ENGR 476-02

Lab 3: Djikstra Routing Project

By: Ahmad El Shakoushy

12/2/18

Note: I will paste all of my code in the following pages of this document, but I will also be submitting this in the form of a zip file which includes all of my .java files as well as the input map txt file.

Node Class:

```
import java.util.ArrayList;

public class Node {
    public String name;
    int cost_from_start;
    boolean is_start = false;
    boolean is_end = false;
    boolean visited = false;  //unvisited by false
    Node prev;
    public ArrayList<Node> neighbors; //we add to this upon the creation of an edge

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getCost_from_start() {
        return cost_from_start;
    }

    public void setCost_from_start(int cost_from_start) {
        this.cost_from_start = cost_from_start;
    }

    public boolean isIs_start() {
        return is_start;
    }

    public void setIs_start(boolean is_start) {
        this.is_start = is_start;
    }

    public boolean isIs_end() {
        return is_end;
    }

    public void setIs_end(boolean is_end) {
        this.is_end = is_end;
    }

    public boolean isVisited() {
        return visited;
    }

    public void setVisited(boolean visited) {
        this.visited = visited;
    }
    public void add_neighbor(Node node_to_add){  //allows for addition of a neighbor
to the node
        this.neighbors.add(node_to_add);
    }

    public Node getPrev() {
        return prev;
    }
```

```
    public void setPrev(Node prev) {
        this.prev = prev;
    }

    public Node(String name, boolean is_start, boolean is_end){
        this.name = name;
        this.is_start = is_start;
        this.is_end = is_end;
        this.neighbors = new ArrayList<>();
        if(this.isIs_start()){
            this.cost_from_start = 0;
        }else{
            this.cost_from_start = 1000;
        }
    }

}
```

Main:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;
import java.util.StringTokenizer;
@SuppressWarnings("Duplicates")

public class Main {

    public static void main(String[] args) {

        Node current; // we will hold the current node here

        //new nodes (later is_start and is_end will be taken from user input)
        ArrayList<Node> nodes = new ArrayList<>();
        //ArrayList<Edge> edges = new ArrayList<>();
        HashMap<String, Integer> costMap = new HashMap<>();
        String result = "";
        int final_cost = 0;
        int num_visited = 0;
        Node first_obj = null;
        Node last_obj = null;


        //prompting user for start and end nodes
        System.out.print("Please enter the start node: ");
        Scanner s = new Scanner(System.in);
        String choice_start = s.next();

        System.out.print("Please enter the end node: ");
        Scanner sr = new Scanner(System.in);
        String choice_end = sr.next();

        //reading file
        try {
            File file = new File("src/Djikstra_Files/djikstra_input.txt");
```

```
            BufferedReader b = new BufferedReader(new FileReader(file));
            String line;

            StringTokenizer st;
            while ((line = b.readLine()) != null) {
                st = new StringTokenizer(line);
                String name1 = st.nextToken();
                boolean name1_exists = false; //tells us whether that node has already
been created
                String name2 = st.nextToken();
                boolean name2_exists = false; //tells us whether that node has already
been created
                int value = Integer.parseInt(st.nextToken());


                for (int i = 0; i < nodes.size(); i++) {
                    if (nodes.get(i).name.equals(name1)) { //1st node exists?
                        name1_exists = true;
                    }
                    if (nodes.get(i).name.equals(name2)) { //2nd node exists?
                        name2_exists = true;
                    }
                }


                if (!name1_exists) { //if the node for name1 doesn't exist, we create
it and add to nodes list
                    Node temp1;
                    boolean put_in_front = false;

                    if (choice_start.equals(name1)) { //start node
                        temp1 = new Node(name1, true, false);
                        put_in_front = true;
                    } else if (choice_end.equals(name1)) { //end
                        temp1 = new Node(name1, false, true);
                        last_obj = temp1;
                    } else { //not start or end
                        temp1 = new Node(name1, false, false);
                    }
                    if (put_in_front) {
                        nodes.add(0, temp1);
                        first_obj = temp1;
                    } else {
                        nodes.add(temp1);
                    }

                }
                if (!name2_exists) { //if the node for name2 doesn't exist, we create
it and add to nodes list
                    Node temp2;
                    boolean put_in_front = false;

                    if (choice_start.equals(name2)) { //start
                        temp2 = new Node(name2, true, false);
                        put_in_front = true;
                    } else if (choice_end.equals(name2)) { //end
                        temp2 = new Node(name2, false, true);
                        last_obj = temp2;
                    } else { //not start or end
                        temp2 = new Node(name2, false, false);
                    }
                    if (put_in_front) {
                        nodes.add(0, temp2);
```

```
                        first_obj = temp2;
                    } else {
                        nodes.add(temp2);
                    }
                }

            }
        } catch (Exception x) {
            System.out.println(x.toString());

        }

        //this block is used to set neighbors and fill the HashMap whereas the first
try catch is used to create nodes and populate the nodes list
        try {
            File file = new File("src/Djikstra_Files/djikstra_input.txt");
            BufferedReader br = new BufferedReader(new FileReader(file));
            String line2;

            StringTokenizer stt;
            while ((line2 = br.readLine()) != null) {

                stt = new StringTokenizer(line2);
                String name1 = stt.nextToken();
                String name2 = stt.nextToken();
                int value = Integer.parseInt(stt.nextToken()); //will be used to fille
hash map

                costMap.put(name1 + " " + name2, value); //filling cost map
                costMap.put(name2 + " " + name1, value); //filling cost map

                Node obj1 = null;
                Node obj2 = null;

                for (int i = 0; i < nodes.size(); i++) {   //after this loop, obj1 and
obj2 will hold two adjacent nodes
                    if (nodes.get(i).name.equals(name1)) {
                        obj1 = nodes.get(i);  //set obj1 to first node
                    }
                    if (nodes.get(i).name.equals(name2)) {
                        obj2 = nodes.get(i); //set obj2 to second node
                    }
                }

                if (!obj1.neighbors.contains(obj2)) { //if obj1 doesn't have obj2 as a
neighbor
                    obj1.add_neighbor(obj2);
                }
                if (!obj2.neighbors.contains(obj1)) { //if obj2 doesn't have obj1 as a
neighbor
                    obj2.add_neighbor(obj1);
                }

            }
        } catch (Exception x) {
            System.out.println(x.toString());
        }

        current = nodes.get(0); //current is first one (min)

        while (num_visited < nodes.size()) {

            int temp_min = 100000;
```

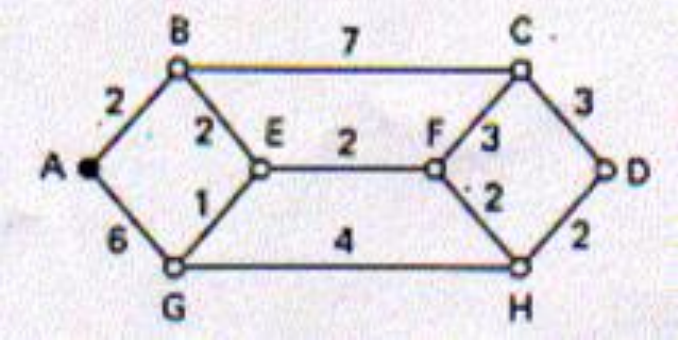
```
            Node temp_min_node = null;

            for (int j = 0; j < current.neighbors.size(); j++) {  //loop through all
neighbors for the node
                int curr_neighbor_cost =
current.neighbors.get(j).getCost_from_start();
                int curr_node_cost = current.getCost_from_start();
                int edge_to_neighbor_cost = costMap.get(current.name + " " +
current.neighbors.get(j).name);

                if (curr_neighbor_cost > curr_node_cost + edge_to_neighbor_cost &&
!current.neighbors.get(j).isVisited()) { //is neighbor cost greater? (we should
overwrite it)
                    current.neighbors.get(j).setCost_from_start(curr_node_cost +
edge_to_neighbor_cost);
                    current.neighbors.get(j).setPrev(current);

                }

            }
            for (int i = 0; i < nodes.size(); i++) {
                if (!current.equals(nodes.get(i))) {
                    if (nodes.get(i).cost_from_start < temp_min &&
!nodes.get(i).isVisited()) { //if current neighbor has a smaller cost, set it to that
as long as its unvisited
                        temp_min = nodes.get(i).getCost_from_start();
                        temp_min_node = nodes.get(i);
                    }
                }

            }

            current.setVisited(true); //setting as visited after checking all neighbor
nodes
            num_visited++;

            if (temp_min_node != null) { //if the next node is not null, we advance
current
                current = temp_min_node; // advancing current
            }

        }


     //displaying the final path/cost
        String str_result = "";
        StringBuilder sb = new StringBuilder();

        int result_cost = last_obj.cost_from_start;

        System.out.println("\n"+"Final cost: " + result_cost);
          if(!choice_end.equals( choice_start) ) { //this check is done for cases
where the start node = end node

              Node before = last_obj.prev;

              while (before != null) {
                  str_result += (">-" + before.name);
                  before = before.prev;
              }

              sb.append(str_result);
              sb = sb.reverse();
```

```
                sb.append(last_obj.name);
                System.out.println(sb);
            }else{
                System.out.println(choice_start +  "->" + choice_end);

            }

    }
}
```

Sample Output:

```
Please enter the start node: A
Please enter the end node: D

Final cost: 10
A->B->E->F->H->D
```

Graph used for input text file:



Map input txt file:

```
A B 2
A G 6
B E 2
B C 7
G E 1
E F 2
F C 3
C D 3
F H 2
G H 4
H D 2
```