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.

What does the program do

The overall function of the program is the following: Given as input a file `Name.txt` containing, for a specific friend called `Name`, the frequency of each character in your messaging history so far, it outputs the optimal binary encoding for each character. You will only need to implement part of the code; the two methods described in Part 3 and Part 4.

Part 1: Background [∞ points]

Before starting to code the respective methods, make sure you have a great understanding of:

- The DFS algorithm.
- Huffman Encoding.

Part 2: Report [30 points]

Write a short report that includes the following information:

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

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

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

(d) Prove that yours is the optimal character-by-character, lossless, prefix-free encoding.

Part 3: First Method - Create a Huffman tree [35 points]

The first of the two methods you need to implement is located in `Program2.java` and is called `computeEncodingTree()`. The input is an ArrayList called `frequencies` of length k containing non-negative values, s.t. `frequencies[i]=m` means that the character with index i appears m times in your messaging history with `Name`. The output should be the the Huffman Tree for those frequencies.

Part 4: Second Method - Find the encoding [35 points]

The second method, also located in `Program2.java`, is called `computeEncoding()`. The input is a Huffman Tree and an `ArrayList` of length k initialized with `""` values. The output should be the `ArrayList` given as input, populated with the huffman encodings, s.t. index i stores the binary encoding for the character with index i.

How-To Encode the Tree

The output for Part 3 and the input for Part 4 is a tree, but as you may know there are many ways to store a directed tree. In the way you will be using, you create an object for each node, with the object having at least three fields: `nodeIndex`, `parent`, and `children`. In this format, if one is given the root, they can access all the nodes of the tree. Thus, the output of your Part 3 method, and the input of your Part 4 method, should be the root of a Huffman Tree.

Instructions

- Download and import the code into your favorite development environment. We will be grading in Java 1.8. **It is YOUR responsibility to ensure that your solution compiles with the standard Java 1.8 configuration (JDK 8).** For most (if not all) students this should not be a problem. If you have doubts, email a TA or post your question on Piazza.
- 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 an input history file, 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 `computeEncodingTree()` and `computeEncoding()` in `Program2.java` so that they have the required functionality. Towards this, you may edit `Node.java` to include extra fields and methods that you find helpful.
- You can modify `Driver2.java` if it helps you to test or debug your program, as we will not grade the contents of `Driver2.java`.
- `Driver2.java` is the main driver program. Use command line arguments to specify an input file.
- 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:

- `Program2.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.