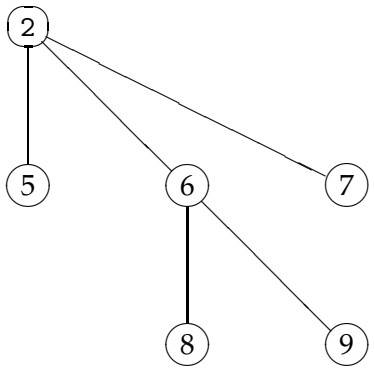
CS 403, Assignment 2

Due on 23 October at 11:59 pm

1. For any type `a`, a tree over `a` consists of a root value of type `a` and a (finite, possibly empty) sequence of sub-trees over `a`. For example, the following is a pictorial representation of a tree over `Integer`; it has root value 2 and three sub-trees, the first and third of which have no sub-trees, and the second of which has two sub-trees:



   Define a HASKELL type `Tree` such that, for any type `a`, `Tree a` is suitable for representing trees over `a`.

2. For any type `a`, a binary tree over `a` is either empty or consists of a value of type `a` and exactly two sub-binary trees over a (either or both of which might be empty). Define a HASKELL type `BinTree` such that, for any type `a`, `BinTree a` is suitable for representing binary trees over `a`.

3. Given a tree `t`, a *trace* in `t` is the sequence of nodes on some paths in the tree that is either empty or starts with the root node of `t` and ends with some (not necessarily leaf) node; let `traces t` be the set of all the traces in `t`. For example exactly all the traces of the binary tree shown above (expressed as HASKELL lists) are:

$$[], [2], [2,5], [2,6], [2,7], [2,6,8], [2,6,9]$$

   You are asked to obtain the traces of a tree (binary or otherwise) using a single function `traces` for both types. For this purpose implement a type class `Traceable` that expresses the property of some HASKELL type to have traces, and then make both trees and binary trees instances of `Traceable`.

4. Make sure that your trees (both general and binary) are instances of `Show`, `Eq`, and `Ord`. The `Ord` and `Eq` instantiation should implement a custom ordering, as follows: For some trees `t`

and `u` we have

$$\texttt{t} \leq \texttt{u} \quad \text{iff} \quad \texttt{traces t} \subseteq \texttt{traces u}$$
$$\texttt{t} == \texttt{u} \quad \text{iff} \quad \texttt{traces t} \subseteq \texttt{traces u} \text{ and } \texttt{traces u} \subseteq \texttt{traces t}$$

Pay attention to the fact that you are thus defining a partial order over trees (that is, two trees may not be comparable with each other).

Implementation note

If you have a type with a parameter (such the type `Tree a` for trees holding values of type `a`) it is often the case that the "big" type (`Tree 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, `Eq` as well). This kind of constraints must be specified at instantiation time. In the example mentioned above the instance `Tree a` of `Eq` will thus go as follows:

```
instance Eq a => Eq (Tree a) where
    ...
```

The fact that the type class `Eq` appears on both sides of `=>` in this particular example is an accident. The type class(es) specified in the constraint can be different from the type class being instantiated.

Submission guidelines

Provide your scripts and the plain English answers to questions as appropriate. Also provide sessions listings that test your functions (*any untested part of your submission will be disregarded*).

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. Provide your submission by email.

Recall that assignments can be solved in groups[1] (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.

[1] It is often easier to learn new concepts by working together in a group—you may learn more if you work with smart people. This being said, the purpose of group assignments is that all people in a group should be involved in creating the solution. You can of course choose to sign an assignment you haven't done, but then you risk not learning the material, and this will come back to haunt you during the mid-term examination.