HUFFMAN CODES

→ Data can be encoded efficiently using Huffman Codes.

→ It is a technique for compressing data; saving of 20% to 90% depending on the characteristics of the file being compressed.

→ Huffman's greedy algorithm uses a table of the frequencies of occurrence of each character - ~~to~~

→ Example:

Suppose we have 10^5 characters in a data file. Normal storage: 8 bits per character (ASCII) means 8×10^5 bits in the file. But we want to compress the file and store it compactly.

→ Suppose only 6 characters appear in the file:

	a	b	c	d	e	f	Total
Frequency	45	13	12	16	9	5	100

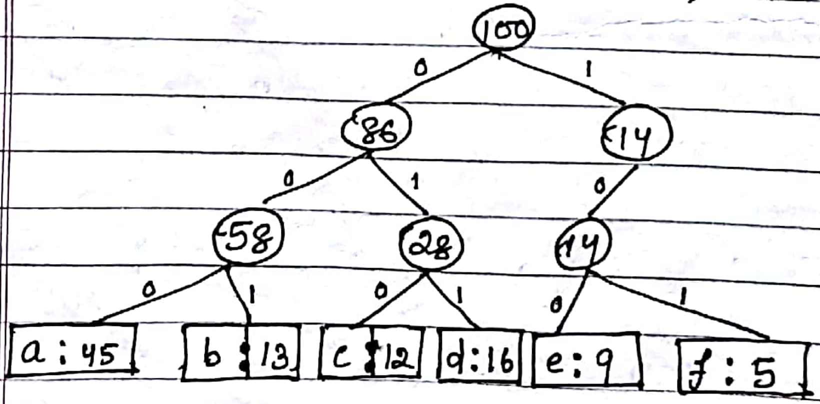
How can we represent the data in a compact way.

(i) Fixed length code: Each letter represented by an equal number of bits. With a fixed length code at least 3 bits per character:

For example:

a	000
b	001
c	010
d	011
e	100
f	101

For a file with 10^5 characters, we need 3×10^5 bits.



(Not optimal)

(ii) Variable length code:

Here each character is encoded according to the frequency of the characters.
The more ~~the~~ frequency ~~of~~ characters are short code words and less frequency characters are long code words.

For Example:

a	0
b	101
c	100
d	111
e	1101
f	1100

Number of bits $= (45\times1+13\times3\times12\times3+16\times3+9\times4+5\times4)*1000$

$= 2.24\times10^5$ bits

Thus 224,000 bits required to represent the file.

→ percentage of data compressed is:

$3,00000-224,000 = 76,000$

$76000/3,00000 \times 100 = 25\%$

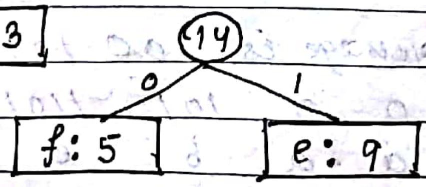
∴ 25% of data is compressed.
So, this is an optimal character code for this file.

→ Example:

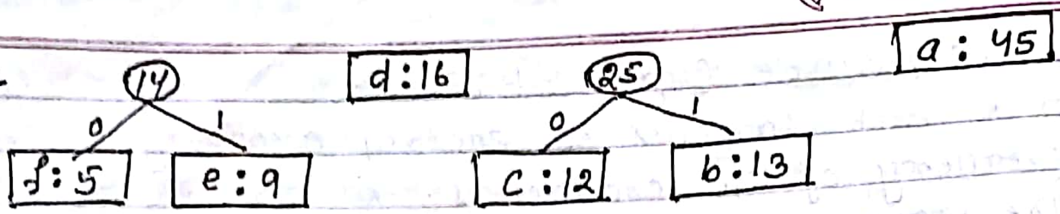
step-1 $\boxed{f:5}$ $\boxed{e:9}$ $\boxed{c:12}$ $\boxed{b:13}$ $\boxed{d:16}$ $\boxed{a:45}$

This algorithm is based on a reduction of a problem with n characters to a problem with n-1 characters. A new character replaces two existing one.

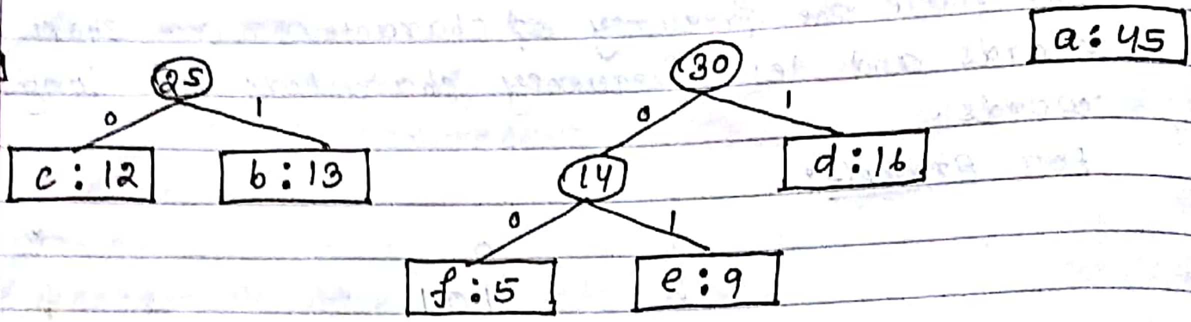
step-2 $\boxed{c:12}$ $\boxed{b:13}$ $\boxed{d:16}$ $\boxed{a:45}$



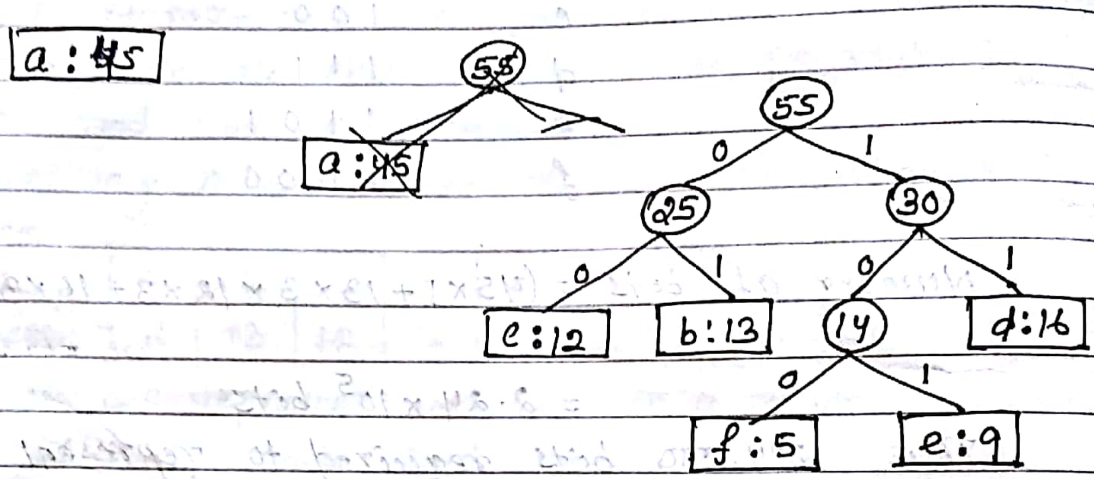
step-3



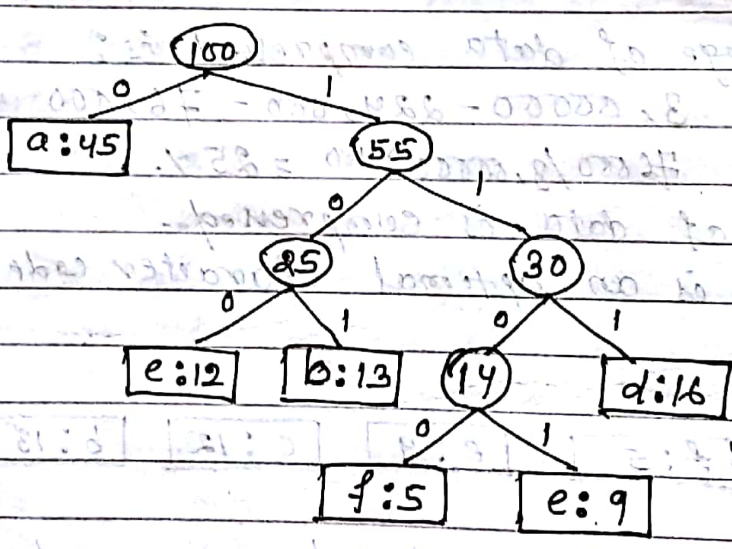
step-4



step-5



step-6



a	b	c	d	e	f
0	101	100	111	1101	1100

→ The encoded message is 00 101 1101. find original msg.

0 0 101 1101

a a b e

∴ The original message is aabe

Algorithm for Huffman Code:

HUFFMAN (C)

1. $n \leftarrow |C|$
2. $Q \leftarrow C$ — $O(n)$
3. for $i \leftarrow 1$ to $n-1$
4. do allocate a new node Z
5. $left[Z] \leftarrow x \leftarrow$ EXTRACT-MIN(Q)
6. $right[Z] \leftarrow y \leftarrow$ EXTRACT-MIN(Q)
7. $f[Z] \leftarrow f[x] + f[y]$
8. INSERT(Q, Z)
9. return EXTRACT-MIN(Q) ▷ Return the root of the tree

$\left.\begin{array}{l}\text{lines 3–8}\end{array}\right\}(n-1)$

Complexity:

→ For implementing Huffman's algorithm, we need a min priority queue.

→ The min priority queue can be implemented by binary min-heap.

→ A set C of n characters are initialized to Q which takes $O(n)$ times by using the BUILD-MIN-HEAP procedure.

→ Each time it will take 2 nodes whose frequency is minimum and it will delete the 2 nodes from the priority queue and insert a new node whose value is the sum of the frequency of the two deleted nodes which is called merging.

→ The for loop is running $(n-1)$ times

Each heap operation takes $O(\log n)$

So, total time required is $O(n \log n)$