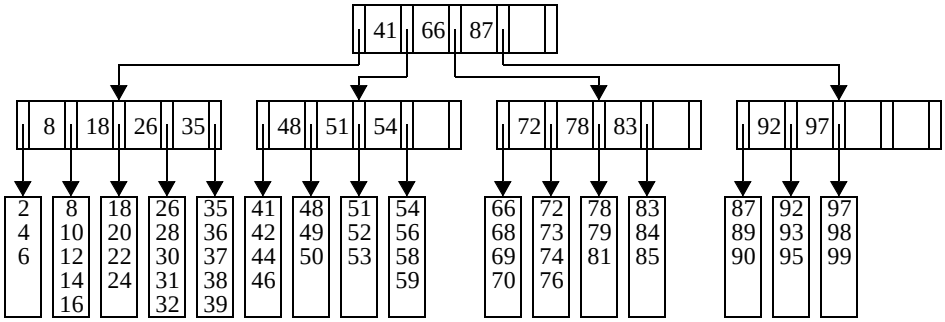

Figure 18.72 B-tree of order 5

Nodes must be half full. This guarantees that the tree does not degenerate into a simple binary or ternary tree.

An example of a B-tree of order 5 is shown in Figure 18.72. Notice that all nonleaf nodes have between three and five children (and thus between two and four keys); the root could possibly have only two children. Here, we have $L = 5$. It happens that L and M are the same in this example, but this is not necessary. Since L is 5, each leaf has between three and five data items. Requiring nodes to be half full guarantees that the B-tree does not degenerate into a simple binary or ternary tree. Although there are various definitions of B-trees that change this structure, mostly in minor ways, this definition is one of the popular forms.

We choose the maximum M and L that allow a node to fit in one disk block.

Each node represents a disk block, so we choose M and L based on the size of the items that are being stored. As an example, suppose one block holds 8,192 bytes. In our Florida example, each key uses 32 bytes. In a B-tree of order M, we would have $M - 1$ keys, for a total of $32M - 32$ bytes, plus M branches. Since each branch is essentially a number of another disk block, we can assume that a branch is 4 bytes. Thus the branches use $4M$ bytes. The total memory requirement for a nonleaf node is thus $36M - 32$. The largest value of M for which this is no more than 8,192 is 228. Thus we would choose $M = 228$. Since each data record is 256 bytes, we would be able to fit 32 records in a block. Thus we would choose $L = 32$. We are guaranteed that each leaf has between 16 and 32 data records and that each internal node (except the root) branches in at least 114 ways. Since there are 10,000,000 records, there are at most 625,000 leaves. Consequently, in the worst case, leaves would be on level 4. In more concrete terms, the worst-case number of accesses is given by approximately $\log_{M/2} N$, give or take 1. (For example, the root and the next level could be cached in main memory, so over the long run disk accesses would be needed only for level 3 and deeper.)

The remaining issue is how to add and remove items from the B-tree. The ideas involved are sketched next. Note that many of the themes seen before recur.

If the leaf contains room for a new item, we insert it and are done.

We begin by examining insertion. Suppose we want to insert 57 into the B-tree in Figure 18.72. A search down the tree reveals that it is not already in the tree. We can add it to the leaf as a fifth child. Note that we may have to reorganize all the data in the leaf to do this. However, the cost of doing this is negligible when compared to that of the disk access, which in this case also includes a disk write.