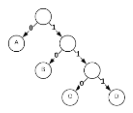


Decoding Huffman codes

In 1952 David Huffman discovered an optimal compression algorithm for data. The algorithm assigns binary codes to items of data and generates a unique coding scheme based on that data. After transmittal, the binary code and its coding scheme are decoded to get back the original message. Today's zip files, networks, TVs, and jpegs use variants of the Huffman compression scheme. In this first part of the lab, we will only decode a Huffman message. As an example, suppose the message is:

```
0101011111011111000001001100100111 00
```

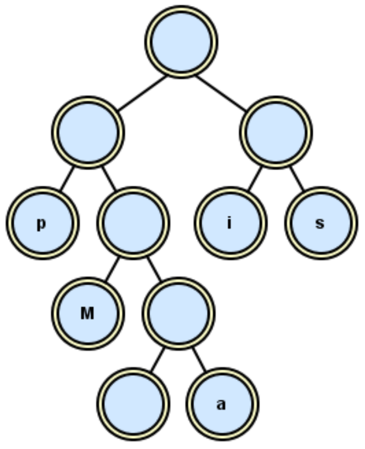
and its Huffman coding scheme is:

`<space>`	`0110`
`M`	`010`
`a`	`0111`
`i`	`10`
`p`	`00`
`s`	`11`

Then what is the decoded message? Try to figure it out before continuing.

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

In a computer, the Huffman coding scheme arrives as a text file from which you must construct a binary tree. The zeroes and ones describe the path to get to the leaf, where a "0" means a left branch and a "1" means a right branch. The tree's nodes all have values of `null`, except for each leaf, which has the value of the letter. The scheme above turns into a tree looking like this:



After the Huffman tree is built, the binary text is decoded by following the zeroes and ones to each leaf. As you can see for yourself, each path is unique, i.e., no character's code is a prefix of any other character's code.

Part 1: Assignment

Write the program **`deHuffman`** that prompts the user to read two files, the binary message and the Huffman coding scheme. Then decode and output the message. At the beginning, decode the two files **`message.maips.txt`** and **`scheme.maips.txt`** and see if your tree is like the tree above. Then decode the other text files in the folder.

Huffman Coding

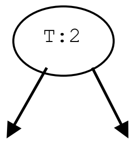
For text, the algorithm assigns binary codes (0 and 1) to letters so that the most frequently occurring letters have the shortest codes. This typically results in a 20-90% size reduction.

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

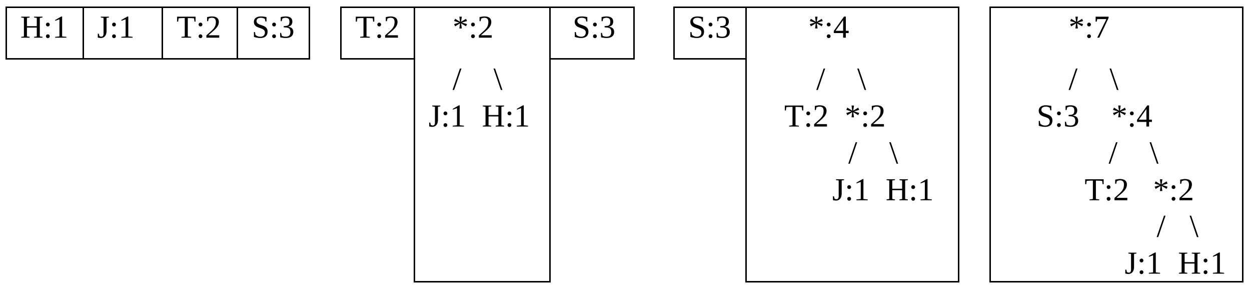
T	2
J	1
H	1
S	3

- Make a frequency table of the letters. What data structure will you use in your program?

- 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?



- Use the priority queue of nodes to make the Huffman tree, as follows:
  Repeat until one node is in the priority queue:
  - remove the two pairs with the lowest frequency.
  - 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 "*".
  - put the third node into the priority queue.



- Read the string "TJHSSTS" letter-by-letter and search the tree for the letter. As you go, build the path, where going left is 0 and going right is 1. This method is recursive, of course. I found the **`find`** method in Binary Search Tree to be helpful. When you search for "T", you get "10". "J" returns "110". "H" returns "111". "S" returns "0".

Part 2: 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.
Lastly, report how much the original message was compressed as a percentage.

```
1011011100100

T10
J110
H111
S0
```