Lecture 08: AVL Trees

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Announcements

- EX02 Due Tonight
- EX03 Due Monday
- Exam 1
  - Friday Week 5 (14 days from now)
  - https://courses.cs.washington.edu/courses/cse332/25su/exams/midterm.html

Today

- Recap: Dictionary ADT
- Review: Binary Search Trees
  - Trees
  - Basics, Properties, Operations
- Balanced BSTs?
- AVL Tree
  - Basics, Properties, Operations

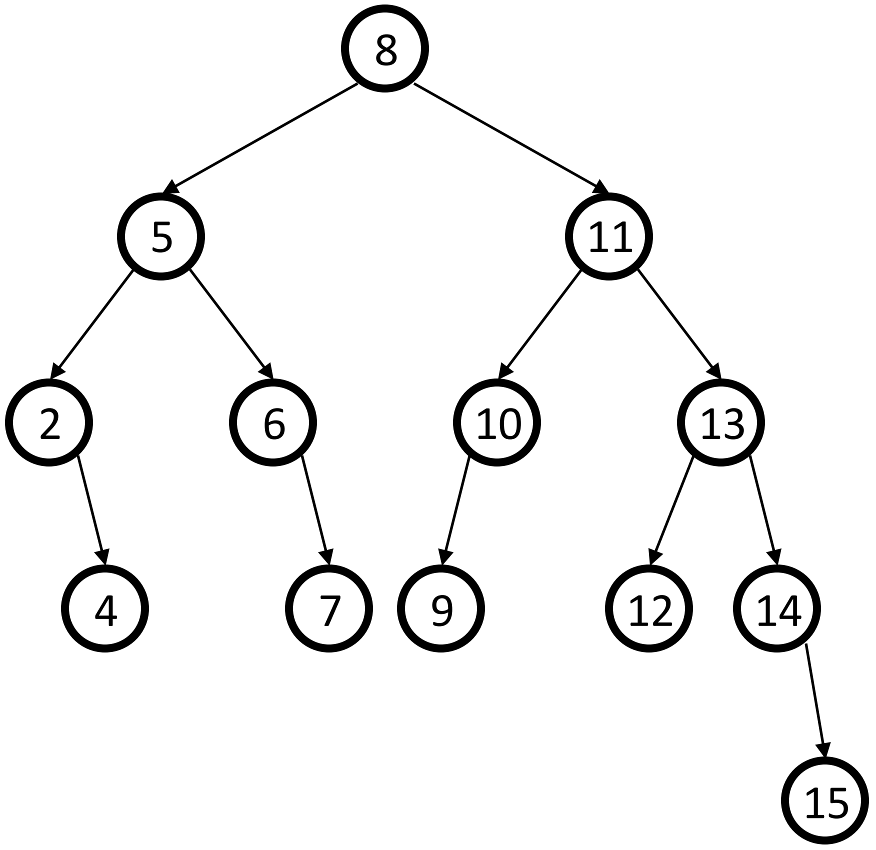
AVL Tree: Data Structure

Structural Property

- Each node has $\leq$ 2 children
- Left subtree and right subtree of *every* `node` **heights differ by at most 1**

Order Property

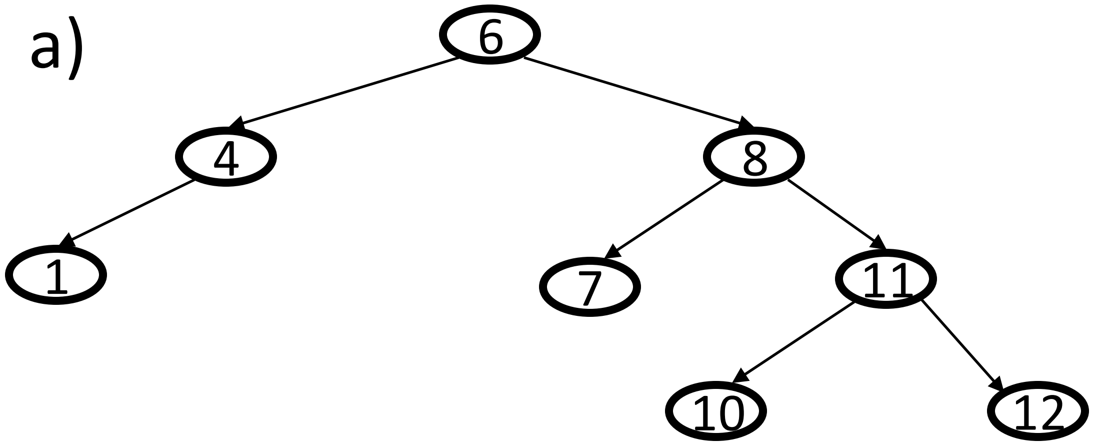
- All keys in left subtree $<$ node's key
- All keys in right subtree $>$ node's key



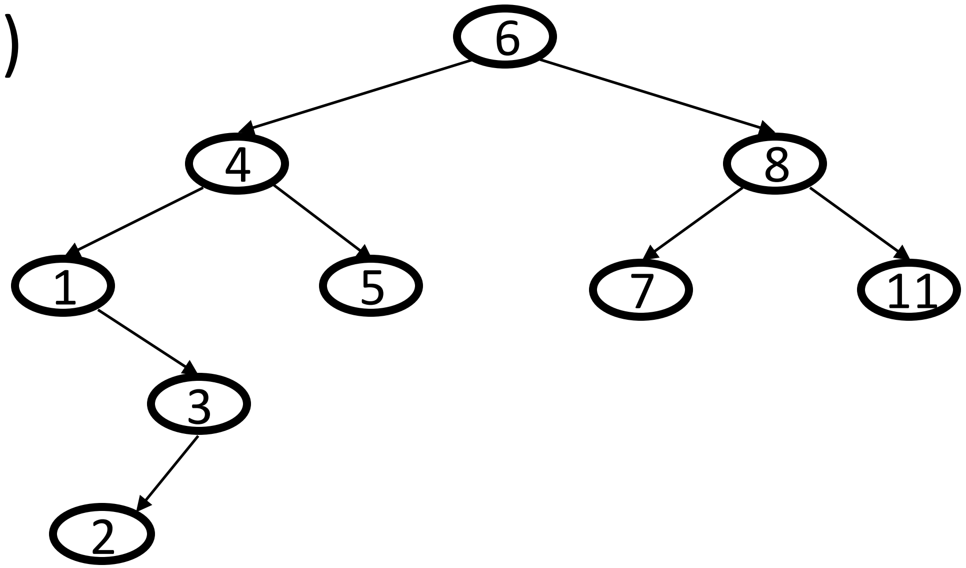
Notice: BST but with 1 extra property

AVL Tree: Is this an AVL Tree?

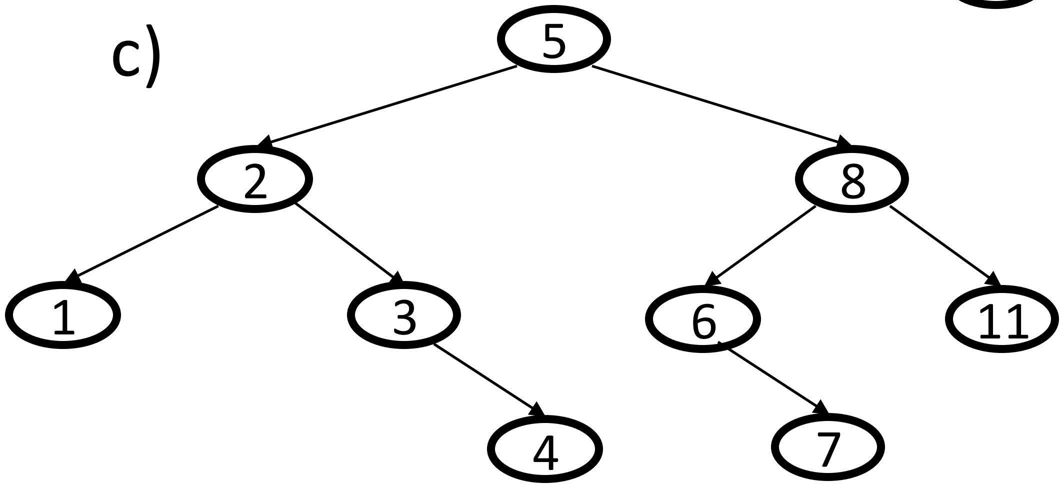
a)



b)

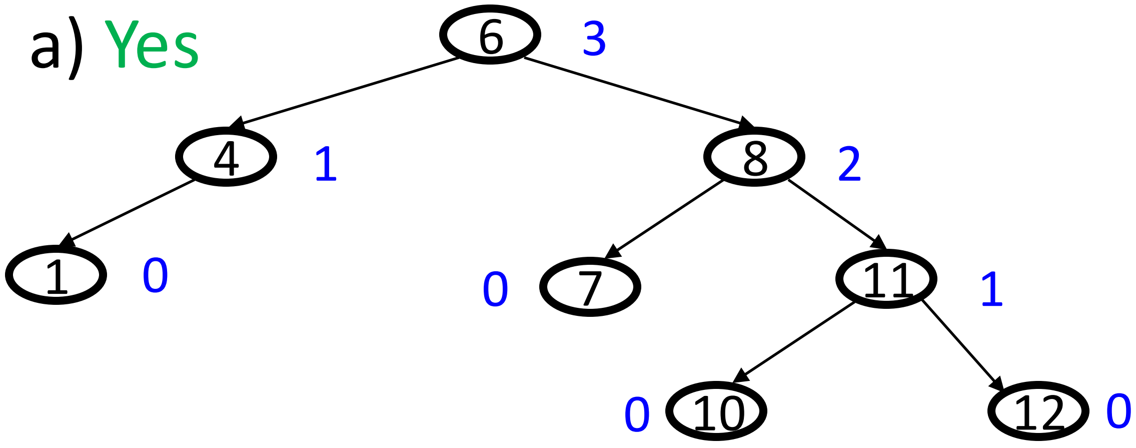


c)

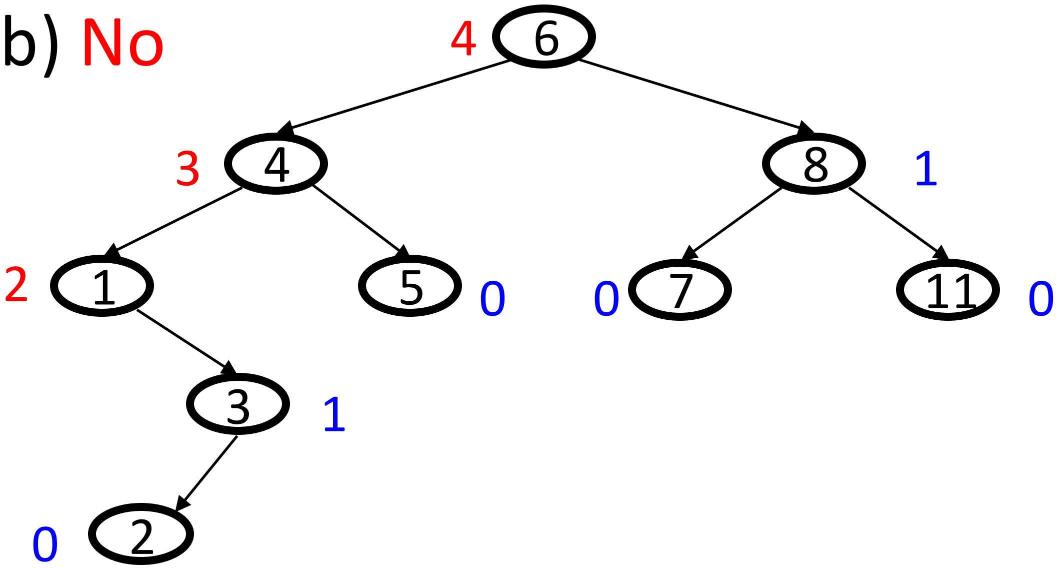


AVL Tree: Is this an AVL Tree? (Soln.)

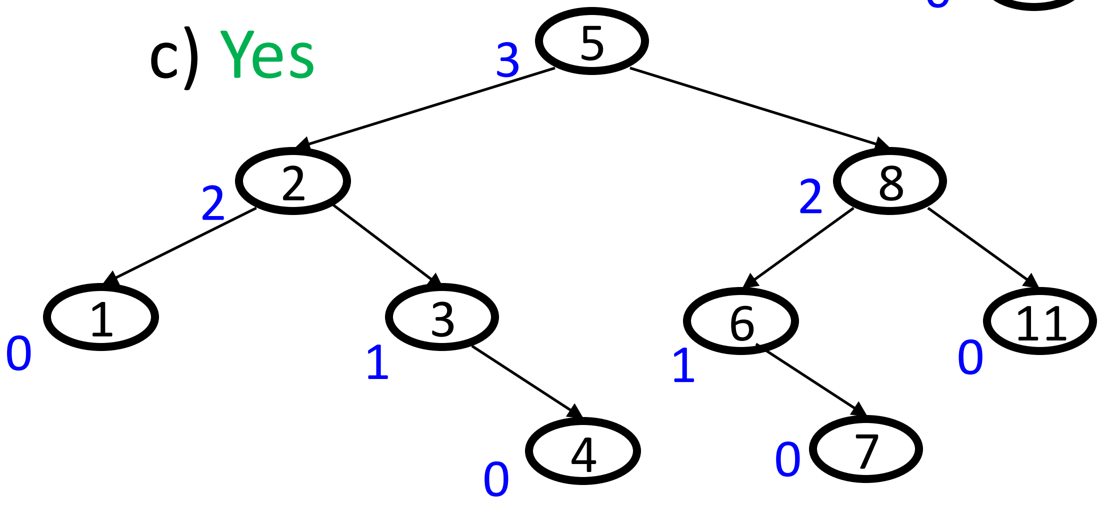
a) Yes



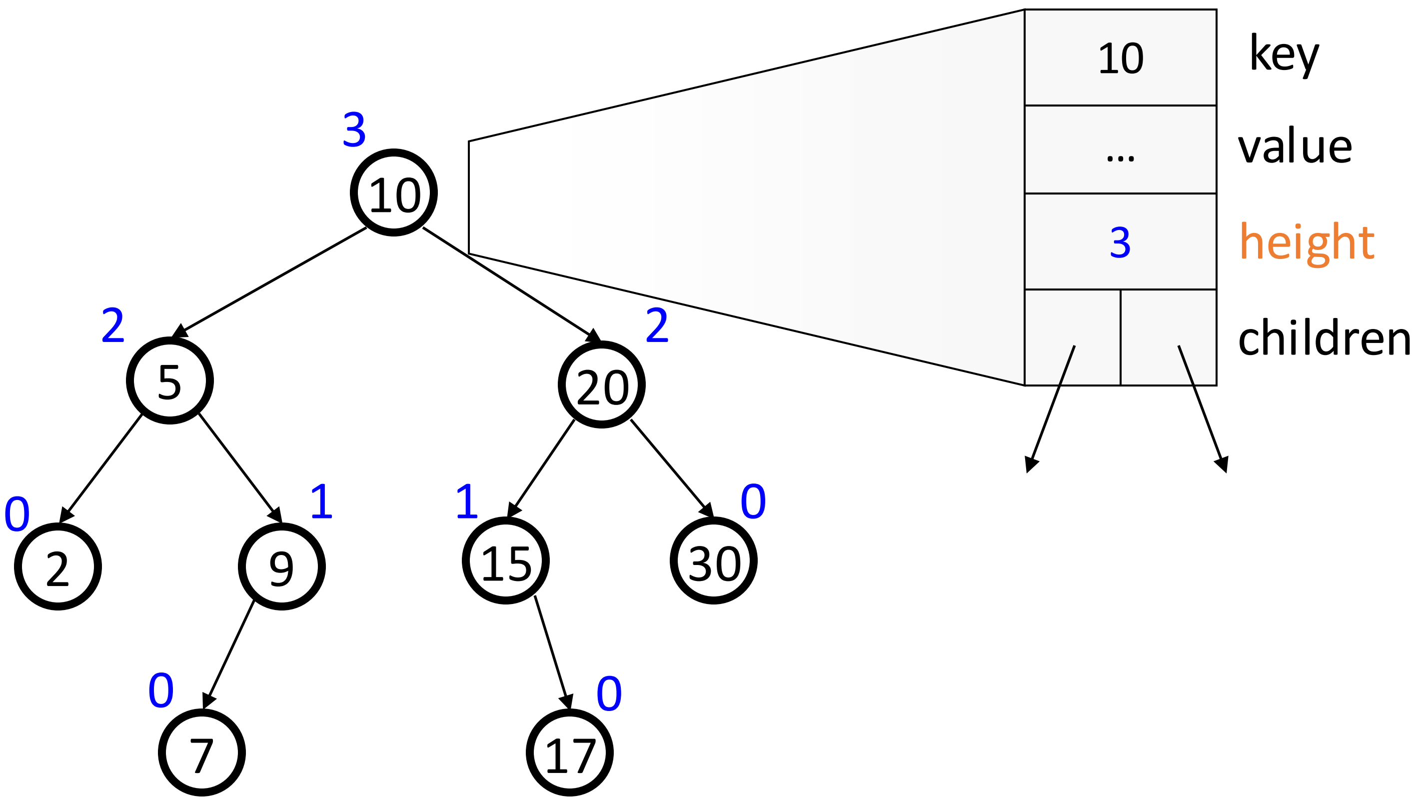
b) No



c) Yes



AVL Tree: Node Visualization



AVL Tree: Operations

- `find`
  - Same as BST
- `insert`
  - BST `insert`
  - Check and Fix Balance (4 cases)
- `delete`
  - Lazy Deletion:
    - `find`
    - Mark as deleted
  - Non-lazy Deletion:
    - BST `delete`
    - Check and Fix Balance

Today

- Recap: AVL Tree
- AVL Tree `insert`
  - General
  - Single Rotation
  - Double Rotation
- AVL Tree Conclusions

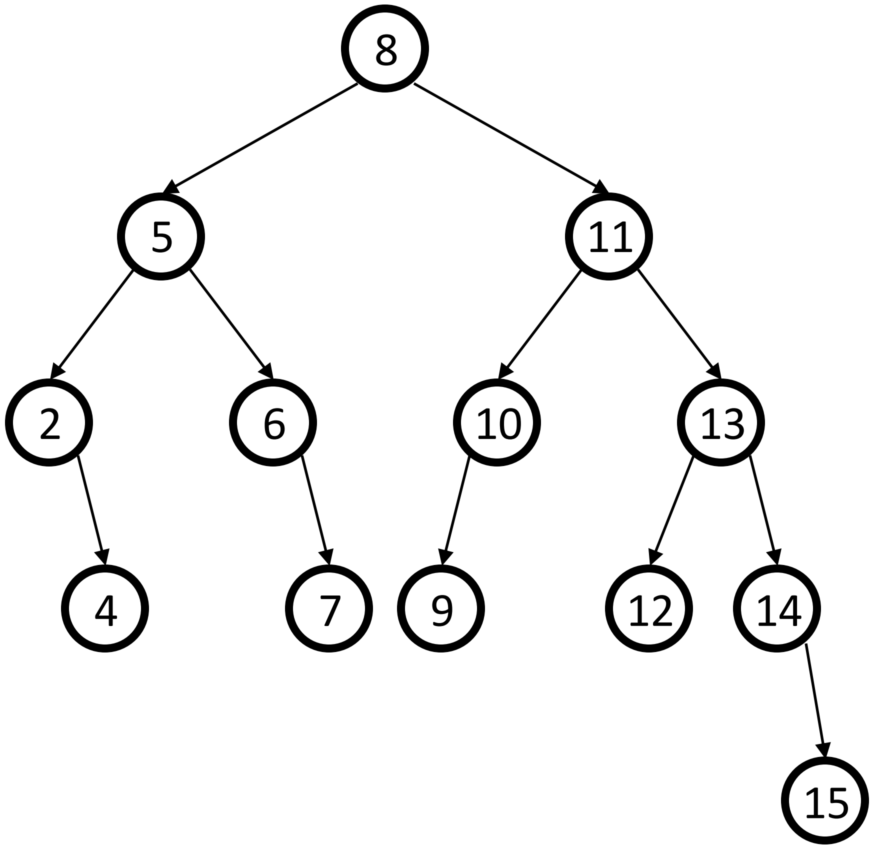
AVL: Data Structure

Structural Property

- Each node has $\leq$ 2 children
- Left subtree and right subtree of *every* `node` **heights differ by at most 1**

Order Property

- All keys in left subtree $<$ node's key
- All keys in right subtree $>$ node's key



Notice: BST but with 1 extra property

AVL: Operations

- `find`
  - Same as BST
- `insert`
  - BST `insert`
  - Check and Fix Balance (4 cases)
- `delete`
  - Lazy Deletion:
    - `find`
    - Mark as deleted
  - Non-lazy Deletion:
    - BST `delete`
    - Check and Fix Balance

Today

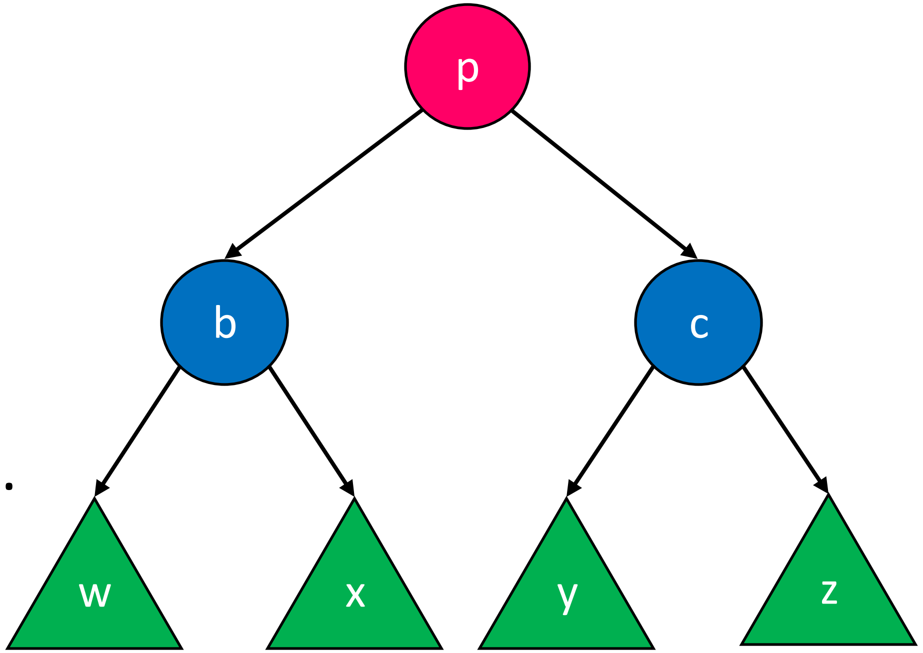
- Recap: AVL Tree
- AVL Tree `insert`
    - General
    - Single Rotation
    - Double Rotation
- AVL Tree Conclusions

AVL: `insert`

Let p be the **problem node** where an imbalance occurs:

The inserted node is in the

1. Left-Left: left subtree of the left child of p.
2. Right-Left: right subtree of the left child of p.
3. Left-Right: left subtree of the right child of p.
4. Right-Right: right subtree of the right child of p.

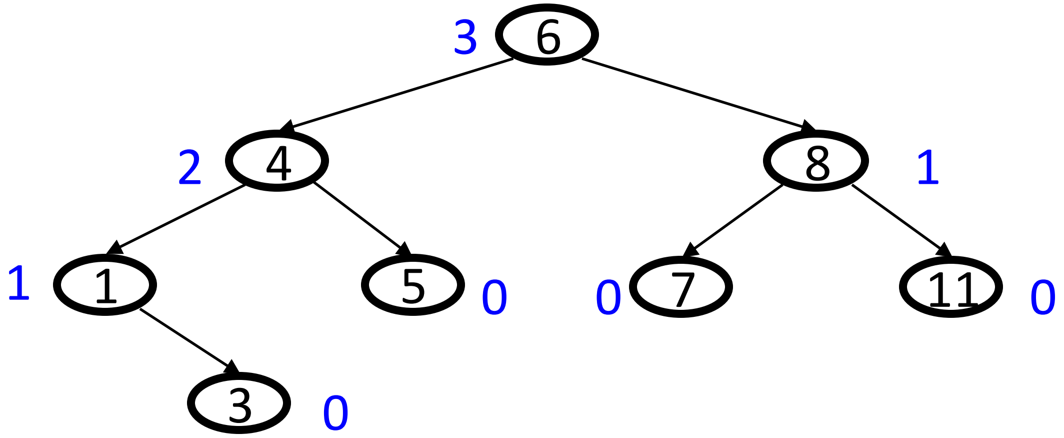


Idea:

- Cases 1 & 4 are solved by a single rotation
- Cases 2 & 3 are solved by a double rotation

AVL: `insert`, Algorithm (finding + fixing p)

1. BST `insert`
   - After: Every node's **height** in the path to the bottom *may* have changed
2. **Recursive Backtracking**:
   - Calculate new height
   - Detect height imbalance
3. If imbalance: find case + rotate

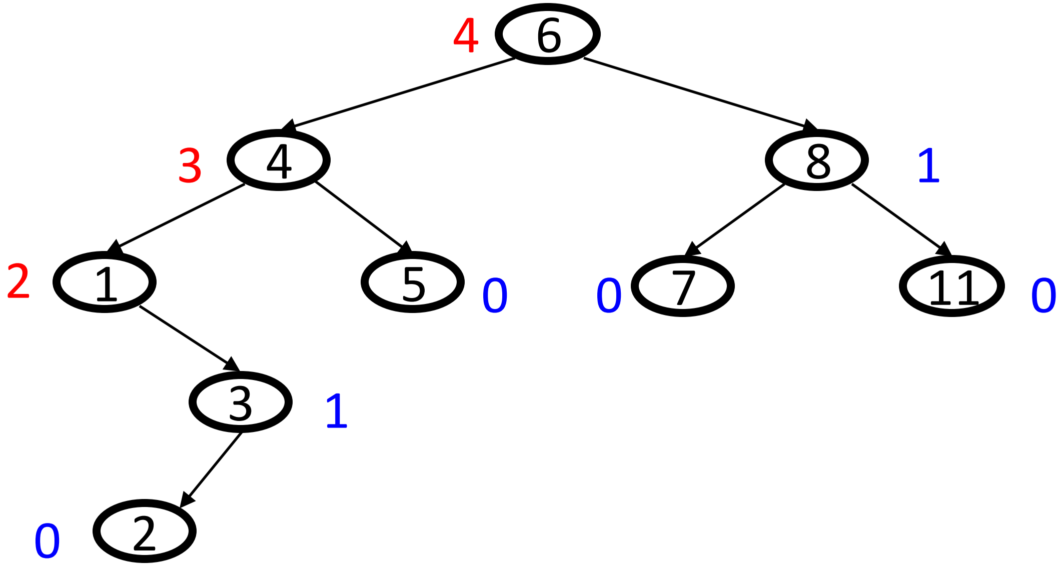


Notes:

- Only 1 deepest imbalanced node (p)
- Rebalancing deepest p = everything else is balanced

Conclusion: Only 1 p needs balancing

AVL: `insert`, Only 1 p needs balancing

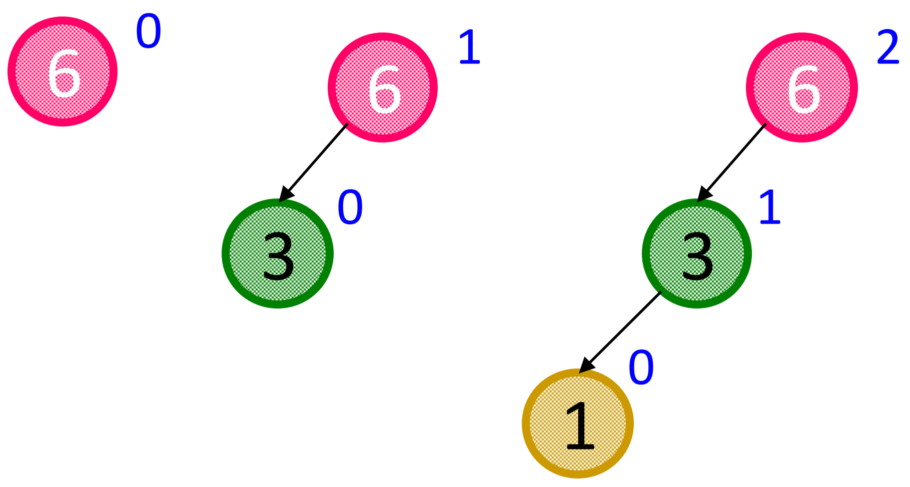


Today

- Recap: AVL Tree
- AVL Tree `insert`
  - General
  - Single Rotation
  - Double Rotation
- AVL Tree Conclusions

AVL: `insert` Case 1 Left-Left Example 1

1. `insert(6)`
2. `insert(3)`
3. `insert(1)` violates balance property



p Happens to be at the root

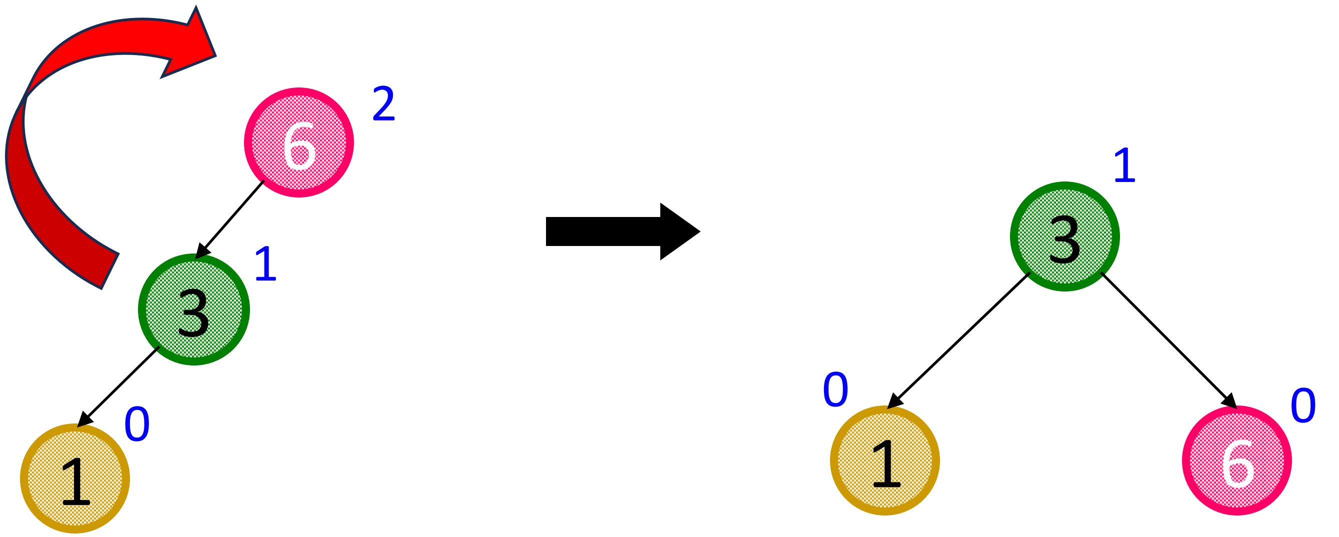
What is the only way to fix this?

Single Rotation!

AVL: `insert`, Left-Left Single Rotation

The basic operation we'll use to rebalance

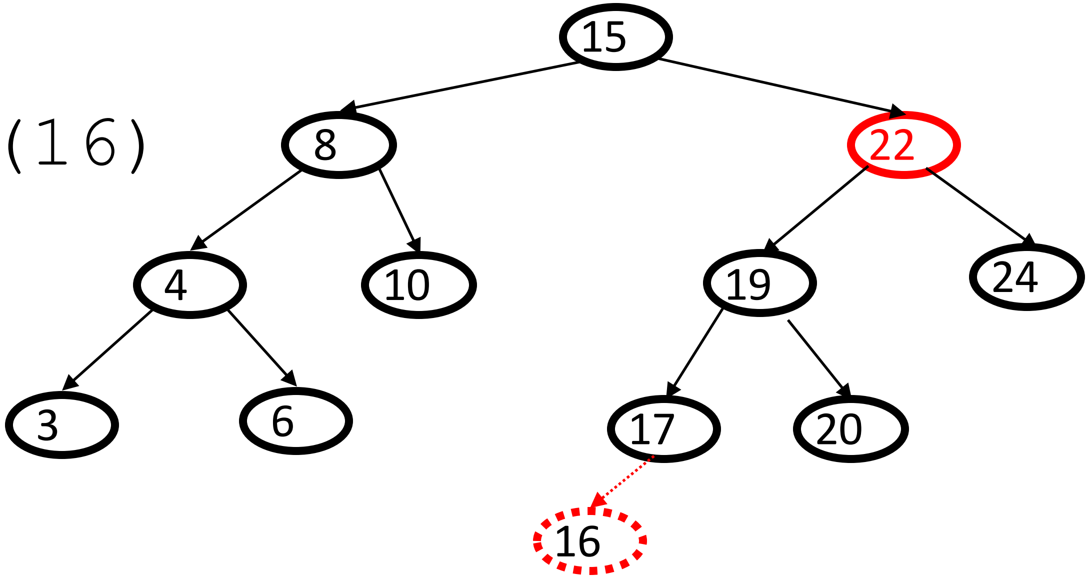
1. Move child of p to position of p
2. p becomes the "other" child
3. Other subtrees move (based on what BST allows)



Any Questions?

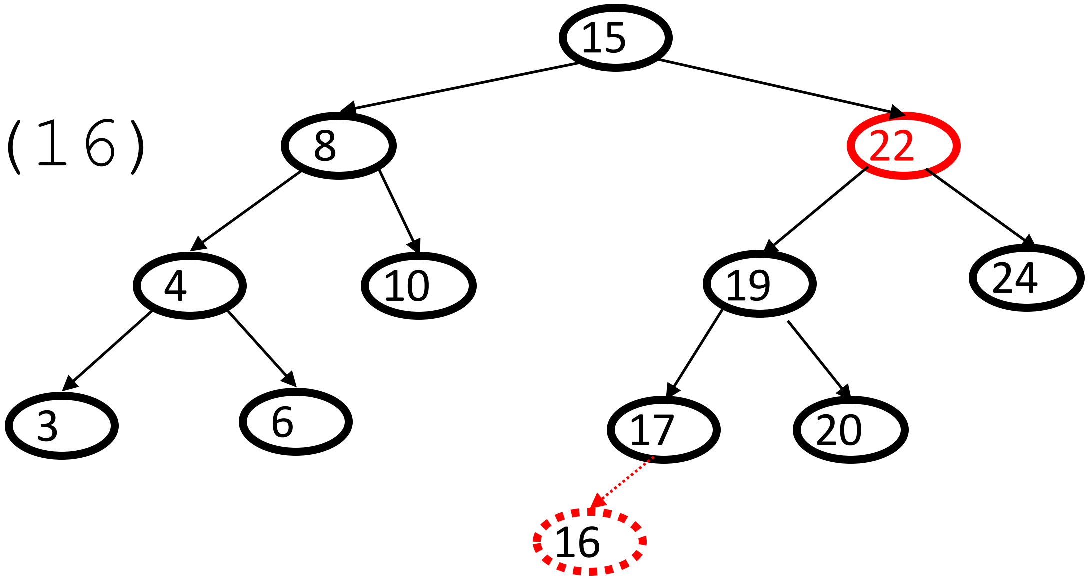
AVL: `insert` Case 1 Left-Left Example 2

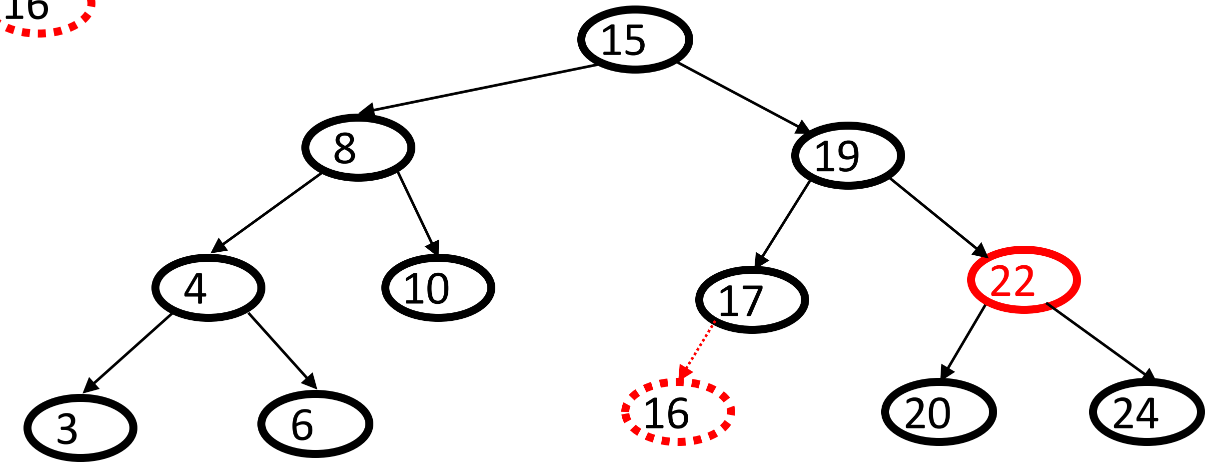
`insert(16)`



AVL: `insert` Case 1 Left-Left Example 2 (Soln.)

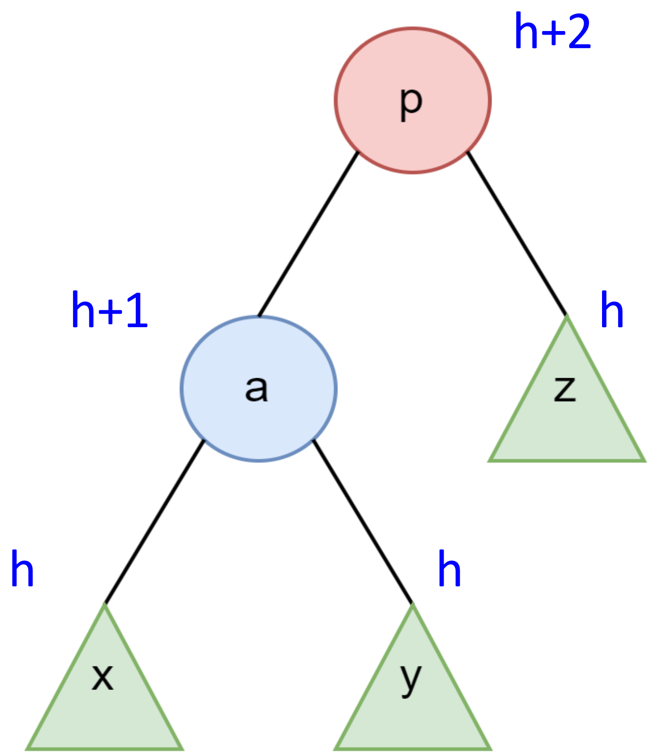
`insert(16)`





AVL: Single Rotation Pseudocode

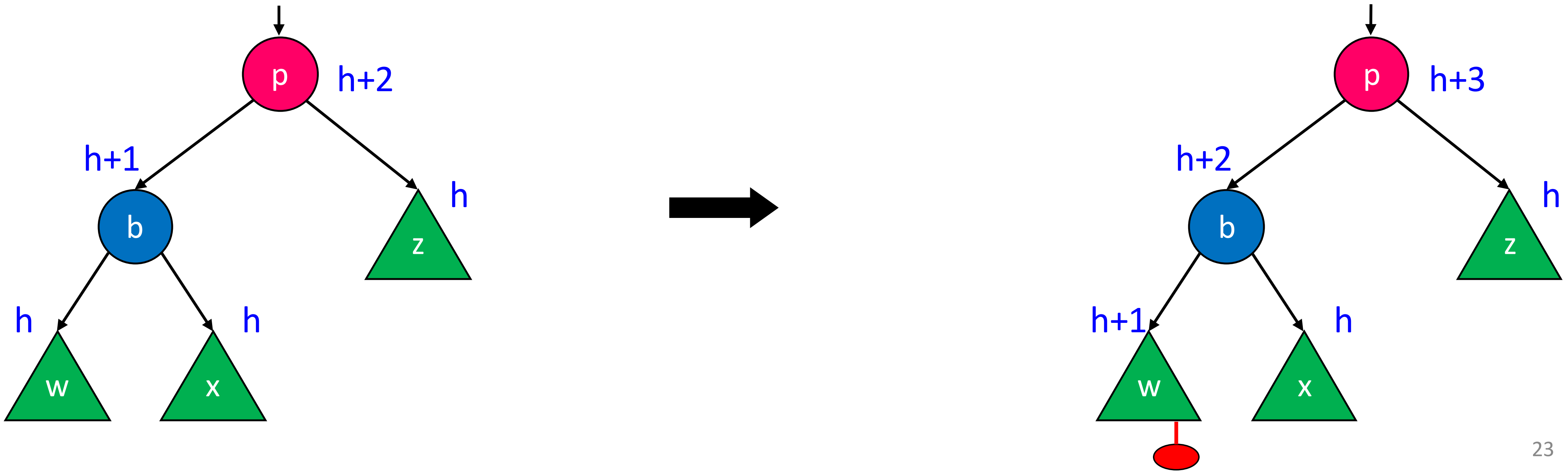
```
Node RotateWithLeft(Node root) {
  Node temp = root.left
  root.left = temp.right
  temp.right = root
  root.height = max(root.left.height(),
                    root.right.height()) + 1
  temp.height = max(temp.left.height(),
                    temp.right.height()) + 1
  root = temp
  return root
}
```



AVL: `insert`, Left-Left Single Rotation

Node p imbalanced due to insertion somewhere in Left-Left "Grandchild subtree" increasing height

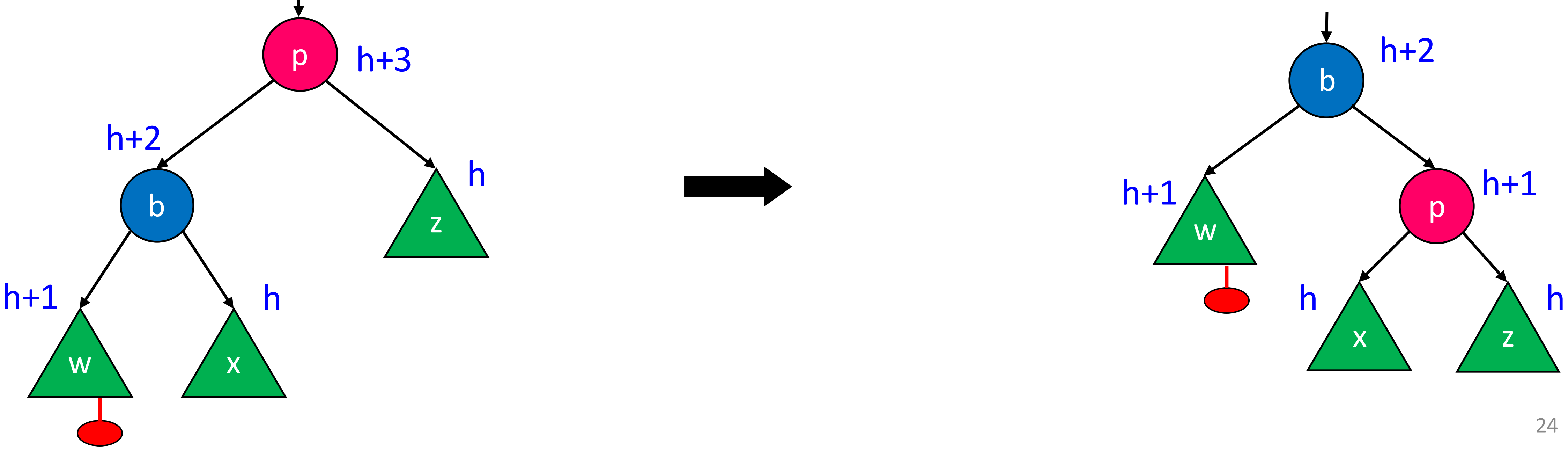
1. Insert a node at w: p becomes imbalanced



AVL: `insert`, Left-Left Single Rotation

Node p imbalanced due to insertion somewhere in Left-Left "Grandchild subtree" increasing height

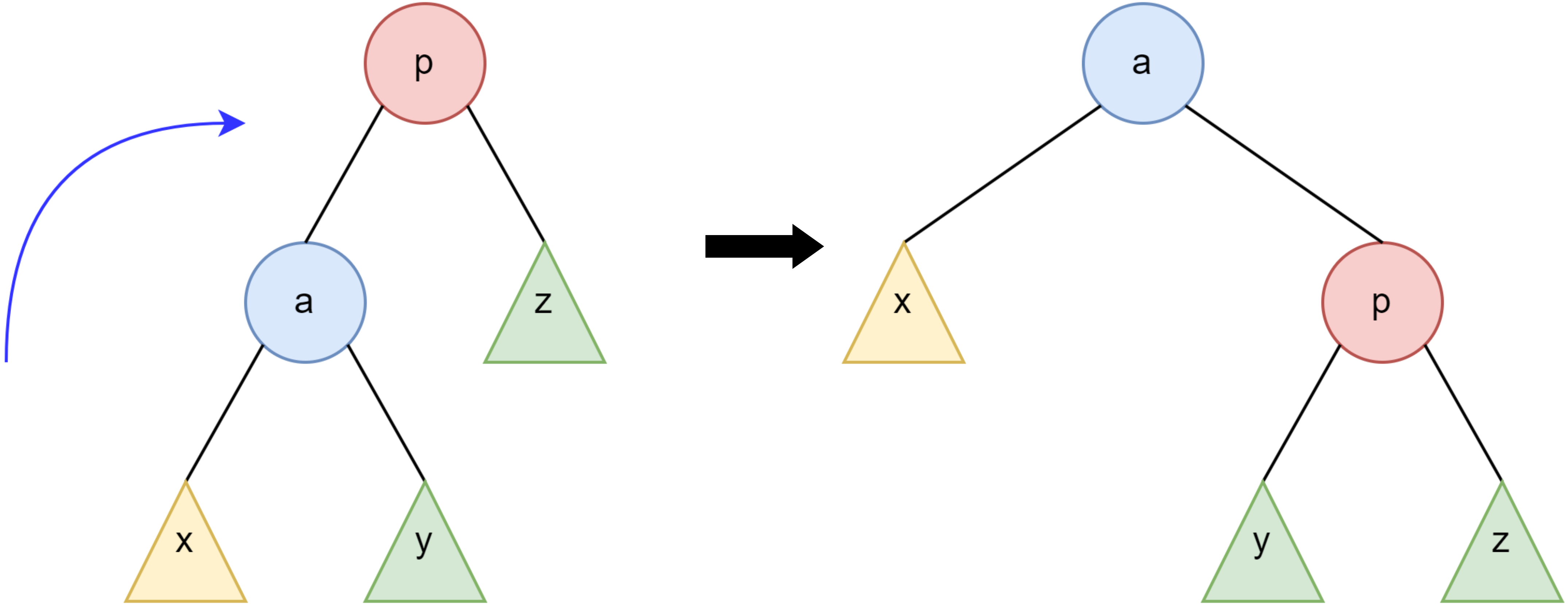
1. Insert a node at w: p becomes imbalanced
2. Next, rotate at p, using BST fact: w < b < x < p < z



Any Questions?

AVL: `insert`, Case 1 Left-Left

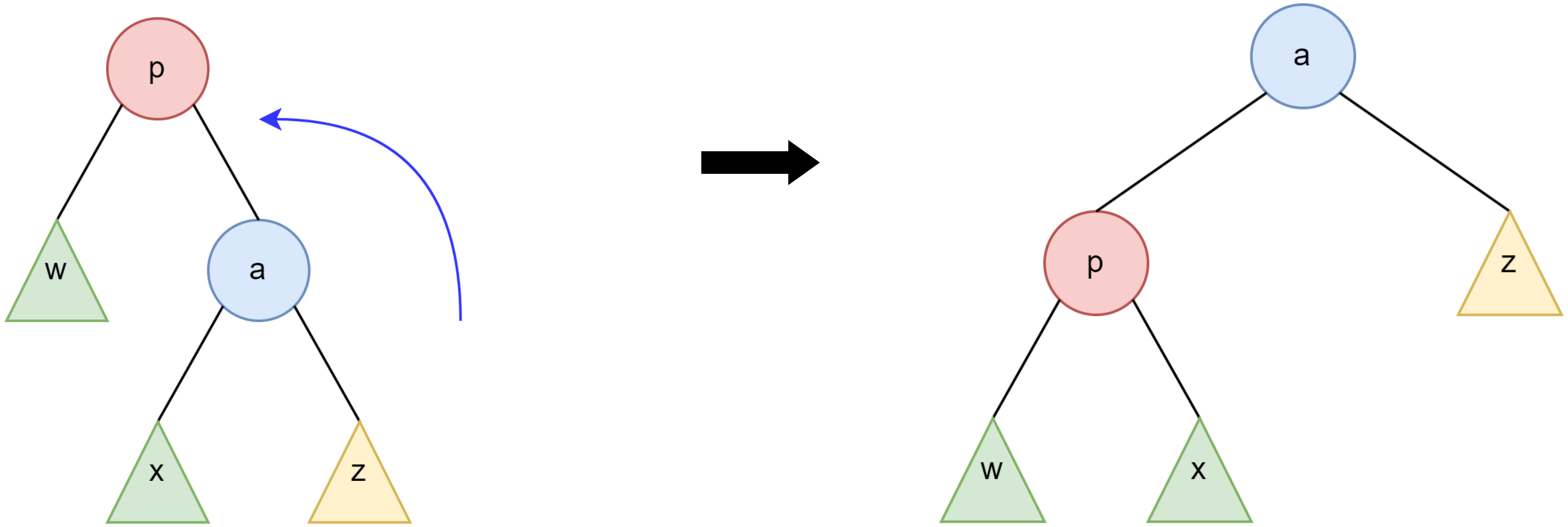
After `RotateWithRight`



AVL: `insert`, Case 4 Right-Right

- The same but mirrored

After `RotateWithLeft`

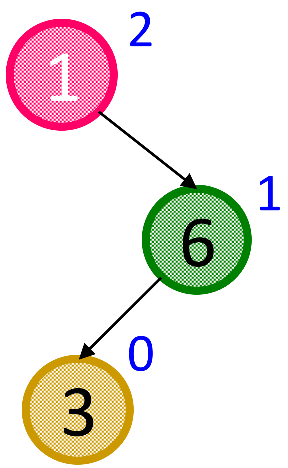


Today

- Recap: AVL Tree
- AVL Tree `insert`
  - General
  - Single Rotation
  - Double Rotation
- AVL Tree Conclusions

AVL: `insert` Case 3 Left-Right Example 1

1. `insert(1)`
2. `insert(6)`
3. `insert(3)` violates balance property



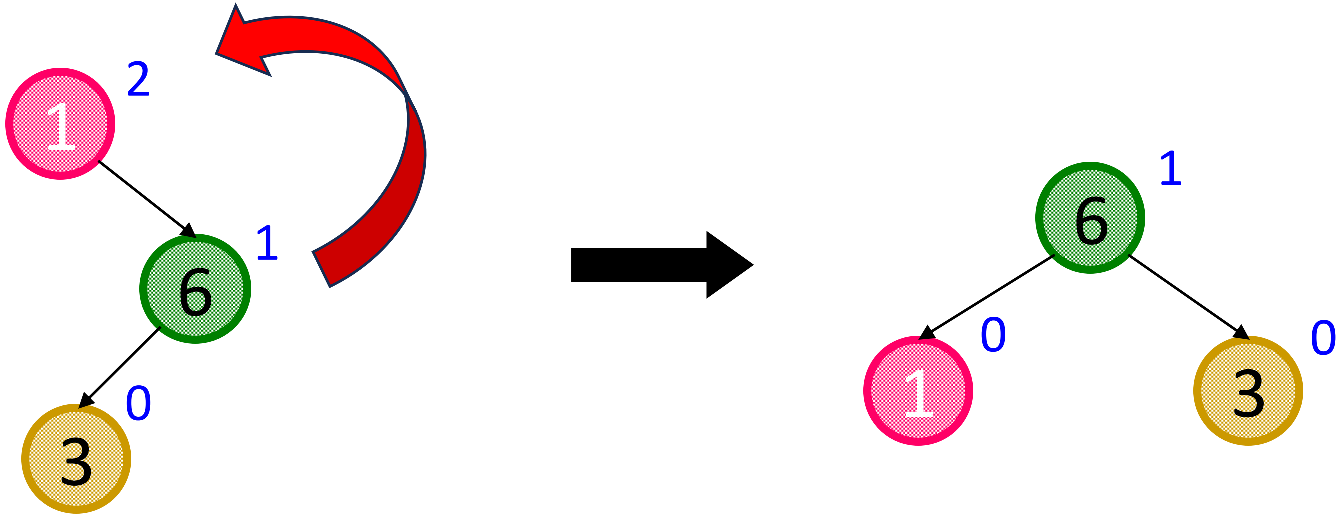
p Happens to be at the root

What is the only way to fix this?

Uhh try single rotation (?)

AVL: `insert` Left-Right Attempted Fix 1

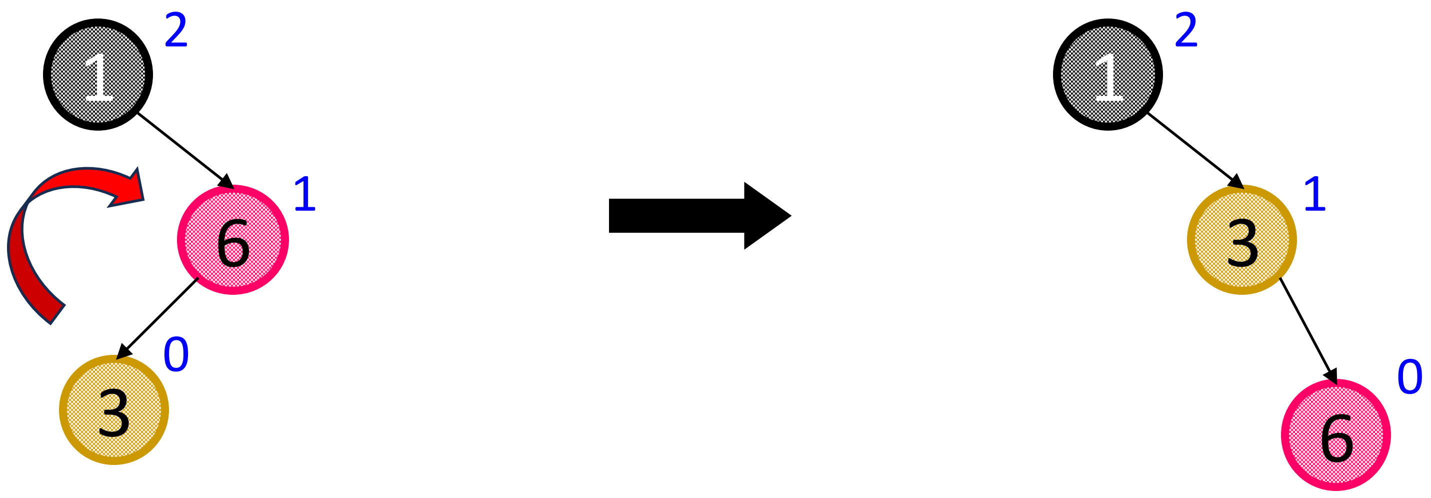
Try Single (Counter ClockWise) Rotation on 1



Is there a problem here? Order Property violated

AVL: `insert` Left-Right Attempted Fix 2

Try Single (ClockWise) Rotation on 1

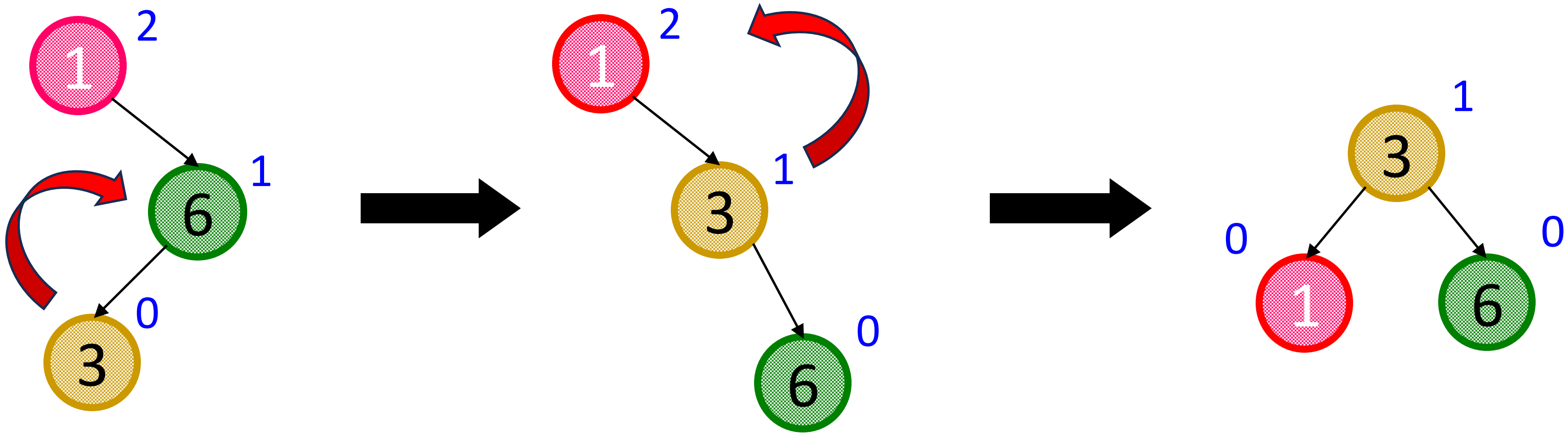


Is there a problem here? Balance not fixed!

AVL: `insert` Left-Right Real Fix

Attempted Fix 1: Order Property violated

Attempted Fix 2: Balance not fixed



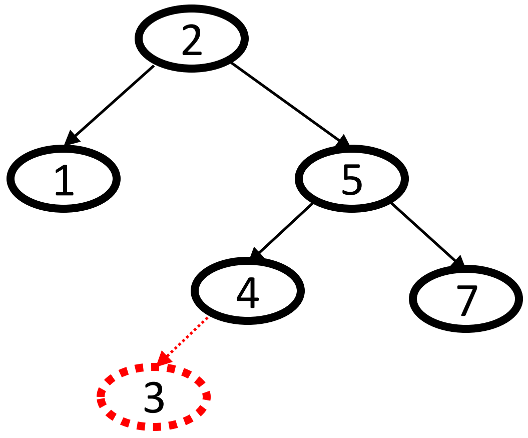
Real Fix: Double Rotation!

1. Rotate p's child and p's grandchild
2. Rotate p and p's new child

Any Questions?

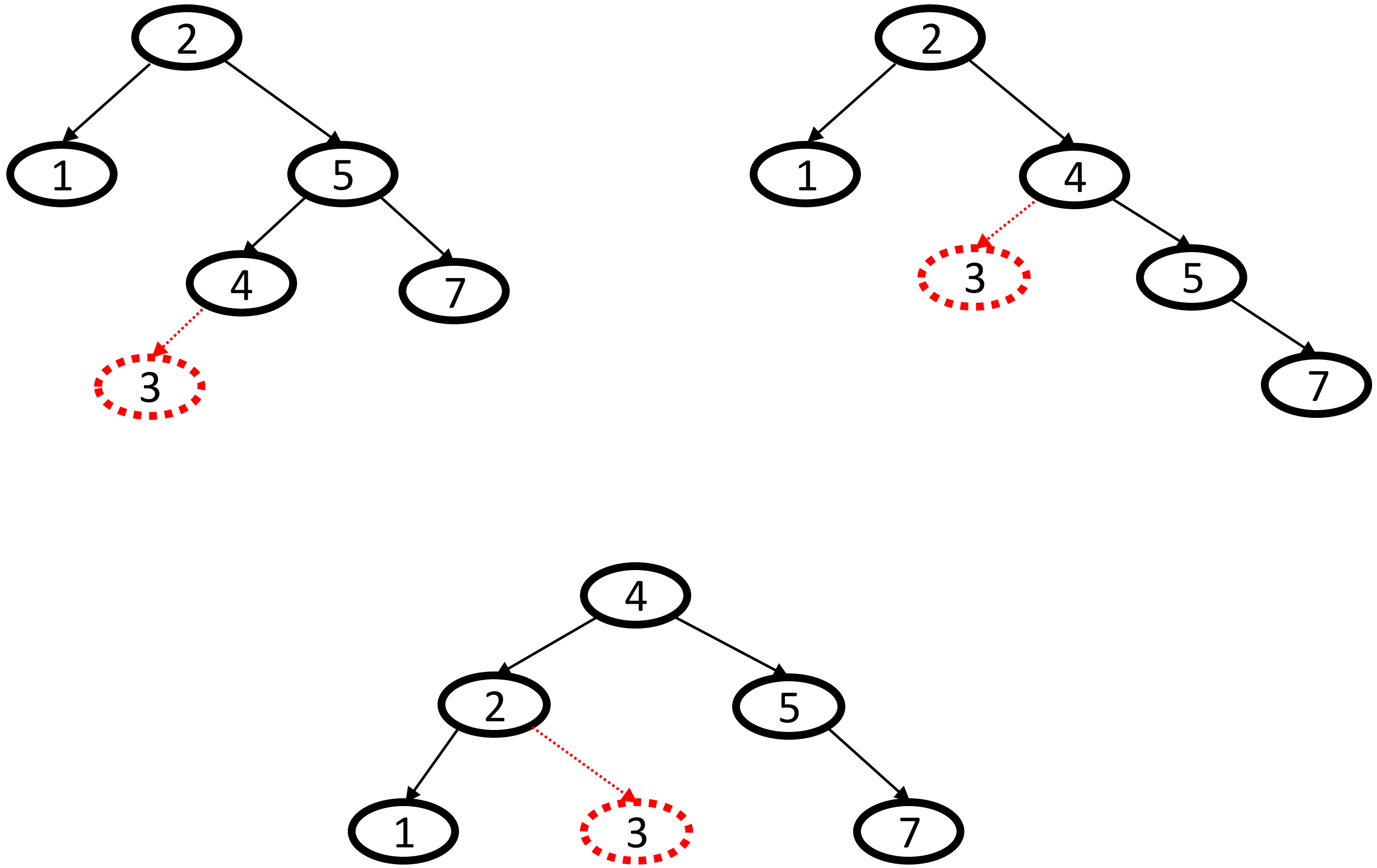
AVL: `insert` Case 3 Left-Right Example 2

`insert(3)`



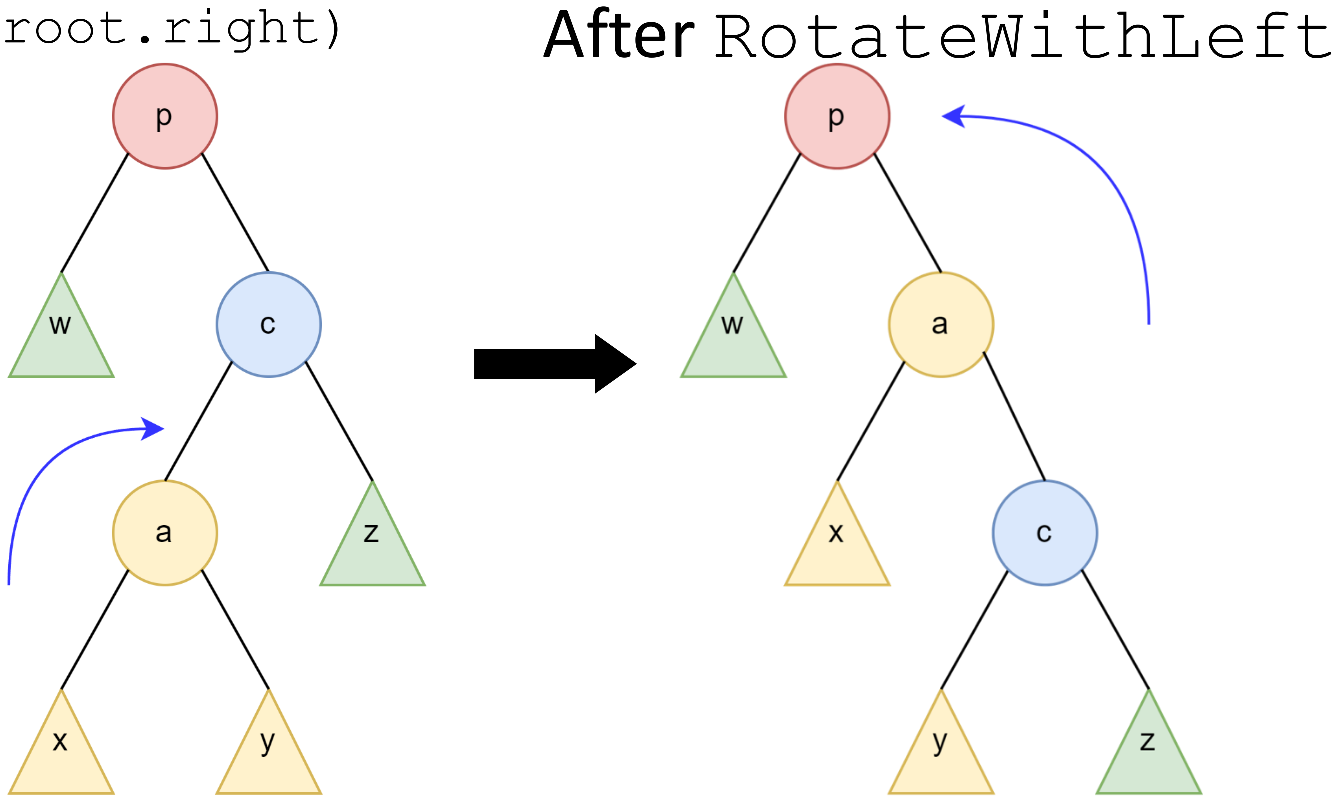
AVL: `insert` Case 3 Left-Right Example 2 (Soln.)

`insert(3)`



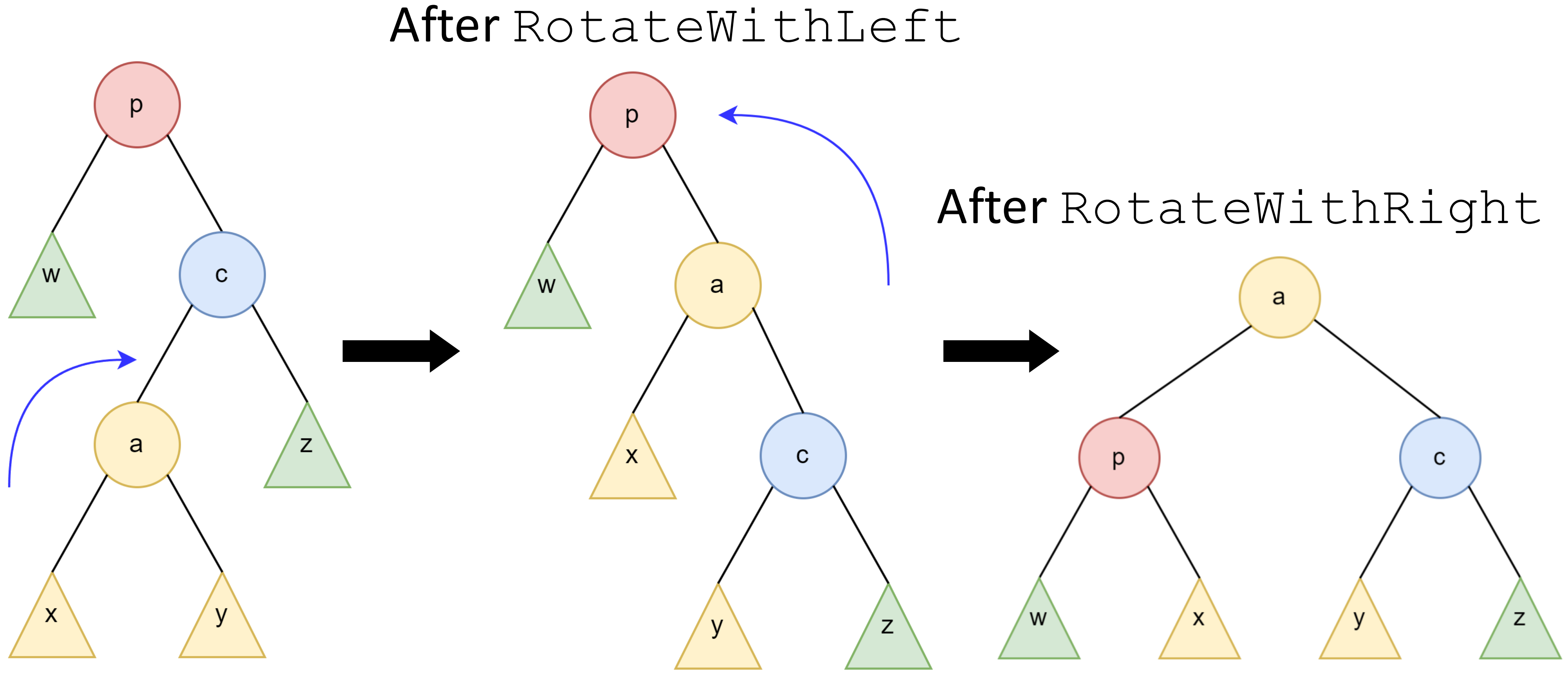
AVL: Double Rotation Pseudocode

```
Node DoubleRotateWithRight(Node root) {
  root.right = RotateWithLeft(root.right)
  root = RotateWithRight(root)
  return root
}
```



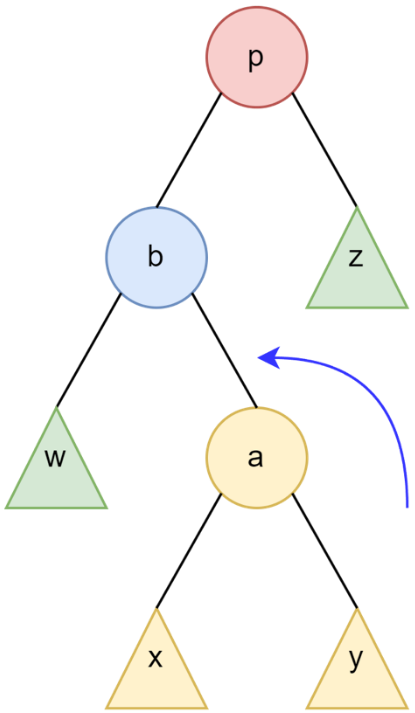
Any Questions?

AVL: `insert`, Case 3 Left-Right

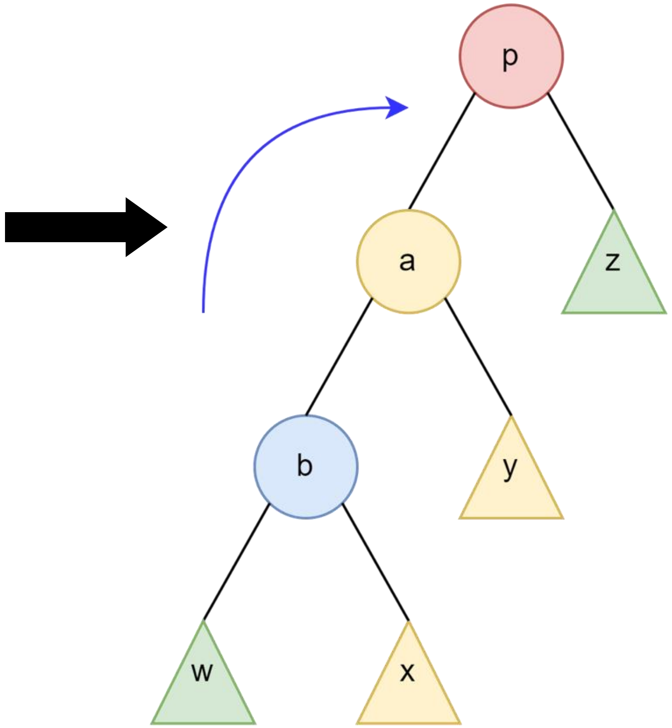


AVL: `insert`, Case 2 Right-Left

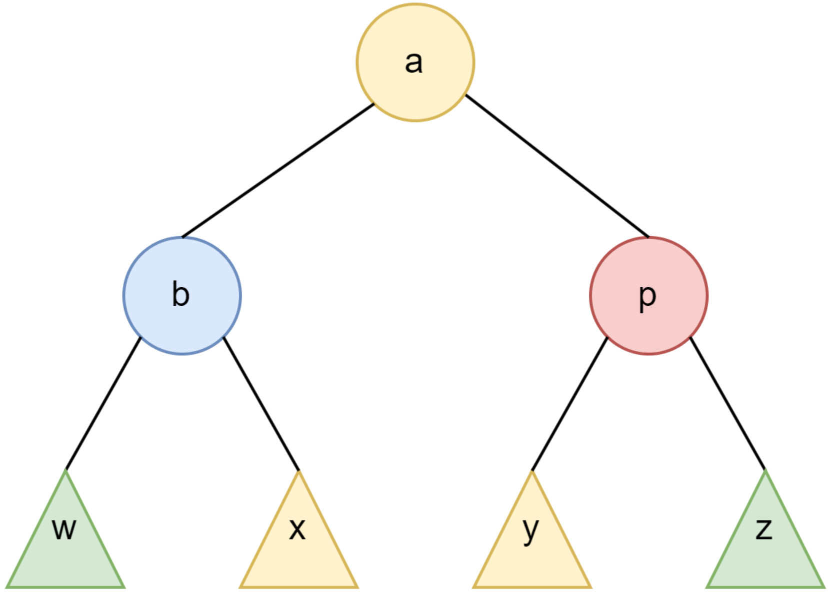
- The same but mirrored



After `RotateWithRight`



After `RotateWithLeft`



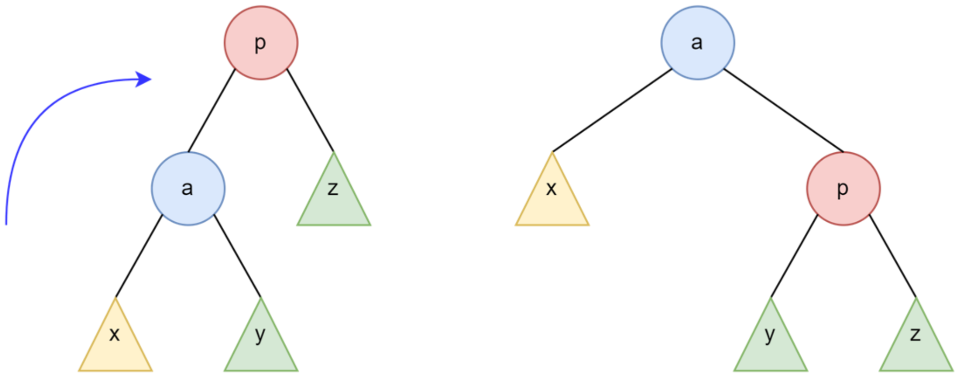
AVL: `insert`.

1. BST `insert`
2. Recursive Backtracking: Detect height imbalance
3. If imbalance: Find case + Rotate,
    1. Left-Left: left subtree of the left child of p.
    2. Right-Left: right subtree of the left child of p.
    3. Left-Right: left subtree of the right child of p.
    4. Right-Right: right subtree of the right child of p.

Assuming tree was balanced before insert (it is), only one case occurs

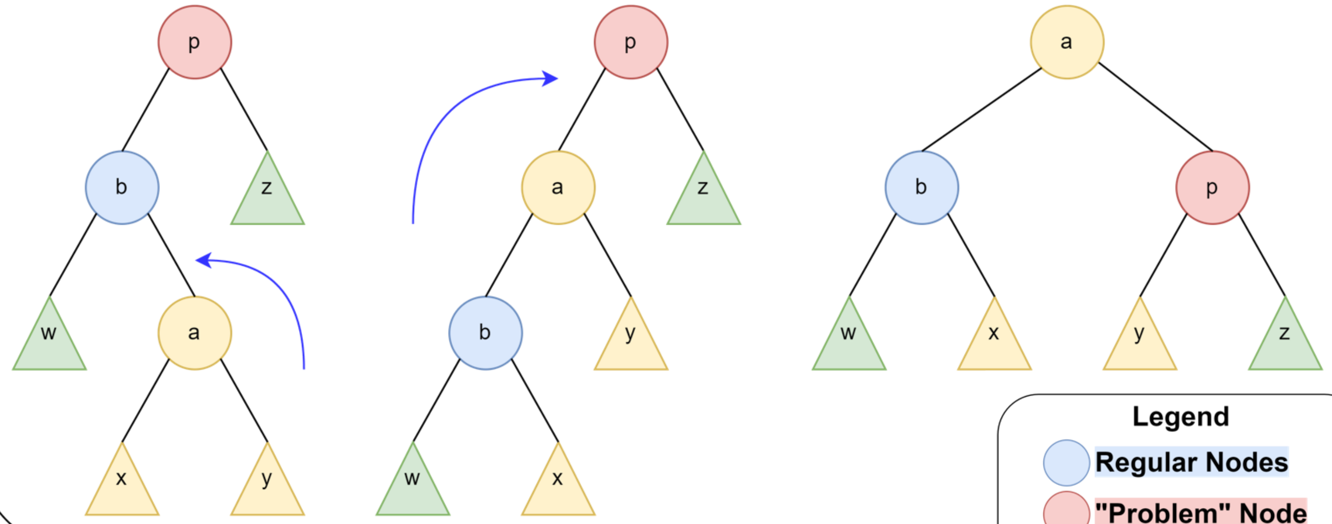
Case 1:Left-Left

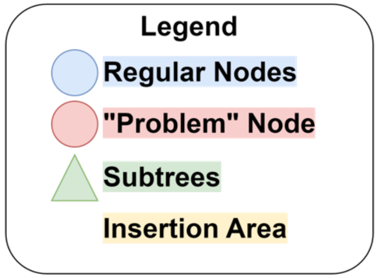
The insertion is in the Left subtree of the Left child of the problem node



Case 2:Right-Left

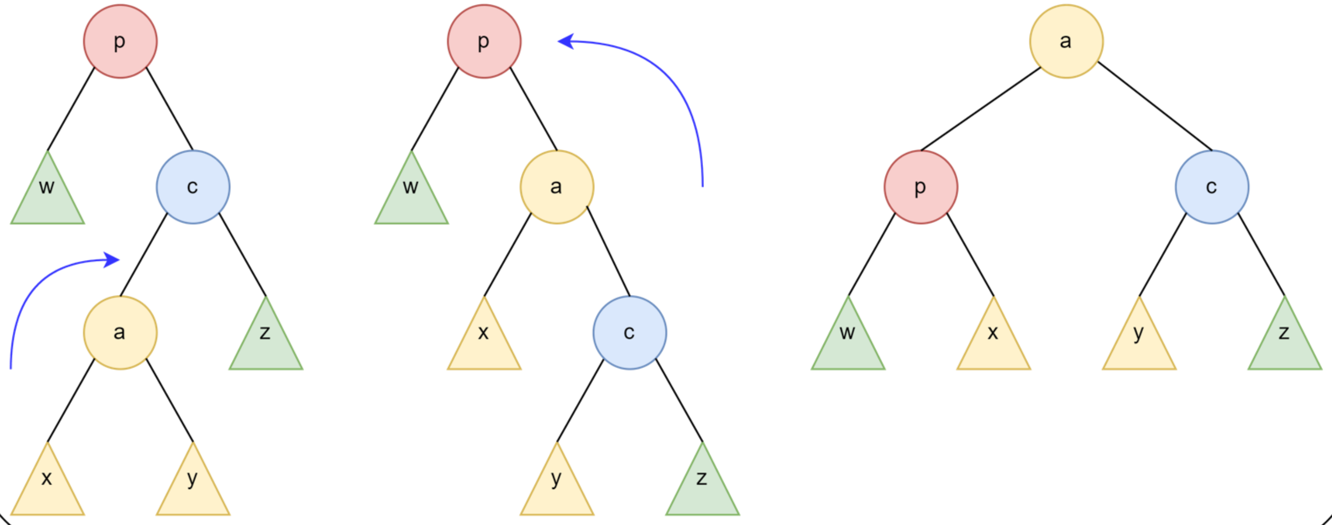
The insertion is in the Right subtree of the Left child of the problem node





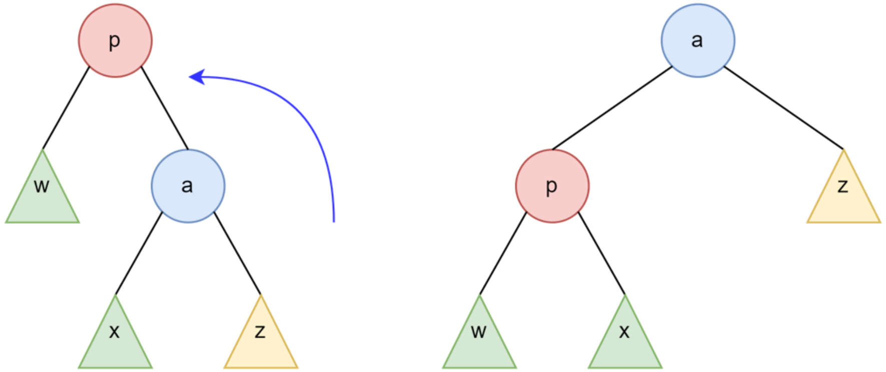
Case 3:Left-Right

The insertion is in the Left subtree of the Right child of the problem node



Case 4:Right-Right

The insertion is in the Right subtree of the Right child of the problem node



Any Questions?

AVL: `insert` Exercise: `1 2 5 3 4`

Today

- Recap: AVL Tree
- AVL Tree `insert`
  - General
  - Single Rotation
  - Double Rotation
- AVL Tree Conclusions

Let $S(h)$ be the minimum # of nodes in an AVL tree of height h, then:

$$S(h) = \begin{cases} 0 & if\ h = -1 \\ 1 & if\ h = 0 \\ 1 + S(h-1) + S(h-2) & otherwise \end{cases}$$

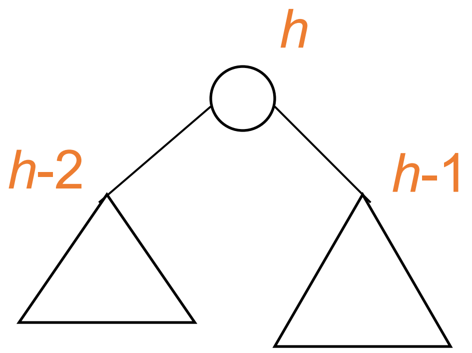
h	**Minimal AVL Tree**	**S(h)**

AVL: The shallowness bound

Let $S(h)$ = the minimum number of nodes in an AVL tree of height h

- If we can prove that $S(h)$ grows exponentially in h, then a tree with n nodes has a logarithmic height

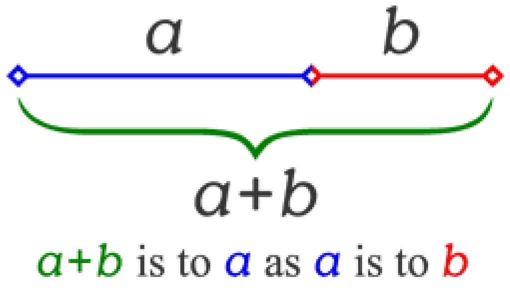
- Step 1: Define $S(h)$ inductively using AVL property



- Step 2: Show this recurrence grows really fast
  - Similar to Fibonacci numbers
  - Can prove for all h, $S(h) > \phi^h - 1$

    Golden ratio $\phi = \frac{1+\sqrt{5}}{2} \approx 1.62$
  - Growing faster than 1.62^h is "plenty exponential"

AVL: The Golden Ratio

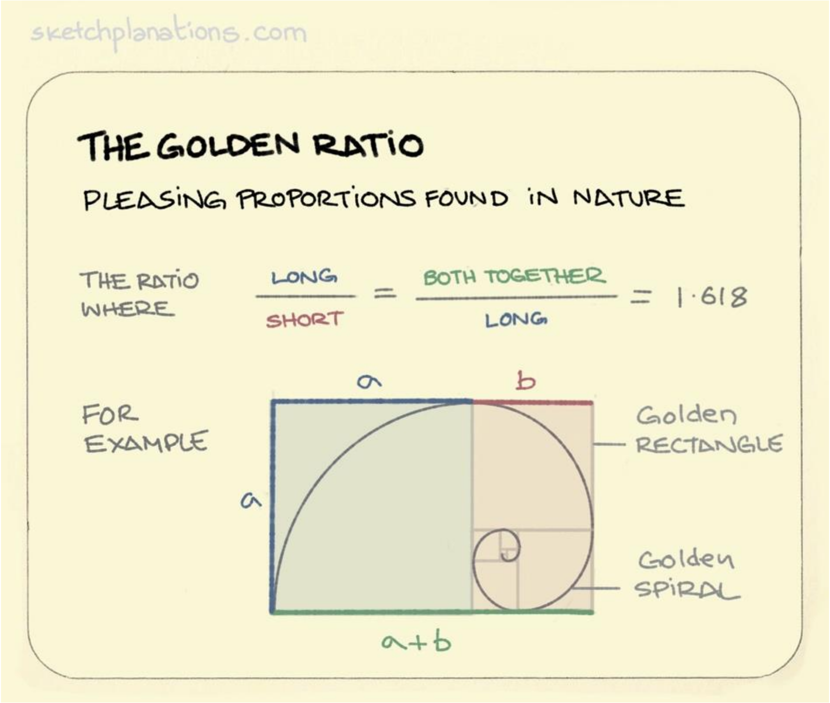


This is a special number $\phi = \frac{1+\sqrt{5}}{2} \approx 1.62$

- Aside: Since the Renaissance, many artists and architects have proportioned their work (e.g., length:height) to approximate the *golden ratio*: If $(a+b)/a = a/b$, then $a = \phi b$

- We will need one special arithmetic fact about ϕ :

$$\begin{aligned}
\phi^2 &= ((1+5^{1/2})/2)^2 \\
&= (1 + 2*5^{1/2} + 5)/4 \\
&= (6 + 2*5^{1/2})/4 \\
&= (3 + 5^{1/2})/2 \\
&= 1 + (1 + 5^{1/2})/2 \\
&= 1 + \phi
\end{aligned}$$



AVL: Height Proof

$$S(h) = \begin{cases} 0 & if\ h = -1 \\ 1 & if\ h = 0 \\ 1 + S(h-1) + S(h-2) & otherwise \end{cases}$$

Theorem: For all $h \geq 0$, $S(h) > \phi^h - 1$

Proof: By induction on h

Base cases:

$S(0) = 1 > \phi^0 - 1 = 0$ $\quad$ $S(1) = 2 > \phi^1 - 1 \approx 0.62$

Inductive case ($k > 1$):

Show $S(k+1) > \phi^{k+1} - 1$ assuming $S(k) > \phi^k - 1$ and $S(k-1) > \phi^{k-1} - 1$

$S(k+1) = 1 + S(k) + S(k-1)$	by definition of S
$> 1 + \phi^k - 1 + \phi^{k-1} - 1$	by induction
$= \phi^k + \phi^{k-1} - 1$	by arithmetic (1-1=0)
$= \phi^{k-1}(\phi + 1) - 1$	by arithmetic (factor ϕ^{k-1})
$= \phi^{k-1}\phi^2 - 1$	by special property of ϕ
$= \phi^{k+1} - 1$	by arithmetic (add exponents)

AVL: Height

TL;DR Last few slides show:

$$h \in \Theta(\log n)$$

AVL: Efficiency?

- `find`: $\Theta(\_\_\_\_\_)$
  - Tree is balanced
- `insert`: $\Theta(\_\_\_\_\_)$
  - Tree starts balanced
  - Rotation is $\Theta(1)$, Root->Deepest Descendant: $\Theta(\log n)$
  - Tree ends balanced
- `buildTree`: $\Theta(\_\_\_\_\_)$
- `delete`
  - Lazy Deletion: $\Theta(\_\_\_\_\_\_)$
  - Non-lazy Deletion: $\Theta(\_\_\_\_\_\_)$

AVL: Efficiency? (Soln.)

- `find`: $\Theta(\log n)$
  - Tree is balanced
- `insert`: $\Theta(\log n)$
  - Tree starts balanced
  - Rotation is $\Theta(1)$, Root->Deepest Descendant: $\Theta(\log n)$
  - Tree ends balanced
- `buildTree`: $\Theta(n \log n)$
- `delete`
  - Lazy Deletion: $\Theta(\log n)$
  - Non-lazy Deletion: $\Theta(\log n)$

AVL: Tradeoffs

Pros:

1. All operations logarithmic worst-case because trees are always balanced
2. Height balancing adds no more than a constant factor to the speed of `insert` and `delete`

Cons:

1. Difficult to program & debug
2. More space for height field
3. Asymptotically faster but rebalancing takes a little time
4. Most large searches are done in database-like systems on disk and use other structures (e.g., B-trees)

Any Questions?

Timeline

- AVL Tree
  - Basics, Properties, Operations
- AVL Tree `insert`
  - Single Rotation
  - Double Rotation
- AVL Tree Conclusions
- Hashing
  - Hash Function
  - ChainingHashTable