CSCB63 Winter 2021

Week 2 - Balanced Trees

Anna Bretscher

January 18, 2021

Why Balanced Trees?

- What is the worst case complexity of `insert`, `delete`, `search` in a binary search tree?

Why Balanced Trees?

- What is the worst case complexity of `insert`, `delete`, `search` in a binary search tree?
- $O(n)$

Why Balanced Trees?

- What is the worst case complexity of `insert, delete, search` in a binary search tree?
- $O(n)$
- We need to do something better...

WHY BALANCED TREES?

- What is the worst case complexity of `insert, delete, search` in a binary search tree?
- $O(n)$
- We need to do something better...
  - AVL trees

Why Balanced Trees?

- What is the worst case complexity of `insert`, `delete`, `search` in a binary search tree?
- $O(n)$
- We need to do something better...
  - AVL trees
  - B-trees

Why Balanced Trees?

- What is the worst case complexity of `insert, delete, search` in a binary search tree?
- $O(n)$
- We need to do something better...
  - AVL trees
  - B-trees
  - Splay trees

AVL Trees were invented by *A*delson-*V*elskii and *L*andis in 1962.

An AVL tree is similar to a *BST* in that it

- stores values in the *internal nodes* and

AVL TREES

AVL Trees were invented by *A*delson-*V*elskii and *L*andis in 1962.

An AVL tree is similar to a *BST* in that it

- stores values in the *internal nodes* and
- has a property *relating* the values stored in a *subtree* to the values in the *parent node*.

AVL TREES

AVL Trees were invented by *A*delson-*V*elskii and *L*andis in 1962.

An AVL tree is similar to a *BST* in that it

- stores values in the *internal nodes* and
- has a property *relating* the values stored in a *subtree* to the values in the *parent node*.

but different from a *BST* because

AVL TREES

AVL Trees were invented by *A*delson-*V*elskii and *L*andis in 1962.

An AVL tree is similar to a *BST* in that it

- stores values in the *internal nodes* and
- has a property *relating* the values stored in a *subtree* to the values in the *parent node*.

but different from a *BST* because

- The height of an *AVL* tree is $O(\log n)$.

AVL Trees

AVL Trees were invented by *A*delson-*V*elskii and *L*andis in 1962.

An AVL tree is similar to a *BST* in that it

- stores values in the *internal nodes* and
- has a property *relating* the values stored in a *subtree* to the values in the *parent node*.

but different from a *BST* because

- The height of an *AVL* tree is $O(\log n)$.
- Each *internal node* has a *balance property* equal to -1, 0, 1.

AVL Trees

AVL Trees were invented by *A*delson-*V*elskii and *L*andis in 1962.

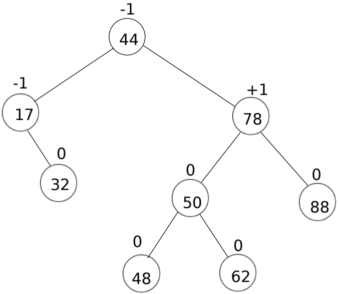
An AVL tree is similar to a *BST* in that it

- stores values in the *internal nodes* and
- has a property *relating* the values stored in a *subtree* to the values in the *parent node*.

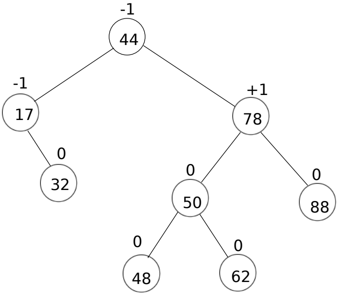
but different from a *BST* because

- The height of an *AVL* tree is $O(\log n)$.
- Each *internal node* has a *balance property* equal to -1, 0, 1.
- Balance value = *height* of the *left* subtree - *height* of the *right* subtree.

AVL Trees



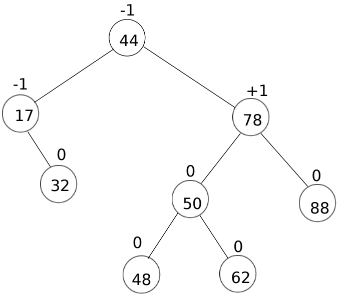
Q. What is the purpose of the *balance* property?



Q. What is the purpose of the *balance* property?

A. Ensures that the *height* is always a function of $\log n$.

AVL TREES

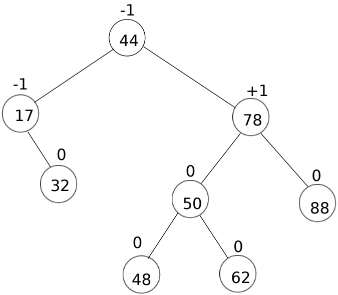


Q. What is the purpose of the *balance* property?

A. Ensures that the *height* is always a function of $\log n$.

Q. What information will we need to store in order to update the *balance factors* easily?

AVL TREES



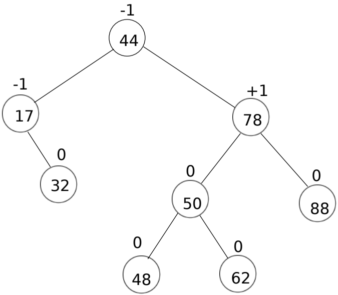
Q. What is the purpose of the *balance* property?

A. Ensures that the *height* is always a function of $\log n$.

Q. What information will we need to store in order to update the *balance factors* easily?

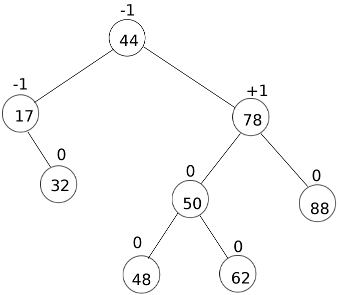
A. The *height* of the tree rooted at each node.

AVL Operations



Searching in an AVL tree is the *same* as a BST.

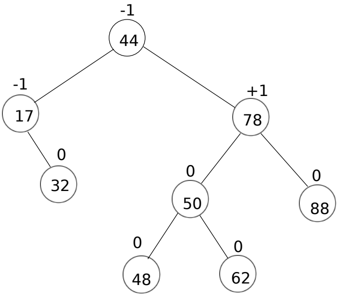
AVL Operations



Searching in an AVL tree is the *same* as a BST.

Consider *inserting* 6 into the tree above.

AVL Operations



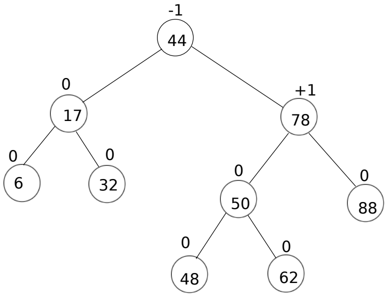
Searching in an AVL tree is the *same* as a BST.

Consider *inserting* 6 into the tree above.

Q: What are the new *balance factors* in the tree after inserting 6?

A:

AVL Operations



Searching in an AVL tree is the *same* as a BST.

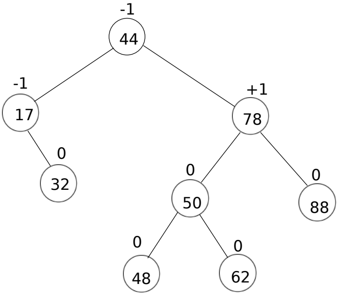
Consider *inserting* 6 into the tree above.

Q: What are the new *balance factors* in the tree after inserting 6?

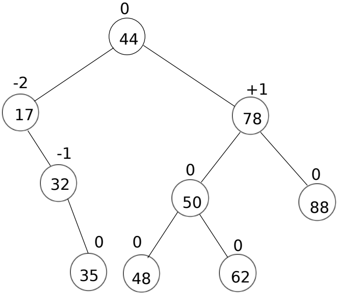
A: 17 has value 0

AVL Insertion

Q. Let's insert 35. What are the *balance factors* now?

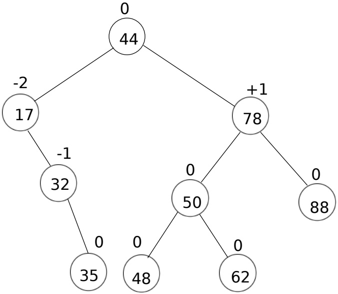


Q. Let's insert 35. What are the *balance factors* now?



AVL Insertion

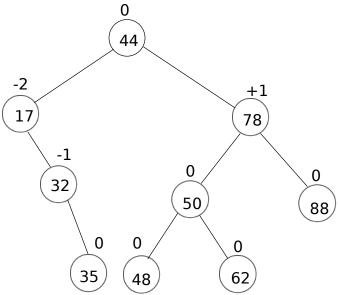
Q. Let's insert 35. What are the *balance factors* now?



A. 32 has -1, 17 has -2. 44 has 0.

AVL Insertion

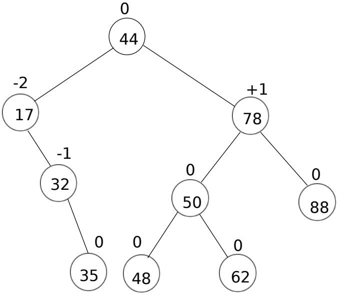
Q. Let's insert 35. What are the *balance factors* now?



A. 32 has -1, 17 has -2. 44 has 0.

Q. How do we solve this problem?

Q. Let's insert 35. What are the *balance factors* now?

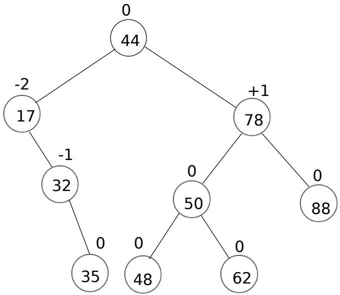


A. 32 has -1, 17 has -2. 44 has 0.

Q. How do we solve this problem?

- We resolve the problem by doing a **single rotation**. How should we *rotate*?

Q. Let's insert 35. What are the *balance factors* now?



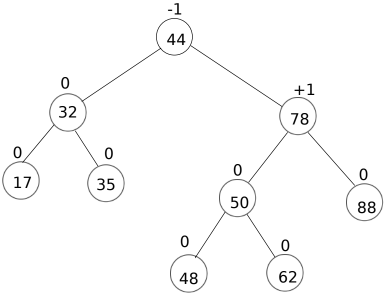
A. 32 has -1, 17 has -2. 44 has 0.

Q. How do we solve this problem?

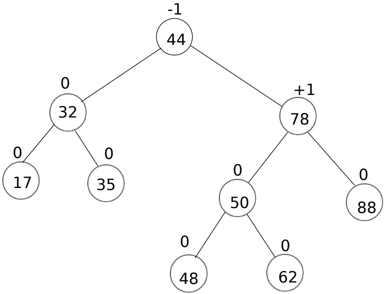
- We resolve the problem by doing a **single rotation**. How should we *rotate*?
- Counter clock-wise. 17 comes down, 32 moves up.

AVL Insertion

- We rotate counter clock-wise. 17 comes down, 32 moves up.



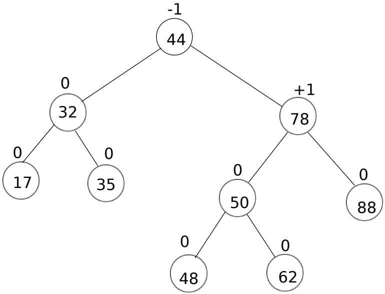
- We rotate counter clock-wise. 17 comes down, 32 moves up.



Q. How do we update the *balance factors*?

AVL INSERTION

- We rotate counter clock-wise. 17 comes down, 32 moves up.

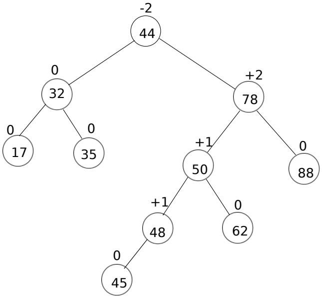


Q. How do we update the *balance factors*?

A. Update the heights first and then update balance factors.

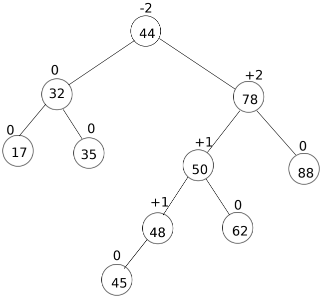
AVL Insertion

Now let's insert *45*.



- Notice the *balance factors* now. How should we resolve the problem?

Now let's insert *45*.

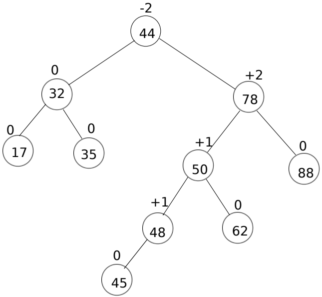


- Notice the *balance factors* now. How should we resolve the problem?

A. Do a *single rotation* clock-wise about the *78*. 50 goes *up*, 78 *down*.

AVL INSERTION

Now let's insert *45*.



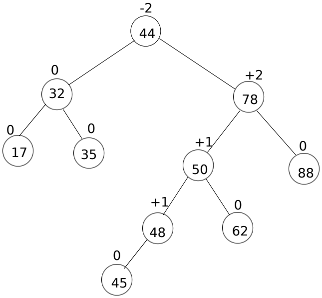
- Notice the *balance factors* now. How should we resolve the problem?

A. Do a *single rotation* clock-wise about the *78*. 50 goes *up*, 78 *down*.

Q. What happens to the subtree rooted at 62?

AVL INSERTION

Now let's insert *45*.



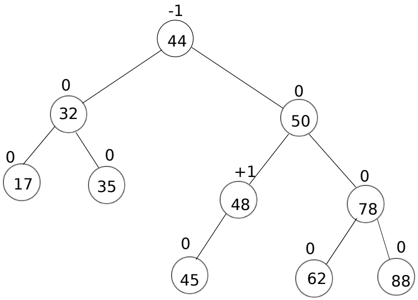
- Notice the *balance factors* now. How should we resolve the problem?

A. Do a *single rotation* clock-wise about the *78*. 50 goes *up*, 78 *down*.

Q. What happens to the subtree rooted at 62?

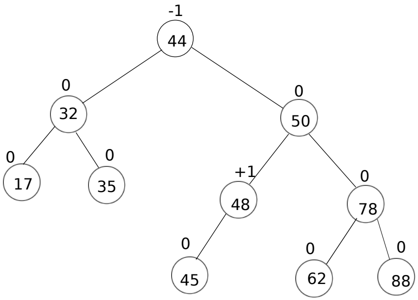
A. It becomes the left subtree of 78.

AVL Insertion



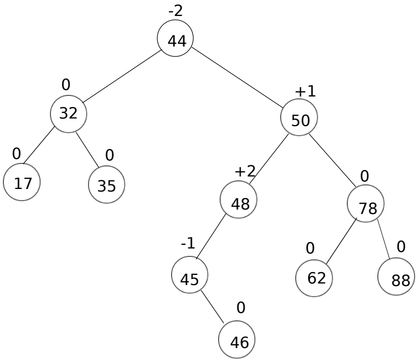
- Notice the updated *balance factors*.

AVL Insertion



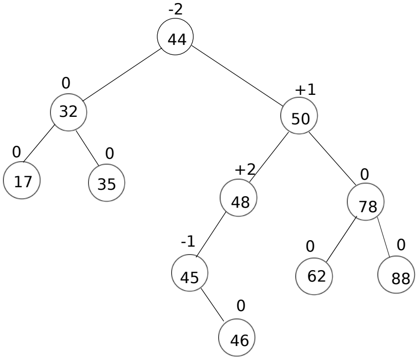
- Notice the updated *balance factors*.
- Let's insert *46* this time.

AVL Insertion



Q. Can we do a rotation about 48?

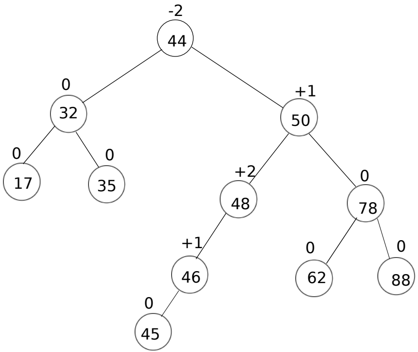
AVL Insertion



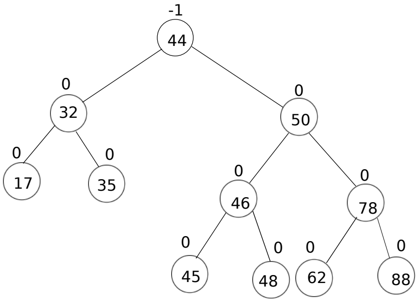
Q. Can we do a rotation about 48?

A. NO. Need a **double rotation**.

Double Rotation

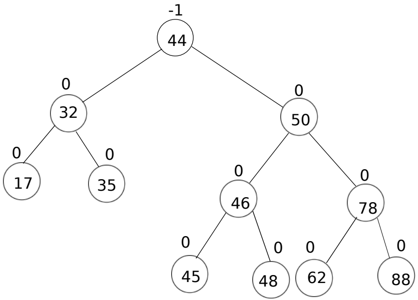


Double Rotation



Q. How do we know that we need a *double rotation*?

DOUBLE ROTATION



Q. How do we know that we need a *double rotation*?

A. Have change in sign of the balance factors (+2, -1 or -2, +1).

Delete

- If the key is a leaf node, delete and rebalance

Delete

- If the key is a leaf node, delete and rebalance
- If the key is an internal node, replace with predecessor/successor and rebalance.

- Since the tree is balanced the height of an *AVL* tree is $O(\log n)$ (we will prove this shortly).

Complexity

- Since the tree is balanced the height of an *AVL* tree is $O(\log n)$ (we will prove this shortly).
- This means *insert*, *delete* and *search* are all $O(\log n)$.

COMPLEXITY

- Since the tree is balanced the height of an *AVL* tree is $O(\log n)$ (we will prove this shortly).
- This means *insert*, *delete* and *search* are all $O(\log n)$.
- Searching for the location to *insert* or *delete*, takes $O(\log n)$.

Complexity

- Since the tree is balanced the height of an *AVL* tree is $O(\log n)$ (we will prove this shortly).
- This means *insert*, *delete* and *search* are all $O(\log n)$.
- Searching for the location to *insert* or *delete*, takes $O(\log n)$.
- *Rebalancing* takes at most $O(\log n)$.

If there are *n* nodes, what is the *maximum* possible *height h*?

Q. What is *another* way to say this in terms of the number of nodes *n*?

A.

AVL Tree Height

If there are *n* nodes, what is the *maximum* possible *height h*?

Q. What is *another* way to say this in terms of the number of nodes *n*?

A. If the *height* is *h*, what is the *minimum* possible number of *nodes*?

AVL TREE HEIGHT

If there are n nodes, what is the *maximum* possible *height* h?

Q. What is *another* way to say this in terms of the number of nodes n?

A. If the *height* is h, what is the *minimum* possible number of *nodes*?

Let *minsize(h)* be the minimum number of nodes for an AVL tree of height h.

AVL Tree Height

Let *minsize(h)* be the minimum number of nodes for an AVL tree of height *h*.

Then, we can say:

$$\begin{aligned} \mathit{minsize}(0) &= 0 \\ \mathit{minsize}(1) &= 1 \\ \mathit{minsize}(h+2) &= 1 + \mathit{minsize}(h+1) + \mathit{minsize}(h) \end{aligned}$$

Q. Does this *remind* you of anything?

A. Fibonacci...

Let *minsize(h)* be the minimum number of nodes for an AVL tree of height *h*.

Then, we can say:

$$
\begin{aligned}
\mathit{minsize}(0) &= 0 \\
\mathit{minsize}(1) &= 1 \\
\mathit{minsize}(h+2) &= 1 + \mathit{minsize}(h+1) + \mathit{minsize}(h)
\end{aligned}
$$

Q. Does this *remind* you of anything?

A.

Let *minsize(h)* be the minimum number of nodes for an AVL tree of height *h*.

Then, we can say:

$$\begin{aligned} minsize(0) &= 0 \\ minsize(1) &= 1 \\ minsize(h+2) &= 1 + minsize(h+1) + minsize(h) \end{aligned}$$

Q. Does this *remind* you of anything?

A. Fibonacci...

AVL Tree Height

$$minsize(h+2) = 1 + minsize(h+1) + minsize(h)$$

and *Fibonacci*:

Can prove by *induction*:

$$minsize(h) = fib(h+2) - 1$$

and given the golden ratio $\phi = (\sqrt{5}+1)/2 = 1.618\ldots$ that

AVL Tree Height

$$minsize(h+2) = 1 + minsize(h+1) + minsize(h)$$

and *Fibonacci*:

Can prove by *induction*:

$$minsize(h) = fib(h+2) - 1$$

and given the golden ratio $\phi = (\sqrt{5}+1)/2 = 1.618\ldots$ that

$$minsize(h) = \frac{\phi^{h+2} - (1-\phi)^{h+2}}{\sqrt{5}} - 1$$

AVL Tree Height

$$\begin{aligned} minsize(h) &= \frac{\phi^{h+2}}{\sqrt{5}} - \frac{(1-\phi)^{h+2}}{\sqrt{5}} - 1 \\ &> \frac{\phi^{h+2}}{\sqrt{5}} - 1 - 1 \\ n \geq minsize(h) &> \frac{\phi^{h+2}}{\sqrt{5}} - 1 - 1 \end{aligned}$$

Solving for h:

$$\frac{\phi^{h+2}}{\sqrt{5}} - 2 < n$$

$$\begin{aligned} h &< \frac{\lg(n+2)}{\lg\phi} + \frac{\sqrt{5}}{\lg\phi} + 2 \\ &= 1.44\lg(n+2) + \text{constant} \\ &\in O(\lg n) \end{aligned}$$