Name: Shashank Sinha

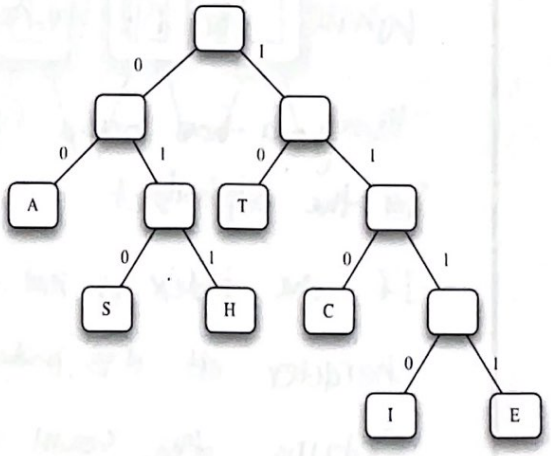
Name: Abhi Kulkarni

Lab - Huffman Encoding for Text Compression

A Huffman code is a variable length code that minimizes the length of the message. Given a string X defined over some alphabet, such as the Unicode character set, the goal is to efficiently encode X into a small binary string (using only the characters 0 and 1). The basic idea is that symbols that are frequently encountered in the message will be represented by short codes. On the other hand, infrequent symbols will have longer codes.

Text compression is useful in any situation where we wish to reduce bandwidth for digital communications, to minimize the time needed to transmit the text. Likewise, text compression is useful for storing large documents more efficiently, to allow a fixed-capacity storage device to contain as many documents as possible.

A binary tree represented the encoding of a string. Each edge of the tree represents a bit in a code-word, with an edge to a left child representing a "0" and an edge to a right child representing a '1". Each leaf is associated with a specific character., and the code-word for that character is defined by the sequence of bits associated with the edges in the path from the root of the tree to the leaf of the character.



The tree shown here was constructed from the message:

THE CAT IS THAT CAT

The code for each letter is read by Write the code for each letter in the message:

SYMBOL	CODE
A	00
C	110
E	1111
H	011
I	1110
S	010
T	10

Getting the Frequencies

The structure of the Huffman tree must depend on the ***frequencies*** of the letters, where we have, for each character *c*, a count *f(c)* of the number of times *c* appears in the string X. Therefore, the first task is to compile a count of each letter in the message. Consider the following message:

CHEESES CHEESES

T CLEESE SELLS THE CHEESES

How many times does each character appear in the message?

Complete the table.

SYMBOL	COUNT
L	3
C	4
E	14
H	4
S	9
T	2

Design an **algorithm** for a method that computes the frequency count for the letters in a message. The message will be stored in a file, so the method will have as input the name of the file. It will return an integer array of size 256 containing the frequencies of each letter of the alphabet. The method will iterate to get the characters one at a time from the file. **While the code is given here for the method header, for this lab you are to only provide the algorithm, not the code.**

```
/** Get the frequency of the characters */
  public static int[] getCharacterFrequency(String filename) {
```

Initialize array characterCount with size 256.

While the file hasnext, store each line in a string

Using a for loop, find the index of each character in the alphabet.

If the index is not -1, increment the count of that character at it's index

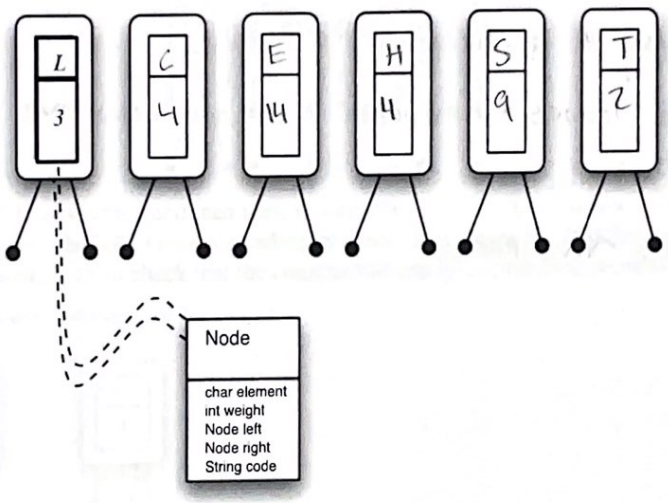
Return the count of each character.

Creating the Initial Forest of Trees

Once the counts have been done, the next step is to create a collection (forest) of Huffman trees. To start there will be one tree for each possible symbol. Each of the trees will have a single node, where the data portion contains the character and the frequency (designated as "weight") of the character. The frequency will be the count for that character found in the character frequency array.

Fill in the initial trees for the characters and counts that were found in the previous section.

Forest of Huffman trees




```
public class Node {
    char element;      // the character
    int weight;        // weight of this subtree
    Node left;         // left subtree
    Node right;        // right subtree
    String code = ""; // code from the root

    /** Create an empty node */
    public Node() {
    }

    /** Create a node with the weight and character */
    public Node(int weight, char element) {
      this.weight = weight;
      this.element = element;
    }
}
```

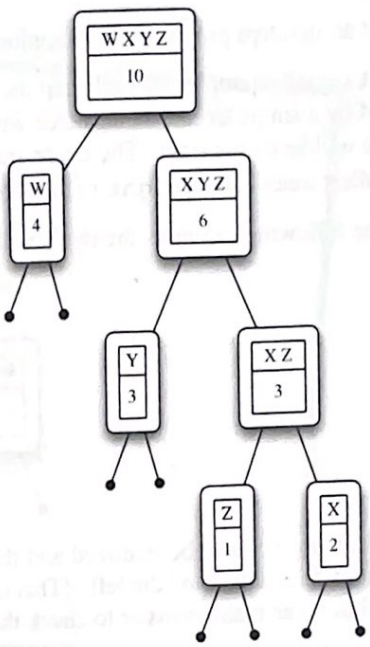
Design the ***algorithm*** for a method that creates the forest of trees given an array of the counts. These will be the leaves in the Huffman tree. The method header is:

```
/** Get a Huffman tree from the codes */
  public static Comparable<T> getHuffmanTree(int[] counts) {
```

- Create a new heap that uses <T>
- Create a for loop that will add a new leaf node whenever the frequency of a character is > 0.
- Using the lowest 2 nodes, make a new tree with their combined frequencies
- Remove the two nodes

Step 4: The final step is to combine the last two trees in the forest.

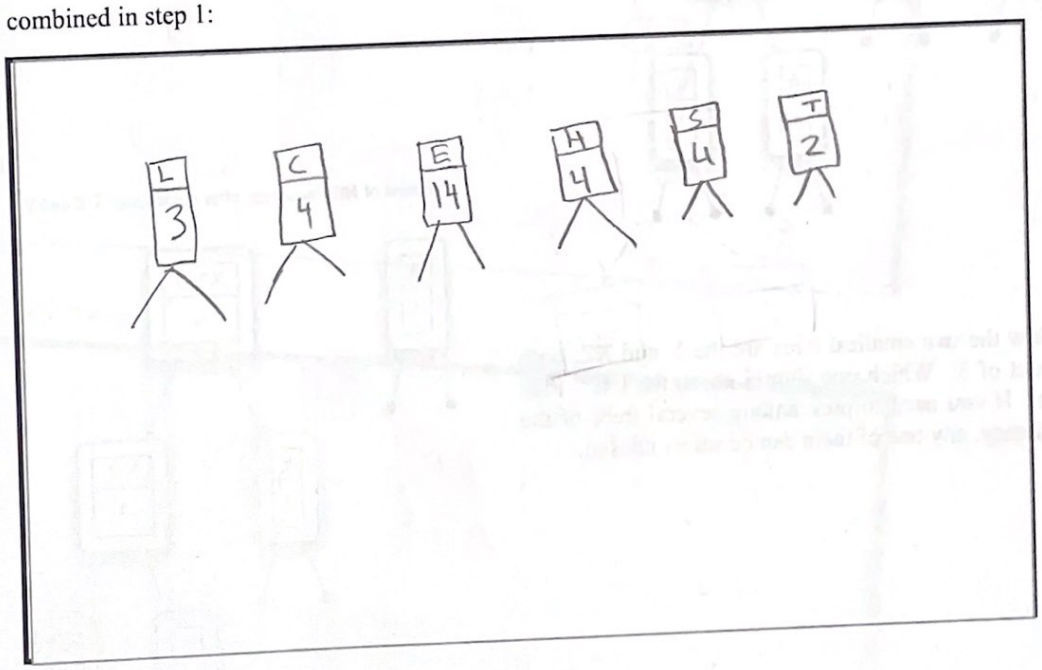
Final forest of Huffman trees



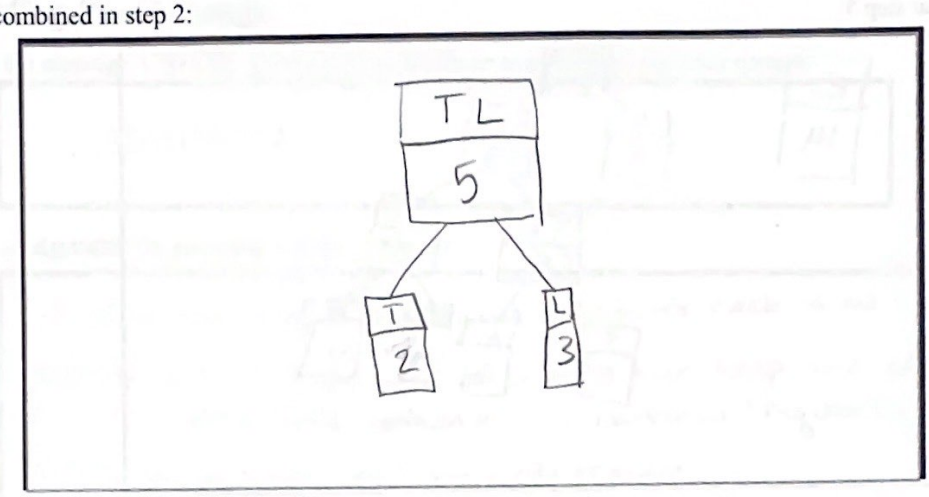
Practice

Looking back at the example forest of trees constructed on page 3, show the trees combined at each step and the final Huffman tree resulting from the initial forest that was constructed in the previous section.

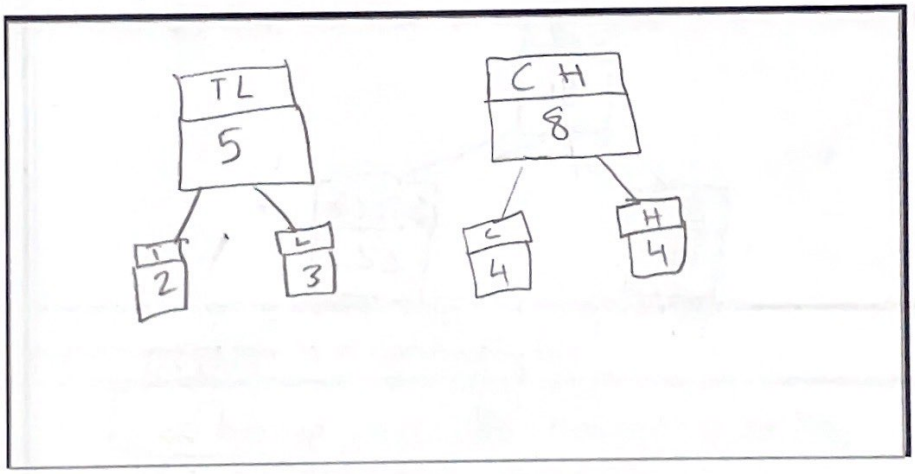
Trees combined in step 1:



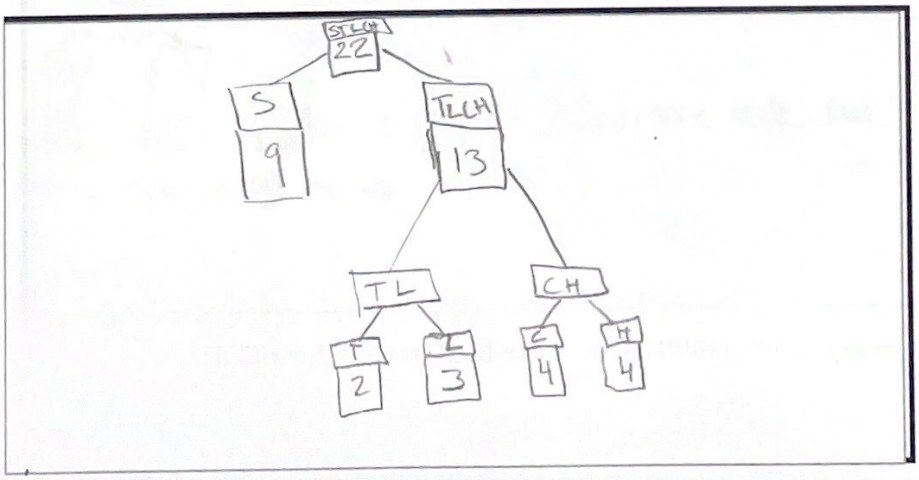
Trees combined in step 2:



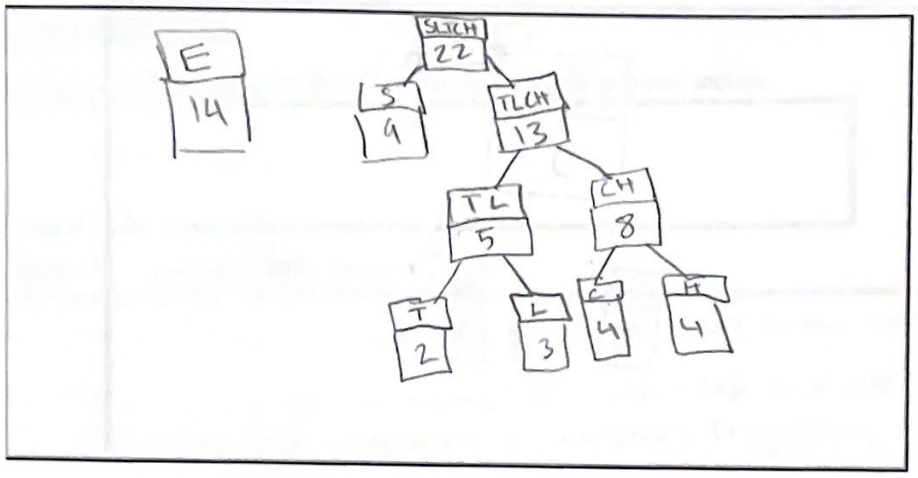
Trees combined in step 3:



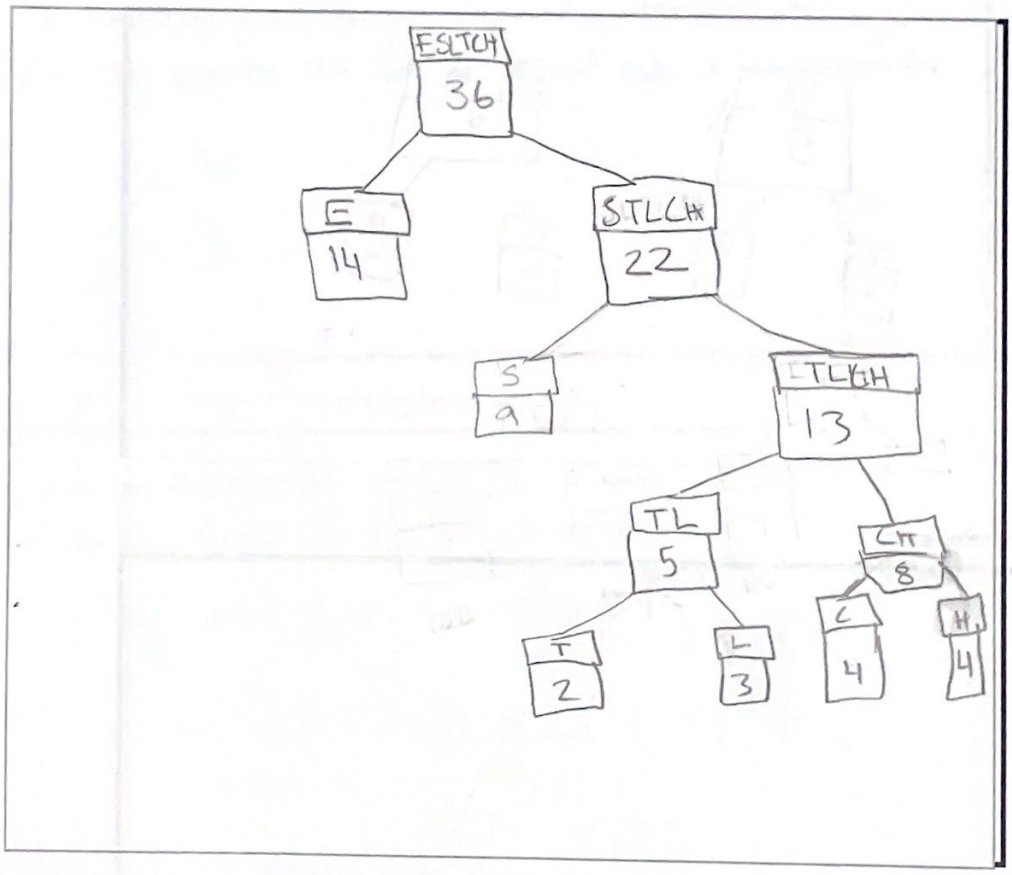
Trees combined in step 4:



Trees combined in step 5:



Final tree:



If it is correct, the message 111011110101100 will either decode to CHEST or HCEST.

Encoding the Message

Encode the message, CHEESE, using the final Huffman tree from the previous section.

111011100100

Design an algorithm for encoding a single character.

Build a min heap that contains a node for each of the characters
Extract 2 min. frequency nodes from min heap and add a new internal node containing the combined frequency
Keep doing that until one node remains
To encode for a single character, traverse through the tree from the root node to the leaf node of the character

Develop an algorithm for encoding a file and displaying the result.

Based on the number of ASCII characters in the file, count the frequency of each character.
Store the characters as tree nodes

Using their frequencies, create a new tree node that will be the parent of two nodes

Encode the file by traversing through it.