CS 403, Assignment 2

Due on 12 October at 11:59 pm

The first part of this assignment considers the integer representations discussed in the first assignment.

1. Make both the long integer and sparse integer representations developed in Assignment 1 proper data types. Then make these types instances of the type class `Integral`.

   Note that in order to be able to perform this instantiation you also need to make the types instances of other classes. It is your responsibility to discover and implement all the necessary instances.

The second part of the assignment is about binary search trees. An implementation of such trees is available on the course's Web site. You are now asked to modify this implementation as follows:

2. Define a type class `Ordered` that defines a binary search tree-like interface. That is, the interface should include the functions `insert` (for inserting a new value), `member` (for finding out whether a given value exists), `findMin` and `findMax` (which return the minimum and the maximum value, respectively), `delete` (for removing a value) and `traverse` (for traversing the stored values thus returning a sorted list). Provide partial implementations of these functions if possible.

3. Make then the type `BST` an instance of `Ordered`.

Note in passing that the binary search trees are not the only potential instance of `Ordered`. B-trees are the first additional example that comes to mind.

Implementation note

If you have a type with a parameter (such the type `BST a` for trees holding values of type `a`) it is often the case that the "big" type (`BST a`) can be an instance of some class (say, `Eq`) only if the "inner" type (`a`) is an instance of a certain type class (say, `Ord`). This kind of constraints must be specified at instantiation time. In the example mentioned above the instance `BST a` of `Eq` will thus go as follows:

```
instance Ord a => Eq (BST a) where
    ...
```

Note that I used `BST`, `Eq`, and `Ord` merely as examples, I am not claiming that the particular example above makes actual sense.

Submission guidelines

Submit a single plain text file that can be loaded in the Haskell *interpreter*. It would be nice if you can submit a literate script (`.lhs` extension), but this is not required. Your script will only be loaded in the interpreter and will not go anywhere near the compiler, so *do not* provide anything having to deal with the Haskell compiler such as a `main` function or I/O operations.

Provide in your script comments that demonstrate and test your functions. Such comments should include copy-and-paste content from the terminal where you ran and tested your program. Note that any part of your work that has not been thus demonstrated will not be marked.

Finally, include at the top of your script a comment with the names and emails of all the collaborators.

Once the script is assembled as explained above submit it by email.

Recall that assignments can be solved in groups (of maximum three students), and a single solution per group should be submitted. Also recall that a penalty of 10% per day will be applied to late submissions.