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

#define MAX_V 10 // Adjust the maximum number of cities if needed

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

// Function to solve the Traveling Salesman Problem using dynamic
programming
int tsp(int V, int graph[MAX_V][MAX_V], int mask, int pos, int dp[ ][1 <<
MAX_V]) {
    // If all cities have been visited
    if (mask == (1 << V) - 1)
        return graph[pos][0];

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

    int ans = INT_MAX;

    // Try to visit unvisited cities
    for (int i = 0; i < V; i++) {
        if ((mask & (1 << i)) == 0) {
            int newAns = graph[pos][i] + tsp(V, graph, mask | (1 << i), i, dp);
            ans = min(ans, newAns);
        }
    }

    // Save the result in dp table and return
    return dp[pos][mask] = ans;
}

// Function to initialize the dp table and call the tsp function
```

```
int tsp_dynamic_programming(int V, int graph[MAX_V][MAX_V]) {
    int dp[MAX_V][1 << MAX_V];

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

    return tsp(V, graph, 1, 0, dp);
}

int main() {
    int V;

    printf("Enter the number of cities (max %d): ", MAX_V);
    scanf("%d", &V);

    if (V <= 0 || V > MAX_V) {
        printf("Invalid number of cities. Exiting...\n");
        return 1;
    }

    int graph[MAX_V][MAX_V];

    printf("Enter the distance matrix:\n");

    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    int result = tsp_dynamic_programming(V, graph);

    printf("Optimal Tour Cost: %d\n", result);

    return 0;
}
```

Or

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

#define MAX_V 10

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

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

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

```
int tsp_dynamic_programming(int V, int graph[MAX_V][MAX_V]) {
    int dp[MAX_V][1 << MAX_V];

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

    return tsp(V, graph, 1, 0, dp);
}

int main() {
    int V;

    printf("Enter the number of cities (max %d): ", MAX_V);
    scanf("%d", &V);

    if (V <= 0 || V > MAX_V) {
        printf("Invalid number of cities. Exiting...\n");
        return 1;
    }

    int graph[MAX_V][MAX_V];

    printf("Enter the distance matrix:\n");

    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    int result = tsp_dynamic_programming(V, graph);

    printf("Optimal Tour Cost: %d\n", result);

    return 0;
}
```