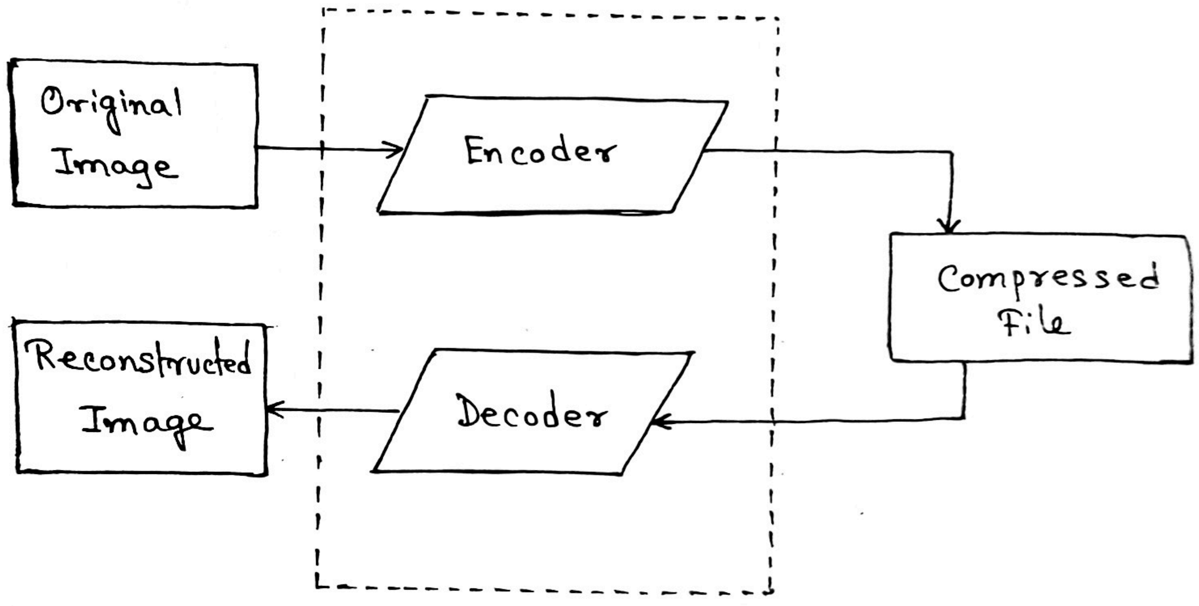
Image Compression :

Image compression is a process that makes image files smaller so that less memory space is required and less bandwidth is required while transferring the image. Image Compression most often works either by removing bytes of information from the image, or by using an image compression algorithm to rewrite the image file in a way that takes up less storage space.



Types of image compression :

There are mainly two types of image compression. They are,

i) Lossy Compression

ii) Lossless Compression.

Lossy Compression :

The lossy image compression techniques essentially lose some information with an acceptable compromise on image quality. That is lossy image compression technique retains the most significant information for the image without keeping every single pixel.

A lossy image compression method normally consists of two steps. In the first step the image data is transformed into the frequency domain using one of Discrete cosine, walsh-Hadamard or karhunen-loeve transforms. In the Second step a part of the information is dropped to get a compact data using some kind of quantization technique.

Lossless compression :

The lossless image compression techniques are methods to represent an image in a compact and efficient way without loosing any information of the original image. Some lossless image compression techniques are Huffman, arithmatic, run-length, dictionary-based coding techniques etc.

while lossless compression can reduce image file size by as much as 40% , it is still less effective than lossy compression for reducing image file size and optimizing images.

Run length encoding:

Run length encoding is a lossless compression algorithm that helps us encode larger runs of repeating items by only sending one item from the run and a counter showing how many times this item is repeated.

Run length encoding is useful when it comes to image compression, because images happen to have long runs pixels with identical colors.

Suppose we have an image input stream as "aaaabbaba". If we apply run length algorithm on the input stream the output would become "a4b2aba".

The compression ratio of a run length encoding can be expressed as,

$$\text{Compression ratio} = \frac{\text{length of uncompressed data}}{\text{length of compressed data}}$$

The efficiency of this encoding depends on,

- number of repeated character occurrences in data to be compressed
- average repeated character length.

Huffman encoding:

Huffman encoding is a data compression technique that is used to ~~comp~~ lossless compression of image file. This technique was developed by David A. Huffman. Huffman encoding uses a variable - length code for each data in the file. The code length are determined by the frequency of each data in the file. The most common data are assigned the shortest codes and the less common characters are assigned longer codes.

Working of Huffman encoding:

Suppose that we have the following sequence of data

10	12	16	16	17	17	17	12	12	16	12	16	12	16	12

Let each data occupies 8 bits. There are a total of 15 data in the above sequence. Thus a total of 8*15 = 120 bits are required.

Using Huffman encoding we can compress this sequence of data.

Step 1: Calculate the frequency of each data in the sequence.

1	6	5	3
10	12	16	17

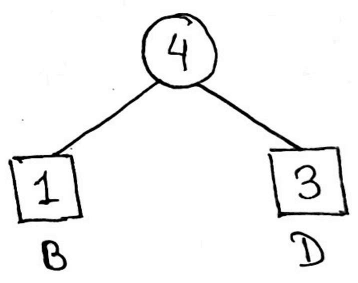
Step 2: Sort the data in increasing order of the frequenc
These are stored in a priority queue.

1	3	5	6
10	17	16	12

Step 3: Make each unique data as a leaf node.

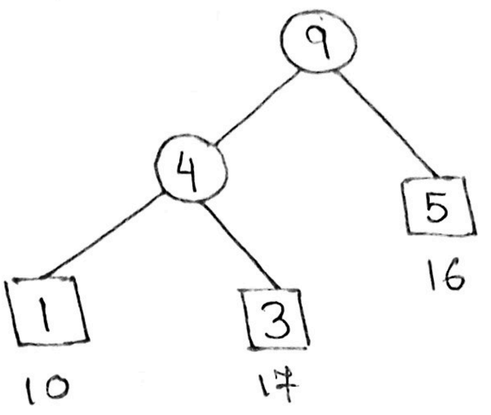
Step 4: Create an empty node Z. Assign the minimum frequency data to the left of Z and assign the Second minimum frequency to the right of Z. Set the value of Z as the Sum of two minimum frequencies.

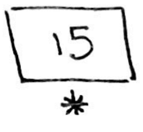
4	5	6
*	16	12

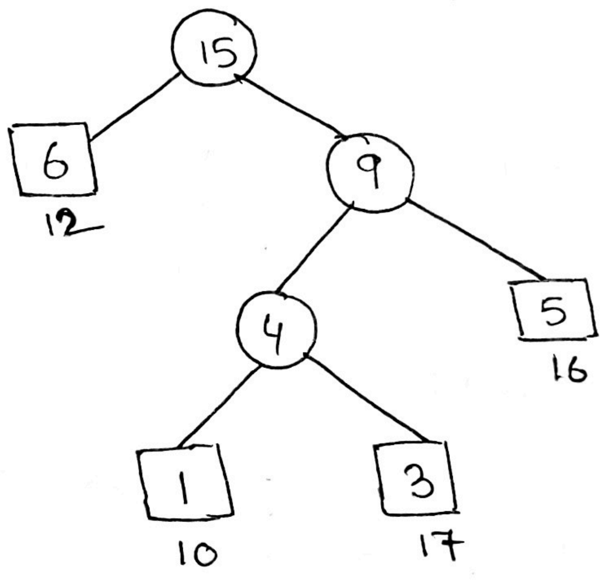


Step 5: Repeat step 3 and step 4 for all the data

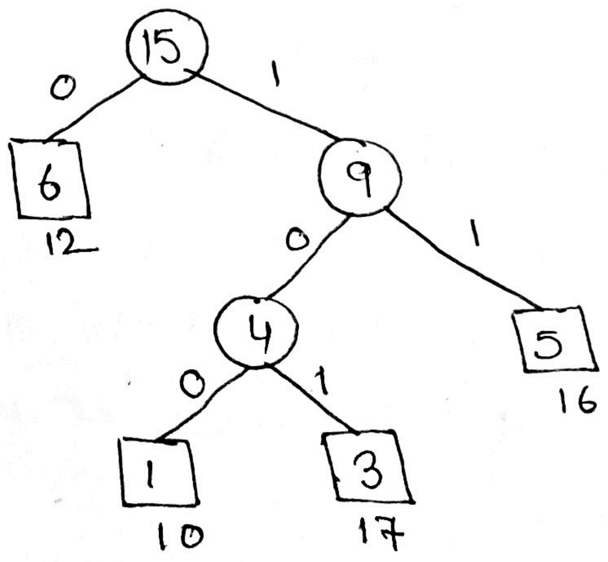
9	6
*	12







Step 6: For each non-leaf node assign 0 to the left edge and 1 to the right edge.



Thus the code for each data using Huffman coding is,

Data	Frequency	Code	Size
16	5	11	5×2 = 10
10	1	100	1×3 = 3
12	6	0	6×1 = 6
17	3	101	3×3 = 9
4×8 = 32 bits	15 bits		28 bits

After encoding the size of the data is reduced to 32 bits + 15 bits + 28 bits = 75 bits.

Arithmatic coding:

Arithmatic coding generates nonblock codes. In arithmatic coding, a one-to-one correspondence between Source Symbol and code words does not exist. Instead an entire Sequence Source Symbol is assigned a single arithmatic code.

The code word itself defines an interval of real numbers 0 and 1. As the number of Symbols in the message increases the interval used to represent it becomes Smaller and the number of bits required to represent the interval becomes larger.

Let us consider a five symbol sequence or message $a_1a_2a_3a_3a_4$. At the start of the coding process, the message is assumed to occupy the entire half-open interval $[0,1)$

Source Symbol	Probability	Initial Subinterval
a_1	0.2	$[0.0, 0.2)$
a_2	0.2	$[0.2, 0.4)$
a_3	0.4	$[0.4, 0.8)$
a_4	0.2	$[0.8, 1.0)$

According to the above table, this interval is subdivided initially into four regions based on the probabilities of each symbol.

The narrowed range is then subdivided in accordance with the original source symbol probabilities and the process continues with the next message symbol. In this manner, symbol a_2 narrows the subinterval to $[0.04, 0.08)$, a_3 further narrows it to $[0.056, 0.072)$ and so on. The final message symbol which must be reserved as a special end-of-message indicator narrows the range to $[0.06752, 0.0688)$ of course any number within this subinterval for example 0.068 can be used to represent the message.