Homework 2: Trees
CIS 352: Programming Languages
28 January 2016, Version 3

I looked up my family tree and found out I was the sap. — R. Dangerfield

Background

The starter file for this assignment is: http://www.cis.syr.edu/courses/cis352/coursework/hw02.hs.

Chapter 7 of LYAHGG has an extended example on general binary search trees which you should read over. This homework concerns binary and multiway trees with character labels.¹

Binary trees  For binary trees we use a type definition similar to the one in LYAHFGG:

```haskell
data BTree = Empty | Branch Char BTree BTree
  deriving (Eq,Show)
```

Empty is an empty binary tree and (Branch c tl tr) constructs an BTree node with label c, left subtree tl, and right subtree tr. Thus, (Branch c Empty Empty) is a leaf.

Example: t1 in below represents the tree in Figure 1.

```
t1 = Branch 'm'
  (Branch 't'
    (Branch 'a' Empty Empty)
    Empty)
  (Branch 'q'
    (Branch 'm' Empty Empty)
    (Branch 'w' Empty Empty))
```

To count the number of Branch-nodes in a Tree we can do something like the following.

```
bcount :: BTree -> Int
bcount Empty = 0
bcount (Branch _ tl tr) = 1+(bcount tl)+(bcount tr)
```

Multiway trees  For multiway trees we use the type definition:

```haskell
data MTree = Node Char [MTree]
  deriving (Eq,Show)
```

(Node c [t₁, t₂, ..., tₖ]) constructs an MTree node with label c and with subtrees t₁, t₂, ..., tₖ. Thus, (Node c []) is a leaf. Note that there are no empty MTrees.

Grading Criteria

- For each problem, you get 10 points for correctness.
- For a problem where you need to supply tests, you get 2 additional points for good tests.
- Omitting your name on the source code looses you 5 points.

¹ N.B. These are not search trees.
Example: t2 below represents the tree in Figure 2.

\[
t2 = \text{Node 'u'}
\begin{align*}
&\quad \text{[Node 'c' []}, \\
&\quad \text{Node 'q' []}, \\
&\quad \text{Node 'n'} \\
&\quad \quad \text{[Node 'm' []}, \\
&\quad \quad \text{Node 'g' []}, \\
&\quad \quad \text{Node 'j' []}], \\
&\quad \text{Node 'y'} \\
&\quad \quad \text{[Node 'z' []]}
\end{align*}
\]

To count the number of Node’s in an MTree we can do something like the following:

\[
mcount1 :: \text{MTree} \to \text{Int} \\
mcount1 \ (\text{Node } n \ ts) = 1 + \text{sumCounts} \ ts
\]

\[
\text{sumCounts} :: \text{[MTree]} \to \text{Int} \\
\text{sumCounts} [ ] = 0 \\
\text{sumCounts} \ (t:ts) = \text{mcount1} \ t + \text{sumCounts} \ ts
\]

or better yet:²

\[
mcount2 \ (\text{Node } n \ ts) = 1 + \text{sum} \ (\text{map} \ \text{mcount2} \ ts)
\]

or equivalently:

\[
mcount3 \ (\text{Node } n \ ts) = 1 + \text{sum} \ [\text{mcount3} \ t | t <- ts]
\]

Testing

- For \text{bmaxDepth}, \text{mmaxDepth}, \text{blevel}, \text{mlevel}, and \text{postorder} you need to come up with your own tests.
- For \text{bleaves}, run: \text{quickCheck bleaves\_prop}.
- For \text{mleaves}, run: \text{quickCheck mleaves\_prop}.
- For \text{postorder}, run: \text{quickCheck postorder\_prop}.
- For \text{reconstruct}, run \text{quickCheck recon\_prop}.
- For \text{makeTrees}, run \text{makeTreesTest}, \text{QuickCheck} is not involved.

These test functions are in the \text{hw01.hs} file.

What to hand in

Upload to Blackboard: (i) Your source code, (ii) transcripts of your test runs, and (iii) your cover sheet.

²Recall:

\[
\begin{align*}
\text{map} & :: (a -> b) -> [a] -> ([a],[a]) \\
\text{concat} & :: ([a]) -> [a] \\
\text{concatMap} & :: (a -> [b]) -> [a] -> [b] \\
\text{map} & :: (a -> b) -> [a] -> [b] \\
\text{span} & :: (a -> Bool) -> [a] -> ([a],[a]) \\
\text{splitAt} & :: \text{Int} -> [a] -> ([a],[a])
\end{align*}
\]
**Definition.** The depth of a node in a (rooted) tree is the length of the path from the root to the node. Thus the root has depth 0. By convention, empty trees have depth $-1$.

- **Problem 1 (12 points): Maximum depth of a BTree**
  Define a function
  
  $$\text{bmaxDepth} :: \text{BTree} \rightarrow \text{Int}$$
  
  that, given a BTree $t$ returns the maximum depth of any Branch-node in $t$.

- **Problem 2 (12 points): Maximum depth of a MTree**
  Define a function
  
  $$\text{mmaxDepth} :: \text{MTree} \rightarrow \text{Int}$$
  
  that, given a MTree $t$ returns the maximum depth of any Node in $t$.

- **Problem 3 (10 points): Collecting BTree leaves**
  Define a function
  
  $$\text{bleaves} :: \text{BTree} \rightarrow \text{String}$$
  
  such that $\text{bleaves} t$ returns the list of labels of the leaves of BTree $t$.

- **Problem 4 (10 points): Collecting MTree leaves**
  Define a function
  
  $$\text{mleaves} :: \text{MTree} \rightarrow \text{String}$$
  
  such that $\text{mleaves} t$ returns the list of labels of the leaves of MTree $t$.

- **Problem 5 (12 points): BTree levels**
  Define a function
  
  $$\text{blevel} :: \text{Int} \rightarrow \text{BTree} \rightarrow \text{String}$$
  
  such that $\text{blevel} k t$ returns the list of all the Branch-labels of nodes at level $k$ in BTree $t$.

- **Problem 6 (12 points): MTree levels**
  Define a function
  
  $$\text{mlevel} :: \text{Int} \rightarrow \text{MTree} \rightarrow \text{String}$$
  
  such that $\text{mlevel} k t$ returns the list of all the Node-labels at level $k$ in MTree $t$. 

**Examples.** For $t_1$ as in Figure 1:

- $\text{bmaxDepth Empty} \sim -1$
- $\text{bmaxDepth } (\text{Branch } 'x' \text{ Empty Empty}) \sim 0$
- $\text{bmaxDepth } t_1 \sim 2$

**Examples.** For $t_2$ as in Figure 2:

- $\text{mmaxDepth } (\text{Node } 'x' \text{ [ ]}) \sim 0$
- $\text{mmaxDepth } t_2 \sim 2$

**Examples.**

- $\text{bleaves Empty} \sim ""$
- $\text{bleaves } t_1 \sim "amq"$

**Recall:** String $\equiv [\text{Char}]$.

**Example.** $\text{bleaves } t_2 \sim "cqmgjz"$

**Examples.**

- $\text{blevel } 0 \ t_1 \sim ""$
- $\text{blevel } 1 \ t_1 \sim "x"$
- $\text{blevel } 2 \ t_1 \sim "tw"$
- $\text{blevel } 3 \ t_1 \sim "amq"$
- $\text{blevel } 4 \ t_1 \sim ""$

**Examples.**

- $\text{mlevel } 0 \ t_2 \sim ""$
- $\text{mlevel } 1 \ t_2 \sim "u"$
- $\text{mlevel } 2 \ t_2 \sim "cqny"$
- $\text{mlevel } 3 \ t_2 \sim "mgjz"$
- $\text{mlevel } 4 \ t_2 \sim ""$
Homework 2: Trees

Recollect. Preorder, inorder, and postorder tree traversals are discussed in: http://en.wikipedia.org/wiki/Tree_traversal. Below are functions for building preorder and inorder lists labels of a BTree.

- preorder Empty = ""
- preorder (Branch c tl tr) = [c] ++ preorder tl ++ preorder tr

- inorder Empty = ""
- inorder (Branch c tl tr) = inorder tl ++ [c] ++ inorder tr

Problem 7 (12 points): Postorder Traversals

Define a function

postorder :: BTree -> String

that, given a BTree t, produces a postorder list of labels of t.

Example. (postorder t1) \sim \text{"atmqwx"}

Definition. A BTree has the binary search-tree property when for each Branch-node (Branch c tl tr) (i) all of Char-labels in Branch-nodes of tl are < c and (ii) all of Char-labels in Branch-nodes of tr are > c.

Problem 8 (10 points): From Traversals to Trees

Define a function

reconstruct :: String -> BTree

such that, for each BTree t with the binary search-tree property, we have:

\[ t == \text{reconstruct (postorder } t) \]

Example: For t3 from Figure 4: t1 satisfies the max-heap property, (postorder t1) \sim \text{"bkewqm"}, and (reconstruct "bkewqm") \sim

Branch 'm'
  (Branch 'e'
    (Branch 'b' Empty Empty)
    (Branch 'k' Empty Empty)
  )
  (Branch 'q'
    Empty
    (Branch 'w' Empty Empty)
  )

That is, (t3 == (reconstruct "bkewqm")) \sim \text{True}.

Hint: Consider a helper function

\[ \text{step :: BTree -> Char -> BTree} \]

such that if [c_0, \ldots, c_k] is the input string and t_k is the max-heap you've built for [c_0, \ldots, c_{k-1}], then (step t_k c_k) is the max-heap for [c_0, \ldots, c_k].

Figure 4: Tree t3
Problem 9 (10 points): Building (lots of) BTrees

Define a function

\[
\text{makeTrees :: Int \to [BTree]}
\]

that, given an integer \( n \geq 0 \), returns the list of all the BTrees with \( n \) Branch-nodes with ‘x’ as the label each Branch. Your answer should have any of the BTrees repeated. For example:

```haskell
*Main> makeTrees 0
[Empty]

*Main> makeTrees 1
[Branch ‘x’ Empty Empty]

*Main> makeTrees 2
[Branch ‘x’ (Branch ‘x’ Empty Empty) Empty,
 Branch ‘x’ (Branch ‘x’ Empty Empty) Empty]

*Main> makeTrees 3
[Branch ‘x’ (Branch ‘x’ Empty (Branch ‘x’ Empty Empty)) Empty,
 Branch ‘x’ (Branch ‘x’ Empty (Branch ‘x’ Empty Empty)) Empty,
 Branch ‘x’ (Branch ‘x’ Empty (Branch ‘x’ Empty Empty)) Empty,
 Branch ‘x’ (Branch ‘x’ Empty (Branch ‘x’ Empty Empty)) Empty]
```

Run `makeTreesTest` to test your version of `makeTrees`.

Warning: These lists get really long, really fast,\(^4\) i.e., for each \( n \geq 0 \),

\[
\text{length(makeTrees } n \text{)} = \frac{(2n)!}{(n+1)!n!} \approx \frac{4^n}{\sqrt{\pi n^3}}.
\]

Don’t try printing out the value of \( \text{length(makeTrees } n \text{)} \) for \( n > 8 \) and don’t try computing \( \text{length(makeTrees } n \text{)} \) for \( n > 15 \).