AVL Trees

- AVL Property
- Insertion
- Deletion
- Summing it all up
AVL Trees

- AVL trees are special BSTs:
  - Keep track of **height** of every subtree.
  - At each node, require left and right subtrees to differ in height by 1 or less.
  - by G. M. Adelson-Velsky and E. M. Landis (1962)

- Leads to a tree of height $\Theta(\log n)$. 
AVL Trees

Suppose total tree height = 4.

In an AVL tree, the L, R subtrees can be what height?
AVL Trees

Suppose total tree height = 4.

In an AVL tree, the L, R subtrees can be what height?
Rule: they may differ in height by 1 or less.
AVL Trees

They might be perfectly balanced.
(Difference in height: zero)
AVL Trees

The right side might be taller by one.
(Difference in height: one)
AVL Trees

The left side might be taller by one.
(Difference in height: one)
AVL Trees

And so on.
AVL Trees

- $h=0$
- $h=1$
- $h=2$
- $h=3$
- $h=4$
AVL Trees
AVL Trees

- $h=0$
- $h=1$
- $h=2$
- $h=3$
- $h=4$
- Height = 1
- Height = 2
- Height = 3
- Height = 4
AVL Trees
AVL Trees
Insight:
“Balanced” doesn't mean “perfect.”
This tree has only 12 nodes...out of 31 possible! (height=4)

Q. What about empty subtrees?
A. It works to assume they have height of negative one.
AVL Trees: Balance factors

Actually, there is no need to store height. A few bits will do. You only need to indicate balance (=) or heavy side (>, <).
AVL Trees: Balance factors

Actually, there is no need to store height. A few bits will do. You only need to indicate balance (=) or heavy side (>, <).
AVL Trees: Balance factors

Actually, there is no need to store height. A few bits will do. You only need to indicate balance (=) or heavy side (> or <).
AVL Trees: Balance factors

Actually, there is no need to store height. A few bits will do. You only need to indicate balance (=) or heavy side (>), <).
AVL Trees: Balance factors

Actually, there is no need to store height. A few bits will do. You only need to indicate balance (=) or heavy side (> or <).

NOTE: It is only a few bits, but each AVL tree node **must** store this balance factor, and keep it up to date.
AVL Trees: Balance factors

Actually, there is no need to store height. A few bits will do. You only need to indicate balance (=) or heavy side (>, <).

NOTE: It is only a few bits, but each AVL tree node **must** store this balance factor, and keep it up to date.

During insertion & deletion, a node may temporarily exceed ±1 height diff., which we will denote using >>, <<.
AVL Tree Shape: recursive def

• Base case:
  – If an AVL tree has height zero or one, it must have one of the following shapes:

![Base case diagrams]

• Inductive case: if the height $h \geq 2$,
  – subtrees $L, R$ have AVL tree shape,
  – at least one subtree has height $h-1$, and
  – other subtree has height $h-1$ or $h-2$. 

$\hfill \text{height } h \quad L \quad R$
AVL Tree Height

• Claim:
  The height of an AVL tree of $n$ nodes is $O(\log n)$.

• Idea:
  Show that if an AVL tree has height $h$, it holds over $F_h$ nodes, where $F_h$ is the $h$-th Fibonacci number:

\[
\begin{align*}
F_0 &= 0 \\
F_1 &= 1 \\
F_2 &= 1 \\
F_3 &= 2 \\
F_4 &= 3 \\
\end{align*}
\]
AVL Tree Height

• Base case: height of zero or one
  – By definition, if an AVL tree has height zero or one, it must have one of the following shapes:

  \[
  \begin{array}{cccc}
  \text{h=0} & \text{h=1} & \text{h=1} & \text{h=1} \\
  \end{array}
  \]

  Height \( h = \_ \). Number of nodes: \_.
  Is that at least \( F_h \)?
  \( F_h = \_ \), and ...
  so ..................
AVL Tree Height

• Base case: height of zero or one
  – By definition, if an AVL tree has height zero or one, it must have one of the following shapes:

    - Height $h = 0$. Number of nodes: 1
      Is that at least $F_h$?
      $F_h = 0$, and $0 < 1$, so the claim holds.
AVL Tree Height

- Base case: height of zero or one
  - By definition, if an AVL tree has height zero or one, it must have one of the following shapes:

  - Height $h = 0$. Number of nodes: 1
    Is that at least $F_h$?
    $F_h = 0$, and $0 < 1$, so the claim holds.

  - Height $h = 1$. Number of nodes: 2
    Is that at least $F_h$?
    $F_h = 1$, and $1 < 2$, so the claim holds.
AVL Tree Height

- Base case: height of zero or one
  - By definition, if an AVL tree has height zero or one, it must have one of the following shapes:

  1. Height $h = 0$. Number of nodes: 1
     Is that at least $F_h$?
     $F_h = 0$, and $0 < 1$, so the claim holds.

  2. Height $h = 1$. Number of nodes: 2
     Is that at least $F_h$?
     $F_h = 1$, and $1 < 2$, so the claim holds.

  3. Height $h = 1$. Number of nodes: 2
     Is that at least $F_h$?
     $F_h = 1$, and $1 < 2$, so the claim holds.
AVL Tree Height

- Base case: height of zero or one
  - By definition, if an AVL tree has height zero or one, it must have one of the following shapes:

  - Height $h = 0$. Number of nodes: 1
    Is that at least $F_h$? $F_h = 0$, and $0 < 1$, so the claim holds.

  - Height $h = 1$. Number of nodes: 2
    Is that at least $F_h$? $F_h = 1$, and $1 < 2$, so the claim holds.

  - Height $h = 1$. Number of nodes: 2
    Is that at least $F_h$? $F_h = 1$, and $1 < 2$, so the claim holds.

  - Height $h = 1$. Number of nodes: 3
    Is that at least $F_h$? $F_h = 1$, and $1 < 3$, so the claim holds.
Proof (cont'd):

**Inductive Case (height ≥ 2):**

Consider a tree with height \( h \geq 2 \). Then at least one subtree of root, (WLOG, \( L \)) has height \( h-1 \).

- The height of the other subtree, (WLOG, \( R \)) is either \( h-1 \) or \( h-2 \). That is, \( R \) has height *at least* \( h-2 \).
Proof (cont'd):

Inductive Case (height $\geq 2)$: By the I.H. we know that subtree $L$ has at least $F_{h-1}$ nodes, and subtree $R$ has at least $F_{h-2}$ nodes. Thus, this whole tree has at least

$$1 + F_{h-1} + F_{h-2}$$

nodes . . . .
AVL Tree Height

Proof (cont'd):

Inductive Case (height \( \geq 2 \)):

By the I.H. we know that subtree \( L \) has at least \( F_{h-1} \) nodes, and subtree \( R \) has at least \( F_{h-2} \) nodes. Thus, this whole tree has at least

\[
1 + F_h
\]

nodes, which clearly is greater than \( F_h \).

- Thus, the inductive case also holds.
Proof (cont'd):

Conclusion:

We have shown that an AVL tree of height $h$ has at least $F_h$ nodes.

We know* that the Fibonacci sequence grows exponentially, so, the number of nodes is

$$n = \Omega(1.618^h)$$

and so,

$$h = O(\log n).$$

* Do we?
Fibonacci numbers do that, btw.

- Fibonacci numbers grow exponentially quickly:
  \[ F_n = \Theta \left( \phi^n \right) \]
  where \( \phi = \frac{1+\sqrt{5}}{2} \approx 1.61803 \).

- Puzzle for you:
  Let \( \hat{\phi} = \frac{1-\sqrt{5}}{2} \). Show \( \phi^2 = \phi + 1 \) and \( \hat{\phi}^2 = \hat{\phi} + 1 \).

  Using induction, show \( F_n = \frac{1}{\sqrt{5}} \left( \phi^n - \hat{\phi}^n \right) \). (D. Bernoulli, 1728)
AVL Trees

- AVL Property
- Insertion
- Deletion
- Summing it all up
AVL Trees

• How to insert into an AVL Tree?
  – Insert new leaves
  – Update heights
  – Do rotations if necessary
Inserting into an AVL Tree (Examples)

In these two trees...

Any insertion will cause the height to grow.

But no insertion can possibly violate the AVL shape property.
Inserting into an AVL Tree (Examples)

But what about this tree?

*Some* insertions require rotations to fix the shape property.
Inserting into an AVL Tree (Examples)

Nothing to fix!
Inserting into an AVL Tree (Examples)

Inserting node A into this tree causes an imbalance at C.

Can you see why?
Inserting into an AVL Tree (Examples)

1. Go up from new node, check balance at each ancestor.
2. Fix *lowest unbalanced ancestor*.
3. If path goes left-left or right-right then *one rotation* is enough. Otherwise, *two* are required.

E.g., rotate right at C to balance.

*This is not a splay tree: Different rules apply. Don't even think of zig-zig. The rule here is *one rotation*.***
Inserting into an AVL Tree (Examples)

Inserting node A into this tree causes an imbalance. But rotating right at C would not help us.

Can you see why?

This is not a splay tree, so don't call it “zig-zag.”
Inserting into an AVL Tree (Examples)

Rotate left at B, and then this is equivalent to the previous case.
Inserting into an AVL Tree
(Examples)

A <
B =
A =

C >>
B =
A =

Rotate left at B, and then this is equivalent to the previous case.

This is known as a “double rotation.”

(Yes, just like zig zag.)
Inserting into an AVL Tree (Examples)

Conclusions

After an insertion to these trees, rotations **NEVER** required.

But the height grows!

After an insertion to these trees, rotations sometimes required.

But the height **NEVER** grows!
Inserting into an AVL Tree (Practice)

• Insert into an AVL tree (alphabetical order):
  – Bulbasaur
  – Charmander
  – Squirtle
  – Caterpie
  – Weedle
  – Pidgey
  – Pikachu
  – Snorlax
  – Magikarp

• Now, insert in this order (new tree):
  – Bulbasaur
  – Weedle
  – Caterpie
  – Squirtle
  – Charmander
  – Snorlax
  – Magikarp
  – Pikachu
  – Pidgey
AVL Trees – Insertion (Summary)

To insert in an AVL tree:
- Do normal insertion
- Do “rebalance” back toward the root:
  - Update height info among ancestors.
  - Maybe do 1 or 2 rotations (depends on link directions).

Cost:
- $O(\log n)$ to insert
- $O(\log n)$ to update height fields
- $O(1)$ to do rotations
AVL Trees

- AVL Property
- Insertion
- Deletion
- Summing it all up
Deletion in an AVL Tree

Two steps to deletion:

• Delete

• Rebalance, similar to the algorithm of Insert()
  – Go up from where the node was removed, to root.
  – Update balance factors of each ancestor.
  – Help (each) unbalanced ancestor $x$, $x$'s higher child $y$, and $y$'s higher child $z$.
  – Rotations at multiple levels may be required!
Multiple Rotations

• As with insertion, our algorithm works up from the deletion point
  – Update heights.
  – Perform rotations where there are imbalances.
  – Unbalanced ancestors will be on the path from deletion point toward root, but,
  – Rotations also involve off-path nodes.

• The rotation after a deletion can reduce height of a subtree – which means that multiple rotations are possible
Multiple Rotations

Warmup:
Is this an AVL tree?
Balance factors?
Multiple Rotations
Suppose we delete L from the tree.
Multiple Rotations

This creates an imbalance at K, which we fix with a single rotation.
This creates a new imbalance at H, which we fix with another rotation.
This reduced the height of the tree – meaning that we could have had even more rotations, further up.
Multiple Rotations: your turn

Delete L from this AVL tree.
Multiple Rotations: your turn

1. Replace L with its successor, P.
Multiple Rotations: your turn

2. Go up and check balance factors.
3. From unbalanced node, find deepest child and grandchild.
Multiple Rotations: your turn

4. Repair with 1 rotation if path is L-L or R-R; or “double” if path zig-zags.
Go up and repeat. This might involve non-ancestor nodes.
Multiple Rotations: your turn
Multiple Rotations: your turn
Multiple Rotations: your turn
AVL Trees – Deletion

• To delete from an AVL tree:
  – Do normal deletion
  – Do “rebalance” back toward the root
    • Increase some heights
    • Do rotations (maybe at multiple levels)

• Cost:
  – $O(\log n)$ to delete
  – $O(\log n)$ to update height fields
  – $O(\log n)$ to do rotations
AVL Trees (summary)

- AVL Property: child heights differ by no more than 1
  - Does not minimize tree height.
  - But it's close enough to achieve height of $O(\log n)$.
  - (Sometimes it's best to be mediocre?)

- Insertion/Deletion
  - Do the normal op, then fix any unhappy ancestors.
  - Insertion might trigger single or double rotation.
  - Deletion might trigger $O(\log n)$ rotations.
AVL Trees (summary)

- Two rotation types during an AVL rebalance:
  - Single rotation at the root of unbalanced subtree.
  - Single rotation if the links from the unbalanced node go left-left or right-right towards deepest leaf.
  - Double rotation: first at child, then root.
  - Double rotation if the links from unbalanced node zig-zag toward deepest leaf.

- Advantages: guaranteed $O(\log n)$ height BST and $O(\log n)$ time-cost of basic operations.

- Disadvantage: additional code complexity