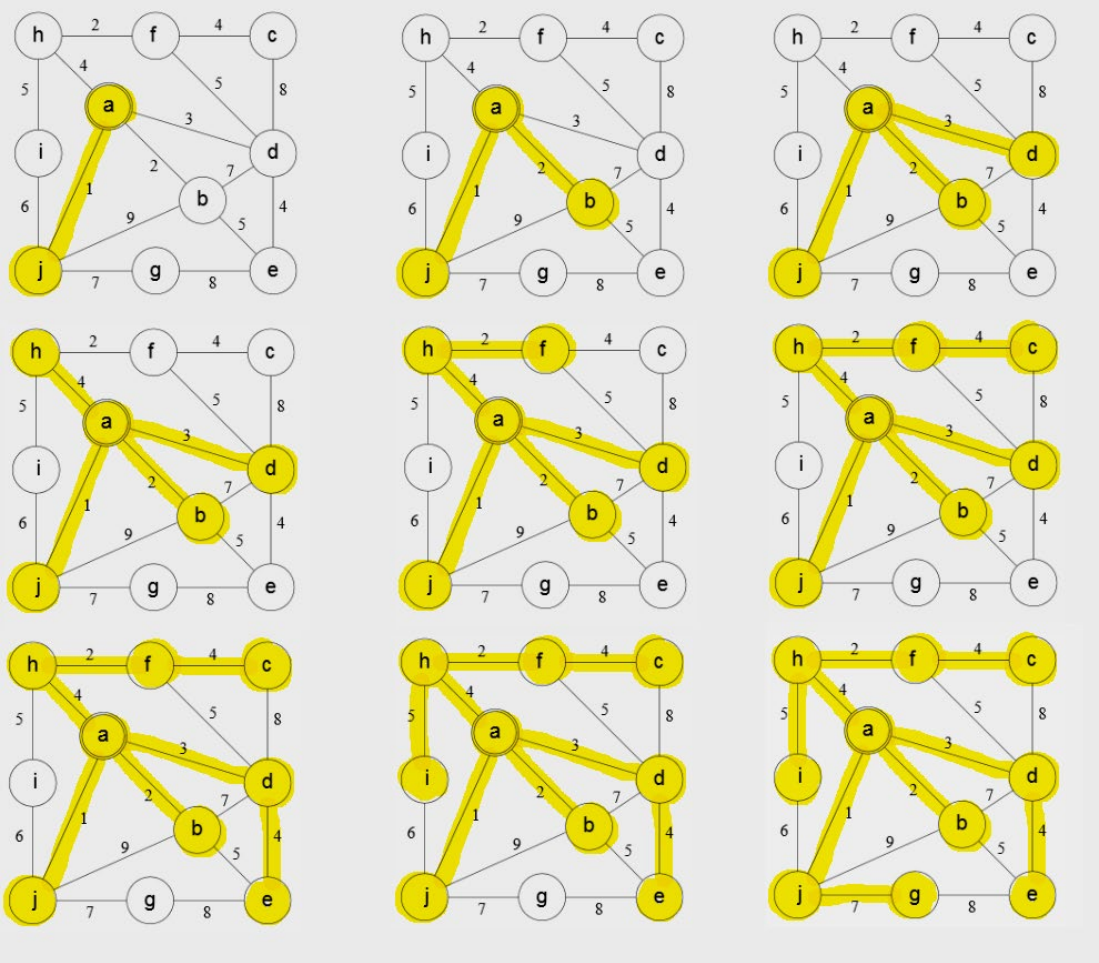
Sangyun Lee
Hw4

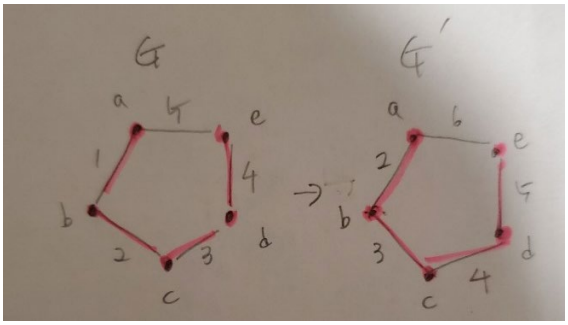
Q1.

Weight of MST = 2+4+4+5+1+2+3+4+7 = 32

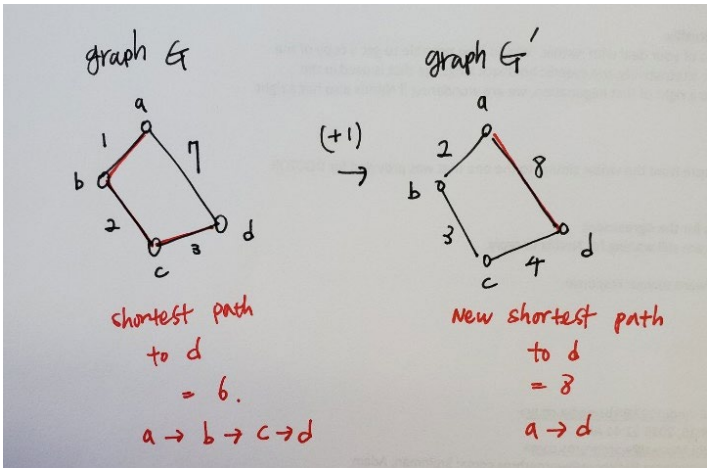


Q2.

a) MST will not be changed. Number edges required for spanning tree is |V|-1. So, adding weight doesn't change number of vertices, so G' still have the same number of vertices. And, since all edges' weight is increased by 1. So, ordering of edges is not changed, so MST will not be changed.



b) Shortest paths can be changed. Let say, there is a graph G with weighted edges as follow E(a,b) = 1, E(b,c) = 2, E(c,d) = 3, E(a,d) = 7. And we want to find the shortest path from a to d. Then current total weight of the shortest path is 6 which is a to b, b to c, c to d. However, there is a graph G' that all edges' weight is increased by 1. Then the weight of the shortest path of G' is 8 which is a to d. Since the total weight of previous shortest path was changed to 9 which is greater than 8.



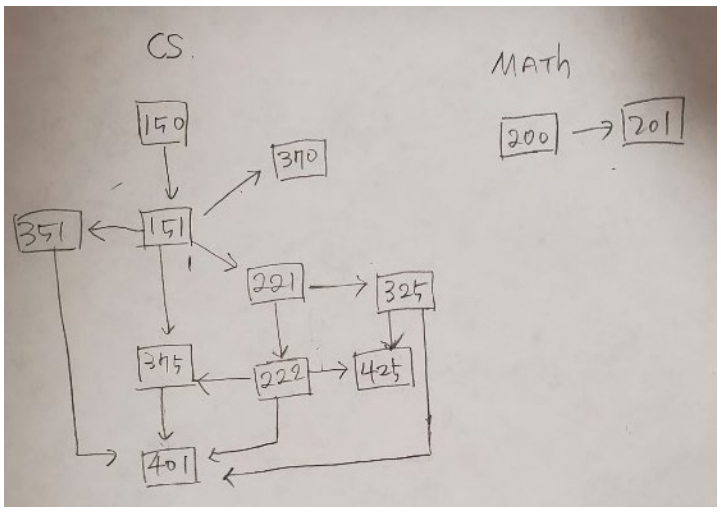
Q3.

a) Since it is about to find a path that consist of edges which is greater or equal to a particular weight W. We can use Breath First Search which is to find shortest path. And Adding a function to check the weights of edge. And if its weight is less than the particular W, then ignoring it. Check the next edge.

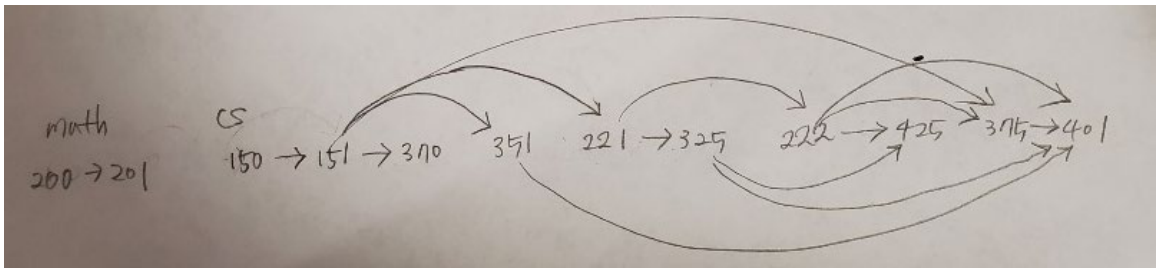
b) Check the weights and comparing with the give W take constant time. So the Running time will be same as BFS which is O(V+E)

Q4.

a)



b)



Adjacency List
CS150 -> CS151
CS151 -> CS370 -> CS221 -> CS351 -> CS375
CS221 -> CS222 -> CS325
CS222 -> CS375 -> CS425 -> CS401
CS325 -> CS425 ->CS401
CS351 -> CS401
CS370 ->
CS375 -> CS401
CS401 ->
CS425 ->
MATH200 -> MATH 201
MATH201 ->

c)

Term1 – cs150, math200
Term2 – cs151, math 201
Term3 – cs221, cs351, cs370
Term4 – cs222, cs325
Term5 – cs425, cs375
Term6 – cs401

d)

The longest path is 5 (cs150 -> cs151-> cs221 ->cs222 ->cs375 -> cs401). In order to find the longest path, I trace back from end vertex which is cs401 to cs150. And the length the longest path is 6 (5+1) this represent the number of terms you have to take to complete the courses.

Q5.

a) I have tried some different way to do this part. First, time, I used the DFS search with coloring and check the cross edges. I tweaked the cross-edge code to check the current vertex ‘u’ team and its adjacent team ‘v’. Because the edges represent the rival relationship. Vertex u team must be different from adjacent vertices team. If it is the same team, then it has a conflict. However, if test size is getting large, my code is somewhat unstable.

So I add bipartite graph check. Using the edges, I gave a color from left side, and then check if its opposite vertex has color or same color, if it doesn't have a color, then give a different color. If it has the same color, then it has conflict.

Bipartite Graph
For int I = 0, I < edges, i++
    Add edges
Color[0] <- Red
    For int I = 0, I < vertices, i++
    Check color vertex u, vertex v (directed vertex)
    If vertex v no color, then give a color
    Else if vertex u color == vertex v, then it is not bipartite graph.

If it is bipartite graph run DFS

color[u] ← GRAY
time ← time + 1
visited[u] ← time
for each vertex v adjacent to u
    if color[v] == Gray
        return true
    if color[v] == White && recursive_function()
        return true;
    if color[v] ← BLACK
        if team[u] == team[v]
            return true
        else team[u] == opponent team of team[v]
color[u] ← BLACK
time ← time + 1
if team[u]==Noteam
    if(time%2 == 0) then team[u] == "Beavers"
    else "Ducks"
processed[u] ← time
return false

So, if return true, it has either cycle or Cross edge(duplicated team)

b)

running time of checking bipartite graph is O(V*E) since it checks vertices and edges by looping program as much as the size of vertices and edges with outer and inner loop.

Allocating team name and comparing team is taking constant time, so it doesn't affect the overall running time. So, the running time of DFS is O(V+E);

c)

code will be submitted on TEACH