Program Space and Time Complexity:

Depends on which operation is called.

➔ If compress is called it will take O(h) time where h is the height of the BST.

➔ If decompress is called it will also take O(n.h) where h is the height of the tree.

➔ If build it called it will take O(n log n)

used Binary Search Tree, MinHeap and LinkedList in the project because

➔ BST has an average case complexity of O(h) for insert and search

➔ MinHeap has O(log n) insert and search

The program is composed of several files, where the main executable file is fileCompressor and other header files namely,

➔ BST.h,BST.c

➔ Huffman.h,Huffman.c

➔ minHeap.h,minHeap.c

User Detailing

The program is designed to take 5 inputs from the command line. Specifically, those are executable file, flags the user wants, and path to the file or directory and the If, for some reason, user inputs more than 5 inputs or less than 3 inputs the or the arguments provided to the command line are provided in the different order then such inputs are detected and displayed as f errors and the program is exited. If inputs are provided correctly, then program goes to read the file or files from the directories and based performs the assigned flag. If flag includes -c and -d. The program also reads the codebook and prompts an error if that codebook is not provided. If -b is called the program reads the file/files, make tokens and store into LinkedList. Then a minheap is formed from the linkedList, using llToMinHeap function. and then huffmanTree is created using huffmanTreeCreation function and makes and writes to a HuffmanCode file using getHuffmanCode function. Similarly if compress is called, codebook is read and is stored as BST and then token from compressed files are read searched in BST and if found they are replaced. In decompress, a codebook is read and compressed file/file path is read and then Huffman tree is re created from the codebook tokens based on codebook tokens program traverses the tree and writes to a new file the replaced token.

fileCompressor consists of

```
int createFile(char flag, char *path)
```

Creates a new file named “HuffmanCodebook” if flag is b otherwise creates whatever filepath is given and returns the file descriptor.

Char flag = flag is passed either b, c or d

char path = path of the new file to be created

void writeToFile(char *s, int fd)

The function writes the char * s to the file with provided file descriptor fd. It does that using write() system call

char* s = string to write to the file.

int fd= descriptor of the file where the file is to be written.

void searchBST(node *RootPtr,char *s, int f)

Given the root node, the function searches through the BST of codebook tokens searching for string s, once the string is found, it is replaced by the huffmancode of that sequence. This is used to compress the files.

node* RootPtr = the current of the BST, starts with the root of BST.

char *s = String you are searching for basically the string you want to replace with it’s huffcode

int f = File descriptor of the file where the huffcode has to written.

int replace(node *root, char *s ,int f, int idx)

The function writes the actual string to file defined by file descriptor f by traversing the huffmanTree using huffcode (string s).It returns -5 if root is null, 0 if the replacement has taken place and -1 if replacement has not yet taken place.

node* root = starts with the root of HuffmanTree.

char *s = Huffman code which has to be traversed to reach the leaf node of the tree to replace the string

int f = descriptor of the file where the s is to be written.
int idx = current index to check in string s.

void decompress (char *buff, int f)
The function decompresses the buff into actual tokens using replace function and write to the file given by the descriptor

Char* buff = The file tokens of compressed file which are to be replaced.
Int f = descriptor of the file where the decompressed string is to be written.

void llToMinHeap()
Takes all the nodes from the linkedList and put it in the MinHeap.

Void huffmanTreeCreation()

The function basically keeps on removing 2 nodes from the MinHeap and creates a HuffmanTree. The nodes are been removed till no nodes are left in the minheap.

Void makeBSTtokens(char* buff,int fd)

Makes tokens of the char* buff and sends it to the searchBST to get it write to the associated huffmancode to the file descriptor fd.

Char* buff = codebook token which has to be stored into BST

Int fd = file descriptor of file to write.

Node * buildTree(node *t1, node *t2, int *ptr)

Builds the a node whose child are t1 and t2 and returns that node.

Node t1 = first node which is been removed

Node t2 = second node which is been removed from minheap

void readFile(char* fname, char* buffer, int size, int* track, char flag)

This function is going to read a file and store it content in a buffer. If the flag is c or d, then it is a recursive call. If it is the recursive call, then it is going to create a new file based on the flag. If the flag is c, then it is going to call makeBSTtokens with buffer and fileDescriptor as the arguments. If the flag is d, then it is going to call decompress with buffer and fileDescriptor as well. If it is not recursive, then it is just going to read the file fname, and store the content into the buffer.

Char* fname = fname is the file path which is going to read in this method

Char* buffer = all the contents from the given file is going to get stored in it.

Int size = size of the buffer

Int* tract = to keep track of the number of bytes that has been read

Char flag = current flag is sent in this char variable.

---

void readDir(char* pathName, char* buffer, int size, int* t, int* access, char flag)

This function is going to open a directory and read all the files inside that directory. It is going to process one file at a time and store it contents in a buffer. If the flag is c or d, then it is a recursive call and it perform other functions based on it. If it is the recursive call, then it is going to create a new file based on the flag. If it is not recursive, then it is just going to read all the files fname one by one, and store the content into the buffer.

Char* fname = fname is the file/dir path which is going to read in this method

Char* buffer = all the contents from the given file is going to get stored in it.

Int size = size of the buffer

Int* t = to keep track of the number of bytes that has been read

Char flag = current flag is sent in this char variable.

void makeCtokens(char *c, char flag)

It creates a token from the codebook and sends the token inside the insertBSTtoken function if the flag is c. If the flag is not c but d, then it send the token to HuffcodeToTree method.

Char *c = it is the codebook

Char flag = current flag is sent in this char variable.

void readFile(char* fname, char* buffer, int size, int* track, char flag)

This function is going to read a file and store it content in a buffer. If the flag is c or d, then it is a recursive call. If it is the recursive call, then it is going to create a new file based on the flag. If the flag is c, then it is going to call makeBSTtokens with buffer and fileDescriptor as the arguments. If the flag is d, then it is going to call decompress with buffer and fileDescriptor as well. If it is not recursive, then it is just going to read the file fname, and store the content into the buffer.

Char* fname = fname is the file path which is going to read in this method

Char* buffer = all the contents from the given file is going to get stored in it.

Int size = size of the buffer

Int* tract = to keep track of the number of bytes that has been read

Char flag = current flag is sent in this char variable.

---

void readDir(char* pathName, char* buffer, int size, int* t, int* access, char flag)

This function is going to open a directory and read all the files inside that directory. It is going to process one file at a time and store it contents in a buffer. If the flag is c or d, then it is a recursive call and it perform other functions based on it. If it is the recursive call, then it is going to create a new file based on the flag. If it is not recursive, then it is just going to read all the files fname one by one, and store the content into the buffer.

Char* fname = fname is the file/dir path which is going to read in this method

Char* buffer = all the contents from the given file is going to get stored in it.

Int size = size of the buffer

Int* t = to keep track of the number of bytes that has been read

Char flag = current flag is sent in this char variable.

void makeCtokens(char *c, char flag)

It creates a token from the codebook and sends the token inside the insertBSTtoken function if the flag is c. If the flag is not c but d, then it send the token to HuffcodeToTree method.

Char *c = it is the codebook

Char flag = current flag is sent in this char variable.

Node* createNode(char* hCode, char* str)

It create a new node and allocate the space for the node. It also allocated the space for the char* data field of the struct node. It sets all the index in the node str to '\0' and then copy the char string str in the node str. Returns the node pointer with str stored in it. Also, sets left and the next pointer to null.

Char *hCode = huffmanCode is stored in this for the given string.

Char * str = char characters/ string to be stored in the node.

Node* huffcodeToTree(node* huffmanTree, char* huffCode, char* str, int a)

It create a tree from the Huffman code. It is 0 it goes to the left, if it is 1 it goes to right but once the string is over a token is stored as the leaf node.

Node* huffmanTree = root of the Huffman tree code

Char* huffCode = string/Huffman code that is to be converted in the Huffman tree