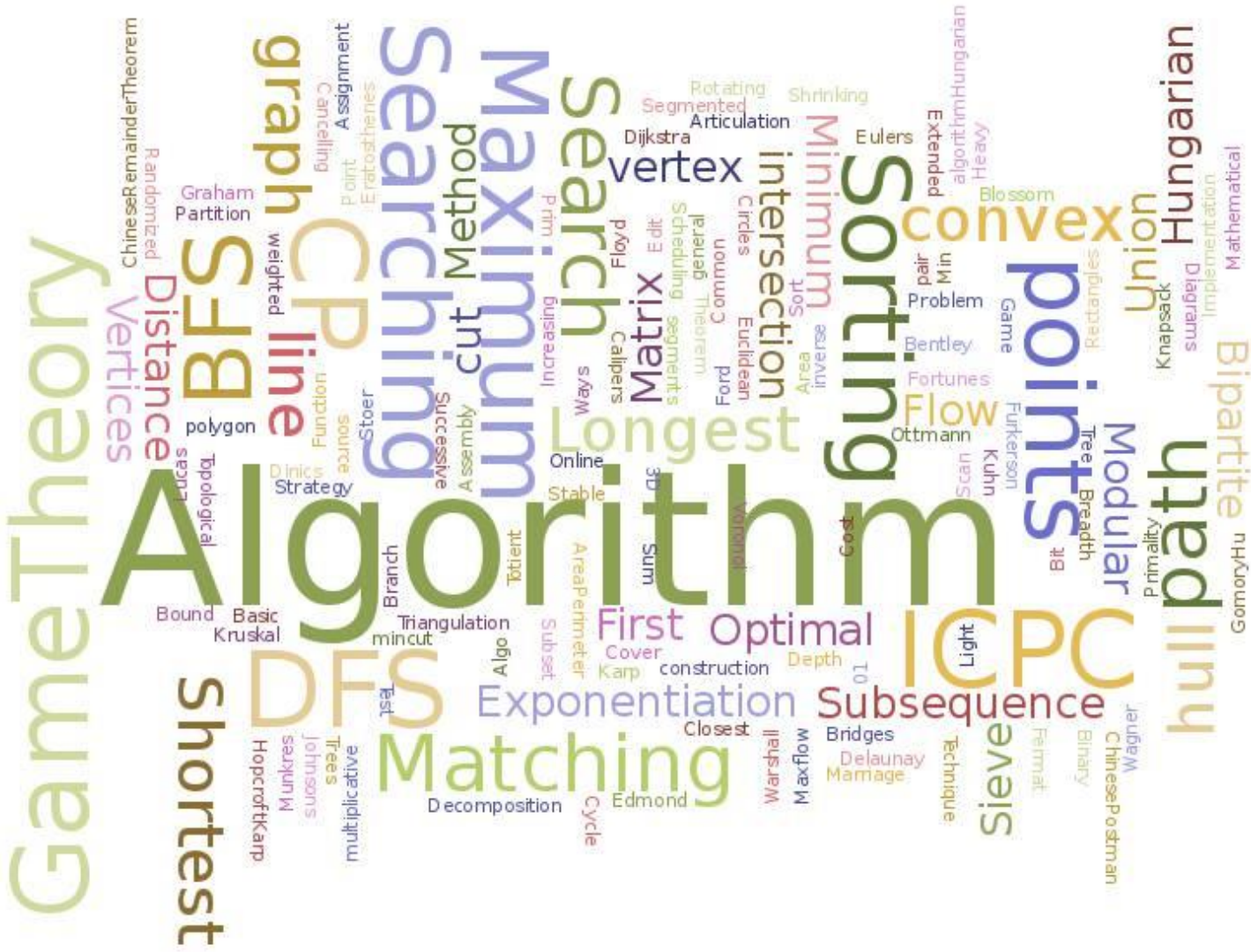


HUFFMAN CODE

INTRODUCTION

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted. However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods.

APPLICATION FUNCTIONALITY

Our application implementation provides the following functionalities

- Compress a txt file
- Extract a compressed txt File
- Compress a folder **BONUS**
- Extract a compressed folder **BONUS**
- Compress a binary file **BONUS**
- Extract a binary file **BONUS**

ALGORITHM USED & COMPLEXITY

- The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.
- The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.
- There are mainly two major parts in Huffman Coding
  1) Build a Huffman Tree from input characters.
  2) Traverse the Huffman Tree and assign codes to characters.
- For Complexity: O(nlogn) where n is the number of unique characters.

  If the input array is sorted, there exists a linear time algorithm.

HEADER FORMAT OF THE COMPRESSED FILE

The header file is used to reconstruct the tree in decompression
The header file format consists of the following:

- Number characters appeared in the table.
- Each Character assigned with its frequency (length code)

COMPRESSION FORMAT

After storing the header file (Frequency Table) we do the following:

- Read a char from the input file at a time
- Lookup in table for its Huffman Code
- Write the compressed code in compressed file as bytes (binary).

In order to write byte by byte , we handled it in a string in a manner to write each 8 chars in this string as a one-byte char in new binary file.

DESCRIPTION OF OUR IMPLEMENTATION

Our Huffman code implementation classes are classified as follow:

- BinaryTree Class: Implementation for the structure of Huffman Tree having in each node a HuffmanCode object.
- HuffmanCode Class: Where we hold a char and a frequency for each character in the file
- Encode Class : where compressing occurs
- Decode Class : Where Decompression Occurs

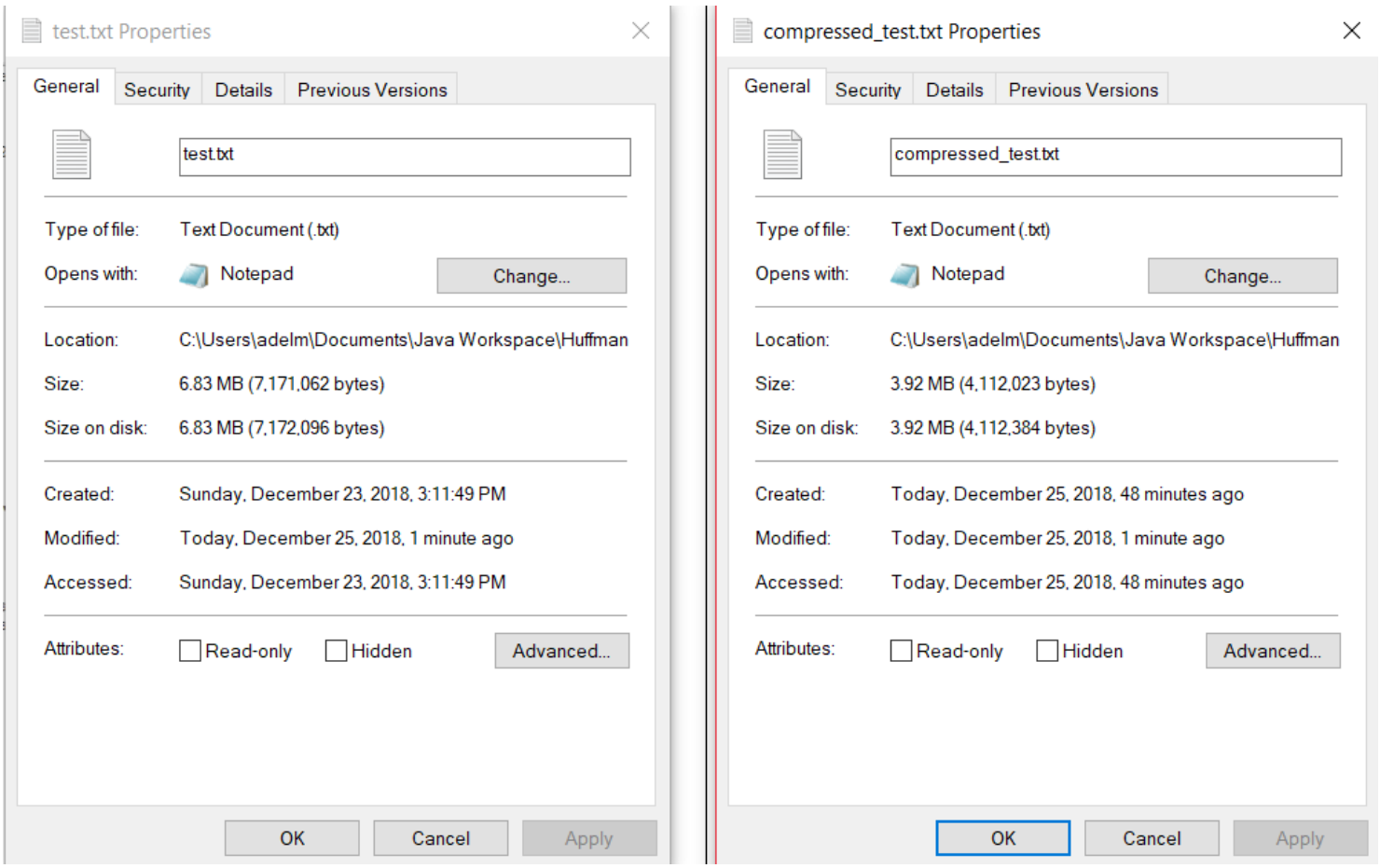
DATA STRUCTURES USED

We used the following data structures in our implementation:

- HashMap<char, int>: Used to build the frequency table given an input file.
- Binary Tree: Used to build a Huffman tree given a frequency table
- HashMap<char, string>: Used to hold the Huffman code table for each character.

SCREENSHOTS

```
//////////////////////////Compressing File: test.txt//////////////
-------------------------
Character Frequency Table
-------------------------
 ="1", □="1", a="3", b="4", c="2"
----------
charLength
----------
2="[a, b, c]", 3="[□,  ]"
Reverse
[3, 2]
FirstLength: 3
base: 000
binary base: 0
Incremented base: 001
Incremented base: 010
binary base: 2
Incremented base: 10
Incremented base: 11
Incremented base: 00
-----------
Prefix Code
-----------
□="000",  ="001", a="01", b="10", c="11"
charsss: 5
----------------------
Compression Statistics
----------------------
Original File Size      10 bytes
Compressed File Size    14 bytes
Compression Ratio       -40.0 %
Compression Time        13 ms.
```


test.txt Properties
General
Security
Details
Previous Versions
test.txt
Type of file: Text Document (.txt)
Opens with: Notepad
Change...
Location: C:\Users\adelm\Documents\Java Workspace\Huffman
Size: 6.83 MB (7,171,062 bytes)
Size on disk: 6.83 MB (7,172,096 bytes)
Created: Sunday, December 23, 2018, 3:11:49 PM
Modified: Today, December 25, 2018, 1 minute ago
Accessed: Sunday, December 23, 2018, 3:11:49 PM
Attributes: Read-only Hidden
Advanced...
OK
Cancel
Apply
compressed_test.txt Properties
General
Security
Details
Previous Versions
compressed_test.txt
Type of file: Text Document (.txt)
Opens with: Notepad
Change...
Location: C:\Users\adelm\Documents\Java Workspace\Huffman
Size: 3.92 MB (4,112,023 bytes)
Size on disk: 3.92 MB (4,112,384 bytes)
Created: Today, December 25, 2018, 48 minutes ago
Modified: Today, December 25, 2018, 1 minute ago
Accessed: Today, December 25, 2018, 48 minutes ago
Attributes: Read-only Hidden
Advanced...
OK
Cancel
Apply

THANK YOU