Binary Search Trees: AVL Tree Implementation

Daniel Kane

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

Data Structures Fundamentals
Algorithms and Data Structures

Learning Objectives

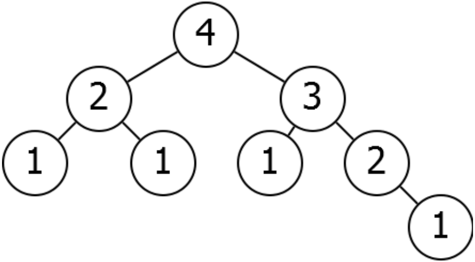
- Implement AVL trees.
- Understand the cases required for rebalancing algorithms.

Outline

1. AVL Trees
2. Insert
3. Delete

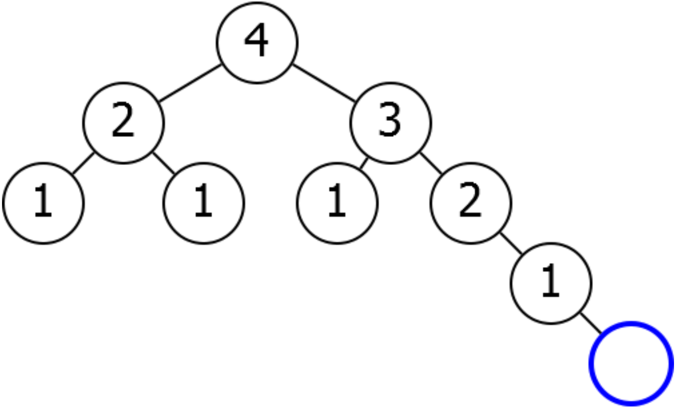
AVL Trees

Need ensure that children have nearly the same height.



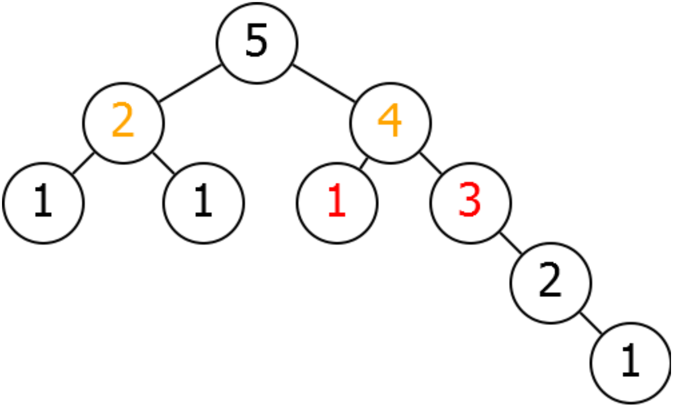
Problem

Updates to the tree can destroy this property.



Problem

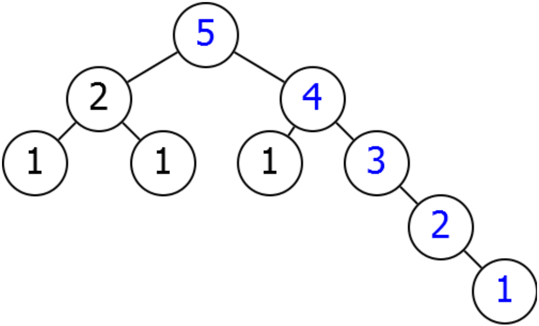
Updates to the tree can destroy this property.



Need to correct this.

Errors

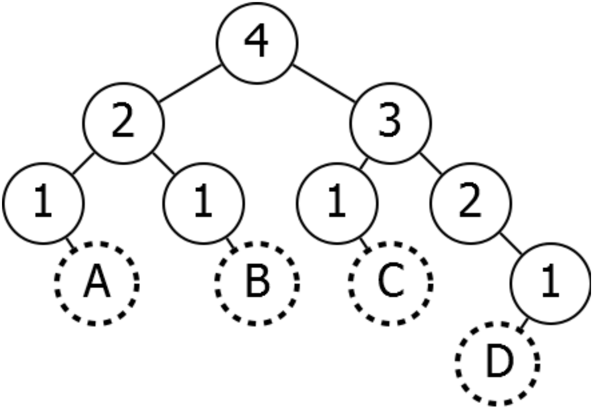
Heights stay the same except on the insertion path.



Only need to worry about this path.

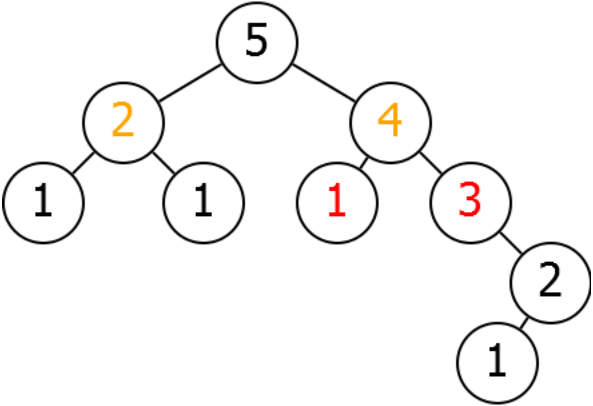
Problem

Which insertion would require the tree to be rebalanced in order to maintain the AVL property?



Problem

Which insertion would require the tree to be rebalanced in order to maintain the AVL property?



Outline

Insertion

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

Idea

$\texttt{AVLInsert}(k, R)$

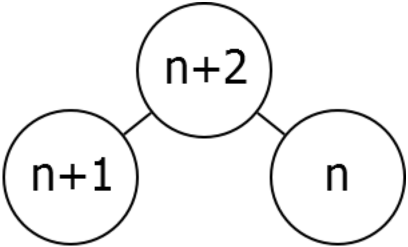
$\texttt{Insert}(k, R)$
$N \leftarrow \texttt{Find}(k, R)$
$\texttt{Rebalance}(N)$

Rebalancing

If

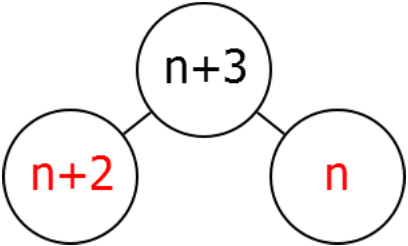
$$|N.\texttt{Left}.\texttt{Height} - N.\texttt{Right}.\texttt{Height}| \leq 1$$

fine.



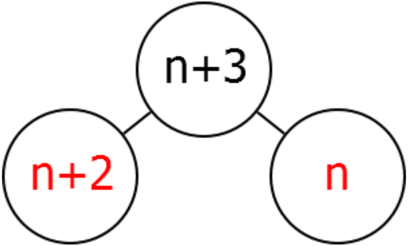
Problem

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:
```

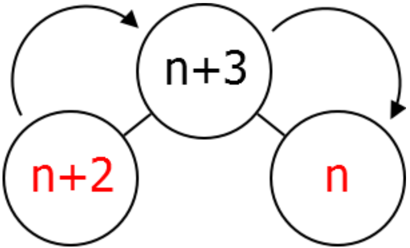
Adjust Height

`AdjustHeight(`N`)`

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

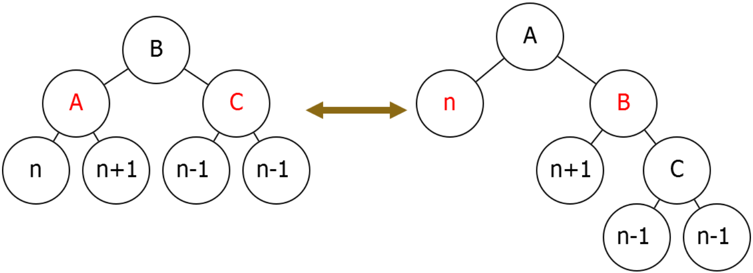
Rebalancing

If left subtree too heavy, rotate right:



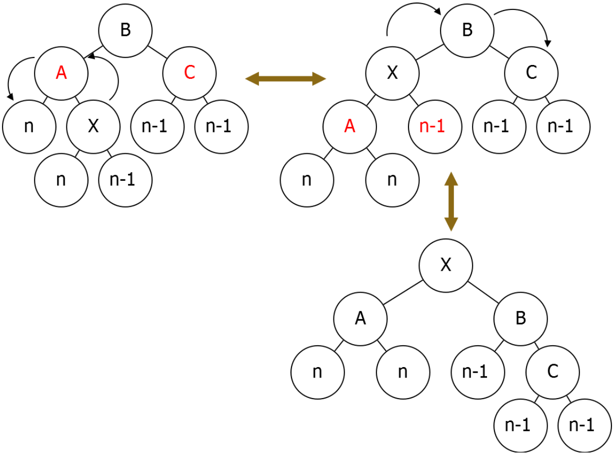
Bad Case

Doesn't work in this case.



Fix

Must rotate left first.



Rebalance

RebalanceRight(N)

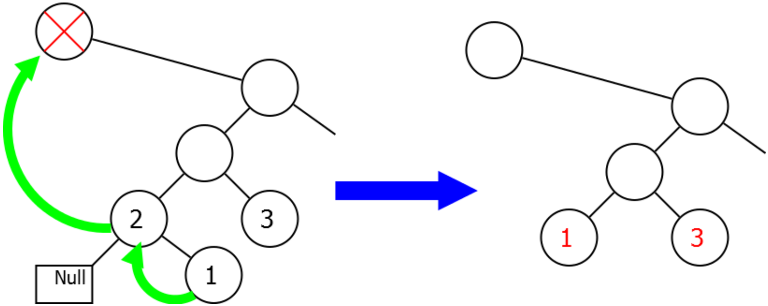
$M \leftarrow N$.Left
if M.Right.Height $>$ M.Left.Height:
  RotateLeft(M)
RotateRight(N)
AdjustHeight on affected nodes

Outline

1. AVL Trees
2. Insert
3. Delete

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.