AVL Trees

Data Structures and Parallelism

Announcements

Exercise 1 grades on gradescope.

Regrade policy

Exercises: Submit a regrade request via gradescope

If the initial regrade isn't sufficient, send email to Robbie

- Include a written explanation of why you think the grading isn't correct.
- (especially on autograded section of projects) if you can explain a mistake you made and why it's not worth as many points as it cost you, we'll consider adjusting the rubric.
- tl;dr – we're humans, talk to us.

Project 1 – let us know if you want to use late day(s).

More Announcements

Exercises 2,3 due today.

Exercise 4 is out: make sure you have “version 2”

- Ordering of parts changed from version 1 to version 2.
- Due Wednesday at noon.

Project 2 out this weekend.

If you haven’t told us your partner, tell us ASAP.

We’ll be pairing unpaired people today.

Old midterms to study from will go up over the weekend.

- In the meantime, there are some old midterms on 18wi’s site.

Outline

Last time:

Dictionaries are **extremely** common data structures

Main operations:

- Find
- Insert
- Delete

BSTs are good in the average case, but bad in the worst case.

Today:

How to adapt BSTs so you never get the worst case.

Avoiding the Worst Case

An AVL tree is a binary search tree that also meets the following rule

AVL condition: For every node, the height of its left subtree and right subtree differ by at most 1.

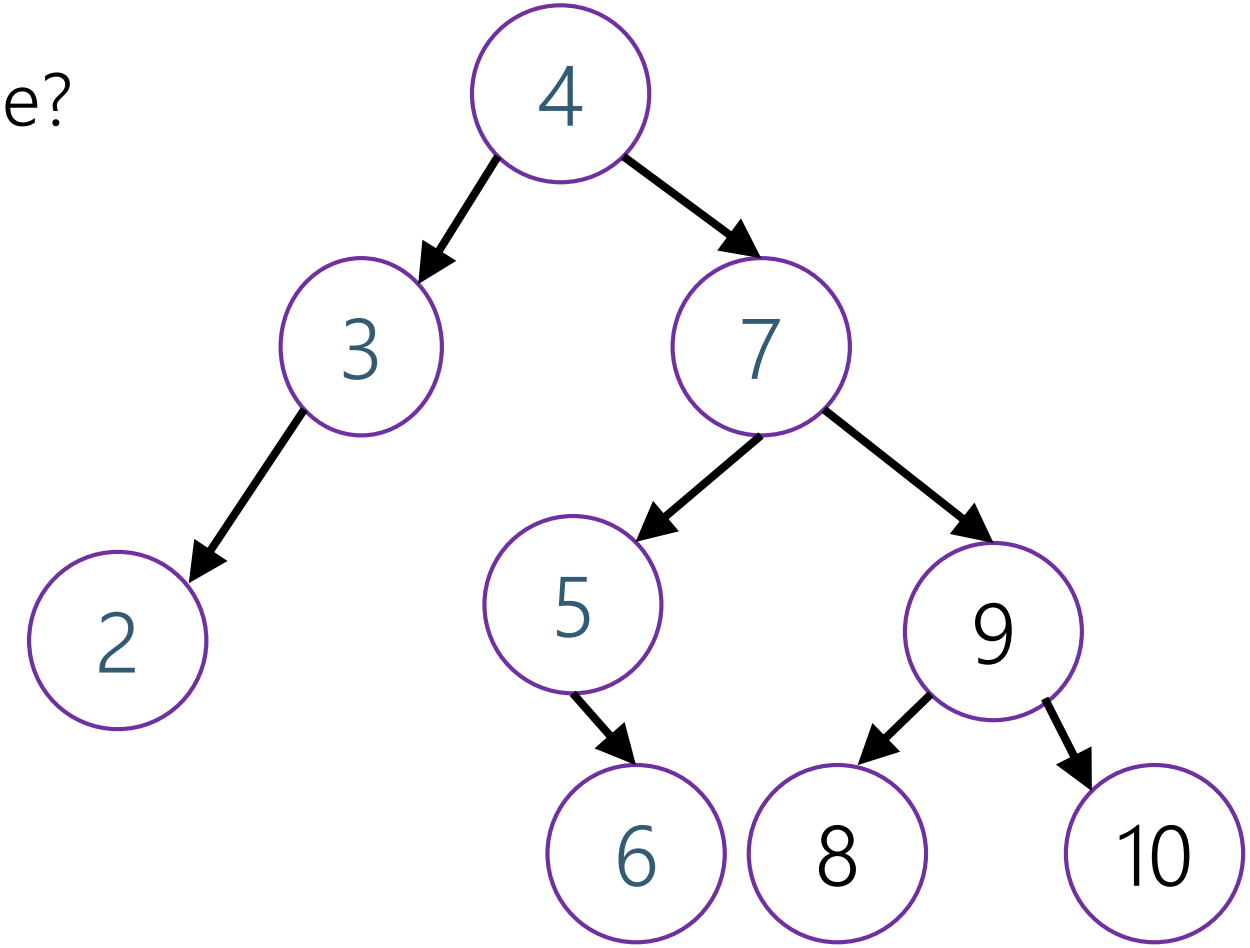
This will avoid the worst case! We have to check:

1. We must be able to maintain this property when inserting/deleting
2. Such a tree must have height $O(\log n)$.

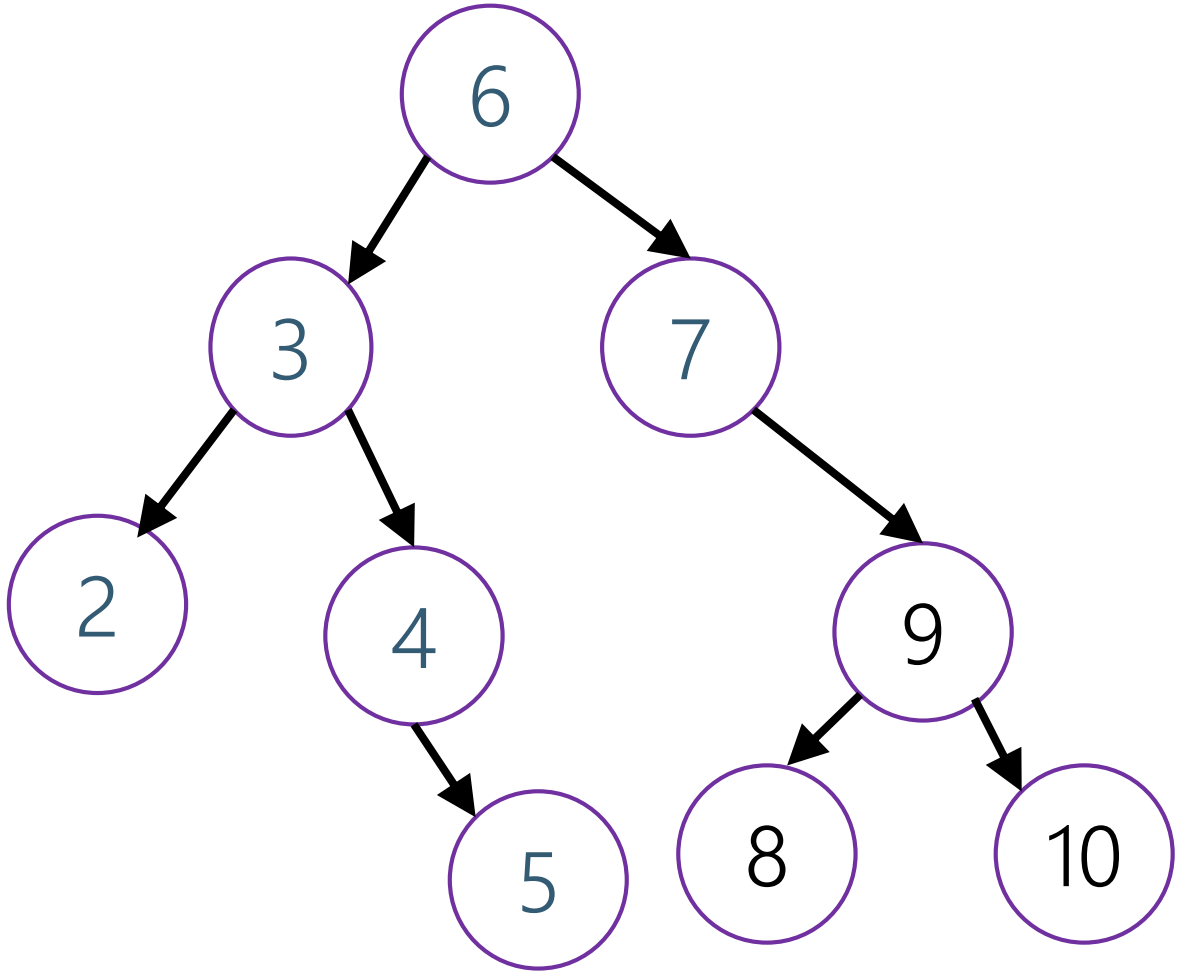
Warm-Up

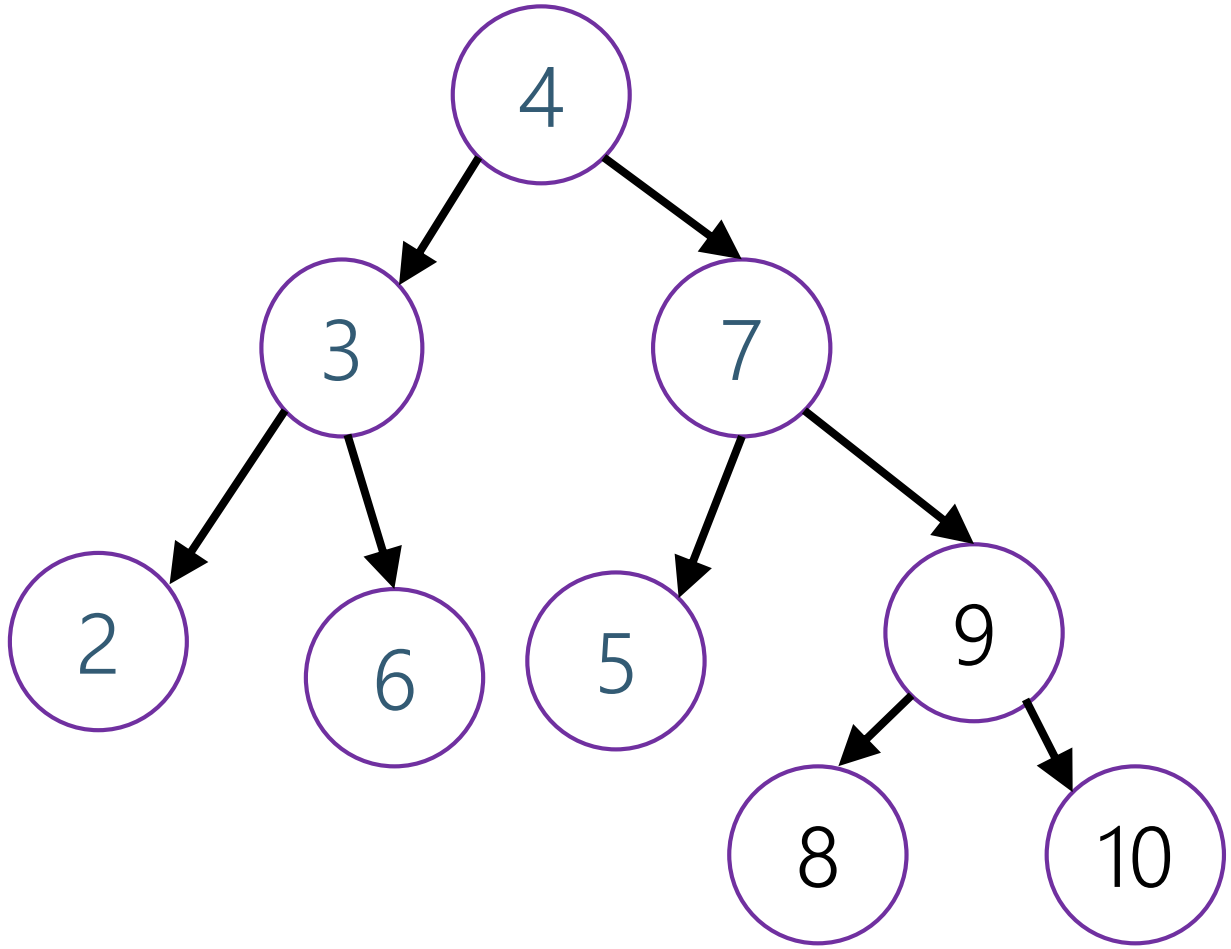
AVL condition: For every node, the height of its left subtree and right subtree differ by at most 1.

Is this a valid AVL tree?



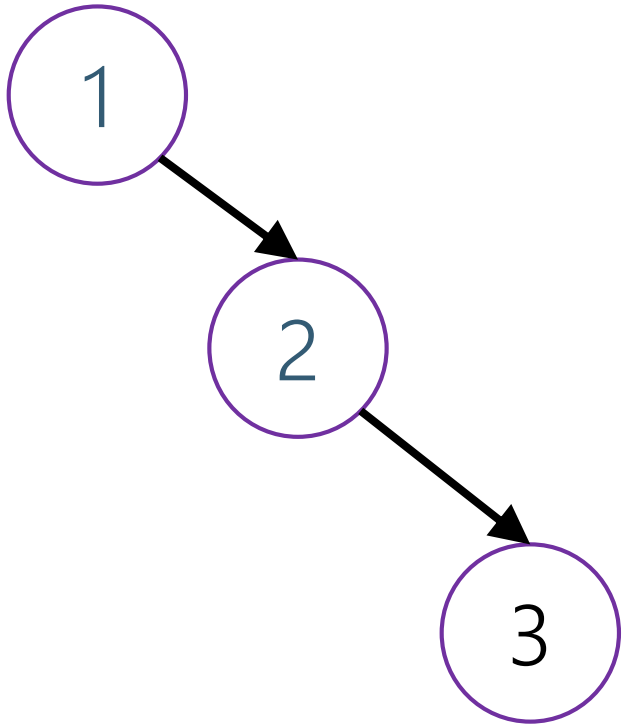
Are These AVL Trees?

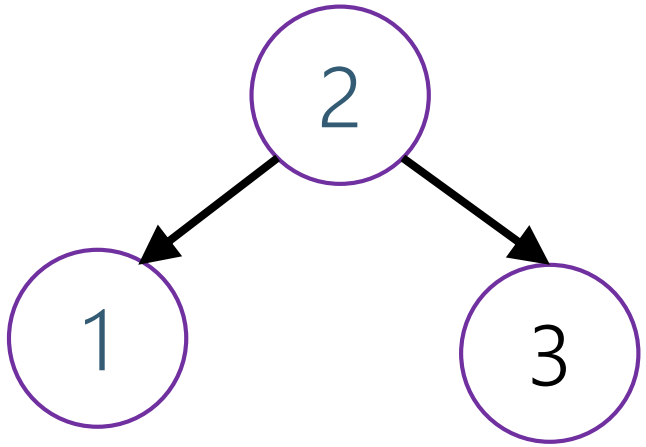




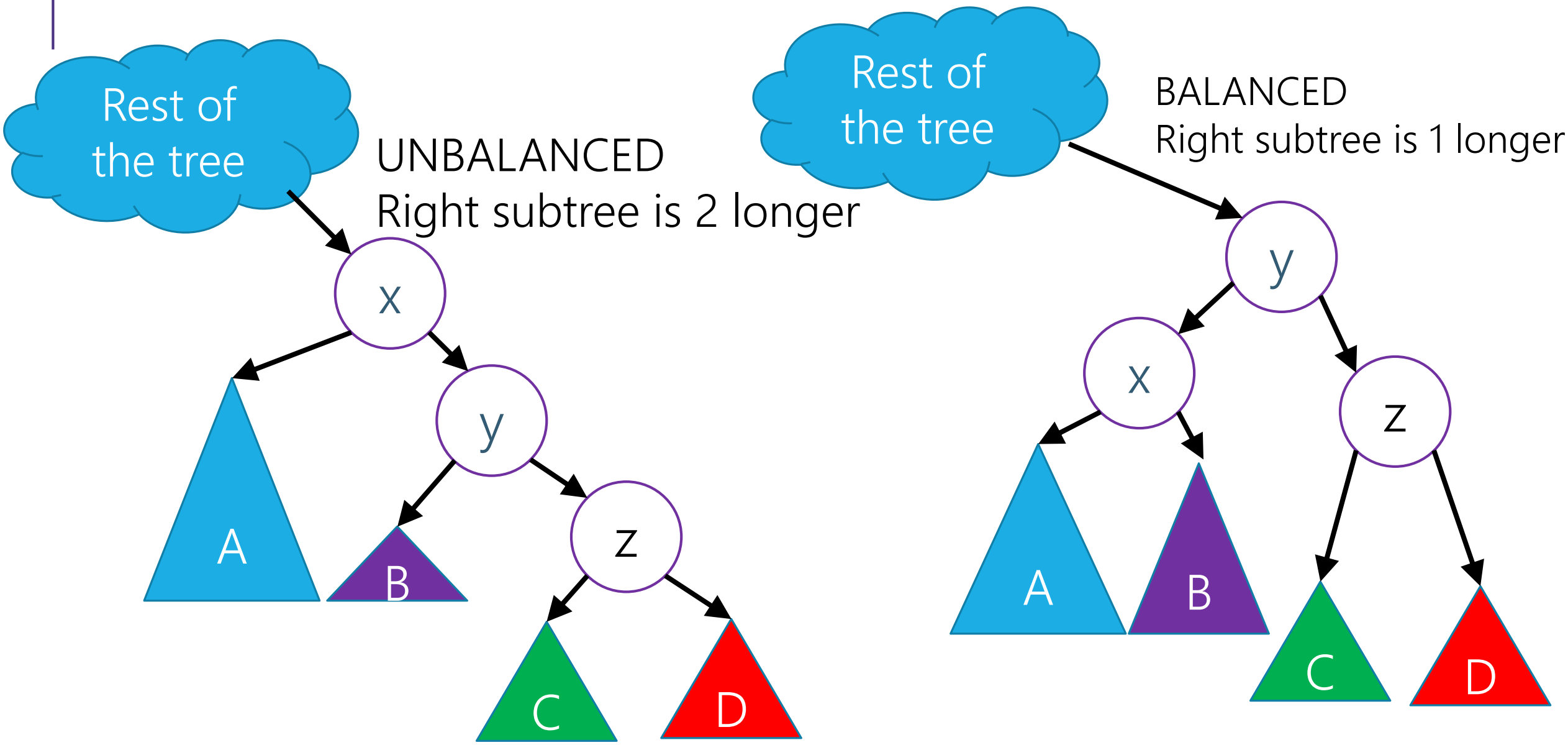
Insertion

What happens if when we do an insertion, we break the AVL condition?



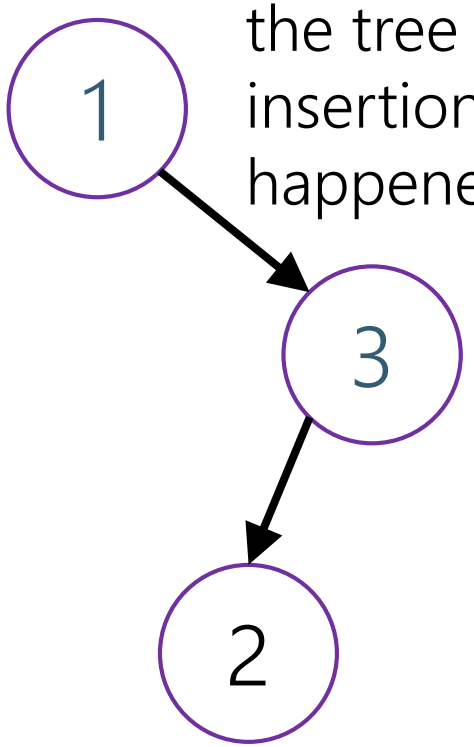


Left Rotation



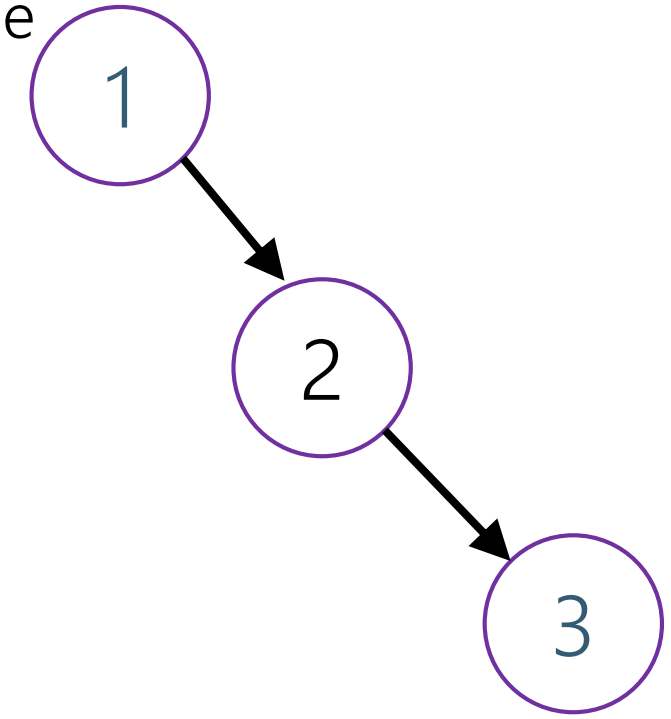
It Gets More Complicated

There's a "kink" in the tree where the insertion happened.

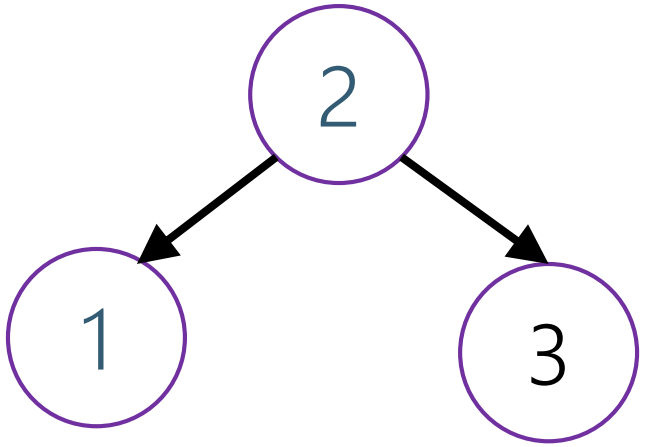


Can't do a left rotation

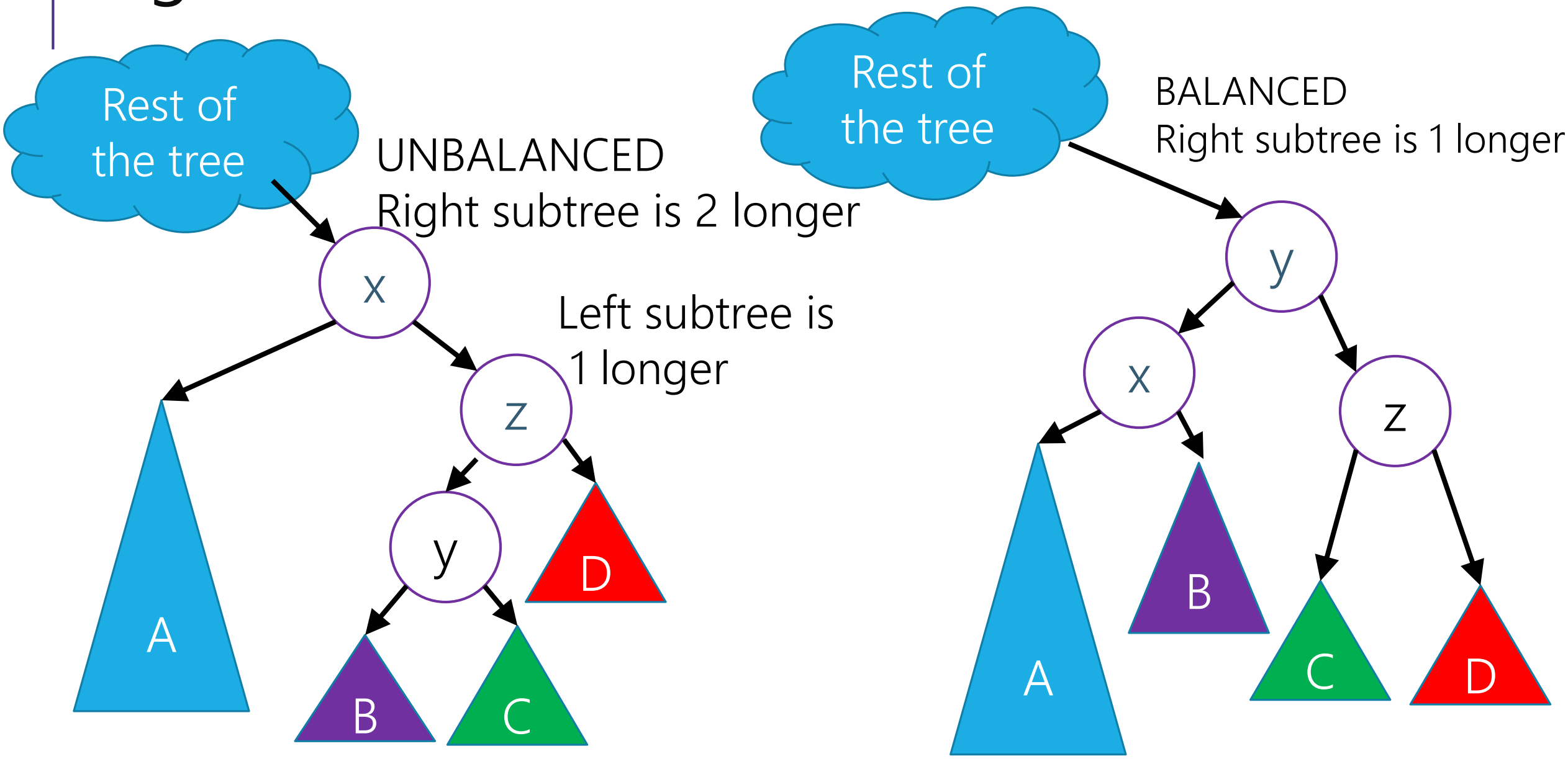
Do a "right" rotation around 3 first.



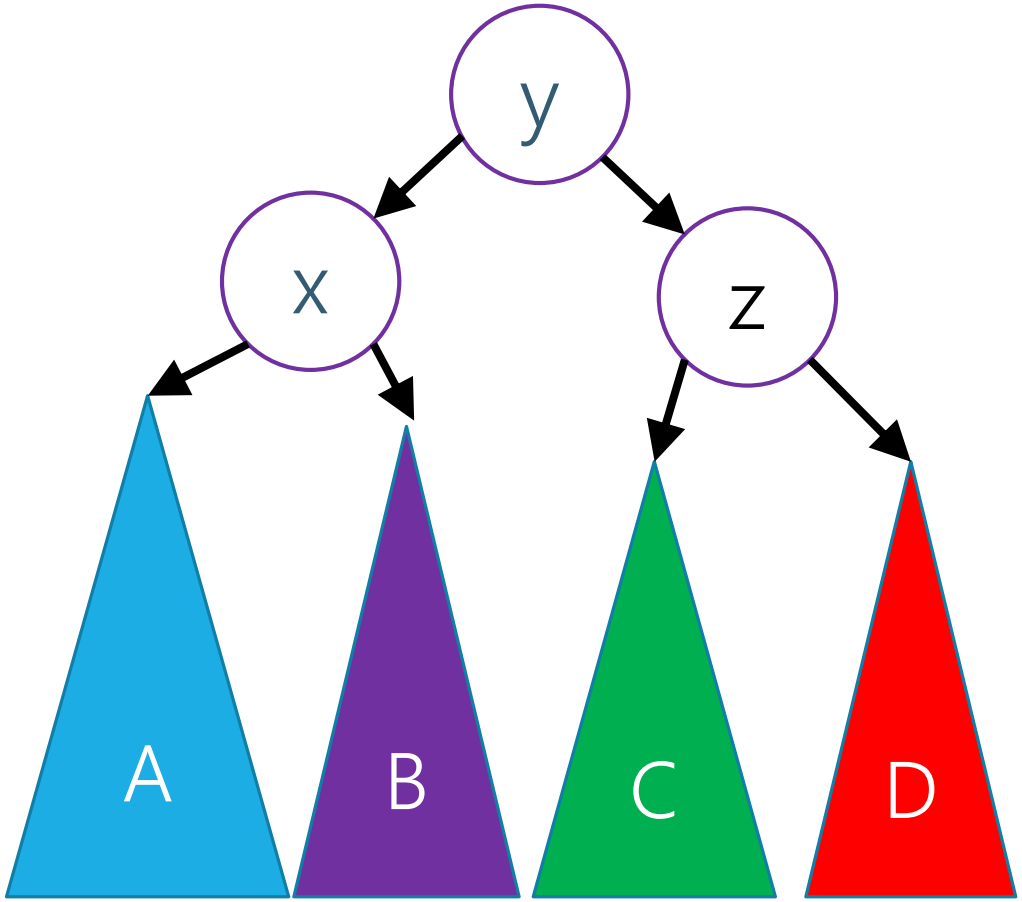
Now do a left rotation.



Right Left Rotation

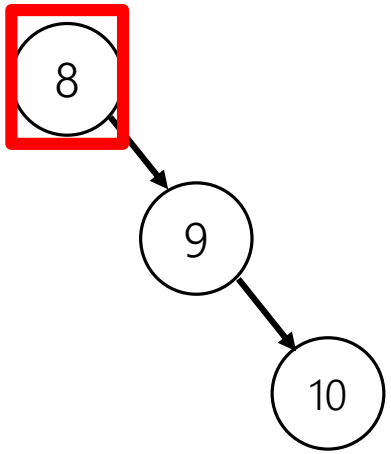


Four Types of Rotations

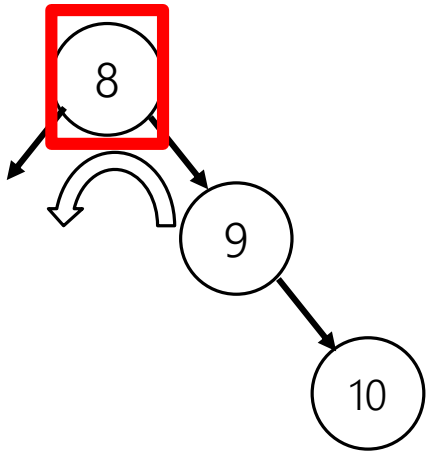


Insert location	Solution
Left subtree of left child (A)	Single right rotation
Right subtree of left child (B)	Double (left-right) rotation
Left subtree of right child (C)	Double (right-left) rotation
Right subtree of right child(D)	Single left rotation

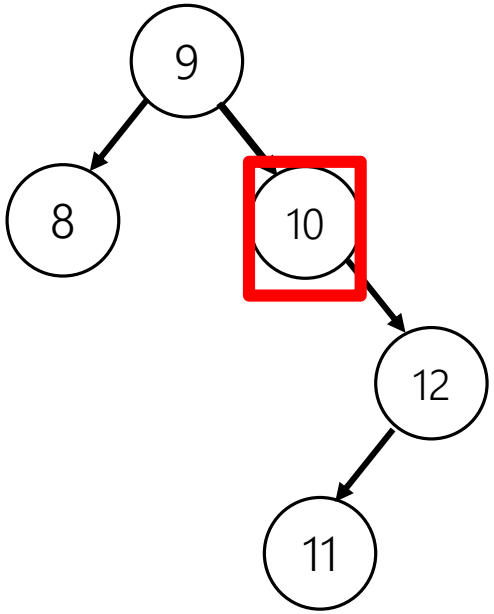
AVL Example: 8,9,10,12,11



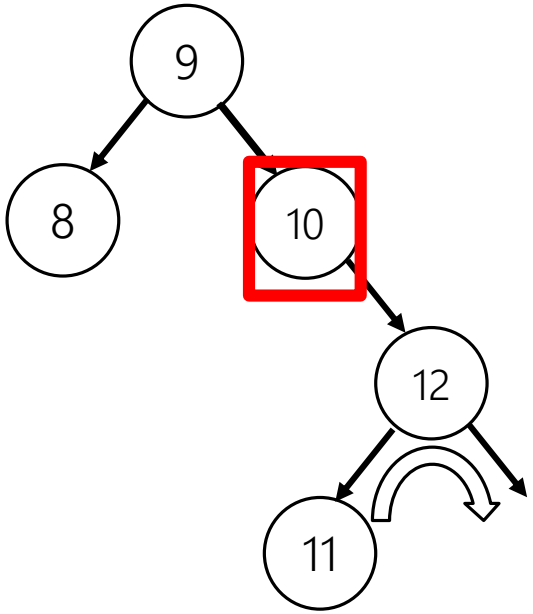
AVL Example: 8,9,10,12,11



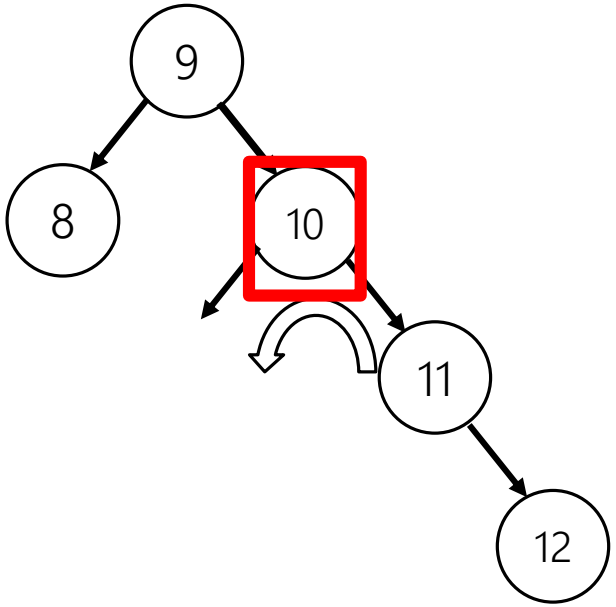
AVL Example: 8,9,10,12,11



AVL Example: 8,9,10,12,11



AVL Example: 8,9,10,12,11



How Long Does Rebalancing Take?

Assume we store in each node the height of its subtree.

How do we find an unbalanced node?

How many rotations might we have to do?

How Long Does Rebalancing Take?

Assume we store in each node the height of its subtree.

How do we find an unbalanced node?

- Just go back up the tree from where we inserted.

How many rotations might we have to do?

- Just a single or double rotation on the lowest unbalanced node.
- A rotation will cause the subtree rooted where the rotation happens to have the same height it had before insertion.

Deletion

In Project 2: Just do lazy deletion!

Alternatively: a similar set of rotations is possible to rebalance after a deletion.

The textbook (or Wikipedia) can tell you more. You can implement these yourself in “Above and Beyond”

Aside: Traversals

What if, to save space, we didn't store heights of subtrees.

How could we calculate from scratch?

We could use a "traversal"

```
int height(Node curr){
    if(curr==null) return -1;
    int h = Math.max(height(curr.left),height(curr.right));
    return h+1;
}
```

Three Kinds of Traversals

```
InOrder(Node curr){
    InOrder(curr.left);
    doSomething(curr);
    InOrder(curr.right);
}
```

```
PreOrder(Node curr){
    doSomething(curr);
    PreOrder(curr.left);
    PreOrder(curr.right);
}
```

```
PostOrder(Node curr){
    PostOrder(curr.left);
    PostOrder(curr.right);
    doSomething(curr);
}
```

Traversals

If we have n elements, how long does it take to calculate height?

$\Theta(n)$ time.

The recursion tree (from the tree method) IS the AVL tree!

We do a constant number of operations at each node

In general, traversals take $\Theta(n \cdot f(n))$ time,

where `doSomething()` takes $\Theta\big(f(n)\big)$ time.

Where Were We?

We used rotations to restore the AVL property after insertion.

If h is the height of an AVL tree:

It takes $\mathbf{O}(h)$ time to find an imbalance (if any) and fix it.

So the worst case running time of insert? $\mathbf{\Theta}(h)$.

Deletion? With lazy deletion just the time to find, i.e. $\mathbf{\Theta}(h)$.

Is h always $O(\log n)$? YES! These are all $\mathbf{\Theta}(\log n)$. Let's prove it!

Bounding the Height

Suppose you have a tree of height h, meeting the AVL condition.

AVL condition: For every node, the height of its left subtree and right subtree differ by at most 1.

What is the minimum number of nodes in the tree?

If $h = 0$, then 1 node

If $h = 1$, then 2 nodes.

In general?

Bounding the Height

In general, let $N()$ be the minimum number of nodes in a tree of height h, meeting the AVL requirement.

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

Bounding the Height

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

We can try unrolling or recursion trees.

Let’s try unrolling

$$N(h) = N(h-1) + N(h-2) + 1$$

$$= N(h-2) + N(h-3) + 1 + N(h-2) + 1$$

$$= 2N(h-2) + N(h-3) + 1 + 1$$

$$= 2(N(h-3) + N(h-4) + 1) + N(h-3) + 1 + 1$$

$$= 3N(h-3) + 2N(h-4) + 2 + 1 + 1$$

$$= 3(N(h-4) + N(h-5) + 1) + 2N(h-4) + 2 + 1 + 1$$

$$= 5N(h-4) + 3N(h-5) + 3 + 2 + 1 + 1$$

$$= 5(N(h-5) + N(h-6) + 1) + 3N(h-5) + 3 + 2 + 1 + 1$$

$$= 5N(h-6) + 8N(h-5) + 5 + 3 + 2 + 1 + 1$$

…

Bounding the Height

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

When unrolling we'll quickly realize:

- Something with Fibonacci numbers is going on.
- It's going to be hard to exactly describe the pattern.

The real solution (using deep math magic beyond this course) is

$N(h) \geq \phi^h - 1$ where ϕ is $\frac{1+\sqrt{5}}{2} \approx 1.62$

The Proof

To convince you that the recurrence solution is correct, I don't need to tell you where it came from.

I just need to prove it correct via induction.

We'll need this fact: $\phi + 1 = \phi^2$

It's easy to check by just evaluating $\left(\frac{1+\sqrt{5}}{2}\right)^2$

The Proof

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

$$\phi + 1 = \phi^2$$

Base Cases: $\phi^0 - 1 = 0 < 1 = N(0) \quad \phi^1 - 1 = \phi - 1 \approx 0.62 < 2 = N(1)$

Inductive Step

Inductive Hypothesis: Suppose that $N(h) > \phi^h - 1$ for $h < k$.

Inductive Step: We show $N(k) > \phi^k - 1$.

$N(k) = N(k-1) + N(k-2) + 1$ definition of $N()$

$> \phi^{k-1} - 1 + \phi^{k-2} - 1 + 1$ by IH (note we need a strong hypothesis here)

$= \phi^{k-1} + \phi^{k-2} - 1$ algebra

$= \phi^{k-2}(\phi + 1) - 1$

$= \phi^{k-2}(\phi^2) - 1$ fact from last slide

$= \phi^{k+1} - 1$

What's the point?

The number of nodes in an AVL tree of height h is always at least $\phi^h - 1$

So in an AVL tree with n elements, the height is always at most $\log_\phi(n+1)$

In big-O terms, that's enough to say the number of nodes is $O(\log n)$.

So our AVL trees really do have $O(\log n)$ worst cases for insert, find, and delete!

Wrap Up

AVL Trees:

$O(\log n)$ worst case `find`, `insert`, and `delete`.

Pros:

Much more reliable running times than regular BSTs.

Cons:

Tricky to implement

A little more space to store subtree heights

Other Dictionaries

There are lots of flavors of self-balancing search trees

"Red-black trees" work on a similar principle to AVL trees.

"Splay trees"

- Get $O(\log n)$ amortized bounds for all operations.

"Scapegoat trees"

"Treaps" – a BST and heap in one (!)

Similar tradeoffs to AVL trees.

Next week: A completely different idea for a dictionary

Goal: $O(1)$ operations on average, in exchange for $O(n)$ worst case.