# Lab 6

Assignment based on Dynamic Programming strategy (e.g., all pair shortest path, Traveling Salesperson problem)

# Code:

#include<bits/stdc++.h>

using namespace std;

const int MAX\_V = 10; // Define the maximum value of V

const int INF = numeric\_limits<int>::max(); // Represents infinity

int travllingSalesmanProblem(int graph[][MAX\_V], int V, int s) {

    vector<int> vertex;

    for (int i = 0; i < V; i++) {

        if (i != s)

            vertex.push\_back(i);

    }

    int min\_path = INT\_MAX;

    do {

        int current\_pathweight = 0;

        int k = s;

        for (int i = 0; i < vertex.size(); i++) {

            current\_pathweight += graph[k][vertex[i]];

            k = vertex[i];

        }

        current\_pathweight += graph[k][s];

        min\_path = min(min\_path, current\_pathweight);

    } while (next\_permutation(vertex.begin(), vertex.end()));

    return min\_path;

}

void printSolution(int dist[][MAX\_V], int V) {

    cout << "The following matrix shows the shortest distances between every pair of vertices \n";

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            if (dist[i][j] == INF)

                cout << "INF"

                     << " ";

            else

                cout << dist[i][j] << " ";

        }

        cout << endl;

    }

}

void floydWarshall(int dist[][MAX\_V], int V) {

    for (int k = 0; k < V; k++) {

        for (int i = 0; i < V; i++) {

            for (int j = 0; j < V; j++) {

                if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF && dist[i][k] != INF))

                    dist[i][j] = dist[i][k] + dist[k][j];

            }

        }

    }

    printSolution(dist, V);

}

int main() {

    int choice;

    cout << "Menu:\n";

    cout << "1. All Pairs Shortest Path\n";

    cout << "2. Traveling Salesman Problem\n";

    cout << "Enter your choice: ";

    cin >> choice;

    switch (choice) {

        case 1: {

            int V;

            cout << "Enter the number of vertices: ";

            cin >> V;

            int graph[MAX\_V][MAX\_V];

            cout << "Enter the adjacency matrix of the graph:" << endl;

            for (int i = 0; i < V; i++) {

                for (int j = 0; j < V; j++) {

                    cin >> graph[i][j];

                    if (graph[i][j] == 0 && i != j) {

                        graph[i][j] = INF; // Convert 0 to INF (no edge)

                    }

                }

            }

            floydWarshall(graph, V);

            break;

        }

        case 2: {

            int V;

            cout << "Enter the number of vertices: ";

            cin >> V;

            int graph[MAX\_V][MAX\_V];

            cout << "Enter the adjacency matrix of the graph:" << endl;

            for (int i = 0; i < V; i++) {

                for (int j = 0; j < V; j++) {

                    cin >> graph[i][j];

                }

            }

            int s;

            cout << "Enter the starting vertex (0-based indexing): ";

            cin >> s;

            cout << "Minimum cost to visit all vertices starting from vertex " << s << ": " << travllingSalesmanProblem(graph, V, s) << endl;

            break;

        }

        default:

            cout << "Invalid choice! Please enter a valid option.\n";

    }

    return 0;

}

# Results



