

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**CERTIFICATE**

Certified that the Mini Project entitled “**The Travelling Salesman Problem**” is carried out by **Nikhil Dwivedi (1NH17EC729), Rithwik Shome (1NH17EC738) and Sahib Arora (1NH17EC742)** bonafide students of **New Horizon College of Engineering**, **Bengaluru** in partial fulfilment for the award of **Bachelor of Engineering** in **Electronics & Communication** of the **Visvesvaraya Technological University** during the year **2019-20**.

It is certified that all corrections/suggestions indicated for internal assessment has been incorporated in the report deposited to the department library. The Mini Project report has been approved as it satisfies the academic requirements in respect of the Mini Project work prescribed for the said degree.

Signature of HOD Signature of Internal Guide

Dr Sanjeev Sharma Dr Sanjeev Sharma

Professor and HOD Professor & HOD

Dept. of ECE Dept. of ECE

NHCE, Bangalore NHCE, Bangalore

EXTERNAL EXAMINER

Name of Examiner Signature with Date

1.

2.

**ACKNOWLEDGEMENT**

We would like to thank the management, Dr Mohan Maghnani, Chairman of New Horizon Educational Institutes for providing us the necessary infrastructure and creating a good environment.

We would like to express our sincere gratitude to our principal, Dr Manjunatha for giving us this opportunity to undertake this project.

We also wish to express our gratitude to Mr Sanjeev Kumar, HOD of Electronics & Communication Engineering Department, for his kind hearted support.

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our guide Mr Sanjeev Sharma and project coordinator Mr Naveen for their valuable guidance, encouragement and help throughout this project. The useful suggestion for this Mini Project and cooperative behaviour are sincerely acknowledged.

We also wish to express our indebtedness to our parents as well as our family members whose blessings and support always helped us to face the challenges ahead. At the end we would like to express our sincere thanks to all our friends who helped us directly or indirectly during this project work.

Rigorous hard work has been put in this project to ensure that it proves to be the best. We hope that this project will prove to be a breeding ground for the next generation of students and will guide them in every possible way.

|  |  |  |
| --- | --- | --- |
| **TABLE OF CONTENTS** | | |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| 1 | **Introduction** | 4 |
| 2 | **Analysis and Design** | 5 |
| 2.1 | Objectives of the Project | 6 |
| 2.2 | Requirement Specification | 6 |
| 3 | **Design** | 7 |
| 3.1  3.2  3.3 | Algorithm  Design of Algorithm  Nature of Algorithm | 8  8  10 |
| 3.4 | Flow Chat (Block Diagram) | 11 |
| 3.5 | Data Structures | 12 |
| 4  4.1  4.2 | **Implementation**  Logic  Code | 13  13  14 |
| 5  5.1  5.2 | **Sample Output**  Console Output  File Output | 20  20  21 |
| 6 | **Conclusion** | 28 |

**CHAPTER 1**

**INTRODUCTION**

The travelling salesman problem is a classic mathematical problem. It is used for optimization in the field of computer science and operational research. The problem statement is: given a list of cities and the distance between every pair, find the shortest path that visit each and every city and return to the origin. The problem is best represented by using graphs, where every node is city and the edge between them is the distance between the cities.

This problem has a wide range of application even in its purest form. The oblivious application of this problem is in planning and logistics. By modifying the problem, other problems from different field starts to appear, from DNA sequencing in biology to tracking astronomical object in astronomy. One example of this problem is figuring out the least amount of movement a drill has to do in order to make required number of holes in any surface. Here the hole required to be made is the nodes and the movement of the drill from one hole to the next are the edges between the nodes.

In the field of computational complexity, the most basic algorithm to solve the problem falls into the category of NP- complete problems. As the number of nodes(cities) increases, the number of possible paths increases superpolynomially. Therefore, the time required to find the solution increases beyond any practical time much faster. This one of the most aggressively studies optimization problem and there are few exact and heuristic algorithm that solve the problem better than the most basic problem. But nothing has reduced the time below superpolynomially.

The problem work by assuming that a Hamiltonian cycle exist, which is that a possible path that starts from a node and end there by covering each and every node in the process exist. The cyclic path is called Hamiltonian cycle

**CHAPTER 2**

**ANALYSIS AND DESIGN**

**2.1 OBJECTIVES OF THE PROJECT**

The objective of the project is to demonstrate an algorithm to solve the traveling salesman problem. The console application focuses on one of the purest applications of the problem and solves it using a working algorithm.

In the console application, the program will prompt the user to enter the number of cities, their names and the distances between each and every city. Then the program will run the algorithm written in C programming language, and print the shorted path in terms of cities and the minimum distance required to travel to cover the path. The program also stores the input and output in a file for later reference.

**2.2 REQUIREMENT SPECIFICATION**

**2.2.1. Hardware Requirement**

* **Processor: X86 Compatible processor with 1.7 GHz Clock speed or above**
* **RAM: 512 MB or more**
* **Hard disk: 500 GB or more**

**2.2.2. Software Requirements**

* **Dev C/C++**
* **Windows XP or above Operating System**
* **Data structures: Graphs**

**CHAPTER 3**

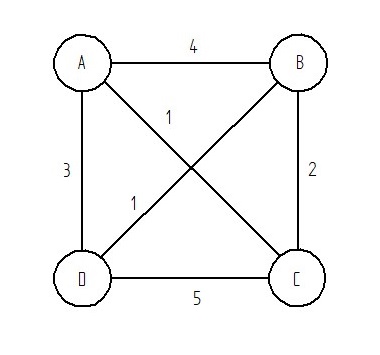
**DESIGN**

**3.1. Algorithm**

1. Enter the number of cities, places, etc.
2. Prompt the user to enter the distance between each and every city.
3. Prompt the user to enter the staring city.
4. Repeat step 2, until all cities are done.
5. Print the cost matrix, which is a graphical representation of all cities with their distances in a matrix.
6. Find the path with the lowest cost or distance, this is an indirect way of calculating the (n-1)! path.
7. Go to the starting city and print it, then mark that city as done.
8. Find the minimum valued element in the row corresponding to the city number.
9. Print the corresponding city name of that element and mark that city as done.
10. Go to the next city, after recording the distance travelled or the cost.
11. Go to step 7, until all the rows (cities) are marked as done.
12. After printing the path, print the distance travelled or the total cost of the given path.

**3.2. Design of Algorithm**

Consider the following graph and its matrix representation.



**0 4 1 3**

**4 0 2 1**

**1 2 0 5**

**3 1 5 0**

The above graph has the following path:

1. A > B > C > D > A
2. A > B > D > C > A
3. A > C > B > D > A
4. A > C > D > B > A
5. A > D > B > C > A
6. A > D > C > B > A

Now there is a tree like structure in the path, which means that when the program is in node A, its looks for the nearest node and keeps on doing that until it reaches the starting node (A).

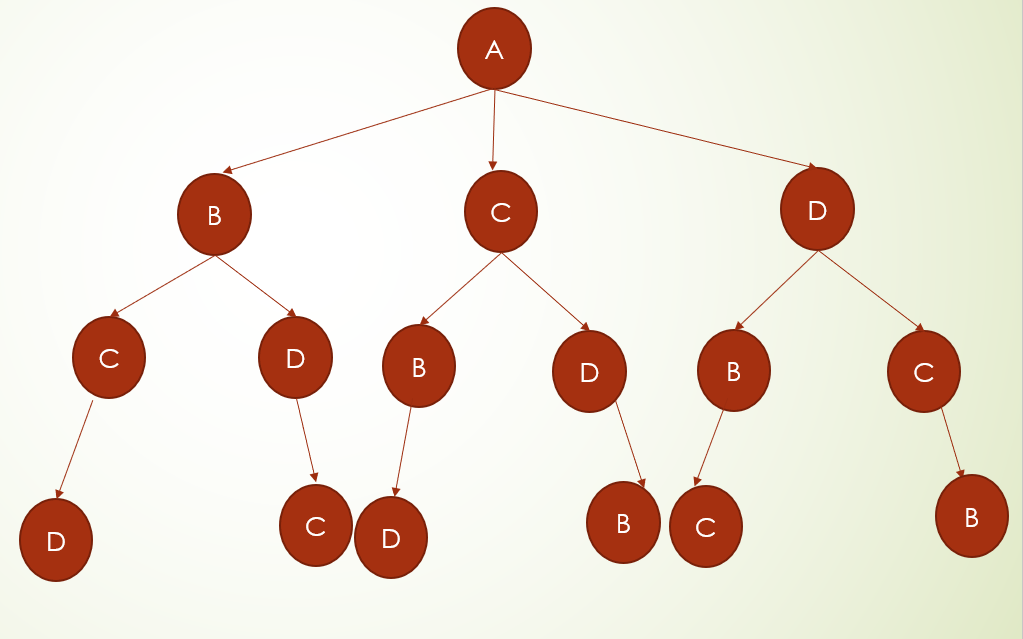
So, the algorithm calculates the distances from node A to B, C or D, which is represented by the weight of the edges. It them moves to the least weighted node and repeats the process until it reaches the starting node.

In the above example the solution is:

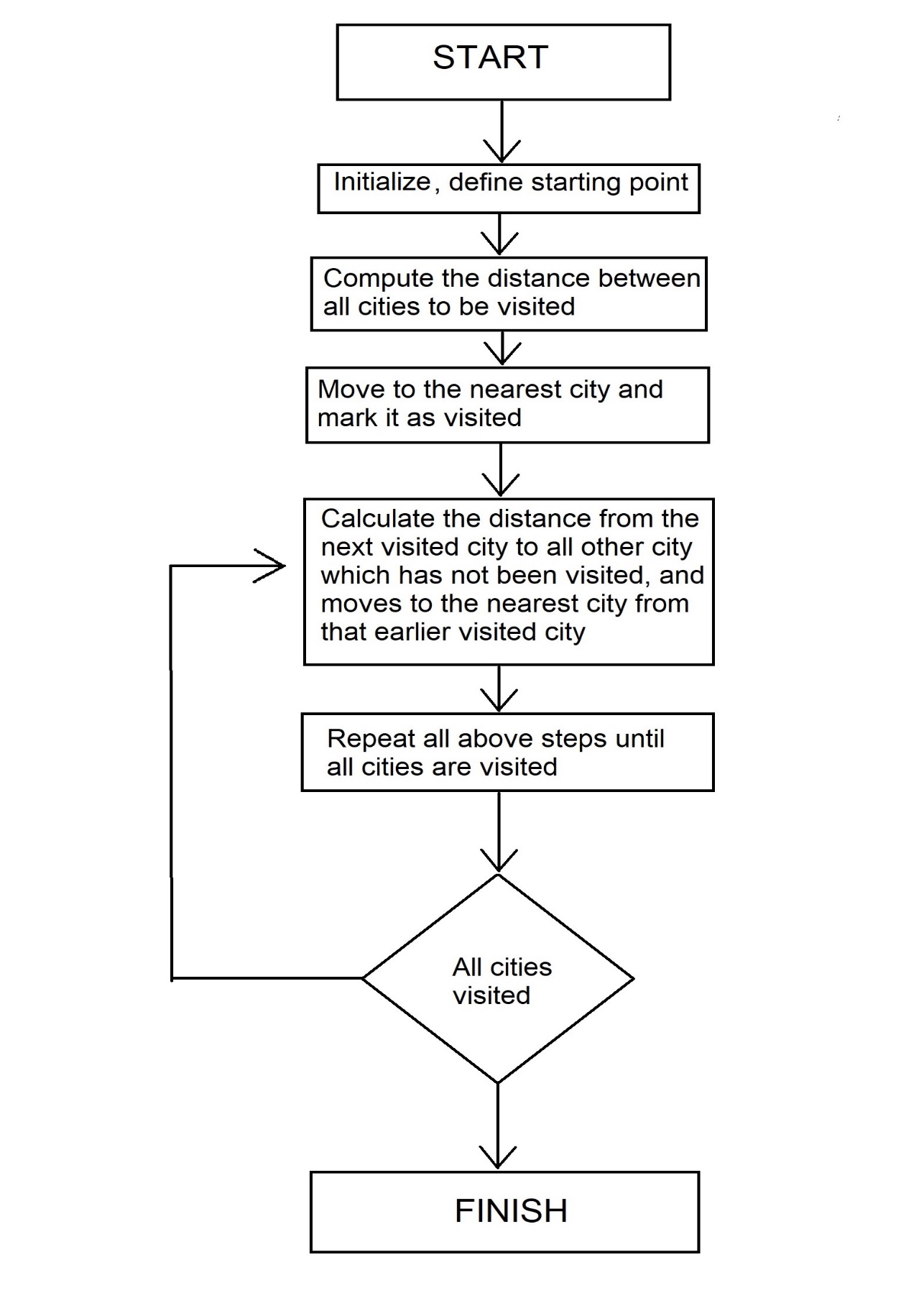
A > C > B > D > A

When the algorithm is in node A, it sees node C as the least weighted edge and moves there. Then it goes to the next least weighted node but does not consider the edge between the node C and node A. Then it goes to node B and does not consider the edges between the node B and node A, C, and B. Then it goes to D and it sees that all the nodes are visited so it returns back to node A (starting node). Thus, completing the cycle and solving the travelling salesman’s problem.

**3.2. Design Nature**



**3.2. Flow Chart (Block Diagram)**



**3.3. Function Flow Block Diagram**

main()

input()

calc(int city)

least\_city(int i)

storage()

**3.4. Data Structures**

The list of data structures used in the program are:

1. city\_names[50][50]: This 2D array is used to store the name of all the cities in string (character array).
2. arr[10][10]: This 2D array is used to store the array representation of the graph, which is in turn used to represent the travelling salesman problem, where the cities are nodes and the edges are the distances between the nodes.
3. done[10]: This 1D array is used to mark the city that have been visited and won’t have to visited again.
4. result[10]: This 1D array is used to store the path city in order.

**CHAPTER 4**

**IMPLEMENTATION**

**4.1. Logic**

The calc(int city) function is used to parse through the nodes(cities) and store the output in the result() array. It also directly prints the visited node on the console screen. The function calls itself until the staring city is visited and its breaks after storing and printing the staring node.

**C code:**

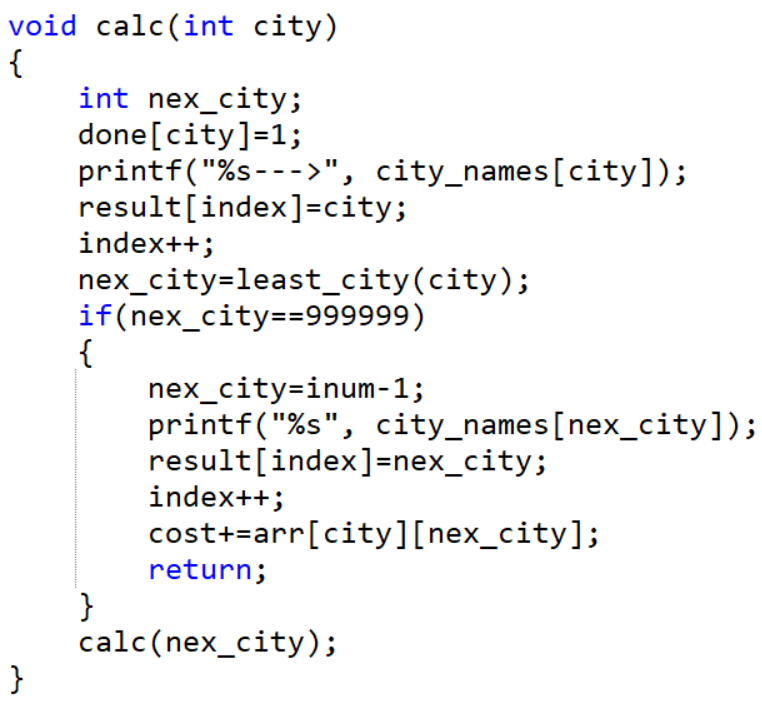


Figure 4.1.1

The least\_city(int i) function is used to determine which node have the least edge value from the current node. In order words, which next what is nearest city from the current city. For example, in a setup of 4 cities A, B, C and D and if we are in city B, then the function will calculate the least distance required to travel to either A, C and D.

**C code:**

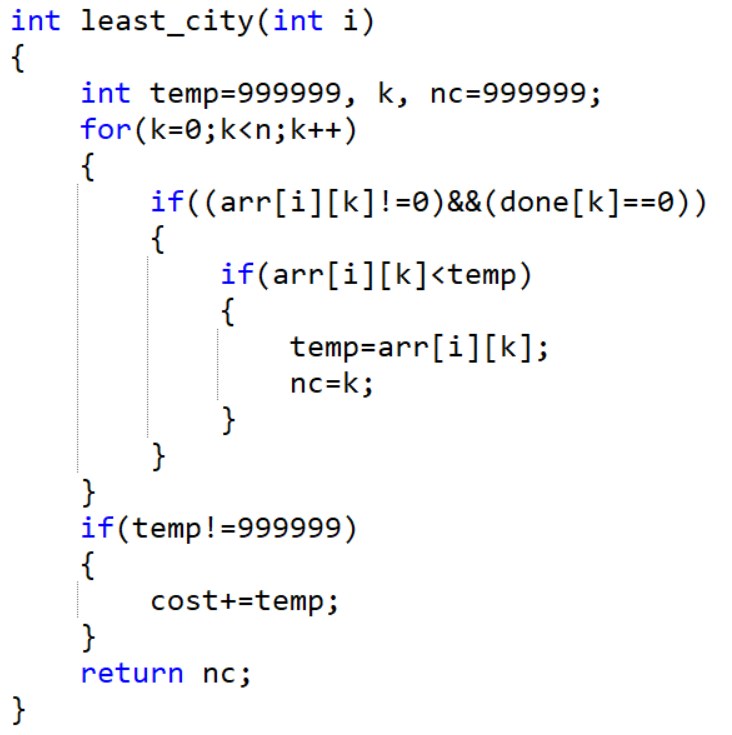
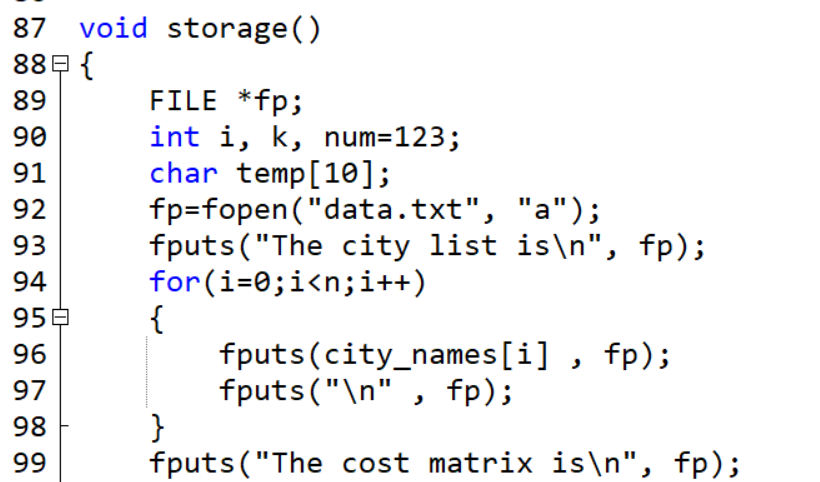
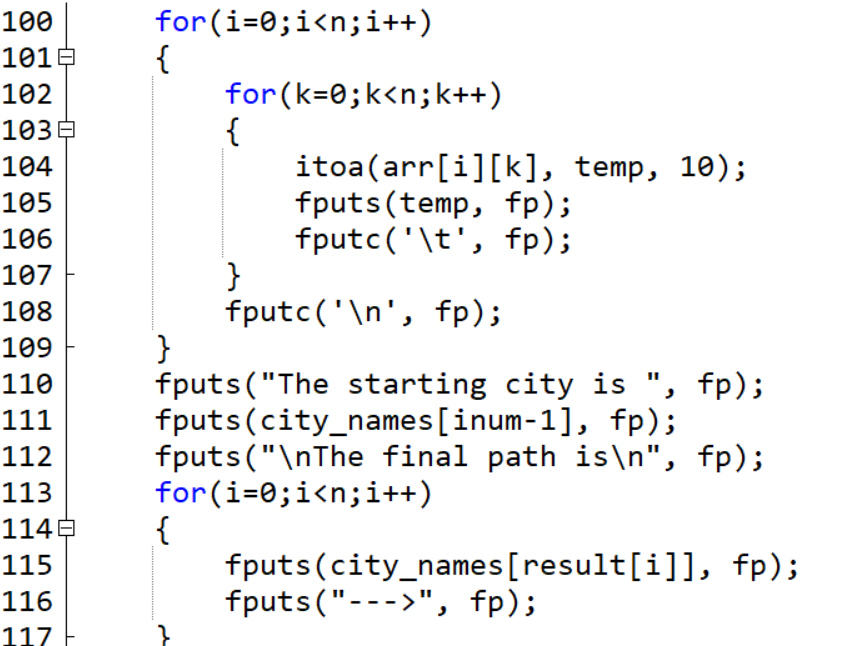
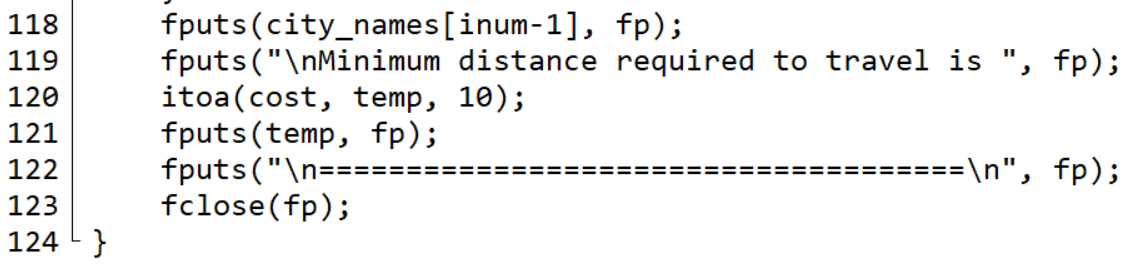


Figure 4.1.2

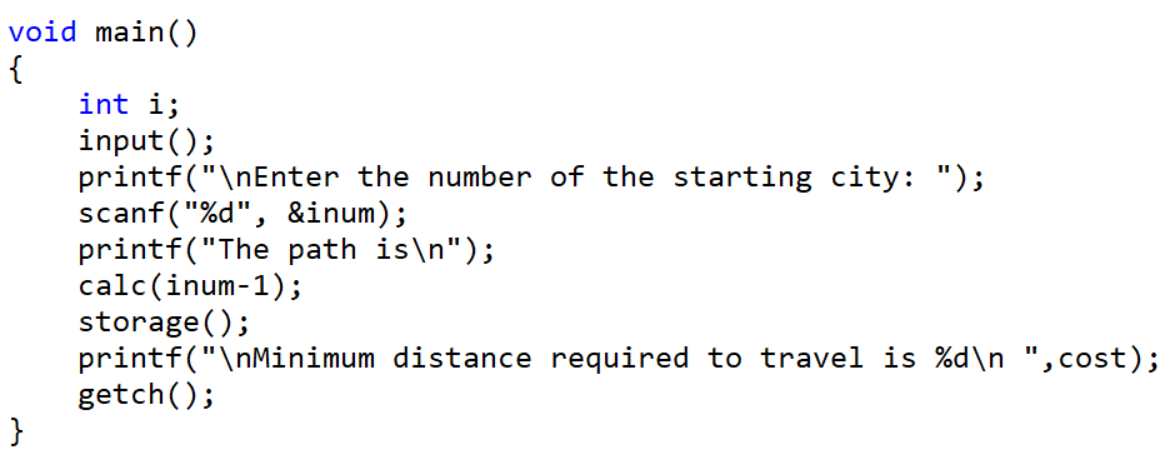
The storage function is used to store the input and output of the console in a file.



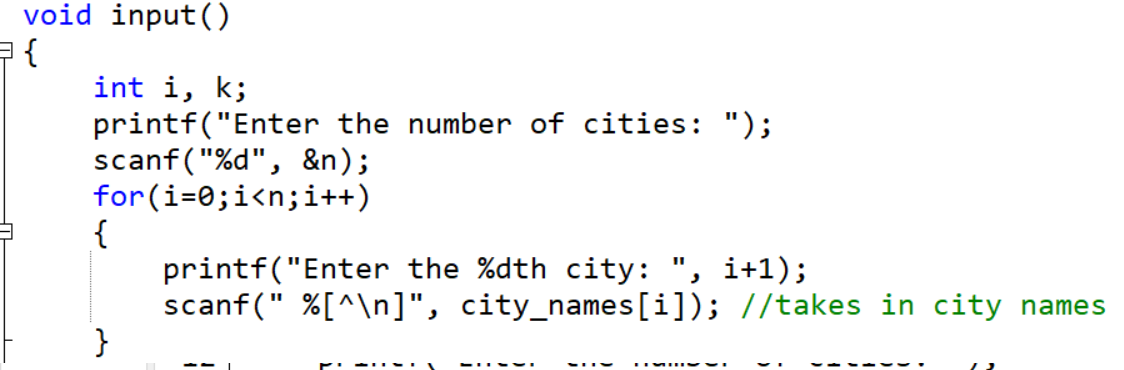


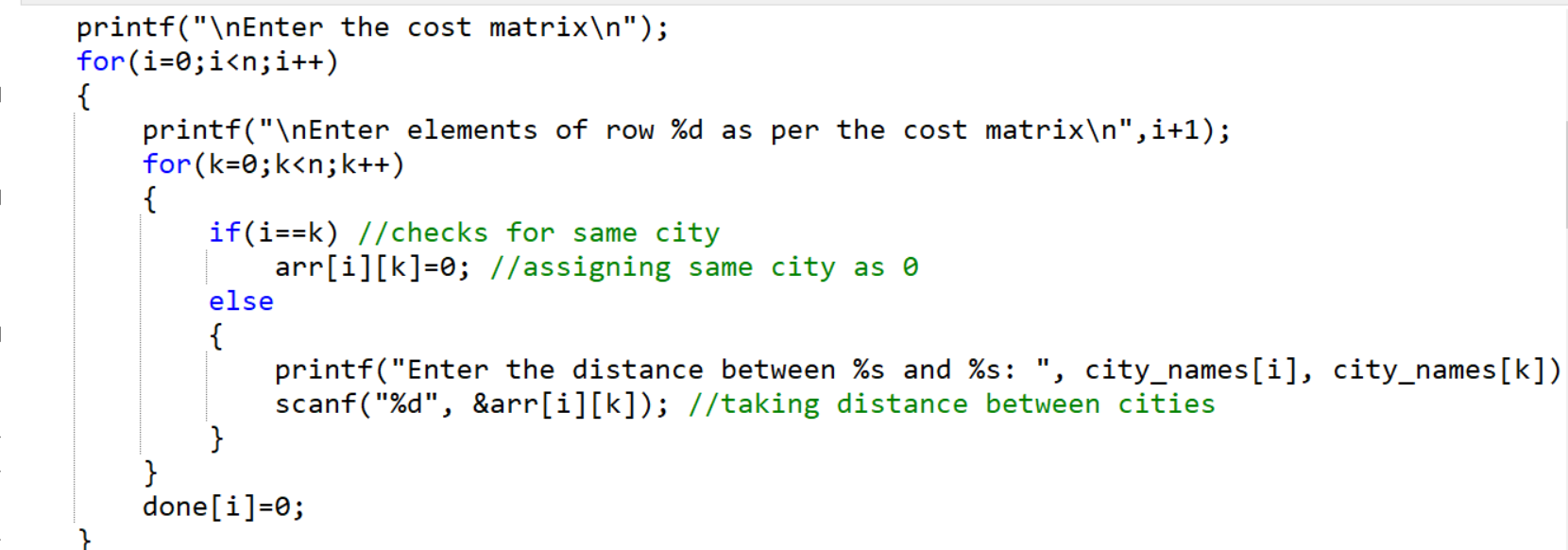


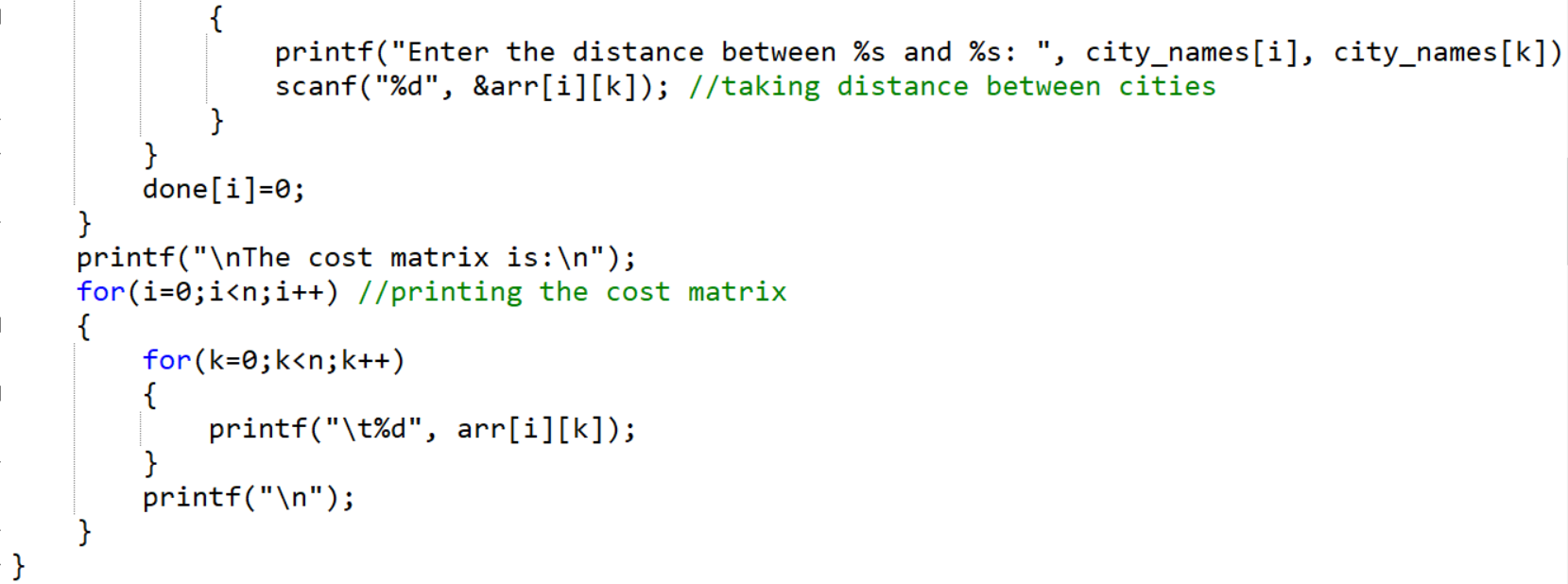
The main function is used to call the other function in the required order.



The input function take input from the user and stores it in the appropriate data structures.







**CHAPTER 5**

**SAMPLE OUTPUT**

**5.1. Console Output**

The output of the sample code is:

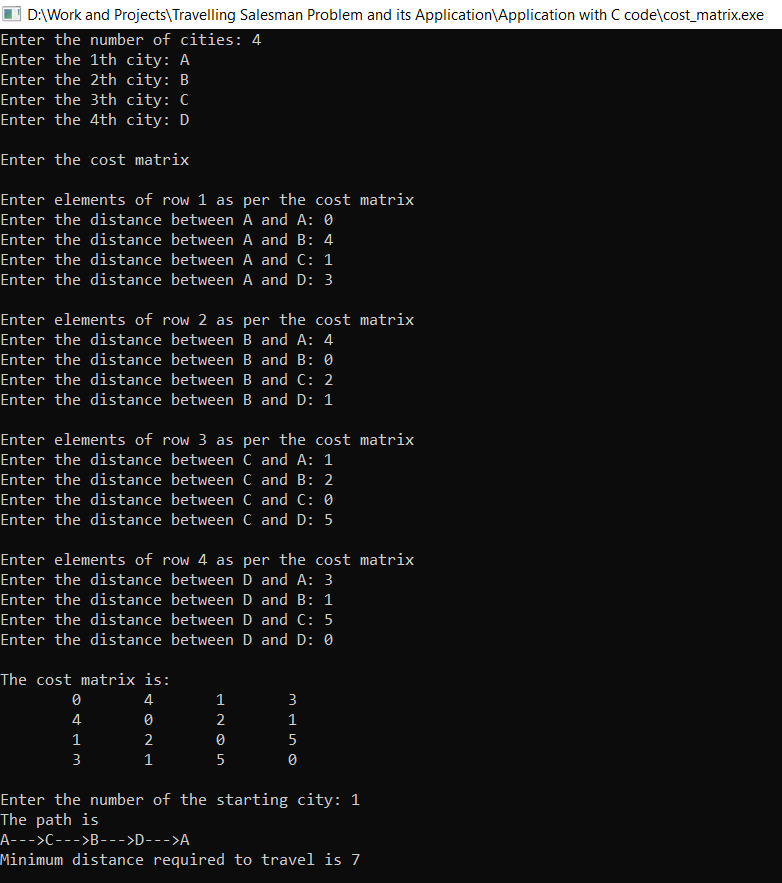


Figure 5.1.1

**5.2. File Output**

The output stored in the file:

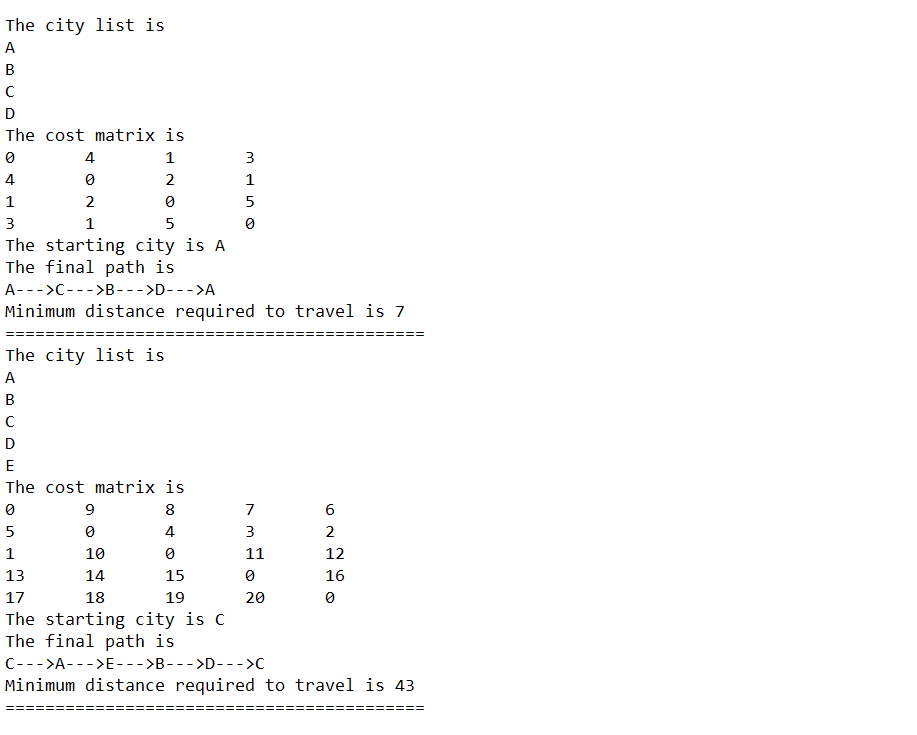
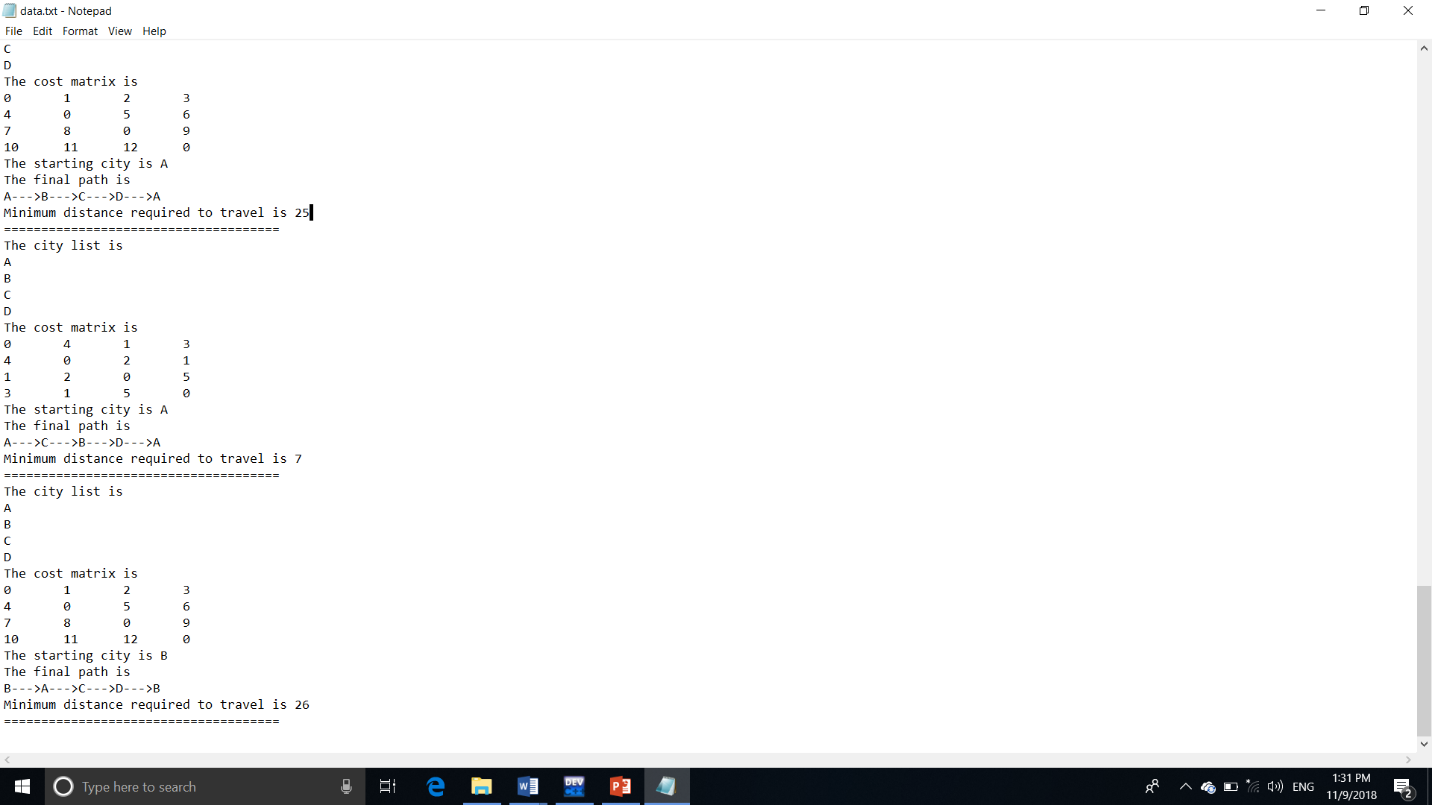


Figure 5.1.2



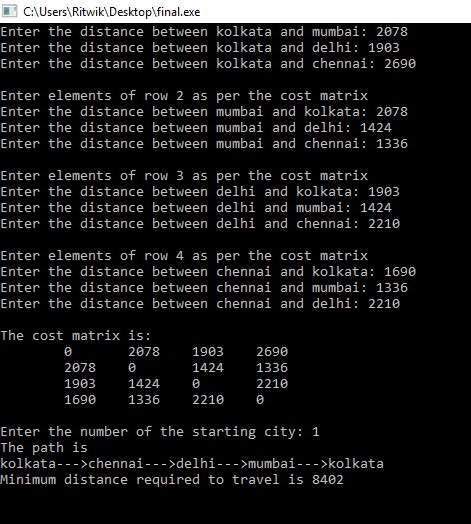
**5.3. Practical Example**

In the following example we have taken a real life example between the cities Kolkata, Mumbai, Delhi and Chennai whose distances from one another are given as

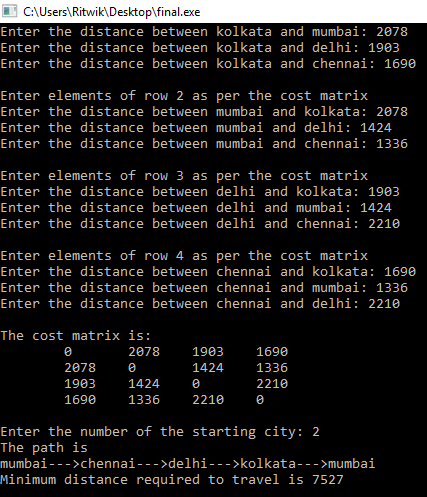
|  |  |  |
| --- | --- | --- |
| **City 1** | **City 2** | **Distance (kms)** |
| Kolkata | Mumbai | 2078 |
| Kolkata | Delhi | 1903 |
| Kolkata | Chennai | 2690 |
| Mumbai | Kolkata | 2078 |
| Mumbai | Delhi | 1424 |
| Mumbai | Chennai | 1336 |
| Delhi | Kolkata | 1903 |
| Delhi | Mumbai | 1424 |
| Delhi | Chennai | 2210 |
| Chennai | Kolkata | 1690 |
| Chennai | Mumbai | 1336 |
| Chennai | Delhi | 2210 |

Three cases have been taken here with the salesman starting from each of the three cities

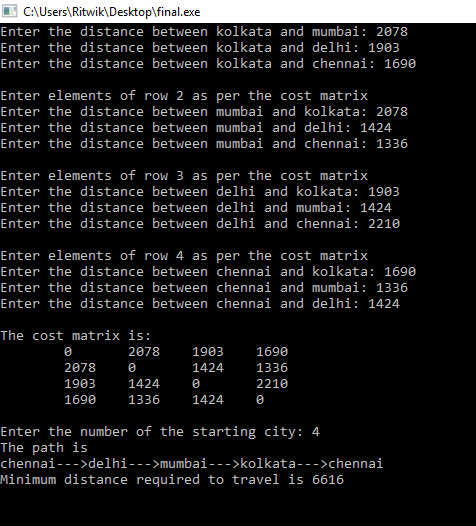
1. *Salesman starting from Kolkata*



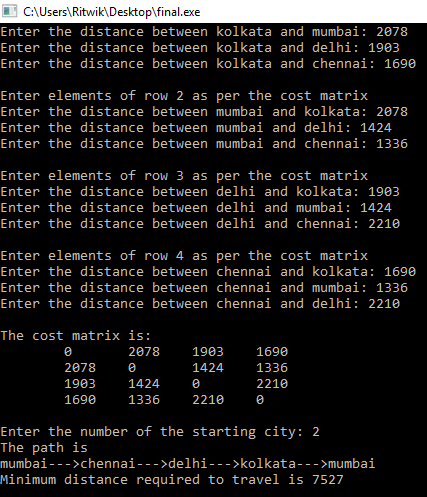
1. *Salesman starting from Mumbai*



1. *Salesman starting from Chennai*



1. *Salesman starting from Mumbai*



**CHAPTER 6**

**CONCLUSION**

While doing this project, we learnt a lot about the development of algorithm and then converting the algorithm into C code. We realized the importance of being about to think logically and analytically to solve a problem rather than knowing a lot of programming language. This project is a great example of how an algorithm can scale into something unpractical if the number of input increase slightly. This project was a window into the field of computational complexity.

This project increased our problem-solving skill and made us a better engineer. We leant about reusability and how to make my code as much as reusable as possible. The project displays the TSP in a very user-friendly way and thus makes it approachable for normal users. The project achieves this by taking one of the, perhaps the purest application of TSP and solving it.