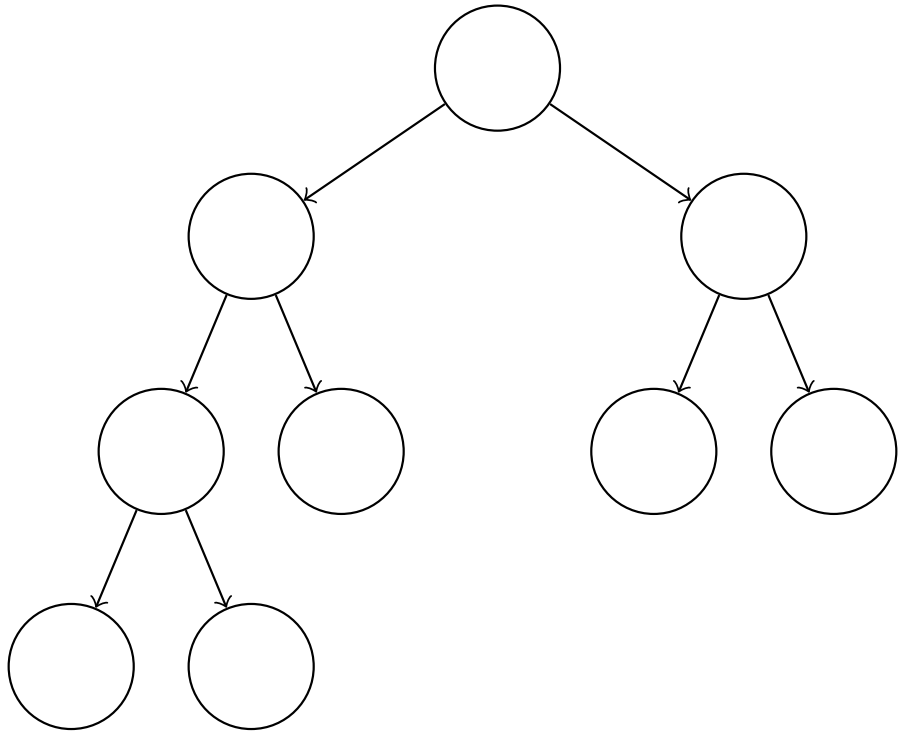
# COMP105 Lecture 22

# Trees

# Trees

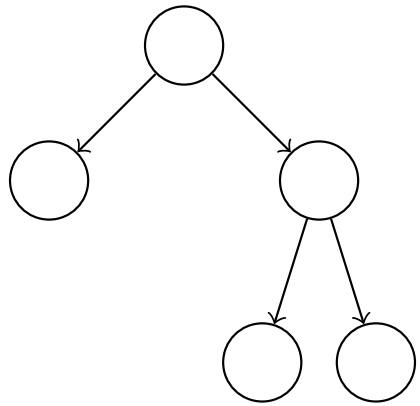


A **tree** is composed of

- Leaf nodes
- Branch nodes

# A tree type in Haskell

```
data Tree = Leaf | Branch Tree Tree deriving (Show)
```



```
Branch Leaf (Branch Leaf Leaf)
```

# Recursion on trees

We can write **recursive** functions that process trees

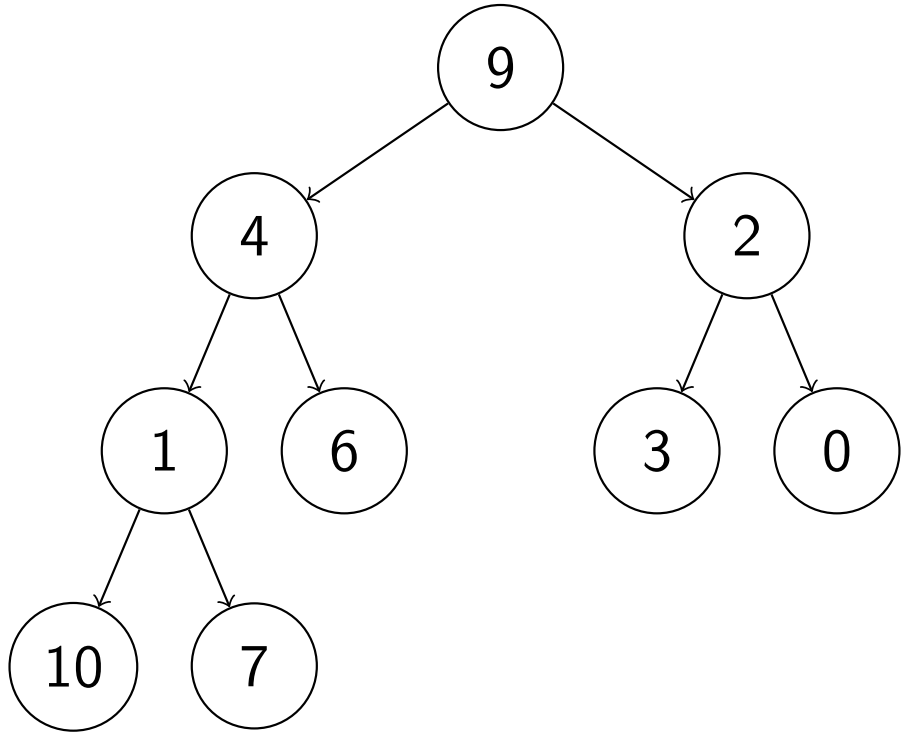
- Usually the recursive case will process both branches

```
size :: Tree -> Int

size (Leaf) = 1
size (Branch x y) = 1 + size x + size y
```

```
ghci> size (Branch Leaf (Branch Leaf Leaf))
5
```

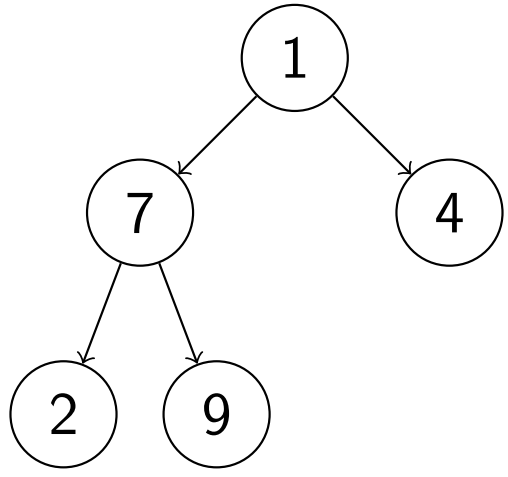
# Trees with data




Nodes in a tree often hold **data**

# Trees with data

```
data DTree a = DLeaf a
             | DBranch a (DTree a) (DTree a)
                                   deriving (Show)
```




```
DBranch 1 (DBranch 7 (DLeaf 2) (DLeaf 9)) (DLeaf 4)
```

# Recursion on trees with data

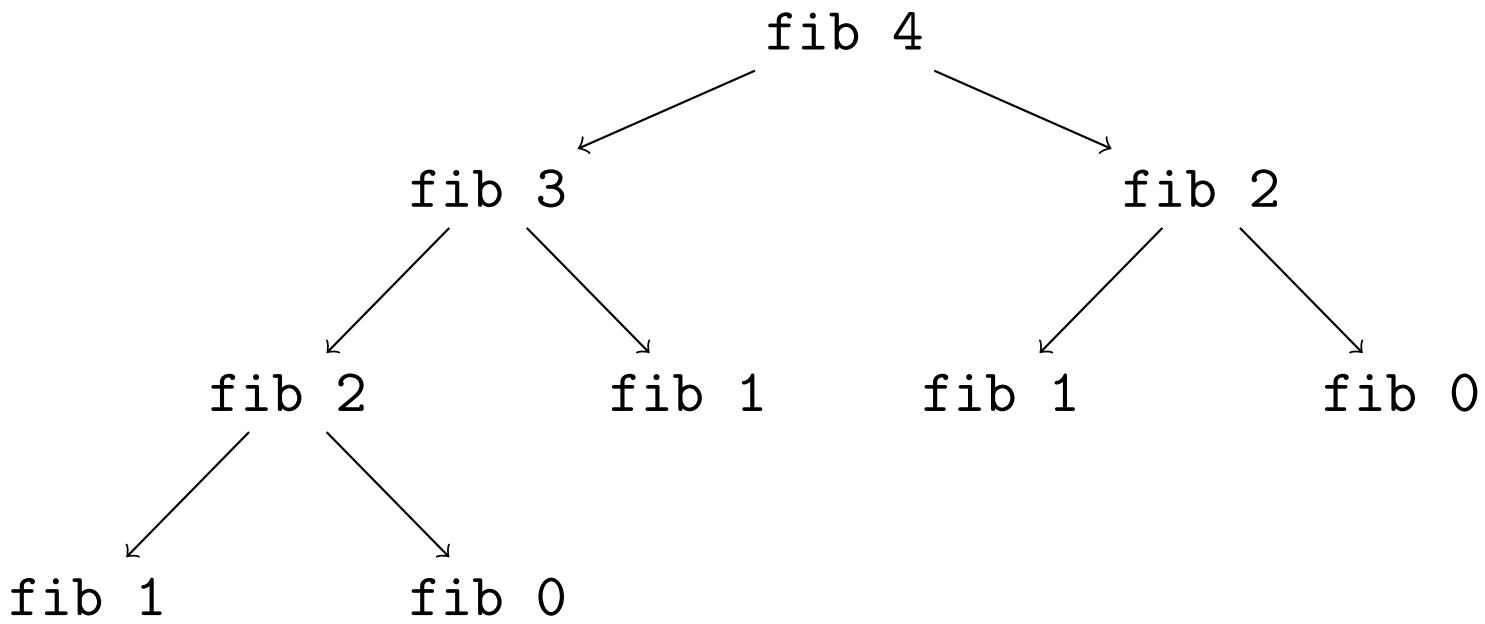
```
tree_sum :: Num a => DTree a -> a

tree_sum (DLeaf x) = x
tree_sum (DBranch x l r) = x + tree_sum l + tree_sum r
```

```
ghci> tree_sum (DBranch 11 (DLeaf 2) (DLeaf 9))
22
```

# Example: Fibonacci numbers

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```



# Example: Fibonacci numbers

How many recursive calls does the code make?

- Let's build the call tree

```
fib_tree :: Int -> Tree

fib_tree 0 = Leaf
fib_tree 1 = Leaf
fib_tree n = Branch (fib_tree (n-1)) (fib_tree (n-2))
```

```
ghci> fib_tree 4
Branch (Branch (Branch Leaf Leaf) Leaf)
                                  (Branch Leaf Leaf)
```

# Example: Fibonacci numbers

```
fib_calls n = size (fib_tree n)
```

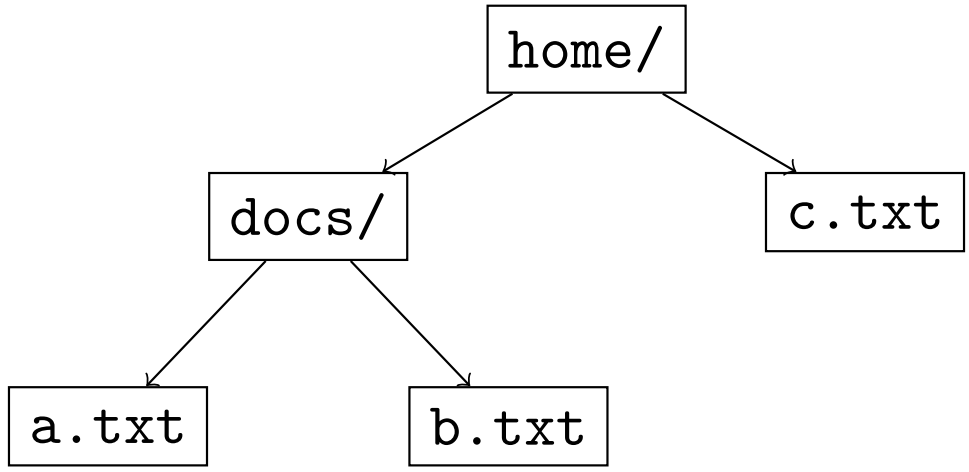
```
ghci> fib_calls 10
177

ghci> fib_calls 20
21891

ghci> fib_calls 30
2692537
```

# Example: Finding a file

Suppose that we have a directory structure




Write a function that, given a filename finds the path to that file

# Example: Finding a file

We can formulate the files as a data tree

```
let fs =
  DBranch "home/"
     (DBranch "docs/" (DLeaf "a.txt") (DLeaf "b.txt"))
     (DLeaf "c.txt")
```

# Example: Finding a file

Note that the file might not exist

- So we will use the maybe type

```
ghci> find_file "a.txt" fs
Just "home/docs/a.txt"
```

```
ghci> find_file "d.txt" fs
Nothing
```

# Example: Finding a file

```
find_file file (DLeaf x)
    | x == file = Just file
    | otherwise = Nothing


find_file file (DBranch x l r) =
    let
        left = find_file file l
        right = find_file file r
    in
        case (left, right) of
            (Just y, _) -> Just (x ++ y)
            (_, Just y) -> Just (x ++ y)
            (_, _)      -> Nothing
```

# Exercises

1. Write a function `leafSum :: DTree Int -> Int` that sums all of the leafs of the tree (and ignores the numbers in the branch nodes)

2. Write a function `treeElem :: Eq a => a -> DTree a -> Bool` that takes an element x and a tree and returns `True` if x is in the tree, and `False` otherwise