PFL 2024/2025 Practical Assignment 1

Group Members - T10_G05

- **Member 1**: Eduardo Cunha (`up202207126`)

  Contribution: 50%

  - Tasks:
    - Functions `cities`, `distance`, `pathDistance` and `isStronglyConnected`.
    - Implemented the `travelSales` function, including the greedy TSP approximation and auxiliary functions.

- **Member 2**: Miguel Sousa (`up202207986`)

  Contribution: 50%

  - Tasks:
    - Functions `areAdjacent`, `adjacent` and `rome`
    - Implemented the `shortestPath` function, including the Dijkstra algorithm and auxiliary functions.

Implementation of `shortestPath`

Explanation

The `shortestPath` function computes the shortest path between two cities in a given `RoadMap`. It uses Dijkstra's algorithm to find the shortest path. The function returns a list of paths, where each path is represented as a list of cities.

Auxiliary Data Structures

- **`RoadMap`**: A list of tuples representing the connections between cities and their distances.
- **`Path`**: A list of cities representing a path.
- **`Distance`**: An integer representing the distance between cities.

Algorithm

1. **Initialization**: Start with the initial city and set its distance to 0.
2. **Distance Update**: Use the `updateDistances` function to update the distances of adjacent cities.
3. **Find Minimum Distance**: Use the `findMinDistance` function to find the city with the minimum distance that has not been visited.
4. **Path Construction**: Construct the path by recursively visiting cities with the minimum distance until the goal city is reached.

Justification

- Dijkstra's algorithm is chosen for its efficiency in finding the shortest path in graphs with non-negative weights.

- The use of lists for distances and paths simplifies the implementation and is sufficient for the problem size.

Implementation of `travelSales`

Explanation

The `travelSales` function computes a path that visits all cities in the graph and returns to the starting city, with the smallest total distance. It uses a greedy approximation algorithm for the Traveling Salesman Problem (TSP).

Auxiliary Data Structures

- **`RoadMap`**: A list of tuples representing the connections between cities and their distances.
- **`Path`**: A list of cities representing a path.
- **`Distance`**: An integer representing the distance between cities.

Algorithm

1. **Initialization**: Start from an initial city and initialize the path and total distance.
2. **Nearest Neighbor Heuristic**: Use the `closestCity` function to find the closest unvisited city and add it to the path.
3. **Path Construction**: Recursively visit the closest unvisited city until all cities are visited.
4. **Return to Start**: Add the starting city to the end of the path to complete the cycle.

Justification

- The nearest neighbor heuristic is chosen for its simplicity and reasonable performance for small to medium-sized graphs.
- The use of lists for paths and distances simplifies the implementation and is sufficient for the problem size.