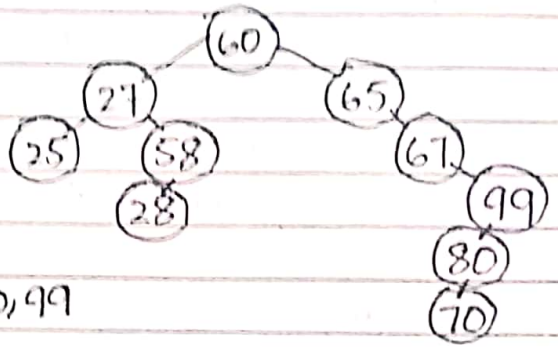
Dated:

TREE TRANSVERSING METHODS:

There are three types of tranversing method;
As we consider given tree:



Inorder tranversing:
Sequence always be
Left, Root Node, Right
i.e;
25, 27, 28, 58, 60, 65, 67, 70, 80, 99

Pre order tranversing:
sequence will be
Node, Left, Right
i.e.
60, 27, 25, 58, 28, 65, 67, 99, 80, 70

Post Order transversing:
sequence will be
left, Right, Node
i.e
25, 28, 58, 27, 70, 80, 99, 67, 65, 60

Inorder: seedha

Pre: Node pehle

Post: Node akhir mein.

ENCODING SCHEME:

Unicode: is a system usually of 32 bits, within unicode each character is provided a certain decimal number for its identification.

ASCII: is a system usually consist 8 bits per character of a message
Upper case letter : 65 - 90
Lower case letter : 97 - 122

Hauffman: is a system which consist of lesser bits and are widely used in compression file method. aka Greedy technique.

Dated:

HAUFFMAN / GREEDY TECHNIQUE:

By using this encoding scheme we can store a message in lesser bits then ~~we use in oth~~ other encoding scheme.

for example:

if we have a message with following value stored

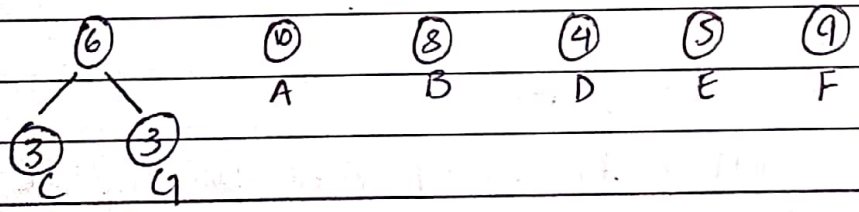
A	B	C	D	E	F	G
10	8	3	4	5	9	3

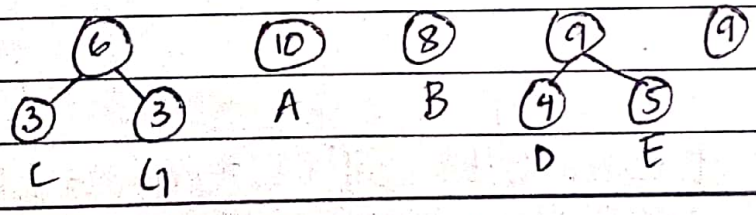
firstly, we identify that how many characters are there and how many times. following are the frequency of character and no of character used in the message. for the making of tree we arrange it in following order; NOte that no's are the frequency of character.

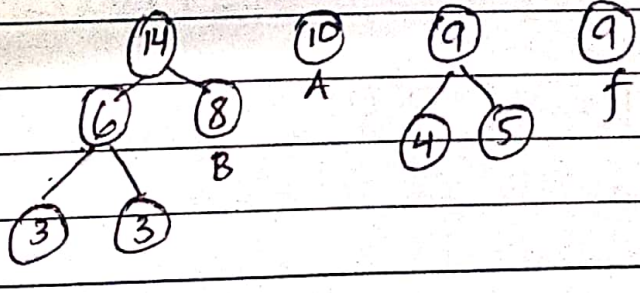
(10)	(8)	(3)	(4)	(5)	(9)	(3)
A	B	C	D	E	F	G

NOW,

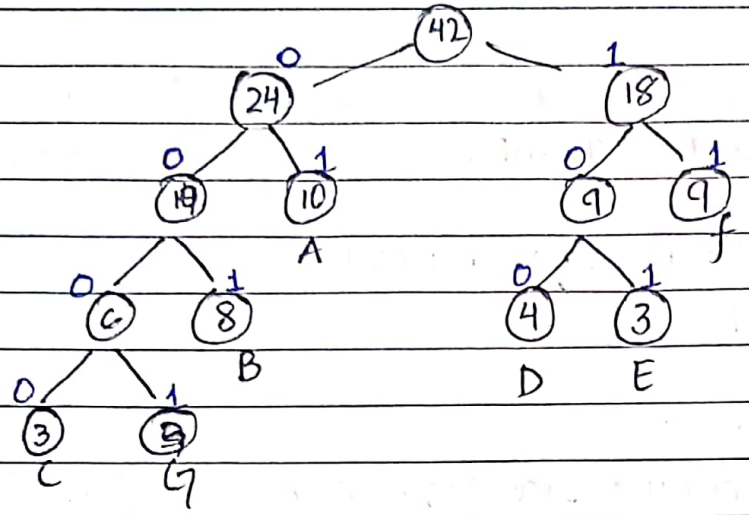
By using lowest or minimum bit heapstore method we arrange it in tree







Dated:



Now, give left side child name "0"
give right side child name "1"

Comparision:

Char	ASCII	Hauffman
A	0100 0001	01
B	0100 0010	001
C	0100 0011	0000
D	0100 0100	100
E	0100 0101	101
F	0100 0110	11
G	0100 0111	0001

As we can clearly see that with greedy technique we use minimum required amount of bits.

for example

Message= A AA B BA DEG

In Ascii= total used bits will be 72 bits. = 8x9

In Hauffman = total used bits will be 24 bits.

010101001001011001010001

1.8 Data Compression

For the purpose of storing or transferring data, it is often helpful (and sometimes mandatory) to reduce the size of the data involved while retaining the underlying information. The technique for accomplishing this is called **data compression.** We begin this section by considering some generic data compression methods and then look at some approaches designed for specific applications.

Generic Data Compression Techniques

Data compression schemes fall into two categories. Some are **lossless,** others are **lossy.** Lossless schemes are those that do not lose information in the compression process. Lossy schemes are those that may lead to the loss of information. Lossy techniques often provide more compression than lossless ones and are therefore popular in settings in which minor errors can be tolerated, as in the case of images and audio.

In cases where the data being compressed consist of long sequences of the same value, the compression technique called **run-length encoding,** which is a lossless method, is popular. It is the process of replacing sequences of identical data elements with a code indicating the element that is repeated and the number of times it occurs in the sequence. For example, less space is required to indicate that a bit pattern consists of 253 ones, followed by 118 zeros, followed by 87 ones than to actually list all 458 bits.

Another lossless data compression technique is **frequency-dependent encoding,** a system in which the length of the bit pattern used to represent a data item is inversely related to the frequency of the item's use. Such codes are examples of variable-length codes, meaning that items are represented by patterns of different lengths as opposed to codes such as Unicode, in which all symbols are represented by 16 bits. David Huffman is credited with discovering an algorithm that is commonly used for developing frequency-dependent codes, and it is common practice to refer to codes developed in this manner as **Huffman codes.** In turn, most frequency-dependent codes in use today are Huffman codes.

As an example of frequency-dependent encoding, consider the task of encoded English language text. In the English language the letters *e*, *t*, *a*, and *i* are used more frequently than the letters *z*, *q*, and *x*. So, when constructing a code for text in the English language, space can be saved by using short bit patterns to represent the former letters and longer bit patterns to represent the latter ones. The result would be a code in which English text would have shorter representations than would be obtained with uniform-length codes.

In some cases, the stream of data to be compressed consists of units, each of which differs only slightly from the preceding one. An example would be consecutive frames of a motion picture. In these cases, techniques using **relative**

encoding, also known as **differential encoding,** are helpful. These techniques record the differences between consecutive data units rather than entire units; that is, each unit is encoded in terms of its relationship to the previous unit. Relative encoding can be implemented in either lossless or lossy form depending on whether the differences between consecutive data units are encoded precisely or approximated.

Still other popular compression systems are based on **dictionary encoding** techniques. Here the term *dictionary* refers to a collection of building blocks from which the message being compressed is constructed, and the message itself is encoded as a sequence of references to the dictionary. We normally think of dictionary encoding systems as lossless systems, but as we will see in our discussion of image compression, there are times when the entries in the dictionary are only approximations of the correct data elements, resulting in a lossy compression system.

Dictionary encoding can be used by word processors to compress text documents because the dictionaries already contained in these processors for the purpose of spell checking make excellent compression dictionaries. In particular, an entire word can be encoded as a single reference to this dictionary rather than as a sequence of individual characters encoded using a system such as ASCII or Unicode. A typical dictionary in a word processor contains approximately 25,000 entries, which means an individual entry can be identified by an integer in the range of 0 to 24,999. This means that a particular entry in the dictionary can be identified by a pattern of only 15 bits. In contrast, if the word being referenced consisted of six letters, its character-by-character encoding would require 48 bits using 8-bit ASCII or 96 bits using Unicode.

A variation of dictionary encoding is **adaptive dictionary encoding** (also known as dynamic dictionary encoding). In an adaptive dictionary encoding system, the dictionary is allowed to change during the encoding process. A popular example is **Lempel-Ziv-Welsh (LZW) encoding** (named after its creators, Abraham Lempel, Jacob Ziv, and Terry Welsh). To encode a message using LZW, one starts with a dictionary containing the basic building blocks from which the message is constructed, but as larger units are found in the message, they are added to the dictionary—meaning that future occurrences of those units can be encoded as single, rather than multiple, dictionary references. For example, when encoding English text, one could start with a dictionary containing individual characters, digits, and punctuation marks. But as words in the message are identified, they could be added to the dictionary. Thus, the dictionary would grow as the message is encoded, and as the dictionary grows, more words (or recurring patterns of words) in the message could be encoded as single references to the dictionary.

The result would be a message encoded in terms of a rather large dictionary that is unique to that particular message. But this large dictionary would not have to be present to decode the message. Only the original small dictionary would be needed. Indeed, the decoding process could begin with the same small dictionary with which the encoding process started. Then, as the decoding process continues, it would encounter the same units found during the encoding process, and thus be able to add them to the dictionary for future reference just as in the encoding process.

To clarify, consider applying LZW encoding to the message

XYX XYX XYX XYX

starting with a dictionary with three entries, the first being *x*, the second being *y*, and the third being a space. We would begin by encoding *xyx* as 121, meaning that the message starts with the pattern consisting of the first dictionary entry, followed by the second, followed by the first. Then the space is encoded to produce 1213. But, having reached a space, we know that the preceding string of characters forms a word, and so we add the pattern *xyx* to the dictionary as the fourth entry. Continuing in this manner, the entire message would be encoded as 121343434.

If we were now asked to decode this message, starting with the original three-entry dictionary, we would begin by decoding the initial string 1213 as *xyx* followed by a space. At this point we would recognize that the string *xyx* forms a word and add it to the dictionary as the fourth entry, just as we did during the encoding process. We would then continue decoding the message by recognizing that the 4 in the message refers to this new fourth entry and decode it as the word *xyx*, producing the pattern

xyx xyx

Continuing in this manner we would ultimately decode the string 121343434 as

xyx xyx xyx xyx

which is the original message.

Questions & Exercises

1. List four generic compression techniques.
2. What would be the encoded version of the message

   xyx yxxxy xyx yxxxy yxxxy

   if LZW compression, starting with the dictionary containing *x*, *y*, and a space (as described in the text), were used?
3. Why would GIF be better than JPEG when encoding color cartoons?
4. Suppose you were part of a team designing a spacecraft that will travel to other planets and send back photographs. Would it be a good idea to compress the photographs using GIF or JPEG's baseline standard to reduce the resources required to store and transmit the images?
5. What characteristic of the human eye does JPEG's baseline standard exploit?
6. What characteristic of the human ear does MP3 exploit?
7. Identify a troubling phenomenon that is common when encoding numeric information, images, and sound as bit patterns.