Algorithm A (*AVL Tree Insertion*). Given a set of nodes which form an AVL tree `T`, and a key to insert `K`, this algorithm will insert the node into the tree while maintaining the tree's balance properties. Each node is assumed to contain `KEY`, `BAL`, `LLINK`, `RLINK`, and `PARENT` fields. For any given node `N`, `KEY(N)` gives the key field of `N`, `BAL(N)` gives the balance field of `N`, `LLINK(N)` and `RLINK(N)` are pointers to `N`'s left and right subtrees, respectively, and `PARENT(N)` is a pointer to the node of which `N` is a subtree. Any or all of these three link fields may be Λ, which for `LLINK(N)` and `RLINK(N)` indicates that `N` has no left or right subtree, respectively, and for `PARENT(N)` indicates that `N` is the root of the tree. The `BAL` field must be able to represent an integer on the range $[-2, +2]$. The tree has a field `ROOT` which is a pointer to the root node of the tree.

You can find an implementation of this algorithm, as well as many others, in **libdict**, which is available on the web at `http://www.crazycoder.org/libdict.html`.

A1. [Initialize.] Set $\mathtt{N} \leftarrow \mathtt{ROOT(T)}$, $\mathtt{P} \leftarrow \mathtt{Q} \leftarrow \Lambda$.

A2. [Find insertion point.] If $\mathtt{N} = \Lambda$, go to step A3. If $\mathtt{K} = \mathtt{KEY(N)}$, the key is already in the tree and the algorithm terminates with an error. Set $\mathtt{P} \leftarrow \mathtt{N}$; if $\mathtt{BAL(P)} \neq 0$, set $\mathtt{Q} \leftarrow \mathtt{P}$. If $\mathtt{K} < \mathtt{KEY(N)}$, then set $\mathtt{N} \leftarrow \mathtt{LLINK(N)}$, otherwise set $\mathtt{N} \leftarrow \mathtt{RLINK(N)}$. Repeat this step.

A3. [Insert.] Set $\mathtt{N} \leftarrow \mathtt{AVAIL}$. If $\mathtt{N} = \Lambda$, the algorithm terminates with an out of memory error. Set $\mathtt{KEY(N)} \leftarrow \mathtt{K}$, $\mathtt{LLINK(N)} \leftarrow \mathtt{RLINK(N)} \leftarrow \Lambda$, $\mathtt{PARENT(N)} \leftarrow \mathtt{P}$, and $\mathtt{BAL(N)} \leftarrow 0$. If $\mathtt{P} = \Lambda$, set $\mathtt{ROOT(T)} \leftarrow \mathtt{N}$, and go to step A8. If $\mathtt{K} < \mathtt{KEY(P)}$, set $\mathtt{LLINK(P)} \leftarrow \mathtt{N}$; otherwise, set $\mathtt{RLINK(P)} \leftarrow \mathtt{N}$.

A4. [Adjust balance factors.] If $\mathtt{P} = \mathtt{Q}$, go to step A5. If $\mathtt{LLINK(P)} = \mathtt{N}$, set $\mathtt{BAL(P)} \leftarrow -1$; otherwise, set $\mathtt{BAL(P)} \leftarrow +1$. Then set $\mathtt{N} \leftarrow \mathtt{P}$, and $\mathtt{P} \leftarrow \mathtt{PARENT(P)}$, and repeat this step.

A5. [Check for imbalance.] If $\mathtt{Q} = \Lambda$, go to step A8. Otherwise:

i. If $\mathtt{LLINK(Q)} = \mathtt{N}$, set $\mathtt{BAL(Q)} \leftarrow \mathtt{BAL(Q)} - 1$. If $\mathtt{BAL(Q)} = -2$, go to step A6, otherwise, go to step A8.

ii. If $\mathtt{RLINK(Q)} = \mathtt{N}$, set $\mathtt{BAL(Q)} \leftarrow \mathtt{BAL(Q)} + 1$. If $\mathtt{BAL(Q)} = +2$, go to step A7, otherwise, go to step A8.

A6. [Left imbalance.] If $\mathtt{BAL(LLINK(Q))} > 0$, rotate `LLINK(Q)` left. Rotate `Q` right. Go to step A8.

A7. [Right imbalance.] If $\mathtt{BAL(RLINK(Q))} < 0$, rotate `RLINK(Q)` right. Rotate `Q` left. Go to step A8.

A8. [All done.] The algorithm terminates successfully.

Rotations in AVL Trees

Algorithm R (*Right Rotation*). Given a tree `T` and a node in the tree `N`, this routine will rotate `N` right.

R1. [Do the rotation.] Set $\mathtt{L} \leftarrow \mathtt{LLINK(N)}$ and $\mathtt{LLINK(N)} \leftarrow \mathtt{RLINK(L)}$. If $\mathtt{RLINK(L)} \neq \Lambda$, then set $\mathtt{PARENT(RLINK(L))} \leftarrow \mathtt{N}$. Set $\mathtt{P} \leftarrow \mathtt{PARENT(N)}$, $\mathtt{PARENT(L)} \leftarrow \mathtt{P}$. If $\mathtt{P} = \Lambda$, then set $\mathtt{ROOT(T)} \leftarrow \mathtt{L}$; if $\mathtt{P} \neq \Lambda$ and $\mathtt{LLINK(P)} = \mathtt{N}$, set $\mathtt{LLINK(P)} \leftarrow \mathtt{L}$, otherwise set $\mathtt{RLINK(P)} \leftarrow \mathtt{L}$. Finally, set $\mathtt{RLINK(L)} \leftarrow \mathtt{N}$, and $\mathtt{PARENT(N)} \leftarrow \mathtt{L}$.

R2. [Recompute balance factors.] Set $\mathtt{BAL(N)} \leftarrow \mathtt{BAL(N)} + (1 - \mathtt{MIN(BAL(L), 0)})$, $\mathtt{BAL(L)} \leftarrow \mathtt{BAL(L)} + (1 + \mathtt{MAX(BAL(N), 0)})$.

The code for a left rotation is symmetric. At the risk of being repetitive, it appears below.

Algorithm L (*Left Rotation*). Given a tree `T` and a node in the tree `N`, this routine will rotate `N` left.

L1. [Do the rotation.] Set $\mathtt{R} \leftarrow \mathtt{RLINK(N)}$ and $\mathtt{RLINK(N)} \leftarrow \mathtt{LLINK(R)}$. If $\mathtt{LLINK(R)} \neq \Lambda$, then set $\mathtt{PARENT(LLINK(R))} \leftarrow \mathtt{N}$. Set $\mathtt{P} \leftarrow \mathtt{PARENT(N)}$, $\mathtt{PARENT(R)} \leftarrow \mathtt{P}$. If $\mathtt{P} = \Lambda$, then set $\mathtt{ROOT(T)} \leftarrow \mathtt{R}$; if $\mathtt{P} \neq \Lambda$ and $\mathtt{LLINK(P)} = \mathtt{N}$, set $\mathtt{LLINK(P)} \leftarrow \mathtt{R}$, otherwise set $\mathtt{RLINK(P)} \leftarrow \mathtt{R}$. Finally, set $\mathtt{LLINK(R)} \leftarrow \mathtt{N}$, and $\mathtt{PARENT(N)} \leftarrow \mathtt{R}$.

L2. [Recompute balance factors.] Set $\mathtt{BAL(N)} \leftarrow \mathtt{BAL(N)} - (1 + \mathtt{MAX(BAL(R), 0)})$, $\mathtt{BAL(R)} \leftarrow \mathtt{BAL(R)} - (1 - \mathtt{MIN(BAL(N), 0)})$.