CS526

Final Project

Dhiraj Totwani

PseudoCode:

Define the Matrix class

```
// example of matrix class
class <Matrix> matrix
    private String startNode
    private String endNode
    private String weight
    //Constructor with default values
    //getters & setters
```

Define direct distance class

```
class <DD> direct distance
    private String Node
    private String Distance
    //Constructor with default values
    //getters & setters
```

Define the read input graph method(ArrayList<Matrix> list)

```
    Read all nodes from graph input file and store them in a data structure:
    Create a multidimensional array like ArrayList<String []>
    Use it to store the graph input tokens
    Create string[] array to store the tokens
    Create a new scanner variable
    While loop through the input file data until the end
        Add string[] array of tokens to the multidimensional array

    //start and end nodes would be stored separately to make it easy to combine the weights
    //the graph text file contains a tilde "~" in the first empty position to make that a zero index
    For loop through multidimensional array
        Store the specific start and end node values in their arrays
        ArrayList endNodeList {{ ~}, {A}, {B}, { C}.........}
        ArrayList startNodeList {{~}, {A}, {B}, {C}.........}
    Loop through each row of multidimensional array skipping the values associated with the
    end  and start nodes
    Matrix m = new Matrix();
    Assign startNode, endNode, and weight to m
    Add the matrix object to the ArrayList<Matrix> in the parameter// ex. list.add(m)
```

Define the read input Direct Distance method(ArrayList<DD> list)

Read all nodes and distance from direct distance input file and store in ArrayList of Direct Distance class/object type <DD>

```
    Create string [] to store tokens
    Create a scanner object to scan the direct distance input file
    While loop through each line of tokens
        Create a DD class object variable
        Create a string variable and store the node value from token[0]
        Create an integer variable, then integer parse the tokens[1] distance value and
        store it
        Assign the node and string value to the DD class object
        Add the DD object to the list in the parameters // example list.add(d)
```

Define the find weight method(string node1, string node2)

```
    Use the read Graph method to get the arraylist of matrix class
    Find the weight value of the nodes in the param (node1, node2)
        Loop through the arraylist of Matrix objects to see if it contains both node1 and
        node2
        Then return the weight value stored in that matrix object
```

Define the Direct Distance method(node)

Use the node in the parameter to loop through the direct distance input array and find the direct distance

Algorithm 1 & 2

```
Initialize variable that will hold a string value for the previous node.// for backtracking purpose
Initialize the variable that will hold the ArrayList<Matrix> that will contain the graph.
Use the read method to read the input file and assign it to the ArrayList<Matrix> array.
Initialize the ArrayList that will hold the sequence of nodes
Initialize the ArrayList that will hold the shortest path of nodes
Initialize an integer variable that will hold the shortest path length
Use direct distance method to check if the input is the destination node.
If the input is not the destination node then continue the program. Else if it is the destination node or an invalid node value skip and prompt user to enter another node.
    Create a while loop
        Create an ArrayList to store adjacent nodes of the current value of n
        get adjacent nodes and store them in the list
        if the adjacent node list size is only one and its value is the previous node
            remove the dead end from the matrix array and the ArrayList holding the shortest path

            reassign the search node value of n to the previous node
            restart the while loop  with the updated matrix array// removed dead end
        Initialize a string variable to stored the node value with the smallest dd
        Check if any node is already in the path sequence
            If it is remove it from the adjacent node ArrayList
```

Only Algorithm 1-(Find and return the adjacent node with smallest direct distance using the direct distance method)

Only Algorithm 2-(Find the return the adjacent node with smallest direct distance by calculating the sum of the direct distance method(current node) and weight method (current node, and it’s adjacent nodes)

```
Assign the smallest direct distance node to the Initialized string variable
If the direct distance of the node is 0, then the final node is reached
    Add the final node to the shortest path and sequence ArrayLists
    Print sequence of the nodes from it’s ArrayList
    Print the Shortest Path from it’s ArrayList
    Print the Shortest Path Length
        Use the w() method to get the length of the 2 paths
        Add the short path values together in the shortest path length
        variable
        Print (“Shortest Path Length: “ + shortest path length variable)
Else
    Add the selected smallest node to the shortest path and sequence
    ArrayLists
    the previous node variable equals the current node n
    the current node n variable equals the selected smallest node
```

Create Main program to test the Algos

```
While Loop
    Read input from user and store in variable
    Validate user input is type character
    Call the first Algorithm method
    Print an empty line
    Call the next Algorithm
    Print an empty line
```

Data Structures Used:

- Multidimensional Arrays
- Array Lists
- Normal Arrays

I used a multidimensional array in the beginning to help parse the data in the Graph_input text file. The file had to be stored in a multidimensional array. I used the combination of an ArrayList and a normal String[] array to hold the tokens returned from the scanner object. That allowed me to use the ArrayList methods "*.contains, .remove, .get, .add".* That made it easier to find, update, and remove node values in the ArrayList.

ArrayLists are a very convenient data structure for the purpose of this project because it gave me flexibility to find, remove, update, and return values whenever I needed. I could get the exact value I needed without having to use too many loops with complex patterns of index values. I could simply use the index value of the specific matrix and use the .endNode, .startNode, or .weight function to get the specific element.