Huffman Code

Computer Science Society
Programming Contest
Fall 2008

We know that characters are represented in binary using fixed-length codes (e.g. 7-bit ASCII or 16-bit UNICODE). Since some characters occur in text more frequently than others, can't we save bits in the long run if we use shorter codes for frequent characters and longer codes for infrequent characters? Yes; however, if one character's code is a prefix of another character's code (e.g. `a`=`010`, `b`=`01011`, `c`=`11`), there's potential ambiguity about where one character ends and the next starts in a string (e.g. whether `01011` encodes the string `b` or the string `ac`). If character codes are *prefix-free*—no code is a prefix of any other code—then no such ambiguity occurs. In 1952, David Huffman discovered an algorithm for producing variable-length prefix-free character codes of minimum average length, based on the frequency of characters in input text.

A *Huffman tree* is a binary tree whose leaves are labeled by characters, and whose subtrees have weights representing character frequencies. The *Huffman code* of a character labeling a leaf is obtained by traversing the path from the root to the leaf, writing `0` or `1` when traversing a left or right branch, respectively. To create a Huffman tree from input text, initialize a set containing one-node (i.e. single-leaf) trees for each distinct character in the input text, with weight equal to the frequency of that character in the text. As long as there is more than one tree in the set, repeatedly do the following:

1. Remove the smallest tree t_1 and next smallest tree t_2 from the set. Compare trees numerically by weight; if the weights are equal, compare them alphabetically by least character labeling a leaf.
2. Construct a new tree t with left subtree t_1, right subtree t_2, and weight equal to the sum of the weights of t_1 and t_2.
3. Add t to the set.

The resulting Huffman tree has a leaf for each distinct character in the input text, from which the characters' Huffman codes can be obtained.

Input Format

The input text consists of a stream of ASCII characters. Only the graphical characters (with ASCII codes between 32 and 126 inclusive) are counted; all others (e.g. tabs, newlines and carriage returns) are ignored.

Output Format

Produce a Huffman code for each graphical ASCII character, based on the frequency they occur in the input text. Output a line containing the character, its Huffman code, and its frequency in parentheses. Sort the lines of output alphabetically by Huffman code.

Input Sample

```
Mississippi has a number of
repeated letters.
```

Output Sample

```
i 000 (4)
l 00100 (1)
m 00101 (1)
n 00110 (1)
o 00111 (1)
  010 (5)
u 01100 (1)
. 011010 (1)
M 011011 (1)
a 0111 (3)
e 100 (6)
p 1010 (3)
r 1011 (3)
s 110 (6)
t 1110 (3)
b 111100 (1)
d 111101 (1)
f 111110 (1)
h 111111 (1)
```