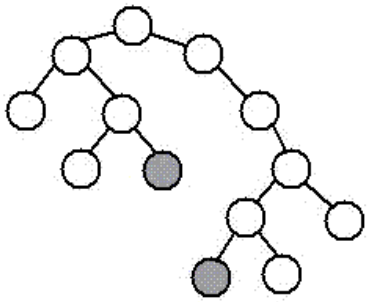
Problem Set 3

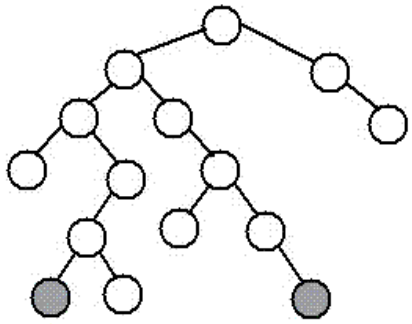
1. Consider a tree defined by the following datatype:

```
data Tree a = Node (Tree a) a (Tree a) | Null
```

The diameter of a tree is the length of the longest path from one leaf to another, counted by the number of nodes in the path. Write a function `dia t` that will return the diameter of the tree `t`. The figure below illustrates the notion of diameter. (15 Marks)

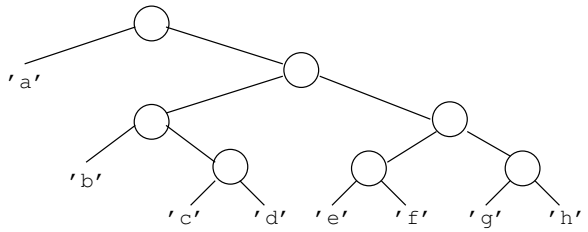


diameter, 9 nodes, through root



diameter, 9 nodes, NOT through root

2. Consider the binary tree shown in the figure. Assume that it is bound to the variable `ht`.



Such a tree represents a binary encoding of the characters at the leaves. As examples: `'a'` is reached from the root by just traversing a left branch. Representing a traversal of a left branch as `0`, the code of `'a'` is `[0]`. Similarly the code for `'b'` is `[1,0,0]`, because the path from the root to `'b'` traverses a right branch followed by two left branches.

(a) Write a function called
`decode :: [Int] -> Btree Char -> [Char]`,
which takes a code consisting of a list of `0`s and`1`s and a tree such as above and returns the list of characters corresponding to the code. The function should return an appropriate error message if the code does not correspond to any valid string of characters. For example:
`decode [1,1,1,1,1,1,0,0,0,1,0,1,1] ht = ['h', 'e', 'a', 'd']`

(b) Write a function called `encode:: [Char] -> Btree Char -> [Int]`, which does the opposite.
`encode ['f', 'a', 'd'] ht = [1,1,0,1,0,1,0,1,1]`

3. A radix tree is defined as:

```
data RadixTree = Leaf Color | Node Colour RadixTree RadixTree

data Color = Black | White
```

A radix tree represents a set of binary strings. Given a binary string, a radix tree is traversed by starting from the root and going down the tree in the following manner. At any node, if the binary string starts with a zero, then we go down the left subtree, else we go down the right subtree. The first bit then discarded and the same step is repeated. The binary string is contained in the tree, if and only if the traversal ends in a white node or leaf.

Write a function `insert` which will take a radix tree and insert a binary string in the tree. Assume that binary strings are represented as lists of 0's and 1's.

```
insert :: BinaryString -> RadixTree -> RadixTree
```

4. One way of building a tree is to take a list `[x0, x1, x2,...]` and build a tree with `x0` as the label of the root, `x1` and `x2` as the labels of the two immediate subtrees, `x3`, `x4`, `x5` and `x6` as the labels of the four granddaughter trees, and so on. Assume that the tree that has to be built is defined as

```
data Htree a = Null | Fork a (Htree a) (Htree a)
```

   (a) To do this, first define a function called levels which will divide a list into sublists of lengths which are successive powers of two.

```
levels = [a] -> [[a]]
```

   (b) Next, the function mktree takes the list of lists returned by levels and builds a list consisting of a single tree.

```
mktree :: [[a]] -> [Htree a]
mktree = foldr addLayer [Null]
```

   (c) Define `addLayer`. To keep things simple, assume that the number of elementsis $2^n - 1$ for some n.

5. Consider the syntax for a function call in a language like C. An example is: `f(g(x,l), h(l), 5)`. A natural representation for this expression is in terms of a general tree, defined as:

```
data Gtree a = Gnode a [Gtree a]

Gnode 'f' [Gnode 'g' [Gnode 'x' [], Gnode 'l' []],
           Gnode 'h' [Gnode 'l' []],
           Gnode 5 []]
```

Now consider the same expression written in Haskell as `f (g x l) (h l) 5)`. This can be represented as a kind of Binary tree, defined as:

```
data Btree a = Bnode ltree rtree | Leaf a
```

Using this, the expression is written as:

```
Bnode (Bnode (Bnode (Leaf 'f')
                    (Bnode (Bnode (Leaf 'g') (Leaf 'x'))
                           (Leaf 'l')))
              (Bnode (Leaf 'h') (Leaf 'l')))
      (Leaf 5)
```

Write functions `g-to-b` to convert a C-style expression to an equivalent Haskell-style expression. Similarly write a function `b-to-g` to convert a Haskell-style expression to a C-style expression.