Smita Koralahalli Channabasappa skoralah@uncc.edu
Student ID: 801008161

Project 2-Algorithm Implementation
ITCS-6114/002 Data Structures and Algorithms

Based on the mileage map provided design and implement algorithms to
1. Find a shortest path tree from Charlotte to the five major cities in Florida: Jacksonville, Tallahassee, Orlando, Tampa, & Miami (the tree may involve other cities).

Answer:
Employed Dijkstra algorithm to find the shortest path tree from Charlotte to all the cities.
The algorithm works as follows:

- For a given source vertex it finds the path with the lowest cost between that vertex and every other vertex.
- When the destined vertex to which the shortest path to be found is reached the algorithm is stopped.
- In my implementation the vertices of the graph represent cities and edge path costs represent distances between pairs of cities connected by a direct road.
- Thus Dijkstra's algorithm finds the shortest route between one city and all other cities.

Project Implementation:

- At the beginning all the vertices are initialized to infinity and source vertex to zero.
- Priority queue is implemented where priority is given for the vertex with minimum weight and its neighbors are explored.
- If the weighted value of the source vertex and its edge is greater than neighbor target vertex its value is updated with target value and added in the queue.
- The original value is simultaneously removed from queue.
- This action is performed until the queue is empty by **poll()** method which is used to retrieve and remove the head of this queue, or returns null if this queue is empty.

Switch case is implemented with 1 indicating distance between Charlotte to all the five cities which is the implementation to part 1.

```
1
Distance to Jacksonville from Charlotte : 388.0
Path: [Charlotte, Columbia, Savannah, Jacksonville]


Distance to Tallahassee from Charlotte: 474.0
Path:[Charlotte, Columbia, Augusta, Tifton, Tallahassee]


Distance to Orlando from Charlotte: 533.0
Path: [Charlotte, Columbia, Savannah, Jacksonville, DaytonaBeach, Orlando]


Distance to Tampa from Charlotte: 587.0
Path: [Charlotte, Columbia, Savannah, Jacksonville, JctIntUS, Tampa]


Distance to Miami from Charlotte: 740.0
Path: [Charlotte, Columbia, Savannah, Jacksonville, DaytonaBeach, Cocoa, VeroBea
ch, Miami]
```

Snapshot 1 : Distance from Charlotte to all 5 cities

Smita Koralahalli Channabasappa skoralah@uncc.edu
Student ID: 801008161

Project 2-Algorithm Implementation
ITCS-6114/002 Data Structures and Algorithms

Figure 1 depicts the shortest distance with the corresponding path to reach the destined city for all five cities.

In a similar manner distance between corresponding cities are found out with
2 indicating shortest distance from Jacksonville to remaining 4 cities
3 indicating shortest distance from Tallahassee to remaining 3 cities
4 indicating shortest distance from Orlando to remaining 2 cities
5 indicating shortest distance from Tampa to the final remaining city Miami.

```
skoralah@smita:/media/skoralah/Smita/data structures/proj2_final$ java Dijkstra2
Enter a number between 1 to 5
1       Charlotte to all cities
2       Jacksonville to all cities and so on..
2
Distance to Tallahassee from Jacksonville: 166.0
Path:[Jacksonville, Tallahassee]


Distance to Orlando from Jacksonville: 145.0
Path: [Jacksonville, DaytonaBeach, Orlando]


Distance to Tampa from Jacksonville: 199.0
Path: [Jacksonville, JctIntUS, Tampa]


Distance to Miami from Jacksonville: 352.0
Path: [Jacksonville, DaytonaBeach, Cocoa, VeroBeach, Miami]


skoralah@smita:/media/skoralah/Smita/data structures/proj2_final$
```

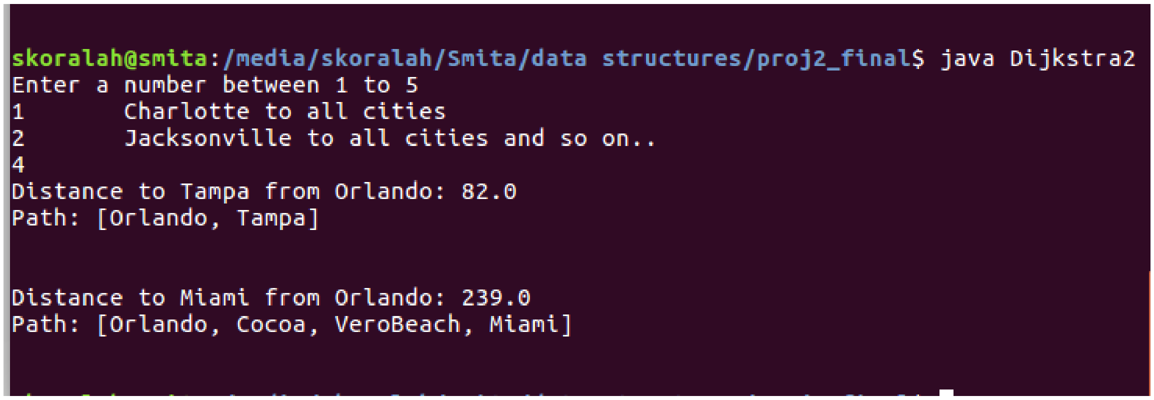
Snapshot 2: Distance from Jacksonville to 4 cities.

```
skoralah@smita:/media/skoralah/Smita/data structures/proj2_final$ java Dijkstra2
Enter a number between 1 to 5
1       Charlotte to all cities
2       Jacksonville to all cities and so on..
3
Distance to Orlando from Tallahassee: 245.0
Path: [Tallahassee, JctIntUS, Orlando]


Distance to Tampa from Tallahassee: 244.0
Path: [Tallahassee, Tampa]


Distance to Miami from Tallahassee: 472.0
Path: [Tallahassee, JctIntUS, Miami]


skoralah@smita:/media/skoralah/Smita/data structures/proj2_final$
```

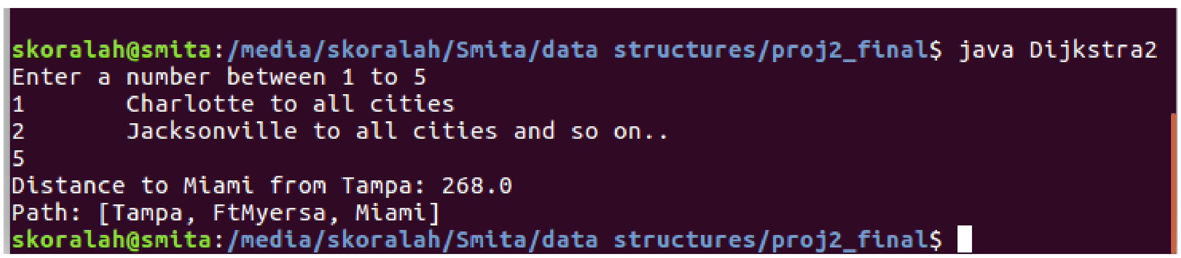
Snapshot 3: Distance from Tallahassee to 3 cities.

Smita Koralahalli Channabasappa skoralah@uncc.edu
Student ID: 801008161

Project 2-Algorithm Implementation
ITCS-6114/002 Data Structures and Algorithms



Snapshot 4: Distance from Orlando to Tampa and Miami.



Snapshot 5: Distance from Tampa to Miami.

In a matrix form :

Cities	**Charlotte**	**Jacksonville**	**Tallahassee**	**Orlando**	**Tampa**	**Miami**
Charlotte	0	388	474	533	587	740
Jacksonville	388	0	166	145	199	352
Tallahassee	474	166	0	245	244	472
Orlando	533	145	245	0	82	239
Tampa	587	199	244	82	0	268
Miami	740	352	472	239	268	0

2. Find a shortest distance trip from Charlotte to visit all the above 5 cities in Florida then back to Charlotte. You are allowed to visit a city more than once and to drive on a same highway more than once.

Smita Koralahalli Channabasappa skoralah@uncc.edu
Student ID: 801008161

Project 2-Algorithm Implementation
ITCS-6114/002 Data Structures and Algorithms

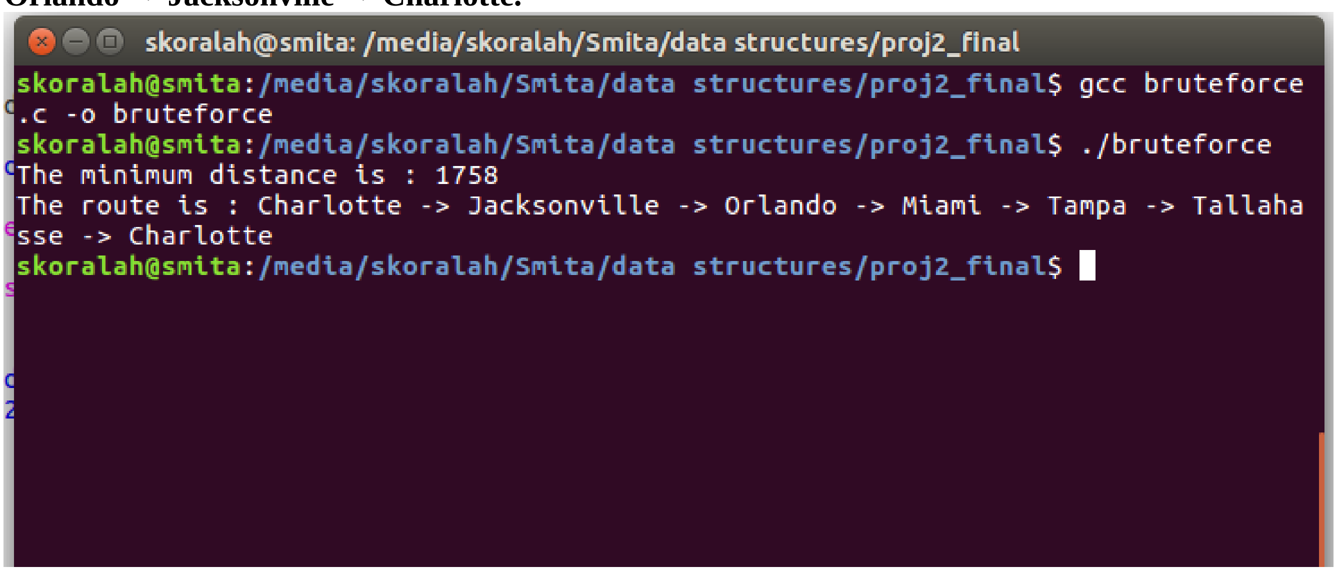
Implementation :
In order to find the shortest distance trip there are different possible solutions. Since there are 5 different cities from Charlotte there are 5! = 120 different possibilities to traverse from Charlotte and arrive back to Charlotte.
The task becomes tedious as number of cities goes on increasing. To overcome this I have implemented brute force algorithm which basically generates all possible routes with the given input combination.

The approach employed is:

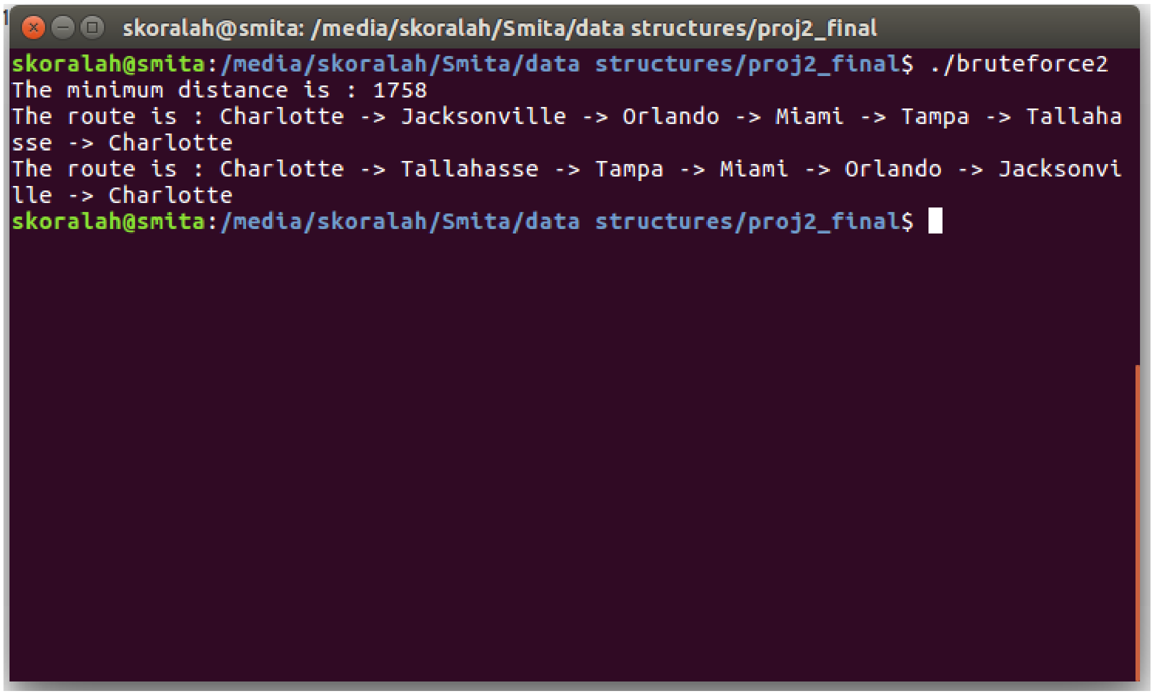
- Input all the values that is the distances obtained from Dijkstra algorithm between 5 cities in terms of 5*5 matrix. Distance to same city is initialized to zero.
- A dummy string “12345” is taken as input which corresponds to 5 different cities and given as input to permute function.
- Permute is creating different permutations of string "12345".
- Then used each of those permutation to calculate distance for that permutation and proceed.
- Thus all 120 possible permutations are done which in turn calls swap function for internal swap for example swapping numbers 1 and 2 in string 12345 making 21345.
- The distance for each route is been calculated accordingly in permute function through min_dist and then compared to obtain the minimum distance.
- Since the routes are in terms of numbers toint function basically computes the corresponding city for the number by doing some mathematical calculations indexing each number to each city.

Through this the minimum distance obtained is 1758 and the route from Charlotte → Jacksonville → Orlando → Miami → Tampa → Tallahassee → Charlotte

The same route also exists viceversa that is Charlotte → Tallahassee → Tampa → Miami → Orlando → Jacksonville → Charlotte.



Snapshot 6: Shortest distance trip from Charlotte to all cities and back to Charlotte



Snapshot 7: Shortest distance trip from Charlotte to all cities and back to Charlotte and other way round.

Appendices
Source Code:

Program [1] – Dijkstra Algorithm

```
import java.util.PriorityQueue;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

class Vertex implements Comparable<Vertex>
{
   public final String name;
   public Edge[] adjacencies;      //edge class for distance+weight
   public double minDistance = Double.POSITIVE_INFINITY; //at the beginning initialized to infinity
   public Vertex previous;
   public Vertex(String argName) { name = argName; }  //constructor declaration
   public String toString() { return name; }
   public int compareTo(Vertex other)
   {
```

Smita Koralahalli Channabasappa skoralah@uncc.edu
Student ID: 801008161

Project 2-Algorithm Implementation
ITCS-6114/002 Data Structures and Algorithms

```
            return Double.compare(minDistance, other.minDistance);  //fn to compare distance
        }

    }


    class Edge
    {
        public final Vertex target;
        public final double weight;
        public Edge(Vertex argTarget, double argWeight)
        { target = argTarget; weight = argWeight; }   //constructor
    }

    /********Class declaration to compute shortest paths with source initialized to zero******
    *******Then its neighbours are polled if queue is not empty and replaced recursively****
    ******if the destined value is less than source value******************************/
    public class Dijkstra2
    {
        public static void computePaths(Vertex source)
        {
            source.minDistance = 0.;   //source distance set to zero


          // create priority queue
            PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();

        // insert values in the queue
        vertexQueue.add(source);        //initially the source with 0(minimum) is added to queue

    /******enter the queue and poll through all values********/
    /******if queue is not empty******************************/
        while (!vertexQueue.isEmpty()) {
            Vertex u = vertexQueue.poll();

                // Visit each edge exiting u
                for (Edge e : u.adjacencies)
                {
                    Vertex v = e.target;          //final destination to be reached
                    double weight = e.weight;     //weight of the edge
                    double distanceThroughU = u.minDistance + weight; // distance of source + its weight
            if (distanceThroughU < v.minDistance) {
                vertexQueue.remove(v);
```

Smita Koralahalli Channabasappa skoralah@uncc.edu
Student ID: 801008161

Project 2-Algorithm Implementation
ITCS-6114/002 Data Structures and Algorithms

```
/*******If source distance weight< adjacent destination******/
/*******replace its value with source weighted value*********/
            v.minDistance = distanceThroughU ;
            v.previous = u;                //mark its parent
            vertexQueue.add(v);            //add the next least neighbour value to queue
        }
            }
        }
    }

/*********Simple ArrayList to store the names of cities in the shortest path from*****
********source to destination*******************************************/
    public static List<Vertex> getShortestPathTo(Vertex target)
    {
        List<Vertex> path = new ArrayList<Vertex>();
        for (Vertex vertex = target; vertex != null; vertex = vertex.previous)
            path.add(vertex);

//as queue is first in first out the source city will be removed first so collections.reverse is called
 //to display in the order from source to destination//
        Collections.reverse(path);
        return path;
    }

    public static void main(String[] args)
    {
        // mark all the vertices
        Vertex Charlotte = new Vertex("Charlotte");
        Vertex Asheville = new Vertex("Asheville");
        Vertex Willmington = new Vertex("Willmington");
        Vertex Atlanta = new Vertex("Atlanta");
        Vertex Columbia = new Vertex("Columbia");
        Vertex Knoxville = new Vertex("Knoxville");
        Vertex Chattanooga = new Vertex("Chattanooga");
        Vertex Florence = new Vertex("Florence");
        Vertex Augusta = new Vertex("Augusta");
        Vertex Charleston = new Vertex("Charleston");
        Vertex Savannah = new Vertex("Savannah");
        Vertex Raleigh = new Vertex("Raleigh");
        Vertex Unknown = new Vertex("Unknown");
        Vertex Birmingham = new Vertex("Birmingham");
        Vertex Montgomery = new Vertex("Montgomery");
        Vertex Tifton = new Vertex("Tifton");
        Vertex Mobile = new Vertex("Mobile");
```

```
    Vertex Tallahassee = new Vertex("Tallahassee");
    Vertex Jacksonville = new Vertex("Jacksonville");
    Vertex DaytonaBeach = new Vertex("DaytonaBeach");
    Vertex JctIntUS = new Vertex("JctIntUS");
    Vertex Orlando = new Vertex("Orlando");
    Vertex Tampa = new Vertex("Tampa");
    Vertex Cocoa = new Vertex("Cocoa");
    Vertex VeroBeach = new Vertex("VeroBeach");
    Vertex FtMyersa = new Vertex("FtMyersa");
    Vertex Miami = new Vertex("Miami");
    Vertex KeyWest = new Vertex("KeyWest");

  // set the edges and weight
  Charlotte.adjacencies = new Edge[]
  {
     new Edge(Asheville, 112),
     new Edge(Willmington, 203),
     new Edge(Atlanta, 240),
     new Edge(Columbia, 94)
  };

        Asheville.adjacencies = new Edge[]
        {
            new Edge(Charlotte, 112),
            new Edge(Chattanooga, 195),
            new Edge(Atlanta, 205),
            new Edge(Augusta, 172),
            new Edge(Columbia, 159)
        };

        Willmington.adjacencies = new Edge[]
        {
            new Edge(Charlotte, 203),
            new Edge(Florence, 119),
            new Edge(Charleston, 167),
            new Edge(Raleigh, 124)
        };

        Atlanta.adjacencies = new Edge[]
        {
            new Edge(Charlotte, 240),
            new Edge(Asheville, 205),
            new Edge(Chattanooga, 113),
            new Edge(Birmingham, 150),
```

```
        new Edge(Montgomery, 168),
        new Edge(Tifton, 182),
        new Edge(Savannah, 255),
        new Edge(Augusta, 150),
        new Edge(Columbia, 214),
        new Edge(Knoxville, 224)
};

Columbia.adjacencies = new Edge[]
{
        new Edge(Charlotte, 94),
        new Edge(Asheville, 159),
        new Edge(Atlanta, 214),
        new Edge(Augusta, 69),
        new Edge(Savannah, 158),
        new Edge(Charleston, 113),
        new Edge(Florence, 80),
        new Edge(Raleigh, 205)
};

Knoxville.adjacencies = new Edge[]
{
        new Edge(Atlanta, 224),
        new Edge(Unknown, 178),
        new Edge(Chattanooga, 111)
};

Unknown.adjacencies = new Edge[]
{
        new Edge(Knoxville, 178),
        new Edge(Chattanooga, 129),
        new Edge(Birmingham, 194)
};

Chattanooga.adjacencies = new Edge[]
{
        new Edge(Asheville, 195),
        new Edge(Atlanta, 113),
        new Edge(Knoxville, 111),
        new Edge(Unknown, 129),
        new Edge(Birmingham, 145)
};

Florence.adjacencies = new Edge[]
```

```
{
        new Edge(Willmington, 119),
        new Edge(Columbia, 80),
        new Edge(Raleigh, 147),
        new Edge(Savannah, 172)
};

Augusta.adjacencies = new Edge[]
{
        new Edge(Asheville, 172),
        new Edge(Atlanta, 150),
        new Edge(Columbia, 69),
        new Edge(Tifton, 222),
        new Edge(Jacksonville, 260),
        new Edge(Savannah, 124),
        new Edge(Charleston, 139)
};

Charleston.adjacencies = new Edge[]
{
        new Edge(Willmington, 167),
        new Edge(Columbia, 113),
        new Edge(Augusta, 139),
        new Edge(Savannah, 106)
};

Raleigh.adjacencies = new Edge[]
{
        new Edge(Willmington, 124),
        new Edge(Columbia, 205),
        new Edge(Florence, 147)
};

Savannah.adjacencies = new Edge[]
{
        new Edge(Atlanta, 255),
        new Edge(Columbia, 158),
        new Edge(Florence, 172),
        new Edge(Augusta, 124),
        new Edge(Charleston, 106),
        new Edge(Montgomery, 354),
        new Edge(Tallahassee, 244),
        new Edge(Jacksonville, 136)
};
```

```
        Birmingham.adjacencies = new Edge[]
        {
                new Edge(Atlanta, 150),
                new Edge(Chattanooga, 145),
                new Edge(Unknown, 194),
                new Edge(Mobile, 240),
                new Edge(Montgomery, 91),
    new Edge(Tifton, 286)
    };

        Montgomery.adjacencies = new Edge[]
        {
                new Edge(Atlanta, 168),
                new Edge(Savannah, 354),
                new Edge(Birmingham, 91),
                new Edge(Mobile, 172),
                new Edge(Tallahassee, 202),
                new Edge(Tifton, 200)
        };

        Tifton.adjacencies = new Edge[]
        {
                new Edge(Atlanta, 182),
                new Edge(Augusta, 222),
                new Edge(Birmingham, 286),
                new Edge(Montgomery, 200),
                new Edge(Tallahassee, 89),
                new Edge(JctIntUS, 185),
                new Edge(Jacksonville, 149)
        };

        Mobile.adjacencies = new Edge[]
        {
                new Edge(Birmingham, 240),
                new Edge(Montgomery, 172),
                new Edge(Tallahassee, 244)
        };

        Tallahassee.adjacencies = new Edge[]
        {
                new Edge(Savannah, 244),
                new Edge(Montgomery, 202),
                new Edge(Tifton, 89),
```

```
        new Edge(Mobile, 244),
        new Edge(Jacksonville, 166),
        new Edge(JctIntUS, 170),
        new Edge(Tampa, 244)
};

Jacksonville.adjacencies = new Edge[]
{
        new Edge(Augusta, 260),
        new Edge(Savannah, 136),
        new Edge(Tifton, 149),
        new Edge(Tallahassee, 166),
        new Edge(JctIntUS, 105),
        new Edge(DaytonaBeach, 91)
};

DaytonaBeach.adjacencies = new Edge[]
{
        new Edge(Jacksonville, 91),
        new Edge(Cocoa, 66),
        new Edge(Orlando, 54)
};

JctIntUS.adjacencies = new Edge[]
{
        new Edge(Tifton, 185),
        new Edge(Tallahassee, 170),
        new Edge(Jacksonville, 105),
        new Edge(Tampa, 94),
        new Edge(Miami, 302),
        new Edge(Orlando, 75)
};

Orlando.adjacencies = new Edge[]
{
        new Edge(DaytonaBeach, 54),
        new Edge(JctIntUS, 75),
        new Edge(Tampa, 82),
        new Edge(Cocoa, 44)
};

Tampa.adjacencies = new Edge[]
{
        new Edge(Tallahassee, 244),
```

```
            new Edge(JctIntUS, 94),
            new Edge(Orlando, 82),
            new Edge(VeroBeach, 137),
            new Edge(FtMyersa, 125)
        };

        Cocoa.adjacencies = new Edge[]
        {
            new Edge(DaytonaBeach, 66),
            new Edge(Orlando, 44),
            new Edge(VeroBeach, 55)
        };

        VeroBeach.adjacencies = new Edge[]
        {
            new Edge(Miami, 140),
            new Edge(Tampa, 137),
            new Edge(Cocoa, 55)
        };

        FtMyersa.adjacencies = new Edge[]
        {
            new Edge(Tampa, 125),
            new Edge(Miami, 143)
        };

        Miami.adjacencies = new Edge[]
        {
            new Edge(JctIntUS, 302),
            new Edge(VeroBeach, 140),
            new Edge(FtMyersa, 143),
            new Edge(KeyWest, 151)
        };

        KeyWest.adjacencies = new Edge[]
        {
            new Edge(Miami, 151)
        };


    Scanner sc = new Scanner(System.in);
  System.out.println("Enter a number between 1 to 5" + '\n' +"1" +'\t' +"Charlotte to all cities" +'\n'
+"2"+'\t' +"Jacksonville to all cities and so on..");
    int city = sc.nextInt();
```

```
/*Distance from CHARLOTTE to all 5 cities*/
switch(city)

{
case 1:
computePaths(Charlotte); // run Dijkstra
System.out.println("Distance to " + Jacksonville + " from " + Charlotte + " : " +
Jacksonville.minDistance);
List<Vertex> patha = getShortestPathTo(Jacksonville);
System.out.println("Path: " + patha + '\n' + '\n');

computePaths(Charlotte); // run Dijkstra
System.out.println("Distance to " + Tallahassee + " from " + Charlotte + ": " +
Tallahassee.minDistance);
List<Vertex> pathb = getShortestPathTo(Tallahassee);
System.out.println("Path:" + pathb + '\n' + '\n');

computePaths(Charlotte); // run Dijkstra
System.out.println("Distance to " + Orlando + " from " + Charlotte + ": " + Orlando.minDistance);
List<Vertex> pathc = getShortestPathTo(Orlando);
System.out.println("Path: " + pathc + '\n' + '\n');

computePaths(Charlotte); // run Dijkstra
System.out.println("Distance to " + Tampa + " from " + Charlotte + ": " + Tampa.minDistance);
List<Vertex> pathd = getShortestPathTo(Tampa);
System.out.println("Path: " + pathd + '\n' + '\n');

computePaths(Charlotte); // run Dijkstra
System.out.println("Distance to " + Miami + " from " + Charlotte + ": " + Miami.minDistance);
List<Vertex> pathe = getShortestPathTo(Miami);
System.out.println("Path: " + pathe + '\n' + '\n');
break;

/*Distance from JACKSONVILLE to remaining 4 cities*/
case 2:
computePaths(Jacksonville); // run Dijkstra
System.out.println("Distance to " + Tallahassee + " from " + Jacksonville + ": " +
Tallahassee.minDistance);
List<Vertex> pathf = getShortestPathTo(Tallahassee);
System.out.println("Path:" + pathf + '\n' + '\n');

computePaths(Jacksonville); // run Dijkstra
System.out.println("Distance to " + Orlando + " from " + Jacksonville + ": " +
Orlando.minDistance);
```

```
List<Vertex> pathg = getShortestPathTo(Orlando);
System.out.println("Path: " + pathg + '\n' + '\n');

   computePaths(Jacksonville); // run Dijkstra
System.out.println("Distance to " + Tampa + " from " + Jacksonville + ": " + Tampa.minDistance);
List<Vertex> pathh = getShortestPathTo(Tampa);
System.out.println("Path: " + pathh + '\n' + '\n');

   computePaths( Jacksonville); // run Dijkstra
System.out.println("Distance to " + Miami + " from " + Jacksonville + ": " + Miami.minDistance);
List<Vertex> pathi = getShortestPathTo(Miami);
System.out.println("Path: " + pathi + '\n' + '\n');
   break;

   /*Distance from TALLAHASSEE to remaining 3 cities*/
   case 3:
   computePaths(Tallahassee); // run Dijkstra
System.out.println("Distance to " + Orlando + " from " + Tallahassee + ": " +
Orlando.minDistance);
List<Vertex> pathj = getShortestPathTo(Orlando);
System.out.println("Path: " + pathj + '\n' + '\n');

   computePaths(Tallahassee); // run Dijkstra
System.out.println("Distance to " + Tampa + " from " + Tallahassee + ": " + Tampa.minDistance);
List<Vertex> pathk = getShortestPathTo(Tampa);
System.out.println("Path: " + pathk + '\n' + '\n');

   computePaths(Tallahassee); // run Dijkstra
System.out.println("Distance to " + Miami + " from " + Tallahassee + ": " + Miami.minDistance);
List<Vertex> pathl = getShortestPathTo(Miami);
System.out.println("Path: " + pathl + '\n' + '\n');
   break;
   /*Distance from ORLANDO to remaining 2 cities*/

   case 4:
   computePaths(Orlando); // run Dijkstra
System.out.println("Distance to " + Tampa + " from " + Orlando  + ": " + Tampa.minDistance);
List<Vertex> pathm = getShortestPathTo(Tampa);
System.out.println("Path: " + pathm + '\n' + '\n');

   computePaths(Orlando); // run Dijkstra
System.out.println("Distance to " + Miami + " from " + Orlando + ": " + Miami.minDistance);
List<Vertex> pathn = getShortestPathTo(Miami);
System.out.println("Path: " + pathn + '\n' + '\n');
```

```
            break;

            /*Distance from TAMPA to remaining 1 city*/
            case 5:
            computePaths(Tampa); // run Dijkstra
          System.out.println("Distance to " + Miami + " from " + Tampa + ": " + Miami.minDistance);
          List<Vertex> patho = getShortestPathTo(Miami);
          System.out.println("Path: " + patho);
            break;

            default:
            System.out.println("Please enter valid case number between 1 to 5");
            break;
            }
    }
    }
```

Program [2] – Brute Force Method

```
#include <stdio.h>
#include <stdlib.h>

/*******swap function*******/
void swap(char *x, char *y)
{
   char temp;
   temp = *x;
   *x = *y;
   *y = temp;
}

/****Ascii to integer conversion for cities initializing
*****from 0 and so on**********************************/
int toint(char* a)
{
  int i = 0;
  int val = 0;
  for(i = 0 ; i < 5 ; i++)      //all cities except charlotte from index 0
  {
    val = val*10;
    //convert each character in a string and add up to the value
    val += (a[i] - '0' );
```

```
  }
  return val;
}

/****Recursively computed all 5! permutations and comparison of all distances****/
/****to find the least distance*****/
void permute(char *a, int l, int r, int** arr, int* min_dist, int* ans){

   int i,j,k;
   int dist = 0;
   if (l == r){              //same city initialized to zero
      k = 0;
      for (i = 0 ; i < 5 ; i++){
        j = a[i] - '0';
        dist = dist + arr[k][j];     //recursive distance computation
        k = j;
      }
      dist += arr[0][k];

      if(dist < *min_dist){                 // recursively compare with min_dist

        *ans = toint(a);
        *min_dist = dist;
      }
   }

   else
   {
       for (i = l; i <= r; i++)                  //if not less then swap internally and call permute again
       {
          swap((a+l), (a+i));
          permute(a, l+1, r , arr, min_dist, ans);
          swap((a+l), (a+i));                     //backtrack
       }
   }
}

int main()
{

    int** arr = (int**)malloc(6 * sizeof(int*));

    int i,j;
```

```
 for(i = 0 ; i < 6 ; i++){
   arr[i] = (int*)malloc(6* sizeof(int));
 }

 for(i=0;i<6;i++){
   arr[i][i] = 0;
 }

/****Cost matrix is initialized******/
/** 0 : Charlotte*****/
/** 1 : Jacksonville**/
/** 2 : Tallahasse****/
/** 3 : Orlando*******/
/** 4 : Tampa*********/
/** 5 : Miami*********/

/****Charlotte to all cities*********/
 arr[0][1] = 388;
 arr[0][2] = 474;
 arr[0][3] = 533;
 arr[0][4] = 587;
 arr[0][5] = 740;

/****Jacksonville to all cities*********/
 arr[1][0] = 388;
 arr[1][2] = 166;
 arr[1][3] = 145;
 arr[1][4] = 199;
 arr[1][5] = 352;

/****Tallahasse to all cities*********/
 arr[2][0] = 474;
 arr[2][1] = 166;
 arr[2][3] = 245;
 arr[2][4] = 244;
 arr[2][5] = 472;

/****Orlando to all cities*********/
 arr[3][0] = 533;
 arr[3][1] = 145;
 arr[3][2] = 245;
 arr[3][4] = 82;
 arr[3][5] = 239;
```

```
/****Tampa to all cities*********/
  arr[4][0] = 587;
  arr[4][1] = 199;
  arr[4][2] = 244;
  arr[4][3] = 82;
  arr[4][5] = 268;

/****Miami to all cities*********/
  arr[5][0] = 740;
  arr[5][1] = 352;
  arr[5][2] = 472;
  arr[5][3] = 239;
  arr[5][4] = 268;

  char str[] = "12345"; // string in initialized with dummy value
  int ans = 12345;      // dummy
  int min_dist = 10000; //dummy


  permute(str, 0, 4, arr , &min_dist, &ans);

 // printf("%d ---- %d \n", min_dist, ans);

  char names[6][15] = {"Charlotte" , "Jacksonville" , "Tallahasse", "Orlando" , "Tampa" , "Miami"};

  printf("The minimum distance is : %d \n", min_dist);


 /**Extraction of integers to index names of places in the names array*****/
 /**ans=12345 initialized so e=12345%10=5=miami and so on*****************/
  int a, b,c,d,e;
  e = ans%10;      //miami
  ans = ans/10;
  d = ans%10;      //tampa
  ans = ans/10;
  c = ans%10;      //orlando
  ans = ans/10;
  b = ans%10;      //tallahasse
  ans = ans/10;
  a = ans%10;      //jacksonville
  //printf("%d %d %d %d %d \n", a,b,c,d,e);
  printf("The route is : Charlotte -> %s -> %s -> %s -> %s -> %s -> Charlotte \n", names[a] , names[b]
, names[c] , names[d], names[e]);
  return 0;
```

```
}
```