4.5 TRAVELING SALESPERSON PROBLEM WITH CITY ADDITION

Question:

Assume you are solving the Traveling Salesperson Problem for 4 cities (A, B, C, D) with known distances between each pair of cities. Now, you need to add a fifth city (E) to the problem.

AIM

To implement a solution for the Traveling Salesperson Problem in C, and extend it to include a fifth city, updating the distance matrix and computing the optimal tour.

ALGORITHM

1. Represent the cities and distances using a 2D matrix dist[i][j], where dist[i][j] is the distance from city *i* to city *j*.
2. Use a recursive function with memoization or dynamic programming to explore all permutations of city visits.
3. Maintain a visited bitmask to track cities already included in the current path.
4. For each recursive call, try visiting an unvisited city and accumulate the cost.
5. Base case: when all cities are visited, return the cost to return to the starting city.
6. Update the algorithm to handle 5 cities by expanding the matrix and bitmask size.

PROGRAM

```
from itertools import permutations

def tsp_5_cities(matrix):
    n = len(matrix)
    min_cost = float('inf')
    best_path = []

    for perm in permutations(range(1, n)):
        path = [0] + list(perm) + [0]
        cost = sum(matrix[path[i]][path[i+1]] for i in range(n))
        if cost < min_cost:
            min_cost = cost
            best_path = path

    return min_cost, best_path


print("Enter distances between cities A, B, C, D, E (5x5 matrix):")
city_labels = ['A', 'B', 'C', 'D', 'E']
matrix = []

for i in range(5):
    row = list(map(int, input(f"Row {i+1} ({city_labels[i]}): ").split()))
    if len(row) != 5:
        print("Each row must contain exactly 5 integers.")
        exit()
    matrix.append(row)

cost, path = tsp_5_cities(matrix)
route = ' -> '.join(city_labels[i] for i in path)
print(f"\n Minimum path cost: {cost}")
print(f"Optimal route: {route}")
```

Input:

Enter the distance matrix (5x5):
0 10 15 20 25
10 0 35 25 30
15 35 0 30 20
20 25 30 0 15
25 30 20 15 0

Output:

```
Enter distances between cities A, B, C, D, E (5x5 matrix):
Row 1 (A): 0 10 15 20 25
Row 2 (B): 10 0 35 25 30
Row 3 (C): 15 35 0 30 20
Row 4 (D): 20 25 30 0 15
Row 5 (E): 25 30 20 15 0

 Minimum path cost: 85
Optimal route: A -> B -> D -> E -> C -> A
>>> 
```

RESULT:

Thus the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: $O(n^2 \times 2^n)$
- Space Complexity: $O(n \times 2^n)$