1 Immutable Invariants

empty-set

O(n) add (s :: Set, elt :: String) → Set

O(n) member (s :: Set, elt :: String) → Boolean

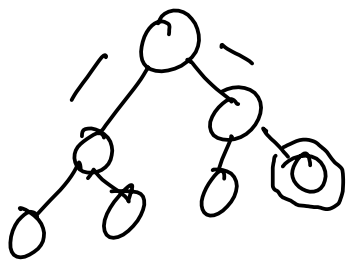
type Set = List<String>

add = link

Trees

```
data BTree:
  | leaf
  | node(v :: Str,
         l :: BTree,
         r :: BTree)
end
```

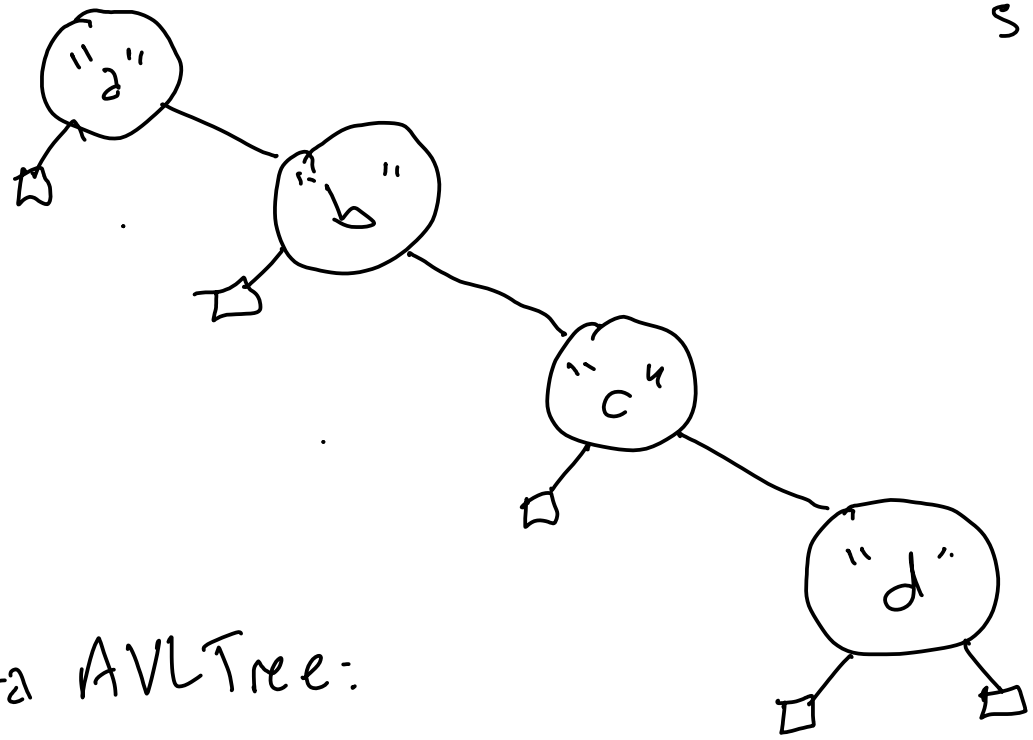
```
fun member(bt :: BTree,
           elt :: Str) -> Boolean:
  cases(BTree) bt:
    | leaf => false
    | node(v, l, r) =>
      if v == elt: true
      else:
        member(l, elt) or member(r, elt)
      end
  end
end
```



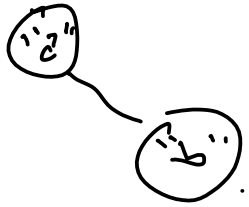
7

everything in l is less than v
r is greater than v

```
fun member(bt:: BTree,
           elt:: Str)→ Boolean:
  cases(BTree) bt:
    | leaf ⇒ false
    | node(v, l, r) ⇒
      if v == elt: true
      else if v < elt:
          member(r, elt)
      else:
          member(l, elt)
      end
  end
end
```



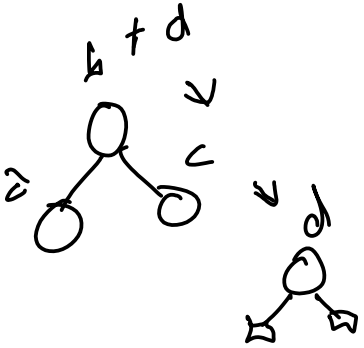
size(l) = size(r)



$|height(l) - height(r)| \leq 1$

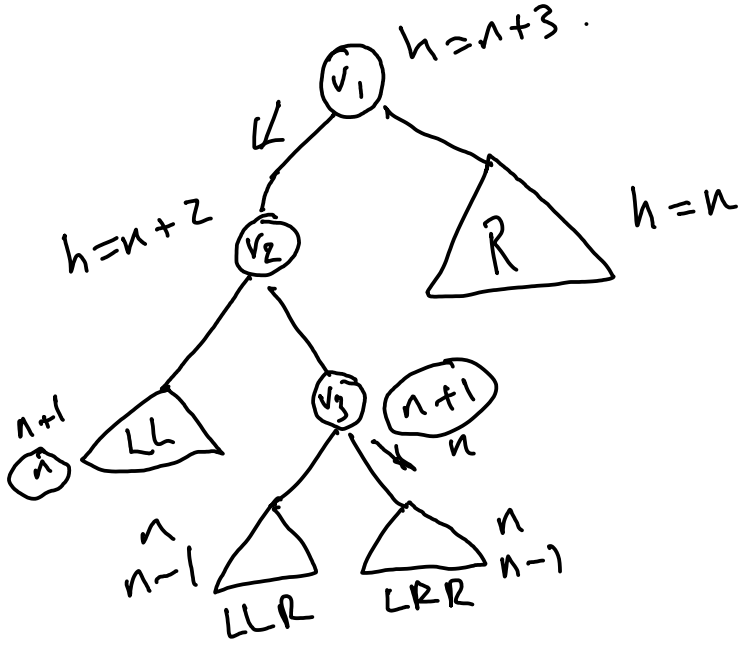
```
data AVLTree:
  | leaf
  | node(h :: Number,
         v :: String,
         l :: AVLTree,
         r :: AVLTree)
```

```
fun add(bt::AVLTree, s::String) -> AVLTree:
  cases(AVLTree) bt:
    | leaf => node(1, s, leaf, leaf)
    | node(h, v, l, r) =>
      if s == v: bt
      else if s < v:
        nl = add(l, s)
        nh = num-max(height(nl), height(r)) + 1
        rebalance(node(nh, v, nl, r))
      else: add(r, s)
      end
```

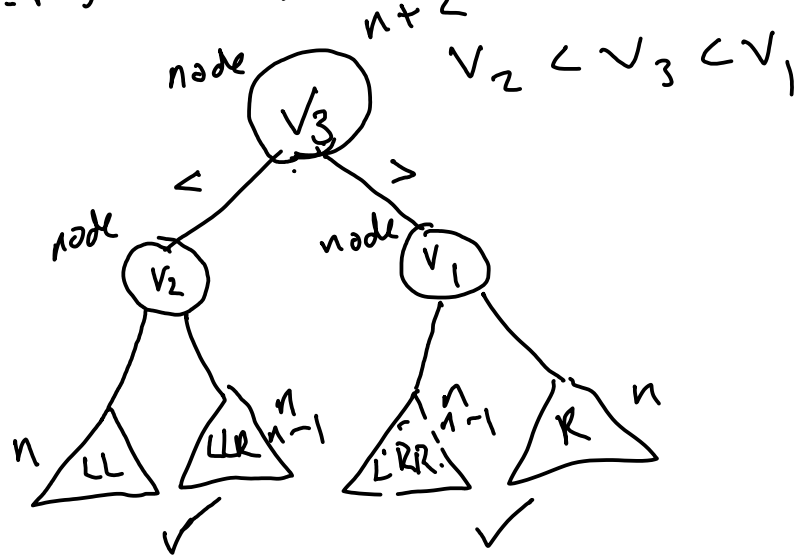


```
fun rebalance (bt :: BrokenAVL) -> AVLTree :
  if |height(bt.l) - height(bt.r)| <= 1 : bt
  else height(bt.l) - height(bt.r) == 2 :
```

bt.l.l



⇒



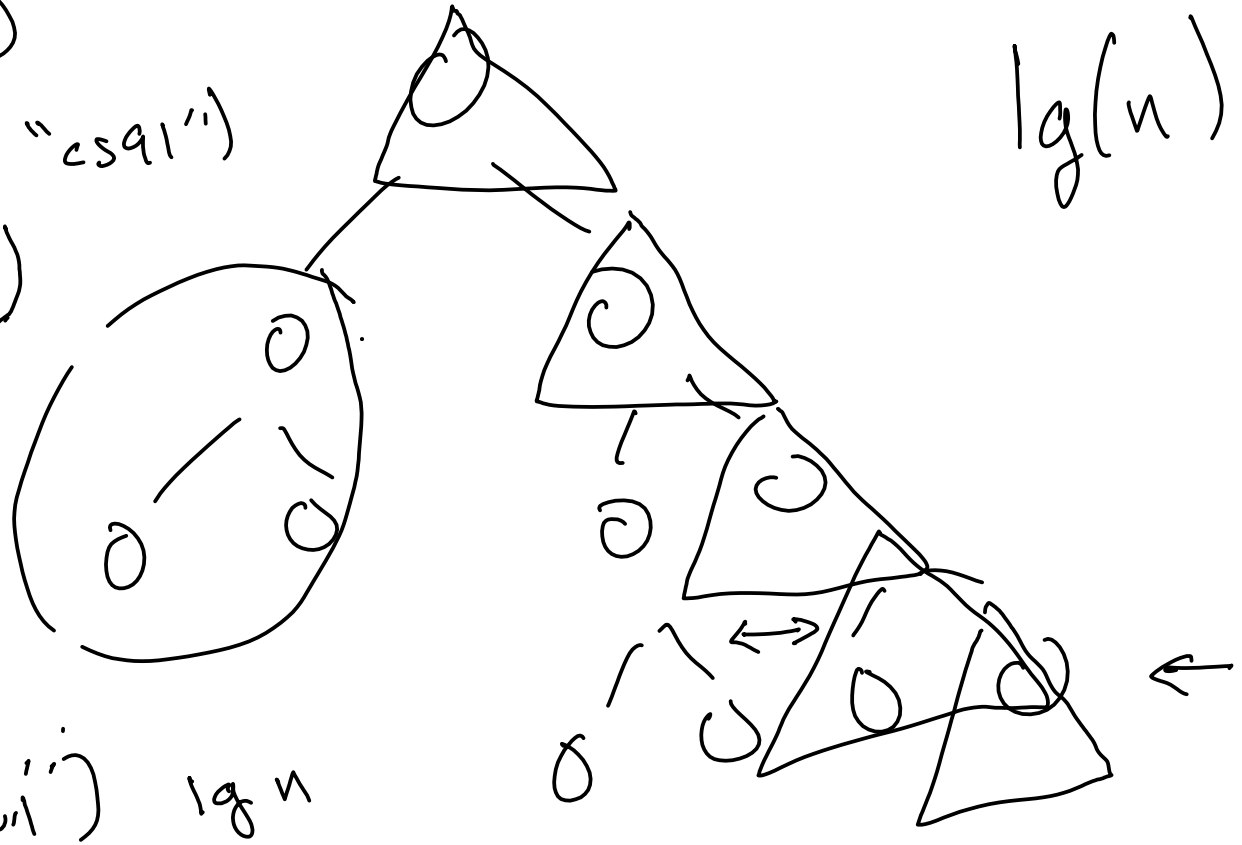
t = node(...)

t_2 = add(t, "cs91")

mem(t, "cs91")

=

false



$\lg(n)$

t_1

t_2 = add(t_1, "uil") $\lg n$

t_3 = add(t_2, "uil") $\lg n$

clone(t_1)