# ECE 368 Project2 Report Xiao Xiao 0026557235

## Description

The main purpose of this project is to write two programs called huff.c and unhuff.c. The program huff.c will generate a compressed file using Huffman coding with a text file input. The other program unhuff.c will generate a text file with a Huffman compressed file input. The extensions are changed as desired in the instruction. There are two structure types and fifteen functions are defined in huffman.h file. Data type tnode contains a character value, a integer value and two children nodes. Data type min heap contains a integer size of the heap and a tnode array.

## huff.c Program

*Build priority queue*

- *Build arrays*

Two arrays with size of 256 were created to collect the input characters and their weights. The indices of each array are corresponding to the ascii code. Each time a new character is read, increment the count in the weights array and add a new character inside the characters array. Then transfer the non zero values to another two arrays.

- *Create min heap*

After two input arrays are determined. Primitive functions like construct_tnode, construct_minheap,insert_tnode,delete_tnode,heapify are prepared before start to build a min heap. A node is constructed for each character inside the characters array. When a new node is constructed, the node will be assigned in to the tnodes array inside the min heap. Function heapify is called multiple times to maintain the root node of the min heap to make sure the root always contains the smallest weight inside the tnodes array.The pseudo EOF is inserted
as a node into the min heap.

*Build Huffman tree*

Two nodes containing smallest weights are deleted from the root of the min heap. Heapify is called each time a node is deleted to maintain the root. Then a new node is generated to combine the weights of the two deleted nodes. The deleted nodes will the be the children of the new node. The new node is inserted back to the min heap. Size of the min heap will be incremented when insert and decremented when delete.

*Write header information*

Two functions are related to generate the header information in the output binary file. The function called write_bits will take an unsigned character input to determine how many bits to write in to the file. A variable called bit_fld is a bit field that will combined the desired output bits into one single byte. Whenever 8 bits are determined, fwrite function is called to write the bit field into the output file. One special case is when a character is needed to be write into the file. A variable called buff will be used to separate the 8 bits of one character. The second function is called print_tree, which will use a pre-order traversal to determine the sequence for the Huffman tree. Bit 0 is written when a non-leaf node encountered. Bit 1 is written followed by the ascii code of the character inside a leaf node. The function write_bits will be called each time some bits need to be written in to the header.

*Write Huffman coding sequence*

A function called find_sequence is used to trace the path inside huffman tree with a given character. 1 indicates right and 0 indicates left. A sequence array is used to collect the path. Traverse the Huffman tree until it meets the desired node and print the sequence using function write_bits.

*Change extension*
The length of the input file name is determined at the beginning. Then sprintf function is called to add ".huff" to the extension.

## unhuff.c Program

### *Read header information*
Two functions are involved in reading the header information. One is called read_bits, which apply the same idea as write_bits. It will take in 8 bits as a byte each time and shift once when reading a single bit. It will also use buff when a character is encountered. The second function is called read_tree, which create a Huffman tree based on the header information. If a single bit read is 0, two children will be created for the root node. On the other hand, if a single bit 1 is read, character store in the header will be read and assigned into the node.

### *Read Huffman coding sequence*
A while loop is implemented as the reverse version of find_sequence. When bit 1 is read then go left. When bit 0 is read then go right. Whenever a leaf node is founded, check whether it is a pseudo EOF. If it is not EOF, print the character inside the node into the output text file, otherwise break the while loop.

*Change extension*
The length of the input file name is determined at the beginning. Then sprintf function is called to add ".unhuff" to the extension.

*Performance tables as shown below.*

| **text0.txt** | | | **text1.txt** | | | **text2.txt** | | |
|---|---|---|---|---|---|---|---|---|
| | **Runtime (sec)** | | | **Runtime (sec)** | | | **Runtime (sec)** | |
| **Ratio** | **huff** | **unhuff** | **Ratio** | **huff** | **unhuff** | **Ratio** | **huff** | **unhuff** |
| 2.25 | 0.001s | 0.001s | 2.375 | 0.001s | 0.001s | 0.8194 | 0.001s | 0.001s |

| **text3.txt** | | | **text4.txt** | | | **text5.txt** | | |
|---|---|---|---|---|---|---|---|---|
| | **Runtime (sec)** | | | **Runtime (sec)** | | | **Runtime (sec)** | |
| **Ratio** | **huff** | **unhuff** | **Ratio** | **huff** | **unhuff** | **Ratio** | **huff** | **unhuff** |
| 0.733 | 2.138s | 0.176s | 0.733 | 4.084s | 0.725s | 0.733 | 5.766s | 0.628s |