Programming Assignment #2

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

Problem Description

Ella is fed up with life on this planet, and decides she is going to move to Mars. Living in Mars generally seems amazing, except for one thing: Exchanging messages with Earth is very expensive; sending a binary sequence of length n, costs n*0.5 USD. Since Ella wants to communicate with her friends on Earth as frequently as possible, she wants to figure out a way to encode her messages character-by-character, so that the encoding is lossless, as short as possible, and prefix-free. Ella knows you are an algorithms mastermind, and turns to you for help. You claim that, if she provides you with the frequency, for each of her friends, of each character in their messaging history so far, you will be able to produce the best such way to encode her messages to that friend. Moreover, if she gives you an encoded message and the frequencies that produced it, you can decode the message to human-readable form.

What does the program do

The overall function of the program is the following: The inputs are (a) a flag that takes either value `-en` or `-de`, representing "encode" and "decode" respectively, (b) a file `Name.txt` containing, for a specific friend called `Name`, the frequency of each character in your messaging history with `Name` so far, and (c) another file called `message.txt`, that either contains a human-readable or a binary message . In both cases, the Huffman tree for the given frequencies is computed. Then, according to the Huffman Tree, and depending to the value of the flag, it outputs either the optimal binary encoding (as a string of 0 and 1) of the human-readable `message.txt`, or the decoding of the binary `message.txt`. You will only need to implement part of the code; the three methods described in Parts 2, 3 and 4.

Part 1: Report [20 points]

Write a short report that includes the following information:

(a) Provide and explain the complexity of the first method you implemented.

(b) Provide and explain the complexity of the second method you implemented.

(c) Provide and explain the complexity of the third method you implemented.

(d) Prove that your encoding is prefix-free.

Part 2: First Method - Create a Huffman tree [40 points]

The first of methods you need to implement is called `constructHuffmanTree()`. The input is an ArrayList called `characters` of length k containing characters, and an ArrayList called `freq` of length k containing non-negative integer values, s.t. `frequencies[i]==m` means that `characters[i]` appears m times in your messaging history with `Name`. The output should be the Huffman Tree for those frequencies (your method will be setting the root field of the object calling the method to the root of the Huffman Tree, from which of course the entire tree can be accessed). For this, you will use the TreeNode class provided. You may not make any modifications to it. The Huffman tree produced must have k leaf nodes, one for each character, and the value field of the leaf node corresponding to `characters[i]` must be `characters[i]`.

Part 3: Second Method - Find the encoding [20 points]
The second method is called `encode()`. The input is a `String` containing a human-readable message. The return value should be a `String` containing the binary encoding of the message, encoded according to the Huffman tree stored in the object calling the method. For example, if the Huffman Tree has encoded character h as 00 and character i as 01, if the input is hi the output should be a String 0001, representing the binary value 0001.

Part 4: Third Method - Find the decoding [20 points]
The third method is called `decode()`. The input is a `String` containing a binary message. The return value should be a `String` containing the human-readable decoding of the message, decoded according to the Huffman tree stored in the object calling the method.

Note: In real life the output of the encoding would be a binary file rather than a text file consisting of "0"'s and "1"'s. Thus looking at the file size of an encoded message does not give out the true compression factor. The choice of dealing with strings "0", "1", rather than binary is in order to ease debugging and grading.

Instructions

- Carefully study the code provided to you, and ensure that you understand it before starting to code your solution. The set of provided files should compile and, if given input files, run successfully before you modify them. If not, there is probably a problem with your Java configuration.
- Your job is to fill in the methods `constructHuffmanTree()`, `encode()` and `decode()` in `HuffManTree.java` so that they have the required functionality.
- You can modify `Main.java` and `TreeNode.java` if it helps you to test or debug your program, but we will be running your code according to the original `Main.java` and `TreeNode.java`.
- `Main.java` is the main driver program. Use command line arguments to specify the flag and the two input files. An example run would be:
  `Main -de John.txt binaryMessage.txt`.

- Do not add a package to your code. If you have a line of code at the top of your Java file that says `package <some package name>;` that is wrong.
- We will be checking programming style. A penalty of up to 10 points will be given for poor programming practices (e.g. do not name your variables foo1, foo2, int1, and int2).
- Before you submit, be sure to turn your report into a PDF and name your PDF file `eid_lastname_firstname.pdf`.

What To Submit

You should submit a single zip file titled `eid_lastname_firstname.zip` that contains:

- `HuffmanEncoder.java`, as well as any extra `.java` source files you created.
- Your report in pdf form, named: `eid_lastname_firstname.pdf`.

Do not put these files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your solution must be submitted via Canvas BEFORE 11:59pm on the day it is due.