Practical 8
Implementing Traveling Salesman Problem

GAHAN M. SARAIYA, 18MCEC10

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  // Author: Gahan Saraiya
3  // GiT: http://github.com/gahan9/
4  // StackOverflow: https://stackoverflow.com/users/story/7664524
5  // Website: http://gahan9.github.io/
6  //
    ↪  ----------------------------------------------------------------------------------------
7  // Implementing Traveling Salesman Problem using Branch and Bound
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #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()
       ↪ returns max bound value)
22     write_log(": ");
23     for (int i = 0; i < no_of_elements; i++) {
24         printf("City-%d >>: ", i);
25         for (int j = 0; j < no_of_elements; j++) {
26             printf("%d ", array[i][j]);
27         }
28         printf("\n");
29     }
30     return 0;
31 }
32
33 int get_nearest_city(int c, int **cost_matrix, int *visited_status, int
   ↪ number_of_cities) {
34     int nearest_neighbour_city = inf;
35     int min = inf, temp;
36     for (int i = 0; i < number_of_cities; i++) {
37         if ((cost_matrix[c][i] != 0) && (visited_status[i] == 0)) {
38             if (cost_matrix[c][i] < min) {
39                 min = cost_matrix[i][0] + cost_matrix[c][i];
40             }
41             temp = cost_matrix[c][i];
42             nearest_neighbour_city = i;
43         }
44     }
45     if (min != inf)
46         cost += temp;
47     return nearest_neighbour_city;
48 }
49
50 int min_cost_calc(int city, int **cities, int *visited_cities, int
   ↪ number_of_cities) {
51     int i, nearest_neighbour_city;
52     visited_cities[city] = 1;
53     printf("%d ===> ", city + 1);
54     nearest_neighbour_city = get_nearest_city(city, cities, visited_cities,
       ↪ number_of_cities);
55
```

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

I.1 Output

```
Enter Number of Cities: 6

Enter Cost Matrix for travelling through 6 cities:

The Cost Matrix is:
City-0 >>: 0 2 8 15 1 10
City-1 >>: 5 0 19 19 3 5
City-2 >>: 6 6 0 2 8 2
City-3 >>: 12 16 3 0 8 17
City-4 >>: 12 5 3 14 0 13
City-5 >>: 3 2 17 19 16 0

The Path is:
1 ===> 6 ===> 5 ===> 4 ===> 3 ===> 2 ===> 1
Minimum Cost of tour: -> 54
```

I.2 Output

```
Enter Number of Cities:

Enter Cost Matrix for travelling through 7 cities:

The Cost Matrix is:
City-0 >>: 0 6 9 8 5 9 2
City-1 >>: 4 0 1 8 3 9 3
City-2 >>: 8 7 0 8 6 8 9
City-3 >>: 4 1 1 0 7 6 1
City-4 >>: 5 8 7 6 0 9 6
City-5 >>: 3 1 3 1 7 0 5
City-6 >>: 9 2 8 4 3 7 0

The Path is:
1 ===> 7 ===> 6 ===> 5 ===> 4 ===> 3 ===> 2 ===> 1
Minimum Cost of tour: -> 34
```