HUFFMAN ALGORITHM

**Abstract:**

Huffman Algorithm, a data compression algorithm was developed by David Huffman in 1951. This technique is the mother of all data compression algorithms. The greatest feature of Huffman Algorithm is that it allows for lossless data compression. Which means the original files or data can be regenerated from the encoded data without any loss of useful information. Huffman tree is a specific method which produces codes that are uniquely decodable, i.e. no code is the prefix of another code. This allows for error-free decoding.

The basic idea is to find the frequency of each character in some data and use a tree to store the characters as well as their frequency. Characters having large frequency get shorter codes while those with small frequency are encoded in longer codes. The code for each character can be generated simply by traversing the tree from the root to the characters respective node.

**Description:**

The implementation of Huffman Algorithm in this project compresses a string of text which is taken as input. The text is then encoded and displayed along with other information such as compression ratio and percentage. The major data structures used in this implementation are **“Unordered Maps”**, **“Priority Queue”**, and **“Tree”**.

Huffman Coding Implementation:

The technique works by creating a binary tree of nodes. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the character itself, the weight (frequency) of the character. Internal nodes contain character weight and links to two child nodes. As a common convention, bit ‘0’ represents following the left child and bit ‘1’ represents following the right child.

Priority queue is used for building the Huffman tree where the node with lowest frequency is given the highest priority. The complete steps are:

1. Create a leaf node for each character and add them to the priority queue
2. While there is more than one node in the queue:

* Remove the two nodes of highest priority (lowest frequency) from the queue
* Create a new internal node with these two nodes as children and with frequency equal to the sum of the two nodes’ frequencies.
* Add the new node to the priority queue.

1. The remaining node is the root node and the tree, is complete.

The path from root to any leaf node stores the optimal prefix code (also called Huffman code) corresponding to the character associated with that leaf node.

Complexity Analysis:

Efficient Priority Queue require **O (log n)** time for insertion. Unordered Maps have a constant time complexity **O(n).** The Huffman tree has n leaves and 2n – 1, nodes as it is a complete tree. Thus, the total complexity of this implementation of Huffman is **O (n log n)**. Where n is the number of characters.