Huffman Encoding

- Huffman Encoding is a lossless data compression algorithm.
- Compress data very efficiently 20% to 90%.
- Idea is to assign variable length code to input characters.
- Length of the assigned codes are based on the frequency of corresponding character.
- Most frequency character get the smallest code and least frequency character get largest code.

Characters	a	b	c	d	e	f
freq (1000)	45	13	12	16	9	5
fixed length code	000	001	010	011	100	101
variable length code	0	101	100	111	1101	1100

- fixed length - Need 3 bits to represent 6 characters.
  a = 000, b = 001, ... f = 101
  This method requires 300,000 bits to code entire file.

- Variable length

· = multiplication $(45.1 + 13.3 + 12.3 + 16.3 + 9.4 + 5.4)1000 = 224,000$ bits
= Save 25% approx.

- C is the alphabet set, for each char c in C attribute.
  c.freq denote freq of c in file.
  $d_T(c)$ denote depth of c's leaf in tree. It is also length of code word for each c.

- So no of bits required

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$ // Cost of the Tree T.

(AJAY RAWAT)

Prefix Codes (or Prefix free Codes)

- It means the codes (bit seq) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character.

- Prefix ensure that there is no ambiguity when decoding the generated bit stream.

- example - let there be 4 character a = 00, b = 01, c = 0, d = 1
  - This led to ambiguity because code assigned to 'c' is prefix of code assigned to 'a' and 'b'.

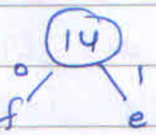
  - If compressed bit stream is 0001, decompressed o/p can be cccd, ccb, acd, ab.

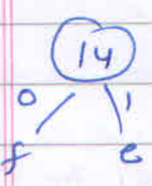
- It is a simple encoding and decoding

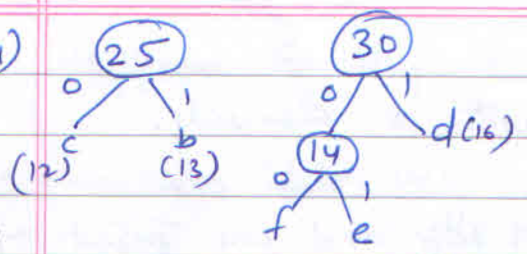
- There are mainly two points in Huffman Coding
  1) To build a Huffman Tree from input characters.
  2) Traverse the Huffman Tree and assign code to characters.

Construction of Huffman Tree

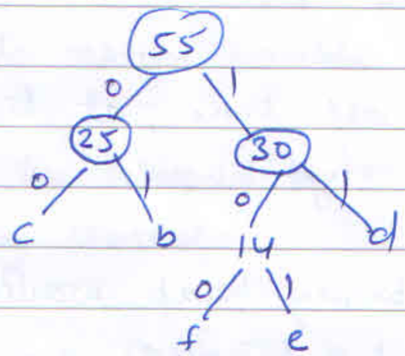
1) f:5 e:9 c:12 b:13 d:16 a:45

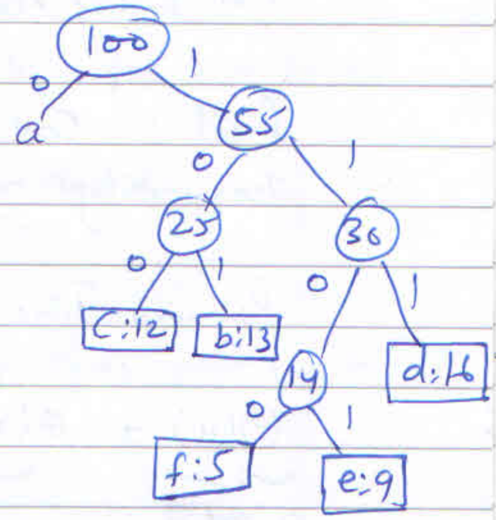
2) c:12 b:13  d:16 a = 45

3)  d:16  a:45

4) 

a:45

5) a:45 

(6) 

Algorithm

- C is a set of n characters
- Algorithm build Tree T corresponding to optimal code in bottom-up manner.
- It begin with set of |C| leaves and performs a sequence of |C|-1 merging operations to create the final tree.
- Algorithm uses min-priority queue Q, keyed on freq attribute

Huffman (C)

```
n = |C|
Q = C                          // Build min-heap.
for i = 1 to n-1
      allocate a new node z.
      z.left = x = Extract_Min(Q)     This order is arbitrary
      z.right = y = Extract_Min(Q)    Switching L, R of any node yield diff codes of same cost.
      z.freq = x.freq + z.freq
      Insert(Q, z)
return Extract-Min(Q)          //return the root of the tree
```

Analysis

- line 2 build a min-queue with n elements.

- n-1 times consist of two Exact_Min and one Insert operation

- final we Call Extract_Min last time, at this point Q has only one element left.

- Running Time for Q a binary heap would be

$$\underbrace{\Theta(n)}_{\text{Build queue}} + \underbrace{\Theta(n\log n)}_{\text{loop}} + \underbrace{O(1)}_{\text{Extract Min}} = \Theta(n\log n)$$

- Can reduced Running Time to $(n\log\log n)$ by replacing min heap with Van Emde Boase tree.

Greedy Algorithm

- Huffman's algo is an example of a greedy algo.

- Strategy is that Combining the two smallest nodes makes both of These character encoding one bit longer (added parent node above them)

- It is better choice to assign rare characters longer bit pattern Than the more frequent characters.

- It lead to an overall optimal Character encoding.

* Huffman Coding works well with range of frequency of character instead with same frequency of characters.