

# Practical 9 Implementing Traveling Salesman Problem

GAHAN M. SARAIYA, 18MCEC10

[18mcec10@nirmauni.ac.in](mailto:18mcec10@nirmauni.ac.in)

## I. INTRODUCTION

Aim of this practical is to Implement **Traveling Salesman Problem (TSP)** using Branch and Bound method.

**Traveling Salesman Problem** is defined as “Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?” It is an NP-hard problem.

- Branch and bound (**BB** or **B&B**) is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization.

## II. IMPLEMENTATION

- Model Traveling Salesman Problem as an undirected weighted graph.
- consider cities as graph's vertices
- consider paths as graph's edges
- consider path's distance as edge's length (weight of edge)
- TSP is minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once.

### I. Main Program - *traveling\_salesman\_problem.c*

---

```
1 //  
2 // -----  
3 // Author: Gahan Saraiya  
4 // GiT: http://github.com/gahan9/  
5 // StackOverflow: https://stackoverflow.com/users/story/7664524  
6 // Website: http://gahan9.github.io/  
7 // -----  
8 // Implementing Traveling Salesman Problem using Branch and Bound  
9  
10 #include <stdio.h>  
11 #include <stdlib.h>  
12 #include <math.h>
```

```
12 #include "../utils/constant.h"
13 #include "../utils/utility.h"
14
15 #define inf 2147483647
16
17 int cost = 0;
18
19 //int matrix[100][100], visited[100], number_of_cities, cost = 0;
20 int display_cost_matrix(int **array, int no_of_elements) {
21     // display given array of given size(no. of elements require because sizeof()
22     // returns max bound value)
23     write_log(": ");
24     for (int i = 0; i < no_of_elements; i++) {
25         printf("City-%d >>: ", i);
26         for (int j = 0; j < no_of_elements; j++) {
27             printf("%d ", array[i][j]);
28         }
29         printf("\n");
30     }
31     return 0;
32 }
33
34 int get_nearest_city(int c, int **cost_matrix, int *visited_status, int
35     number_of_cities) {
36     int nearest_neighbour_city = inf;
37     int min = inf, temp;
38     for (int i = 0; i < number_of_cities; i++) {
39         if ((cost_matrix[c][i] != 0) && (visited_status[i] == 0)) {
40             if (cost_matrix[c][i] < min) {
41                 min = cost_matrix[i][0] + cost_matrix[c][i];
42             }
43             temp = cost_matrix[c][i];
44             nearest_neighbour_city = i;
45         }
46     }
47     if (min != inf)
48         cost += temp;
49     return nearest_neighbour_city;
50 }
51
52 int min_cost_calc(int city, int **cities, int *visited_cities, int
53     number_of_cities) {
54     int i, nearest_neighbour_city;
55     visited_cities[city] = 1;
56     printf("%d ==> ", city + 1);
57     nearest_neighbour_city = get_nearest_city(city, cities, visited_cities,
58         number_of_cities);
```

```
56     if (nearest_neighbour_city == inf) {
57         nearest_neighbour_city = 0;
58         printf("%d", nearest_neighbour_city + 1);
59         cost += cities[city][nearest_neighbour_city];
60         return 1;
61     }
62     return min_cost_calc(nearest_neighbour_city, cities, visited_cities,
63         ↪ number_of_cities);
64 }
65
66 int main() {
67     int number_of_cities;
68     printf("\nEnter Number of Cities: \n");
69     scanf("%d", &number_of_cities);
70     int **cities = malloc(number_of_cities * sizeof(int *));
71     int *visited_cities = malloc(number_of_cities * sizeof(int *));
72     for (int i = 0; i < number_of_cities; i++) {
73         cities[i] = malloc(number_of_cities * sizeof(int));
74     }
75
76     printf("\nEnter Cost Matrix for travelling through %d cities: \n",
77         ↪ number_of_cities);
78     for (int i = 0; i < number_of_cities; i++) {
79         // printf("\n Enter cost from city 1# : %d\n", i + 1);
80         for (int j = 0; j < number_of_cities; j++)
81             if (i == j) {
82                 cities[i][j] = inf;
83                 cities[i][j] = 0;
84             } else {
85                 cities[i][j] = 1 + rand() % 9;
86             }
87         // scanf("%d", &cities[i][j]);
88         visited_cities[i] = 0;
89     }
90
91     printf("\nThe Cost Matrix is:\n");
92     display_cost_matrix(cities, number_of_cities);
93
94     printf("\nThe Path is:\n");
95     min_cost_calc(0, cities, visited_cities, number_of_cities);
96     printf("\nMinimum Cost of tour: -> %d", cost);
97     return 1;
98 }
```

---

## I.1 Output

```
1 Enter Number of Cities: 6
2
3 Enter Cost Matrix for travelling through 6 cities:
4
5 The Cost Matrix is:
6 City-0 >>: 0 2 8 15 1 10
7 City-1 >>: 5 0 19 19 3 5
8 City-2 >>: 6 6 0 2 8 2
9 City-3 >>: 12 16 3 0 8 17
10 City-4 >>: 12 5 3 14 0 13
11 City-5 >>: 3 2 17 19 16 0
12
13 The Path is:
14 1 ==> 6 ==> 5 ==> 4 ==> 3 ==> 2 ==> 1
15 Minimum Cost of tour: -> 54
```

## I.2 Output

```
1 Enter Number of Cities:
2
3 Enter Cost Matrix for travelling through 7 cities:
4
5 The Cost Matrix is:
6 City-0 >>: 0 6 9 8 5 9 2
7 City-1 >>: 4 0 1 8 3 9 3
8 City-2 >>: 8 7 0 8 6 8 9
9 City-3 >>: 4 1 1 0 7 6 1
10 City-4 >>: 5 8 7 6 0 9 6
11 City-5 >>: 3 1 3 1 7 0 5
12 City-6 >>: 9 2 8 4 3 7 0
13
14 The Path is:
15 1 ==> 7 ==> 6 ==> 5 ==> 4 ==> 3 ==> 2 ==> 1
16 Minimum Cost of tour: -> 34
```