PFL Project 1 - Haskell

Grup 08 - T06

up202208511 - Tiago Morais Pimentel Teixeira (50%)
up202300600 - Yuka Sakai (50%)

- Each of us equally participated in the coding of each of the functions and helped the other when necessary.

Installation and Execution

To execute this code, you must ensure that your machine can run GHCi version 9.10.1 or a newer version installed, along with the source code folder. Once you have it installed, open the interpreter and load the .hs files located in the root directory of the source code.

Project Description

This project's goal consists of writing a set of functions and data types to represent countries as a graph of cities (vertices) and the connections between them (edges).

Imports

```
import qualified Data.List
import qualified Data.Array
import qualified Data.Bits
import qualified Data.Ord
```

Types

To represent the graphs, we made use of the following type, that represents each of the edges (consisting of two nodes and a distance):

```
type RoadMap = [(City,City,Distance)]
```

Which must use the following auxiliary types:

```
type City = String
type Distance = Int
type Path = [City]
```

Function Description

1. cities

- **Definition**: Used to return all the cities (unique) in the graph.
- **Arguments**: roadMap
- **Returns**: List of city

2. **areAdjacent**

   - **Definition**: Indicates whether two cities (cidade1, cidade2) are linked directly or not.
   - **Arguments**: roadMap, cidade1, cidade2
   - **Returns**: Boolean

3. **distance**

   - **Definition**: Indicates the distance between two cities (cidade1, cidade2) when they are connected directly and Nothing otherwise.
   - **Arguments**: roadMap, cidade1, cidade2
   - **Returns**: Just value / Nothing

4. **adjacent**

   - **Definition**: Indicates the cities adjacent to a particular given city (i.e. cities with a direct edge between them) and the respective distances to them in a list of tuples.
   - **Arguments**: roadMap, cidade
   - **Returns**: [(city1, dist)]

5. **pathDistance**

   - **Definition**: Indicates the sum of all individual distances in a given path between two cities if all the consecutive pairs of cities are directly connected by roads. Otherwise, it returns a Nothing. It uses pathDistanceReturn and pairCities as auxiliar functions.
   - **Arguments**: roadMap, path
   - **Returns**: Just value / Nothing
   - Auxiliar Functions:
     1. **pairCities**
        - **Definition**: Make pairs of cities using zip to auxiliate in pathDistance and isValidPath functions.
        - **Arguments**: [city]
        - **Returns**: [(city, city)]
     2. **pathDistanceReturn**
        - **Definition**: Calculate the total distance in the path using recursion.
        - **Arguments**: roadMap, result from pairCities
        - **Returns**: Just value / Nothing

6. **rome**

   - **Definition**: Gives the names of the cities with the highest number of roads connecting to them (i.e. the vertices with the highest degree). It uses auxiliar function highestDegree and compareByDegree.

- **Arguments**: roadMap
- **Returns**: [city]
- Auxiliar Functions:
  1. **highestDegree**
     - **Definition**: Takes as argument a list of tuples and collects the highest degree from all cities.
     - **Arguments**: [(city, (length (adjacent roadMap city)))]
     - **Returns**: [city]
  2. **compareByDegree**
     - **Definition**: Custom comparison function for sorting by the second element of a tuple.
     - **Arguments**: (x, degree1), (y, degree2)
     - **Returns**: Ordering

7. isStronglyConnected

- **Definition**: Indicates whether all the cities in the graph are connected in the roadmap (i.e., if every city is reachable from every other city). It uses auxiliar functions as dfs and traverseAll.
- **Time Complexity**: O((V^2) * (V+E)) - where P is the length of a path.
- **Arguments**: roadMap
- **Returns**: Boolean
- Auxiliar Functions:
  1. **dfs**
     - **Definition**: Marks all the visited cities using recursion, being also a function to be used in traverseAll.
     - **Arguments**: roadMap
     - **Returns**: [city]
  2. **traverseAll**
     - **Definition**: Used to check if all cities are reachable from a given starting city.
     - **Arguments**: roadMap, start city, [city]
     - **Returns**: Boolean

8. shortestPath

- **Definition**: Computes all shortest paths that connect two cities (source and target) given as input, by making use of a Dijkstra algorithm. If two or more paths have the same shortest distance, they will be returned. It calls getShortestPaths in order to filter through all paths found by the Dijkstra Algorithm, between source and target.
- **Time Complexity**: O((E+V) * log(V)+ (P*E)) - where P is the length of a path.
- **Arguments**: roadMap, source, target
- **Returns**: [Path]
- Auxiliar Functions:

1. **getShortestPaths**
   - **Definition**: Collects paths that have a distance equal to the smallest one found.
   - **Arguments**: roadMap, source, target
   - **Returns**: [Path]
2. **dijkstra**
   - **Definition**: Calculates all possible paths from source to target nodes.
   - **Arguments**: roadMap, ((dist, path):queue), visited, shortestPaths, target
   - Returns: [Path]
3. **compareByDistance**
   - **Definition**: Custom comparison function for sorting by the first element of a tuple.
   - **Arguments**: (distance1, x), (distance2, y)
   - **Returns**: Ordering

9. travelSales

- **Definition**: Returns a solution of the Traveling Salesman Problem (TSP) where it has to visit each city exactly once and come back to the start point to find the shortest route (total distance is minimum). If the graph does not have a TSP path, then return an empty list.
- **Time Complexity**: O(V!)*((P*E)+E) - where P is the length of a path.
- **Arguments**: roadMap
- **Returns**: Path
- Auxiliar Functions:
  1. **isValidPath**
     - **Definition**: Checks if consecutive cities in a path are connected.
     - **Arguments**: roadMap, path
     - **Returns**: Bool
  2. **minimumByDistance**
     - **Definition**: Calculates the path with smallest distance from a list of paths.
     - **Arguments**: roadMap, paths
     - **Returns**: Path

Conclusion

Upon the conclusion of the project, we can affirm that out of all the functions, we had the most difficulty coding shortestPath, more specifically with the auxiliar function that defined a Dijkstra algorithm, as it involved many steps that initially weren't providing us with the right solutions.