Eric Tashji

Module 11 Graphs and Traversals Application Programming Assignment

Problem: You need to design an efficient algorithm which will run in O(n + m) running time that computes for each station, the set of stations it can reach using with maximum 4 links.

Solution: The following algorithm will compute, for any station, the set of stations it can reach using a maximum of 4 links.

Algorithm findStations(*G, v*):

Input: *G*, the graph being explored, and *v*, the starting vertex.
Output: *depthList*, a list of the reachable stations from *v* using a maximum of 4 links

```
adjacent ← empty stack
adjacent.push(v)
depth ← 0
depthList ← empty list
temp ← empty stack
vertices ← 1

while vertices < the number of vertices in G AND depth < 5 do
        while adjacent.isEmpty() = false do
                depthList.add(adjacent.peek())
                for each edge, e, that is incident to adjacent.peek() in G do
                        If e unexplored then
                                Mark e explored
                                Let w be the vertex on the other end of e
                                temp.push(w)
                                vertices ← vertices + 1
                adjacent.pop()
        adjacent ← temp
        temp ← empty stack
        depth ← depth + 1
if depth < 5 then
        while adjacent.isEmpty() = false do
                depthList.add(adjacent.pop())
```

Output *depthList*

Eric Tashji

Variables: findStations accepts two parameters: ***G***, the map that is being explored by findStations, and ***v***, the vertex whose stations are being found. findStations also establishes five additional variables: ***adjacent***, a stack containing ***v***, which will be used to track the number of adjacent vertices, ***depth***, the depth of the current vertices being explored, ***depthList***, a list of vertices reachable by ***v*** using no more than 4 edges, and ***temp***, an empty stack used to store the elements to be placed in ***adjacent*** for the next iteration.

Explanation: findStations begins by conducting a depth-first search to find the depths of all of the vertices in ***G***, starting with vertex ***v***. Then, as long as the depth was less than 5 and there were more vertices to be found, the depth-first search would keep iterating. The depth-first search would then insert each element found into ***depthList***, which would contain a list of all the vertices reachable from ***v,*** using no more than 4 edges. findStations finishes by outputting ***depthList***.

Runtime: findStations has a runtime of $O(n + m)$. This is because it will iterate over each vertex and edge only once, giving it a runtime of $O(n + m)$, where n is the number of vertices and m is the number of edges.