Overview

The program to be developed is a compression and decompression of any file of ASCII component symbols. It will take a file in and incorporate Huffman compression algorithms to compress a file to make it smaller than the initial file. It will also include a decompression execution that will undo the compression and get the initial file back from this procedure as it initially appeared.

Introduction

This program is made of two different directions. The first will be to compress. Upon starting the program two files are inserted into the arguments for the command line of the program. Then when the program starts it prompts the user for either compression or decompression of the files that were input in with the arguments.

## Scope

The program will compress a single file, decreasing the total space required to store the file as long as it is larger than a kilobyte in initial size.

This system will prompt the user for either compression or decompression on a command line interface level. No graphical user interface will be developed.

This system will use Canonical Huffman encoding algorithm to decrease overall size of the file.

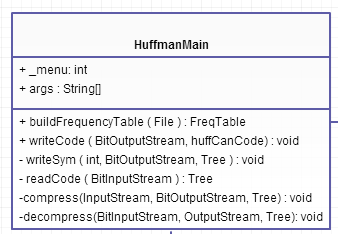
It will be created in the java language

### PROCESSING

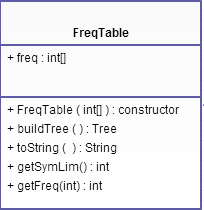
Compression process starts by taking in the input file from argument one. Taking this file and building a frequency table from the inputFile. It is an integer array that counts the symbols that appear in the input. First it takes the file and makes a freqs array that contains 257 space, it adds up freqs[256]++ and attempts to build unless the value currently is -1 for eof. Then returns a new FreqTable(freqs) it creates an instance of frequency table and clones the array created to its local freq[]. Next is to create the table from the frequencies, this is done in freqTable code. The code value is set to the return from Tree buildTree inside of FreqTable which returns a new Tree. Then takes the table built and builds the encoding tree. Then take the encoding tree, and converts from the files current state to a new file.

### COMPONENTS

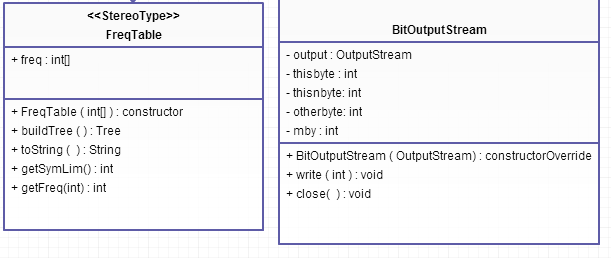
The highest layer is the HuffmanMain file which contains compress and decompress, this interfaces with almost all other files in the code it directly creates freqtable elements and huffCanCode variables. These are each initialized and developed with pre-code done within main that is then handed-off to the code in both FreqTable and huffCanCode.



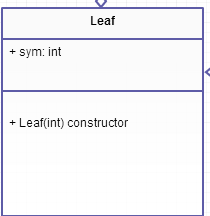
Next the system interacts with the class Frequency table which takes in the built frequency array and creates a code tree from the frequencies within the method buildFrequency



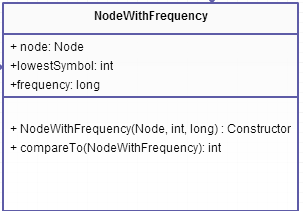
BitOutputStream and BitInputStream read into the values from usage as 8-bit.



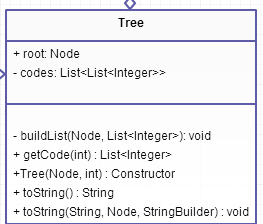
Leaf – this is a leaf object that contains the symbol and a method for setting the integer from a constructor



NodeWithFrequency – Contains the items Node the lowestSymbol and frequency parts for this object. A constructor of Node, int, long and finally a compareTo method for comparing the same object.



Tree – contains the node, and builds the list. Contains the method buildList, getCode, Tree, and toString methods, finally the overridden toString that contains String, Node, StringBuilder.



TESTING

Scenario #1- Small file containing no more than 6 digits

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | File containing “aaaaa” called fin | Compression creates file fout.txt |
| 2 | In command line, change arguments to fout (input) and undo as decoded target file | New file named undo containing encoded files data |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | “aaaaaa” |
| ACTUAL OUTPUT | | “aaaaaa” |
| RESULTS | | PASS |

Scenario #2- Direct copy of a method into file

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Method text is placed into fin (contains many different ASCII bytes) | Input fin as input/fout as output |
| 2 | Run program selecting correct input and output directions | Fout.txt hold correct data of tested material |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | Complete method body in same sequence and formatting |
| ACTUAL OUTPUT | | Complete method body in same sequence and formatting |
| RESULTS | | PASS |

##### APPENDIX:

##### 

This image can be zoomed into to view the image better.