Name: Maryfrances Umeora
URID: mumeora
Quiz CE

Problem 1 (4 points)
We have a connected graph G = (V, E), and a specific vertex u∈V. Suppose we compute a depth-first search tree rooted at *u*, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u, and obtain the same tree T. Prove that G = T. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)

My Proof
Here I will attempt a proof by contradiction.
Suppose that G *does* have an edge e = {*a*, *b*} that does not belong to T.

- As T is a DFS tree, one of the two ends must be an ancestor of the other—say *a* is an ancestor of *b*.
- Since T is also a BFS tree, the distance of the two nodes from *u* in T can differ at most by one.
- But if *a* is an ancestor of *b*, and the previous sentence holds, then *a* must be the direct parent of *b*. This means that {*a*, *b*} is an edge in T. This is a contradiction.

Thus, G doesn't contain any edges that don't belong to T.

Problem 2 (3 + 3 points)
Given is a undirected graph and two of its vertices s and t. Give an O(n+m) algorithm that computes the number of shortest paths from s to t.

My PseudoCode
The basic idea of this is to utilize the BFS algorithm to find the number of shortest paths. I basically first wrote code to do BFS traversal then tweaked it a little to get what I wanted. The main change is that I have an array called `paths` that keeps track of the number of paths to each vertices. If we ever get to a vertex n where we find that there is one more way to get to it than previously known, we increment the value. You will notice in my actual code that I have a few variables and arrays that don't really affect the answer I get in the end (such as the variable `shortestD` and the array `prev`), but I used them in my original implementation of the method so I left them there for future reference.

Most of the work done for this Problem 2 is done in the BFS method in the Graph class. After using a Scanner to read the input file and store the values of E, V, *s*, and *t*, as well as to add the given edges to my graph, I start the heavy work in the BFS method. Below is some pseudocode for what I did, with informative comments along the way:

```
procedure: BFS
Input: integers representing source s and destination t
Output: int representing the number of shortest paths from s to t
Other:- V: number of vertices in the graph
      - adjList[v]: an adjacency list for v ∈ V
      - visited[V]: array of booleans to mark vertices as unvisited
      - dist[V]: array to keep track of distances, filled with -1 at first
      - paths[V]: array to store #of shortest paths, filled with -1 at first
      - queue: a linked list of integers for BFS
--------------------------------------------------------------------------------
 01. dist[s] = 0; //distance from the current node to itself is zero
 02. paths[s] = 1; //guaranteed to find at least one path
 03. queue.add(s);
 04. visited[s] = true;
 05. WHILE queue is not empty
 06.   int focus = queue.poll(); //dequeue a vertex from queue
 07.   FOR n: adjList[focus]
 08.      if !visited[n]
 09.         visited[n] = true;
 10.         dist[n] = dist[focus] + 1;
 11.         paths[n] = paths[focus];
 12.         queue.add(n);
 13.      else
 14.         int alreadyKnownPath = dist[n];
 15.         int newPath = dist[focus] + 1;
 16.         if (newPath == alreadyKnownPath)
 17.            paths[n] += paths[focus];
 18. return paths[t];
```

How to Run my Actual Code

My code needs an input file to create the graph from. The input file should be in the format:

```
1. |E| |V|
2. s t
3. v1 v2 (representing an edge)
4. v3 v4 ….
```

Just like in the input file provided on BlackBoard.
Thus, when running my code in Eclipse, make sure to go to Run Configurations to type in the name of the input file.
Also, the teacher never specified exactly what way the output should be presented, so I just print it out to the console.