|  |
| --- |
| **CSCI 3901** |
|  |
| Assignment 3 |
|  |

# Overview

The objective of this problem is to encode the contents of a file into a sequence of bits for compression of data. The general idea is as follows: characters that occur frequently are represented by fewer bits while those characters which aren’t regularly used are represented by more bits. The variable length codes assigned to each character are unique.

When an encoded file is to be decoded, the tree built through the comparison of frequencies of characters is traversed according to the sequence of bits in the file. While traversal if a leaf node is encountered, the character in the node, corresponds to the sequence of bits, and is printed.

Decoding, in the case of the Adaptive Huffman algorithm, characters are parsed in cycles or levels corresponding to powers of 2. After every cycle, the frequencies of characters are reevaluated and a new Huffman Tree is built. Characters in subsequent cycles will be parsed, based on the new tree. If the feature - ‘Reset’ is enabled, the frequency counts of characters are reset.

# Files & External Data

The system accepts input from a text file and stores the output in another text file for both, encode() and decode() methods

|  |  |  |
| --- | --- | --- |
| **Method** | **Input File** | **Output File** |
| **encode()** | **✔** | **✔** |
| **decode()** | **✔** | **✔** |

File Name cannot be null or empty.

# Data Structures

The approach used to implement the Huffman algorithm primarily utilizes a **TreeSet** (to sort the character-frequency pairs by ascending order of frequency) and **Binary Trees** (to store characters, frequencies and a pointer to other characters.

## Tree.java

This class represents a binary tree with two variables:

**Entry** – Stores characters / strings

**Frequency** – Stores the number of occurrences of the character in the input file.

## encode():

This method accepts an input file from the standard input, parses the contents of the file, encodes it and stores the output in another text file.

### Parameters:

**String input\_filename** - Path to the file which’s to be encoded

**int level** – The maximum number of characters which can be parsed in the last cycle is determined by 2^level.

**boolean reset** – If reset is true, frequency counts are reset at the beginning of every cycle.

**String output\_filename** - The path of the file, in which the output is stored.

## decode():

This method accepts an input file from the standard input, parses the sequence of bits in the file, decodes it and stores the output in another text file.

### Parameters:

**String input\_filename** - Path to the file which’s to be encoded

**String output\_filename** - The path of the file, in which the output is stored.

## codebook():

Returns the codebook (a Map) which stores the sequence of bits for all characters in an input file.

# Assumptions

In addition to the assumptions stated under the section titled “Assumptions” in the document – CSCI 3901 Assignment 12, the following additional assumptions were made.

1. Spaces are to be encoded
2. Carriage Returns are to be encoded

# Algorithm & Design

Approach to **encode**():

Approach to **decode**():

# **References**

1. <https://www.geeksforgeeks.org/integer-max_value-and-integer-min_value-in-java-with-examples/> - How to assign a maximum / infinite value to a variable?