AVL Trees

Idea: More precise balance condition than RB for smaller worst case height.

Definition: A binary search tree is an AVL tree if every non-leaf vertex except the root has a sibling, and sibling heights differs by at most 1.

Terminology:
- A vertex $v$ in a binary search tree is light if it has a sibling of greater height; vertices that are not light are said to be heavy.
- A vertex is left-heavy if it has a heavy left child and a light right child.
- A vertex is right-heavy if it has a light left child and a heavy right child.

Example: All of the following are AVL trees (heavy=black, light=shaded).

Note: In any AVL tree, from every vertex, there is at least one path of heavy vertices to a leaf (since every non-leaf has at least one heavy child).
Idea: Each vertex stores a light-heavy bit. After an INSERT or DELETE, go up to the root from the deepest vertex $x$ that was visited, and at each vertex, if it is left-heavy apply one of the two cases above (right-heavy is symmetric).
(the AVL algorithm continued)

while \( x \neq \text{nil} \) do begin

CASE 1: \( x \) is left heavy:

\( y := \text{LCHILD}(x) \)

if \( y \) is left-heavy then begin

RR(\( x \))
Label \( y \) the same as the label of \( x \).
Label \( x \) heavy.
Label both children of \( x \) heavy.
end

else begin

\( z := \text{RCHILD}(y) \)
RL(\( y \)), RR(\( x \))
Label \( z \) the same as the label of \( x \).
Label \( x \) heavy.
Label the left child of \( y \) and the right child of \( x \) heavy.
end

end

CASE 2: \( x \) is right-heavy: Symmetric to CASE 1.

end
Note:

It can be shown that an AVL tree of $n$ vertices has height $< 1.45 \left\lfloor \log_2(n) \right\rfloor$. 