VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

SIMPLE NAVIGATION SYSTEM USING DIJKSTRA'S ALGORITHM

Submitted by

Candidate Name

NAME	USN
Sameer Singh	1RN20IS131
Sanjana Shenoy	1RN20IS137
Sharanya RP	1RN20IS143
Shreya M Bharadwaj	1RN20IS154

Under the Guidance of

Faculty In charge

Dr. Suresh L

Designation: Professor and HOD

Dept. of ISE, RNSIT

Department of Information Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,

Bengaluru – 560 098

2021 – 2022

TABLE OF CONTENTS

ABSTRACT **3**

INTRODUCTION **4**
1.1 Introduction 4
1.2 Objectives 5
1.3 Key Features 5

ALGORITHM **6**
2.1 Algorithm Design Technique Used 6
2.2 Algorithm 6

DESIGN **7**
3.1 Flowchart 7
3.2 Graph 9

SYSTEM REQUIREMENTS **10**

IMPLEMENTATION MODULES **11**

CODE **12**

RESULT **17**

TIME COMPLEXITY **19**

APPLICATIONS **20**

CONCLUSION & FURTHER ENHANCEMENTS **21**
10.1 Conclusion 21
10.2 Further Enhancements 21
10.3 References 21

ABSTRACT

A navigation system is a computing system that aids in navigation. It contains graphical maps, which determines vehicle location via sensors or information. It also provides coordinates, and directions on a screen. It is a device that has the capability of knowing your current position, and allows you to determine the route to a particular place. It provides the direction a person should travel as well as how much distance it is to reach the destination. The starting location and destination location must be inputted by the user, and the navigation system determines the easiest possible route to travel in.

CHAPTER 1:

INTRODUCTION

1.1 Introduction

A navigation system is a computing system that aids in navigation. It contains graphical maps and coordinates which determines vehicle location via sensors or information. Quite simply, a navigation system helps drivers get from point A to point B.

It provides the direction a person should travel as well as how much distance it is to reach the destination. The directions will be visible on a screen.

The device that has the capability of knowing your current position, and allows you to determine the route to a particular place. The starting location and destination location must be inputted by the user, and the navigation system determines the easiest possible route to travel in.

Planning a route is the easiest and most basic thing a navigation system should be able to do. Usually, a navigation system offers alternative ways based on your preference for the fastest, the cheapest, or the most convenient route. You can also choose to avoid borders, ferries, tolls, highways, etc. The system will guide you to the necessary lane as well as automatically reroute you if you deviate from the initial course. Some embedded navigation systems allow adding up to 50 waypoints. Plus, you can usually see the history of your trips and choose to repeat them or backtrack.

The need to have navigation system is for the following reasons:

- To provide information on nearby vehicles or vessels, or other hazards or obstacles.
- To provide information about the distances between any two cities.
- To showing the route that takes you to your destination with the least number of stops,
- It provides information on traffic conditions and suggesting alternative directions.
- The easy mapping of cites to find the easiest possible route.

Our project is to create a simple navigation system for a vehicle travelling from one city to another.

1.2 Objectives

The Navigation System should be able to do the following:

- Distance between two locations should be easily found.
- Show a route that takes you to your destination with the least number of stops.
- A good navigation system must allow you to add stops.
- Optimized route which includes the desired stop can be calculated.
- Create a real-life application of Dijkstra's algorithm.

1.3 Key Features

- Allows for route planning and waypoints.
- Turn-by-turn directions.
- Provides a map with your route.

CHAPTER 2:

ALGORITHM

2.1 Algorithm Design Technique Used

- Dijkstra's algorithm allows us to implement the shortest distance between two nodes in a graph.
- It is a form of greedy approach.

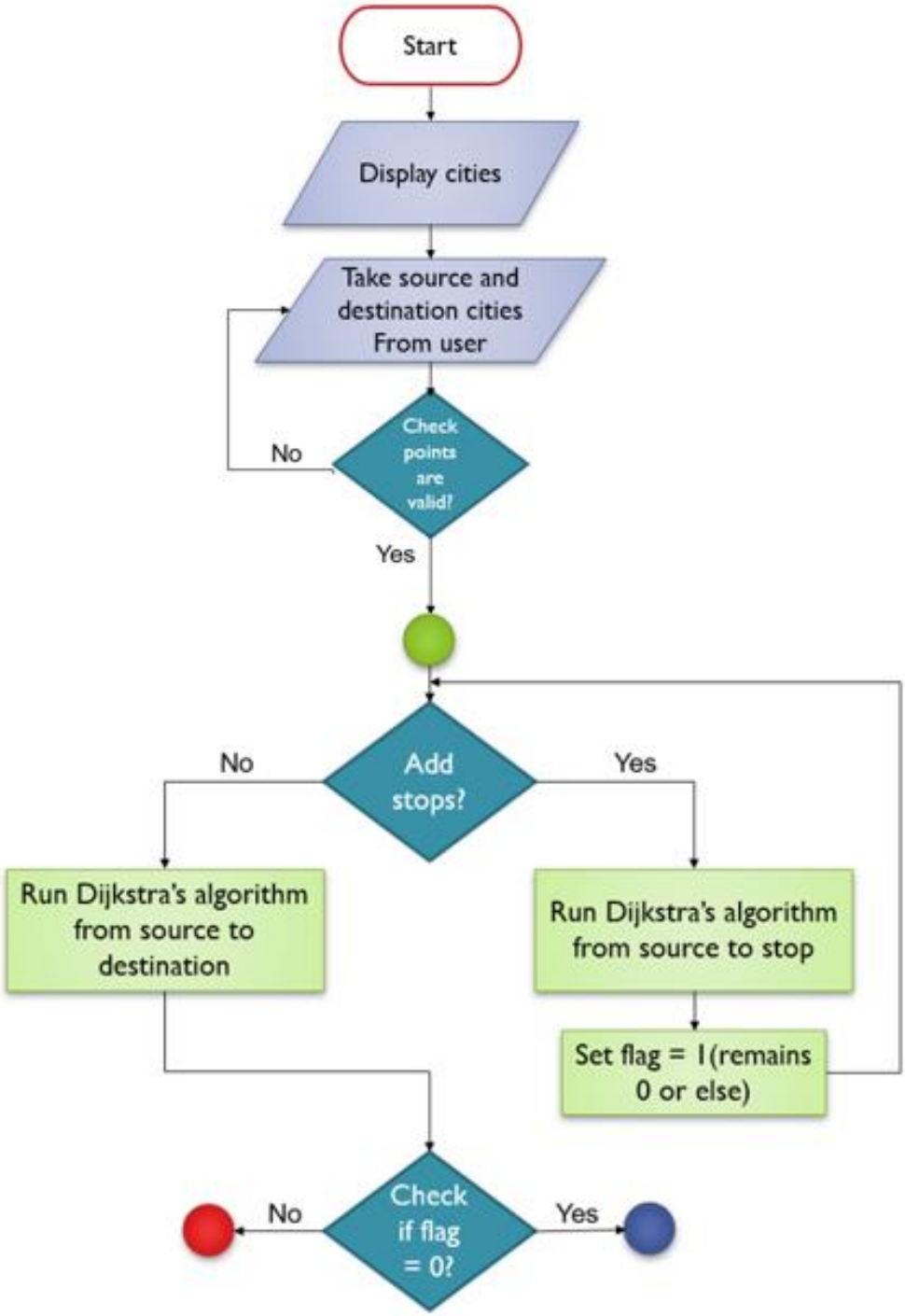
2.2 Algorithm

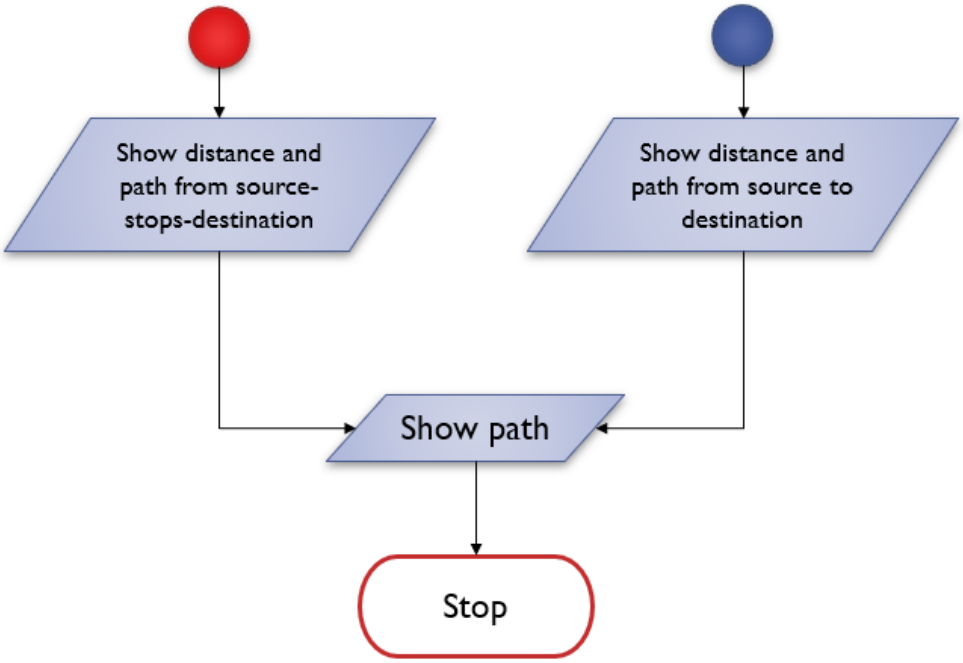
```
Algorithm SSSP(C, n, source)
//input: C is (n*n) matrix where n is the number of vertices
       source is starting vertex
//output: D array that stores the shortest distance
for i<-i to n do
        D[i]=cost(source, i)
Add source to S
for i<-1 to n-1 do
{
        find a vertex w such that
                D[w] is minimum
   Add w to S
                for each vertex v belongs to V-S do
        D[v]=min(D[v], D[w]+ C(w,v))
}
print D array
```

CHAPTER 3:

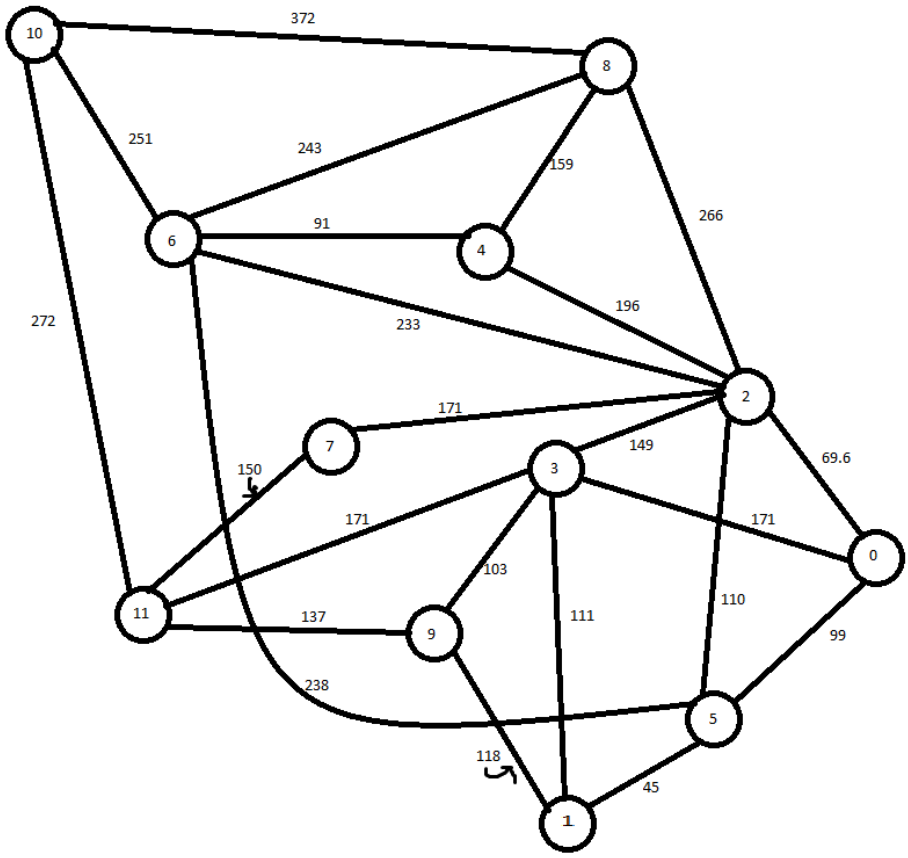
DESIGN

3.1 Flowchart




Show distance and path from source-stops-destination
Show distance and path from source to destination
Show path
Stop

3.2 Graph



Cities:

1. Bangalore
2. Mysore
3. Tumkur
4. Hassan
5. Davangere
6. Mandya
7. Shivamogga
8. Chikkamagluru
9. Ballari
10. Madikeri
11. Karwar
12. Mangalore

CHAPTER 4:

SYSTEM REQUIREMENTS

4.1 Hardware Requirements

- **Processor:** Intel Core2 Quad @ 2.4Ghz on Windows® Vista 64-Bit / Windows® 7 64-Bit / Windows® 8 64-Bit / Windows® 8.1 64-Bit.
- **RAM:** 2GB of RAM
- **Memory:** 256GB Hard drive
- **Keyboard:** MS compatible keyboard
- **Mouse:** MS compatible mouse

4.2 Software Requirements

- **Operating system:** Windows® Vista 64-Bit / Windows® 7 64-Bit / Windows® 8 64-Bit / Windows® 8.1 64-Bit.
- **Front end Programming language:** Java
- **IDE:** Eclipse

CHAPTER 5:

IMPLEMENTATION MODULES

- public static void main(String[] args)):

  Driver code; contains the adjacency matrix that represents the graph of cities. Displays the menu.

- public static String cityName(int vertex):

  Returns the name of the city when the vertex number is passed as an argument.

- private static void dijkstra(int[][] adjacencyMatrix, int startVertex, int endVertex):

  Finds the shortest path between two nodes of the graph. Calls printSolution function.

- private static void printSolution(int startVertex, int endVertex, int[] distances, int[] parents):

  Distance between the two cities is printed. Calls printPath function.

- private static void printPath(int currentVertex, int[] parents, int endVertex):

  Path taken will be printed.

CHAPTER 6:

CODE

```
package map;

import java.util.Scanner;

public class Route {

	static int total = 0;

	public static String cityName(int vertex) {

		String city ="";

		if(vertex == 0) {

			city = "Bangalore";

		}else if(vertex == 1) {

			city = "Mysore";

		}else if(vertex == 2) {

			city = "Tumkur";

		}else if(vertex == 3) {

			city = "Hassan";

		}else if(vertex == 4) {

			city = "Davangere";

		}else if(vertex == 5) {

			city = "Mandya";

		}else if(vertex == 6) {

			city = "Shivamogga";

		}else if(vertex == 7) {

			city = "Chikkamagaluru";

		}else if(vertex == 8) {

			city = "Ballari";
```

```
        }else if(vertex == 9) {
                city = "Madikeri";
        }else if(vertex == 10) {
                city = "Karawar";
        }else if(vertex == 11) {
                city = "Mangalore";
        }
        return city;
}
public static void dijkstra(int[][] graph, int sourceVertex, int destinationVertex) {
        int vertexCount = graph.length;
        boolean[] visitedVertex = new boolean[vertexCount];
        int[] distance = new int[vertexCount];
        for (int i = 0; i < vertexCount; i++) {
                visitedVertex[i] = false;
                distance[i] = Integer.MAX_VALUE;
        }
        distance[sourceVertex] = 0;
        for (int i = 0; i < vertexCount; i++) {
                int u = findMinDistance(distance, visitedVertex);
                visitedVertex[u] = true;
                for (int v = 0; v < vertexCount; v++) {
                        if (!visitedVertex[v] && graph[u][v] != 0 && (distance[u] +
                graph[u][v] < distance[v])) {
                                distance[v] = (int) (distance[u] + graph[u][v]);
                        }
                }
```

```
        }
        System.out.println(String.format("Distance from %s to  %s is %s kms",
cityName(sourceVertex),
        cityName(destinationVertex), distance[destinationVertex]));
        total += distance[destinationVertex];
}
private static int findMinDistance(int[] distance, boolean[] visitedVertex) {
        int minDistance = Integer.MAX_VALUE;
        int minDistanceVertex = -1;
        for (int i = 0; i < distance.length; i++) {
                if (!visitedVertex[i] && distance[i] < minDistance) {
                        minDistance = distance[i];
                        minDistanceVertex = i;
                }
        }
        return minDistanceVertex;
}
public static void main(String[] args) {
        int graph[][] = new int[][] {
{0,    0,    69,   171,  0,    99,   0,    0,    0,    0,    0,    0},
{0,    0,    0,    111,  0,    45,   0,    0,    0,    118,  0,    0},
{69,   0,    0,    149,  193,  110,  233,  171,  266,  0,    0,    0},
{171,  111,  149,  0,    0,    0,    0,    0,    0,    103,  0,    171},
{0,    0,    193,  0,    0,    0,    91,   0,    159,  0,    0,    0},
{99,   45,   110,  0,    0,    0,    238,  0,    0,    0,    0,    0},
{0,    0,    233,  0,    91,   238,  0,    0,    243,  0,    251,  0},
{0,    0,    171,  0,    0,    0,    0,    0,    0,    0,    0,    150},
```

```
{0,   0,   266,  0,   159,  0,   243,  0,   0,   0,   372,  0},
{0,   118,  0,   103,  0,   0,   0,   0,   0,   0,   0,   137},
{0,   0,   0,   0,   0,   0,   251,  0,   372,  0,   0,   272},
{0,   0,   0,   171,  0,   0,   0,   150,  0,   137,  272,  0}
};
int src,last = 0,dest,ch = 1;
int i = 0;
int flag = 1;
int[] stop = new int [11];
Scanner scan = new Scanner(System.in);
System.out.println("Cities:\n1.Bangalore \n2.Mysore \n3.Tumkur \n4.Hassan"+
"\n5.Davangere \n6.Mandya\n7.Shivamogga\n8.Chikkamagluru"+ "\n9.Ballari \n10.Madikeri
\n11.Karwar \n12.Mangalore");
System.out.println("Enter your starting point");
src = scan.nextInt() - 1;
System.out.println("Enter your end point");
dest = scan.nextInt() - 1;
while(ch != 0){
    System.out.println("Do you want to add stops?(enter 1 for yes, 0 for no)");
    ch = scan.nextInt();
        if(ch == 0)
            break;
System.out.println("Cities:\n1.Bangalore\n2.Mysore\n3.Tumkur\n4.Hassan"+
"\n5.Davangere \n6.Mandya\n7.Shivamogga\n8.Chikkamagluru"+ "\n9.Ballari \n10.Madikeri
\n11.Karwar \n12.Mangalore");
System.out.println("\nEnter extra stop");
stop[i] = scan.nextInt() - 1;
if(i==0)
```

```
                        dijkstra(graph, src, stop[i]);
                else
                        dijkstra(graph, stop[i-1], stop[i]);
                flag = 0;
                last = stop[i];
                i++;
        }
        if(flag == 1)
                dijkstra(graph, src, dest);
        else {
                dijkstra(graph, last, dest);
                System.out.println("Total distance is "+ total);
        }
    }
}
```

CHAPTER 7:

RESULT

- Menu

```
Cities:
1.Bangalore
2.Mysore
3.Tumkur
4.Hassan
5.Davangere
6.Mandya
7.Shivamogga
8.Chikkamagluru
9.Ballari
10.Madikeri
11.Karwar
12.Mangalore
Enter your starting point
```

- Selecting cities for first time:

```
Enter your starting point
1
Enter your end point
12
Do you want to add stops?[1 for yes, 0 for no]
```

- Output with no extra stops:

```
Do you want to add stops?[1 for yes, 0 for no]
0

Bangalore -> Mangalore

Distance = 342

Bangalore->Hassan->Mangalore
```

- Output with extra stops:

```
Do you want to add stops?[1 for yes, 0 for no]
1
Cities:
1.Bangalore
2.Mysore
3.Tumkur
4.Hassan
5.Davangere
6.Mandya
7.Shivamogga
8.Chikkamagluru
9.Ballari
10.Madikeri
11.Karwar
12.Mangalore

Enter extra stop
5

Bangalore -> Davangere

Distance = 262

Bangalore->Tumkur->Davangere
Do you want to add stops?[1 for yes, 0 for no]
```

```
Bangalore->Tumkur->Davangere
Do you want to add stops?[1 for yes, 0 for no]
0

Davangere -> Mangalore

Distance = 513

Davangere->Tumkur->Hassan->Mangalore
Total distance is 775
```

CHAPTER 8:

TIME COMPLEXITY

- Time for visiting all vertices =O(V)

Time required for processing one vertex=O(V)

Time required for visiting and processing all the vertices = O(V)*O(V) =O(V^2)

So, the time complexity of Dijkstra's algorithm using adjacency matrix representation comes out to be O(V^2)

- The time complexity of Dijkstra's algorithm can be reduced to O((V+E)logV) using adjacency list representation of the graph and a min-heap to store the unvisited vertices, where E is the number of edges in the graph and V is the number of vertices in the graph.

Time for visiting all vertices =O(V+E)

Time required for processing one vertex=O(logV)

Time required for visiting and processing all the vertices = O(V+E)*O(logV) = O((V+E)logV)

CHAPTER 9:

APPLICATIONS

- Used to find the shortest distance between different cities.
- Used to find the shortest path to the destination passing through a specific city.
- Used in Digital Mapping Services in Google Maps.
- Used in Telephone Network.
- Used to designate file server.

CHAPTER 10:

CONCLUSION & FURTHER ENHANCEMENTS

10.1 Conclusion

- This helps to get the shortest path.
- Code can be optimized to get realtime location data between cities.

10.2 Further Enhancements

- It can be further enhanced to give the time taken to travel from one city to another.
- Can be enhanced to support a GUI.

10.3 References

- https://www.geeksforgeeks.org
- https://www.codechef.com
- https://www.techgig.com
- https://www.google.com/maps