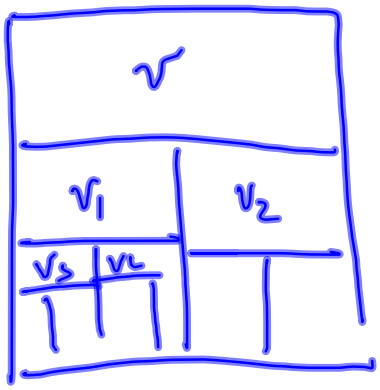
Lists → Trees

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



Recall: list $[v_1, v_2, v_3]$

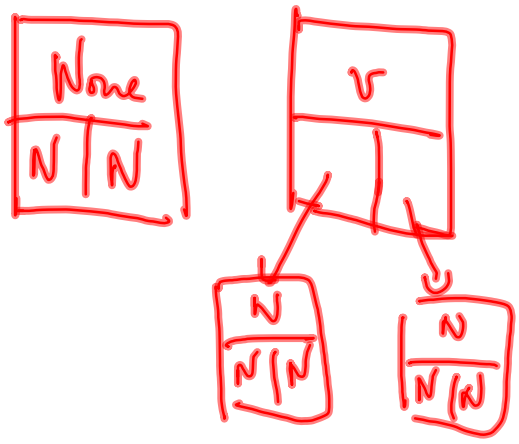
capture this construct as a class

```
class TNode:
    def __init__(self, initval=None):
        self.value = initval
        if initval != None:
            self.left = TNode()
            self.right = TNode()
        else:
            self.left = None
            self.right = None
```

t = TNode()

or

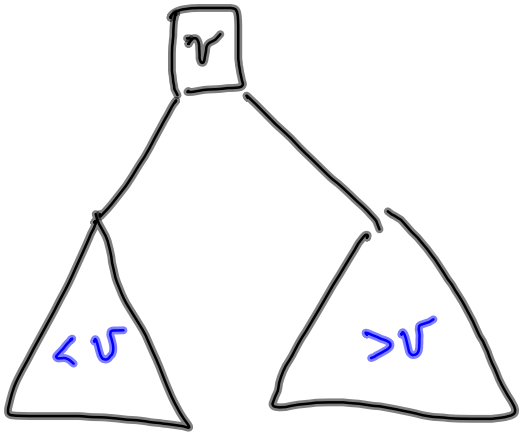
t = TNode(v)



insert Nil v = Node Nil v Nil

Search trees (values are ordered using $\leq$)

Each value is stored at most once



3 operations on search trees:

find v in t return True/False

insert v in t do nothing if v already present in t

delete v from t do nothing if v not present in t

```
find Nil v = False
find (Node tl x tr) v
  | v == x  = True
  | v < x   = find tl v
  | otherwise = find tr v
```

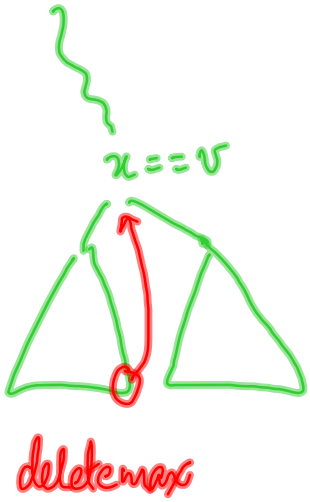
```
class TNode:
    :
    def find(self, v):
        if self.value == None:
            return (False)
        if v == self.value:
            return (True)
        if v < self.value:
            return (self.left.find(v))
        else:
            return (self.right.find(v))
```

```
insert Nil v = Node Nil v Nil
insert (Node tl x tr) v
  | v < x = Node (insert tl v) x tr
  | v > x = Node tl x (insert tr v)
  | otherwise = Node tl x tr
```

```
def insert(self, v):
    if self.value == None:
        self.value = v
        self.left = TNode()
        self.right = TNode()
    if v < self.value:
        self.left.insert(v)
    if v > self.value:
        self.right.insert(v)
```

```
delete Nil v = Nil
delete (Node tl x tr) v
  | v < x  = Node (delete tl v) x tr
  | v > x  = Node tl x (delete tr v)
  | tl == Nil = tr
  | tr == Nil = tl
  | otherwise = Node tnew d tr
      where (tnew, d) = deletemax tl
```

x == v



```
def delete(self, v):
    if self.value == None:
        return
    if v < self.value:
        self.left.delete(v)
        return
    if v > self.value:
        self.right.delete(v)
        return
    if self.left.value == None:
        self.value = self.right.value
        self.left = self.right.left
        self.right = self.right.right
        return
    if self.right.value == None:
        self.value = self.left.value
        self.right = self.left.right
        self.left = self.left.left
        return
    d = self.left.deletemax()
    self.value = d
    return
```

Symmetric

Cannot swap these two!

deletemax (Node tl x Nil) = (tl, x)

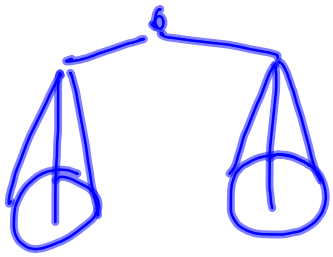
deletemax (Node tl x tr) = (Node tl x tnew, d)

where (tnew, d) = deletemax tr

```
def deletemax(self):
    if self.right.value == None:
        x = self.value
        self.value = self.left.value
        self.right = self.left.right
        self.left = self.left.left
        return(x)
    else:
        d = self.right.deletemax()
        return(d)
```

Balancing the tree

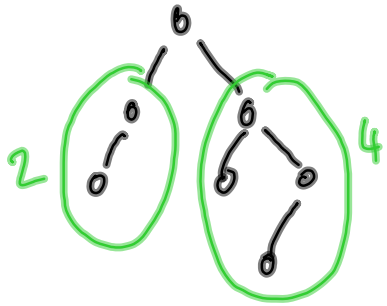
What is balance?



size balance

$|size(left\ tree) - size(right\ tree)| \leq 1$

height balance $|height(left\ tree) - height(right\ tree)| \leq 1$



height balanced, not size balanced

smallest(h) = smallest height balanced tree of height h
nodes in

smallest(0) = 0

smallest(1) = 1

smallest(2) = 2

smallest(3) = 4

:

smallest(h+1) = 1 + smallest(h) + smallest(h-1)

smallest(h) $\geq$ fib(h) fib grows exponentially