Exp_15\b_TravellingSalesman_DP.c

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

#define N 4  // Number of cities
#define INF INT_MAX

int dist[N][N];  // Distance matrix
int dp[1 << N][N];  // DP table: dp[mask][i] = min cost to visit cities in mask ending at city i

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

// Function to solve TSP using Dynamic Programming
int tsp(int mask, int pos) {
    // If all cities have been visited
    if (mask == (1 << N) - 1) {
        return dist[pos][0];  // Return to starting city
    }

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

    int ans = INF;

    // Try visiting all unvisited cities
    for (int city = 0; city < N; city++) {
        // If city is not visited and there is a path
        if ((mask & (1 << city)) == 0 && dist[pos][city] != INF) {
            int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
            ans = min(ans, newAns);
        }
    }

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

int main() {
    // Initialize distance matrix
    printf("Enter the distance matrix (%d x %d):\n", N, N);
    printf("(Enter %d for no direct path)\n", INF);

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &dist[i][j]);
        }
    }

    // Initialize DP table with -1
```

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

    // Start from city 0 with only city 0 visited
    int minCost = tsp(1, 0);

    if (minCost >= INF) {
        printf("\nNo valid tour exists!\n");
    } else {
        printf("\nMinimum cost of TSP tour: %d\n", minCost);
    }

    return 0;
}
```