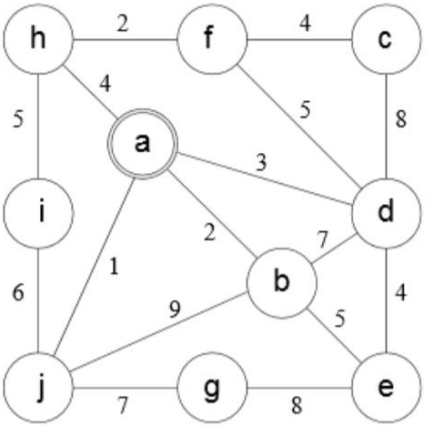
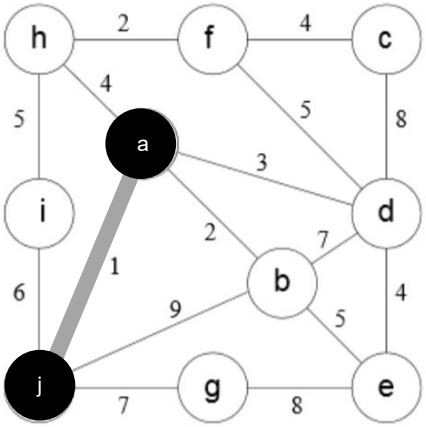
Xiaoying Li
CS 325 -Winter 2020
Homework #5

- **Problem 1**

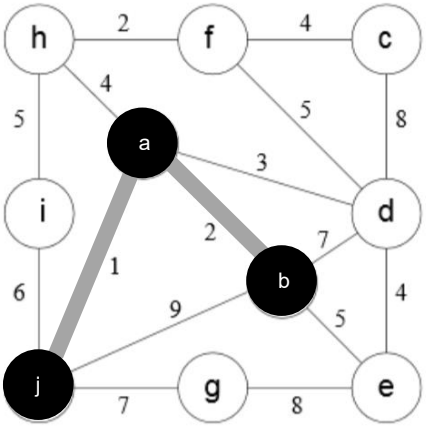
(a)



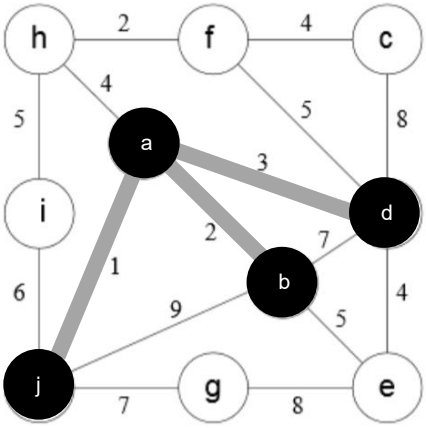
(b)



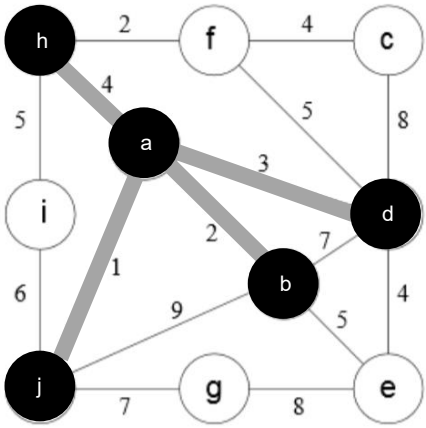
(c)



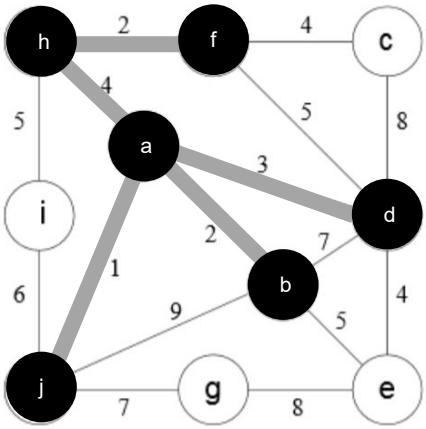
(d)



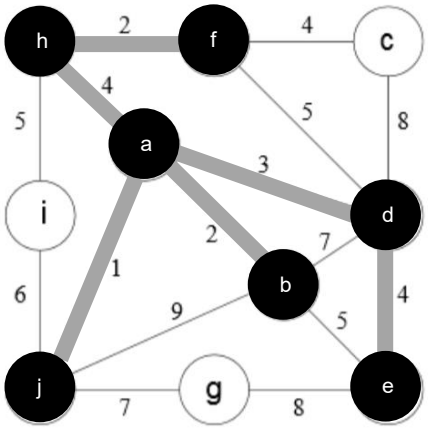
(e)



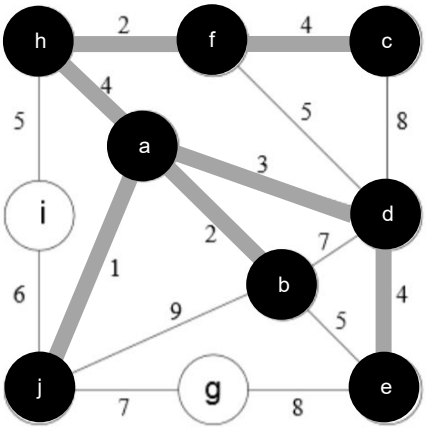
(f)



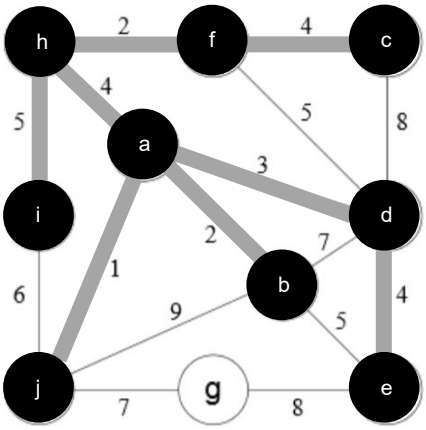
(g)



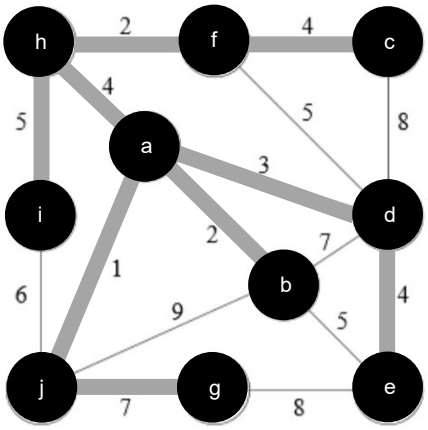
(h)



(i)



(j)



The weight of the minimum spanning tree is 1+2+3+4+2+4+4+5+7=32.

- **Problem 2**

a.

No. The minimum spanning tree doesn't change.

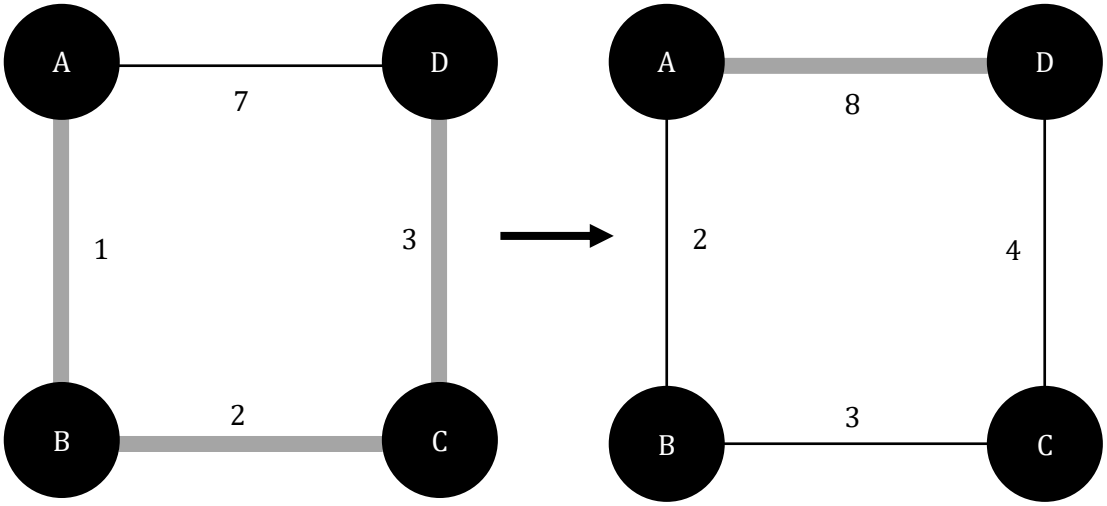
Prove it by running the Prim's Algorithm. At every iteration of Prim's Algorithm, it grows the minimum spanning tree by one edge with the minimum weight that connect the minimum spanning tree to vertices not yet in the minimum spanning tree. And increasing each edge weight by 1 won't change the sorting order of all edge's weights. So, the edge added at every iteration won't change, which means the optimal solution of every subproblem won't change. Therefore, the final optimal solution – the minimum spanning tree won't change.

b.

Yes. The shortest path changes.

Below is an example:

The shortest path from A to D of the left graph, which is the original graph is A-B-C-D=1+2+3=6. After increasing each edge weight is by 1, the shortest path from A to D of the right graph is A-D=8, since now A-B-C-D=2+3+4=9>8. Obviously, the shortest path changes.



- **Problem 3**

a.

Use the Breadth-First Search Algorithm to find a path from s to t, and ignore any edges of weight less than W.

```
BottleneckPath (G, s, t) {
    for each vertex u∈G.V {
        u.color = WHITE;
    }
    Q = {s};
    while (Q not empty) {
        u = DEQUEUE(Q);
        for each v∈G.Adj[u] {
            // Ignore any edges of weight less than W.
            if (v.color == WHITE and w(u, v) >= W) {
                if (v == t) { // Stop when t is reached.
                    return G.V.p;
                }
                v.color = GREY;
                v.p = u;
                ENQUEUE (Q, v);
            }
        }
        u.color = BLACK;
```

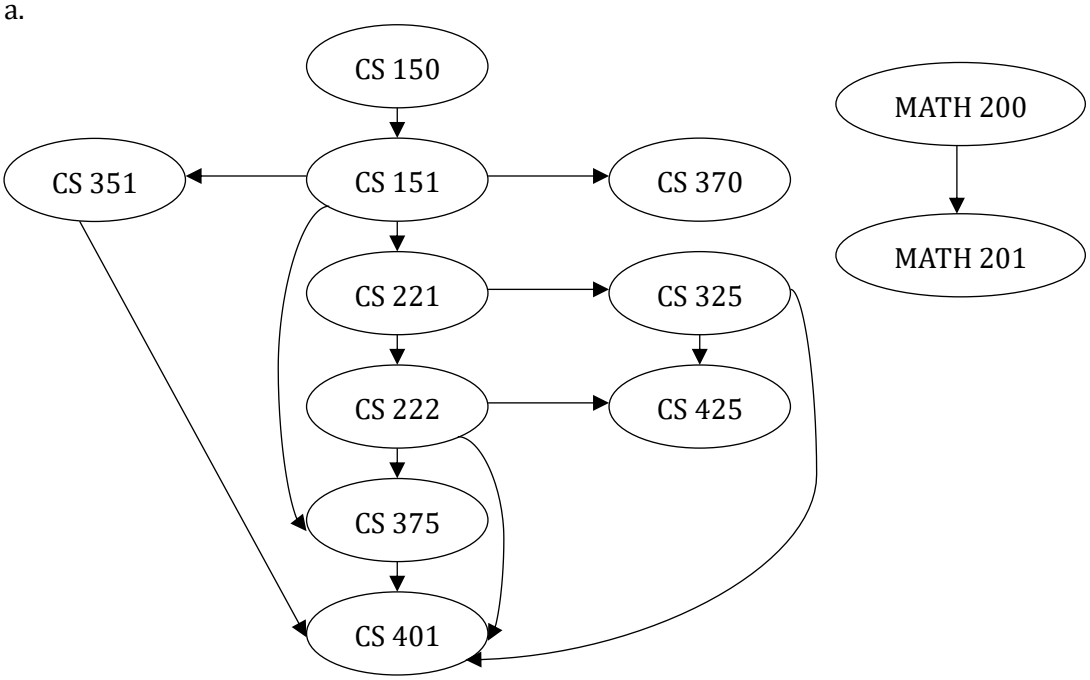
```
    }
}
```

(b)

The running time of the algorithm is same as the Breadth-First Search Algorithm's running time, which is O(V+E).

- **Problem 4**

a.



b. Shown in Picture 4-b on next page.

c.

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

d.

The length of the longest path in the DAG is 5.

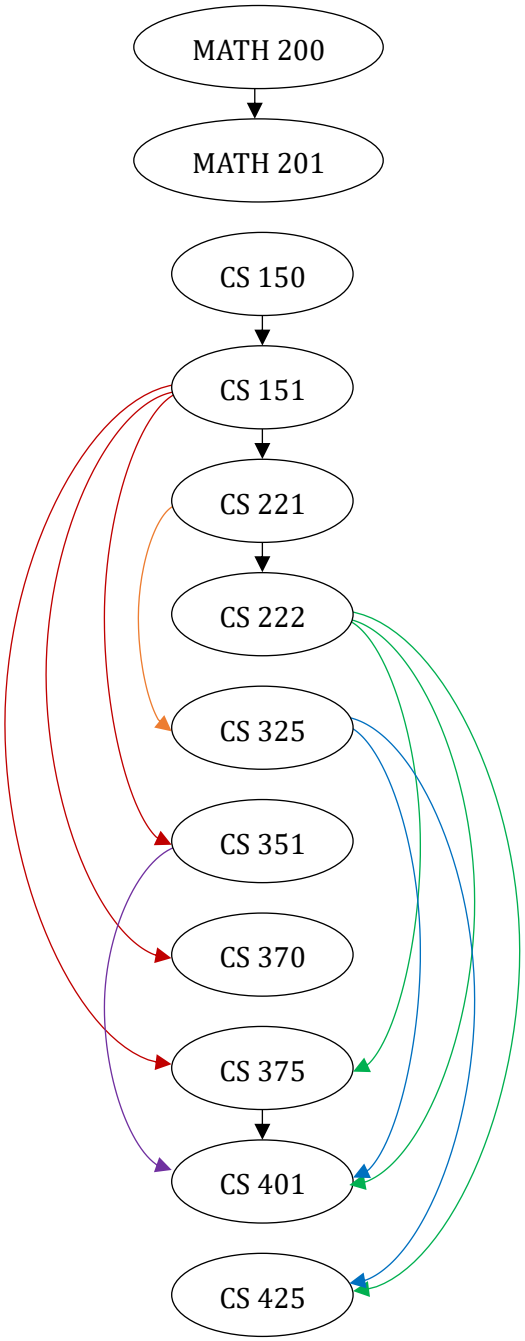
CS 150 → CS 151 → CS 221 → CS 222 → CS 375 → CS 401.

Use the topologically sort result in part b, initialize the distances associated with

vertices in the graph to 0, which is d(v)=0 for all v's in the graph. Then start with the first vertex in the topologically sort, for every its adjacent vertex u, set the distance to max {d(u), d(v)+1}. Repeat this step with the next vertex in the topological sorted order until all vertices have been examined. And the biggest distance is the length of the longest path in the DAG.

The length of the longest path+1 = 5+1 = 6 represented the minimum number of terms required to complete all courses with the given prerequisites.

Picture 4-b:



- **Problem 5**

a.

1) Construct a structure of wrestler, every wrestler has name, type and a bool variable of its visit status. And all wrestlers' types are initialized to be "unknown", and the visit statuses to be "false".
2) Construct a graph of wrestlers using adjacency list, every wrestler is a vertex, and every wrestler's adjacency list is its rivals.
3) Set the first wrestler's type to be "Babyface", and its visit status to be "true". And set all its adjacencies' type to be "Heel", push them to a queue used to perform BFS algorithm.
4) Perform BFS algorithm to traverse all wrestlers in the graph:

   Start from the first wrestler in the queue, first set its visit status as "false";

   If any of its adjacency's type is "unknown", set this adjacency's type to be the other one between "Babyface" and "Heel", and push it to the BFS queue;

   If any of its adjacency's type is same as its type, which means there are rivalries between the same type of wrestlers, return impossible.
5) If the graph is not connected, the BFS algorithm can only traverse the first connected component, so after step 4, check if all wrestlers' in the graph have been visited, if not, find the first unvisited wrestler, and repeat step 3, 4 and 5 until every wrestler's visit status is "true".
6) If all wrestlers have been traversed and no wrestler has adjacencies with the same type, return true.
7) Output the result.

b.

The algorithm takes $O(n+r)$ times to perform BFS algorithm, $O(n)$ times to designate every wrestler to be "Babyface" or "Heel", $O(r)$ times to check if there are rivalries between same type of wrestlers.

Therefore, the whole algorithm's running time is $O(n+r)$.