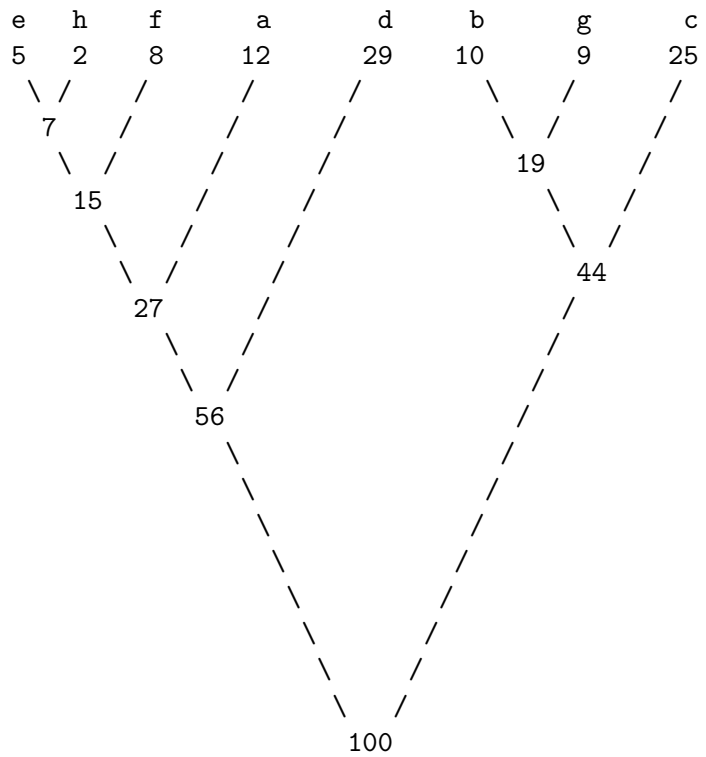
Huffman Compression: An Example

Suppose that we have a bit string x that is to be compressed. Suppose, further, that we decide to view x as being composed of 3-bit blocks. (For convenience, we assume that the length of x is evenly divisible by three.) We interpret each block as a single character. (Under this scenario, there are eight characters in the *alphabet*, one for each of the distinct bit strings of length three.) The frequency with which each of the characters occurs in x is given in the following table. (For convenience, we refer to the eight characters as $a, b, \ldots, h$, as indicated in the table.)

Character	Frequency	Character	Frequency
a (000)	12%	e (100)	5%
b (001)	10%	f (101)	8%
c (010)	25%	g (110)	9%
d (011)	29%	h (111)	2%

(a) Construct a Huffman tree corresponding to these frequencies.

Solution:



(b) Show the binary encoding of each character (consistent with the tree constructed in (a)), i.e., show the resulting Huffman code. **Note:** In order to ensure that the correct answer is

unique, follow this rule in using the tree constructed in (a) to determine the encoding of each character: From each non-leaf node there are two edges connecting it to its children; one edge is to be labeled 0 and the other 1. Use 0 to label the edge that goes to the subtree having the leaf with the alphabetically smallest label (among all labels in the two subtrees). For example, if a node's two children are the roots of subtrees whose leaves are labeled by c, d, and e and by a and f, respectively, then the edge into the latter subtree should be labeled 0, because a is the smallest among all the labels in the two subtrees.

Solution:

```
a : 000          e : 00100
b : 100          f : 0011
c : 11           g : 101
d : 01           h : 00101
```

(c) Compute the ratio between the lengths of the compressed text and the original. Show your work. Keep in mind that three bits were used to store each symbol in the original bit string, whereas the number of bits used, on average, for storing a symbol in the compressed version is the sum

$$\sum_{x \in \{a,b,\ldots,h\}} len(x) \cdot freq(x)$$

where $len(x)$ is the length (in bits) of the code for x and $freq(x)$ is the frequency with which x occurs in the original file.

Solution:

$$\begin{aligned}
& \textstyle\sum_{x \in \{a,b,\ldots,h\}} len(x) \cdot freq(x) \\
= \;& (3)(.12) + (3)(.1) + (2)(.25) + (2)(.29) + (5)(.05) + (4)(.08) + (3)(.09) + (5)(.02) \\
= \;& (2)(.25 + .29) + (3)(.12 + .1 + .09) + (4)(.08) + (5)(.05 + .02) \\
= \;& (2)(.54) + (3)(.31) + (4)(.08) + (5)(.07) \\
= \;& 1.08 + 0.93 + 0.32 + 0.35 \\
= \;& 2.68
\end{aligned}$$

The ratio is thus 2.68/3.0, which is slightly less than 90%. Thus, not much compression was realized. Huffman coding typically works much better when a larger alphabet is chosen, such as when the file is interpreted as being composed of 8-bit blocks (which makes for an alphabet of 256 symbols) or when (in the case of a file of text) the alphabet is taken to consist of all the words (e.g., *the*, *cat*, *computer*), punctuation symbols (e.g., ",", "."), and whitespace characters (e.g., space, tab, newline) occurring in the text.