**Huffman Coding**

In 1952, David Huffman created an optimal compression algorithm. For text, the algorithm assigns binary codes (0 and 1) to letters so that the most frequently occurring letters have the shortest codes. This typically results in a 20 to 90% reduction of the file size. Huffman compression is used in a wide variety of practical applications, including .zip files, .jpegs, fax machines, computer networks, and high-definition television.

|  |  |
| --- | --- |
| T | 2 |
| J | 1 |
| H | 1 |
| S | 3 |

As an example, let's compress the string "TJHSSTS" by Huffman coding.

1. Make a frequency table of the letters. (What data structure will you use in your program?) Tree or Map?
2. For each letter, put the letter-frequency pair into a HuffmanTreeNode, which is like our ordinary TreeNode, but which has fields that store two items of data. HuffmanTreeNode also has a compareTo in which nodes with a lower frequency are “smaller than” nodes with a higher frequency. (Why is this important?)
3. Add each HuffmanTreeNode to a priority queue (or a min-heap). Process the priority queue to make a binary tree:

Repeat until one node is in the priority queue:

* 1. remove the two pairs with the lowest frequency.
  2. make them children of a new node, with a frequency equal to the sum of frequencies of the children. The "letter" of this third node can be "\*".
  3. put the new node back into the priority queue.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H:1 | J:1 | T:2 | S:3 |  | T:2 | \*:2 | S:3 |  | S:3 | \*:4 |  | \*:7 |
|  |  |  |  |  |  | / \  H:1 J:1 |  |  |  | / \  T:2 \*:2  / \  H:1 J:1 |  | / \  S:3 \*:4  / \  T:2 \*:2  / \  H:1 J:1 |

(Your specific tree may have the children in different places. Therefore, your binary code and scheme will be different from mine. But that's okay because your binary code matches your scheme and my binary code matches my scheme.)

1. Make a map to store the scheme. The map for the tree above is: {S=0, T=10, H=110, J=111}. You need a recursive helper method. Traverse the whole tree. When you reach a leaf, return its value and the path of zeros and ones. You build the path during the recursion, where going left concatenates a 0 and going right concatenates a 1.
2. Make the two files. Process the string "TJHSSTS" letter-by-letter and use the map to output the binary message.

1011011100100

T10

J110

H111

S0

Then process the map to output the scheme in the format that deHuffman expects.

**Assignment**

TJHSSTS

TJ

Huffman prompts the user to enter two strings, the message and the "middle part" of the filename.

Huffman creates and outputs two text files:

1. the message's Huffman binary code, saved as "message."+middlePart+".txt"

1011011100100

1. the Huffman coding scheme, saved as "scheme."+middlePart+".txt"

T10

J110

H111

S0

You may then use your deHuffman.java to recover the original message.