A family tree is a tree that represents the family relationships of a person. In a family tree, each person is represented as a node, and the connections between people are represented as edges. The root of the tree is usually the person's own parents, and the leaves are the person's own children. This type of tree is often used in genealogy to trace ancestry and family history.
Handy library functions for this homework include:

- `break :: (a -> Bool) -> [a] -> ([a],[a])`
- `concat :: [[a]] -> [a]`
- `concatMap :: (a -> [b]) -> [a] -> [b]`
- `map :: (a -> b) -> [a] -> [b]`
- `splitAt :: Int -> [a] -> ([a],[a])`

**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 (8 points): Maximum depth of a BTree**

Define a function

```haskell
bmaxDepth :: BTree -> Int
```

that, given a `BTree t` returns the maximum depth of any `Branch`-node in `t`. *Examples:* `(bmaxDepth Empty) ~ 0`, `(bmaxDepth (Branch 'x' Empty Empty)) ~ 0`, and `(bmaxDepth t1) ~ 2`, where `t1` is as in the first column of page 1.

**Problem 2 (8 points): Maximum depth of a MTree**

Define a function

```haskell
mmaxDepth :: MTree -> Int
```

that, given a `MTree t` returns the maximum depth of any `Node` in `t`. *Examples:* `(mmaxDepth (Node 'x' [])) ~ 0` and `(mmaxDepth t2) ~ 2`, where `t2` is as in the second column of page 1.

**Problem 3 (10 points): Collecting BTree leaves**

Define a function

```haskell
bleaves :: BTree -> String
```

such that `(bleaves t)` returns the list of labels of the leaves of `BTree t`. *Examples:* `(bleaves Empty) ~ "", (bleaves t1) ~ "amw"`. (Recall that `String = [Char]`.)

**Problem 4 (10 points): Collecting MTree leaves**

Define a function

```haskell
mleaves :: MTree -> String
```

such that `(mleaves t)` returns the list of labels of the leaves of `MTree t`. *Example:* `(mleaves t2) ~ "cqmgjz"`.

**Problem 5 (10 points): BTree levels**

Define a function

```haskell
blevel :: Int -> BTree -> String
```

such that `(blevel k t)` returns the list of all the `Branch`-labels of nodes at level `k` in `BTree t`. *Examples:* `(blevel 0 t1) ~ "", (blevel 1 t1) ~ "m", (blevel 2 t1) ~ "tq", (blevel 3 t1) ~ "amw", and (blevel 4 t1) ~ ""."
Problem 6 (10 points): MTree levels

Define a function

\[ \text{mlevel} :: \text{Int} \to \text{MTree} \to \text{String} \]

such that \( \text{(mlevel k t)} \) returns the list of all the Node-labels at level \( k \) in MTree \( t \). Examples: \( \text{(mlevel 0 t2)} \sim "" \), \( \text{(mlevel 1 t2)} \sim "u" \), \( \text{(mlevel 2 t2)} \sim "cqny" \), \( \text{(mlevel 3 t2)} \sim "mgjz" \), and \( \text{(mlevel 4 t2)} \sim "" \).

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.

\[
\text{preorder Empty} = "" \\
\text{preorder (Branch c tl tr)} = [c] ++ \text{preorder tl} ++ \text{preorder tr} \\
\text{inorder Empty} = "" \\
\text{inorder (Branch c tl tr)} = \text{inorder tl} ++ [c] ++ \text{inorder tr}
\]

Note that \( \text{length(preorder t)} == \text{length(inorder t)} \).

Problem 7 (10 points): Postfix Traversals

Define a function

\[ \text{postfix} :: \text{BTree} \to \text{String} \]

that, given a BTree \( t \), produces a postorder list of labels of \( t \). Example: \( \text{(postfix t1)} \sim "atmwqm" \).

Problem 8 (10 points): From Traversals to Trees

Define a function

\[ \text{reconstruct} :: \text{String} \to \text{String} \to \text{BTree} \]

such that, for each BTree \( t \) with no repeated labels, we have:

\[ \text{t} == \text{reconstruct (preorder t)} \ (\text{inorder t}) \]

Example: Suppose \( t3 = \text{Branch 'a' (Branch 'b' (Branch 'c' Empty Empty) Empty) (Branch 'd' (Branch 'e' Empty Empty) (Branch 'f' Empty Empty))} \), i.e.:

Then \( \text{(preorder t3)} \sim "abcdef", \ (\text{inorder t3)} \sim "cbaedf" \), and \( \text{(reconstruct "abcdef" "cbaedf") \sim (preorder t3)} \sim Branch 'a' \ (\text{Branch 'b' (Branch 'c' Empty Empty) Empty}) \ (\text{Branch 'd' (Branch 'e' Empty Empty) (Branch 'f' Empty Empty)} \) which equals \( t3 \), i.e., \( \text{(t3 == (reconstruct "abcdef" "cbaedf")\sim True} \).

(Hint: Consider how one traversal tell you how to break up the other.)
Problem 9 (12 points): Building (lots of) BTrees

Define a function

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

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:

*Main> makeTrees 0
[Empty]

*Main> makeTrees 1
[Branch 'x' Empty Empty]

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

*Main> makeTrees 3
[Branch 'x' Empty (Branch 'x' Empty (Branch 'x' Empty Empty)),
Branch 'x' Empty (Branch 'x' (Branch 'x' Empty Empty) Empty),
Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' Empty Empty),
Branch 'x' (Branch 'x' Empty (Branch 'x' Empty Empty)) Empty,
Branch 'x' (Branch 'x' (Branch 'x' Empty Empty) Empty) Empty]

Run \text{makeTreesTest} to test your version of \text{makeTrees}.

Warning: For \( n \geq 0 \), \( \text{length(makeTrees n)} = \frac{(2n)!}{(n+1)!n!} \), see: \text{http://en.wikipedia.org/wiki/Catalan_number}.
So these lists get really long, really fast. Don’t try printing out the value of \( \text{makeTrees n} \) for \( n > 8 \) and don’t try computing \( \text{length(makeTrees n)} \) for \( n > 13 \).

Hints: Consider using a list comprehension. Also, for a BTree \( t \) to be in the list \( \text{makeTrees n} \), there must be a \( k \in \{0, \ldots, n-1\} \) and a BTree \( t_1 \) with \( k \)-many Branch-nodes and another BTree \( t_2 \) with \( (n-k-1) \)-many Branch-nodes such that \( t = (\text{Branch 'x' } t_1 t_2) \).

Testing

- For bmaxDepth, mmaxDepth, blevel, mlevel, and postfix you need to come up with your own tests.
- For blevels, run: \text{quickCheck blevels_prop}.
- For mlevels, run: \text{quickCheck mlevels_prop}.
- For postfix, run: \text{quickCheck postfix_prop}.
- For reconstruct, run \text{quickCheck recon_prop}.
- For makeTrees, run \text{makeTreesTest}, QuickCheck is not involved.

These test functions are in the \text{hw02.hs} file.

What to hand in (via Blackboard)

- Your source code.
- Transcripts of your test runs.
- Your cover sheet.

Grading criteria

- The homework is out of 100 points.
- Each problem is, roughly, 70% correctness and 30% testing.
- You loose 5 points if you omit your name in the source code.

What is the drawing stuff in \text{hw02.hs}?

If you are running Mac OS X and have the GraphViz application installed (see the installation section of \text{http://www.graphviz.org}), then \( \text{drawBTree t} \) will draw the BTree \( t \) for you, and similarly, \( \text{drawMTree t} \) will draw the MTree \( t \). Note that graphviz runs on most OS’s and the function \text{dump} is the only system-dependent function. So, adapting this to other OS’s may not be too hard.