Homework 2 — assigned Friday 1 October — due Monday 11 October

General instructions

Forthcoming.

2.1 Binary Trees (20pts)

We can define binary trees without any interesting content as follows:

```
data T = Leaf | Node T T
```

A path from the root to any subtree consists of a series of instructions to go left or right, which can be represented using another datatype:

```
data P = GoLeft P | GoRight P | This
```

where the path `This` denotes the whole tree. Given some tree, we would like to find all paths, i.e., the list of all paths from the root of the given tree to each of its subtrees. Write a function `allpaths :: T -> [P]` to do so.

For instance, `allpaths (Node Leaf (Node Leaf Leaf))` should evaluate to `[This,GoLeft This,GoRight This,GoRight (GoLeft This),GoRight (GoRight This)]` (but the ordering of the paths is immaterial).

2.2 General Trees (20pts)

For the type of trees with arbitrary branching,

```
data Tree a = Node a [Tree a]
```

define a suitable fold function `foldTree`, and show how it can be used to implement a map function for trees.

2.3 Graphs (50pts)

Many representations for graphs are possible, but here we use a very simple one. We represent a directed graph as a set of edges, and an edge as a pair of numbers which are the vertex labels:

```
type Graph = [(Int, Int)]
```

1. (10pts) Are all directed graphs representable in this fashion?
2. (10pts) (Consider the graph as undirected for this question.) Write a function `isTree :: Graph -> Bool` with the obvious meaning.

3. (10pts) Write a function `isDAG :: Graph -> Bool` with the obvious meaning.

4. (10pts) A rooted dag is a dag with a unique distinguished vertex (the root) from which all vertices are reachable. Write a function `isRootedDAG :: Graph -> Bool` with the obvious meaning.

5. (10pts) Assuming that the given graph is a rooted dag, write a function `calcDepth :: Graph -> [(Int, Int)]` to produce a list of pairs of the form (vertex, depth), with one pair for each vertex in the graph, where the depth is the length of the shortest path from the root to the vertex.

2.4 Numbers (50pts)

Numerals can be represented as lists of integers. For instance, decimal numerals can be expressed as lists of integers from 0 to 9. The integer 12345678901234567890 might be represented as the Haskell list
`[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0] :: [Int]`. However, the representation should allow a radix (base) other than 10 as well.

We use the following type abbreviation:

```
type Numeral = (Int, [Int])
```

where the first component of the pair is the radix and the second the list of digits.

The above example number is then represented as:

```
example = (10, [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0])
```

Write the following functions:

1. (10pts) `makeLongInt: Integer -> Int -> Numeral`, such that `makeLongInt` n r computes the list representation of the integer n in radix r. You can assume that $n \geq 0$, and that $r > 1$. For example, `makeLongInt 123 10` should evaluate to `(10, [1,2,3])`.

2. (10pts) `evaluateLongInt: Numeral -> Integer`, such that `evaluateLongInt (`r`, `l`)` converts a numeral back to a Haskell integer. You can assume that l is a valid list for radix r. For example, `evaluateLongInt (10, [1,2,3])` should evaluate to `123`.

3. (10pts) `changeRadixLongInt: Numeral -> Int -> Numeral`, such that `changeRadixLongInt` n r computes the representation of the same number as n in a new radix r. For example,
`changeRadixLongInt (10, [1,2,3]) 8` should evaluate to `(8, [1,7,3])`; on the other hand, `changeRadixLongInt (10, [1,2,3]) 16` should evaluate to `(16, [7,11])`. This computation should be carried out without the use of Haskell's built-in `Integer` arithmetic.

4. (10pts) `addLongInts: Numeral -> Numeral -> Numeral`, such that `addLongInts` *a b* computes the sum of the numbers given by the numerals *a* and *b*. If *a* and *b* use the same radix, that radix should be used for the result. If *a* and *b* use different radices, the result should use the larger one. For example, `addLongInts (10, [1,2,3]) (3, [1])` should evaluate to `(10, [1,2,4])`. This computation should be carried out without the use of Haskell's built-in `Integer` arithmetic.

5. (10pts) `mulLongInts: Numeral -> Numeral -> Numeral`, such that `mulLongInts` *a b* computes the product of the numbers given by the numerals *a* and *b*. If *a* and *b* use the same radix, that radix should be used for the result. If *a* and *b* use different radices, the result should use the larger one. For example, `mulLongInts (10, [1,2,3]) (3, [1])` should evaluate to `(10, [1,2,3])`. This computation should be carried out without the use of Haskell's built-in `Integer` arithmetic. It is not permissible to implement the multiplication of *a* and *b* as $\sum_1^a b$.

2.5 Games (40pts)

We can represent a tic-tac-toe board as a list of fields, thus:

```
data Field = B | R | G
             deriving (Eq, Ord, Show)
type Board = [Field]
```

with the convention that `B` stands for an unoccupied field, `R` stands for a field occupied by the first player ("red"), and `G` stands for a field occupied by the second player ("green"); a board is a list with 9 fields in row-major order. Thus the board

```
BRG
BRG
BRB
```

(a final board of a game that ended in red's victory) is represented as `[B,R,G,B,R,G,B,R,B]`.

It is possible to play this game perfectly, that is, to prevent the opponent from winning. Write two functions, `strategyForRed, strategyForGreen :: Board -> Int`, that implement a perfect strategy for red and green, respectively.

That is, given a board position `b`, the result of `strategyForRed b` is a number between 0 and 8, namely the index of the field into which red should move when executing the chosen strategy; the function only needs to be defined for those board positions that are actually reachable if the strategy is followed by red—but the opponent is always free to make any legal move.

2.6 Programs (20pts)

Prove that for all finite lists `xs`,

```
foldr f e xs = foldl (flip f) e (reverse xs)
```

How to turn in

Submission instructions: use the turnin facility provided by the department; see course wiki for details.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.