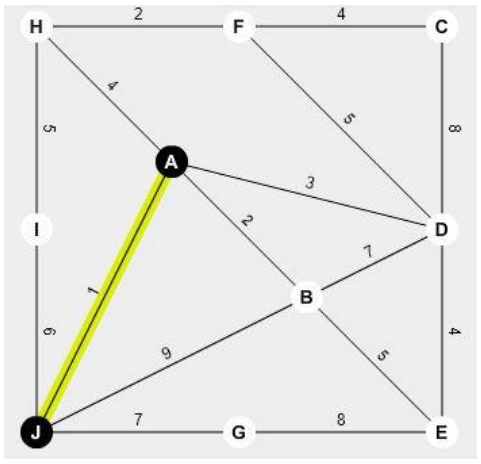
Erin Alltop
CS325 – Winter 2017
Homework 5
11-1-17

1. Prim's Algorithm – The weight of this minimum spanning tree is 32.

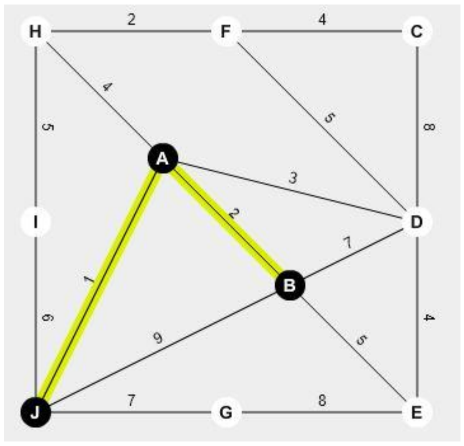
a)



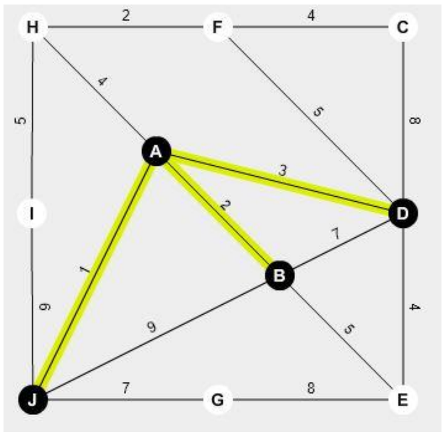
b)



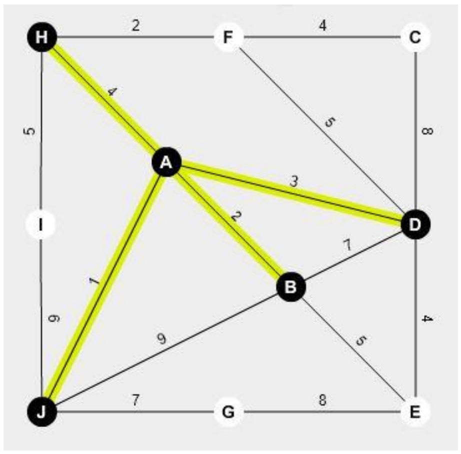
c)



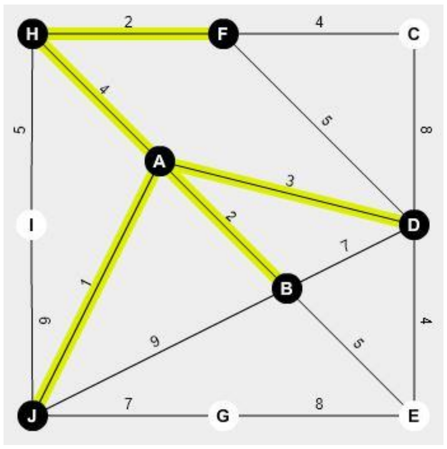
d)



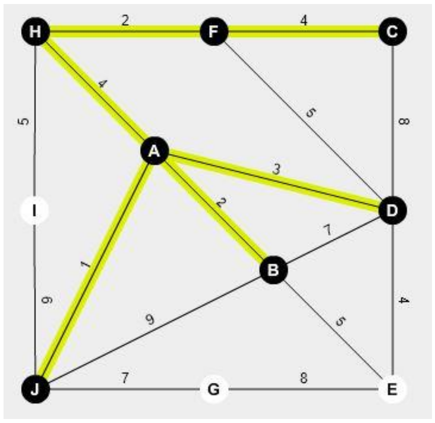
e)



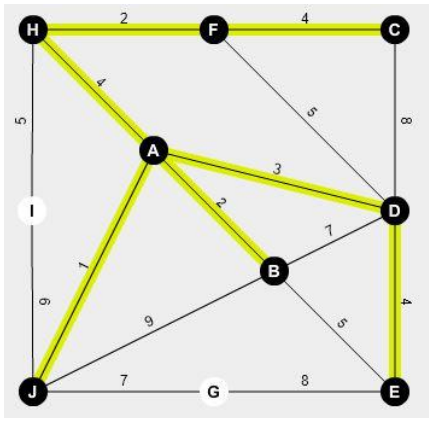
f)



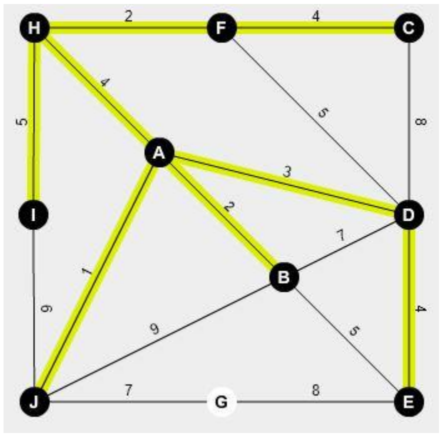
g)



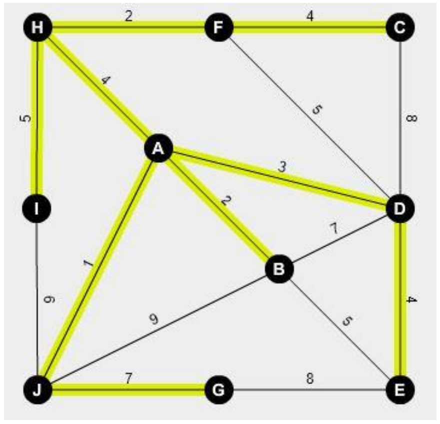
h)



i)



j)



2.

a. No, the minimum spanning tree does not change if all of the edge weights are increased by one. This can be seen clearly because increasing all of the edge weights by 1 is increasing all of the edge weights by the same constant, therefore this would not change the minimum spanning tree solutions.

b. Similarly to part a, the shortest paths of the minimum spanning tree would not change either because increasing every weight by the same constant would result in the same solutions. However, it is possible for the minimum spanning tree to have multiple shortest path solutions, so the shortest path could potentially change, but in the same way that it could be changed whether the edge weights are n or n+1. For example, if there are two possible greedy choice (e.g. two edges of weight 4 that can be selected), the algorithm might choose one of the two shortest paths.

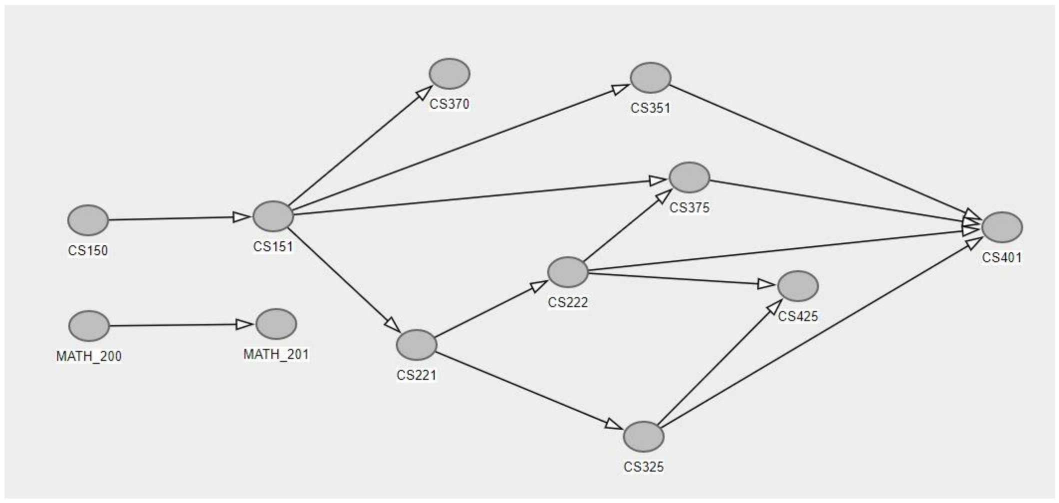
3.

a. The simplest and most efficient algorithm to solve this problem is to use the existing Bread-First Search algorithm and to not include any edges that are less than weight W. In a BFS algorithm we would start at the source node and search each neighbor node until we have visited every node. In this algorithm, we would simply check each edge node first to determine if it is less than weight W. If so, we will ignore if and move one, otherwise we will include it in our results.

b. This extra step would not change the BFS algorithm running time in any significant way as it includes only a simple comparison, so the running time would still be O(V+E), which is the same as the normal BFS algorithm.

4.

a. Directed Acyclic Graph



b. One possible topological sort of this graph is: CS150, MATH_200, CS151, MATH_201, CS370, CS221, CS351, CS222, CS325, CS375, CS425, CS401. This sort allows the student to take all classes after having met all necessary prerequisites.

c. Order of classes that can be taken in the fewest amount of terms.

Term 1	CS150, MATH_200
Term 2	CS151, MATH_201
Term 3	CS370, CS221, CS351
Term 4	CS222, CS325, CS375
Term 5	CS425, CS401

d. The longest path in this graph is 5, which is from vertices CS150, CS151, CS221, CS222, CS375, CS401. I was able to find this by looking at the graph and determining the longest path that still meets the prerequisites of the class before. This is the same as the minimum number of terms needed to complete the program because of the dependencies of each class on its prerequisites. Even though there are short paths available, some classes cannot be completed until others are completed first.

5.

a. If we were to represent the wrestlers and vertices and the rivalries as edges on a graph, we could use a Breadth-First Search to search all vertices and edges to determine the rivalries that exist between Babyfaces and Heels. We could do this by having the first wrestler be a Babyface and neighboring vertices be Heels, continuing until we run out of vertices. This would guarantee that as many wrestlers as possible have a rivalry with someone of the opposite faction.

Pseudocode:

```
BabyHeel_Seach(graph, start) // using BFS
    Start.color = 'white'
    Visited = [ ]  //explored vertices
    Queue = [start] //vertices to be checked
    While queue: //while queue is not empty
        Vertex = queue.pop(0) //get first vertex
        If not in Visited, add to visited list
        Vertex.color = opposite of adjacent vertex
        Check for neighbors and add to queue to be checked
    If an edge exists and both vertices have the same color : return false
    If all edges have been checked and none have the same color : return true
```

b. The running time of the algorithm would be O(V + E) which is the same as the Breadth-First Search.