PFL - Project 1

Group Members

Name	UP Number	Contribution
João Paulo Silva Santos	202006525	50%
Leonardo de Sousa Magalhães Teixeira	202208726	50%

We contributed equally to the project, working closely together on the design, coding and testing of each function. We also shared responsibility for planning and writing the report to ensure everything was thorough, clear and well-organized.

Shortest Path Implementation Overview

We implemented the `shortestPath` using Breadth-First Search to find the shortest path between two cities in the RoadMap. ### `bfs` Function - The `bfs` function explores all possible paths from the current city to the goal city using a breadth-first approach, collecting all valid paths between the start city and the goal city, keeping track of their total distance.

- We use a queue to manage the paths being explored. Each entry in the queue consists of a list of cities (the current path) and the total distance traveled. We defined a Queue type for improved readibility.

```
type Queue = [(Path, Distance)]
```

`bfsPaths` Function

- This function initiates the BFS process to find all paths from the starting city to the goal city.
- It calls the `bfs` function starting with a queue containing the initial city and a distance of 0. It collects all valid paths discovered during the search.

`shortestPath` Function

- This function finds the shortest path between two specified cities.
- It uses `bfsPaths` to retrieve all paths from the start to the goal city. It then identifies the minimum distance from these paths and filters the results to return only the paths with the minimum distance. It also ensures that the shortest path from a city `c` to itself is [`c`].

Traveling Salesperson Problem Implementation Overview

We implemented a solution to the Traveling Salesperson Problem using a modified Breadth-First Search approach to find a path that visits all cities in a roadmap and returns to the starting city.

`bfsTsp` Function

This function performs a breadth-first search to explore all possible paths from the starting city, attempting to visit all cities in the roadmap. - The function dequeues the first path from the queue and checks if all cities have been visited. - If all cities are visited, it adds the current path and its distance to the result list. - If not, it appends all not visited adjacent cities to the queue, building new paths with new distances. - This continues until all paths have been explored and stored in the result list.

`minCostPath` Function

The `minCostPath` function returns the path with the smallest distance from a list of paths.

`findPathWithReturn` Function

This function finds the shortest possible path that visits all cities exactly once and ends at a city adjacent to the starting city. - Calls `bfsTsp` to get a list of all valid paths that visit every city. - Filters for paths that start and end at adjacent cities. - If there are valid paths, it returns the one with the minimum distance, otherwise it returns `Nothing`.

`travelSales` Function

The `travelSales` function is the main function that completes the TSP tour. - Defines the starting city as the first city in the cities list. - Calls `findPathWithReturn` to get a path that completes the TSP. - If `findPathWithReturn` returns `Nothing`, it retuns `[]`. Otherwise, it appends the `startCity` at the end of the path, returning a complete tour.

Property-based Testing

In this project, we tested our functions using property-based testing to test their robustness and correctness. In order to do the tests, we imported 'Test.QuickCheck', that allowed us to use the quickCheck function, that generates multiple inputs to ensure that our functions work as expected across various scenarios. By making these tests, we were able to spot an error in the 'isStronglyConnected' function, that was not handling the case where the roadmap is empty.

```
ghci> runTests
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
```

```
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
```

How to run QuickCheck

1. Import `Test.QuickCheck` on the project code.

```
import Test.QuickCheck
```

2. Uncomment the `runTests` function in the project code.
3. Install the QuickCheck library.

```
$ cabal install --lib QuickCheck
```

4. Load the project in ghci

```
$ ghci project1.hs
```

   In case the compiler gives an error like `Could not load module Data.Array`, use the command `:set -package array` inside ghci. This will load a local environment package, which can later be removed by deleting the file specified in the message after entering `ghci`.