Huffman Coding

Compressing and Decompressing Strings using Binary Search Trees

The idea of the Huffman algorithm is to build a Tree based on the characters in the string, with the most frequent characters being the quickest to access. The tree accomplishes this by placing the high priority characters higher up on the list so that when converted to binary, the output is as small as possible.

The main class used in the project is the Tree class, of which Leaf and Node extend from. The Leaf contains a character and a frequency, where a Node contains two more Trees, which can either be a Leaf or a new Node. The algorithm operates as follows.

1. Build a Queue based on the priority of the characters, order from less frequent to most frequent.
2. Take the two front Trees and create a new node n
3. Reinsert n into the queue
4. Sort the Queue
5. Repeat until there is nothing left

The logic behind exactly how and why this is an optimal way to compress Strings can be found on Wikipedia or Rosetta Code:

<https://en.wikipedia.org/wiki/Huffman_coding>

<https://rosettacode.org/wiki/Huffman_coding>

TODO

binaryTreeRunner:

* buildTree
  + Given a sorted Linked List of Nodes with a character value
* createMap
  + Complete the Hashmap by creating Keys with the values as the binary code needed to access the character on the Tree
* getTree
  + return Tree
* getMap
  + return HashMap
* getCompression
  + Create a string that gives the binary search representation of the original phrase

TreeReciever

* Decode()
  + Given a string in binary, complete the method that will iterate by each character and use it to navigate the tree created during the construction of the TreeReceiver Object

Complete Classes:

* Tree
  + Superclass for Node and Leaf
* Leaf
  + Extension of Tree that includes a character and frequency
* Node
  + Extension of Tree that contains two pointers to Trees
* LLNode
  + Linked list implementation