Reading
This assignment covers material in Chapter 14 of *Haskell: The Craft of Functional Programming* (HCFP), which is available in CST 2-197.

Logistics
This homework is officially due in class on **Wednesday, March 26**. However, it comes with an automatic extension: anything submitted by **5pm on Friday, March 28** will be accepted as being on time.

You should work singly or in pairs on this assignment. You should turn in a hard copy of your source code and a transcript demonstrating convincingly that your code is correct. You should also submit your code electronically.

Exercises (due Wednesday, March 26)

1. HCFP 14.32 (you’ll need to read about Either on the previous page).

2. HCFP 14.37 (be sure to answer the opening question, as well).

3. HCFP 14.38 (you are not required to solve it the way suggested).


5. Recall the function findPath from the March 19 lecture. Fill in the blanks in the code below to write an equivalent version of the function, using the built-in maybe function:

```haskell
findPath :: Eq a => BTree a -> a -> Maybe [Direction]
findPath Empty _ = Nothing
findPath (BNode v lc rc) val |
| v == val = Just []
| otherwise = maybe (maybe ______________________
| | | | ______________________
| | | | ______________________ )
| | | | -----------------------
| | | -----------------------
```

6. Write a Haskell function

```haskell
leastGreater :: Int -> BTree Int -> Maybe Int
```

such that *leastGreater n tree* returns the least value in the tree *tree* that is strictly larger than *n*; when no such value exists, the function should return *Nothing*.

For example, using the tree *tree2* from the March 19 lecture, your function should have the following behavior:

```
Main> leastGreater 2 tree2
Just 5

Main> leastGreater 27 tree2
Nothing
```

Note: There are many ways to write ugly code for this question, and a few ways to write pretty legible code. The latter will receive more points than the former.