Name: - Yash Vardhan Sharma **Enrolment no: - BT21EEE101**

MINOR PROJECT

(DATA STRUCTURES)

HUFFMAN CODING IMPLEMENTATION

(Using Binary Tree, Priority Queues, Hashmaps)

Huffman Coding is one approach followed for **Text Compression**. Text compression means reducing the space requirement for saving a particular text. Huffman Coding is a lossless data compression algorithm, i.e., it is a way of compressing data without the data losing any information in the process. It is useful in cases where there is a series of frequently occurring characters.

Working of the algorithm: -

B	**C**	**A**	**A**	**D**	**D**	**D**	**C**	**C**	**A**	**C**	**A**	**C**	**A**	**C**

Each character takes 8 bits and here in the table, total of 15 characters are present, 15*8 =120 bits. We can compress this size using **Huffman Coding** algorithm.

Huffman Coding creates a tree by calculating the frequencies of each character of the string and then assigns them some unique code so that we can retrieve the data back using these codes.

We can achieve this using the following steps:

- Begin with calculating the frequency of each character value in the given string.

Frequency of String

1	**5**	**6**	**3**
B	**C**	**A**	**D**

- Sort the characters in ascending order concerning their frequency and store them in a priority queue, say **PQ**.

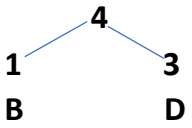
- Each character should be considered as different leaf node.

1	3	5	6
B	D	A	C

Characters sorted according to the frequency

- Make an empty node, say **z**. The left child of z marked as the minimum frequency and right child, the second minimum frequency. The value of z is calculated by summing up the first two frequencies.

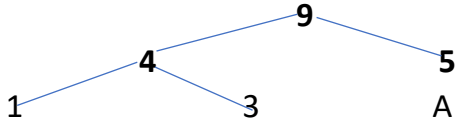
4	5	6
.	A	C



Getting the sum of the least numbers

- Now, remove the two characters with the lowest frequencies from the priority queue Q and append their sum to the same.
- Simply insert the above node z to the tree.
- For every character in the string, repeat steps 3 to 5.

6	9
C	.



- Repeat the 3 and 5 step for all the characters and assign 0 to the left edge and 1 to the right edge.

The size of the table: -

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

Size before encoding: 120 bits
Size after encoding: 32 + 15 + 28 = 75 bits

- While **encoding**, use the Hashmap for storing the data, alphabets/characters as the key and codes as the values.
- While **decoding**, use another hashmap for storing the data in reverse manner, codes as the keys and alphabets as the values.

Time Complexity: -

- In case of encoding, inserting each character into the priority queue takes O(log n) time. Therefore, for the complete array, the time complexity becomes O(nlog n).
- For storing the values or generating the huffmanCodes, we are using unordered map (hashmap). This added the additional complexity of O(n).
- The process takes place for the decoding in reverse manner.
- The total time complexity becomes: O(n + nlogn) = O(nlogn).

Applications: -

- They are used for transmitting fax and text.
- They are used by conventional compression formats like PKZIP, GZIP, etc.