**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 codes are 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?)
2. For each letter, put the letter-frequency pair into a TreeNode which has been modified to store 2 items of data, as well as pointers to the left-child and right-child. Add each node to a priority queue (or a min-heap). The modified TreeNode class has to be written so that the letters with the lowest frequency are the first to be removed from the priority queue. (Why does the lowest frequency have to come out first?)
3. Use the priority queue of nodes to make the Huffman tree, as follows:

Repeat until one node is in the priority queue:

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

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

(Your specific tree may look a little differently. That's okay because of the way the tree is used to build the binary code.)

1. Read the string "TJHSSTS" letter-by-letter and search the tree for the letter. As you go, concatenate the path, where going left concatenates a 0 and going right concatenates a 1. This method is recursive, of course. When you search for "T", you return "10". "J" returns "110". "H" returns "111". "S" returns "0".



**Assignment**

Prompt the user to read two strings, the message and the "middle part". The program outputs two files, the message's Huffman binary code, saved as ("message."+middlePart+".txt") and its Huffman coding scheme,   
saved as ("scheme."+middlePart+".txt").

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