

This algorithm is written using the conventions of Dr. D. E. Knuth in his *Art of Computer Programming* series. You can find this algorithm, as well as source code implementing it, on the web at <http://www.msu.edu/user/pfaffben/avl>.

Algorithm D (*Deletion from AVL tree*). Given a set of nodes which form an AVL-balanced binary tree, this algorithm searches for a given argument K and deletes it from the tree, and rebalances the tree if necessary. The nodes of the tree are assumed to contain **KEY**, **LLINK**, **RLINK**, and **B** fields. **KEY** is arbitrary data stored in a node, which must have $<$, $=$, and $>$ operations defined for it. **LLINK** and **RLINK** are pointers to the node's left and right subtrees, respectively, and either or both may be Λ . **B** is the height of the node's right subtree minus the height of its left subtree, and must be -1 , 0 , or $+1$.

The tree has a special header node in location **HEAD**. **LLINK(HEAD)** is a pointer to the root of the tree.

For convenience in description, the algorithm uses the notation **LINK(a, P)** as a synonym for **LLINK(P)** if $a = -1$, and for **RLINK(P)** if $a = +1$.

The algorithm makes use of a pair of auxiliary arrays named P and a , which are used as stacks. P_k is used to store pointers to nodes which may need to be rebalanced. a_k stores -1 or $+1$ such that **LINK(a_k, P_k)** = P_{k+1} .

- D1.** [Initialize.] Set $a_0 \leftarrow -1$, $P_0 \leftarrow \text{HEAD}$, $k \leftarrow 1$, $P \leftarrow \text{LLINK}(\text{HEAD})$. (The pointer variable P will move down the tree.)
 - D2.** [Compare.] If $K < \text{KEY}(P)$, go to D3; if $K > \text{KEY}(P)$, go to D4; and if $K = \text{KEY}(P)$, go to D5.
 - D3.** [Move left.] Set $P_k \leftarrow P$, $a_k \leftarrow -1$, $k \leftarrow k + 1$, $P \leftarrow \text{LLINK}(P)$. If $P \neq \Lambda$, go to D2; otherwise, terminate (the tree does not contain K).
 - D4.** [Move right.] Set $P_k \leftarrow P$, $a_k \leftarrow +1$, $k \leftarrow k + 1$, $P \leftarrow \text{RLINK}(P)$. If $P \neq \Lambda$, go to D2; otherwise, terminate (the tree does not contain K).
 - D5.** [Is **RLINK** null?] Set $Q \leftarrow \text{LOC}(\text{LINK}(a_{k-1}, P_{k-1}))$. (Q now points to the link that was followed to arrive at P .) If **RLINK**(P) $\neq \Lambda$, go to D6. Otherwise, set **CONTENTS**(Q) $\leftarrow \text{LLINK}(P)$. Then, if $Q \neq \Lambda$, set **B**(**CONTENTS**(Q)) $\leftarrow 0$, and go to D10.
 - D6.** [Find successor.] Set $R \leftarrow \text{RLINK}(P)$. If **LLINK**(R) $\neq \Lambda$, go to D7. Otherwise, set **LLINK**(R) $\leftarrow \text{LLINK}(P)$, **CONTENTS**(Q) $\leftarrow R$, **B**(R) $\leftarrow \text{B}(P)$, $a_k \leftarrow +1$, $P_k \leftarrow R$, $k \leftarrow k + 1$, and go to D10.
 - D7.** [Set up to find null **LLINK**.] Set $S \leftarrow \text{LLINK}(R)$, $l \leftarrow k$, $k \leftarrow k + 1$, $a_k \leftarrow -1$, $P_k \leftarrow R$, $k \leftarrow k + 1$.
 - D8.** [Find null **LLINK**.] If **LLINK**(S) = Λ , go to D9. Otherwise, set $R \leftarrow S$, $S \leftarrow \text{LLINK}(R)$, $a_k \leftarrow -1$, $P_k \leftarrow R$, $k \leftarrow k + 1$, and repeat this step.
 - D9.** [Fix up.] Set $a_l \leftarrow +1$, $P_l \leftarrow S$, **LLINK**(S) $\leftarrow \text{LLINK}(P)$, **LLINK**(R) $\leftarrow \text{RLINK}(S)$, **RLINK**(S) $\leftarrow \text{RLINK}(P)$, **B**(S) $\leftarrow \text{B}(P)$, and **CONTENTS**(Q) $\leftarrow S$.
 - D10.** [Adjust balance factors.] Set $k \leftarrow k - 1$. If $k = 0$ then terminate successfully. Otherwise, set $S \leftarrow P_k$, and consider the following cases:
 - i) If **B**(S) = 0 , set **B**(S) $\leftarrow -a_k$, and terminate successfully.
 - ii) If **B**(S) = a_k , set **B**(S) $\leftarrow 0$, and repeat this step.
 - iii) Otherwise, **B**(S) = $-a_k$. Set $R \leftarrow \text{LINK}(-a_k, S)$. Go to D11 if **B**(R) = 0 ; go to D12 if **B**(R) = $-a_k$; otherwise, **B**(R) = a_k , and go to D13.
 - D11.** [Single rotation with balanced R .] Set **LINK**($-a_k, S$) $\leftarrow \text{LINK}(a_k, R)$, **LINK**(a_k, R) $\leftarrow S$, **B**(R) $\leftarrow a_k$, **LINK**(a_{k-1}, P_{k-1}) $\leftarrow R$, and terminate successfully.
 - D12.** [Single rotation with unbalanced R .] Set **LINK**($-a_k, S$) $\leftarrow \text{LINK}(a_k, R)$, **LINK**(a_k, R) $\leftarrow S$, **B**(S) $\leftarrow \text{B}(R) \leftarrow 0$, **LINK**(a_{k-1}, P_{k-1}) $\leftarrow R$, and go to D10.
 - D13.** [Double rotation.] Set $P \leftarrow \text{LINK}(a_k, R)$, **LINK**(a_k, R) $\leftarrow \text{LINK}(-a_k, P)$, **LINK**($-a_k, P$) $\leftarrow R$, **LINK**($-a_k, S$) $\leftarrow \text{LINK}(a_k, P)$, **LINK**(a_k, P) $\leftarrow S$. Update balance factors as follows:
 - i) If **B**(P) = $-a_k$, set **B**(S) $\leftarrow a_k$ and **B**(R) $\leftarrow 0$.
 - ii) If **B**(P) = 0 , set **B**(S) $\leftarrow \text{B}(R) \leftarrow 0$.
 - iii) Otherwise, **B**(P) = a_k . Set **B**(S) $\leftarrow 0$ and **B**(R) $\leftarrow -a_k$.
- Finally, set **B**(P) $\leftarrow 0$ and **LINK**(a_{k-1}, P_{k-1}) $\leftarrow P$, and go to D10. ■