Boston University

METCS 526: Final Project

By: Aidan Duffy

Section I, Pseudocode:

Algorithm One (G, exclude):

Input: an undirected, weighted, connected graph G and a list of Nodes exclude, if needed, otherwise it is null

Output: the node *v* that, among all neighboring nodes of *n*, has the smallest return value of *dd(v)*, or *direct distance* to the destination node Z.

minimum_distance = Integer.MAX_VALUE

best_node = null

For each node *v* which shares an edge with node *n* **do**:

    if *dd(v)* < minimum_distance and v is not in exclude **then:**

        minimum_distance = *dd(v)*

        best_node = v

return best_node

Algorithm Two(G, exclude):

Input: an undirected, weighted, connected graph G and a list of Nodes exclude, if needed, otherwise it is null

Output: the node *v* that, among all neighboring nodes of *n*, has the smallest return value of *dd(v)* + *w(n,v)*, or *direct distance* to the destination node Z plus the weight of the edge shared by node *n* and node *v*.

minimum_distance = Integer.MAX_VALUE

best_node = null

For each node *v* which shares an edge with node *n* **do**:

    if *w(n,v)* + *dd(v)* < minimum_distance and v is not in exclude **then:**

        minimum_distance = *dd(v)* + *w(n,v)*

        best_node = v

return best_node

Both of these algorithms will run iteratively until the current node *n* is Z. The main loop will also check if backtracking is ever needed (if we reach a dead end, go back one node and add the current node to the exclude list; if all the children of a node are in the exclude list, then return to the node prior to the current node in the shortest path list and add the current node to the exclude list).

Section II, Data Structures Used:

I created three new classes, a Node, an Edge, and a Graph.

For the *Node*, which was a single vertex on the graph, it contained:

1. **identifier:** a character for its letter value (ex: 'A', 'B',…, or 'Z')
2. **direct_distance:** an integer which is the Node's direct distance to destination Node Z.

For the *Edge*, which was the object for a single edge between two vertices, it contained:

1. **weight:** an integer value that represents the weight of the given edge
2. **nodeOne, nodeTwo:** two Nodes (order is irrelevant, I changed .equals function to ensure this) that make up the edge

For the *Graph*, it contained:

1. **nodes:** an ArrayList of all of the Nodes (or vertices) in this graph
2. **edges:** a HashMap of all of the edges in this graph where the key is one Node, and the value is another Node; the construction ensures no duplicates exist (ie. A:B and B:A will not both be in there)
3. **position:** the Node which is the current position of our graph traversal (ie: if we start at 'J', then this will initially be set to J)
4. **nodesWithEdges:** a HashMap of all of the nodes in the graph mapped to an ArrayList of this node's edges

In my *project file*, it contained:

1. **characterNode**: a HashMap with the character ID as the key and the actual Node as the value
    a. this was used for the two setup methods, Graph.constructGraph and Node.setDirectDistances since the input files contain just the character with the associated node.
2. **path**: an ArrayList which stores the characters that shows the full path from the given algorithms
3. **shortestPath:** an ArrayList which stores the characters of the nodes that shows the final, shortest/ideal path from the given algorithms
4. **exclude:** an ArrayList of nodes to exclude in searching for the best path, mostly used for backtracking while executing each algorithm