CSCI 2300
Introduction to Algorithms: HW2

Sean Hyde
RIN: 662096071

DUE: September 12th, 2024

1 Question 1:

`Solution:`

To determine if a graph is a bipartite set we need to first partition the graph into two groups, V_1 & V_2. We can accomplish this by performing BFS on a random node and assign it to V_1. As BFS searches, all of the neighboring nodes of the starting node will be set to V_2. We then set all neighboring nodes of V_2 to V_1, and continue this process, alternating between setting the neighbors of V_1 to V_2 and vice versa.

It is also given that BFS will naturally check the edges so BFS will be able to detect if there is an edge between two nodes within the same set. If there is an edge between nodes within the same set, we will return false. If there are no nodes in the same set with an edge, we will return true and we will know that the graph is a bipartite graph.

1.1 Proof:

My algorithm will return true if during the BFS search it is able to assign all neighboring nodes of a randomly chosen node, V_1 to V_2, and all neighboring nodes in the set of V_2 to V_1 while also verifying that no edges between nodes in the same set exist. If an edge between nodes from the same set exist, we will return false. If no nodes from the same set have an edge we will return true because the properties of a bipartite graph are, "that there are no edges between vertices in the same set."

1.2 Runtime Analysis:

Any subsequent steps within the algorithm other than BFS would just be a coefficient such as "+ 1", which is trivial to Big O notation, so the algorithms runtime is linear since the algorithm uses BFS, which would be:

```
O(V+E)
```

2 Question 2:

`Solution:`

If a cycle exists inside of the undirected graph g that has a particular edge e, which starts at a node u and connects to a target node v, we can check if there is a cycle by removing e and seeing if there is another path to v starting from node u without that particular edge e. If there is, then we can add e back in which will conclude and verify a cycle.

To start, we will remove edge e, then perform BFS beginning with the starting node u. BFS will begin and assign distances to all nodes of the undirected graph and find if there is another path to the target node v. After all nodes have been assigned distances and BFS is finished, we can now perform a simple if-else check to see if there was a path to the target node v from the starting node u without the given edge e being included. If that is true, then when we include e, we can conclude that there is a cycle, and we will also know the length of the cycle, but we have to add 1 because we added e back in as the nodes were assigned a distance during BFS. If it was false and a path was not found, we can conclude that there is not a cycle in the graph g with a particular edge e.

2.1 Runtime Analysis:

The runtime of the algorithm is linear as it uses BFS which is given as being a linear runtime searching algorithm:

```
O(V+E)
```

3 Question 3:

`Solution:`

To begin solving this question, we need to know the distances from the starting nodes in each graph, G_1 and G_2. We can begin by doing a BFS search from node s in G_1 and node t in G_2. After BFS is completed in both graphs, we will now know the distances from s for all nodes in G_1 and the same for G_2.

From here, we must calculate the distance for each potential edge from a node in $G1$ to a node in $G2$. The distance from the node would be calculated by:

```
first node's distance + 1 (edge that is being considered) + second node's distance
```

Then, we save each result for each potential edge and pick the smallest result, which would result in the shortest path between s and t among all the edges in E'.

3.1 Runtime Analysis:

The algorithms runtime is linear as the algorithm uses BFS twice which would be:

```
O(2v+2E)
```

As we know, Big O notation ignores the coefficients, so the final runtime for the algorithm would be linear which is:

```
O(V+E)
```