99 questions/54A to 60

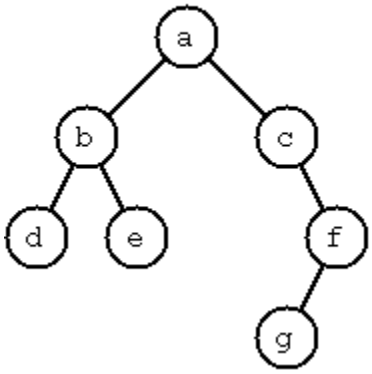
From HaskellWiki

< 99 questions

This is part of Ninety-Nine Haskell Problems, based on Ninety-Nine Prolog Problems (https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/) .

1 Binary trees

A binary tree is either empty or it is composed of a root element and two successors, which are binary trees themselves.



In Haskell, we can characterize binary trees with a datatype definition:

```
data Tree a = Empty | Branch a (Tree a) (Tree a)
            deriving (Show, Eq)
```

This says that a `Tree` of type `a` consists of either an `Empty` node, or a `Branch` containing one value of type `a` with exactly two subtrees of type `a`.

Given this definition, the tree in the diagram above would be represented as:

```
tree1 = Branch 'a' (Branch 'b' (Branch 'd' Empty Empty)
                               (Branch 'e' Empty Empty))
                   (Branch 'c' Empty
                               (Branch 'f' (Branch 'g' Empty Empty)
                                           Empty))
```

Since a "leaf" node is a branch with two empty subtrees, it can be useful to define a shorthand function:

```
leaf x = Branch x Empty Empty
```

Then the tree diagram above could be expressed more simply as:

```
tree1' = Branch 'a' (Branch 'b' (leaf 'd')
                                (leaf 'e'))
                    (Branch 'c' Empty
                                (Branch 'f' (leaf 'g')
                                            Empty)))
```

Other examples of binary trees:

```
-- A binary tree consisting of a root node only
tree2 = Branch 'a' Empty Empty

-- An empty binary tree
tree3 = Empty

-- A tree of integers
tree4 = Branch 1 (Branch 2 Empty (Branch 4 Empty Empty))
                 (Branch 2 Empty Empty)
```

2 Problem 54A

(*) Check whether a given term represents a binary tree

In Prolog or Lisp, one writes a predicate to do this.

Example in Lisp:

```
* (istree (a (b nil nil) nil))
T
* (istree (a (b nil nil)))
NIL
```

Non-solution:

Haskell's type system ensures that all terms of type
`Tree a`
are binary trees: it is just not possible to construct an invalid tree with this type. Hence, it is redundant to introduce a predicate to check this property: it would always return
`True`
.

3 Problem 55

(**) Construct completely balanced binary trees

In a completely balanced binary tree, the following property holds for every node: The number of nodes in its left subtree and the number of nodes in its right subtree are almost equal, which means their difference is not greater than one.

Write a function cbal-tree to construct completely balanced binary trees for a given number of nodes. The predicate should generate all solutions via backtracking. Put the letter 'x' as information into all nodes of the tree.

Example:

```
* cbal-tree(4,T).
T = t(x, t(x, nil, nil), t(x, nil, t(x, nil, nil))) ;
T = t(x, t(x, nil, nil), t(x, t(x, nil, nil), nil)) ;
etc......No
```

Example in Haskell, whitespace and "comment diagrams" added for clarity and exposition:

```
*Main> cbalTree 4
[
-- permutation 1
--     x
--    / \
--   x   x
--        \
--         x
Branch 'x' (Branch 'x' Empty Empty)
           (Branch 'x' Empty
                       (Branch 'x' Empty Empty)),

-- permutation 2
--     x
--    / \
--   x   x
--      /
--     x
Branch 'x' (Branch 'x' Empty Empty)
           (Branch 'x' (Branch 'x' Empty Empty)
                       Empty),

-- permutation 3
--     x
--    / \
--   x   x
--    \
--     x
Branch 'x' (Branch 'x' Empty
                       (Branch 'x' Empty Empty))
           (Branch 'x' Empty Empty),

-- permutation 4
--     x
--    / \
--   x   x
--  /
-- x
```

```
Branch 'x' (Branch 'x' (Branch 'x' Empty Empty)
                       Empty)
           (Branch 'x' Empty Empty)
]
```

Solutions

4 Problem 56

(**) Symmetric binary trees

Let us call a binary tree symmetric if you can draw a vertical line through the root node and then the right subtree is the mirror image of the left subtree. Write a predicate symmetric/1 to check whether a given binary tree is symmetric. Hint: Write a predicate mirror/2 first to check whether one tree is the mirror image of another. We are only interested in the structure, not in the contents of the nodes.

Example in Haskell:

```
*Main> symmetric (Branch 'x' (Branch 'x' Empty Empty) Empty)
False
*Main> symmetric (Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' Empty Empty))
True
```

Solutions

5 Problem 57

(**) Binary search trees (dictionaries)

Use the predicate add/3, developed in chapter 4 of the course, to write a predicate to construct a binary search tree from a list of integer numbers.

Example:

```
* construct([3,2,5,7,1],T).
T = t(3, t(2, t(1, nil, nil), nil), t(5, nil, t(7, nil, nil)))
```

Then use this predicate to test the solution of the problem P56.

Example:

```
* test-symmetric([5,3,18,1,4,12,21]).
```

```
Yes
* test-symmetric([3,2,5,7,4]).
No
```

Example in Haskell:

```
*Main> construct [3, 2, 5, 7, 1]
Branch 3 (Branch 2 (Branch 1 Empty Empty) Empty) (Branch 5 Empty (Branch 7 Empty Empty))
*Main> symmetric . construct $ [5, 3, 18, 1, 4, 12, 21]
True
*Main> symmetric . construct $ [3, 2, 5, 7, 1]
True
```

Solutions

6 Problem 58

(**) Generate-and-test paradigm

Apply the generate-and-test paradigm to construct all symmetric, completely balanced binary trees with a given number of nodes.

Example:

```
* sym-cbal-trees(5,Ts).
Ts = [t(x, t(x, nil, t(x, nil, nil)), t(x, t(x, nil, nil), nil)), t(x, t(x, t(x, nil, nil), nil), t(x, n
```

Example in Haskell:

```
*Main> symCbalTrees 5
[Branch 'x' (Branch 'x' Empty (Branch 'x' Empty Empty)) (Branch 'x' (Branch 'x' Empty Empty)
```

Solutions

7 Problem 59

(**) Construct height-balanced binary trees

In a height-balanced binary tree, the following property holds for every node: The height of its left subtree and the height of its right subtree are almost equal, which means their difference is not greater than one.

Construct a list of all height-balanced binary trees with the given element and the given maximum height.

Example:

```
?- hbal_tree(3,T).
T = t(x, t(x, t(x, nil, nil), t(x, nil, nil)), t(x, t(x, nil, nil), t(x, nil, nil))) ;
T = t(x, t(x, t(x, nil, nil), t(x, nil, nil)), t(x, t(x, nil, nil), nil)) ;
etc......No
```

Example in Haskell:

```
*Main> take 4 $ hbalTree 'x' 3
[Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' Empty (Branch 'x' Empty Empty)),
 Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' (Branch 'x' Empty Empty) Empty),
 Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' Empty E
 Branch 'x' (Branch 'x' Empty (Branch 'x' Empty Empty)) (Branch 'x' Empty Empty)]
```

Solutions

8 Problem 60

(**) Construct height-balanced binary trees with a given number of nodes

Consider a height-balanced binary tree of height H. What is the maximum number of nodes it can contain?

Clearly, MaxN = 2**H - 1. However, what is the minimum number MinN? This question is more difficult. Try to find a recursive statement and turn it into a function
`minNodes`
that returns the minimum number of nodes in a height-balanced binary tree of height H. On the other hand, we might ask: what is the maximum height H a height-balanced binary tree with N nodes can have? Write a function
`maxHeight`
that computes this.

Now, we can attack the main problem: construct all the height-balanced binary trees with a given number of nodes. Find out how many height-balanced trees exist for N = 15.

Example in Prolog:

```
?- count_hbal_trees(15,C).
C = 1553
```

Example in Haskell:

```
*Main> length $ hbalTreeNodes 'x' 15
1553
*Main> map (hbalTreeNodes 'x') [0..3]
[[Empty],
```

```
[Branch 'x' Empty Empty],
[Branch 'x' Empty (Branch 'x' Empty Empty),Branch 'x' (Branch 'x' Empty Empty) Empty],
[Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' Empty Empty)]]
```

Solutions

Category:

- Tutorials
