AVL Trees

An AVL tree is a binary search tree in which for every internal node the heights of its two subtrees differ by at most 1.
AVL Trees

An AVL tree is a binary search tree in which for every internal node the heights of its two subtrees differ by at most 1.
AVL Trees
What is the Maximum Height of an AVL Tree?

Let \( n(h) \) = minimum number of nodes in an AVL tree of height \( h \).
What is the Maximum Height of an AVL Tree?

Let $n(h)$ = minimum number of nodes in an AVL tree of height $h$.

$n(0) = 1$, $n(1) = 3$, $n(2) = 5$, $n(3) = 9$, $n(4) = 15$, ...

$n(h) = 1 + n(h-1) + n(h-2) > 2n(h-2)$
Solve the recurrence equation for h even

\[ n(0) = 1 \]
\[ n(h) > 2n(h-2) \]
\[ 2n(h-2) > 2^2n(h-2\times2) \]
\[ 2^2n(h-2\times2) > 2^3 n(h-2\times3) \]
\[ \ldots \]
\[ 2^i n(h-2\times i) > 2^{i+1} n(h-2\times(i+1)) = 0 \]

Then, \[ n(h) > 2^{i+1} n(0) = 2^{i+1} \]
Solve the recurrence equation for h even

Since \( h - 2 \times (i + 1) = 0 \), then \( i + 1 = h/2 \) and so
\[
n(h) = n > 2^{i+1} = 2^{h/2}
\]

Therefore, taking logarithms on both sides we get
\[
h/2 \leq \log_2 n
\]
and so
\[
\text{height} = h < 2 \log_2 n, \text{ so height is } O(\log n)
\]
Re-Balancing AVL Trees

To re-balance an AVL tree we always rebalance the smallest un-balanced subtree.
Single Rotations

Single Rotation

RR

LL

Single Rotation
Double Rotations

Double Rotation

RL

Double Rotation

LR
Re-Balancing AVL Trees

If the tree becomes unbalanced due to an insertion **ONE** rotation will re-balance the tree.
Insertion

- Insertion is as in a binary search tree
- Re-balance if needed
Insertion Example, continued

unbalanced...

...balanced
**Algorithm** putAVL \((r, k, \text{data})\)

**In:** Root \(r\) of an AVL tree, record \((k, \text{data})\)

**Out:** \{Insert \((k, \text{data})\) and re-balance if needed\}

\[
\text{put}(r, k, \text{data}) \quad // \text{Algorithm for binary search trees}
\]

Let \(p\) be the node where \((k, \text{data})\) was inserted

\[\text{while } (p \neq \text{null}) \text{ and (subtrees of } p \text{ differ in height } \leq 1) \text{ do}\]

\[p = \text{parent of } p\]

\[\text{if } p \neq \text{null then} \text{ rebalance subtree rooted at } p \text{ by performing appropriate rotation}\]
Re-Balancing AVL Trees

When a single and a double rotation can be applied to an un-balanced subtree the single rotation always re-balances the subtree.
Re-Balancing AVL Trees

If the tree becomes unbalanced due to a removal **SEVERAL** rotations might be needed to re-balance the tree.
Removal

- Removal begins as in a binary search tree, which means the node removed will become a leaf.
- Re-balance if needed.
Algorithm removeAVL \((r, k)\)

**In:** Root \(r\) of an AVL tree, key \(k\) to remove

**Out:** \{Remove \(k\) and re-balance if needed\}

```plaintext
remove(r,k)  // Algorithm for binary search trees
Let \(p\) be the parent of the node that was removed
while \((p \neq \text{null})\) do {
    if subtrees of \(p\) differ in height > 1 then
        rebalance subtree rooted at \(p\) by performing appropriate rotation
    \(p = \text{parent of } p\)
}
AVL Tree Performance

- AVL tree storing $n$ items
  - The data structure uses $O(n)$ space
  - A single rotation takes $O(1)$ time
    - using a linked-structure binary tree
  - Get takes $O(\log n)$ time
    - height of tree is $O(\log n)$, no re-balancing needed
  - Put takes $O(\log n)$ time
    - initial get operation takes $O(\log n)$ time
    - rebalancing the tree takes $O(1)$ time, as at most one rebalancing operation is needed
  - Removal takes $O(\log n)$ time
    - initial get operation takes $O(\log n)$ time
    - rebalancing the tree needs $O(\log n)$ time as several rebalancing operations might be needed