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

Experiment No. 9

Title: Travelling Salesman Problem

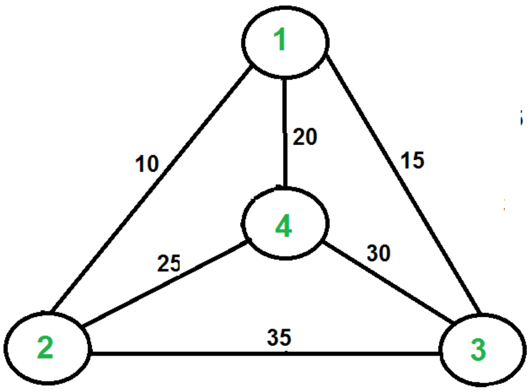
Aim: To study and implement Travelling Salesman Problem.

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.
```

Code :

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

#define V 4  // Number of vertices in the graph

// Function to find the minimum of two numbers
int min(int a, int b) {
   return (a < b) ? a : b;
}

// Function to solve TSP using dynamic programming
int tsp(int graph[][V], int mask, int pos, int n, int dp[][V]) {
   // If all vertices have been visited
   if (mask == (1 << n) - 1) {
      return graph[pos][0]; // Return cost of going back to the starting city
   }

   // If this subproblem has already been computed
   if (dp[mask][pos] != -1) {
      return dp[mask][pos];
```

```
    }

    int ans = INT_MAX;

    // Try to go to an unvisited city
    for (int city = 0; city < n; city++) {
        if ((mask & (1 << city)) == 0) { // If city has not been visited
            int newAns = graph[pos][city] + tsp(graph, mask | (1 << city), city, n, dp);
            ans = min(ans, newAns);
        }
    }

    return dp[mask][pos] = 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[1 << V][V]; // Dynamic programming table to store results of
subproblems

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

    int minCost = tsp(graph, 1, 0, V, dp); // Starting from city 0

    printf("Minimum cost of TSP: %d\n", minCost);
```

```
    return 0;
}
```

Output :

```
PS E:\Testing_Lang> cd "e:\Testing_Lang\" ; if ($?) { gcc test.c -o test } ; if ($?) { .\test }
Minimum cost of TSP: 80
PS E:\Testing_Lang> 
```

Conclusion:

Travelling Salesman Problem has been successfully implemented. This program demonstrates solving the TSP using dynamic programming. It initializes a 2D array dp to store the results of subproblems. The tsp function recursively explores all possible permutations of cities and computes the minimum cost of the tour. It uses bitmasking to keep track of visited cities efficiently.