Day 19

Assignment 2: Traveling Salesman Problem

Create a function int FindMinCost(int[,] graph)
that takes a 2D array representing the graph where graph[i][j] is the cost to travel from city i to city j. The function should returnthe minimum cost to visit all cities and return to the starting city. Use dynamic programming for this solution.

A)
Java implementation of the Traveling Salesman Problem (TSP) using dynamic programming. This approach utilizes bitmasking to represent subsets of cities and memoization to store already computed states, making it efficient for solving the problem.

Java code for the FindMinCost function:

```
import java.util.Arrays;
public class TravelingSalesman {

  // Function to find the minimum cost to visit all cities and return to the starting city
  public static int FindMinCost(int[][] graph) {
    int n = graph.length;
    int[][] dp = new int[n][1 << n];
    for (int[] row : dp) {
      Arrays.fill(row, -1);
    }
    return tsp(0, 1, graph, dp);
  }

  // Helper function for the TSP using dynamic programming and bitmasking
  private static int tsp(int pos, int mask, int[][] graph, int[][] dp) {
    int n = graph.length;

    // Base case: if all cities have been visited
    if (mask == (1 << n) - 1) {
      return graph[pos][0]; // Return to the starting city
    }

    // Check if the result is already computed
    if (dp[pos][mask] != -1) {
      return dp[pos][mask];
    }

    int minCost = Integer.MAX_VALUE;

    // Try to go to an unvisited city
    for (int city = 0; city < n; city++) {
      if ((mask & (1 << city)) == 0) {
```

```
            int newCost = graph[pos][city] + tsp(city, mask | (1 << city), graph, dp);
            minCost = Math.min(minCost, newCost);
        }
      }

      // Save the result in dp array and return
      dp[pos][mask] = minCost;
      return minCost;
   }

   public static void main(String[] args) {
      // Example usage
      int[][] graph = {
         {0, 10, 15, 20},
         {10, 0, 35, 25},
         {15, 35, 0, 30},
         {20, 25, 30, 0}
      };

      System.out.println("The minimum cost to visit all cities and return to the starting city is: " +
FindMinCost(graph));
   }
}
```

Explanation:

1. Initialization:

- dp is a 2D array where dp[i][mask] represents the minimum cost to visit all cities in the subset represented by mask starting from city i.
- mask is a bitmask where the bit at position j is set if city j is visited.

2. Recursive Function tsp:

- pos is the current city.
- mask is the current set of visited cities.
- If all cities are visited (mask == (1 << n) - 1), return the cost to return to the starting city.
- If the result for this state is already computed, return the saved result.
- Iterate over all cities and recursively compute the cost for the unvisited cities, updating the minimum cost.

3. Main Function FindMinCost:

- Initializes the dp array and calls the tsp function starting from city 0 with the initial mask 1 (indicating that the first city is visited).

4. Example Usage:

- The main method provides an example graph and prints the minimum cost to visit all cities and return to the starting city.

This approach ensures that each state is computed only once, making it efficient for solving the Traveling Salesman Problem for a moderate number of cities.