

**DATA STRUCTURE PROJECT**

**HUFFMAN TEXT COMPRESSION**

**CODING**

**TEAM MEMBERS**

* **JUNAID SAYANI (21K-3401) (LEADER)**
* **FATIMA SALEEM (21K-3178)**
* **SHAFAQ SALEEM (21K-3198)**

**Description:**

Text compression means to convert the text in some other format. It is usually performed to reduce the size of file and to reduce the transmitting time. Text compression works by finding similar strings within a text file, and replacing those strings with a temporary binary representation to make the overall file size smaller. Lossless Compression reduces a file's size with no loss of quality. For lossless compression, we rewrite the data of the original file in a more efficient way mostly in the form of binary string. For encoding technique, variable-length encoding will be used as it requires less space to store a file.

**Methodology:**

For the compression of text files, Huffman Coding Algorithm will be used while retaining the uniqueness of the compressed text. For retaining the uniqueness of the text, the Prefix rule will be used; that is, to make sure that the algorithm generates uniquely decodable code and that none of the codes are prefixed to the other because at some point during the decoding process, two or more characters may have the same prefix of code. The prefix rule, is used to avoid collisions in our encoding and decoding.

**Data Structures Involved:**

1. Priority Queue will be used where every item has a priority associated with it and element with high priority is de-queued first.
2. Binary Tree will be used where tree is complete and the item at root must be minimum along all the items present in heap.
3. Min Heap implementation will be used, for faster sorting (Heap Sort), for faster re-arranging of inserted data at leaf nodes (Heapify) and obviously for priority queues.

**Steps to Build Huffman’s (Binary) Tree:**

1. Create a leaf node for every character in the input. Build a Minimum Heap of all leaf nodes.
2. For the Minimum Heap, get the top two nodes (say N1 and N2) with minimum frequency.
3. Create a new internal node N3 with frequency equal to the sum of frequency of nodes N1 and N2. Make N1 as the left child of N3 and N2 as the right child of N3. Add this new node N3 to the Minimum Heap.
4. Repeat steps 2 and 3 till the point, the Minimum Heap has only one node.

**Steps to print codes from Huffman Tree:**Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered. All the left nodes are given value of 0, and right nodes value of 1, hence all letters are represented as combination of unique binary code which can help us traverse the tree for decoding back to original.

**Time Complexity: O(nlogn)**

Time Complexity is O(nlogn) because we are extracting minimum nodes 2\*(n-1) times and each time whenever a node is being extracted from the heap then a function called heapify () (heapify is a benefit of heap, decreases our insertion and deletion time, and rearranging time, it simply makes our code efficient, priority queues are implemented with ease due to heapify) is being called to rearrange the element according to their priority. Heapify method takes O(logn) time.

**Features:**

* Decreases file size (compressing algorithm)
* Greedy Algorithm, combined with faster data structures result in an efficient time saving encoding
* Unique Binary Codes for letters, help avoid collisions, implements prefix-rule.
* Decoding of Lossless compression, quality isn’t compromised