DESIGN AND ANALYSIS OF ALGORITHM

PRACTICAL-5

NAME: VEDANT BHUTADA

ROLL: 69

BATCH: A4

Aim: A traveling salesman is getting ready for a big sales tour. Starting at his hometown, suitcase in hand, he will conduct a journey in which each of his target cities is visited exactly once before he returns home. Given the pairwise distances between cities, what is the best order in which to visit them, so as to minimize the overall distance traveled?

```
def tsp(C, source):
    n = len(C)

    # function to calculate the minimum distance and path for a set of cities
    def g(i, S):
        if not S:
            return (C[i - 1][source - 1], [i, source]) if i != source else (float('inf'), [])

        min_distance = float('inf')
        min_path = []
        for j in S:
            if j != i:
                distance, path = g(j, S - {j})
                distance += C[i - 1][j - 1]
                if distance < min_distance:
                    min_distance = distance
                    min_path = [i] + path

        return min_distance, min_path

    def tsp_for_source(source):
        all_cities = set(range(1, n + 1))
        all_cities.remove(source)

        # Calculate the minimum distance and path starting and ending at the source city
        min_distance, min_path = g(source, all_cities)

        # Add the final leg of the path to return to the source
        min_distance += C[min_path[-1] - 1][source - 1]
        #min_path.append(source)

        return min_distance, min_path

    results = {}
    for source_city in range(1, n + 1):
        best_distance, best_path = tsp_for_source(source_city)
        results[source_city] = (best_distance, best_path)

    return results

# Define a distance matrix C where C[i][j] represents the distance between city i and city j.
C = [[0, 10, 15, 20],
     [5, 0, 9, 10],
     [6, 13, 0, 12],
     [8, 8, 9, 0]]

# Calculate best TSP distances and paths for cities 1, 2, 3, and 4
for source_city in range(1, len(C) + 1):
    result = tsp(C, source_city)
    best_distance, best_path = result[source_city]
    print(f"best TSP Distance from city {source_city} to city {source_city}: {best_distance}")
    path_str = ' -> '.join(map(str, best_path))
    print("best TSP Path:", path_str)
```

```
best TSP Distance from city 1 to city 1: 35
best TSP Path: 1 -> 2 -> 4 -> 3 -> 1
best TSP Distance from city 2 to city 2: 35
best TSP Path: 2 -> 4 -> 3 -> 1 -> 2
best TSP Distance from city 3 to city 3: 35
best TSP Path: 3 -> 1 -> 2 -> 4 -> 3
best TSP Distance from city 4 to city 4: 35
best TSP Path: 4 -> 3 -> 1 -> 2 -> 4
```