FINAL PROJECT

Find shortest path algorithms

Name: Onel Hirmez

Class: CS 526

Algorithm 1 Pseudocode:

```
Algorithm Algorithm1(input, nodeArray, nodeObjects):
    Input: input is a String for the user input letter for starting node, nodeArray is a String[]
        array containing all the node names, nodeObjects is an ArrayList of the node
        Objects of the Node class.
    Output: Prints the path taken to get to the final node Z (including backtracks) and the
        shortest path to get there.
    Create a new ArrayList “path”, and a new Stack “shortestPath”.
    Set the user chosen node as the current node.
    While current node is not equal to Z node do:
        Add current node to “path” and “shortestPath”.
        Set current node is visited to true.
        Initialize new list “compareDD”
        For every entry in current node’s edges do
            If edge node is already visited then
                Continue
            Add current edge to list “compareDD”
        If “currentDD” is empty then
            Pop last entry in our stack “shortestPath”
            Set the popped node to the current node
            Continue
        For each element in “compareDD” do
            If element directDistance is less than minimum directDistance then
                Set node with minimum directDistance to current node
    Print the list “path” and the Stack “shortestPath”
```

Algorithm 2 Pseudocode:

```
Algorithm Algorithm2(input, nodeArray, nodeObjects):
    Input: input is a String for the user input letter for starting node, nodeArray is a String[]
        array containing all the node names, nodeObjects is an ArrayList of the node
        Objects of the Node class.
    Output: Prints the path taken to get to the final node Z (including backtracks) and the
        shortest path to get there.
    Create a new ArrayList "path", and a new Stack "shortestPath".
    Set the user chosen node as the current node.
    While current node is not equal to Z node do:
        Add current node to "path" and "shortestPath".
        Set current node is visited to true.
        Initialize new list "compareWeight"
        For every entry in current node's edges do
            If edge node is already visited then
                Continue
            Add current edge to list "compareWeight"
        If "currentWeight" is empty then
            Pop last entry in our stack "shortestPath"
            Set the popped node to the current node
            Continue
        For each element in "compareWeight" do
            Sum up directDistance and edge weight to total
            If element total is less than minimum total then
                Set node with minimum total to current node
    Print the list "path" and Stack "shortestPath".
```

Data Structures Used:

- The nodes were initially stored in a String[] array called nodeArray. It only consisted of the letter names of each node.
- The edges were stored in a HashMap called edges with keys as Strings and values as integers. Each key represented an adjacent node and each value the weight of the edge.
- The nodes were finally stored in a HashMap with keys as Strings and values as the Map edges mentioned above. This way each key had the name of the node and each value the edges of the node.

- nodeObjects is an ArrayList containing nodes as the objects of the Node class that each have different attributes such as isVisited and getEdges.
- A String[] ArrayList called path was created to store all the nodes that the algorithm visits including the backtracking until it gets to the final node.
- A String[] Stack called shortestPath was used to store the shortest path to get to the final node. Not including the backtracks.
- ArrayLists that contained Node objects called compareDD and compareWeight were created in order to help compare the direct distances and the weights of the edges to choose the minimum one.