



**Experiment No. : 6**

**Title: Implement Travelling Salesman Problem using  
Dynamic approach**



**Batch: A1      Roll No.: 16010422013**  
**Experiment No.: 6**

**Aim:** To Implement Travelling Salesman Problem for minimum 6 vertices using Dynamic approach and analyse its time Complexity.

### Algorithm of Travelling Salesman Problem:

#### Algorithm: Traveling-Salesman-Problem

```

C ({1}, 1) = 0
for s = 2 to n do
    for all subsets S ∈ {1, 2, 3, ... , n} of size s and containing 1
        C (S, 1) = ∞
    for all j ∈ S and j ≠ 1
        C (S, j) = min {C (S - {j}, i) + d(i, j) for i ∈ S and i ≠ j}
Return minj C ({1, 2, 3, ..., n}, j) + d(j, i)

```

#### Algorithm 1: Dynamic Approach for TSP

**Data:** *s*: starting point; *N*: a subset of input cities; *dist()*: distance among the cities

**Result:** *Cost*: TSP result

*Visited*[*N*] = 0;

*Cost* = 0;

**Procedure TSP(*N*, *s*)**

```

    Visited[s] = 1;
    if |N| = 2 and k ≠ s then
        Cost(N, k) = dist(s, k);
        Return Cost;
    else
        for j ∈ N do
            for i ∈ N and visited[i] = 0 do
                if j ≠ i and j ≠ s then
                    Cost(N, j) = min ( TSP(N - {i}, j) + dist(j, i))
                    Visited[j] = 1;
                end
            end
        end
    end
    Return Cost;
end

```

### Explanation and Working of Travelling Salesman Problem:

#### Problem Statement

A traveller needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city?

### Solution

Travelling salesman problem is the most notorious computational problem. We can use brute-force approach to evaluate every possible tour and select the best one. For  $n$  number of vertices in a graph, there are  $(n - 1)!$  number of possibilities.

Instead of brute-force using dynamic programming approach, the solution can be obtained in lesser time, though there is no polynomial time algorithm.

Let us consider a graph  $G = (V, E)$ , where  $V$  is a set of cities and  $E$  is a set of weighted edges. An edge  $e(u, v)$  represents that vertices  $u$  and  $v$  are connected. Distance between vertex  $u$  and  $v$  is  $d(u, v)$ , which should be non-negative.

Suppose we have started at city 1 and after visiting some cities now we are in city  $j$ . Hence, this is a partial tour. We certainly need to know  $j$ , since this will determine which cities are most convenient to visit next. We also need to know all the cities visited so far, so that we don't repeat any of them. Hence, this is an appropriate sub-problem.

For a subset of cities  $S \in \{1, 2, 3, \dots, n\}$  that includes 1, and  $j \in S$ , let  $C(S, j)$  be the length of the shortest path visiting each node in  $S$  exactly once, starting at 1 and ending at  $j$ .

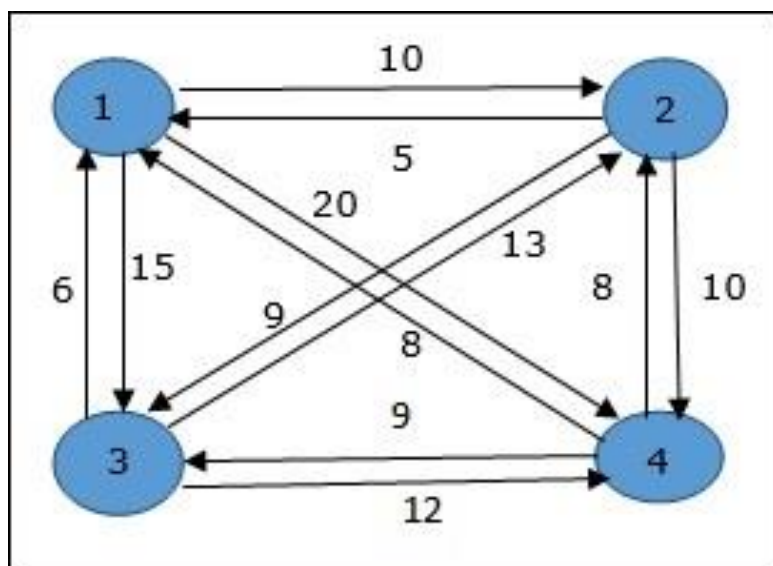
When  $|S| > 1$ , we define  $C(S, 1) = \infty$  since the path cannot start and end at 1.

Now, let express  $C(S, j)$  in terms of smaller sub-problems. We need to start at 1 and end at  $j$ . We should select the next city in such a way that

$$C(S, j) = \min_{i \in S, i \neq j} \{C(S - \{j\}, i) + d(i, j)\}$$

### Example

In the following example, we will illustrate the steps to solve the travelling salesman problem.



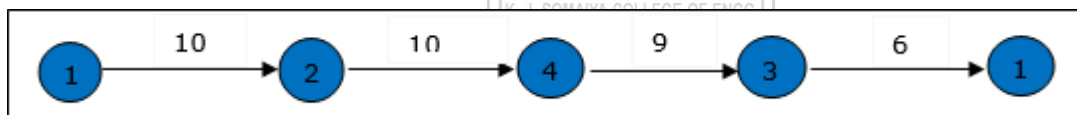
From the above graph, the following table is prepared.

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

The minimum cost path is 35.

Start from cost {1, {2, 3, 4}, 1}, we get the minimum value for  $d[1, 2]$ . When  $s = 3$ , select the path from 1 to 2 (cost is 10) then go backwards. When  $s = 2$ , we get the minimum value for  $d[4, 2]$ . Select the path from 2 to 4 (cost is 10) then go backwards.

When  $s = 1$ , we get the minimum value for  $d[4, 3]$ . Selecting path 4 to 3 (cost is 9), then we shall go to then go to  $s = \Phi$  step. We get the minimum value for  $d[3, 1]$  (cost is 6).



### Derivation of Travelling Salesman Problem:

#### Time complexity Analysis

The time complexity of the dynamic programming solution for the Traveling Salesman Problem (TSP) implemented here is  $O(n^2 \cdot 2^n)$ , where  $n$  is the number of vertices (cities). Here's the breakdown of the time complexity:

- The number of subproblems is  $(n \times 2^n)$ , where  $n$  represents the number of vertices and  $2^n$  represents the number of subsets of vertices (i.e., different combinations of visited cities).
- For each subproblem, we iterate through all the cities to find the optimal solution. This takes  $O(n)$  time.
- Hence, the overall time complexity is  $O(n^2 \cdot 2^n)$ .

This time complexity is exponential, which means the algorithm might not be efficient for large values of  $n$ . However, dynamic programming helps avoid redundant calculations and significantly improves the efficiency compared to a brute-force approach, which would have a time complexity of  $O(n!)$ .

**Program(s) of Travelling Salesman Problem:**

```

#include <stdio.h>
#include <limits.h>

#define V 4

int min(int a, int b) {
    return (a < b) ? a : b;
}

int tsp(int graph[V][V], int mask, int pos, int dp[V][1 << V]) {
    if (mask == (1 << V) - 1) {
        return graph[pos][0];
    }

    if (dp[pos][mask] != -1) {
        return dp[pos][mask];
    }

    int ans = INT_MAX;

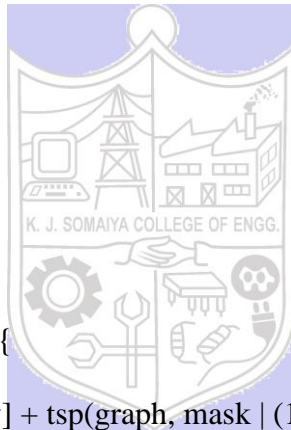
    for (int city = 0; city < V; city++) {
        if ((mask & (1 << city)) == 0) {
            int newAns = graph[pos][city] + tsp(graph, mask | (1 << city), city, dp);
            ans = min(ans, newAns);
        }
    }

    return dp[pos][mask] = ans;
}

int main() {
    int graph[V][V] = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    int dp[V][1 << V];
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < (1 << V); j++) {
            dp[i][j] = -1;
        }
    }

```



```

    }
}

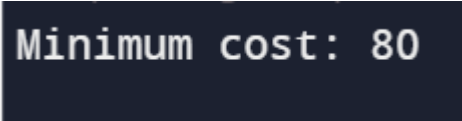
int mask = 1;
int pos = 0;

printf("Minimum cost: %d\n", tsp(graph, mask, pos, dp));

return 0;
}

```

### Output(o) of Travelling Salesman Problem:



Minimum cost: 80

**Post Lab Questions:-** Explain how Travelling Salesman Problem using greedy method is different from Dynamic Method in detail.

The Traveling Salesman Problem (TSP) is a classic problem in computer science and optimization. It involves finding the shortest possible route that visits each city exactly once and returns to the original city. Two common methods to solve the TSP are the greedy method and the dynamic programming method. Let's explore the differences between these two approaches:

#### 1. Greedy Method:

- The greedy method is based on making locally optimal choices at each step with the hope of finding a globally optimal solution.
- At each step, the greedy algorithm selects the nearest unvisited city from the current city.
- The algorithm continues this process until all cities are visited, and then it returns to the starting city.
- Greedy algorithms are relatively simple and easy to implement.
- However, the solution obtained using the greedy method is not guaranteed to be optimal. In many cases, the solution may be close to optimal, but there is no guarantee that it will always find the shortest possible route.

#### 2. Dynamic Programming Method:

- The dynamic programming method breaks down the problem into smaller subproblems and solves each subproblem only once, storing the solutions in a table (memoization).
- It systematically evaluates all possible solutions and selects the one with the minimum cost.
- The dynamic programming method guarantees finding the optimal solution because it explores all possible combinations of cities and selects the one with the minimum cost.
- This approach is more complex and requires more computational resources compared to the greedy method.
- However, dynamic programming ensures optimality, making it suitable for situations where the optimal solution is required.

In summary, the main differences between the greedy method and the dynamic programming method for solving the TSP are in their approach to finding solutions and the guarantees they

provide regarding optimality. Greedy algorithms offer simplicity and efficiency but may not always produce optimal solutions, while dynamic programming ensures optimality at the cost of increased computational complexity.

---

**Conclusion: (Based on the observations): Thus we have successfully implemented TSP using dynamic approach.**

---

---

**Outcome: Implement Greedy and Dynamic Programming algorithms**

---

**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Cormen ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.

