Gurumanie Singh Dhiman (COMS3110 HW4)

(1)
```
BFS(G, v1, v2){
      v1.explored = true, v1.layer = 0, v1.numShortestPaths = 1,
      Q = new Queue
      Q.enqueue(v1)
      while(!Q.isEmpty){
            u = Q.dequeue()
            for each neighbor v of u {
                  if v.explored == false {
                        Q.enqueue(v)
                        v.layer = u.layer + 1
                        v.explored = true
                        v.numShortestPaths = u.numShortestPaths
                  }
                  else if v.layer = u.layer + 1 {//shorter path found
                        v.numShortestPaths = v.numShortestPaths +
                        u.numShortestPaths   //adds the shorter path
            }
      }
      return v2.numShortestPaths
}
```

The initialization steps and queue/dequeue ops are all constant. Since each node is added once, it is basically equal to the number of Vertices (V). Then when it comes to checking the neighbors and updating the properties for each neighbor it is constant time. Since each edge is being added/explored a constant number of times, it is basically the number of Edges (E). This makes the runtime O(V+E).

(2)
```
findPath(G){ // uses reverseG because if v can reach all other nodes, then in G all nodes can reach v
    reverseG = ReverseGraph(G) // helper method given below
    for each v in reverseG {
        count = 0
        visited[] = {false for all v initially}
        Q = new Queue
        Q.enqueue(v)
        visited[v] = true
        count = 1
        while(!Q.isEmpty){
            u = Q.dequeue()
            for each neighbor w in reverseG.adj[u]{
                if not visited[w]{
                    visited[w] = true
                    count += 1
                    Q.enqueue(w)
                }
            }
        }
        if count == |V|{return true}
    }
    return false
}
```

Reversing the graph takes up O(|V|+|E|) time since we create using graph G with same number of edges and vertices. Then for each vertex that exists within our reverseG we do BFS on it which has a runtime of O(|V|+|E|) from class. This evaluates our total to O(V(V+E)) since we run BFS on |V| number of vertices.

```
ReverseGraph(G) {

reverseG = new Graph with same v as G

     for each u in G {

          for each vertex v in G.adj[u]{

            reverseG.adj[v].add(u)  //reverse edge

          }

     }

    return reverseG

}
```

(3)

```
findingBipartite(G, c1, c2) {

     for every vertex v in G { v.explored = false } // O(V)

     for each vertex s in G { // O(V)

          if (!s.explored) {

               s.layer = 0

               s.color = c1

               s.explored = true

               Q = new Queue

               Q.enqueue(s)

          while(!Q.isEmpty){

               u = Q.dequeue() // O(1) for enqueue and dequeue

               for each neighbor v of u { // O(E)

                    if v.explored == false {
```

```
                        Q.enqueue(v)
                        v.layer = u.layer + 1
                        if u.color == c1 {
                            v.color = c2
                        } else {
                            v.color = c1
                        }
                        v.explored = true
                    } else if v.color == u.color {
                        return nil
                    }
                }
            }
        }
}
```

Even though the outermost loop iterates over all the vertices in the graph, since BFS only happens on unexplored nodes. Therefore, the runtime for the above is O(V+E) since each node and edge is explored only once during the BFS.

(4)

Initial d-value	0	∞	∞	∞	∞	∞	∞	∞
Iteration	A	B	C	D	E	F	G	H
1	0	1	∞	5	∞	∞	∞	∞
2	0	1	3	5	10	∞	∞	∞
3	0	1	3	4	10	5	∞	8
4	0	1	3	4	10	5	∞	8
5	0	1	3	4	10	5	∞	6
6	0	1	3	4	10	5	∞	6
7	0	1	3	4	10	5	∞	6