Binary Search Trees: AVL Trees

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Data Structures
Data Structures and Algorithms

Learning Objectives

- Understand what the height of a node is.
- State the AVL property.
- Show that trees satisfying the AVL property have low depth.

Learning Objectives

- Implement AVL trees.
- Understand the cases required for rebalancing algorithms.

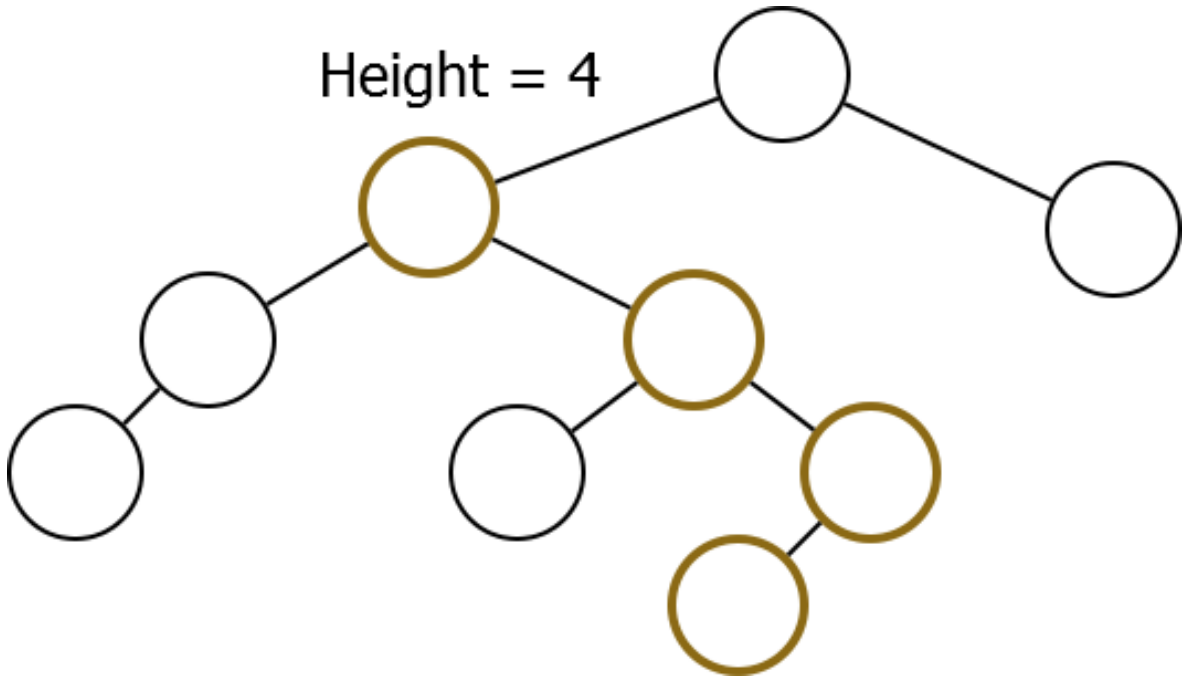
Balance

- Want to maintain balance.
- Need a way to measure balance.

Height

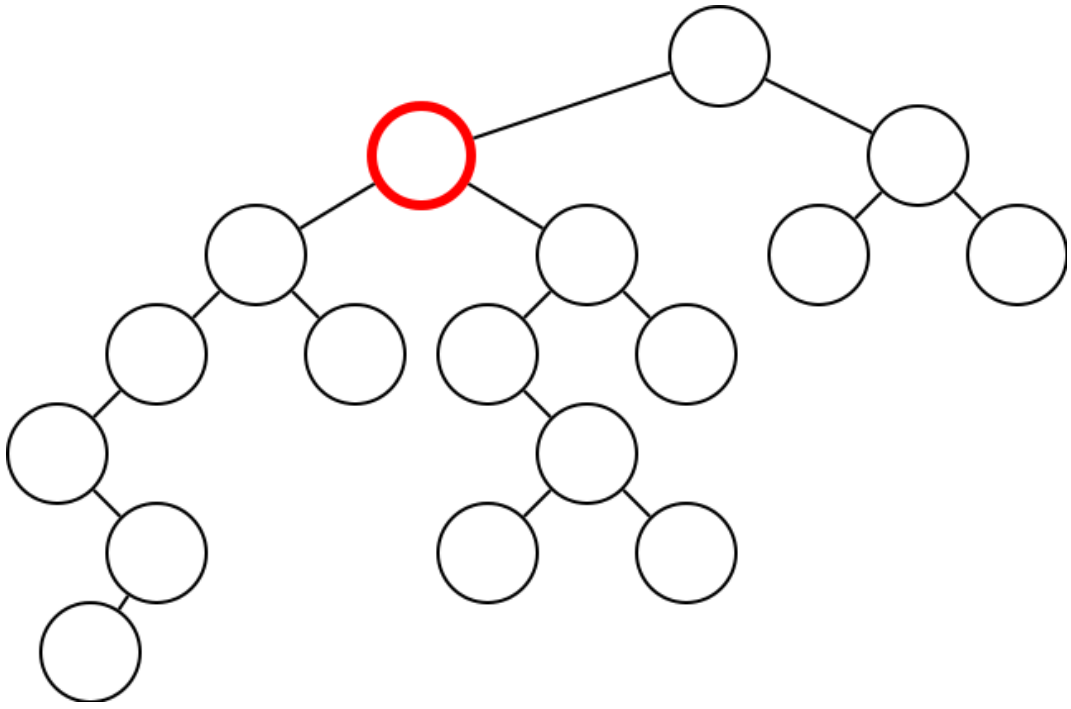
Definition

The height of a node is the maximum depth of its subtree.



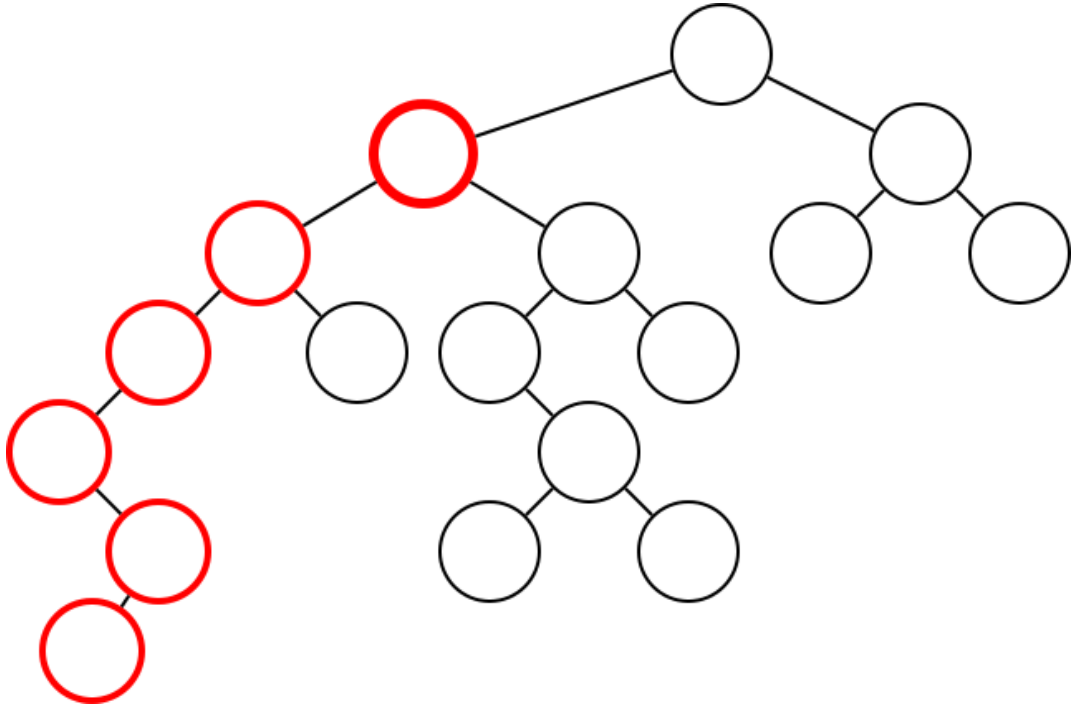
Problem

What is the height of the selected node?



Problem

What is the height of the selected node?



Recursive Definition

N.`Height` equals

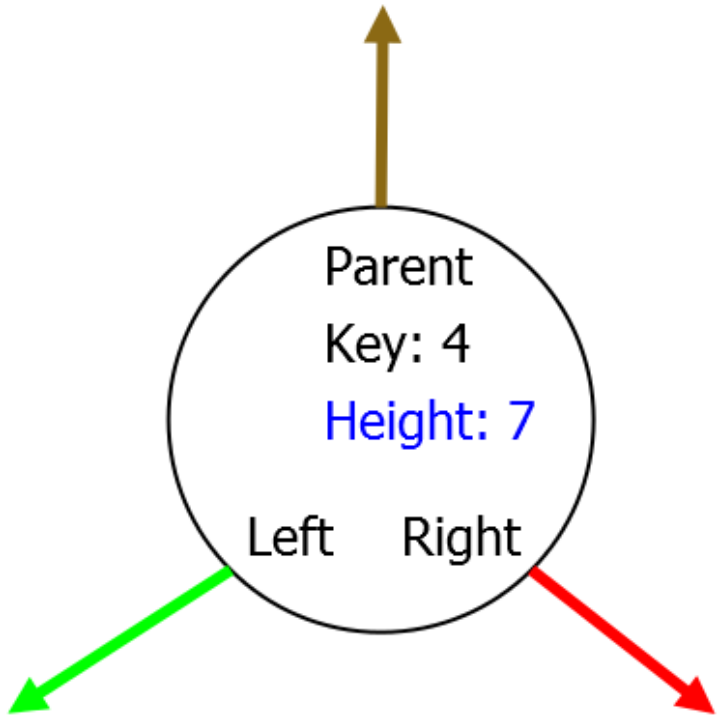
1 if N is a leaf,

$1 + \max(N.\texttt{Left.Height}, N.\texttt{Right.Height})$

otherwise.

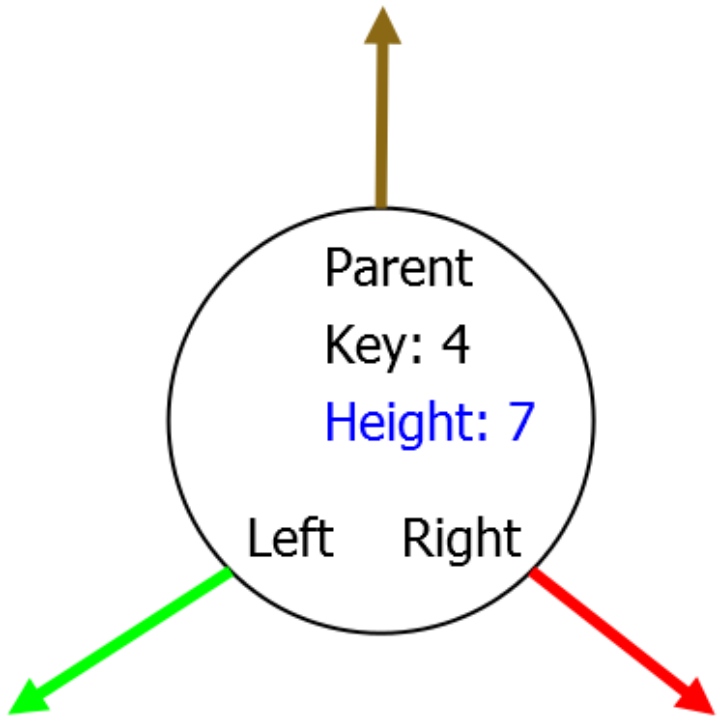
Field

Add height field to nodes.



Field

Add height field to nodes.



(Note: We'll have to work to ensure that this is kept up to date)

Balance

- Height is a rough measure of subtree size.
- Want size of subtrees roughly the same.
- Force heights to be roughly the same.

AVL Property

AVL trees maintain the following property:
For all nodes N,

$$|N.\texttt{Left.Height} - N.\texttt{Right.Height}| \leq 1$$

We claim that this ensures balance.

Idea

Need to show that AVL property implies $\mathrm{Height} = O(\log(n))$.

Idea

Need to show that AVL property implies $\mathrm{Height} = O(\log(n))$.
Alternatively, show that large height implies many nodes.

Result

Theorem

Let N be a node of a binary tree satisfying the AVL property. Let $h = N.\texttt{Height}$. Then the subtree of N has size at least the Fibonacci Number F_h.

Recall

$$F_n = \begin{cases} 0, & n = 0\,, \\ 1, & n = 1\,, \\ F_{n-1} + F_{n-2}, & n > 1\,. \end{cases}$$

Recall

$$F_n = \begin{cases} 0, & n = 0\,, \\ 1, & n = 1\,, \\ F_{n-1} + F_{n-2}, & n > 1\,. \end{cases}$$

$F_n \geq 2^{n/2}$ for $n \geq 6$.

Proof

Proof.

By induction on h.

Proof

Proof.

By induction on h.

If $h = 1$, have one node.

Proof

Proof.

By induction on h.

If $h = 1$, have one node.

Otherwise, have one subtree of height $h - 1$ and another of height at least $h - 2$.

By inductive hypothesis, total number of nodes is at least $F_{h-1} + F_{h-2} = F_h$. □

Large Subtrees

So node of height h has subtree of size at least $2^{h/2}$.

In other words, if n nodes in the tree, have height $h \leq 2\log_2(n) = O(\log(n))$.

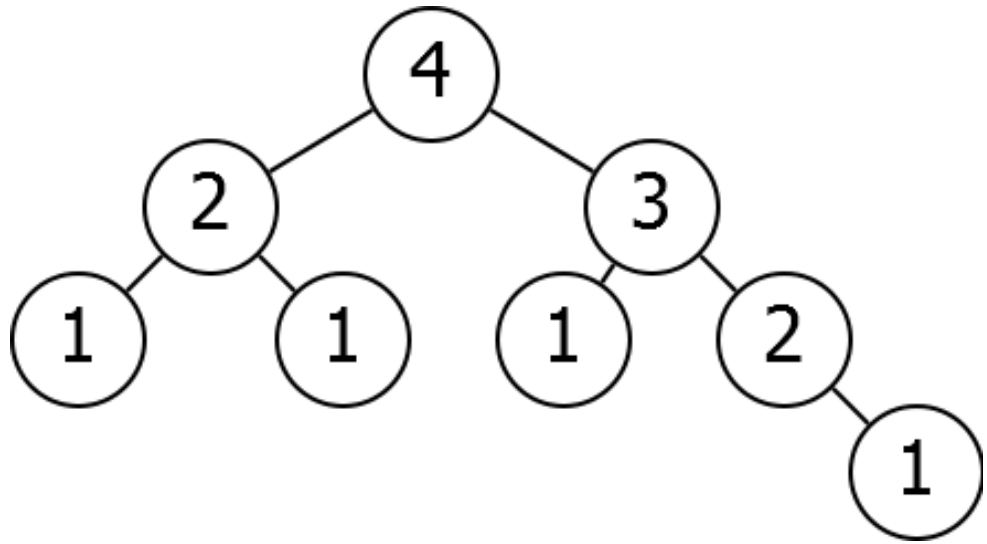
Conclusion

AVL Property

If you can maintain the AVL property, you can perform operations in $O(\log(n))$ time.

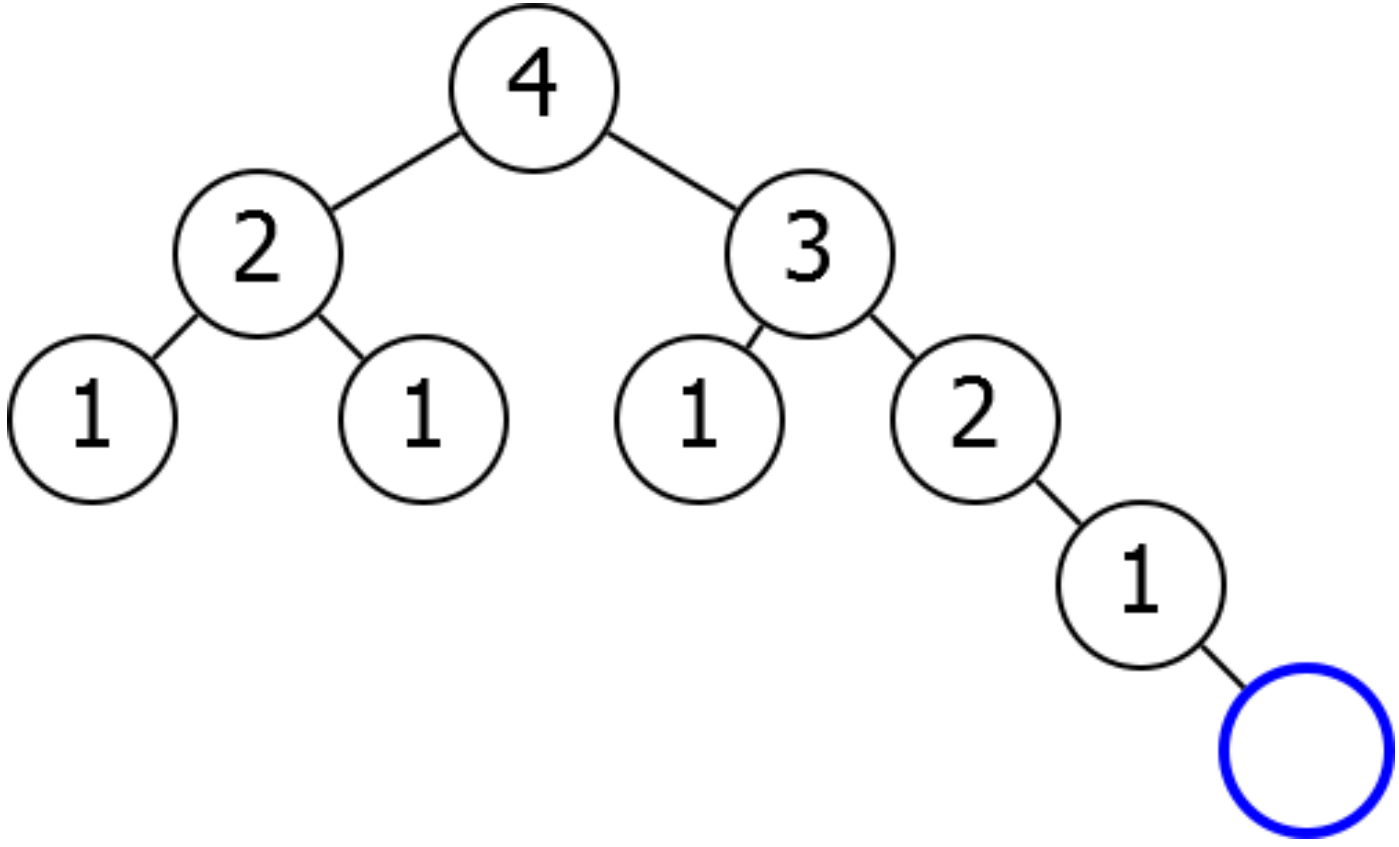
AVL Trees

Need ensure that children have nearly the same height.



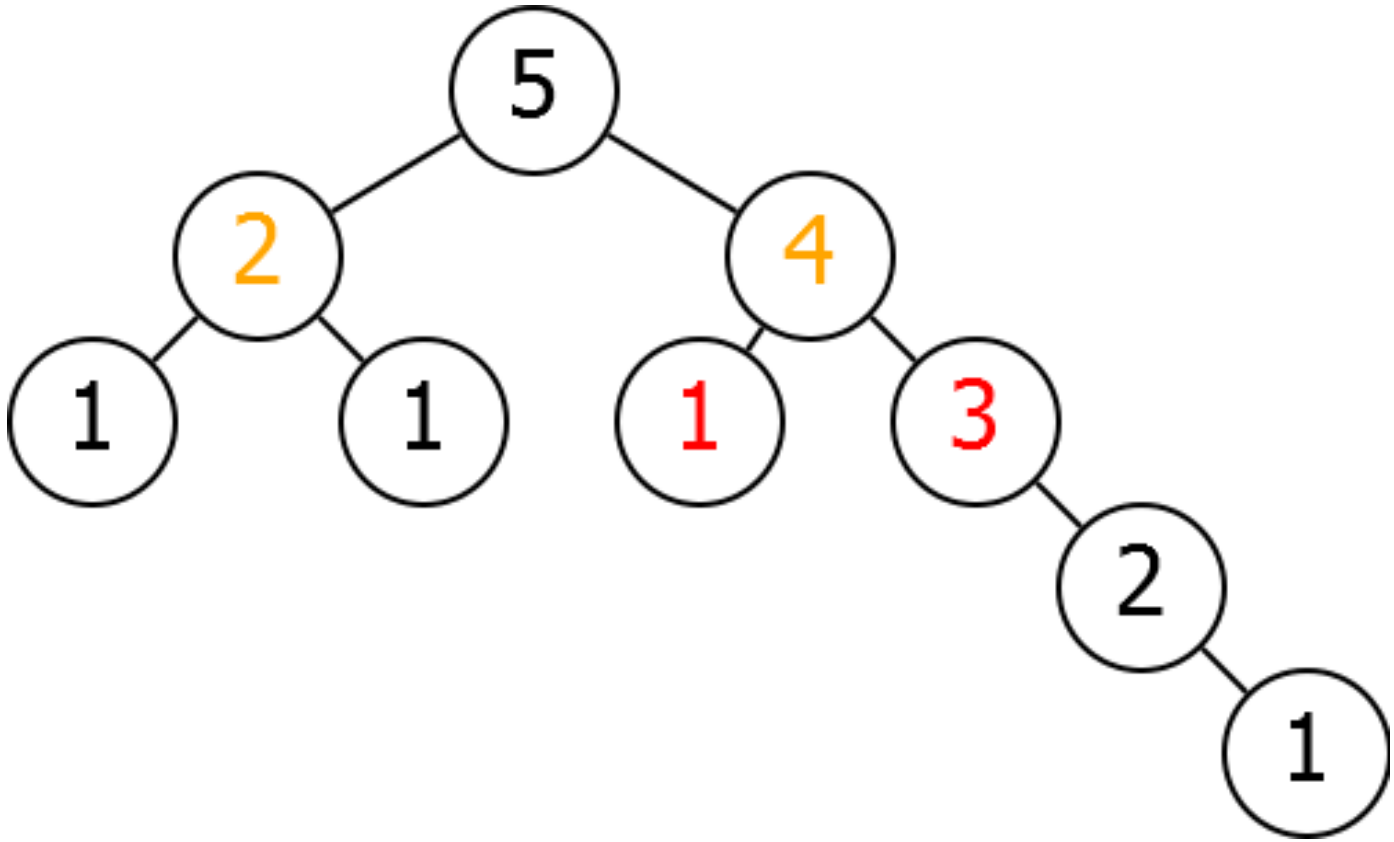
Problem

Updates to the tree can destroy this property.



Problem

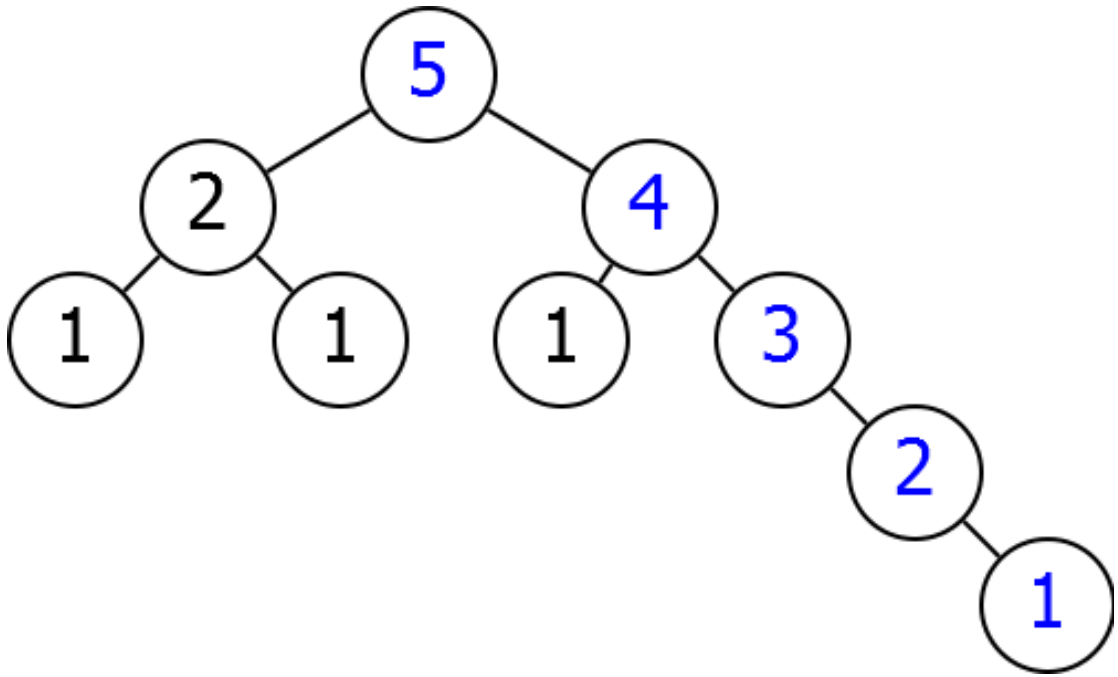
Updates to the tree can destroy this property.



Need to correct this.

Errors

Heights stay the same except on the insertion path.



Only need to worry about this path.

Insertion

We need a new insertion algorithm that involves rebalancing the tree to maintain the AVL property.

Idea

AVLInsert(k, R)

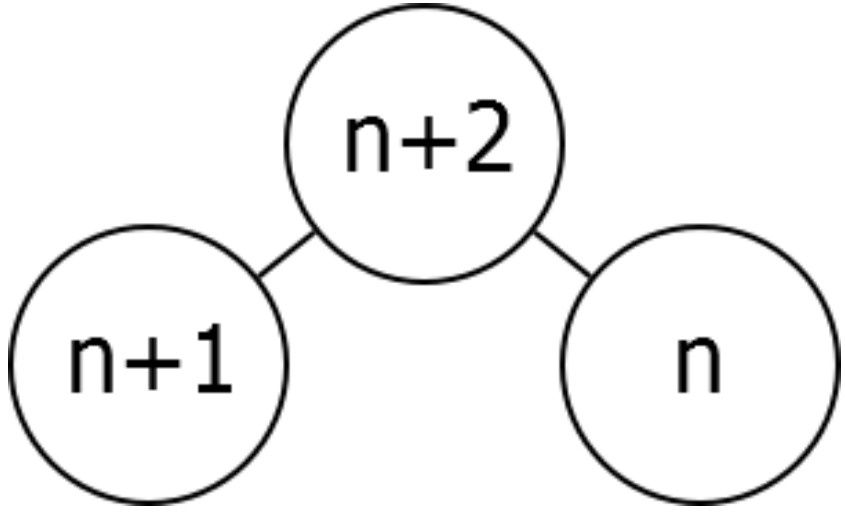
Insert(k, R)
$N \leftarrow$ Find(k, R)
Rebalance(N)

Rebalancing

If

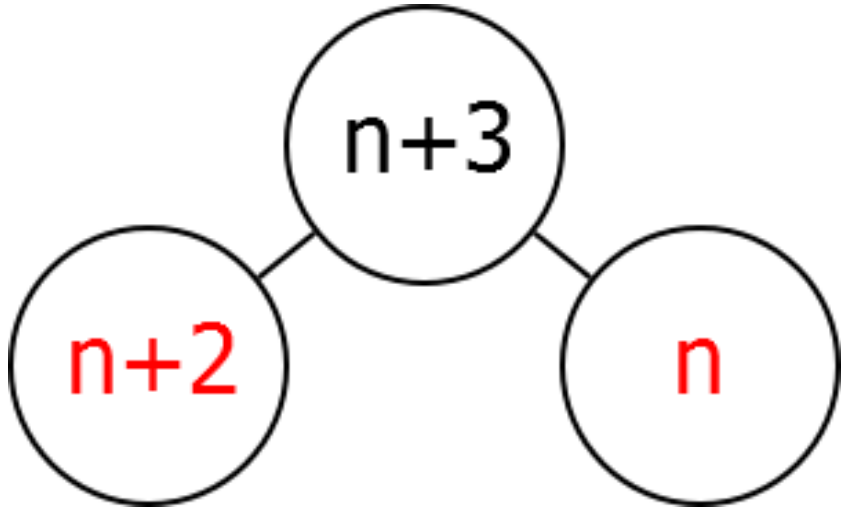
$$|N.\texttt{Left.Height} - N.\texttt{Right.Height}| \leq 1$$

fine.



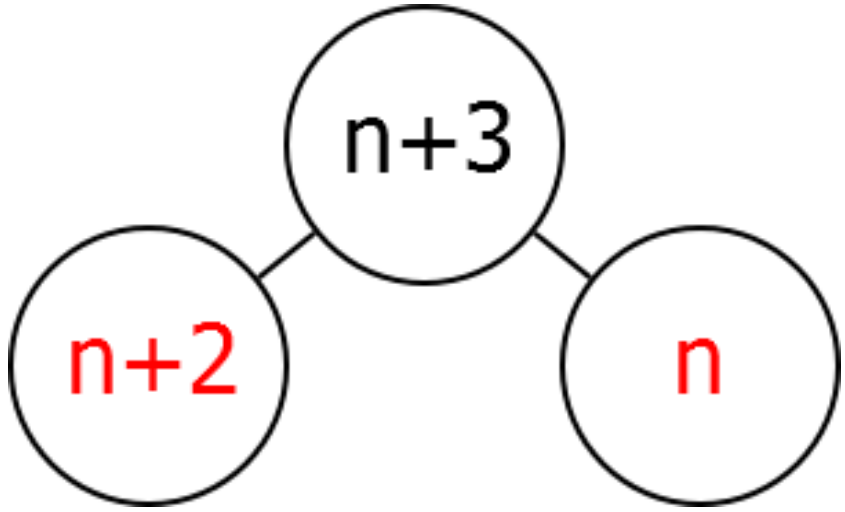
Problem

Difficulty if heights differ by more.



Problem

Difficulty if heights differ by more.



Never more than 2.

Code

Rebalance(N)

```
P ← N.Parent
if N.Left.Height > N.Right.Height+1:
  RebalanceRight(N)
if N.Right.Height > N.Left.Height+1:
  RebalanceLeft(N)
AdjustHeight(N)
if P ≠ null:
  Rebalance(P)
```

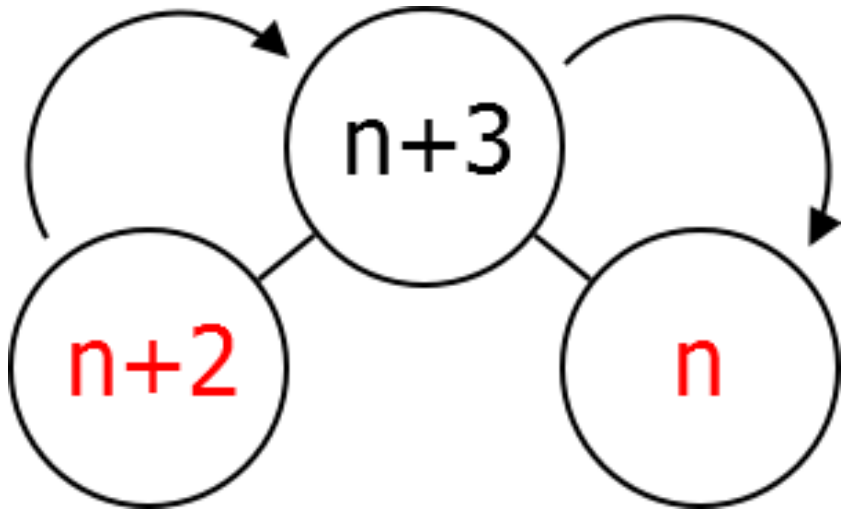
Adjust Height

AdjustHeight(N)

$$N.\texttt{Height} \leftarrow 1 + \max(N.\texttt{Left.Height}, N.\texttt{Right.Height})$$

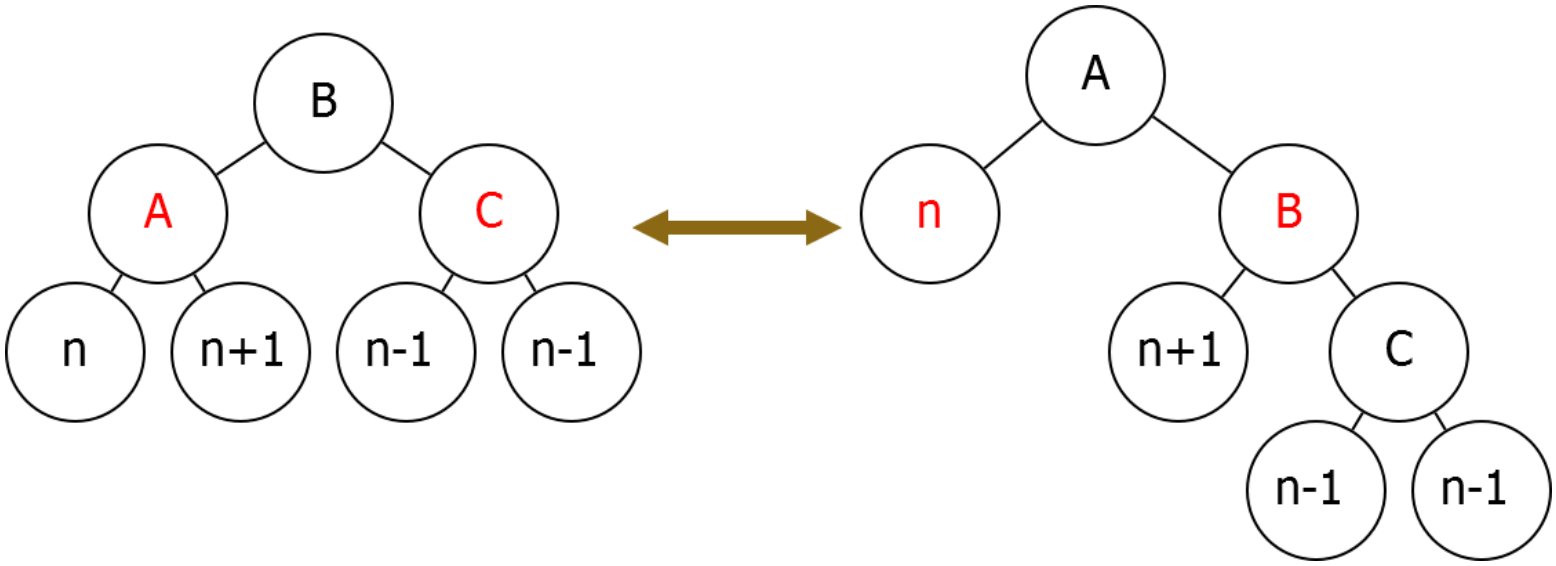
Rebalancing

If left subtree too heavy, rotate right:



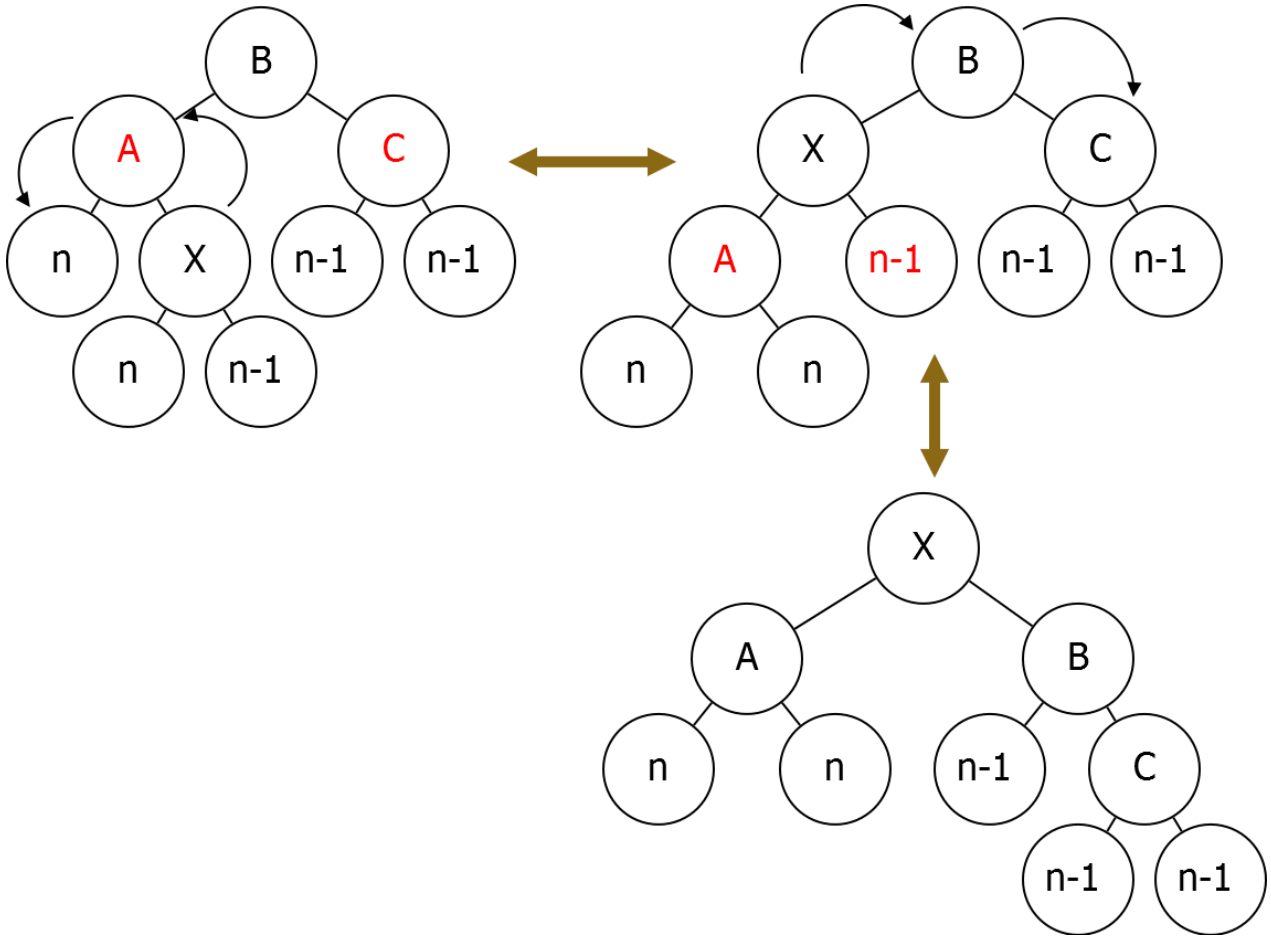
Bad Case

Doesn't work in this case.



Fix

Must rotate left first.



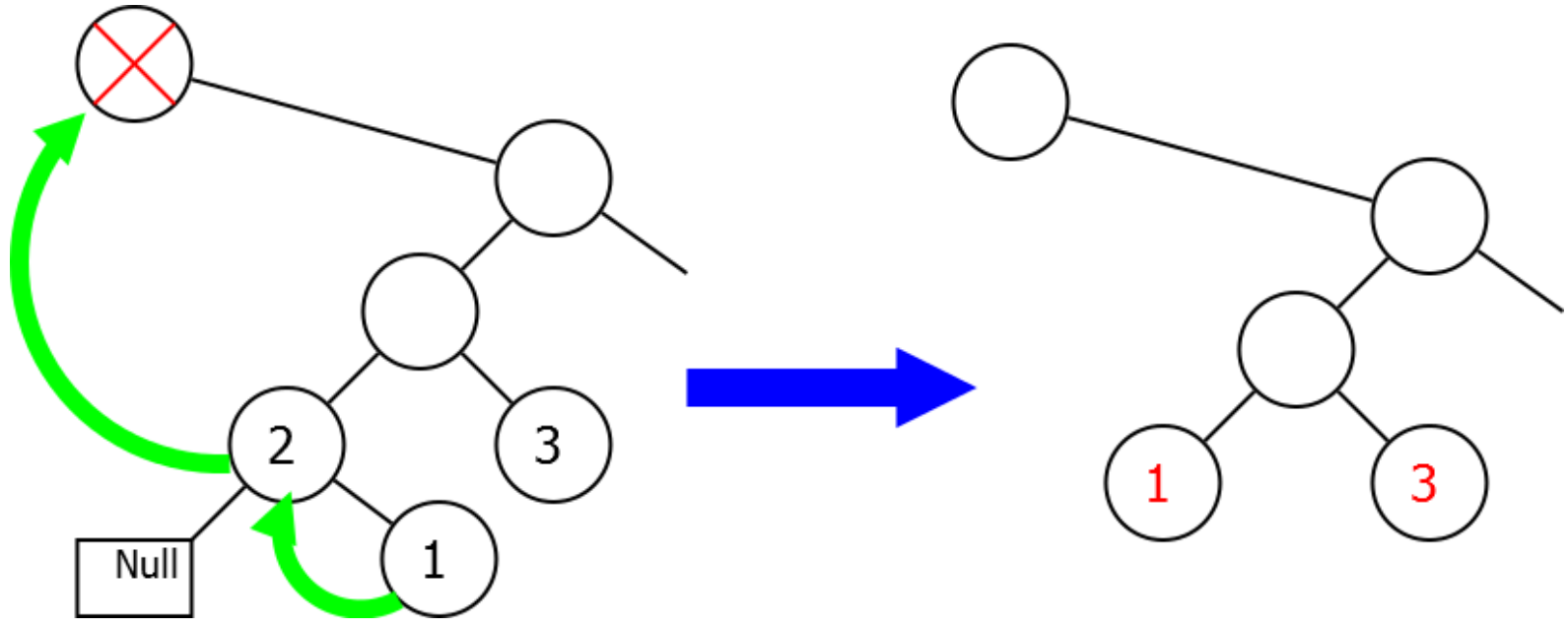
Rebalance

RebalanceRight(N)

$M \leftarrow N$.Left
if M.Right.Height $>$ M.Left.Height:
  RotateLeft(M)
RotateRight(N)
AdjustHeight on affected nodes

Delete

Deletions can also change balance.



New Delete

AVLDelete(N)

Delete(N)
$M \leftarrow$ Parent of node replacing N
Rebalance(M)

Conclusion

Summary

AVL trees can implement all of the basic operations in $O(\log(n))$ time per operation.