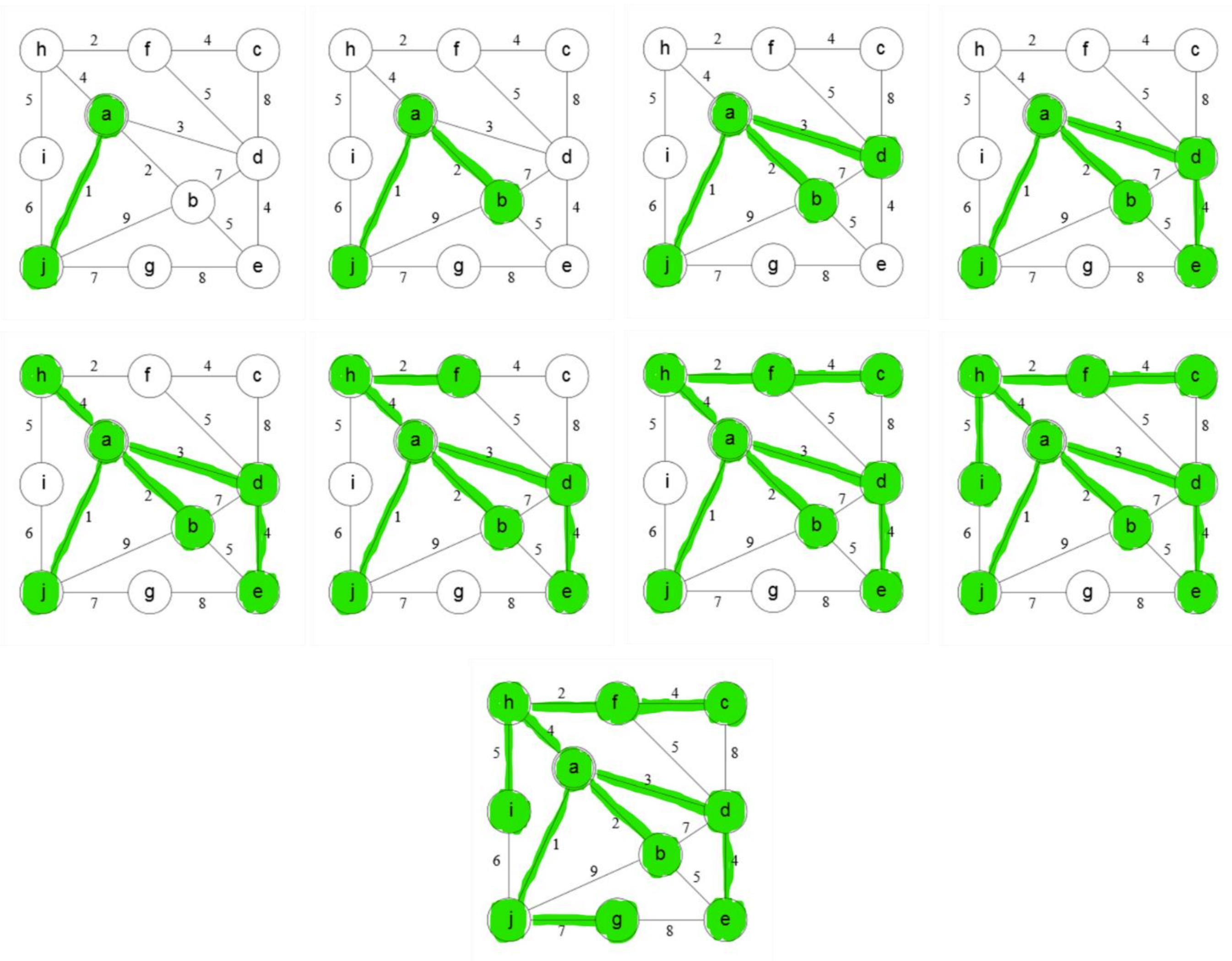
Stephen Townsend
CS 325-400
11/5/17

Assignment 5

1. *(3 points)* 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.



Stephen Townsend
CS 325-400
11/5/17

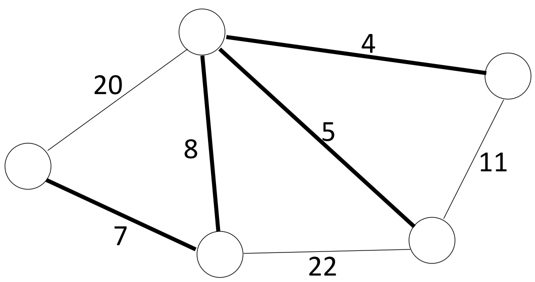
2. *(6 points)* Now suppose each edge weight is increased by 1: the new weights w'(u,v) = w(u,v) + 1.

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

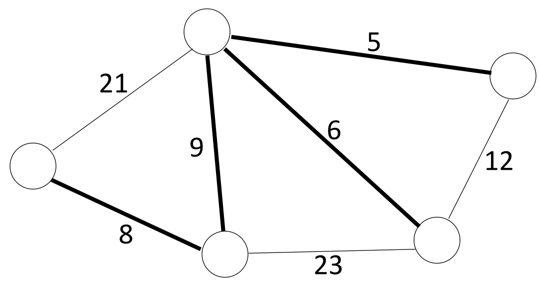
The minimum spanning tree would not change because each edge's weight is growing at the same rate. As a result, the graph will have the same structure and will not be different. If we wanted to prove this then it can be seen by using Prim or Kruskal's to find the MST. If these two algorithms are correct then they will follow the same path to get the same result and display the same structure.

Use Prims Algorithm to find the MST.

w(u,v)



w'(u,v)



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

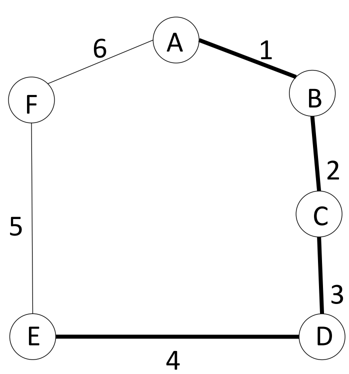
Yes the shortest path can change. For example, if you have a graph where one side has more edges then the other. If you look at the example below you will see that with w(A, E) the shortest path is A -> B -> C -> D -> E. Once we use w'(A, E), the shortest path changes to A -> F -> E.

Using Kruskal Algorithm to find the MST.

w(u,v)

A -> F -> E = 11

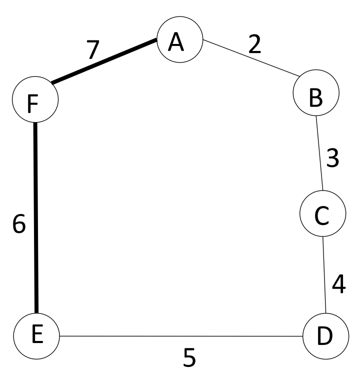
A -> B -> C -> D -> E = 10



w'(u,v)

A -> F -> E = 13

A -> B -> C -> D -> E = 14



Stephen Townsend
CS 325-400
11/5/17

3. *(4 points)* 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.

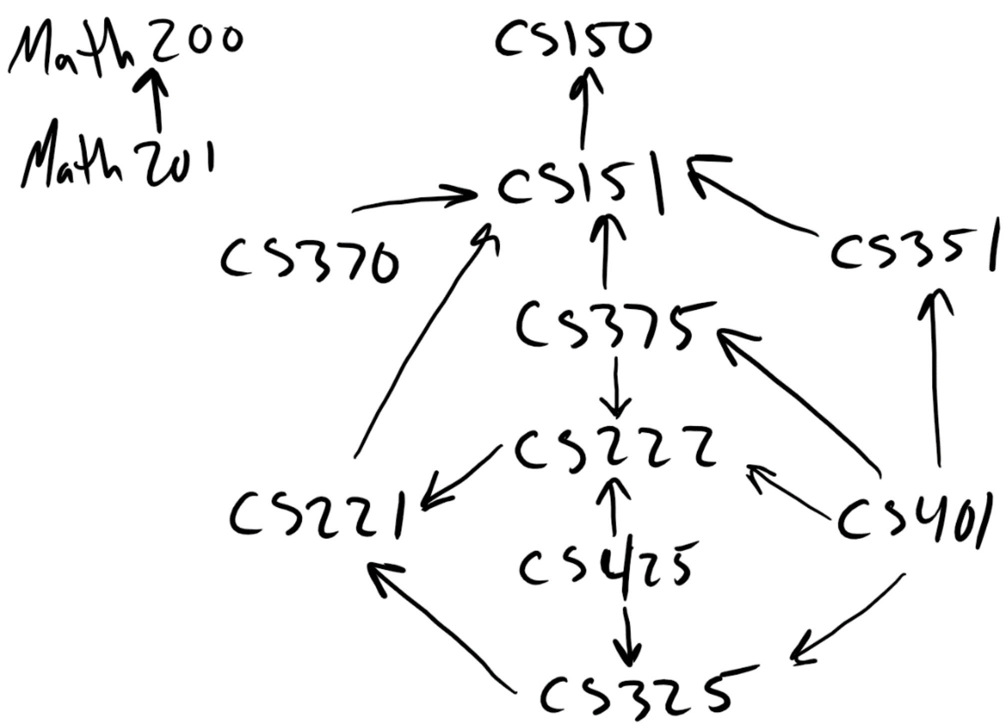
An efficient algorithim could to use a modified BFS implementation. Since the weight of the edges needs to be atleast that of weight W we could go ahead and ignore the edges whos weight is less then W. The algorighm would start at s and check the near by edges. If a edge has a weight less than W then it can disregard that edge as a possible path. The process will repeat untill we run out of edges or when it finds its way way to t.

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

The running time will be based on the algorithm transversing the graph. As a result, with a typical BFS algoorithim, it should run in Θ(V+E) time. This means that the running time is dependant on the amount of V and V-1 edges.

4. *(5 points)* List of courses and prerequisites for a factious CS degree.

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



Stephen Townsend
CS 325-400
11/5/17

(b) Give a topological sort of the graph.

Math 200
Math 201
CS 150
CS 151
CS 221
CS 222
CS 325
CS 425
CS 351
CS 370
CS 375
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.

- Quarter One
    - Math 200
    - CS 150
- Quarter Two
    - Math 201
    - CS 151
- Quarter Three
    - CS 221
    - CS 351
    - CS 370
- Quarter Four
    - CS 222
    - CS 325
    - CS 375
- Quarter Five
    - CS 375
    - CS 425
- Quarter Six
    - 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 I found in reverse order is:

CS 401
CS 375
CS 351
CS 325
CS 222
CS 221
CS 151
CS 150

To do this I worked backwards from the class with the highest prerequisites. I wrote down the prerequisites and check what their prerequisites were. This allowed me to gather a path of classes that needed to be taken and giving what I believe to be the longest path. The length is 7. This is the minimum number of classes needed to complete the CS degree with the given prerequisites.

5. *(12 points)* Babyfaces vs Heels

(a) Pseudocode

```
Function BFS(verticies, edges)
for i=0; i < # of verticies
        arr[0][i] = verticies
        if(arr[0][i] = verticies)
                arr[j+1][i] = edge;              //Place edge in row under the verticy
                arr[i+1][j] = vert;              //Place returning edge under rival
Check(arr[][])

Function Check(arr[][])
BabyfaceArr[]
HeelsArr[]
for j = 0; j < # of verticies
        if (arr[j + 1][i] !empty
                heels[j] = arr[j + 1][i];
        else
                babyface[j] = arr[j + 1][i];
```

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

Since I am looking to use the BFS algorithm, the running time should be equivalent to Θ(V + E).