Synopsis

The Huffman coding algorithm is a compression technique that assigns variable-length codes to characters based on their frequencies in the input data. This code can either encode (compress) or decode (decompress) data using Huffman coding.

Huffman Encoding Algorithm

1. Frequency Counting determines character frequencies in the input data.
2. Create a priority queue (min-heap) based on the frequencies of symbols.
3. Building Huffman Tree:

While there is more than one node in the priority queue:

Remove the two nodes of the lowest frequency.

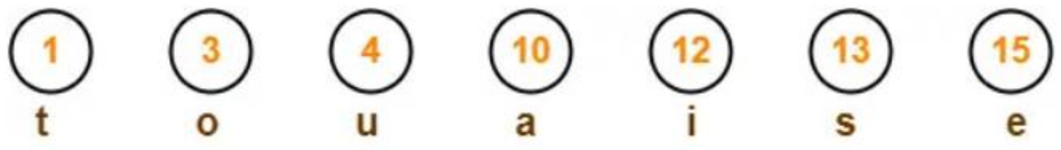
Create a new internal node with these nodes as children, and a frequency equal to the sum of their frequencies.

Insert the new node back into the priority queue.

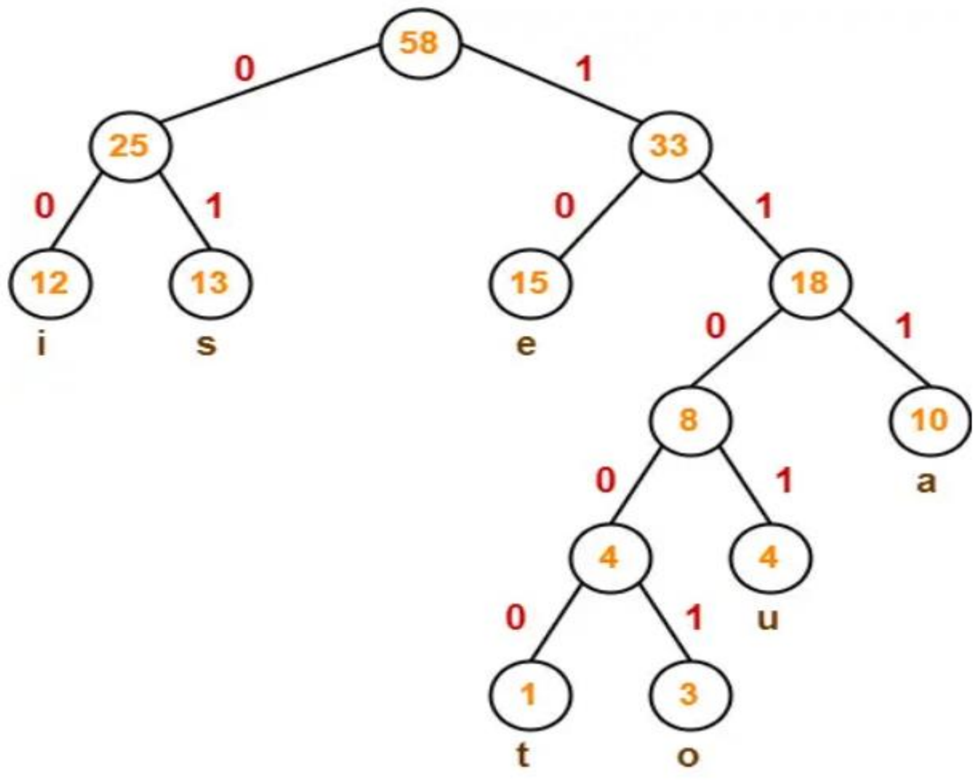
4. Assigning Binary Codes Traverse the tree, Assign '0' to left edges and '1' to right edges.
5. Encoding Data Replace characters with Huffman codes and write to output.
6. Creating Header ,Store file size and decoding info in a header.
7. Decoding Data Reconstruct the tree, decode bits, and recover original text.

Pictorial Representation:

Consider a text having few alphabets with following frequencies(count)



Huffman Tree



Huffman codes:

e-10

i-00

s-01

a-111

u-1101

t-11000

o-11001

Functions Used:-

1. determine_counts(FILE *f):
   - Counts the frequency of each character in the input file.
   - Uses the `counts` array to store the count of each character.
2. initialize():
   - Initializes data structures, including memory allocation for frequency tracking.
   - Allocates memory for the `counts` array.
3. allocateTree():
   - Allocates memory for Huffman tree nodes.
   - Allocates memory for the `nodes`, `parent_indx`, and `leaf_indx` arrays.
4. addNode(int index, int count):
   - Adds a node to the Huffman tree based on its count.
   - Maintains the order of nodes in the tree.
5. addLeaves():
   - Adds leaves to the Huffman tree based on character frequencies.
   - Calls `addNode` for each character with a non-zero frequency.
6. buildTree():
   - Builds the Huffman tree based on the character frequencies.
   - Calls `addLeaves` to add leaves, and then combines nodes to form the tree.
7. encodeAlphabet(FILE *fout, int character):
   - Encodes a single character using the Huffman tree.
   - Writes the Huffman codes to the output file using `writeBit`.
8. encode(const char *ifile, const char *ofile):
   - Main function for encoding.
   - Determines character frequencies, builds the Huffman tree, and encodes the input data.
   - Calls `writeHeader` to write header information.
9. decode(const char *ifile, const char *ofile):
   - Main function for decoding.
   - Reads the header, builds the Huffman tree, and decodes the input data.

10. decodeBitStream(FILE *fin, FILE *fout):
   - Decodes the bitstream using the Huffman tree.
   - Writes the decoded characters to the output file using `fwrite`.
11. writeBit(FILE *f, int bit):
   - Writes a single bit to the output file.
   - Manages a buffer for writing bits efficiently.
12. readBit(FILE *f):
   - Reads a single bit from the input file.
   - Manages a buffer for reading bits efficiently.
13. writeHeader(FILE *f):
   - Writes the header information to the compressed file during the encoding process.
   - Includes details such as the original size of the input data and the Huffman tree structure.
14. readHeader(FILE *f):
   - Reads the header information from the compressed file.
   - Extracts details like the original size of the input data and the Huffman tree structure.
15. main(int argc, char **argv):
   - The entry point of the program.
   - Parses command-line arguments and calls either `encode` or `decode` based on the specified operation.

ADTs for the Data Structures Used

Array: For the frequency and parent_indx arrays, we have implemented a basic array ADT. This ADT allows us to store data in an organized manner and provides operations such as accessing elements by their index.

Priority Queues: The priority queue is implemented implicitly using an array of nodes (**nodes** array), and the **addNode** function is responsible for maintaining the priority queue property using linked list.

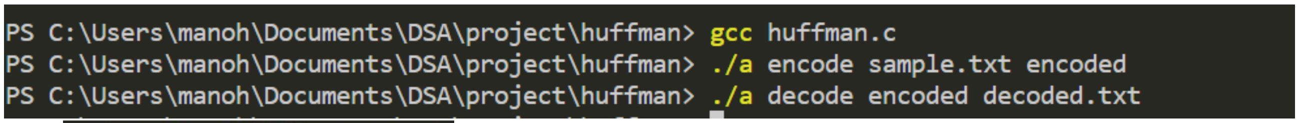
Stack: To temporarily store the path taken while traversing the Huffman tree during encoding of each characters, we have implemented a basic stack ADT for the stack array. This ADT allows us to push, pop, and peek elements from the stack in LIFO (Last In, First Out) order.

Buffer : An array (buffer) used for buffering bits during writing to the output file .To efficiently write bits to the output file.

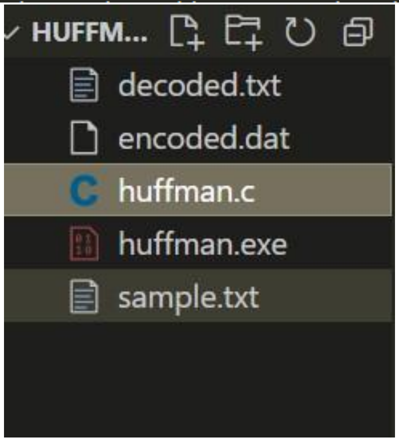
Tree Construction: The functions addNode, addLeaves, buildTree, and allocateTree are involved in constructing the Huffman tree. To build the Huffman tree based on character frequencies.

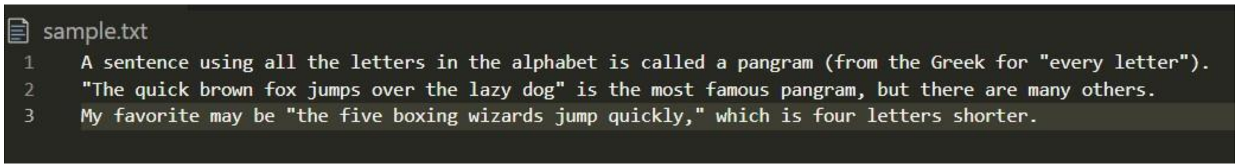
File: To interact with the file system and process the input and output data, we have utilized standard file operations for opening, reading, and writing input and output files. These operations enable us to manage files effectively and handle the input and output data streams.

Output Screenshots:-



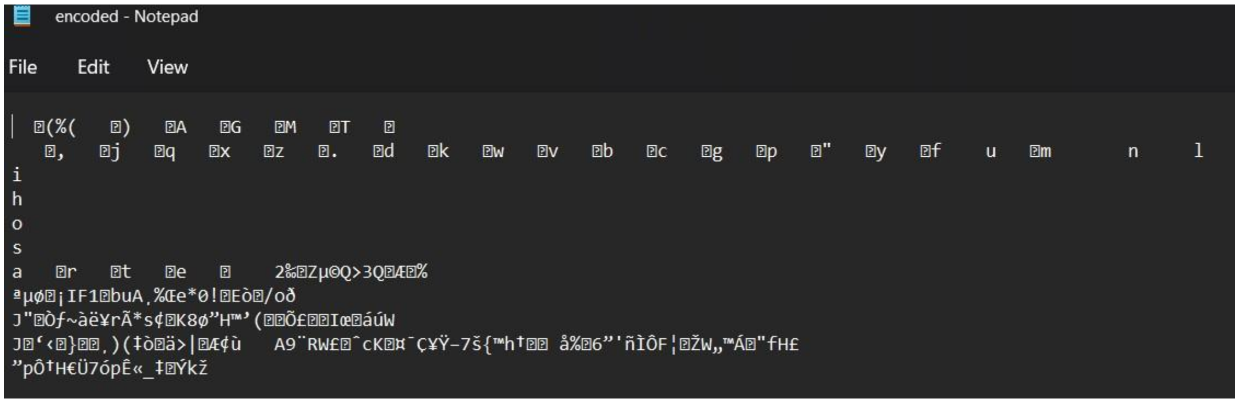
```
PS C:\Users\manoh\Documents\DSA\project\huffman> gcc huffman.c
PS C:\Users\manoh\Documents\DSA\project\huffman> ./a encode sample.txt encoded
PS C:\Users\manoh\Documents\DSA\project\huffman> ./a decode encoded decoded.txt
```





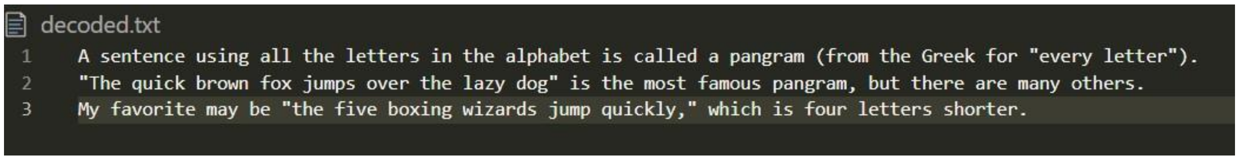
sample.txt

```
A sentence using all the letters in the alphabet is called a pangram (from the Greek for "every letter").
"The quick brown fox jumps over the lazy dog" is the most famous pangram, but there are many others.
My favorite may be "the five boxing wizards jump quickly," which is four letters shorter.
```



encoded - Notepad

```
| ▯(%(   ▯)   ▯A   ▯G   ▯M   ▯T   ▯
  ▯,   ▯j   ▯q   ▯x   ▯z   ▯.   ▯d   ▯k   ▯w   ▯v   ▯b   ▯c   ▯g   ▯p   ▯"   ▯y   ▯f    u   ▯m      n     l
i
h
o
s
a   ▯r   ▯t   ▯e   ▯    2‰▯Zµ©Q>3Q▯Æ▯%
ªµø▯¡IF1▯buA¸%Œe*0!▯Eò▯/oð
J"▯Òƒ~àë¥rÃ*s¢▯K8ø”H™’(▯▯Õ£▯▯Iœ▯áúW
J▯‘<▯}▯▯¸)(‡ò▯ä>|▯Æ¢ù   A9¨RW£▯ˆcK▯¤¯Ç¥Ÿ–7š{™h†▯▯ å%▯6”'ñÌÔF¦▯ŽW„™Á▯"fH£
”pÔ†H€Ü7ópÊ«_‡▯Ýkž
```



decoded.txt

```
A sentence using all the letters in the alphabet is called a pangram (from the Greek for "every letter").
"The quick brown fox jumps over the lazy dog" is the most famous pangram, but there are many others.
My favorite may be "the five boxing wizards jump quickly," which is four letters shorter.
```