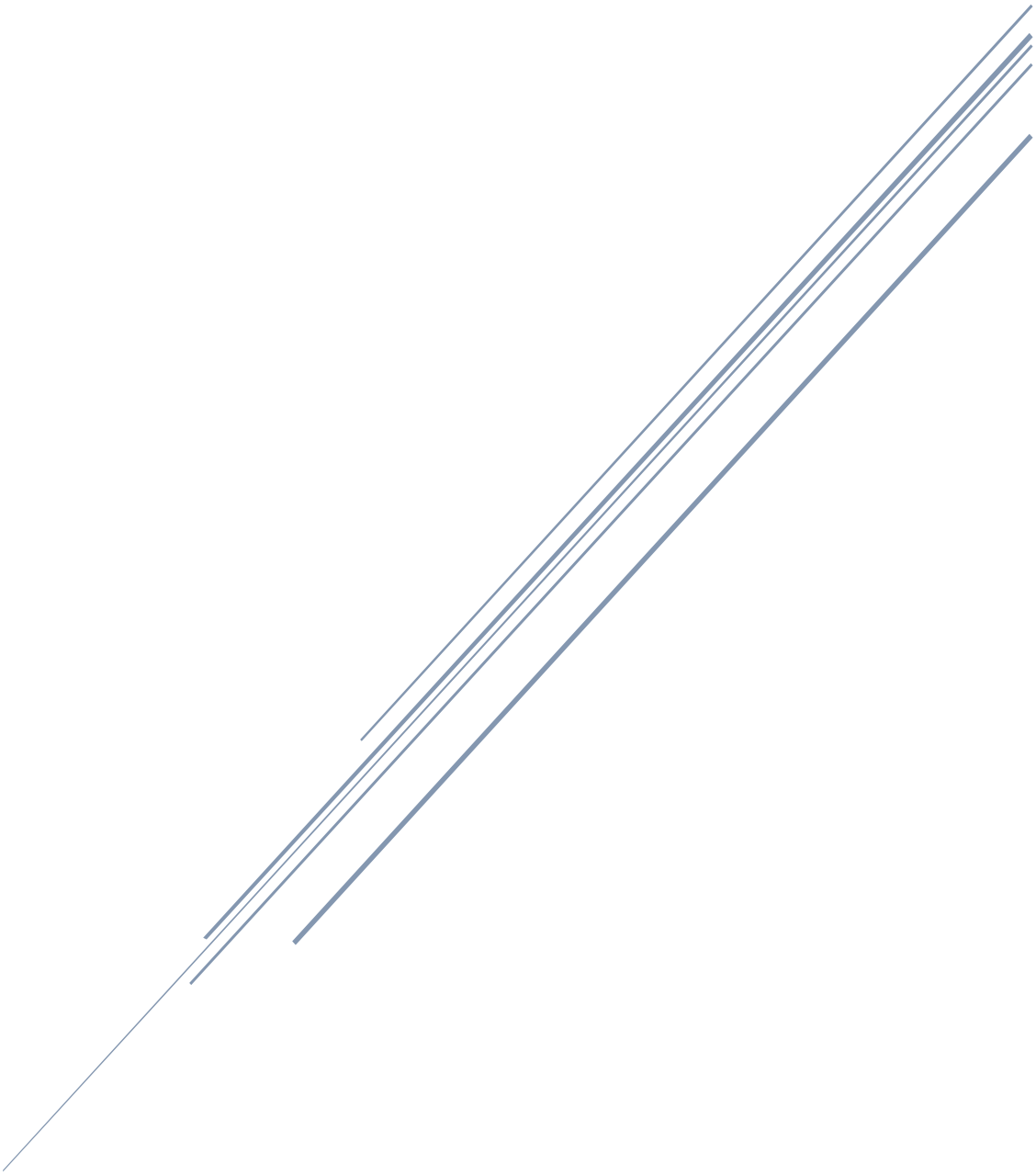
HUFFMAN COMPRESSION

Abhishek Srikanth



Purdue University
ECE 368 – Project 2

Format of compressed files

1. A single unsigned long long integer is the first thing written into the file. This is used to denote the number of characters that have been encoded into the compressed file.
2. The Huffman tree information is then stored. The format for the Huffman tree information is as follows:
    a. While encoding, the Huffman tree is traversed using post-order traversal and the data is hence written the same way. A single bit is used to denote if the node being printed is a leaf node or not.
    b. A bit of value **1** denotes a leaf node and is followed by 8 bits that represent the character at that node. A bit of value **0** denotes a non-leaf node and hence is not followed by any character after it.
3. The file is then encoded using the Huffman tree. Each characters path from the root is encoded into small bits where a bit of value 1 represents the moving right from a node, and 0 represents moving left.

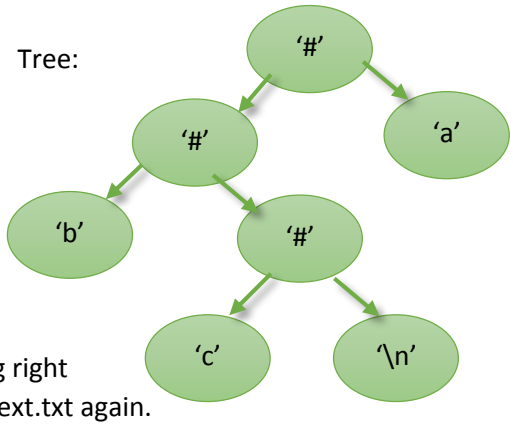
For illustration purposes, the following example has been furbished:

Text.txt: “aaaaabbc\n”

File information:

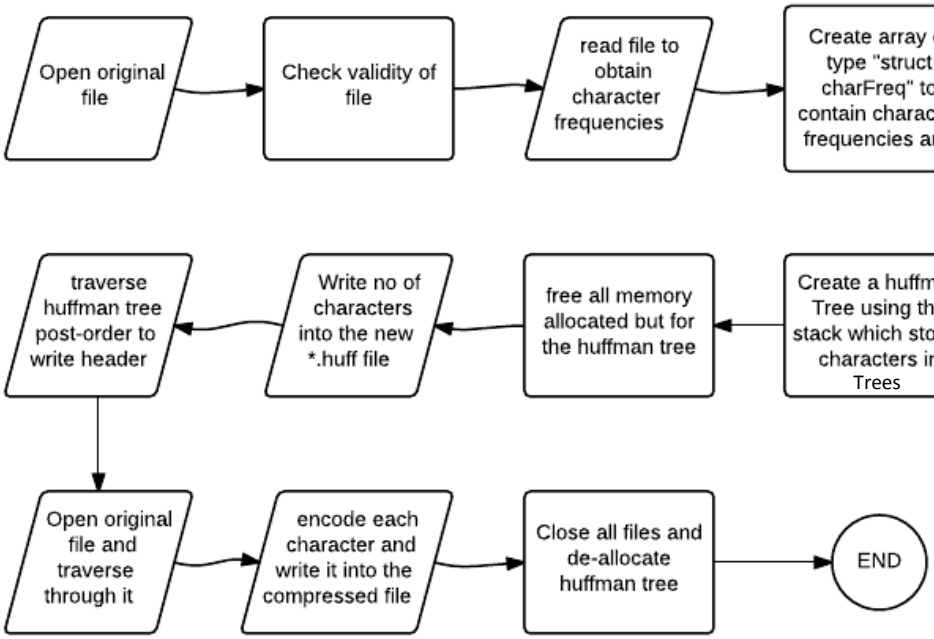
9 // llu that represents number of characters in the file
1011 0001 // **1** represents a leaf node printed in post-order
0**1**01 1000 // and is followed by the character which I have
11**1**0 0001 // underlined
010**0** **01**01 // **0** represents a non-leaf node on the tree
1000 01**0**0 // all values after last **0** are simply wasted/padding
1111 1000 // This is the main body of the file. 1 represents moving right
0010 0110 // and 0 represents moving left. Decoding this we get text.txt again.
EOF character // main body may have extra characters at the end (here a 0 bit is extra).

Tree:



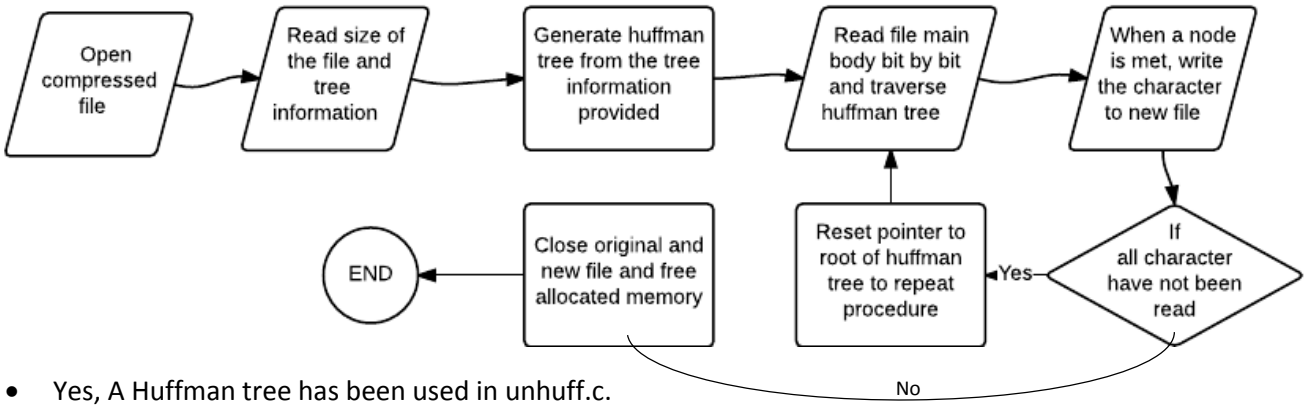
Program flow

Huff.c



- Yes, A Huffman tree has been used in huff.c program
- It is used to generate the header information by traversing in post order.
- It is then used in encoding the input text file by finding the encoded value for each character by traversing from root and then writing the encoded bits into the text file.

Unhuff.c



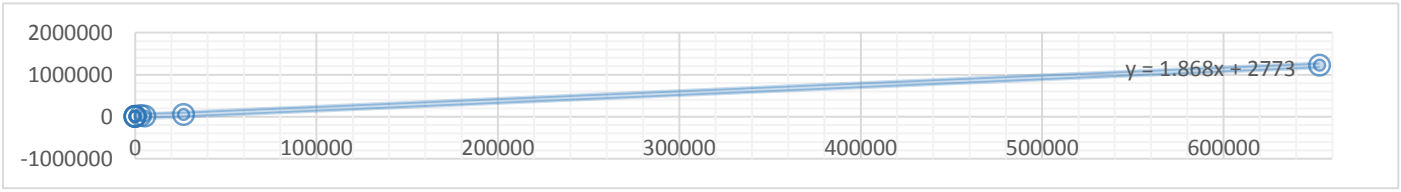
- Yes, A Huffman tree has been used in unhuff.c.
- As the header file is read, all command '1' values are pushed into a stack, and when a command '0' is read, two elements are popped, combined and pushed back into the stack. This eventually creates the huff Tree.
- When the file main body is read, the tree is traversed based on input. 1 moves right in the tree and 0 left.
- Upon reaching a leaf node, the character is printed and the process is repeated again from the tree root.

Sources used

- Problem statement provided for project 2 in ECE 368.
    - Process used to generate Huffman tree from a list of frequencies was understood and similar procedure with stacks was adopted.
- Did NOT use code/resources from ECE 264 assignment.

Results:

Original file Name	Original file Size	*.huff file Size	Decompressed file Size	Is Matching	Compression Ratio
Empty_file	0	2	0	TRUE	0
two_chars	2	5	2	TRUE	0.400
Multiple_same_char	6	5	6	TRUE	1.200
simple	221	177	221	TRUE	1.249
10k	10,030	5,414	10,030	TRUE	1.853
Large_two_char	25,500	3,196	25,500	TRUE	7.979
Half_million	50,156	26,833	50,156	TRUE	1.869
massive	1,222,588	652,993	1,222,588	TRUE	1.872



From slope of the plot we can see that an average file is **compressed by a factor of nearly 2**.
That said, compression depends on the number of different characters, and to prove this, "Large_two_char" is a perfect example of where a large file is compressed in very little space.

Required flags

Need "**–lm**" to link with math library for both huff.c and unhuff.c.
Must compile "**datastruct.h**" along with the c files.
Also, my program may not run with the "-O3" optimization flag.
To make it run, either the flag may be entirely ignored or the optimization flag "**-O2**" may be used.

1. **gcc –Wall –Werror huff.c datastruct.h –o huff –lm –O2**
2. **gcc –Wall –Werror unhuff.c datastruct.h –o unhuff –lm –O2**