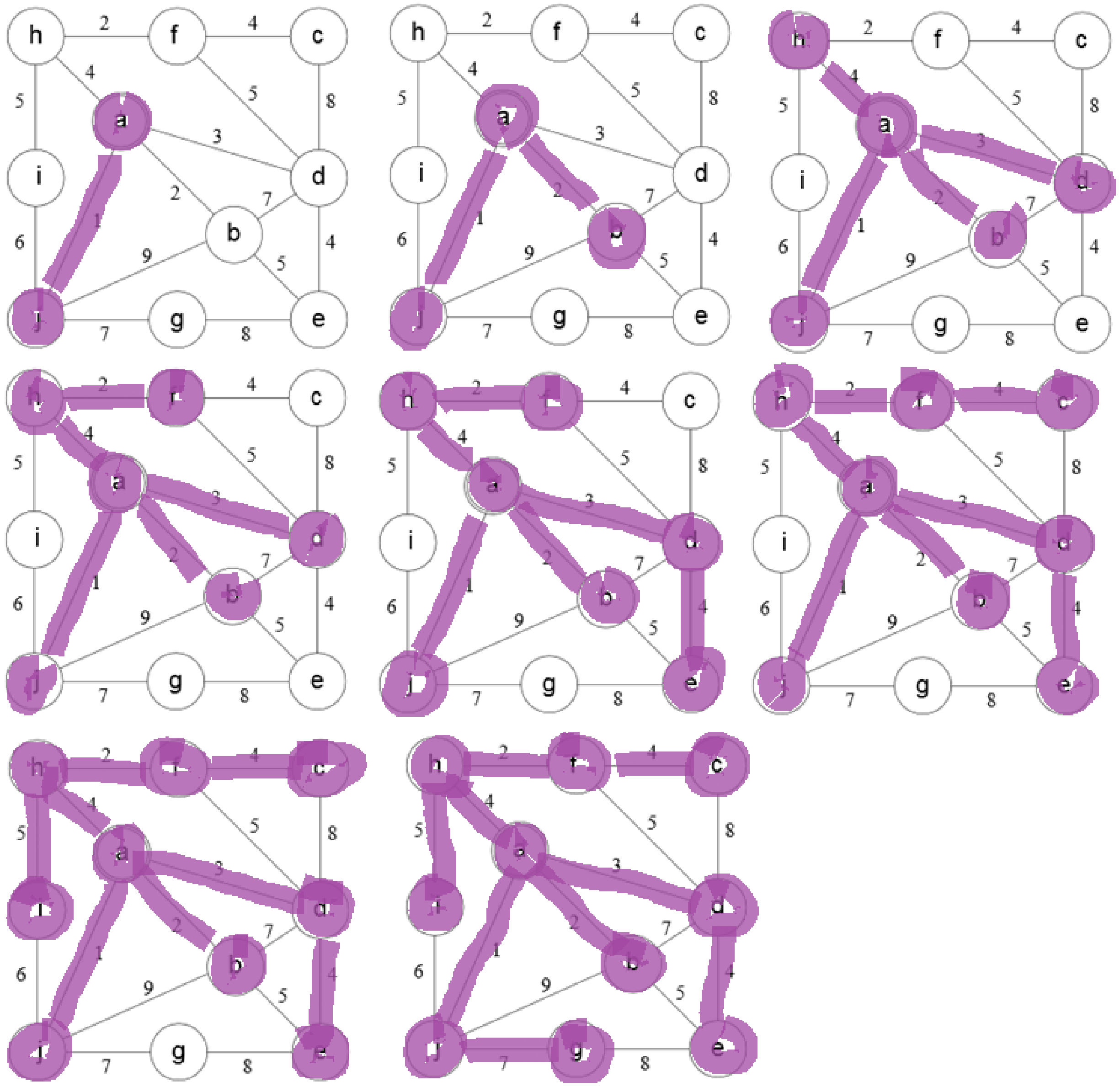
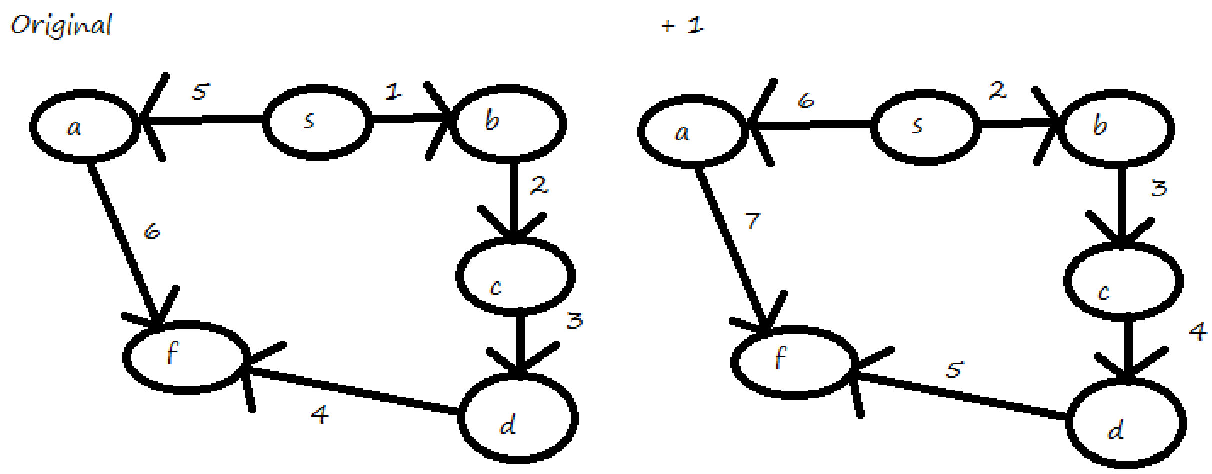
Shannon Jeffers
CS 325
Homework 5

1. The weight of the following MTS would be 32.



2. A. No, the minimum spanning tee would not change. Since every single edge weight in the entire graph gets increased by exactly 1, the relationships between each set of vertices would remain the same. Essentially nothing is changed when it comes to picking the “smallest” edge weight. If we were to choose say Kruskal’s algorithm, which aims to pick the smallest edge weight that does not cause a cycle, it would still pick the same edges in the new graph because all of the edges will have been increased by 1 causing the smallest to…. Remain the smallest and since we know Kruskal’s algorithm is proven to be correct, the MST would still be correct.

B. Yes, in this case there could be a change. If one path contained many small weighted segments to be the shortest path and another path contained a fewer number of heavier segments, adding 1 to every segment in the small weighted segments can easily become longer than the shorter heavier weighted segments. Take a look at the following example of the paths from s to f in the graph

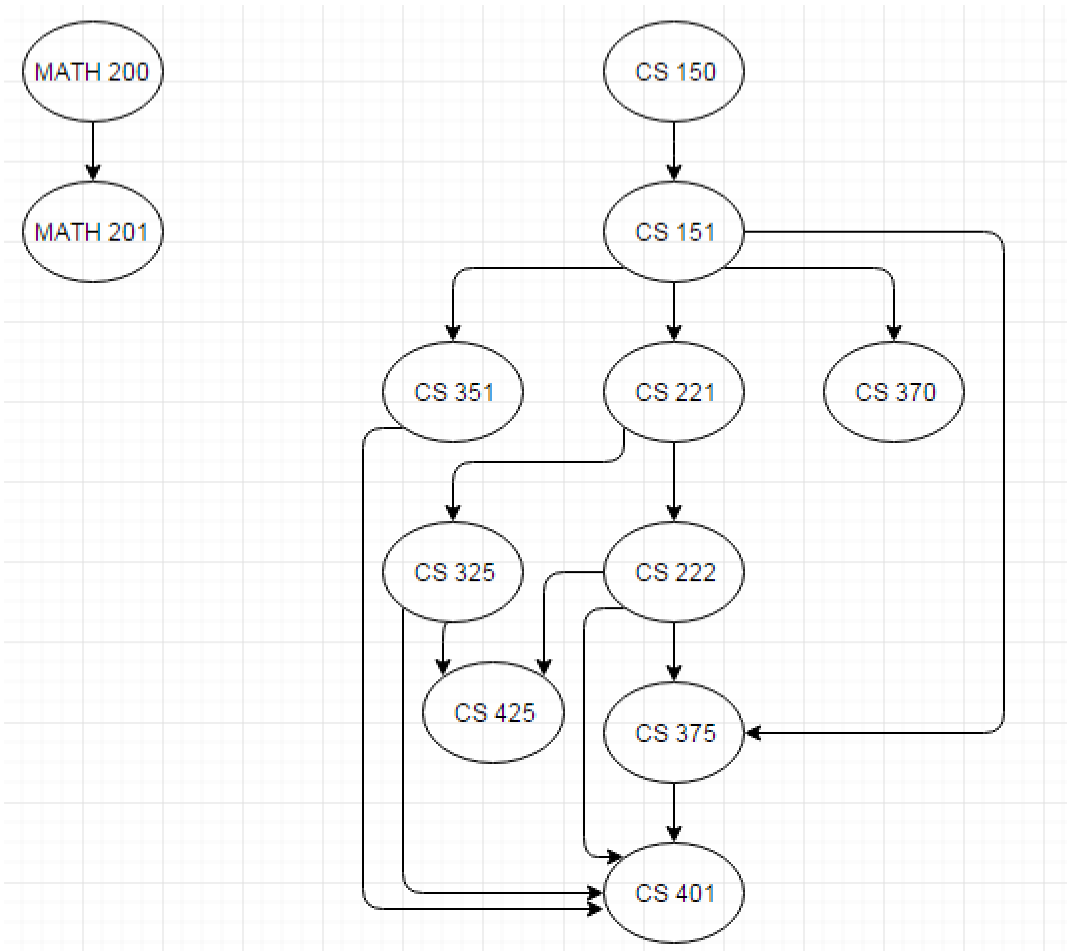


As you can see in the original the left path takes 11 and the right path takes 10, however in the +1 graph the left path takes 13 while the right path takes 14.

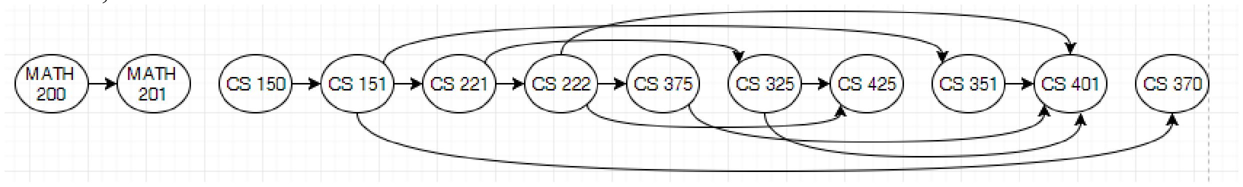
3. A. An efficient algorithm to solve the bottleneck path problem would be to use a modified version of Breadth First Search. This modified version would act as normal except it would ignore edges with weights that were not at least W. In other words It would start by adding the beginning vertex to a queue and then it would pop this vertex off and add it to our solution path. It would then examine the neighbors of our starting vertex. If the weight of the edge between it and the neighbor was at least W, the neighbor would be added to the queue. It would continue in this manner popping the top of the queue and adding to the queue until the queue was empty or the vertex we wished to reach was reached.

B. The running time would not differ from that of traditional BFS, which is O(V+E). It would not differ because the check is a simple conditional that is done in constant time so it does not significantly affect run time.

4. A.



B. MATH 200, MATH 201, CS 150, CS 151, CS 221, CS 222, CS 375, CS 425, CS 351, CS 301, CS 370



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

D. The longest path in the DAG is 5. It consists of 6 verticies and 5 edges and can clearly be seen when looking at my DAG diagram. This is the path that goes from 150, 151, 221, 222, 375 and finally 401. This represents the number of terms it would take in order to complete all the classes -1. That is to say the number of terms = longest path + 1 due to the fact that each edge has 2 vertices.

5. A. This algorithm would be a modified version of the breadth first search. Instead of

changing each color from white to gray to black we would only assign a color once and make sure no connecting nodes have the same color (or instead of color assign a team name.

```
isValid(graph, source):
    Let Q be a new queue
    add source to Q
    while Q is not empty
        u = queue.pop
        indicate that u has been encountered
        if u does not have a team
            u.team = babyface
        for rival in u.rivals
            if rival doesnt have a team
                assign rival opposite team from u
            if rival does have a team and it is the same as u's team
                return false
            if rival hasn't been encountered
                add to Q
    if we made it through all without returning false, return true.
```

B. The running time of this algorithm above would be the same as the running time for BFS, which is O(V+E). This is because every vertex will be added and removed from the queue exactly once, and each adjacency list is explored once. if a graph is disconnected this algorithm may need to be called more than once, which may increase the overall run time of the program, but the algorithm itself would be O(V+E). In the implementation of this algorithm we also need to handle building the graph which would have a run time of O(n) because it uses several disconnected for loops to build the graph up and then set up the teams and checked lists. The above algorithm would be called in a loop to account for any disconnected graphs, which would make the overall run time of the implementation O(V^2 + EV) because the loop that calls the algorithm would run at most V times in the case where every node is disconnected.