FEUP-L.EIC024-2024/2025-1S

Início	sexta-feira, 8 de novembro de 2024 às 18:28
Estado	Prova submetida
Data de submissão:	sexta-feira, 8 de novembro de 2024 às 20:13
Tempo gasto	1 hora 45 minutos
Nota	**12,00** de um máximo de 15,00 (**80**%)

Informação

Information and Rules:

- In this part of the test, you can consult any materials available on the local computer, but not printed materials.
- **The presence of electronic and/or communication devices is forbidden.**
- Any attempt of fraud will be punished with the annulment of the test to all intervenients.
- The maximum duration of the test is as stated below.
- The score of each question is stated below.
- You should answer all questions in the respective answer boxes.
- **You may use GHCI to test your code.**
- **Functions requested in previous questions can be used in the following ones of the same Moodle page, even if you have not implemented them. Do not copy the code from earlier questions to the following ones!**
- **If you implement additional auxiliary functions, you must include them in the answers to all questions where you use them.**
- **Do not copy code from the problem statements (e.g., the "type" and "data" declarations) to the answer boxes!**
- **If you want to write a comment, do it in Haskell: add "--" before the text or use "{-" and "-}."**
- **Document your code and use intuitive variable names to clarify the program's interpretation.**
- **You cannot use library functions or import modules, unless clearly stated otherwise.**
- **Pay attention to syntactic errors (such as wrong code identation).**

Informação

Consider the following types to represent a football league.

```
type Match = ((String,String), (Int,Int))
type MatchDay = [Match]
type League = [MatchDay]

myLeague :: League
myLeague = [
  [(("Porto","Sporting"),(2,2)),(("Benfica","Vitoria SC"),(4,0))],
  [(("Porto","Benfica"),(5,0)),(("Vitoria SC","Sporting"),(3,2))],
  [(("Vitoria SC","Porto"),(1,2)),(("Sporting","Benfica"),(2,1))]
  ]
```

A football league is made up of a sequence of match days. In each match day, a set of matches occurs. The match is a structure with the name of the home and away team, which play against each other, alongside the number of goals of each team. For instance, in the match where Porto was the home team and scored 5 goals and Benfica was the away team and scored 0 goals, the respective value of Match is: (("Porto","Benfica"),(5,0)).

Assume that all teams play exactly one match in each matchday.

To help you test your functions, consider the league presented above.

Pergunta 1

Pontuou 1,000 de 1,000

Implement winner :: Match → String, which, given a match, returns the name of the match's winner or "draw" in case both teams scored the same number of goals.

Examples:

```
ghci> winner (("Porto","Benfica"),(5,0))
"Porto"
ghci> winner (("Porto","Sporting"),(2,2))
"draw"
```

```
winner :: Match → String
winner ((home, away), (hScore,
aScore))
  | hScore > aScore = home
  | hScore < aScore = away
  | otherwise       = "draw"
```

Pergunta 2

Pontuou 1,200 de 1,200

Implement matchDayScore :: String → MatchDay → Int, which, given a team and a matchday, returns the score obtained by that team on their match of that matchday. Winning a match gives a team 3 points, drawing gives 1 point and losing gives 0 points. If the input team does not exist in the league, then return 0.

Examples:

```
ghci> matchDayScore "Porto" (myLeague !! 0)
1
ghci> matchDayScore "Benfica" (myLeague !! 0)
3
ghci> matchDayScore "Vitoria SC" (myLeague !! 0)
0
```

```
participatesIn :: String → Match →
Bool
participatesIn team ((home, away), _)
= team == home || team == away

matchDayScore :: String → MatchDay
→ Int
matchDayScore team [] = 0
matchDayScore team
(match:subDay)
  | not $ team `participatesIn` match
= matchDayScore team subDay
  | winTeam == team               = 3
  | winTeam == "draw"              = 1
  | otherwise                   = 0
```

Informação

To help you out in the next question, you can use the leagueScore and sortByCond functions, presented below, which you can also use later on this exam if needed. DO NOT COPY the code of these three functions to any of the answer boxes of this exam. Just call the functions and assume they will run using the code below.

```
leagueScore :: String -> League -> Int
leagueScore t = foldr (\d acc -> matchDayScore t d + acc) 0

sortByCond :: Ord a => [a] -> (a -> a -> Bool) -> [a]
sortByCond [] _ = []
sortByCond [x] _ = [x]
sortByCond l cmp = merge (sortByCond l1 cmp) (sortByCond l2 cmp) cmp
  where (l1 ,l2) = splitAt (div (length l) 2) l

merge :: Ord a => [a] -> [a] -> (a -> a -> Bool) -> [a]
merge [] l _ = l
merge l [] _ = l
merge (x:xs) (y:ys) cmp
  | cmp x y = x:(merge xs (y:ys) cmp)
  | otherwise = y:(merge (x:xs) ys cmp)
```

Pergunta 3

Pontuou 1,600 de 1,600

Implement ranking:: League → [(String,Int)], which returns a list of pairs (team,score) with the score obtained by each team during the league. The league score is the sum of the points obtained by the team in each matchday. The output list must be ordered by the scores, in decreasing order. In case of a tie in the scores, return the team name that comes first alphabetically.

Examples:

```
ghci> ranking myLeague
[("Porto",7),("Sporting",4),("Benfica",3),("Vitoria SC",3)]
ghci> ranking (tail myLeague)
[("Porto",6),("Sporting",3),("Vitoria SC",3),("Benfica",0)]
ghci> ranking []
[]
```

```
nub :: (Eq a) ⇒ [a] → [a]
nub [] = []
nub (x:xs) = x : nub (filter (/= x) xs)

getTeams :: League → [String]
```

Informação

Note: In the following exercises, there are constraints over the programming techniques. Solutions that do not follow the constraints will receive at most 25% of the question score.

Pergunta 4

Pontuou 1,600 de 1,600

Implement numMatchDaysWithDraws :: League → Int, which returns the amount of matchdays in the league where at least one match ended in a draw.

Constraint: in this exercise, you must use higher-order functions. You are not allowed to define new recursive functions or use list comprehensions.

Example:

```
ghci> numMatchDaysWithDraws myLeague
1
```

```
numMatchDaysWithDraws :: League →
Int
numMatchDaysWithDraws = length .
filter (any ((== "draw") . winner))
```

Pergunta 5

Pontuou 1,600 de 1,600

Implement bigWins :: League → [(Int,[String])], which returns a list of pairs (i,ws), where i is the index of the matchday (with indices starting at 1) and ws are the names of teams from that matchday that won their match with a goal difference of at least 3 goals. Any order of the winners ws of each matchday will be accepted.

Constraint: in this exercise, you must use list comprehensions. You are not allowed to define new recursive functions or use higher-order functions (map, filter, foldl ...).

Example:

```
ghci> bigWins myLeague
[(1,["Benfica"]),(2,["Porto"]),(3,[])]
```

```
bigWins :: League → [(Int,[String])]
bigWins league = [(i, [winner match |
match@(_ , (score1, score2)) ←
matchDay, abs (score1 - score2) >= 3])
| (i, matchDay) ← zip [1..] league]
```

Pergunta 6

Pontuou 2,000 de 2,000

Implement winningStreaks :: League → [(String,Int,Int)], which returns a list of winning streaks, described by a triple (t,s,e), where:

- t is the name of the team involved in the winning streak;
- s and e are the indices of the streak's first and last matchday (with indices starting at 1).

A winning streak is a sequence of wins in 2 or more consecutive matchdays.

Note that the same team may have more than one winning streak and thus appear multiple times in the output. Any order of the output list will be accepted.

Examples:

```
ghci> winningStreaks myLeague
[("Porto",2,3)]
ghci> winningStreaks [[(("RM","Barca"),(4,0))],[(("Barca","RM"),(1,2))],[(("RM","Barca"),(2,2))],[(("RM","Barca"),
(0,4))],[(("Barca","RM"),(3,0))],[(("Barca","RM"),(3,1))]]
[("RM",1,2),("Barca",4,6)]
```

```
-- Equal to nub, alias to prevent
name collisions
nub2 :: (Eq a) ⇒ [a] → [a]
nub2 [] = []
nub2 (x:xs) = x : nub2 (filter (/= x) xs)

-- Equal to getTeams, alias to prevent
name collisions
getTeams2 :: League → [String]
getTeams2 league = nub2 $ [home |
day ← league, ((home, _), _) ← day]
++ [away | day ← league, ((_, away),
_) ← day]

getWins :: String → League →
```

Informação

Consider the following types from your Haskell practical assignment, to represent roadmaps, where roads of a given distance connect cities. To help you out during testing, two roadmaps are provided.

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

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

gTest1 :: RoadMap
gTest1 = [("0","1",10),("0","2",15),("0","3",20),("1","2",35),("1","3",25),("2","3",30)]

gTest2 :: RoadMap -- unconnected graph
gTest2 = [("0","1",4),("2","3",2)]
```

Pergunta 7

Pontuou 1,250 de 1,250

Implement adjacent :: RoadMap → City →[(City,Distance)], which returns the cities adjacent to a particular city (i.e. cities with a direct edge between them) and the respective distances to them. Any order of the output list will be accepted. Note that a city is NOT adjacent to itself.

Examples:

```
ghci> adjacent gTest1 "0"
[("1",10),("2",15),("3",20)]
ghci> adjacent gTest2 "0"
[("1",4)]
```

```
adjacent :: RoadMap → City →
[(City,Distance)]
adjacent roadMap city = [(dest,dist) |
(orig,dest,dist) ← roadMap, orig ==
city] ++ [(orig,dist) | (orig,dest,dist)
```

Pergunta 8

Pontuou 1,750 de 1,750

Implement areConnected :: RoadMap → City → City → Bool, which determines if there is a path, composed of one or more roads, connecting both input cities of the input roadmap. Consider that a city is connected to itself.

Examples:

```
ghci> areConnected gTest1 "0" "3"
True
ghci> areConnected gTest2 "0" "3"
False
```

```
dfs :: RoadMap → City → [City] →
[City] → [City]
dfs _ _ visited [] = visited
dfs roadMap city visited
(top:stackTail)
  | top `elem` visited = dfs roadMap
city visited stackTail
  | otherwise          = dfs roadMap
city (top : visited) (adjs ++ stackTail)
  where
```

Informação

Consider the following type to represent a KD-tree, which stores a set of 2D points. Non-leaf nodes of this tree contain:

- the coordinates of a point (x,y);
- the indication of whether the node is an X or Y node;
- a left subtree, where, for all of its points (x',y'), x' < x if this is a X node, or y' < y if this is a Y node;
- a right subtree, where, for all of its points (x',y'), x' >= x if this is a X node, or y' >= y if this is a Y node.

The root of any non-empty KdTree is always an X node and each level of depth in tree alternates between both kinds of nodes. Thus, the direct non-empty children of the root are Y nodes and the non-empty grandchildren are X nodes (if they exist).

A KD-tree thus behaves like a binary search tree that alternates the criterion used to divide the search space. Consider that the tree does not contain duplicate points (i.e. two or more points with the exact coordinates). The trees can, however, contain several points with the same x coordinate (but different y coordinates), or vice-versa. To help you out during testing, two trees are provided.

```
data KdTree = Empty | Node Char (Int,Int) KdTree KdTree deriving (Eq,Show)

tree1 :: KdTree
tree1 = Node 'x' (3,3) (Node 'y' (2,2) Empty Empty) (Node 'y' (4,4) Empty Empty)

tree2 :: KdTree
tree2 = Node 'x' (3,3) (Node 'y' (2,2) (Node 'x' (1,1) Empty Empty) Empty) (Node 'y' (4,4) (Node 'x' (3,2) Empty Empty)
Empty)
```

Pergunta 9

Pontuou 0,000 de 1,500

Implement insert :: (Int,Int) → KdTree → KdTree, which given a point p and a tree t, returns a new tree with the result of adding p to t. Point p should be added on the appropriate leaf node (Empty), according to the rules of the kd-tree stated before. If point p is already in the tree, then return t.

Examples:

```
ghci> insert (3,2) tree1
Node 'x' (3,3) (Node 'y' (2,2) Empty Empty) (Node 'y' (4,4) (Node 'x' (3,2) Empty Empty) Empty)
ghci> insert (1,1) (insert (3,2) tree1)
Node 'x' (3,3) (Node 'y' (2,2) (Node 'x' (1,1) Empty Empty) Empty) (Node 'y' (4,4) (Node 'x' (3,2) Empty Empty) Empty)
```

insert :: (Int,Int) → KdTree → KdTree

Pergunta 10

Pontuação 1,500

Implement putTreeStr :: KdTree → IO (), which prints the contents of the tree in several lines, in a user-friendly manner. The output format must be exactly as detailed in the examples (including regarding the ' ' and newlines).

Each tree is printed as follows:

- a line for the root's point;
- a line with the condition that all points of the left child must follow;
- the left child (if it exists);
- a line with the condition that all points of the right child must follow;
- the right child (if it exists).

At each level of depth in tree, all of the prints must be prefixed by exactly two additional spaces: " ".

Note that nothing should be printed for empty trees.

Examples:

```
ghci> putTreeStr Empty
ghci> putTreeStr tree1
(3,3)
x<3
  (2,2)
  y<2
  y>=2
x>=3
  (4,4)
  y<4
  y>=4
ghci> putTreeStr tree2
(3,3)
x<3
  (2,2)
  y<2
    (1,1)
    x<1
    x>=1
  y>=2
x>=3
  (4,4)
  y<4
    (3,2)
    x<3
    x>=3
  y>=4
```