CIS 623 Exercises on trees

Trees

Suppose we declare a type `BInTree` for binary trees using the following data declarations:

```
data BinTree a = Empty
               | Node (BinTree a) a (BinTree a)
                 deriving (Eq, Show)
```

Write the following functions in Haskell:

1. `noOfNodes`:

   This function will take a binary tree `T` as input, return the no. of nodes of the `T`.

2. `height`:

   This function will take a binary tree `T` as input, return the height of the tree `T`.

3. `preorder`:

   This function will take a binary tree `T` as input, return an pre-ordering listing of items (of type `a`) stored in the nodes of `T`.

4. `balanced`:

   This function will take a binary tree `T` as input, return `True` if `T` is height balanced. Otherwise, it will return `False`. Note that a binary tree `T` is said to be height balanced if, at each node `n` in `T`, the height of the left subtree (rooted at `n`) and the height of the right subtree (rooted at `n`) is either 0, 1, or -1.

Solution outline

For background, see Ch.6 of the text: Open data structures available at `http://opendatastructures.org/ods-java.pdf`.

```
data BinTree a = Empty
               | Node (BinTree a) a (BinTree a)
                 deriving (Eq , Show)

noOfNodes         :: BinTree a -> Int
noOfNodes Empty                  = 0
noOfNodes (Node lft n rght) =
    1 + noOfNodes lft + noOfNodes rght

height        :: BinTree a -> Int
height Empty = -1
height  (Node lft n rght) =
    1 + maximum [height lft , height rght]

preorder        :: BinTree a -> [a]
preorder Empty = []
preorder (Node lft n rght)
    = n : ((preorder lft) ++ (preorder rght))

balanced :: BinTree a -> Bool
balanced Empty              = True
balanced (Node lft n rght) = (balanced lft)
                             &&
                             (balanced rght)
                             &&
                             abs (height lft - height rght) <= 1
```