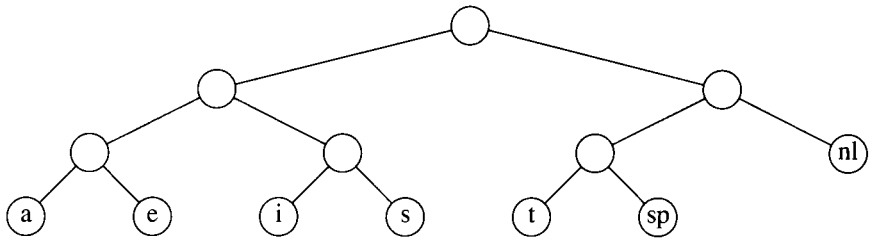

Figure 10.10 A slightly better tree

data structure is sometimes referred to as a *trie*. If character c_i is at depth d_i and occurs f_i times, then the *cost* of the code is equal to $\sum d_i f_i$.

A better code than the one given in Figure 10.9 can be obtained by noticing that the *newline* is an only child. By placing the *newline* symbol one level higher at its parent, we obtain the new tree in Figure 10.10. This new tree has cost of 173, but is still far from optimal.

Notice that the tree in Figure 10.10 is a *full tree:* All nodes either are leaves or have two children. An optimal code will always have this property, since otherwise, as we have already seen, nodes with only one child could move up a level.

If the characters are placed only at the leaves, any sequence of bits can always be decoded unambiguously. For instance, suppose the encoded string is 0100111100010110001000111. 0 is not a character code, 01 is not a character code, but 010 represents *i*, so the first character is *i*. Then 011 follows, giving a *t*. Then 11 follows, which is a *newline*. The remainder of the code is *a*, *space*, *t*, *i*, *e*, and *newline*. Thus, it does not matter if the character codes are different lengths, as long as no character code is a prefix of another character code. Such an encoding is known as a *prefix code*. Conversely, if a character is contained in a nonleaf node, it is no longer possible to guarantee that the decoding will be unambiguous.

Putting these facts together, we see that our basic problem is to find the full binary tree of minimum total cost (as defined above), where all characters are contained in the leaves. The tree in Figure 10.11 shows the optimal tree for our sample alphabet. As can be seen in Figure 10.12, this code uses only 146 bits.

Figure 10.11 Optimal prefix code