
Name: Atharva Paliwal
Roll No: 40

Problem Statement:

A traveling salesman is getting ready for a big sales tour. Starting at his hometown, suitcase in hand, he will conduct a journey in which each of his target cities is visited exactly once before he returns home. Given the pairwise distances between cities, what is the best order in which to visit them, to minimize the overall distance traveled?

CODE:

```
package daa_lab;
/**
 *
 * @author Atharva Paliwal
 */

import java.util.ArrayList;
import java.util.Arrays;
import java.util.*;

public class Daa_Lab {
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int start=0;
        System.out.println("Enter the number of vertices: ");
        int v = sc.nextInt();
        System.out.println("Enter the distance matrix:");
        int[][] graph = new int[v][v];
```

```
        for(int i=0;i<v;i++)
            for(int j=0;j<v;j++){
                System.out.print("City "+(i+1)+" to City "+(j+1)+" = ");
                graph[i][j]=sc.nextInt();
            }

        for(int i=0; i < v;i++){
            long startTime = System.nanoTime();
            TSP t = new TSP(graph,v,start);
            start = i;
            t.cost(t,start);
            t.print(t);
            long endTime = System.nanoTime();
            long timeElapsed = (endTime - startTime);
            System.out.println("Time required : " +timeElapsed + " ns");
        }
    }
}

class TSP {

    int[][] graph;
    int startCity;
    int vertex = 0;
    int cost = 0;
    int[] visited;
    Integer path[] = new Integer[vertex+1];
    public TSP(int[][] graph, int vertex,int startCity){
        this.graph = graph;
        this.vertex = vertex;
```

```
    this.startCity = startCity;

    visited = new int[vertex+1];
    for(int i=0 ; i<vertex;i++){
        visited[i]=0;
    }
}

public void cost(TSP t,int startCity){
    int nextCity = 99999999;
    visited[startCity] = 1;
    t.path = add(t.path,startCity+1);
    nextCity = t.nearestCity(startCity);
    if(nextCity == 99999999){
        nextCity =0;
        cost += graph[startCity][nextCity];
        return;
    }
    t.cost(t,nextCity);
}

int nearestCity(int city){
    int nextCity = 99999999;
    int  minCost = 999999999;
    int temp = 0;
    for(int i = 0; i < vertex;i++){
        if(graph[city][i]!=0 && visited[i]==0){
            if(graph[city][i]+graph[i][city] < minCost){
                minCost = graph[i][0] + graph[city][i];
                temp = graph[city][i];
```

```
                nextCity = i;
            }
        }
    }
    if(minCost != 999999999){
        cost += temp;
    }
    return nextCity;
}

public Integer[] add(Integer arr[], int x){
    List<Integer> arrlist;
    arrlist = new ArrayList<>( Arrays.asList(arr));
    arrlist.remove(null);
    arrlist.add(x);
    arr = arrlist.toArray(arr);
    return arr;
}

public void print(TSP t){
    System.out.println("\nPath starting from City "+ t.path[0] +":");
    for(int i : t.path){
        System.out.print(i+" ---> ");
    }
    System.out.print(t.path[0]);

    System.out.println(" , Cost of travelling is " + t.cost);
}

}
```

OUTPUT-

```
Enter the number of vertices:
4
Enter the distance matrix:
City 1 to City 1 = 23
City 1 to City 2 = 11
City 1 to City 3 = 22
City 1 to City 4 = 1
City 2 to City 1 = 67
City 2 to City 2 = 44
City 2 to City 3 = 33
City 2 to City 4 = 99
City 3 to City 1 = 88
City 3 to City 2 = 17
City 3 to City 3 = 10
City 3 to City 4 = 9
City 4 to City 1 = 34
City 4 to City 2 = 54
City 4 to City 3 = 65
City 4 to City 4 = 21

Path starting from City 1:
1 ---> 4 ---> 3 ---> 2 ---> 1 , Cost of travelling is 150
Time required : 7300400 ns

Path starting from City 2:
2 ---> 3 ---> 4 ---> 1 ---> 2 , Cost of travelling is 99
Time required : 4086900 ns
```

Path starting from City 3:

3 ---> 4 ---> 1 ---> 2 ---> 3 , Cost of travelling is 121

Time required : 665800 ns

Path starting from City 4:

4 ---> 1 ---> 2 ---> 3 ---> 4 , Cost of travelling is 166

Time required : 170900 ns

No. of vertices	Path	Cost	Time (in ns)
4	1 ---> 4 ---> 3 ---> 2 ---> 1 2 ---> 3 ---> 4 ---> 1 ---> 2 3 ---> 4 ---> 1 ---> 2 ---> 3 4 ---> 1 ---> 2 ---> 3 ---> 4	150 99 121 166	7300400 4086900 665800 170900
6	1 ---> 3 ---> 5 ---> 4 ---> 6 ---> 2 ---> 1 2 ---> 4 ---> 1 ---> 3 ---> 5 ---> 6 ---> 2 3 ---> 1 ---> 4 ---> 5 ---> 6 ---> 2 ---> 3 4 ---> 1 ---> 3 ---> 5 ---> 6 ---> 2 ---> 4 5 ---> 1 ---> 3 ---> 4 ---> 6 ---> 2 ---> 5 6 ---> 4 ---> 1 ---> 3 ---> 5 ---> 2 ---> 6	300 313 416 357 296 307	6823600 3437100 1167000 310000 180000 182000
8	1 ---> 2 ---> 3 ---> 7 ---> 8 ---> 6 ---> 4 ---> 5 ---> 1 2 ---> 1 ---> 3 ---> 7 ---> 8 ---> 6 ---> 4 ---> 5 ---> 2 3 ---> 1 ---> 2 ---> 7 ---> 8 ---> 6 ---> 4 ---> 5 ---> 3 4 ---> 7 ---> 1 ---> 2 ---> 3 ---> 8 ---> 6 ---> 5 ---> 4 5 ---> 7 ---> 1 ---> 2 ---> 3 ---> 8 ---> 6 ---> 4 ---> 5 6 ---> 1 ---> 2 ---> 3 ---> 7 ---> 8 ---> 5 ---> 4 ---> 6 7 ---> 1 ---> 2 ---> 3 ---> 8 ---> 6 ---> 4 ---> 5 ---> 7 8 ---> 7 ---> 1 ---> 2 ---> 3 ---> 5 ---> 6 ---> 4 ---> 8	163 162 168 152 133 218 158 183	123990 431000 693300 519600 500300 542900 686100 544800

Analysis:

The Greedy Algorithm is based on building a tour by selecting and using the shortest possible edges. The initial step is to make a list of all edges in order of their increasing length. Once the list is completed, the shortest available edges are repeatedly chosen and included in the tour. While selecting the edges, it is necessary to make sure that the next edge intended to be used will neither close a smaller cycle, excluding some vertices, nor connect to a vertex which already has a degree 2.