1. **Title of the project:** Data Compression Using Huffman Coding Algorithm.
2. **Statement of the Problem:** Design and Implement Huffman Coding Algorithm to perform Text Compression.
3. **Objectives and scope of the project:**

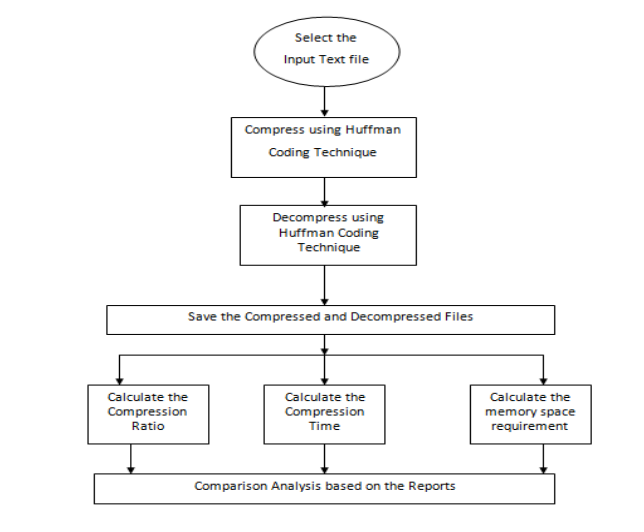
* Audio data compression reduces the transmission bandwidth and storage requirements of audio data. Audio compression algorithms are implemented in software as audio codecs. Lossy audio compression algorithms provide higher compression at the cost of fidelity, are used in numerous audio applications. These algorithms almost all rely on psychoacoustics to eliminate less audible or meaningful sounds, thereby reducing the space required to store or transmit them.
* Video compression uses modern coding techniques to reduce redundancy in video data. Most video compression algorithms and codecs combine spatial image compression and temporal motion compensation. Video compression is a practical implementation of source coding in information theory. In practice most video codecs also use audio compression techniques in parallel to compress the separate, but combined data streams.
* Grammar-Based Codes: They can extremely compress highly-repetitive text, for instance, biological data collection of same or related species, huge versioned document collection, internet archives, etc. The basic task of grammar-based codes is constructing a contextfree grammar deriving a single string. Sequitur and Re-Pair are practical grammar compression algorithms which public codes are available.[2]

1. **Tools/Framework used:**

IDE**:** Visual Studio Code

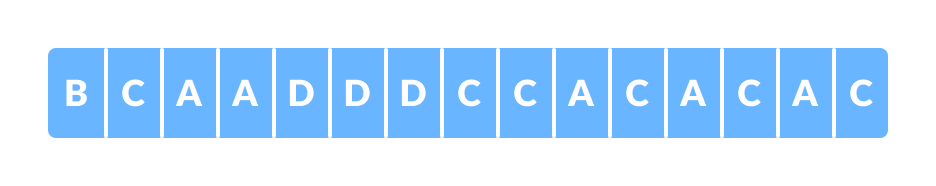
Language used: C

1. **Methodology / Design:[2]**

****

## 5.1 How Huffman Coding works?

Suppose the string below is to be sent over a network.



**Initial string**

Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of 8 \* 15 = 120 bits are required to send this string.

Using the Huffman Coding technique, we can compress the string to a smaller size.

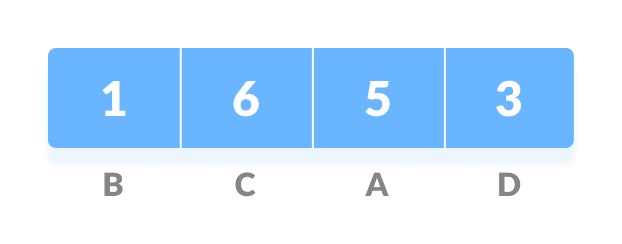
Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

Once the data is encoded, it has to be decoded. Decoding is done using the same tree.

Huffman Coding prevents any ambiguity in the decoding process using the concept of **prefix code** [1] ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.

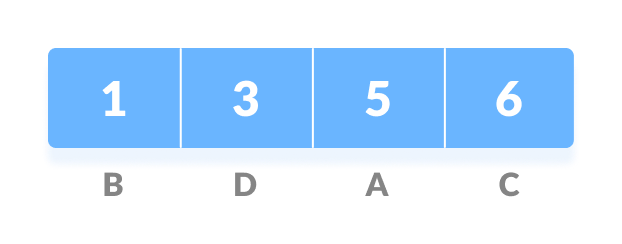
Huffman coding is done with the help of the following steps.

1. Calculate the frequency of each character in the string.



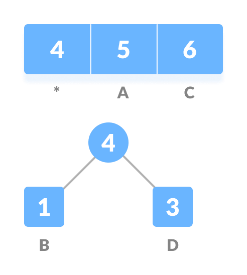
**Frequency of string[1]**

1. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.



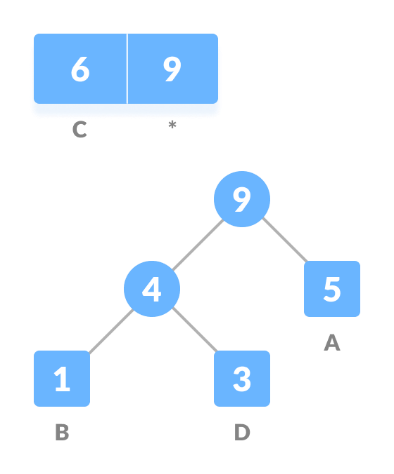
**Characters sorted according to the frequency**

1. Make each unique character as a leaf node.
2. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

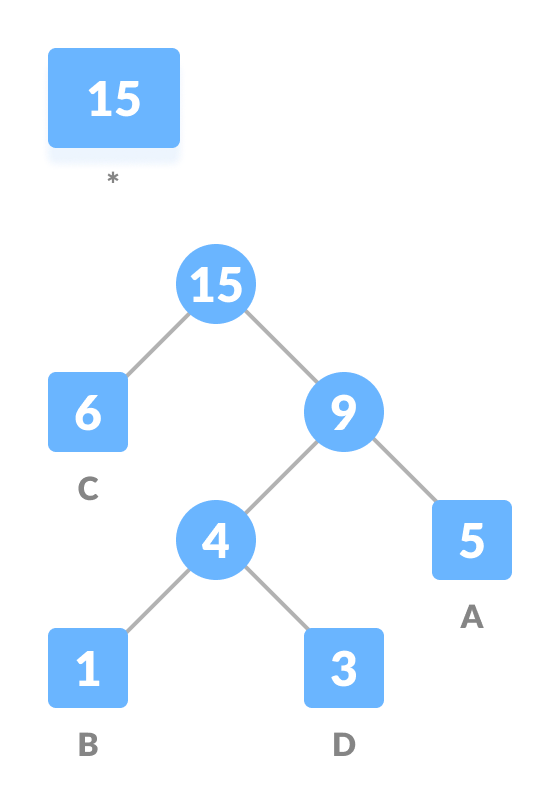


**Getting the sum of the least numbers[1]**

1. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (\* denote the internal nodes in the figure above).
2. Insert node z into the tree.
3. Repeat steps 3 to 5 for all the characters.

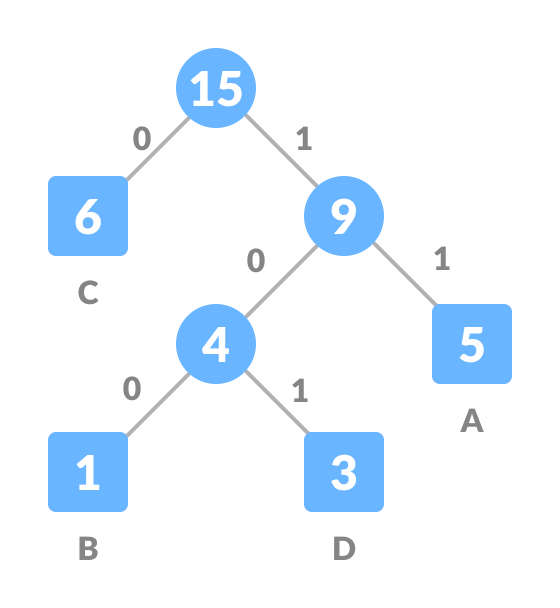


**Repeat steps 3 to 5 for all the characters**.



**Repeat steps 3 to 5 for all the characters.[1]**

1. For each non-leaf node, assign 0 to the left edge and 1 to the right edge.



**Assign 0 to the left edge and 1 to the right edge**

For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

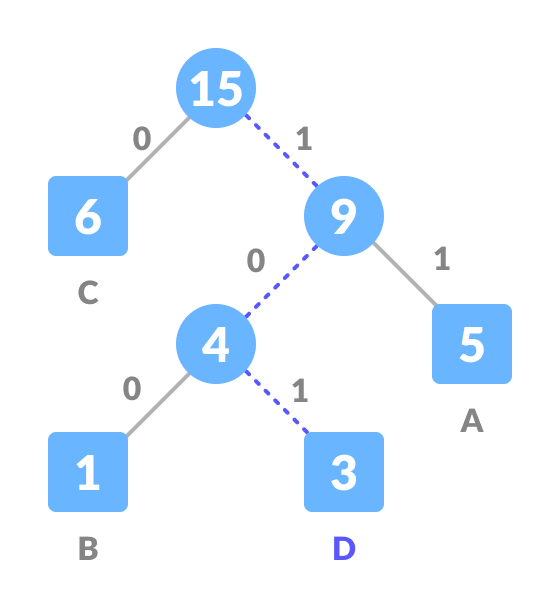
|  |  |  |  |
| --- | --- | --- | --- |
| Character | Frequency | Code | Size |
| A | 5 | 11 | 5\*2 = 10 |
| B | 1 | 100 | 1\*3 = 3 |
| C | 6 | 0 | 6\*1 = 6 |
| D | 3 | 101 | 3\*3 = 9 |
| 4 \* 8 = 32 bits | 15 bits |  | 28 bits |

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to 32 + 15 + 28 = 75.

## 5.2 Decoding the code:

For decoding the code, we can take the code and traverse through the tree to find the character.

Let 101 is to be decoded, we can traverse from the root as in the figure below.[1]

****

**Decoding.[1]**

## 5.3 Huffman Coding Complexity:

The time complexity for encoding each unique character based on its frequency is O(nlog n).

Extracting minimum frequency from the priority queue takes place 2\*(n-1) times and its complexity is O(log n). Thus the overall complexity is O(nlog n).

## Huffman Coding Applications:

* Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc.[3][4]
* For text and fax transmissions.

1. **Algorithm / Pseudocode:**

**Huffman(text)[5]**

create a priority queue Q consisting of each unique character.

sort then in ascending order of their frequencies.

for all the unique characters:

create a newNode

extract minimum value from Q and assign it to leftChild of newNode

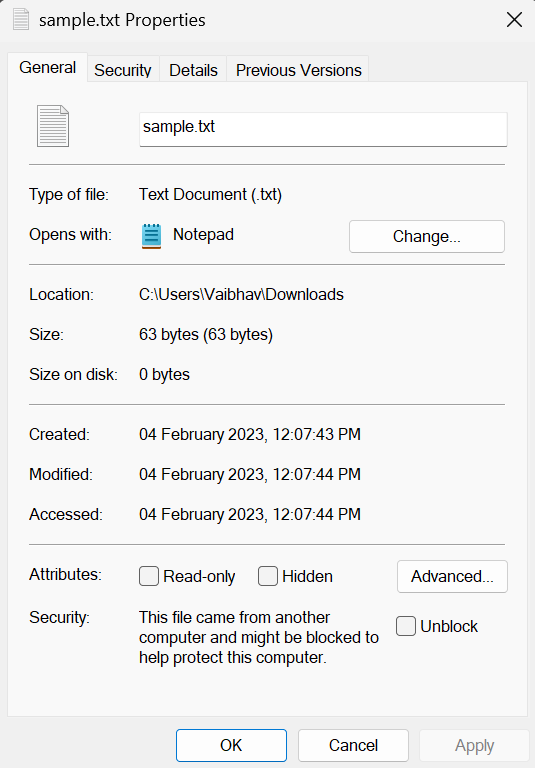
extract minimum value from Q and assign it to rightChild of newNode

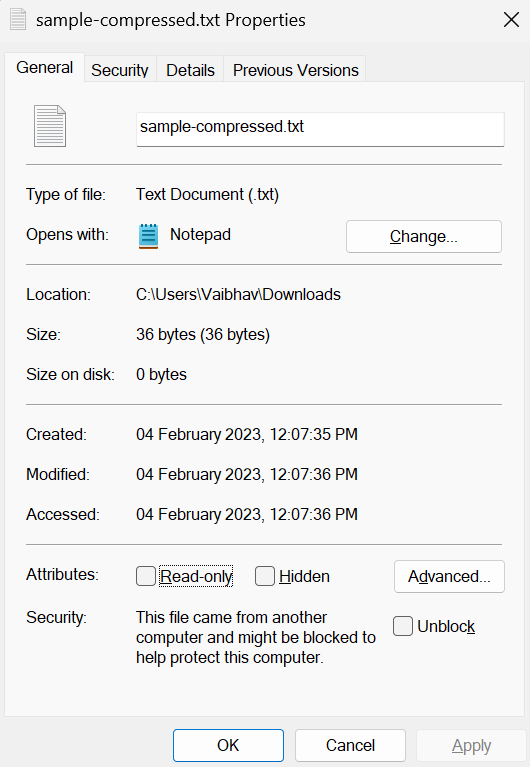
calculate the sum of these two minimum values and assign it to the value of newNode

insert this newNode into the tree

return rootNode

1. **Screenshots:**

****

****

1. **Application and Future enhancement of the project:**
   1. **Application:**

* Data compression: Huffman coding is commonly used to compress data in lossless format.[7]
* Image and audio compression: Huffman coding is a popular algorithm used in image and audio compression, which helps reduce the size of the data while preserving the quality.[4][3]
* Wireless communication: Huffman coding is widely used in wireless communication systems to reduce the transmission data rate and improve the efficiency of data transmission.
* Data storage: Huffman coding is used in data storage systems to reduce the amount of storage space required for storing the data.
* Text compression: Huffman coding is often used to compress text data, such as in data transmission and storage of electronic documents.
* Digital signal processing: Huffman coding is used in digital signal processing to reduce the amount of data required to represent the signal, which helps reduce the computational load.[6]
  1. **Future Enhancement:**

In our project we have performed text compression using Huffman coding algorithm, we can enhance it further to perform image and audio compression in any format. We can also extend the use of Huffman coding algorithm to apply to the above applications.

1. **Conclusion:**

In conclusion, Huffman coding is a well-established and efficient data compression algorithm that has found widespread applications in various fields such as image and audio compression, wireless communication, data storage, and text compression. The algorithm works by assigning shorter codewords to more frequently occurring symbols and longer codewords to less frequently occurring symbols, thereby reducing the number of bits required to represent the data. With its ability to achieve high compression ratios while preserving the quality of the data, Huffman coding has proven to be a valuable tool in the field of data compression.

1. **References:**

[1] Suherman, Suherman & Siahaan, Andysah Putera Utama. (2016). Huffman Text Compression Technique. International Journal of Computer Science and Engineering. 3. 103- 108. 10.14445/23488387/IJCSE-V3I8P124.

[2] Moffat, Alistair. (2019). Huffman Coding. ACM Computing Surveys. 52. 1-35. 10.1145/3342555. [3] P., Rachit. (2016). A fast and Improved Image Compression Technique using Huffman Coding.

[4] Ashida, S. & Kakemizu, H. & Nagahara, Masaaki & Yamamoto, Yutaka. (2004). Sampled-data audio signal compression with Huffman coding. 972 - 976 vol. 2.

[5] Butt, Sadia. (2015). Huffman coding. 1. 9.

[6] S A Chouakri et al 2013 Wavelet transform and Huffman coding based electrocardiogram compression algorithm: Application to telecardiology. J. Phys.: Conf. Ser. 454 012086.

[7] Habib, Ahasn & Islam, Mohammed J & Rahman, M.. (2018). Huffman Based Code Generation Algorithms: Data Compression Perspectives. Journal of Computer Science. 14. 1599-1610. 10.3844/jcssp.2018.1599.1610.