BinarySearchTrees.mkr Page 541 Saturday, March 27, 1999 2:53 AM

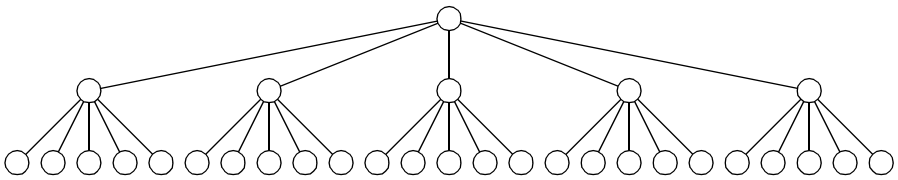

Figure 18.71 5-ary tree of 31 nodes has only three levels

An *M-ary search tree* allows *M*-way branching. As branching increases, the depth decreases.

We want to reduce the number of disk accesses to a very small constant, such as three or four. We are willing to write complicated code to do this because machine instructions are essentially free, so long as we are not ridiculously unreasonable. It should probably be clear that a binary search tree will not work, since the typical red-black tree is close to optimal height. We cannot go below $\log N$ using a binary search tree. The solution is intuitively simple: If we have more branching, we have less height. Thus, while a perfect binary tree of 31 nodes has five levels, a 5-ary tree of 31 nodes has only three levels, as shown in Figure 18.71. An *M-ary search tree* allows *M*-way branching. As branching increases, the depth decreases. Whereas a complete binary tree has height that is roughly $\log_2 N$, a complete *M*-ary tree has height that is roughly $\log_M N$.

We can create an *M*-ary search tree in much the same way as a binary search tree. In a binary search tree, we need one key to decide which of two branches to take. In an *M*-ary search tree, we need $M - 1$ keys to decide which branch to take. To make this scheme efficient in the worst case, we need to ensure that the *M*-ary search tree is balanced in some way. Otherwise, like a binary search tree, it could degenerate into a linked list. Actually, we want an even more restrictive balancing condition. That is, we do not want an *M*-ary search tree to degenerate to even a binary search tree because then we would be stuck with $\log N$ accesses.

The *B-tree* is the most popular data structure for disk-bound searching.

One way to implement this is to use a *B-tree*. The basic B-tree[1] is described here. Many variations and improvements are known, and an implementation is somewhat complex because there are quite a few cases. However, it is easy to see that in principle a B-tree guarantees only a few disk accesses.

The B-tree has five structure properties.

A B-tree of order *M* is an *M*-ary tree with the following properties:[2]

1. The data items are stored at leaves.
2. The nonleaf nodes store up to $M - 1$ keys to guide the searching; key i represents the smallest key in subtree $i + 1$.
3. The root is either a leaf or has between two and *M* children.
4. All nonleaf nodes (except the root) have between $\lceil M/2 \rceil$ and *M* children.
5. All leaves are at the same depth and have between $\lceil L/2 \rceil$ and *L* data items, for some *L* (the determination of *L* is described shortly).

1. What is described is popularly known as a B^+ tree.
2. Rules 3 and 5 must be relaxed for the first *L* insertions.