

Character	Code	Frequency	Total Bits
<i>a</i>	000	10	30
<i>e</i>	001	15	45
<i>i</i>	010	12	36
<i>s</i>	011	3	9
<i>t</i>	100	4	12
<i>space</i>	101	13	39
<i>newline</i>	110	1	3
Total			174

Figure 10.8 Using a standard coding scheme

i's, three *s*'s, four *t*'s, thirteen blanks, and one *newline*. As the table in Figure 10.8 shows, this file requires 174 bits to represent, since there are 58 characters and each character requires three bits.

In real life, files can be quite large. Many of the very large files are output of some program and there is usually a big disparity between the most frequent and least frequent characters. For instance, many large data files have an inordinately large amount of digits, blanks, and *newlines*, but few *q*'s and *x*'s. We might be interested in reducing the file size in the case where we are transmitting it over a slow phone line. Also, since on virtually every machine disk space is precious, one might wonder if it would be possible to provide a better code and reduce the total number of bits required.

The answer is that this is possible, and a simple strategy achieves 25 percent savings on typical large files and as much as 50 to 60 percent savings on many large data files. The general strategy is to allow the code length to vary from character to character and to ensure that the frequently occurring characters have short codes. Notice that if all the characters occur with the same frequency, then there are not likely to be any savings.

The binary code that represents the alphabet can be represented by the binary tree shown in Figure 10.9.

The tree in Figure 10.9 has data only at the leaves. The representation of each character can be found by starting at the root and recording the path, using a 0 to indicate the left branch and a 1 to indicate the right branch. For instance, *s* is reached by going left, then right, and finally right. This is encoded as 011. This

Figure 10.9 Representation of the original code in a tree

