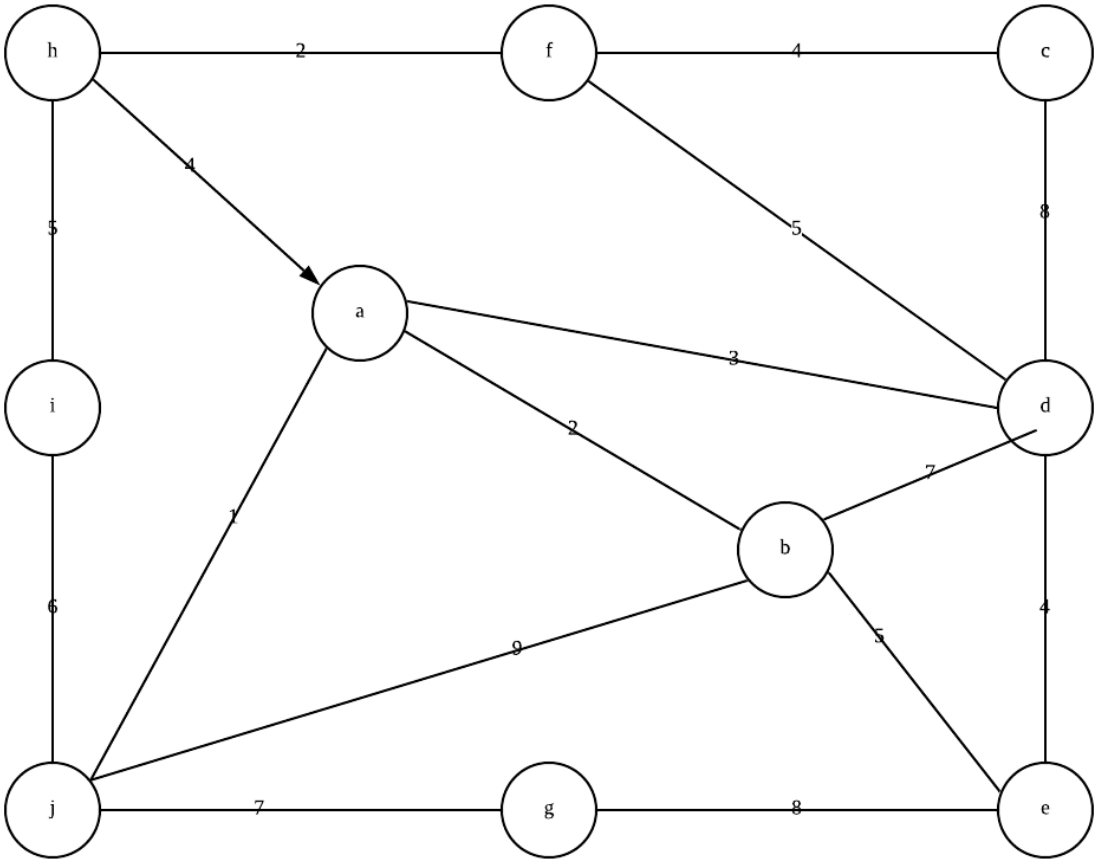
CS325 HW5

Yihong Liu

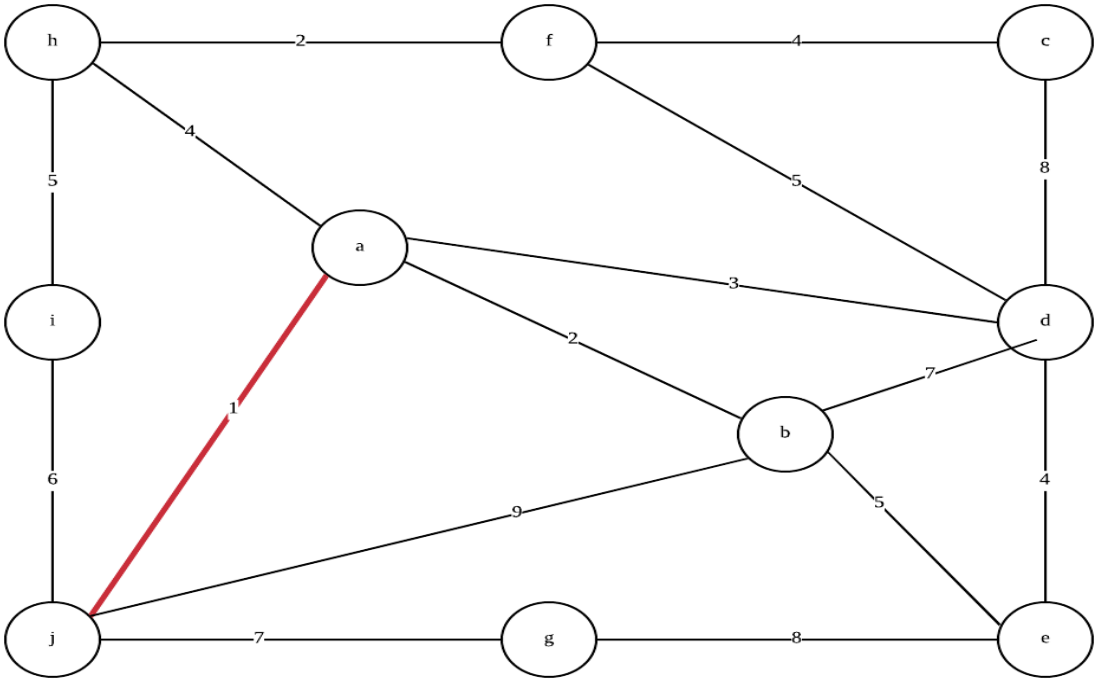
1.

The weight of the minimum spanning tree is 32 and I didn't change the colors for each vertex instead I write which vertexes are chosen already for each step. , here are the steps:

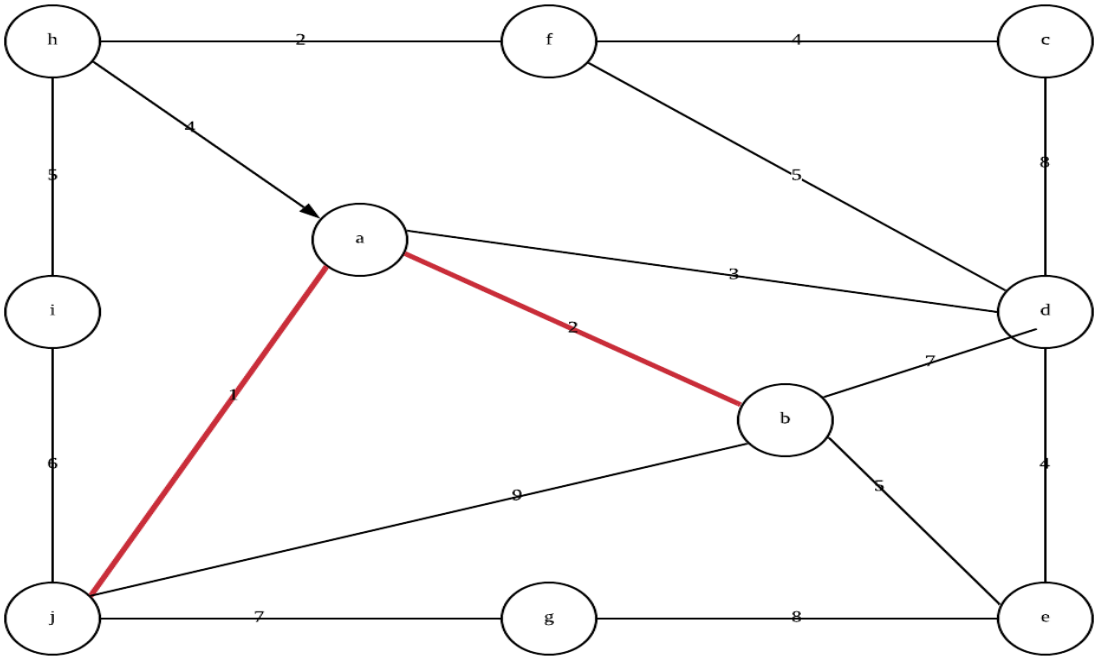
Step 0: no vertex chosen yet.



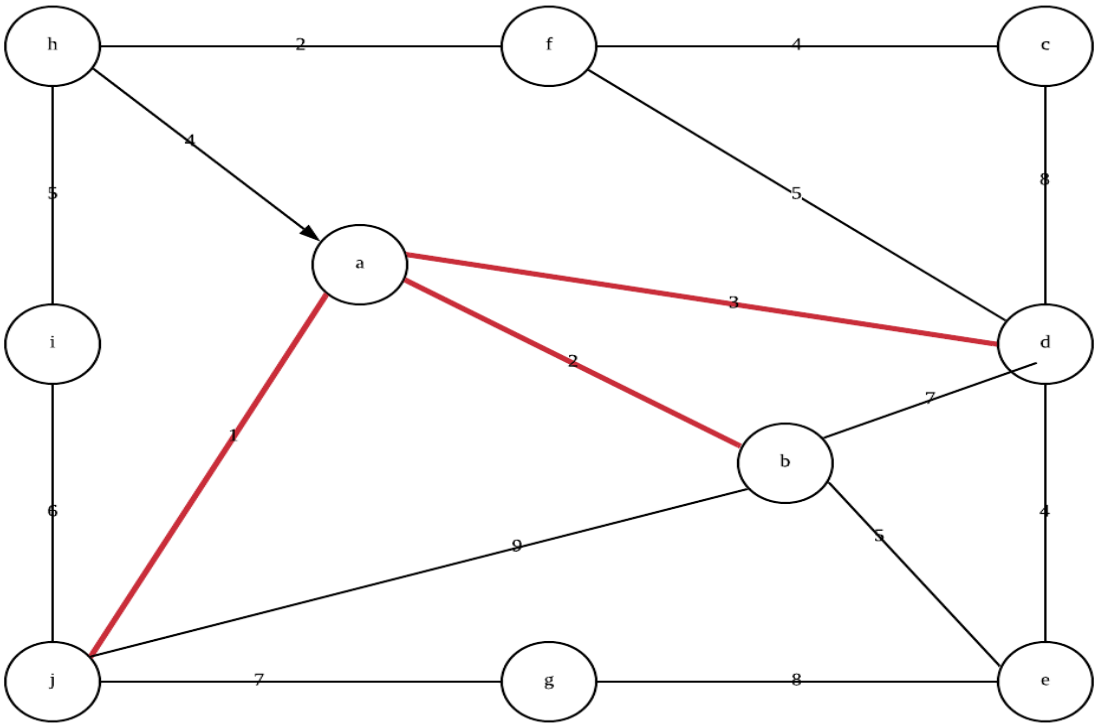
Step1: j , a selected



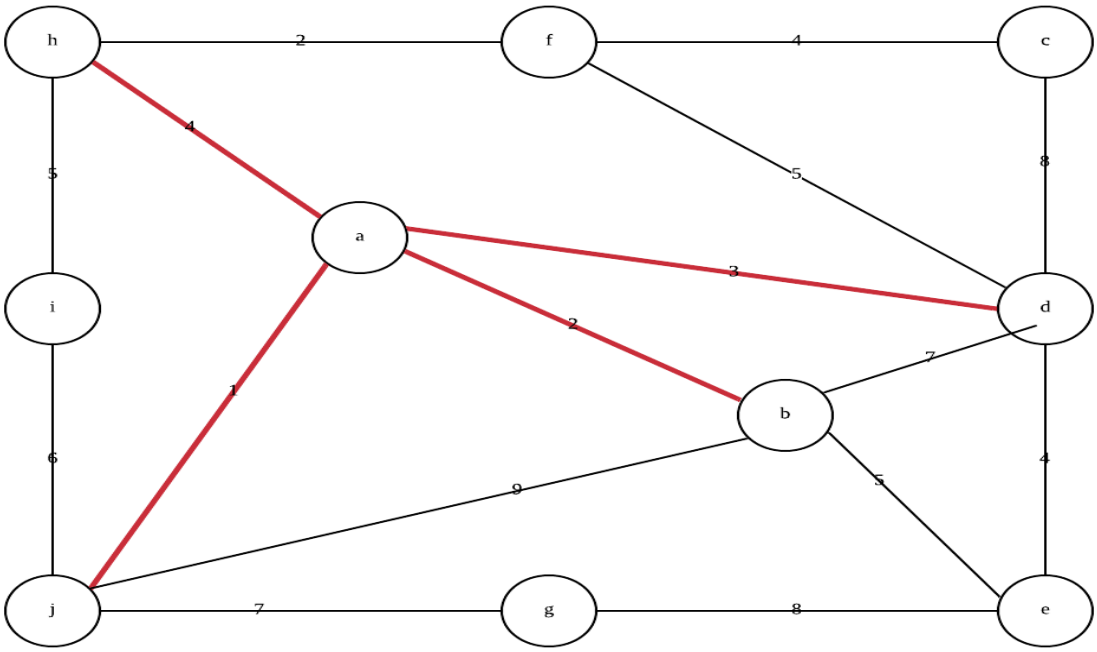
Step2: j , a , b selected



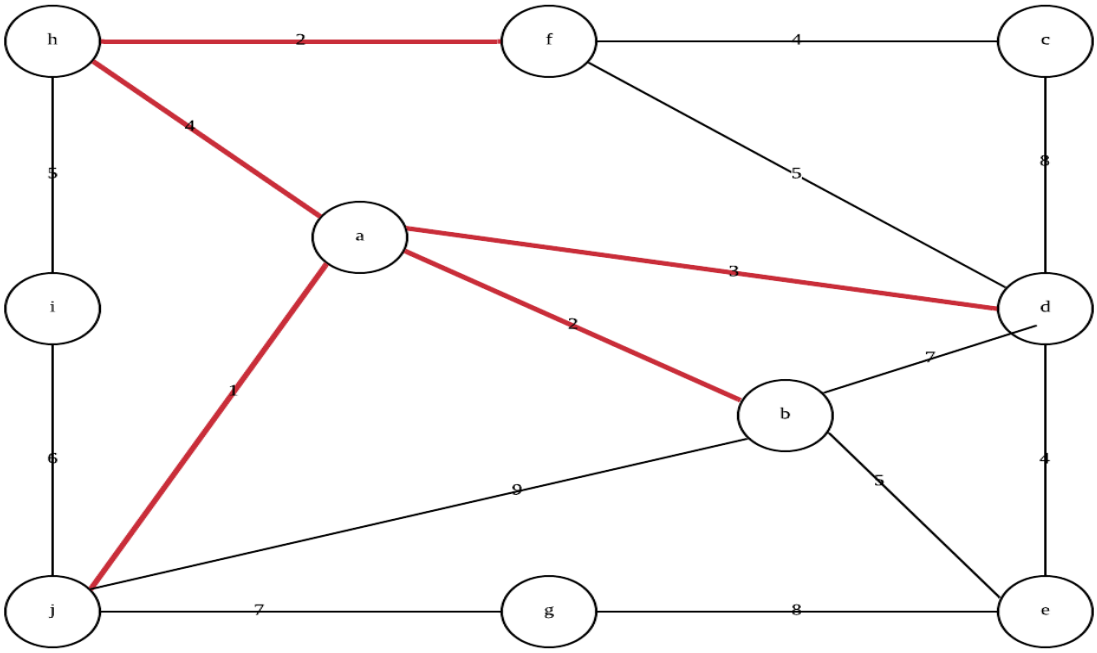
Step3: j, a, b, d selected



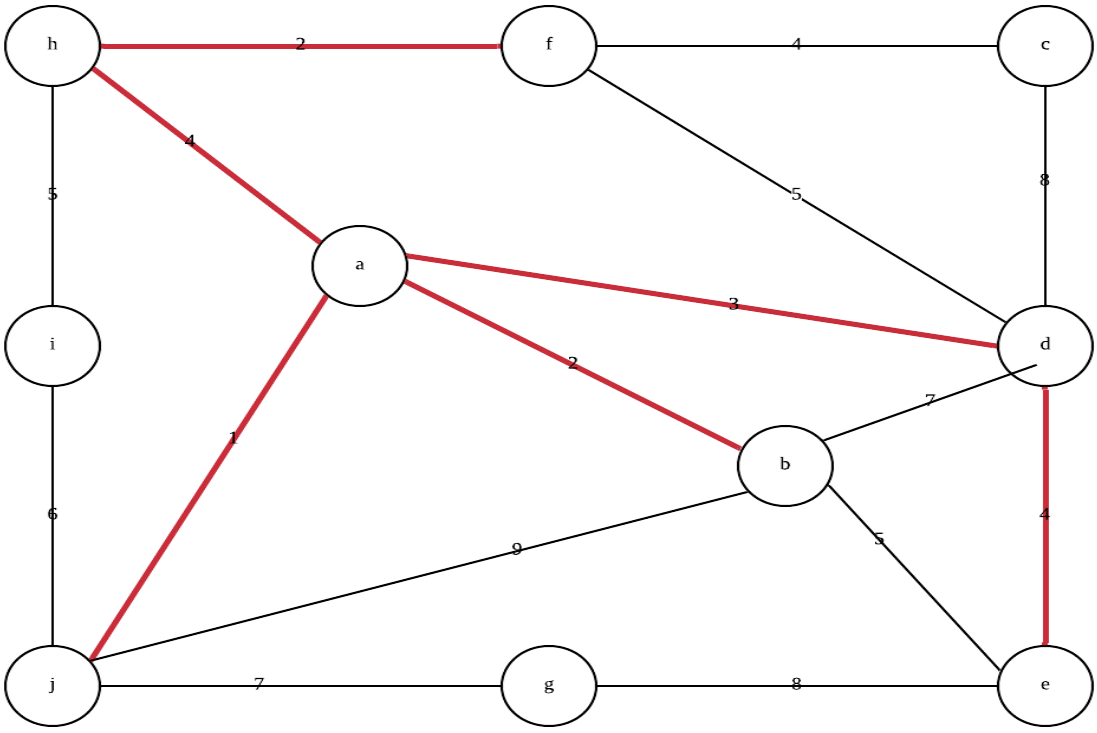
Step4: j, a, b, d, h selected



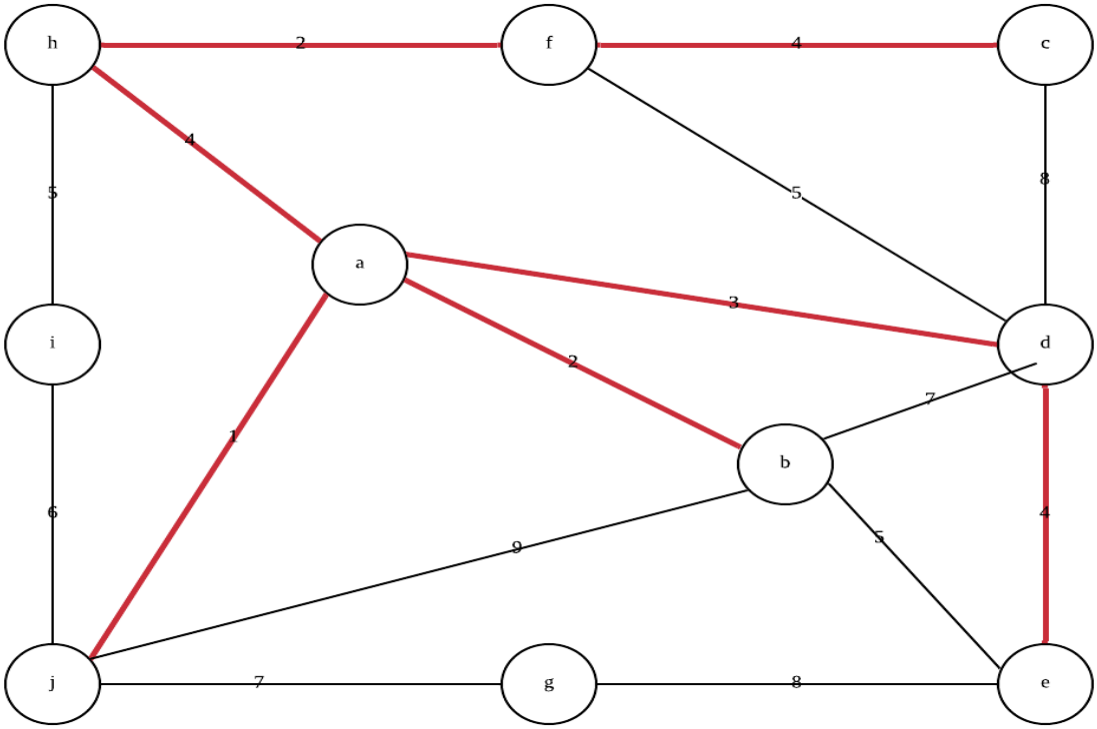
Step5: j, a, b, d, h, f selected



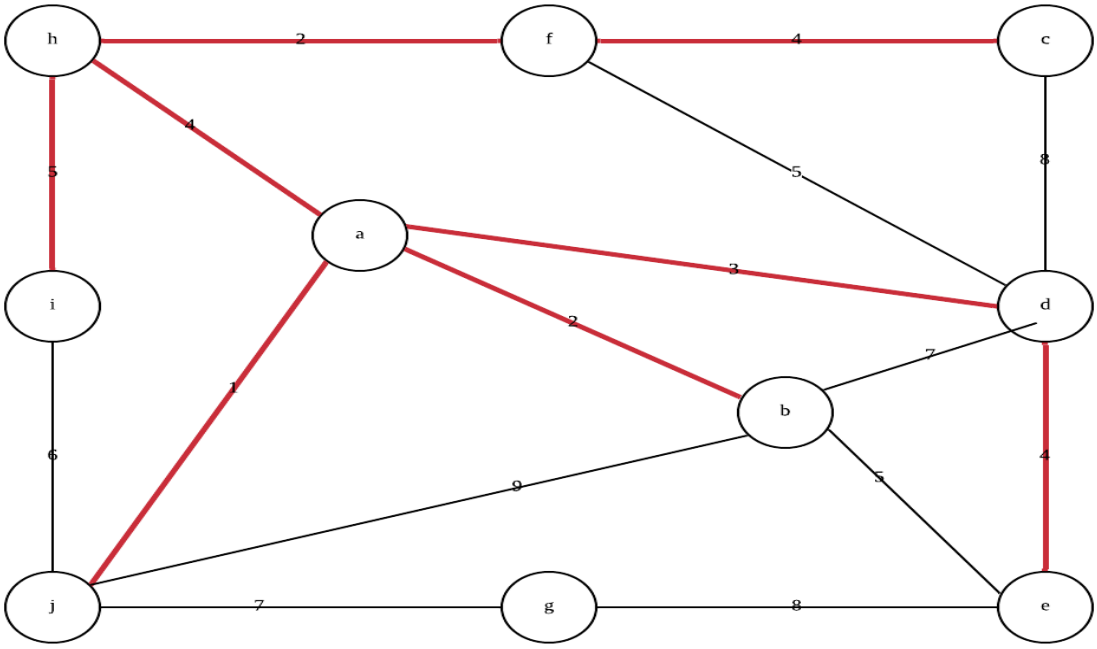
Step6: j, a, b, d, h, f, e selected



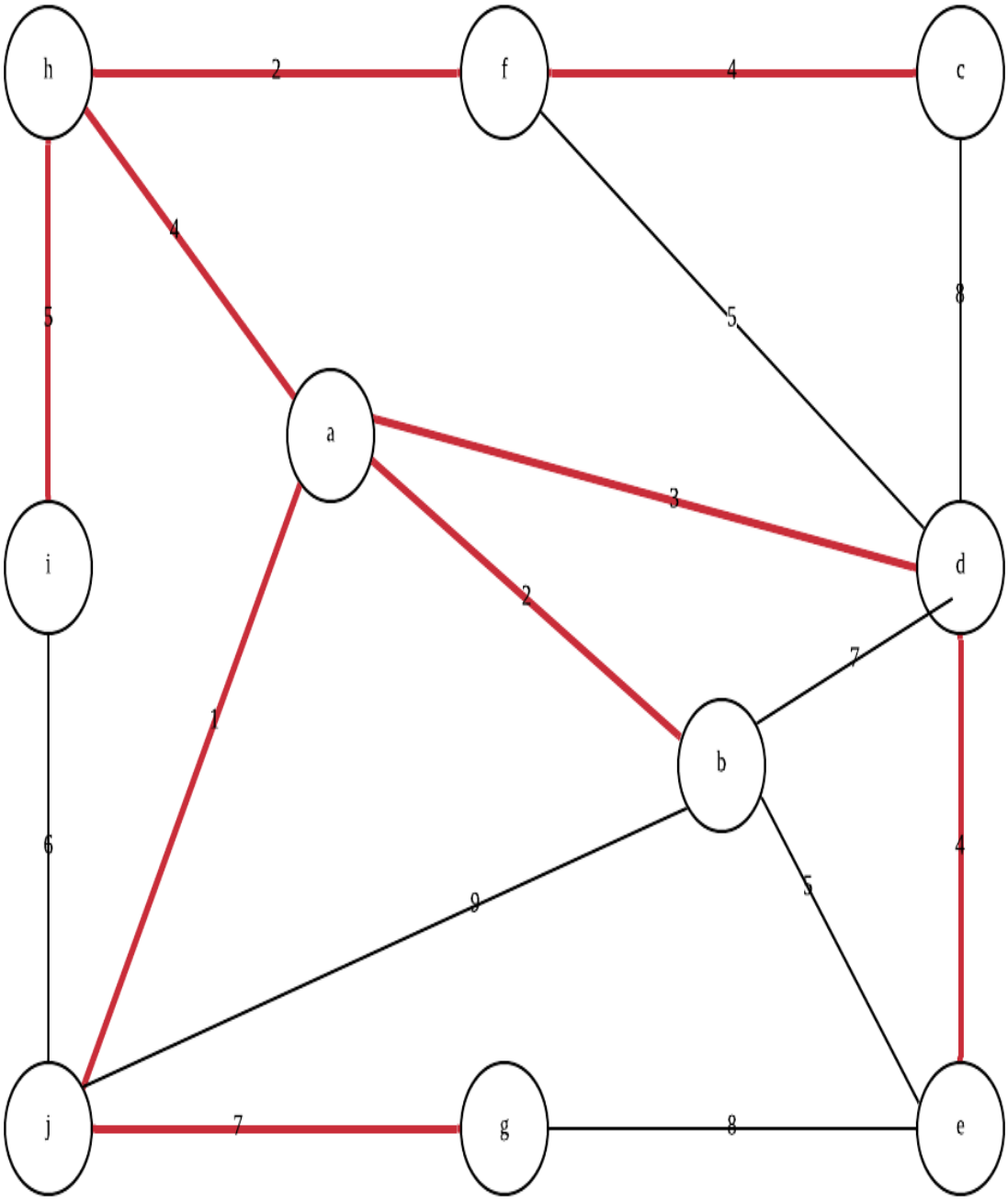
Step7: j, a, b, d, h, f, e, c selected



Step8: j, a, b, d, h, f, e, c, i selected



Step9: j, a, b, d, h, f, e, c, i, g selected, all selected, finished.

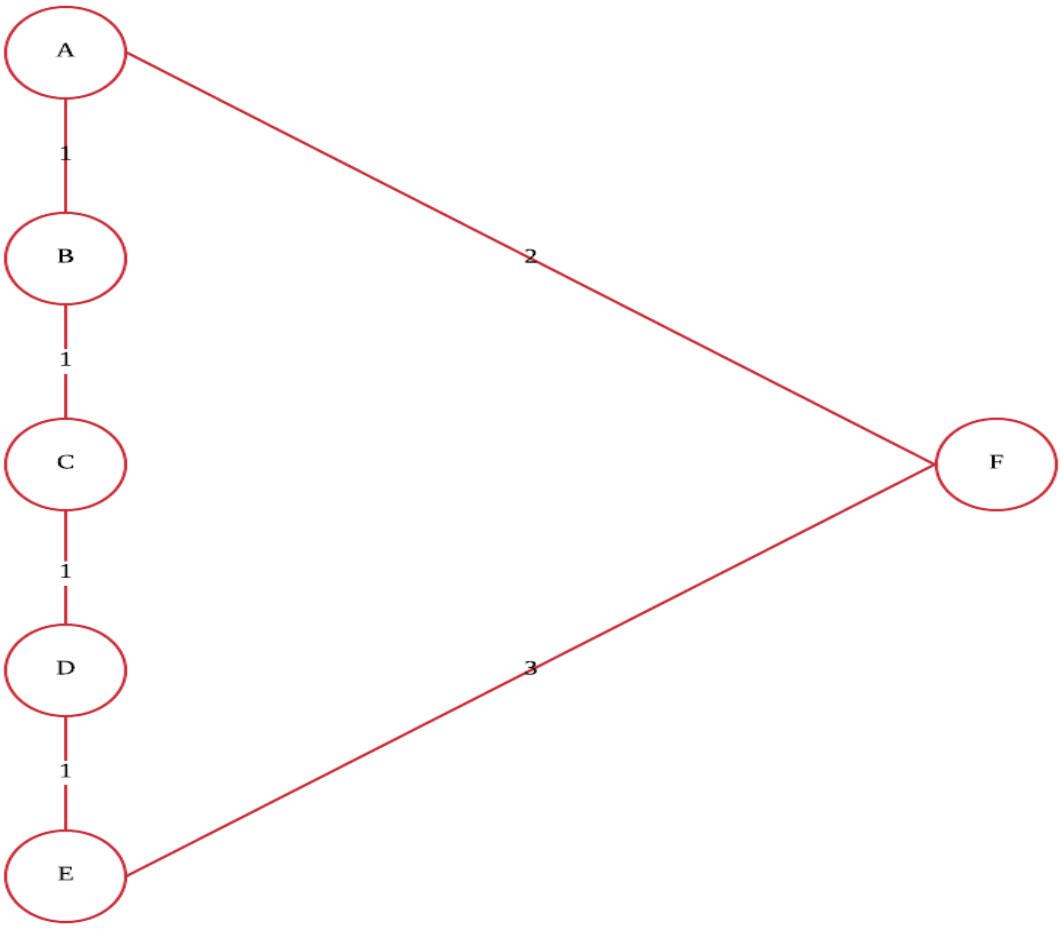


2.a

The minimum spanning tree will not be changed, to understand this is an easier way, that since all the edge's weight are +1, the edge's weight rank is still consistent with previous tree. For example, 1,2,3,4,5,6 change to 2,3,4,5,6,7. The rank is still the same, which means the edges that were picked in the new graph would still be the previous edges to make the minimum spanning tree.

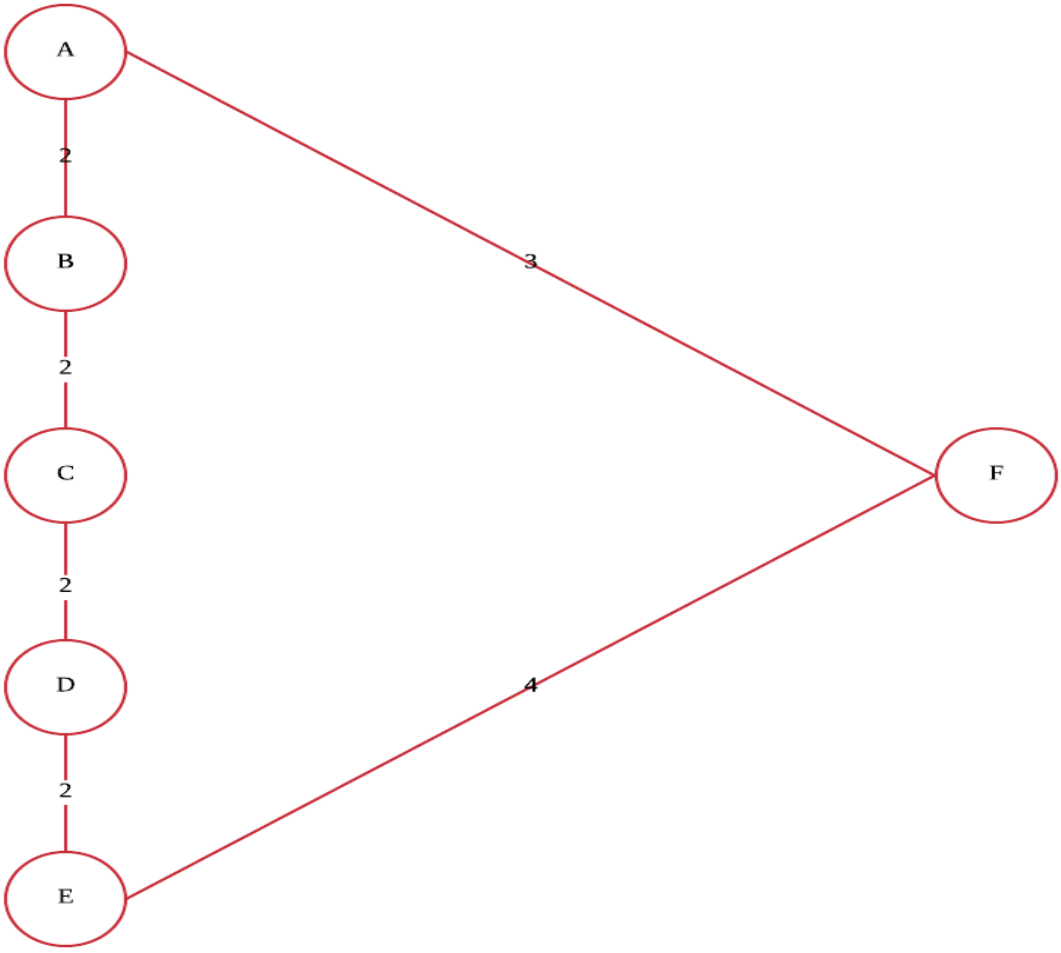
2.b

However, the shortest paths of the path may change in some cases. For example,



The shortest path from A to E now is from A-B-C-D-E 1+1+1+1 = 4.

However, if we add 1 to all the edges, the graph looks like this:



The shortest path now is changed to A – F – E instead of A-B-C-D-E.

Because A-F-E = 3+4 = 7 is smaller than A-B-C-D-E = 2+2+2+2 = 8.

3.a

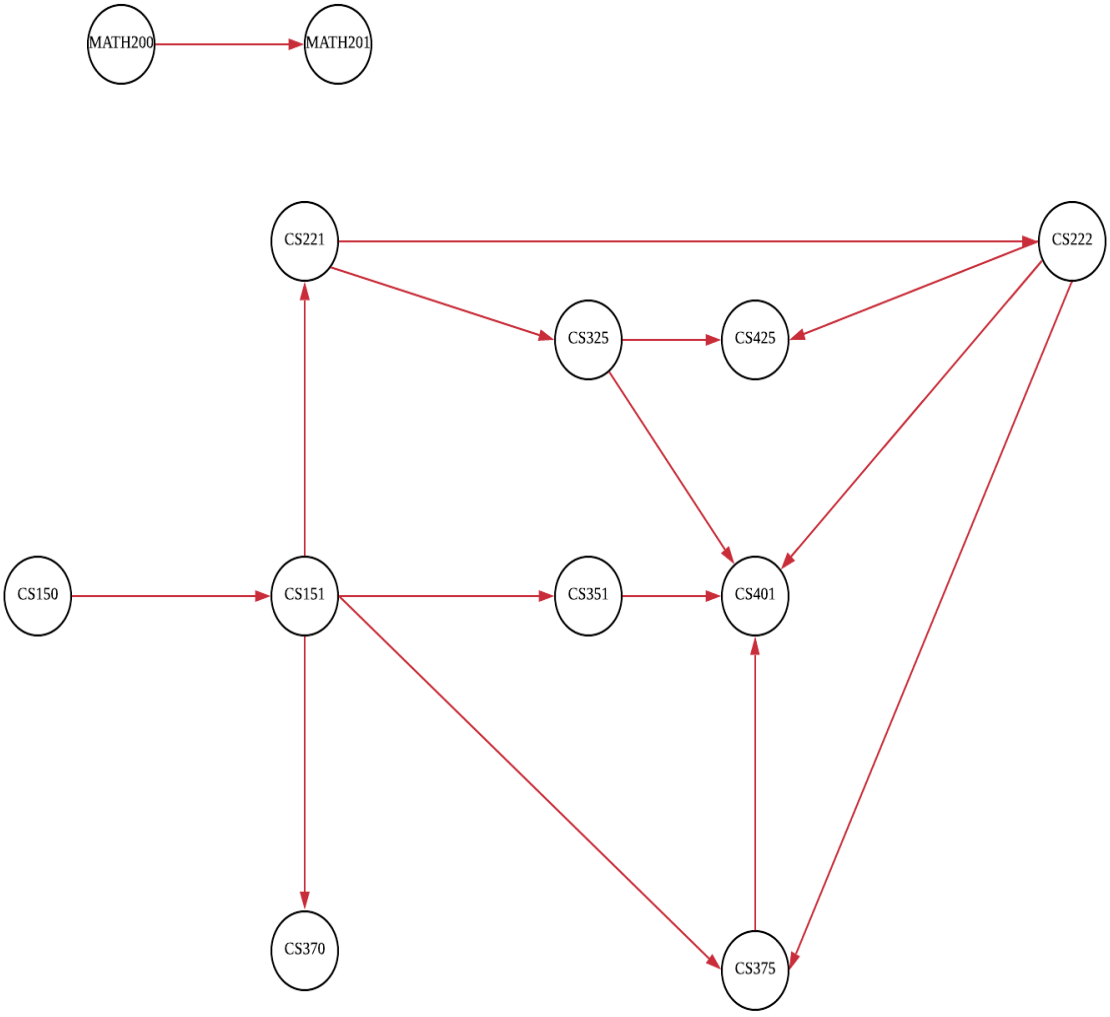
I can run a BFS algorithm, but ignore the path that has weight smaller than W.

3.b

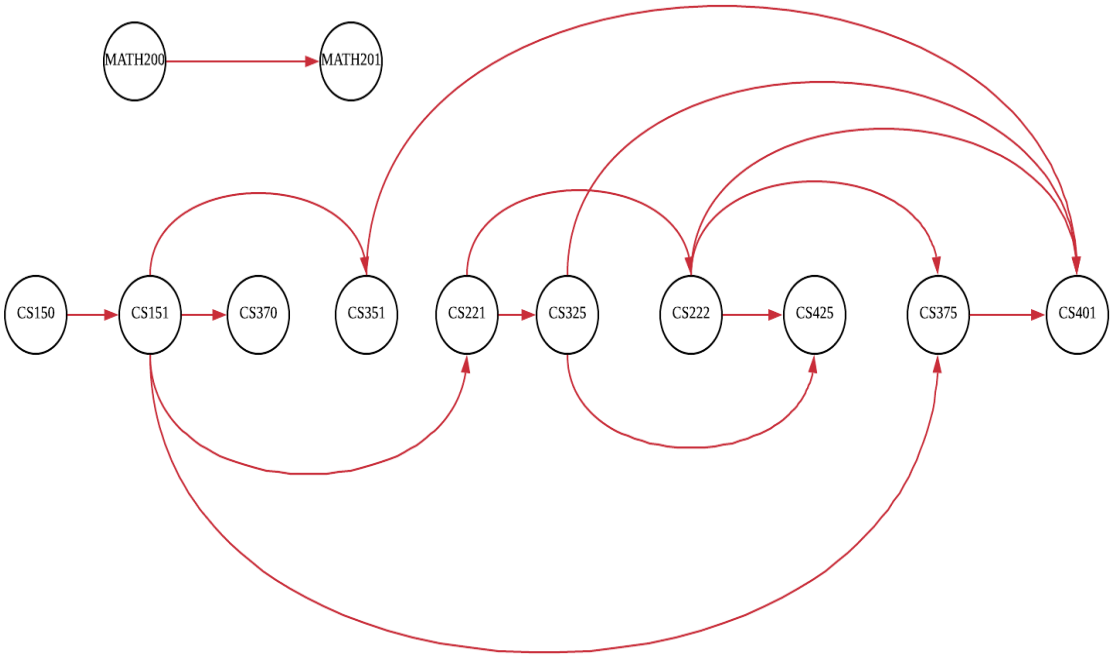
This will take O(V+E) time.

4.a

The directed acyclic graph:



4.b



The topological sort of the graph:

MATH 200 MATH 201 CS150 CS151 CS370 CS 351 CS221 CS325 CS222 CS425 CS375 CS 401

4.c

Term1: MATH 200, CS150

Term2: MATH 201, CS151

Term3: CS370,CS351, CS221

Term4: CS325, CS222

Term5: CS425, CS375

Term6: CS401

4.d

The length of the longest path in the DAG is CS150-CS151-CS221-CS222-CS375-CS401.

The length is 5, I found it based on the topological sorting algorithm, it represents that there are minimum number of terms required to complete all the courses, which is 6 terms at least.

5.a

Treat each wrestler as a vertex, each pair as edge between them.

Draw a graph for the input data first

Class graph:

```
    Def add_vertex()
    Def add_edge()
    Def bfs():
        # Mark all the vertices as not visited
    Color = ‘black’ #represent visited node
    Type = ‘Babyface’
    For v in neighbors:
        self.vertices[v].distance = vertex.distance + 1
        # the neighbor of 'Babyface' is 'Heel'
        self.vertices[v].type = 'Heel'
        q.append(v)

    visited = [False] * (len(self.graph))
    # Create a queue for BFS queue = []
    # Mark the source node as
    while len(queue) not empty do:
        s = queue.pop(0)
        color = ‘red’
        for i in self.graph[s]:
            if visited[i] == False:
            queue.append(i)
            visited[i] = True
    for v in neighbors:
for v in node_u.neighbors:
    node_v = self.vertices[v]
    # if the node is unvisited, enqueue the v to the queue
```

```
if node_v.color == 'black':
    q.append(v)
    # if the distance is bigger than previous node, increase distance by 1
    if node_v.distance > node_u.distance + 1:
        node_v.distance = node_u.distance + 1
    # sine the start node is Babyfaces 0, if the distance is even, the type of
    # that vertex should be Babyfaces
    if node_v.distance % 2 == 0:
        node_v.type = 'Babyface'
    else:
        node_v.type = 'Heel'
    # when u and v have the same type, just quit the program.
    if node_v.type == node_u.type:
        print("Impossible")
        quit()
```

Use the classic bsf algorithm but assign all wrestlers whose distance is even to be babyfaces and all wrestlers whose distance is odd to be heels.

Check if the type is the same

Create a vertex class to initialize the vertex stuff and add neighbor to them

Class vertex:

Def _init_ ()

Def add_neighbor()

5.b

The solution used BFS algorithm, so the time complexity is O(n+r)

5.c

See teach files.