Assignment Three: More Fun with Haskell

Due: 11:59pm, 6 February 2024.

You may do this question either alone or with a partner (i.e., in a group of 1 or 2). Make sure you both understand and can explain your solution. If you are doing this in a group of size 2, you should do two of questions 1-3, and a group of one should do one of them.

Submit your answers in one or more text files (one file for each question 1-3 you did) to using Canvas. Make sure you name(s), student number(s) is at the top of each file. Each file should run with Haskell Platform. Your code needs comments that include the specification of the meaning of all functions. The explanations should be comments in the Haskell file. Both students need to submit to Canvas.

Question One

The game of "twenty questions" lets a user ask yes-no question to find an entity (object) of interest. If the system cannot find the entity, it asks for a question that distinguishes the entity the user thought of from the one it found. Thus it can learn from its mistakes. David started a program but didn't get it finished. It can be found at `https://www.cs.ubc.ca/~poole/cs312/2024/haskell/TwentyQs.hs`

(a) Finish the program so that is asks for the new entity and a question, and uses them to expand the tree. It should have a behaviour something like (note that this has been reformatted to fit):

```
*TwentyQs> go
Do you want to play 20 questions?
yes
Think of an entity
Is it living?
yes
Is it a person?
yes
Is it Justin Bieber?
no
What were you thinking of?
Margaret Atwood
Give a question for which the answer is yes for Margaret Atwood
        and no for Justin Bieber
Has the person written multiple award-winning novels?
Do you want to play 20 questions?
no
QNode "Is it living?"
       (QNode "Is it a person?"
           (QNode "Has the person written multiple award-winning novels?"
                (QLeaf "Margaret Atwood") (QLeaf "Justin Bieber"))
```

```
            (QLeaf "Tahlequah (J-35), a southern resident killer whale"))
        (QNode "Is it a physical object?"
            (QLeaf "Whistler") (QLeaf "CPSC 312"))
*TwentyQs>
```

(b) On some systems, the built-in function `getLine` does not work very well for interaction as it reports all of the characters typed, instead of allowing the user to delete characters that are mistakes. Write a function `fixdel` that removes deleted characters, such as:

```
*TwentyQs> str = "abc\DELd\DEL\DELefg\DELh"
*TwentyQs> fixdel str
"aefh"
```

Incorporate this into to your implementation of twenty questions. Note that this will not change what is echoed on the terminal.

Question Two

A priority queue is a multiset where, elements can be added to (pushed onto) the queue and a smallest element can be removed and returned with a pop operation. For imperative languages a heap is used to implement a priority queue (with side effects). A heap is a representation of a binary tree where the element at the root of each subtree is less than or equal to the values of its children (and so also less then or equal to its descendents).

David made a quick implementation of a priority queue implementation at `https://www.cs.ubc.ca/~poole/cs312/2024/haskell/PQ.hs` (you can ignore the module declaration at the start now). He also implemented a version of heapsort (to sort a list, add the elements to a priority queue and then remove them one at at time) which also reports the size and depth of the priority queue created.

Having balanced trees is supposed to increase performance, but the trees are not balanced. One idea that David had, was to have a tree where the left tree is either the same size or has one element more than the right tree. So when an element is added it should be added to the right tree and then the left and right trees are swapped (which will maintain the invariant). When an element is removed from a tree, it is preferable (but not always possible) to remove it from the left tree and then swap left and right trees.

(a) Implement this simple idea. It should make the depth of the tree in psort smaller. Does it improve performance? (Try it for a size of list where there is a non-trivial runtime.)

(b) Does the simple method of just swapping subtrees guarantee to make a balanced tree? Give an example where the tree is not balanced. Challenge part: implement it so that it is always balanced. Does this improve performance?

(c) What if we want to have a key-value pair where the comparison is only on the key. How can this be done without changing the code? Hint: create a new datatype that implements Ord.

(d) Why is one advantage of this queue representation beyond Haskell? Why might someone want to use this representation than a heap even in an imperative programming language?

Question Three

(a) Write a function *splitsep* that takes a Boolean separator function, a list and constructs a list of the elements between the separators.

```
*Main> :type splitsep
splitsep :: (a -> Bool) -> [a] -> [[a]]
*Main> splitsep (==',') "comma,separated,list,as,in,a,csv,file"
["comma","separated","list","as","in","a","csv","file"]
*Main> splitsep (==',') "csv,,with,missing,elts,,,"
["csv","","with","missing","elts","","",""]
*Main> splitsep (`elem` " ,.?!") "What? is this thing? ... called Love."
["What","","is","this","thing","","","","","","called","Love",""]
*Main> splitsep (==',') []
[""]
*Main> splitsep (\ x -> mod x 2 == 0) [1,3,4,5,7,9,8,8,55,45,48]
[[1,3],[5,7,9],[],[55,45],[]]
```

(b) Write a program that uses splitsep to read in a simple CSV file (comma separated values) and output a list of list of strings where the outside lists are separated by newlines (`'\n'`) and the inside lists are separated by commas in the file. That is, first split on `'\n'` then split on `','`.

For example reading the CSV file containing:

```
Day,Month,Received,Sold
12,May,200,20
10,July,,23
```

should return

```
[["Day", "Month", "Received", "Sold"], ["12", "May", "200", "20"],
["10", "July", "", "23"]]
```

To read a file, you can use

```
file <- readFile filename
```

within a `do` block, where `filename` is the name of the file, and `file` is a string that is the contents of the file. The variable `file` can then be used in other expressions (in particular involving `splitsep`).

(c) Categorize each value as to whether it is number, a string or missing (e.g., using FValue in lecture 9). `readMaybe` is useful

```
ghci> import Text.Read   (readMaybe)
ghci> (readMaybe "1234") :: Maybe Integer
Just 1234
ghci> (readMaybe "12gh34") :: Maybe Integer
Nothing
```

For more of a challenge, determine the type of every column in a CSV file. For example, if everything in a column can be interpretd as a real number, different machine learning algorithms might be used than if some are real and others are not.

Question Four

For each question, specify how long you spend on it, and what you learned. Was the question reasonable? (This question is part of the assignment, so please do it!)