RNS Institute of Technology
Department of Information Science and Engineering

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY (18CSL47)

SIMPLE NAVIGATION SYSTEM

Staff in Charge: Dr. Suresh L
Designation: Professor and HOD

Carried out by
Sameer Singh (1RN20IS131)
Sanjana Shenoy (1RN20IS137)
Sharanya RP (1RN20IS143)
Shreya M Bharadwaj (1RN20IS154)

ESTD : 2001
An Institute with a Difference

CONTENTS

- **Abstract**
- **Introduction**
- **Objective of the project**
- **Algorithm Design Technique**
- **Project Architecture**
- **Implementation**
- **Results**
- **Applications**
- **Conclusion & Future Enhancements**
- **References**

Abstract

- A navigation system is a computing system that aids in navigation
- It contains maps, which determines vehicle location via sensors or information
- It provides direction and distances

Introduction

- A simple navigation system provides information about the distances between any two cities
- Showing the route that takes you to your destination with the least amount of stops is vital

- A good navigation system must also allow you to add stops
- Optimized route which includes the desired stop can be calculated

Objective of the project

- Distance between two cities can be easily found
- Stops can be added to go through certain cities before going to desired destination
- Real life application of Dijkstra's algorithm

Algorithm Design Technique

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

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 ← 1 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
```

System Requirements

Hardware Requirements

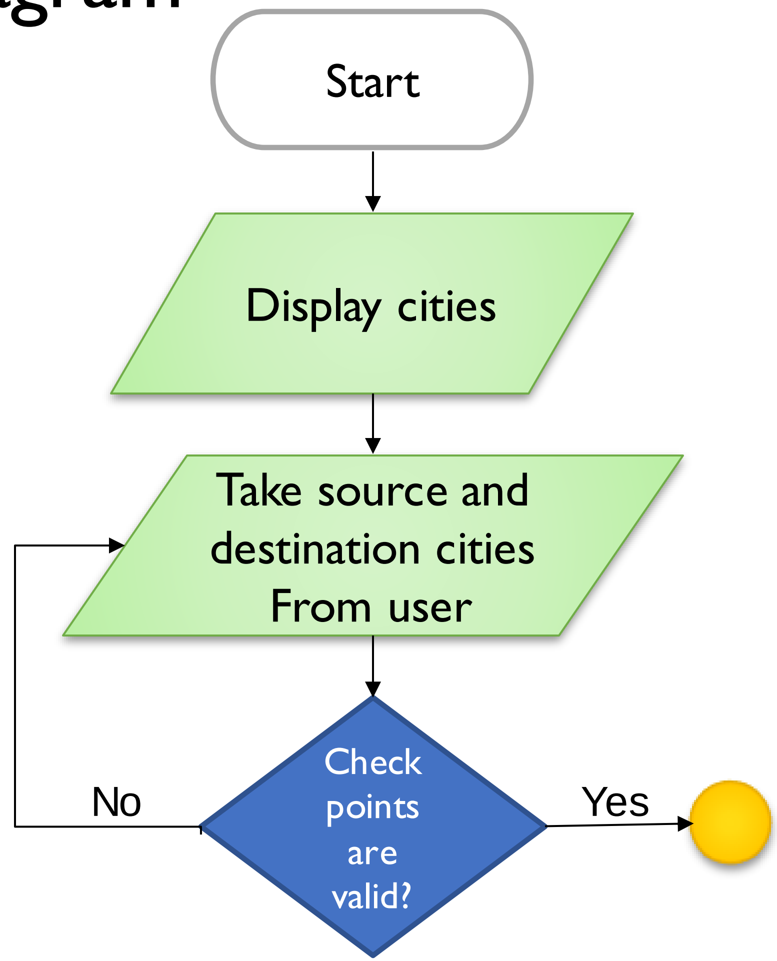
- **Processor:** Intel Core2 Quad @ 2.4Ghz on Windows® Vista 64-Bit / Windows® 7 64-Bit / Windows® 8 64-Bit / Windows® 8.1 64-Bit.
- **RAM:** 2GB of RAM
- **Memory:** 256GB Hard drive
- **Keyboard:** MS compatible keyboard
- **Mouse:** MS compatible mouse

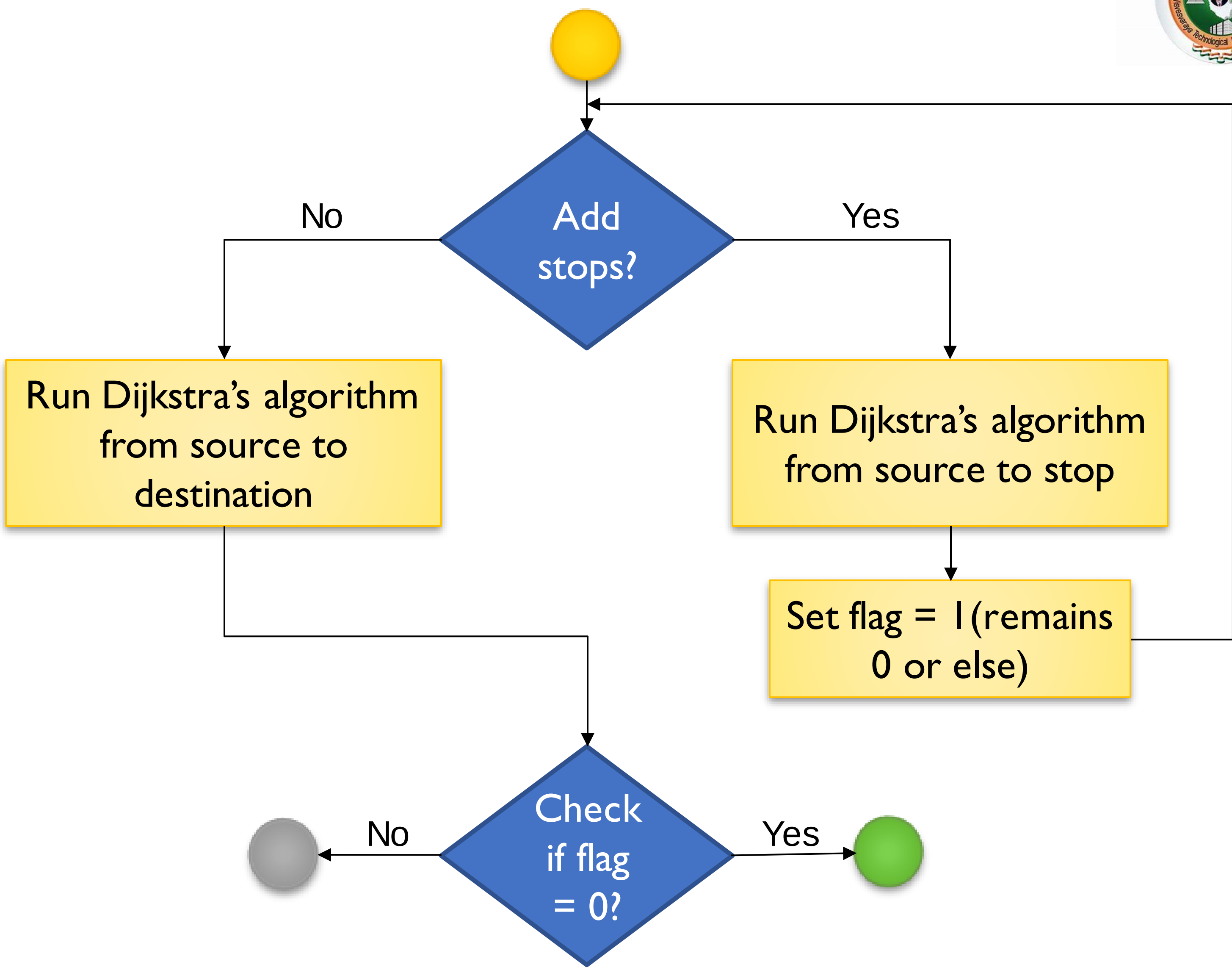
Software Requirements

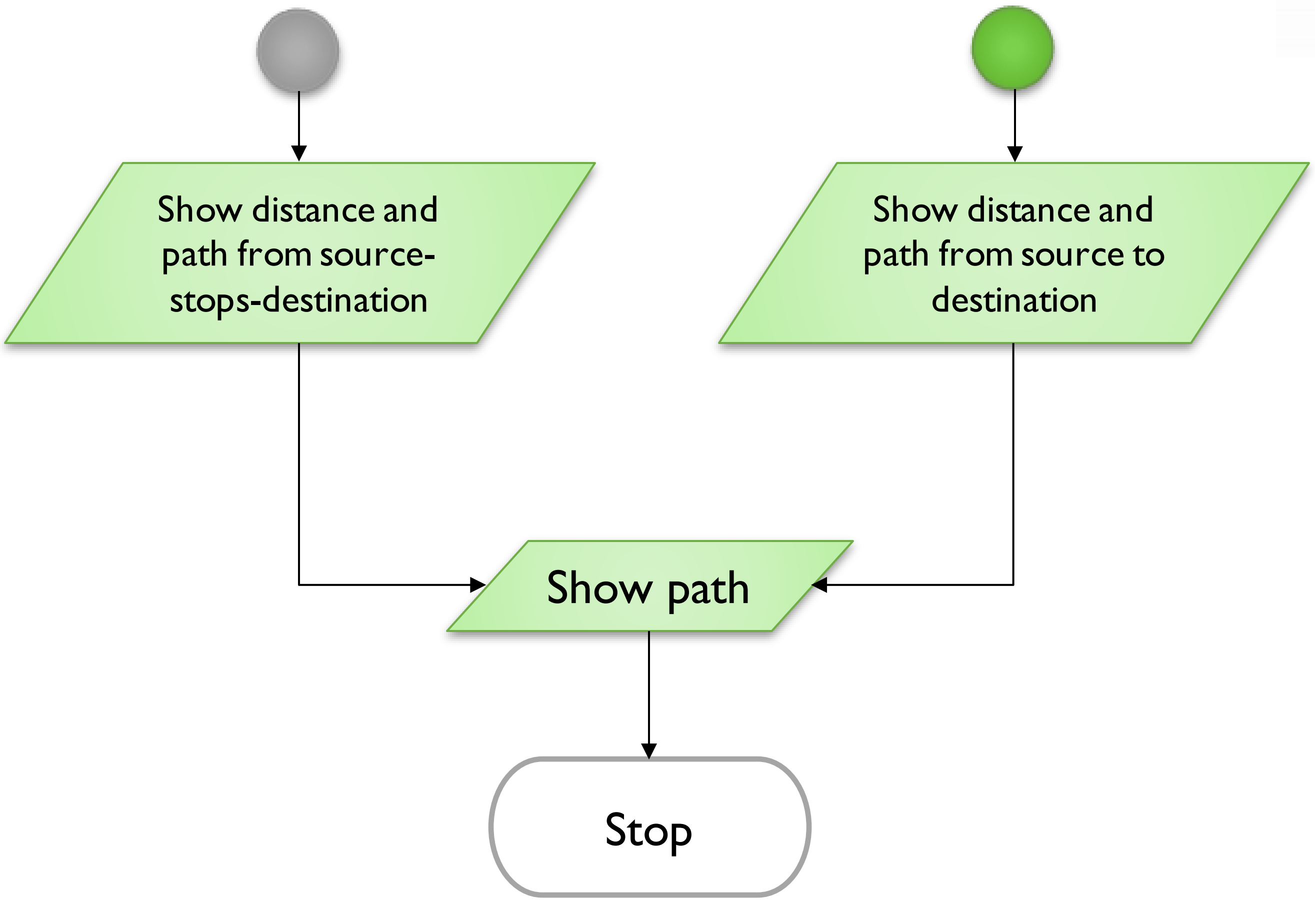
- **Operating system:** Windows® Vista 64-Bit / Windows® 7 64-Bit / Windows® 8 64-Bit / Windows® 8.1 64-Bit.
- **Front end Programming language :** Java
- **IDE :** Eclipse

Project Architecture

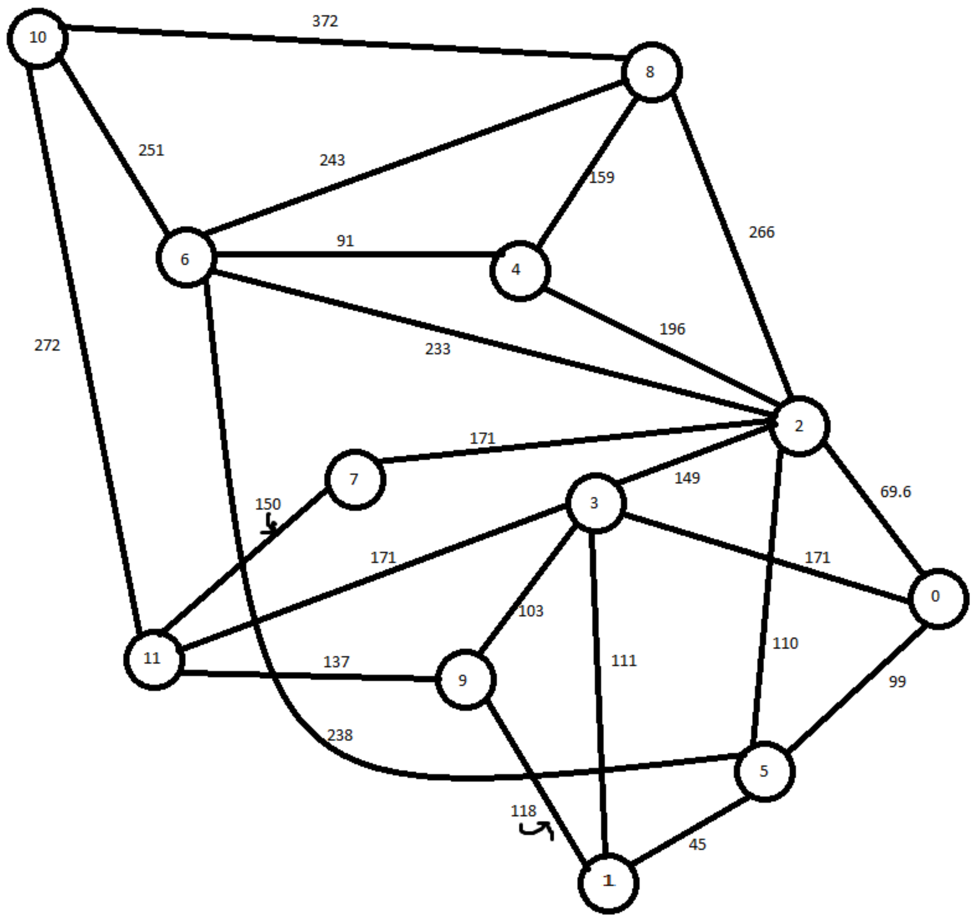
- Flow diagram




Add stops?
No
Yes
Run Dijkstra's algorithm from source to destination
Run Dijkstra's algorithm from source to stop
Set flag = 1(remains 0 or else)
Check if flag = 0?
No
Yes


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

Graph



10
372
8
251
243
159
266
6
91
4
196
272
233
2
171
7
149
69.6
150
3
171
171
0
103
110
11
137
9
111
99
238
5
118
45
1

Graph

- Legend

```
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
```

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.

Code snippets

- main():

```java
public static void main(String[] args)
{
    int[][] adjacencyMatrix = {
            {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;
    if(src == dest) {
        System.out.println("Invalid input!!\nEnter your end point");
        dest = scan.nextInt() ;
    }
```

Code snippets

```java
        while(ch != 0){
            System.out.println("Do you want to add stops?[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(adjacencyMatrix, src, stop[i]);
            else
                dijkstra(adjacencyMatrix, stop[i-1], stop[i]);
            flag = 0;
            last = stop[i];
            i++;
        }
        if(flag == 1)
            dijkstra(adjacencyMatrix, src, dest);
        else {
            dijkstra(adjacencyMatrix, last, dest);
            System.out.println("Total distance is "+ total);

    }
}
```

Code snippets

- cityNames(int vertex):

```
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;
}
```

Code snippets

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

```
private static void dijkstra(int[][] adjacencyMatrix, int startVertex, int endVertex) {
    int nVertices = adjacencyMatrix[0].length;

    int[] shortestDistances = new int[nVertices];

    boolean[] added = new boolean[nVertices];

    for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {
        shortestDistances[vertexIndex] = Integer.MAX_VALUE;
        added[vertexIndex] = false;
    }

    shortestDistances[startVertex] = 0;

    int[] parents = new int[nVertices];

    parents[startVertex] = NO_PARENT;

    for (int i = 1; i < nVertices; i++) {

        int nearestVertex = -1;
        int shortestDistance = Integer.MAX_VALUE;
        for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {
            if (!added[vertexIndex] && shortestDistances[vertexIndex] < shortestDistance) {
                nearestVertex = vertexIndex;
                shortestDistance = shortestDistances[vertexIndex];
            }
        }
```

Code snippets

```
            added[nearestVertex] = true;

            for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {
                int edgeDistance = adjacencyMatrix[nearestVertex][vertexIndex];

                if (edgeDistance > 0 && ((shortestDistance + edgeDistance) < shortestDistances[vertexIndex])) {
                    parents[vertexIndex] = nearestVertex;
                    shortestDistances[vertexIndex] = shortestDistance + edgeDistance;
                }
            }
        }
```

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

```
private static void printSolution(int startVertex, int endVertex, int[] distances, int[] parents) {
    int nVertices = distances.length;
    if (endVertex != startVertex) {
        System.out.println(String.format("\n%s -> %s\n", cityName(startVertex), cityName(endVertex)));
        System.out.println(String.format("Distance = %d\n", distances[endVertex]));
        printPath(endVertex, parents, endVertex);
        System.out.print("\n");
    }
    total += distances[endVertex];
}
```

Code snippets

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

```
private static void printPath(int currentVertex, int[] parents, int endVertex) {

    if (currentVertex == NO_PARENT) {
        return;
    }
    printPath(parents[currentVertex], parents, endVertex);
    if (currentVertex == endVertex) {
        System.out.print(String.format("%s", cityName(currentVertex)));
    } else
        System.out.print(String.format("%s->", cityName(currentVertex)));
}
```

Results

- 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
```

Results

- 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
```

Results

- 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]
```

Results

- Output with extra stops(Contd.):

```
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
```

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)

Application

- 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 Google Maps

Conclusion and future enhancements

- This helps to get the shortest path
- It can be further enhanced to give the time taken to travel from one city to another
- Code can be optimized to get realtime location data between cities
- Can be enhanced to support a GUI

References

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