COMS 228: Introduction to Data Structures, Summer 2013
Homework Assignment 4: Part I

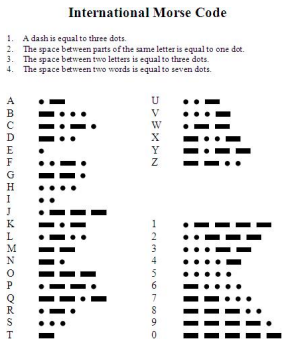
Assignment Overview

In an alternate universe of the movie *Cast Away*, Tom Hanks is stranded on an island with his friend Wilson, a laptop with Eclipse installed and a 4kB floppy disc. Tom decides to send a floppy disc in a bottle and hope that it reaches a society that still has floppy disc drives. Since Tom can only write messages using a very small floppy disc, he needs to encode his message so that it takes a very small amount of space. Your job is to help Tom by writing a system for encoding and decoding messages. In this assignment, you will get practice implementing binary trees, using priority queues, and the Collections and the Map interfaces.

Important Note: Each class file you write needs to be included in the package "edu.iastate.cs228.hw4". Your source files should be located in the directory "edu/iastate/cs228/hw4" as specified by your package name. When submitting your homework through Blackboard, please place the "edu" folder in a zip file. Make sure to include your last name in the name of your zip file.

How to Compress a Message

Tom Hanks is familiar with Morse Code[1], a system of dots and dashes used to represent letters and numbers.



However, converting a message to Morse Code will not give us the smallest message possible. We must build our own system for compressing a message using dots and dashes. Given a message string, we want to build an encoding system with the following properties:

[1] http://en.wikipedia.org/wiki/Morse_code

1. Each character can only be represented by a series of dots(*) and dashes(-)
2. The characters that occur the most often are represented with the least amount of dots and dashes
3. Each character must have a unique representation
4. Converting an English character to its coded representation must take O(1) time and vice versa

Our goal is to build a data structure called a *Morse Tree* that will enable us to obtain all these properties.

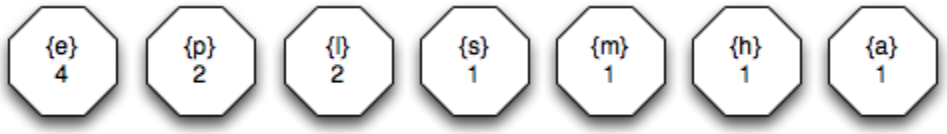
How To Build a Morse Tree:

Building a Morse Tree requires a message string to encode.

1. A Node consists of a set of characters, the frequency of times those characters occurs in the message and possibly child and parent nodes.
2. Construct unique Nodes such that each Node consists of only one character in the message.
3. Take the two nodes with the lowest frequency, call these nodes N1 and N2. Create a new node, N3,
   (a) N3's frequency is equal to the frequencies of N1 + N2
   (b) N3's set of characters is the combination of N1 and N2's sets and
   (c) N3's left child is N1 and N3's right child is N2.
4. Repeat step 3 until there is only one Node
5. That last node is the Morse Tree

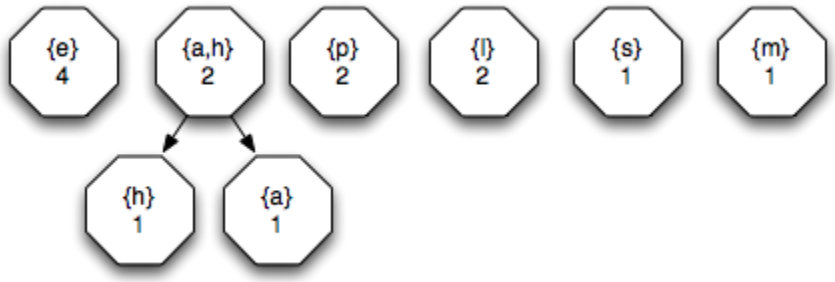
Let's walk through an example if our message is "help me please." First we construct Nodes based off of frequency. Nodes with the same frequency should be ordered based off of alphabetical order. Only lower-case letters should be allowed, upper case letters should be converted to lower-case.[2]

When ordering Nodes, it is *highly suggested* that you use the *PriorityQueue<E>* from Java. One of the constructors takes in an initial size and a Comparator. Pass this constructor an appropriate *Comparator<Node>* and an initial size of 100. Observe that, Nodes with size 1 and 2 are ordered alphabetically.

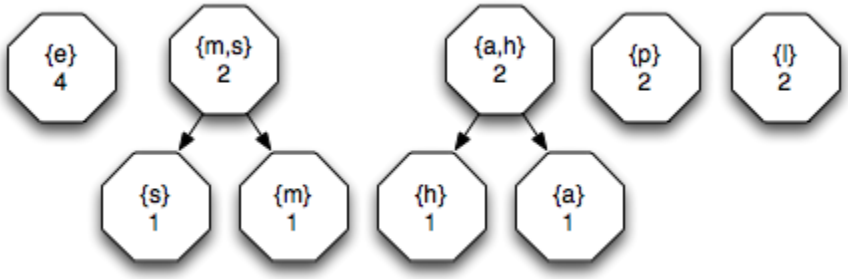


We then pick the two smallest Nodes and combine them:

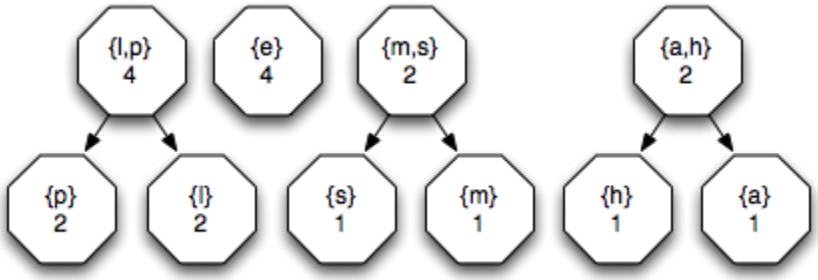
[2]http://www.asciitable.com/



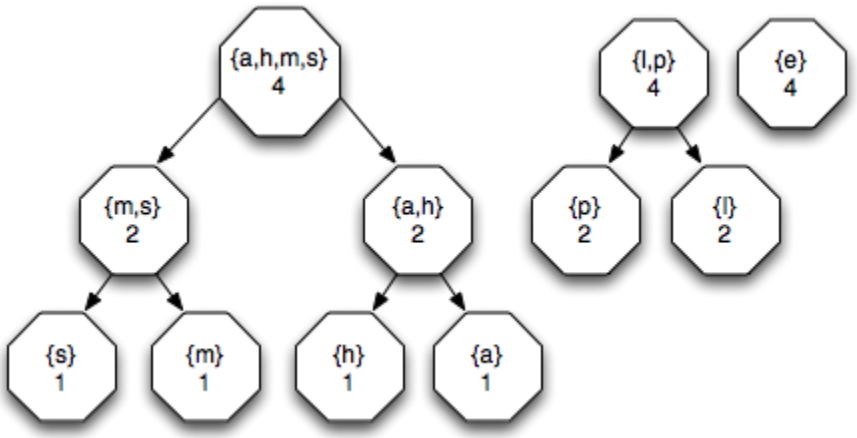
After combining the two smallest Nodes:



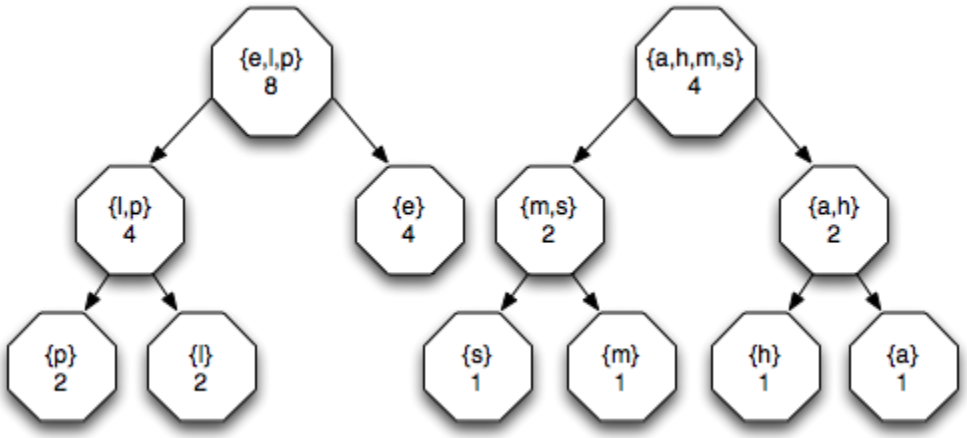
After combining the two smallest Nodes:



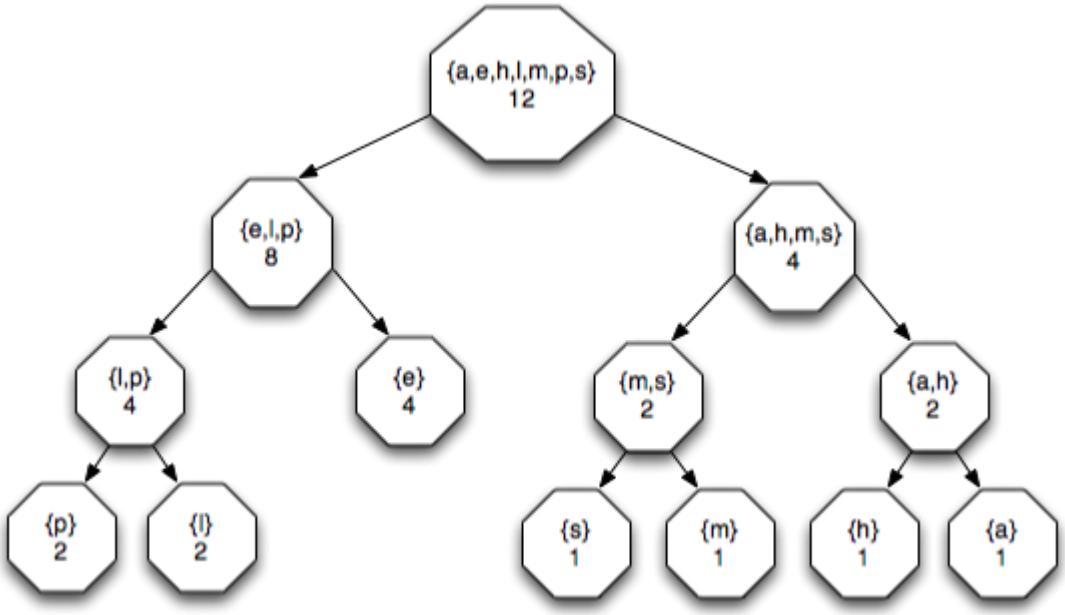
After combining the two smallest Nodes:



After combining the two smallest Nodes:



After combining the two smallest Nodes:



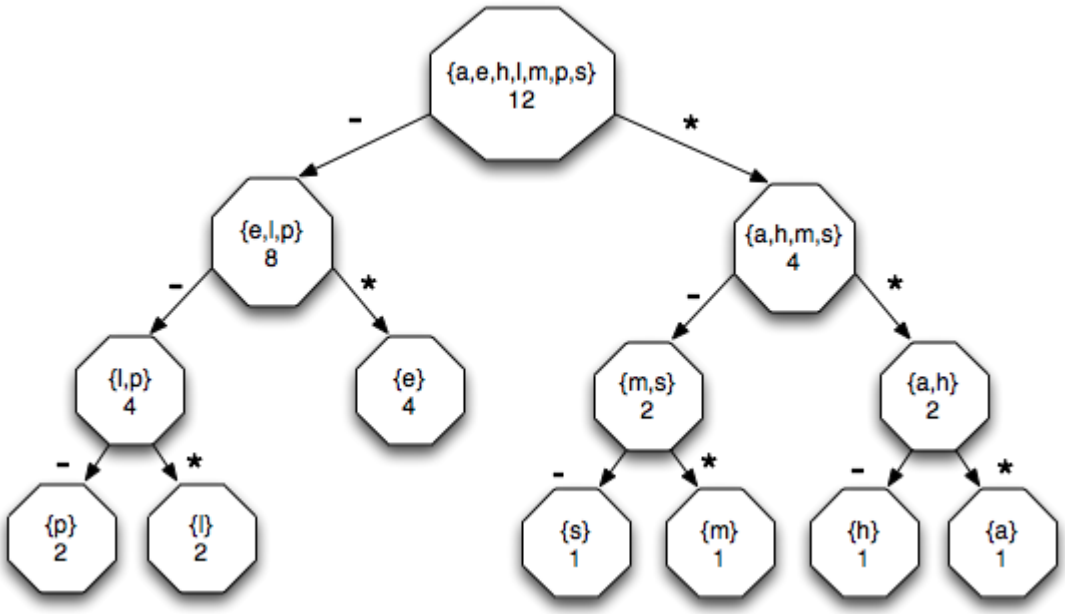
Our tree is finally complete.

Encoding a Character from a Morse Tree

To encode a character in a Morse Tree it HAS to be in the tree. To decode a character in a Morse Tree we follow a simple algorithm:

1. Start at the root of the Morse Tree and an empty string
2. Repeat until a leaf is reached
   (a) if the character is in the left tree, append a "-" to the string and go left
   (b) if the character is in the right tree, append a "*" to the string and go right
3. The resulting string is the encoding of that character

If you look at the resulting tree, the characters that occur most often have the shortest representation.



Character	Frequency	Symbol
e	4	-*
l	2	- - *
p	2	- - -
s	1	*- -
m	1	*-*
h	1	**-
a	1	***

Decoding a Character from a Morse Tree

Given a string from of dots and dashes, we can decode it to a character in a similar way as the encoding process:

1. Select the root node
2. Repeat until you hit a Node
   (a) If the next character in the string is a "-" move to the left
   (b) If the next character in the string is a "*" move to the right

Specification for the Node Class

This class will represent a single node in the MorseTree class. A Node has five properties: a Collection<Character> to keep track of the letters in its subtrees and an int to keep track of the times those

characters occur, two child nodes called left and right and a parent Node. All five properties should be **public.** A Node should be constructed by passing in a Collection<Character> and an int like so:

Node(Collection<Character> collection, int numInstances)

There are many types of Collections in Java's framework. The Collection that you should use in the Node is a *HashSet<Character>*. A HashSet[3] acts just like a *HashMap<Character>* in that it allows (amortized) constant *contains* and *add* operations but, it is different because it will only tell whether a key exists or not. We use a HashSet since it is a Collection and gives a constant access time.

Specification for the MorseTreeClass

The MorseTree class will have three private instance variables: a Map<Character, String> to help quickly encode characters, a Map<String, Character> to help quickly decode Strings of dots and dashes to Characters and a root Node.

A MorseTree has two constructors. After, the constructor is called, the *entire* MorseTree should be built (see: **How To Build a Morse Tree**). The first constructor should take in a Map<Character, Integer>. The Map<Character, Integer> should be a Map whose key is a character and value is the number of times it occurs. This Map will be extremely helpful in building Nodes.

The second constructor should take in a File. The constructor should read text from the file and treat the text file as a message string for a Morse Tree (see: **How To Build a Morse Tree**). Specificially, the text in the file should build a Map<Character, Integer> that keeps track of the times a character occurs.

In addition to the two constructors, MorseTree should have 6 public methods:

Node getRoot()

This method should simply return the root Node.

String encode(String givenString)

This method takes an English string and returns our MorseTree (dot and dash) representation of it. This method should throw an IllegalArgumentException, if a charcter in the givenString is not contained in the MorseTree

[3]http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html

String decode(String morseWord)

This method takes an string of dots and dashes and returns an English representation of it. This method should throw an IllegalArgumentException, if the morseWord is not contained in the MorseTree.

Map<Character,String> getEncoderTable()

This method will return a Map for quickly characters into their dot and dash representation. That is, the Map<Character,String> returned by this method should take in a Character as a key and return a String of dots and dashed as a value.

Map<String,Character> getDecoderTable()

This method will return a Map for quickly turning Strings of dots and dashes into their character values. That is, the Map<String,Character> returned by this method should take in a String of dots and dashes as a key and return a Character as a value.

Iterator<Node> iterator

This method will return an iterator that performs a *level-order-traversal* of the tree. You should have an inner Iterator class that implements the Java Iterator class. The *remove()* method of the iterator should throw a new *UnsupportedOperationException.*

Specification for the MorseClient

In addition to the MorseTree and Node class, build a client that reads in a text file, builds a MorseTree from it and then encodes a message from the user.

The main method takes as input a file that strictly contains text. The program should not ask the user to type in the name of the file. Rather, it should collect the file name by specifying a program argument in Eclipse. You do this by clicking the down arrow on the "run" button, clicking" Run Configurations...", clicking the "Arguments" tab, and then typing the name of the file in the "Program arguments" textbox. When you specify a program argument, the argument gets stored in the String array found in the parameter of the main method (this array is often called "args"). Your program must look for the file you specified relative to the default working directory. This should be the directory that contains the ".settings", "src", and "bin" folders.

After reading the file, the program should prompt the user to input a line of text. If the text has a character that is not in the Morse Tree, the program should inform the user of incorrect input and ask them to try again. Once proper input is given, the program should print the encoded version of the input and ask for input again.