CSE 5311-004 DESIGN AND ANALYSIS OF ALGORITHMS

PROGRAMMING ASSIGNMENT 3

REPORT

Documentation/Explanation of how your code works (include important code snippets)

Before, we start explaining the working of our code, we would like to explain what Dijkstra's Algorithm is in short. So, what is it?

- Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

So, as we are aware of what this document is about, we will start understanding the implementation of the Dijkstra Algorithm in coding point of view.

1. We will first take the number of input vertices for our graph and then we will take vertex name for the same.
2. Now, we will take edges to connect these vertices, before that we will take number of edges we are going to take, then the vertex starts and end point with its weight. You, will see output to enter details this way for point 1 and 2.

```
Enter number of vertices: 5

***Enter name of vertex in serial order***
Enter Character 1 of 5: a
Enter Character 2 of 5: b
Enter Character 3 of 5: c
Enter Character 4 of 5: d
Enter Character 5 of 5: e

How many number connection you would like to make: 6


***Enter Details for Connection 1***
Start Node: a
End Node: b
Distance weight: 1
```

3. Once we have the edges, now we have to make adjacency matrix, we will use the **find** method, in order to find start and end vertex index value to plot in matrix. And then as it is undirected graph, we will store it both [i][j] and [j][i] position the same value for pathmatrix, with the help of this:

```
public int find(String[] a, String target, boolean diff)
{
    for (int i = 0; i < a.length; i++)
        if (target.equals(a[i]))
            if (diff == true)
                return i;
            else
                return 1;
    return -1;
}
```

```
startindex=find(vname, start, true); // using find
endindex=find(vname, end, true); // using find fun
if(startindex != -1 && endindex != -1){  // store
    pathmatrix[startindex][endindex]=distance; //
    pathmatrix[endindex][startindex]=distance;
}
```

4. Then after printing the Adjacency matrix for the same.
5. Now, it's time to find the shortest distance, with the help of dist[] and sptSet[], we will able to achieve develop the Dijkstra's algorithm. We will first find the src index position using **find** method and make the distance to zero. But, before that we will loop through the array for dist[] and sptSet[] and make it infinite and false respectively.
6. Now, we will loop through all the vertices. Thereafter, inside loop, first we will call **mindistance** method, this method will set the min value and also will return in which index value the minimum has been set, while checking whether that index is already used and if our new dist is min than previous, then only replace the value and return min-index.

```
for (int v = 0; v < vertex; v++)
    if (sptSet[v] == false && dist[v] <= min){
        min = dist[v];
        min_index = v;
    }
```

7. Once, got the min_index, set sptSet true for it, and now comes the main Dijsktra logic, where it says, the vertex should not have been visited, the new value should not be zero, the integer must not be max value and the distance + graph should be small than existing distance, if yes then update it to new value.

```
for (int v = 0; v < vertex; v++)
    if (!sptSet[v] && graph[u][v]!=0 && dist[u] != Integer.MAX_VALUE && dist[u]+graph[u][v] < dist[v])
        dist[v] = dist[u] + graph[u][v];  // update dist[v] only if it is not present in sptSet, edge present from u to
```

8. Likewise, loop it for each character except the last remaining, as all other nodes are processed once, it will not have anything to left check further. Then, using **printSolution** method print the output as requested.

TEST Output Adjacency Matrix, Default shortest path, User Shortest Path:

```
*** Adjacency Matrix ***
[
[ 0, 2, 7, 0, 4 ],
[ 2, 0, 0, 1, 0 ],
[ 7, 0, 0, 0, 1 ],
[ 0, 1, 0, 0, 2 ],
[ 4, 0, 1, 2, 0 ],
]

Vertex Distance from: A
A->A(A: 0)
A->B(A, B: 2)
A->C(A, B, D, E, C: 5)
A->D(A, B, D: 3)
A->E(A, B, D, E: 4)

Here are the list of vertices to choose from for user defined starting point
A B C D E
Enter the Starting vertex which you would like to try out?
C

*** Adjacency Matrix ***
[
[ 0, 2, 7, 0, 4 ],
[ 2, 0, 0, 1, 0 ],
[ 7, 0, 0, 0, 1 ],
[ 0, 1, 0, 0, 2 ],
[ 4, 0, 1, 2, 0 ],
]

Vertex Distance from: C
C->A(C, E, D, B, A: 5)
C->B(C, E, D, B: 4)
C->C(C: 0)
C->D(C, E, D: 3)
C->E(C, E: 1)

Do you want to try out other Starting point? y or n
```

Time Complexity of Dijsktra Algorithm: $O(n^2)$

Example where Dijsktra Algorithm is been used:

1. It is used in finding Shortest Path
2. It is used in geographical Maps.
3. To find locations of Map which refers to vertices of graph.
4. It is used in IP routing to find Open Shortest Path First.
5. It is used in telephone network.
6. It's also used in A* algorithm.
7. Apart from this, Google Maps also make use of Dijsktra Algorithm.
8. Intelligent Fire Evacuation System.
9. Interval Valued neutrosophic shortest path problem
10. Word ladder puzzle problem

References:

1. https://www.quora.com/What-are-the-real-life-applications-of-Dijkstras-algorithm
2. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
3. https://www.youtube.com/watch?v=XB4MIexjvY0
4. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
5. https://www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstras-algorithm/introduction
6. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6305611
7. https://ieeexplore.ieee.org/document/7850151
8. https://stackoverflow.com/questions/4412031/what-are-the-applications-of-the-shortest-path-algorithm