Matthew Bramer

February 10, 2022

CS-260 Data Structures and Algorithms

6-1 Programming Journal

Binary tree algorithms can be complicated: Are they worth the effort? Recall the characteristics of sort algorithms that use a binary “divide and conquer” technique.

* Given a key, a ***search*** algorithm returns the first node found matching that key or returns 0 if a matching node is not found. A simple BST search algorithm checks the current node (initially the tree's root), returning that node as a match, else assigning the current node with the left (if key is less) or right (if key is greater) child and repeating. If such a child is 0, the algorithm returns 0 (matching node not found).

If each node of a tree is large (i.e., contains a large amount of data), then available memory may be a limiting factor. What can be done to allow an extremely large tree with hundreds of thousands (or even millions) of nodes to be built and managed?

* Recall that a tree's ***height*** is the maximum edges from the root to any leaf. (Thus, a one-node tree has height 0.)
* Searching a BST is fast if the tree's height is near the minimum. Inserting items in random order naturally keeps a BST's height near the minimum. In contrast, inserting items in nearly-sorted order leads to a nearly-maximum tree height.