An Iterative Algorithm for Emptying a Binary Tree

Farooq Mela <crazycoder@crazycoder.org>

Algorithm E (Empty Binary Tree). Given a set of nodes which form a binary tree $T$, this algorithm will empty the entire tree and free its storage. Each node is assumed to contain LEFT, RIGHT, and PARENT fields. LEFT and RIGHT are pointers to the node’s left and right subtree, respectively, and PARENT is a pointer to the node of which it is a child. Any of these three fields may be $\Lambda$, which for LEFT and RIGHT indicates that there is no left or right subtree, respectively, and for PARENT indicates that is the that the node is root of the tree. The tree has a field ROOT which is a pointer to the root node of the tree.

This algorithm makes use of two pointer-to-nodes $N$ and $P$.

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.

1. [Initialize] Set $N \leftarrow \text{ROOT}(T)$.
2. [Are we done yet?] If $N = \Lambda$, set $\text{ROOT}(T) \leftarrow \Lambda$. The tree is now empty and the algorithm terminates. Otherwise, set $P \leftarrow \text{PARENT}(N)$.
3. [Can we move left?] If $\text{LEFT}(N) \neq \Lambda$, set $N \leftarrow \text{LEFT}(N)$, and go to step E2.
4. [Can we move right?] If $\text{RIGHT}(N) \neq \Lambda$, set $N \leftarrow \text{RIGHT}(N)$, and go to step E2.
5. [Release node] $N$ is now a leaf; release its storage. If $\text{RIGHT}(P) = N$, set $\text{RIGHT}(P) \leftarrow \Lambda$. Otherwise, set $\text{LEFT}(P) \leftarrow \Lambda$.
6. [Move up] Set $N \leftarrow P$, and go to step E2.

Notes on Algorithm E

1. This algorithm runs in $\Theta(n)$ time and makes use of constant space. It does not make use of recursion, but it is still a simple and elegant algorithm. These properties make it superior to algorithms which use an auxiliary stack or recurse to perform the same operation.