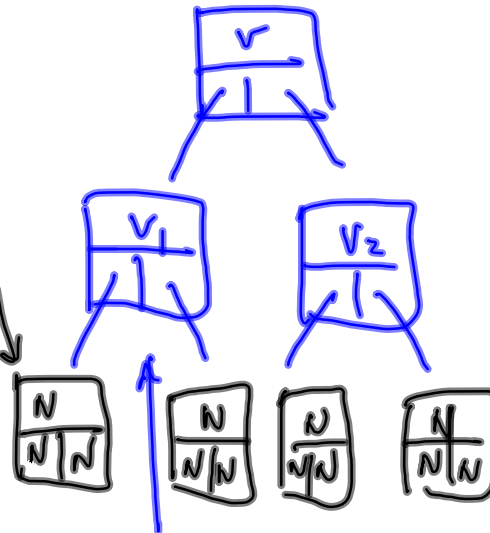
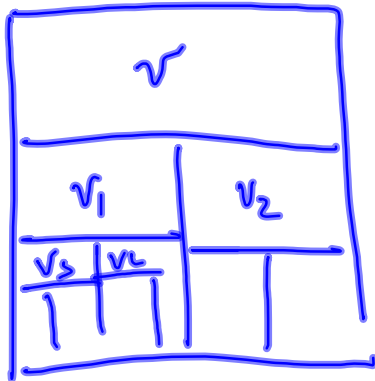


Lists \rightarrow Trees

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



Recall: list [v1, v2, v3]



capture this unit as a class

```
class TNode:
```

```
    def __init__(self, initval=None):
```

```
        self.value = initval
```

```
        if initval != None:
```

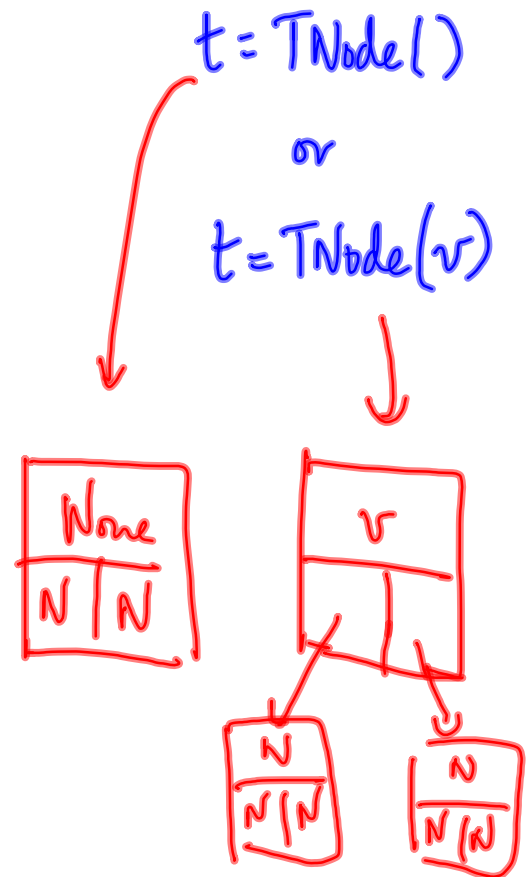
```
            self.left = TNode()
```

```
            self.right = TNode()
```

```
        else:
```

```
            self.left = None
```

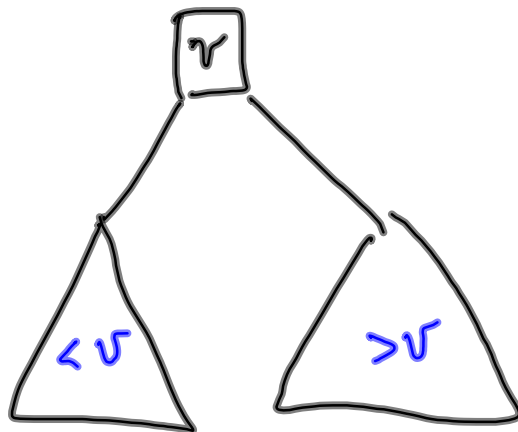
```
            self.right = None
```



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)

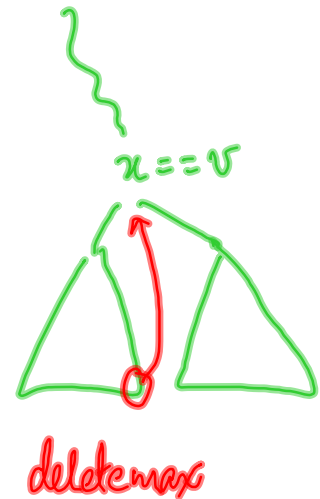
x == v

| tl == Nil = tr

| tr == Nil = tl

| otherwise = Node tnew d tr

where (tnew, d) = deletemax tl



```
def delete(self, v):
```

```
    if self.value == None:
```

```
        return
```

```
    if v < self.value:
```

```
        self.left.delete(v)
```

```
    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!

$\text{deletemax}(\text{Node } tl \ x \ \text{Nil}) = (tl, x)$

$\text{deletemax}(\text{Node } tl \ x \ tr) = (\text{Node } tl \ x \ tnew, d)$

where $(tnew, d) = \text{deletemax } tr$

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

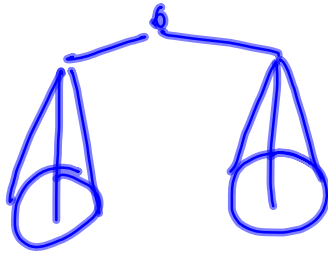
 return(x)

else:

 d = self.right.deletemax()

 return(d)

Balancing the tree
 What is balance?

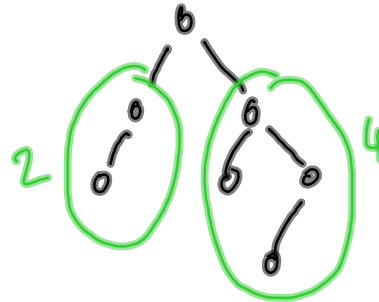


size balance

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

height balance

$$|\text{height}(\text{left tree}) - \text{height}(\text{right tree})| \leq 1$$



height balanced,
 not size balanced

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

$$\text{smallest}(0) = 0$$

$$\text{smallest}(1) = 1$$

$$\text{smallest}(2) = 2$$

$$\text{smallest}(3) = 4$$

$$\vdots$$
$$\text{smallest}(h+1) = 1 + \text{smallest}(h) + \text{smallest}(h-1)$$

$$\text{smallest}(h) \geq \text{fib}(h) \quad \text{fib grows exponentially}$$