**CMSC 132 -- Project 7**

**Huffman Tree Encoding/Decoding**

**Project Due: Sunday 4/28 at 11:00 PM**

## Closed Policy

**This is a closed project.** You are expected to do all of the work on this project without consulting with anyone other than the CMSC 132 instructors and TAs.  Treat this project as though it were a take home exam.  If you have any questions about whether or not a certain topic is okay to discuss with classmates or friends, please ask your instructor.

## Goals

The goals of this project are:

* Practice Huffman Trees
* Practice polymorphism
* Practice recursion

## What's a Huffman Tree?

A Huffman Tree is a special type of binary tree used for data compression.  A small Huffman Tree appears below:



Each leaf in the tree contains a symbol (in this case a Character) and an integer value.  Each non-leaf (internal node) contains just references to its left and right children.  The 0's and 1's that you see are not stored anywhere, we just mentally associate 0 with any left branch and 1 with any right branch.

Each character that appears in the tree is assigned a unique code (a sequence of 0's and 1's) obtained by following the path from the root of the tree to the leaf containing that character.   Below are the codes for the four characters found in this sample tree:

* e is coded by "0"
* b is coded by "100"
* c is coded by "101"
* d is coded by "11"

### Encoding

A sequence of characters can be "encoded" into a String of 0's and 1's by using the codings as described above.  For example, the String "eddbc" would be encoded into the sequence "01111100101".  (That's good compression -- we went from 5 characters down to 11 bits.)

### Decoding

A sequence of 0's and 1's can be "decoded" into a String of Characters by using the codings backwards.  For example, the sequence "11010111" could be decoded into the String "decd".

### Constructing a Huffman Tree from a Stream of Characters

Given a "sample" stream of Characters, one can create a Huffman Tree which will do a very good job of compressing other streams whose characters share the same relative frequency as those in the sample stream used to create the Tree.

1.  The first step is to count the frequency of each character in the "sample" stream of Characters.  Suppose your sample stream is:

"eebbeecdebeeebecceeeddebbbeceedebeeddeeeecceeeedeeedeeebeedeceedebeeedeceeedebee"

Below are the counts for each character that occurs in the stream:

* 49 e's
* 11 b's
* 8 c's
* 12 d's

2.  In this example, the next step is to create four little trees -- one for each character -- each consisting of just a single leaf.  These little trees should be put into some kind of list.  Below is a picture of the list of four little trees:

            

3.  The next step is to select the two trees with the SMALLEST frequencies.  In this example, that would be 8 and 11.  Remove them from the list of trees, and then join them as children of a new node to make a new tree.  Then add this new tree to the list of trees.  Below is the list of trees after this step has been accomplished.  There are now three trees in the list.

      

The "frequency" of a tree that is more than just a single leaf is the sum of the frequencies of all leaves below it.  So the frequency of the first tree above is 19, the sum of it's leaves.

Now we repeat step 3.  (This time, the two smallest frequencies are 19 and 12).  Below is the list of trees after we join them.  (We're down to just TWO trees in the list now).

   

Now we repeat step 3 again.  (This time, the two smallest frequencies are obvious -- 31 and 49).  Below is the list of trees after we join them.  (Just one tree now.)



Since the list now contains just one tree, we are done -- this is the Huffman Tree generated by the "sample" sequence of characters.

## Project Overview

You will write a program that will first construct a Huffman Tree from a sequence of Characters.  Each leaf at the bottom of the tree will represent just a single Character.  We will allow letters, digits, punctuation, and any other type of Character.

Once the Huffman Tree has been built, your program will be able to do two things:

* "encode" a sequence of Characters into a String of 0's and 1's using the Huffman Tree
* "decode" a sequence of 0's and 1's into a String of Characters using the Huffman Tree

## Efficiency Requirement

You may be penalized if your program performs too slowly.  If your implementation takes less than 10 times as long as Fawzi's to get through each release test, then you will be fine.  Otherwise, you will not pass the release tests.  (Fawzi's code is not particularly efficient, so as long as you don't do anything really inefficient, you should be okay.)

## Classes/Interfaces

**HuffmanTree** is an interface that will be implemented by two classes:  **HuffmanInternalNode** and **HuffmanLeaf.**

A **HuffmanLeaf** is a node at the bottom of the Tree.  It has two fields, listed below.  **You may not add any other fields to this class.**

* private Character data -- the Character that this leaf holds
* private int count -- the relative frequency of this character (how many times it occurred in the sequence of Characters used to generate the Huffman Tree.)

A **HuffmanInternalNode** is a node in the tree that is not a leaf.  It has two fields that are self-explanatory.  **You may not add any other fields to this class.**

* private HuffmanTree leftChild, rightChild;

There is a class called **HuffmanTools** that will contain the three static methods that make this project work.  You will be writing these methods.  They are:

* public static HuffmanTree **buildHuffmanTree**(Iterator symbols) -- The iterator "symbols" will provide a sequence of Character objects.  These are used to construct a Huffman Tree, which is returned by the method.
* public static String **encode**(HuffmanTree tree, Iterator symbols)  -- The first parameter (tree) is the root of an existing Huffman Tree.  The second parameter is an Iterator that will provide a sequence of Character objects.  The return value will be a String of '0' and '1' characters that represent the encoding of the Characters in the Iterator.  For example, if "tree" is the Huffman Tree from the example pictured above, and the Iterator gives you the sequence 'e', 'd', 'd', 'b', 'c' (five separate Character objects), then the return value would be the String "01111100101".  (These are not really bits, they are the characters '0' and '1'.)
* public static String **decode**(HuffmanTree tree, Iterator bits)  -- The first parameter is the root of a HuffmanTree.  The second parameter is an Iterator that will provide a sequence of Character objects that are all either '0' or '1'.  The return value will be a String of characters that represent the decoding of the sequence of 0's and 1's.  For example, if "tree" is the Huffman Tree from the example pictured above, and the Iterator gives you the sequence '1', '1', '0', '1', '0', '1', '1', '1' (eight separate Character objects), then the return value should be the String "decd".

There is a class called **InternalTools**, which you may ignore, but which you may find helpful while testing your project.  There are two public methods in there:

* public static Iterator **getCharacterIteratorFromString**(String s) -- You give it a string like "01111100101" or possibly "eddbc", and it will return an Iterator that will cycle through the characters in that String.  (See the file PublicTests.java for examples on how to use this method to test the project.)
* public static Iterator **getCharacterIteratorFromFile**(String fileName) -- You give it the name of a text file found in the project folder and it will return an Iterator that will cycle through the characters that were found in the file.  (You could use a big text file as the "sample" sequence of characters from which you  generate a great big Huffman Tree.)

## What YOU Will Implement

You will write most of the HuffmanInternalNode class, most of the HuffmanLeaf class, and almost all of the HuffmanTools class.  **Please see the comments in the file HuffmanTree.java for information about how to implement the methods of the HuffmanInternalNode and HuffmanLeaf classes.**

You should begin by first reading over the comments for the methods in the HuffmanTools class, and then the  comments for the HuffmanTree interface. This will give you a feel for how the project components fit together.  Then write the methods of the HuffmanLeaf class and the HuffmanInternalNode class.  Finally, write the methods of the HuffmanTools class.

## Grading

Your grade on this project will be calculated as follows:

* Public tests(3) -- 45%
* Release tests(3) -- 45%
* Style -- 10%

[Web Accessibility](https://www.umd.edu/web-accessibility)