Faculdade de Engenharia da Universidade do Porto

Programação Lógica e Funcional

Projeto 1

Group - T05_G13

Adriana Almeida - up202109752
Margarida Fonseca - up202207742

Group Members Contribution

Adriana Almeida - 50%
Margarida Fonseca - 50%

Tasks Performed

To complete this project, we divided the workload and met frequently to explain our progress to each other. This was important because the project followed a sequential structure, and solving certain problems required a solid understanding of the preceding functions.

The first six functions were divided as follows:

- Adriana worked on functions 1, 3, and 5.
- Margarida worked on functions 2, 4, and 6.

To complete the last three functions (7, 8, and 9), we mostly worked together.

shortestPath Implementation

The primary goal of this module is to provide an efficient way to find the shortest path between two cities in a given road map. The implementation utilizes Dijkstra's algorithm, which is well-suited for graphs with non-negative weights.

Types

The following types are defined in the module:

- AdjacencyMatrix: An array representation of the graph (not used in the current implementation).
- PathMatrix: An array representation of paths (not used in the current implementation).

Functionality

The shortestPath function finds the shortest path between two cities in a road map using Dijkstra's algorithm.

- Parameters roadmap: A list of tuples representing the roads between cities and their distances.(start : the starting city; end: the destination city)

- Returns a list of cities representing the shortest path from the starting city to the destination city. If no path exist, it returns an empty list.

Implementation Details

- Initialize Distances and Paths: It initializes distances and paths for each city, setting the distance to the start city as 0 and all others as infinity.
- Dijkstra's Algorithm: iteratively finds the closest unvisited city and updates the distances and paths of its neighbors. -It continues until the destination city is reached or all reachable cities are visited.
- Helper Functions: Several helper functions manage distance and path lookups, updates, and unique city extraction.

travelSales **Implementation**

The travelSales function is designed to find the shortest possible path that visits all cities in a RoadMap and returns the optimal solution.

- This function starts by checking if the graph is connected, if it isn't that nad empty list returns since a graph that is not strongly connected may not have a solution to this problem
- The main function creates a variable allCities (with all cities of the RoadMap) and calls function *aux*
- Function *aux* (auxiliar) will recursively build all potential paths that visit each city exactly once. It uses backtracking to explore all possible routes from the starting city to all other cities.
    - For each city, *aux* finds all adjacent cities that haven't been visited yet. It then recurses on each unvisited neighboring city, adding it to the current path (and marking it as visited (visited cities are saved in a list of cities)
    - When no more unvisited adjacent cities are found, aux checks if the current city can connect back to the start. If it can, the path is closed by adding the starting city to the end,if not it appends "No Direct Connection" as a marker for invalid path.
    - Once all potential paths have been explored, aux filters out invalid paths and paths that don't cover all cities.
    - Then it calculates the distance off all the valid paths and checks for the path with the minimum distance.
- To calculate the distance between all paths without doing to many recalculations a memoization table is implemented
    - The table is a list of lists where each entry corresponds to the distance between the cities that correspond to the entries.
    - In order to map each city to an index and easily use the memoization table mentioned above, an Array mapping each city to the corresponding integer index was created.
    - The selection of the structure used, the Array, is based on the fact that the array allows for lookups with a time complexity of O(n).