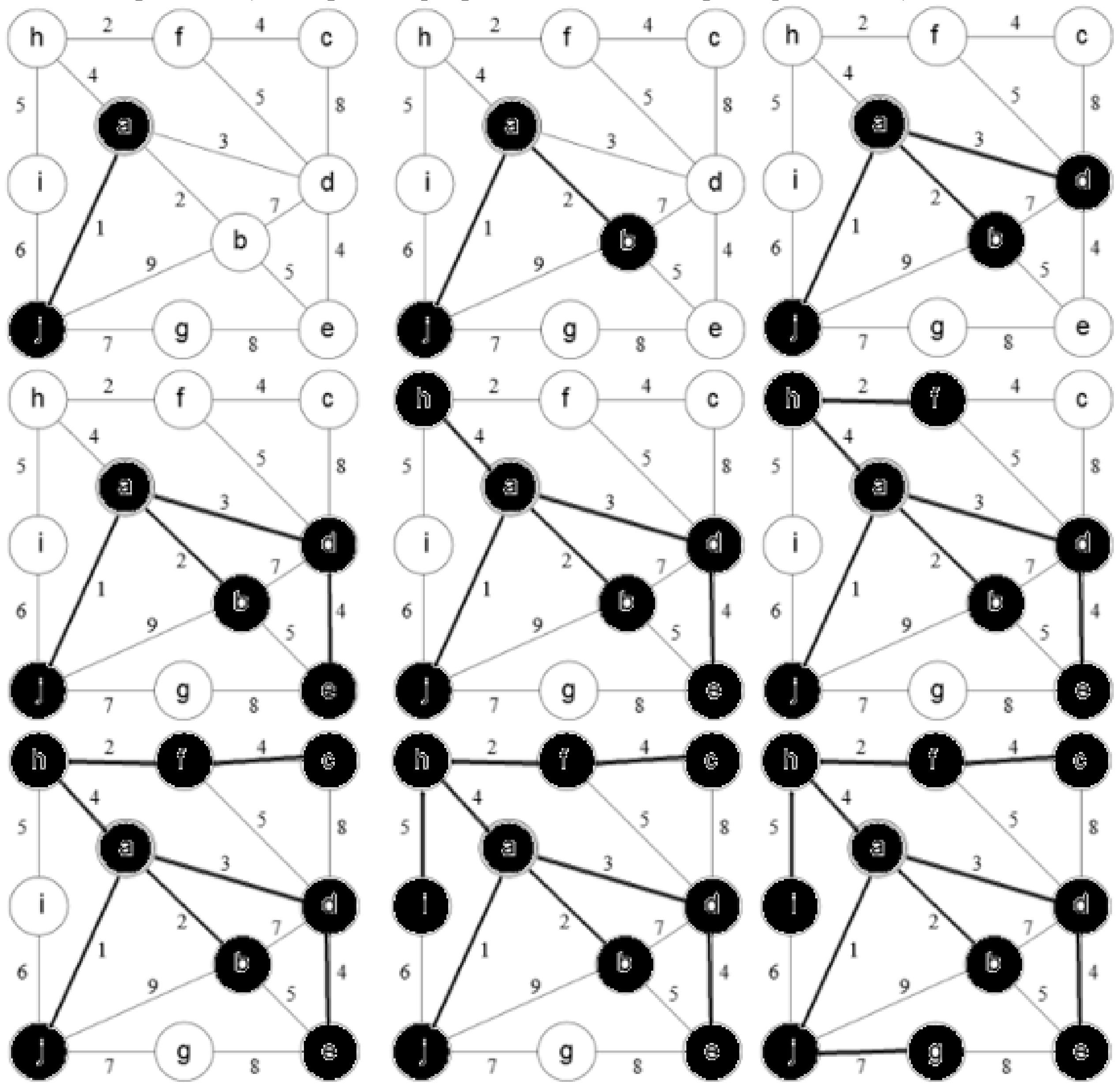
Josh Huff, huffj@oregonstate.edu
Professor Schutfort
CS 325 – Winter 2018
February 18, 2018

Homework 5

---

1. Demonstrate Prim's algorithm on the graph below by showing the steps in subsequent graphs as shown in Figures 23.5 on page 635 of the text. What is the weight of the minimum spanning tree? Start at vertex a.

The MST weight is 32. (The algorithm progresses from left to right, top to bottom).



2. Consider an undirected graph G=(V,E) with nonnegative edge weights w(u,v) >= 0. Suppose that you have computed a minimum spanning tree G, and that you have also computed shortest paths to all vertices from vertex s is a member of V. Now suppose each edge weight is increased by 1: the new weights w'(u,v) = w(u,v) + 1.

(a) Does the minimum spanning tree change? Give an example it changes or prove it cannot change.

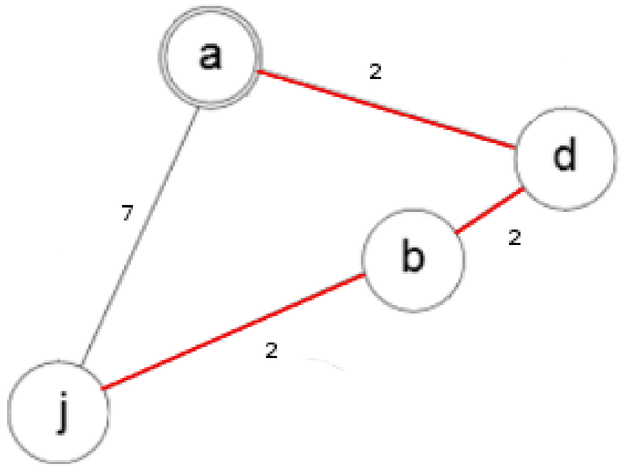
The structure of the MST remains the same.

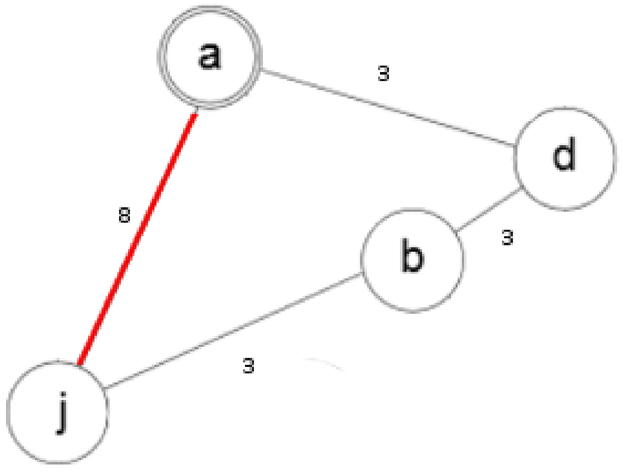
Because every edge weight is uniformly increased by one, the difference of proportions between them do not change. Therefore, the nodes will be visited in the same order. The MST's weight, however, will increase by x – 1, where x is the number of nodes in the MST. For example, the graph from problem 1 has ten nodes: {a, b, c, d, e, f, g, h, i, j}. Increasing the weight of all edges by one will increase the weight from 32 to 41.

(b) Do the shortest paths change? Give an example where they change or prove they cannot change.

The shortest paths have the potential to change.

As seen in the example "before-and-after" graphs below, incrementing each edge weight by one has a cumulative effect on shortest paths solutions. If a shortest path is composed of several edges, and a competing single edge is close enough in weight, the solution may change.





3. In the bottleneck-path problem, you are given a graph G with edge weights, two vertices s and t and a particular weight W; your goal is to find a path from s to t in which every edge has at least weight W.

(a) Describe an efficient algorithm to solve this problem.

The modified BFS is roughly as follows:

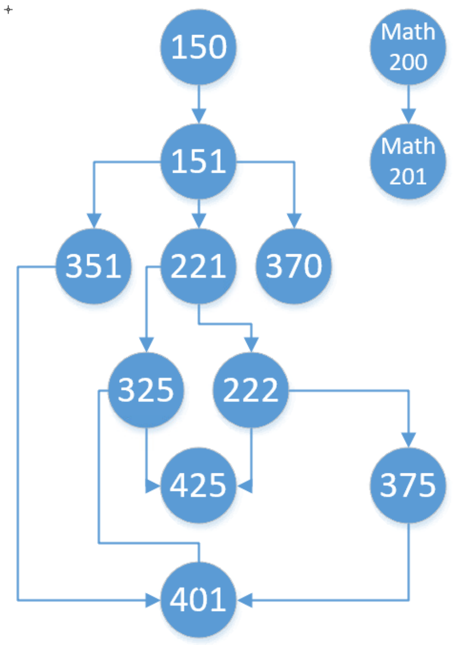
```
Start at some source vertex, s.
Create a list of nodes that have been visited, V.
Create a queue of nodes to be visited, Q.
Add the source vertex to Q.
Repeat the following steps until Q is empty:
    Remove the first item, f, from Q.
    If f is not in V, add f to V.
    For all edges, e, from f:
        if the weight of e is not less than W:
            add e to Q.
If V contains t, it is successful in establishing a path, then you are finished. Return visited.
If it does not, there is no path from s to t with each edge meeting the minimum weight.
```

(b) What is the running time of your algorithm.

This is a variation of the breadth-first search, and thus has a running time of O(V + E).

4.

(a) Draw a directed acyclic graph (DAG) that represents the precedence among the courses.



(b) Give a topological sort of the graph.

CS 150, MATH 200, CS 151, MATH 201, CS 221, CS 351, CS 370, CS 222, CS 325, CS 375, CS 425, CS 401

(c) If you are allowed to take multiple courses at one time as long as there is no prerequisite conflict, find an order in which all the classes can be taken in the fewest number of terms.

Term 1:	CS 150 and MATH 200
Term 2:	CS 151 and MATH 201
Term 3:	CS 221, CS 351, and CS 370
Term 4:	CS 222, CS 325
Term 5:	CS 375, CS 425
Term 6:	CS 401

(d) Determine the length of the longest path in the DAG. How did you find it? What does this represent?

The longest path has 5 edges. It consists of nodes CS150, CS151, CS221, CS222, CS375, CS401.

The longest path can be found using a depth-first search, with CS150 as the start node.

The number of nodes (that is, the longest path plus one for the start node) represents the minimum number of terms a student must enroll in order to earn a CS degree.

5. Suppose there are two types of professional wrestlers: “Babyfaces” (“good guys”) and “Heels” (“bad guys”). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n wrestlers and we have a list of r pairs of rivalries.

(a) Give pseudocode for an efficient algorithm that determines whether it is possible to designate some of the wrestlers as Babyfaces and the remainder as Heels such that each rivalry is between a Babyface and a Heel. If it is possible to perform such a designation, your algorithm should produce it.

```
Assume the designations are possible until proven otherwise (set a designations flag to True)
Create a graph called WWF, and populate it with nodes representing each wrestler
Edges between wrestlers indicate rivalries
Create sets representing visited nodes and each side of the bipartite graph
Start with a source vertex – the first wrestler’s name
Because the graph is not necessarily connected, it is a forest with each wrestler as the root of a tree.
Visit each tree, skipping them if they have already been visited (if they are already in the visited set)
    For each tree, create a queue
    Until the queue is empty, pop the leftmost element and visit its neighboring nodes
        Add that popped node to the visited set
        Then establish if all of those neighbors are in the same set:
            If one is not, then the designation is impossible. You are finished.
            If all the neighbors are in heels, the wrestler is a babyface.
            If all the neighbors are in babyfaces, the wrestler is a heels.
If designation is possible, print the contents of each set. You are finished.
```

(b) What is the running time of your algorithm?

This is a variation of the breadth-first search, and thus has a running time of O(V + E).