Project Cover Page

Course number: CS 526
Assignment name: Fall 2019 Project Assignment
Name: Yudi Mao

Major datastucture used is arraylists and matrix.

Pseudocodes
Algorithm 1

```
/** Method to get adjacent node due to algorithm1. */
public static String get_adjacent_node(String s, List<String> v, List<Integer> d, String[][] w) {
      String this_node = s;
      List<String> vertex = v;
      List<Integer> dd = d;
      String[][] weight = w;
      int n = dd.size(); //number of nodes

      /** Get index of this node. */
      int index = vertex.indexOf(this_node);

      /** Find possible adjacent nodes. */
      /** Create a list to store those nodes. */
      List<String> p_nodes = new ArrayList<>(); //create a temp list to store possible adjacent
nodes
      for (int i = 1; i <= n; i++) {
         if (Integer.parseInt(weight[index + 1][i]) > 0) {
            p_nodes.add(weight[0][i]);
            //System.out.println("Possible node: " + weight[0][i]);
         }
      }

      /** Get the adjacent node due to dd(v). */
      int Temp_dd = 99999;
      for (int j = 0; j < p_nodes.size(); j++) {
         String p_node = p_nodes.get(j);
         int index_2 = vertex.indexOf(p_node);
         int p_dd = dd.get(index_2);
         if (p_dd < Temp_dd) {
            Temp_dd = p_dd;
         }
      }

      /** Return the most adjacent node. */
      int index_3 = dd.indexOf(Temp_dd);
      return vertex.get(index_3);
```

```
    }

    /** Method to get shortest path using algorithm1. */
    public static void get_shortest_path(String s, List<String> v, List<Integer> d, String[][] w) {
        String start_node = s;
        List<String> vertex = v;
        List<Integer> dd = d;
        String[][] weight = w;

        /** Put start node into path. */
        /** Construct two lists to save sequence of all nodes & shortest path. */
        List<String> testVex = new ArrayList<>();
        List<String> pathVex = new ArrayList<>();
        testVex.add(start_node);
        pathVex.add(start_node);

        String this_node = start_node;

        /** Loop when next node is not destination “Z”. */
        while (!this_node.equals("Z")) {
            int index_f = vertex.indexOf(this_node);
            /** If dead end happens, save to sequence but remove last node in shortest path/
*/
            if (pathVex.contains(get_adjacent_node(this_node, vertex, dd, weight))) {
                String dele_node = this_node;
                this_node = get_adjacent_node(this_node, vertex, dd, weight);
                testVex.add(this_node);
                pathVex.remove(dele_node);

                /** Change related matrix number to zero so dead node won’t be picked again.
*/
                int dele_index = vertex.indexOf(dele_node);
                int this_index = vertex.indexOf(this_node);
                weight[this_index + 1][dele_index + 1] = "0";
                weight[dele_index + 1][this_index + 1] = "0";
               /** If dead end didn’t happen, add nodes to both lists. */
            } else {
                this_node = get_adjacent_node(this_node, vertex, dd, weight);
                testVex.add(this_node);
                pathVex.add(this_node);
            }
        }

        /** Print, calculate path length */
        int path_len = 0;

        System.out.println("Algorithm 1:");
```

```
String Seq = "Sequences of all nodes: " + testVex.get(0);
for (int i = 1; i < testVex.size(); i++) {
   Seq = Seq + " -> " + testVex.get(i);
}
System.out.println(Seq);
String Path = "Shortest Path: " + pathVex.get(0);
for (int j = 1; j < pathVex.size(); j++) {
   Path = Path + " -> " + pathVex.get(j);
   int index_f = vertex.indexOf(pathVex.get(j - 1));
   int index_l = vertex.indexOf(pathVex.get(j));
   path_len += Integer.parseInt(
         weight[index_f + 1][index_l + 1]);
}
System.out.println(Path);
System.out.println("Shortest path length: " + path_len);
```

Algorithm2

```
/** Method to get adjacent node due to algorithm2. */
/** Same parts are skipped . */\
   public static String get_adjacent_node_2(String s, List<String> v, List<Integer> d, String[][]
w) {
         /** Construct variables. */

         /** Get index of this node. */

         /** Find possible adjacent nodes. */

         /** Get the adjacent node due to w(n,v)+dd(v) */
         int Temp_dd = 99999;
         int Temp_index = 0;
         for (int j = 0; j < p_nodes.size(); j++) {
            String p_node = p_nodes.get(j);
            int index_2 = vertex.indexOf(p_node);
            int p_dd = dd.get(index_2) + Integer.parseInt(weight[index + 1][index_2 + 1]);
            if (p_dd < Temp_dd) {
               Temp_dd = p_dd;
               Temp_index = index_2;
            }
         }

         /** Return the most adjacent node */
         return vertex.get(Temp_index);
      }

   /** Method to get shortest path using algorithm2. */
   Just change get_adjacent_node to get_adjacent_node_2 in the method to get shortest
path using algoritm1.
```