What is the purpose of the Project?

To use file system API and different data structures to build and store huffman tree in a file named HuffmanCodebook and further use the Huffman Codebook to compress or decompress the text files located in directory

Algorithm for the Project:

As we are given a directory path we will use recursion to go into each folder and store each file's location into a two D array, after storing all the locations we will read each text file. To generate HuffmanCodebook we will traverse through the file storing each word, space or control characters with their respective frequencies in a binary search tree. The binary tree will be then converted into minheap, for converting it into minheap we will traverse through the binary tree and add each node into our minheap. The next step is to generate huffman tree from minheap, for that we now keep taking two nodes at a time from minheap, merge them into single node and insert it back to minheap until only one single node remains (out of those two nodes the one with lower frequency becomes the left child while the one with higher frequency becomes the right child). After finishing this step we will have a huffman tree, which will further be stored in a file called HuffmanCodebook and given as output. Next part of the assignment is to compress and decompress a given text file, for that we use the previously generated HuffmanCodebook. We read the HuffmanCodebook and store it into a binary tree and traverse through it to switch word with code or code with word and out put the compresses or decompressed file.

Analysis of the time and space usage and complexity of our program:

The average case for storing words and their frequency will be O(n log n) as we have used binary search tree to store them. Then converting binary search tree into minheap will take O(n log n) time. Making huffman tree out of minheap will take n-1 deletions and insertions, each taking O(log n) time. The space usage will be (size of each word) + (number of words * size of integer) + (size of left and right pointer) for making HuffmanCodebook. After generating HuffmanCodeBook we clear all nodes previously used. For compressing and decompressing we read the HuffmacodeBook and store it into binary search tree which makes the search time of O(log n) for switching word with code or code with word. For traversing the file it will take O(n) time. Therefore building code book or compress/decompress it will over all take O(log n) time with space mentioned above and traversing the file will take O(n) time.

List of all files and their working for better understanding:

fileCompressor.c :

The first thing we do in fileCompressor.c is to read all the arguments passed from terminal using getopt from <getopt.h> library and increments the flag for compress, decompress, recursive and build HuffmanCodebook. After incrementing flags we now know what the user wants, if the user wants to build the huffmanCodeBook we build the book, if they want to compress or decompress the file we do that. For making HuffmanCodebook we will traverse through the file storing each word, space or control characters and their respective frequencies. For storing all these things we use the binary search tree which we have made in **Frequency.c** file. Next thing comes is to create a Minheap to make huffman tree. We use **MinHeap.c** and traverse through frequency tree and insert each node into minheap array. After all nodes are inserted into our minheap we move to our next step which is to make huffman tree. We get two minimum nodes from our minheap, add their frequency (out of those two words the one with lower frequency becomes left child and the one with higher becomes the right child use the **tree.c** file to make this type of node) and insert it back into tree until single node remains. After the huffman tree is generated we use the **Dictionary.c** to output a file with the huffman tree in it. If the user says to compress to decompress then we read the HuffmanCodebook provided and convert it into binary tree and switch words with code or code with words and out put the compresses or decompressed file.

FrequencyTree.c:

This C file contains binary search tree for storing words/control characters and their respective frequencies. Each node contains a char pointer to store the word, an integer to store frequency and left/right node pointer. It also contains various other functions to support in making of tree such as createFTNode to create node, createFreqTree to initialize frequency binary tree, insertNode to insert node, freeFreqTree to free whole tree, freeFTNode free each node etc.

MinHeap.c:

After the binary tree with words/control characters is created using FrequencyTree.c we insert each node into minheap. minheap is an array declared using malloc which stores all the words in a certain order using their frequency as comparison. We compare each frequency and store in such a way that the array represents as binary tree with the minimum number as root and increases as we move down the tree. When we insert into array we know the index and from that index we can get to the parent index to switch them if parent's frequency is bigger. The parent index would be: the index number we are currently at minus 1 and divide by 2, in the same way left/right child would be the index we currently at times 2 plus 1 and the index we currently at times 2 plus 2. After we get

the final sorted minheap array we get two minimum frequency nodes, add their frequency and merge them. We keep doing this until one node remains in array. Thus getting the huffman tree at index 0.

Tree.c:

Tree.c contains the huffman tree node and functions which are called by minHeap.c to merge two nodes and insert them back into minheap.

Dictionary.c:

Traverses through the huffman tree and writes the tree and code for it into file. For compressing and decompressing it reads the HuffmanCodebook and stores each word and its huffman code into binary search tree

HuffmanCodebook:

The file name HuffmanCodebook is generated at the end with the huffman tree in it.

anytextfile.hcz:

The file with extension .hcz contains a compressed version of a text file**.**