An Iterative Algorithm for AVL Tree Insertion

Farooq Mela <crazycoder@crazycoder.org>

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 N ← ROOT(T), P ← Q ← Λ.

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

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

A4. [Adjust balance factors.] If P = Λ, go to step A5. If LLINK(P) = N, set BAL(P) ← −1; otherwise, set BAL(P) ← +1. Then set N ← P, and P ← PARENT(P), and repeat this step.

A5. [Check for imbalance.] If Q = Λ, go to step A8. Otherwise:
   i. If LLINK(Q) = N, set BAL(Q) ← BAL(Q) − 1. If BAL(Q) = −2, go to step A6, otherwise, go to step A8.
   ii. If RLINK(Q) = N, set BAL(Q) ← BAL(Q) + 1. If BAL(Q) = +2, go to step A7, otherwise, go to step A8.

A6. [Left imbalance.] If BAL(LLINK(Q)) > 0, rotate LLINK(Q) left. Rotate Q right. Go to step A8.

A7. [Right imbalance.] If 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 L ← LLINK(N) and LLINK(N) ← RLINK(L). If RLINK(L) ≠ Λ, then set PARENT(RLINK(L)) ← N. Set P ← PARENT(N), PARENT(L) ← P. If P = Λ, then set ROOT(T) ← L; if P ≠ Λ and LLINK(P) = N, set LLINK(P) ← L, otherwise set RLINK(P) ← L. Finally, set RLINK(L) ← N, and PARENT(N) ← L.

R2. [Recompute balance factors.] Set BAL(N) ← BAL(N) + (1 − MIN(BAL(L), 0)), BAL(L) ← BAL(L) + (1 + 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 R ← RLINK(N) and RLINK(N) ← LLINK(R). If LLINK(R) ≠ Λ, then set PARENT(LLINK(R)) ← N. Set P ← PARENT(N), PARENT(R) ← P. If P = Λ, then set ROOT(T) ← R; if P ≠ Λ and LLINK(P) = N, set LLINK(P) ← R, otherwise set RLINK(P) ← R. Finally, set LLINK(R) ← N, and PARENT(N) ← R.

L2. [Recompute balance factors.] Set BAL(N) ← BAL(N) − (1 + MAX(BAL(R), 0)), BAL(R) ← BAL(R) − (1 − MIN(BAL(N), 0)).