Problem Set: Implementing TSP with Greedy Dijkstra

Problem 1: Implementing Dijkstra's Algorithm

1. Write a Python function dijkstra(matrix, start) that computes the shortest paths from a given starting node to all other nodes in a graph represented by an adjacency matrix.
    - Input:
        - matrix: A 2D list (adjacency matrix) where matrix[i][j] represents the cost of traveling from city i to city j. If there is no direct path, the value is 0.
        - start: The index of the starting city.
    - Output:
        - A list distances where distances[i] is the shortest distance from the starting city to city i.
    - Example:

```
matrix = [
  [0, 10, 15, 20],
  [10, 0, 35, 25],
  [15, 35, 0, 30],
  [20, 25, 30, 0]
]
start = 0
print(Dijkstra(matrix, start))  # Output: [0, 10, 15, 20]
```

Problem 2: Implementing Greedy Best-First Search for TSP

2. Write a Python function tsp_greedy_dijkstra(matrix, start) that solves the Traveling Salesman Problem using a **Greedy Best-First Search** approach.
    - Input:
        - matrix: A 2D list (adjacency matrix) representing the graph.
        - start: The index of the starting city.
    - Output:
        - A tuple (path, total_cost) where:
            - path: A list of city indices representing the order of cities visited (including returning to the starting city).
            - total_cost: The total cost of the path.
    - Requirements:
        - Use the dijkstra function from Problem 1 to precompute the shortest paths between all pairs of cities.
        - At each step, choose the nearest unvisited city based on the precomputed shortest paths.
        - Return to the starting city after visiting all cities.
    - Example:

```
matrix = [
  [0, 10, 15, 20],
  [10, 0, 35, 25],
  [15, 35, 0, 30],
  [20, 25, 30, 0]
]
start = 0
```

```
print(tsp_greedy_dijkstra(matrix, start))  # Output: ([0, 1, 3, 2, 0], 80)
```

Problem 3: Testing the Algorithm

3. Test your tsp_greedy_dijkstra function on the following graphs:
    - Graph 1:

      ```
      matrix = [
          [0, 10, 15, 20],
          [10, 0, 35, 25],
          [15, 35, 0, 30],
          [20, 25, 30, 0]
      ]
      start = 0
      ```

    - Graph 2:

      ```
      matrix = [
          [0, 20, 42, 35],
          [20, 0, 30, 34],
          [42, 30, 0, 12],
          [35, 34, 12, 0]
      ]
      start = 2
      ```

    - Graph 3:

      ```
      matrix = [
          [0, 3, 0, 7, 9],
      ```

[3, 0, 2, 0, 0],

[0, 2, 0, 1, 0],

[7, 0, 1, 0, 5],

[9, 0, 0, 5, 0]

]

start = 4

- For each graph, print the path and total cost.