Practical Assignment 1 - PFL 2024/2025

Group Members and Contributions

- **Filipe Gaio**
  - Contribution: 50%
  - Tasks: All the first easier functions were done in group, both elements worked equally in all of them. Worked on the `dijkstras` and `shortestPath` functions.
- **Henrique Fernandes**
  - Contribution: 50%
  - Tasks: All the first easier functions were done in group, both elements worked equally in all of them. Worked on the `TSP` related functions and all its design.

Shortest Path Function Implementation

Explanation

The `shortestPath` function was implemented using Dijkstra's algorithm to find the shortest path between two cities in a roadmap. Then we do a DFS pruning all paths that exceed the already achieved shortest path' distance to find is there's any other path that also has minimal distance from start to end point. The function returns all paths that have the shortest distance.

Auxiliary Data Structures

- **Priority Queue**: Used to keep track of the cities to be explored, prioritized by their current known shortest distance from the start city.
- **Visited List**: Keeps track of cities that have already been visited to avoid reprocessing them.
- **Paths List**: Stores the current shortest path to each city along with the distance.

Algorithm

1. **Initialization**: Start with the initial city and a distance of 0. Initialize the priority queue with this city.
2. **Exploration**: While there are cities in the priority queue:
   - Extract the city with the smallest known distance.

    - If this city is the destination, return the path and distance.
    - For each adjacent city, calculate the new distance. If this distance is shorter than any previously known distance, update the priority queue and paths list.
3. **Dijkstra's Result**: If the destination city is reached, return the path and distance. Otherwise, return an empty list. The complexity of running Dijkstra's Algorithm is $O(V + E)logV$.
4. **DFS**: We then reiterate through the graph doing a depth first search to check if there are any other paths that also have the same distance as the one found by the dijktras. To be more efficient we prune all the paths that already exceed the minimum distance achieved by the Dijkstras shortest path. The complexity of the DFS is $O(V + E)$, so, since the Dijkstra's complexity is higher, the final time complexity is $O(V + E)logV$.

Travel Sales Function Implementation

Explanation

The `travelSales` function is intended to solve the Traveling Salesman Problem (TSP) for the given roadmap. The implementation is based of the book "Algorithms: a functional programming approach", by Fethi Rabbi and Guy Lapalme. The main difference is that the algorithm in the book assumes the graph is complete (there is a edge between every pair of nodes), while in the project that may not happen.

Auxiliary Data Structures

- **Set** : The `Set` is used to represent a list of integers in a single integer, where each bit of the `Set` represents the presence or absence of an element. It allows for efficient set operations using bitwise arithmetic.
- **Table** : The `Table` is used to store the results of the dynamic programming. In the `TSP`, the table can be indexed using `TspCoord`, which is a tuple where the first element represents the node and the second is a `Set`. Each entry of the table is a `TspEntry`, which is a tuple where the first element is an integer that represents the total distance, and the second is a list of integers that represent the shortest path.

Algorithm

1. **Initialization**: Each city is represented as a node. If the set of intermediate nodes is empty, we check the direct edge between two cities.
2. **Exploration (Dynamic Programming Computation)**: We compute the shortest path for every subset of nodes using `compTsp`. This function calculates the

shortest route for the TSP by checking all possible paths from the current city to the destination. Due to the use of dynamic programming, this problem is solved with complexity $O(n^2 2^n)$.

Types

The types used in this project are defined as follows:

```haskell
type City = String
type Distance = Int
type Path = [City]
type RoadMap = [(City, City, Distance)]
type Set = Int
newtype Table a b = Tbl (Data.Array.Array b a)
  deriving (Show)
type TspCoord = (Int, Set)
type TspEntry = (Int, [Int])
```