Programming languages (TC-2006)

Homework 12

In this homework, you will practice what you know from the Haskell language. Please consider that the purpose of this homework is to allow you to practice and identify strengths and weaknesses. Then, implement these functions as requested and avoid using any built-in functions that already do what you are requested to implement.

1 `distance` (10%)

Write a function in Haskell that receives two pairs (these pairs represent points in a bidimensional space) and returns the Euclidean distance between the two points ($d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$). For example, calling your function with the arguments `(10, 20)` and `(5, 15)` must return 7.07106 because that is the Euclidean distance between those points.

NOTE: This function must be implemented by using tuples. Not using tuples will invalidate your answer.

2 `shift` (15%)

Implement a function in Haskell that shifts a list of numbers (that is treated as a circular one). Right-shifting (moving the elements to the right) will insert the last element of the list into the beggining of the list while shifting all the elements one place to the right. Conversely, left-shifting (moving the elements to the left) inserts the first element of the list into the last position, while shifting all the elements one place to the left. This time, a positive n will indicate to right-shift the list and a negative n will indicate to left-shift it. You might find the functions `init` and `last` useful for implementing this function.

3 `myFilter` (15%)

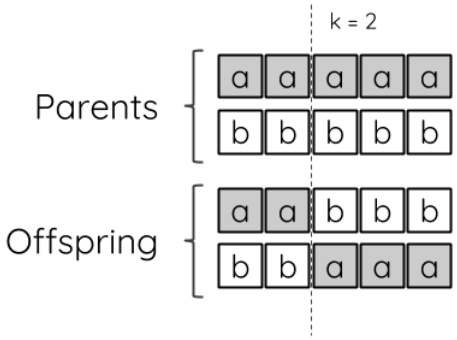
Write a **recursive** function in Haskell that mimics the behavior of the function `filter`. Your function must receive two arguments: a predicate and a list, and the result must be a list containing all the elements in the list provided as argument that satisty the predicate. For this function, **you are not allowed to use higher-order functions or any built-in function that provides the requested behavior**.

4 `crososver` (15%)

A common operation in genetic algorithms requires to take two 'parents' and combine their genetic information to produce two offspring. For this assignment you are requested to implement a crossover operator that works as follows:

- The function receives three parameters: two strings (the 'parents', both of length l) and one integer value (known as the crossover point, $k \in [1, l-1]$).
- The function returns a tuple that contains two strings (the two offspring). The first offspring will contain the first k characters from the first parent and the last $l - k$ characters from the second parent. On the other hand, the second offspring will contain the first k characters of the second parent and the last $l - k$ characters of the first one. For this function, **you are not allowed to use `take` or `drop`**.

The graphical representation of the crossover operation given the parents "aaaaa" and "bbbbb" and a crossover point of 2 is depicted as follows:



5 `xSort` (20%)

Imagine you are given a three column table from a soccer tournament, where each row contains information about a particular team, the points obtained and the goals scored. For example, the following table contains the information about six teams in this tournament:

Team	Points	Goals
Pumas	10	3
America	10	5
Chivas	11	8
Cruz Azul	11	2
Tigres	9	4
Rayados	9	6

The previous table can be represented in Haskell as:

```
[
   ("Pumas", 10, 3), ("America", 10, 5),
   ("Chivas", 11, 8), ("Cruz Azul", 11, 2),
   ("Tigres", 9, 4), ("Rayados", 9, 6)
]
```

Write a function in Haskell that receives a table represented as a list of tuples `([Char, Int, Int])` (as depicted in the previous example) and returns the table sorted by the points (in descending order). In the case of ties, the function must also consider the scored goals. For example, once sorted, the table will be:

```
[
   ("Chivas", 11, 8), ("Cruz Azul", 11, 2),
   ("America", 10, 5), ("Pumas", 10, 3),
   ("Rayados", 9, 6), ("Tigres", 9, 4)
]
```

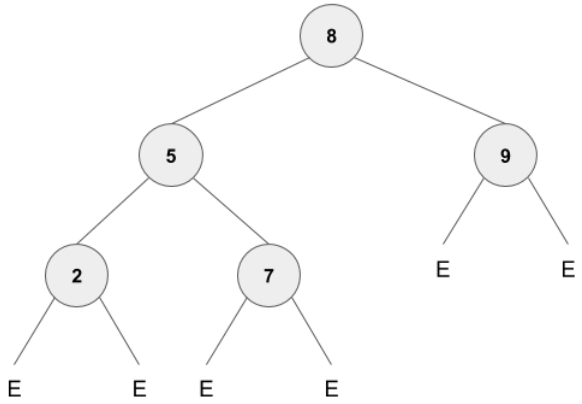
NOTE: For this assignment, use the sorting algorithm of your preference.

6 `treeSum` (25%)

Write a function in Haskell that receives a binary tree (where the nodes contain integer values) and returns the sum of all the nodes in the tree. For this exercise you are requested to use the following user-defined data type:

```
data Tree = Tree Int Tree Tree | E deriving Show
```

For example, the following picture is the graphical representation of the tree generated with the code `(Tree 8 (Tree 5 (Tree 2 E E) (Tree 7 E E)) (Tree 9 E E))`. Please note that, in this example, `E` stands for 'Empty'.



Then, the result summing all the nodes in the tree shown before is 31.

Deliverables

Prepare an HS file that contains the functions requested (in its corresponding module) and submit it to Canvas. **Please, do not submit other formats but HS**. To prepare your HS file, use the code template distributed along with this document. The template contains some test cases for each function to help you verify that your codes work as requested.

I promise to apply my knowledge, strive for its development, and not use unauthorized or illegal means to complete this activity, following the Tecnológico de Monterrey Student Code of Honor.