| Experiment No. 9 |
|---|
| Travelling Salesperson Problem using Dynamic Approach |
| Date of Performance: |
| Date of Submission: |

## Experiment No. 9

**Title:** Travelling Salesman Problem

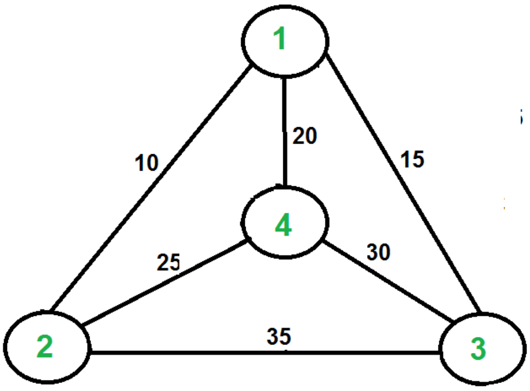
**Aim:** To study and implement Travelling Salesman Problem.

**Objective:** To introduce Dynamic Programming approach

**Theory:**

The **Traveling Salesman Problem (TSP)** is a classic optimization problem in which a salesperson needs to visit a set of cities exactly once and return to the starting city while minimizing the total distance traveled.

Given a set of cities and the distance between every pair of cities, find the **shortest possible route** that visits every city exactly once and returns to the starting point.



For example, consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80. The problem is a famous NP-hard problem. There is no polynomial-time know solution for this problem. The following are different solutions for the traveling salesman problem.

**Naive Solution:**

1) Consider city 1 as the starting and ending point.

2) Generate all (n-1)! Permutations of cities.

3) Calculate the cost of every permutation and keep track of the minimum cost permutation.

4) Return the permutation with minimum cost.

Time Complexity: ?(n!)

**Dynamic Programming:**

Let the given set of vertices be {1, 2, 3, 4,.n}. Let us consider 1 as starting and ending point of output. For every other vertex I (other than 1), we find the minimum cost path with 1 as the starting point, I as the ending point, and all vertices appearing exactly once. Let the cost of this path cost (i), and the cost of the corresponding Cycle would cost (i) + dist(i, 1) where dist(i, 1) is the distance from I to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far.

Now the question is how to get cost(i)? To calculate the cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems.

Let us define a term *C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i*. We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

```
If size of S is 2, then S must be {1, i},

 C(S, i) = dist(1, i)

Else if size of S is greater than 2.

 C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j !=
i and j != 1.
```

**Implemenation:**

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>



#define MAX_CITIES 10



int numCities;

int distance[MAX_CITIES][MAX_CITIES];
int memo[MAX_CITIES][1 << MAX_CITIES];

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

int tsp(int currentCity, int visited) {
    if (visited == (1 << numCities) - 1) // all cities visited
        return distance[currentCity][0];

    if (memo[currentCity][visited] != -1)
        return memo[currentCity][visited];

    int minCost = INT_MAX;
    for (int nextCity = 0; nextCity < numCities; nextCity++) {
        if (!(visited & (1 << nextCity))) {
                int cost = distance[currentCity][nextCity] + tsp(nextCity, visited | (1 << nextCity));
            minCost = min(minCost, cost);
        }
    }

    memo[currentCity][visited] = minCost;
    return minCost;
}

int main() {
    printf("Enter the number of cities (max %d): ", MAX_CITIES);
    scanf("%d", &numCities);

    printf("Enter the distances between cities:\n");
    for (int i = 0; i < numCities; i++) {
        for (int j = 0; j < numCities; j++) {
            scanf("%d", &distance[i][j]);
        }
    }

    // Initialize memoization array
    for (int i = 0; i < numCities; i++) {
        for (int j = 0; j < (1 << numCities); j++) {
            memo[i][j] = -1;
        }
    }

    int minCost = tsp(0, 1); // start from city 0
    printf("Minimum cost for visiting all cities: %d\n", minCost);
    return 0;

}
```

**Conclusion:** Travelling Salesman Problem has been successfully implemented.