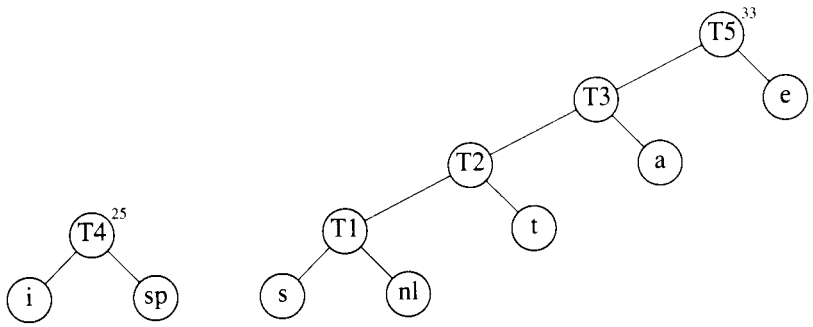

Figure 10.18 Huffman's algorithm after the fifth merge

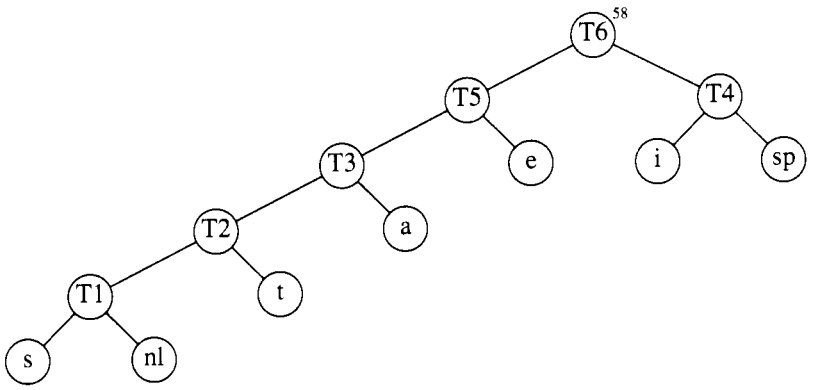

Figure 10.19 Huffman's algorithm after the final merge

We will sketch the ideas involved in proving that Huffman's algorithm yields an optimal code; we will leave the details as an exercise. First, it is not hard to show by contradiction that the tree must be full, since we have already seen how a tree that is not full is improved.

Next, we must show that the two least frequent characters α and β must be the two deepest nodes (although other nodes may be as deep). Again, this is easy to show by contradiction, since if either α or β is not a deepest node, then there must be some γ that is (recall that the tree is full). If α is less frequent than γ, then we can improve the cost by swapping them in the tree.

We can then argue that the characters in any two nodes at the same depth can be swapped without affecting optimality. This shows that an optimal tree can always be found that contains the two least frequent symbols as siblings; thus the first step is not a mistake.

The proof can be completed by using an induction argument. As trees are merged, we consider the new character set to be the characters in the roots. Thus, in our example, after four merges, we can view the character set as consisting of e and the metacharacters $T3$ and $T4$. This is probably the trickiest part of the proof; you are urged to fill in all of the details.