Huffman Codes

Huffman's greedy algorithm uses a table of the frequencies of occurence of each character to build up an optimal way of representing each character as a binary string.

Eg. If we have a 1,00,000 character data file that we wish to store compactly.

character	a	b	c	d	e	f
Freq. (x1000)	45	13	12	16	9	5

If we assign 8 bits to each character then size of file = 8,00,000 bits
= 1,00,000 bytes

If we use 3 bits to represent each character then size of files = 3,00,000 bits
This is called fixed length code.

If we use a variable length code, we can assign least bit code to the most frequent character.

eg,

character	a	b	c	d	e	f
Fixed length codeword	000	001	010	011	100	101
Variable length Codeword	0	101	100	111	1101	1100

size of file = 45,000 × 1 (for a one bit o) + 13,000 × 3 + 12,000 × 3 + 16,000 × 3 + 9,000 × 4 + 5,000 × 4

= 2,24,000 bits

* Variable length code is better than fixed length code.
* To assign binary code to each character the necessary condition is: A code should not be prefix of some other code. This is called prefix codes.
* If we use prefix codes, decoding will be correct

eg. 00 101 1101 = aabe

a b c d e f

1 11 101 1110 1111 100

To encode abc = 111101

To decode 111101 — aaac / bac / ec / aada

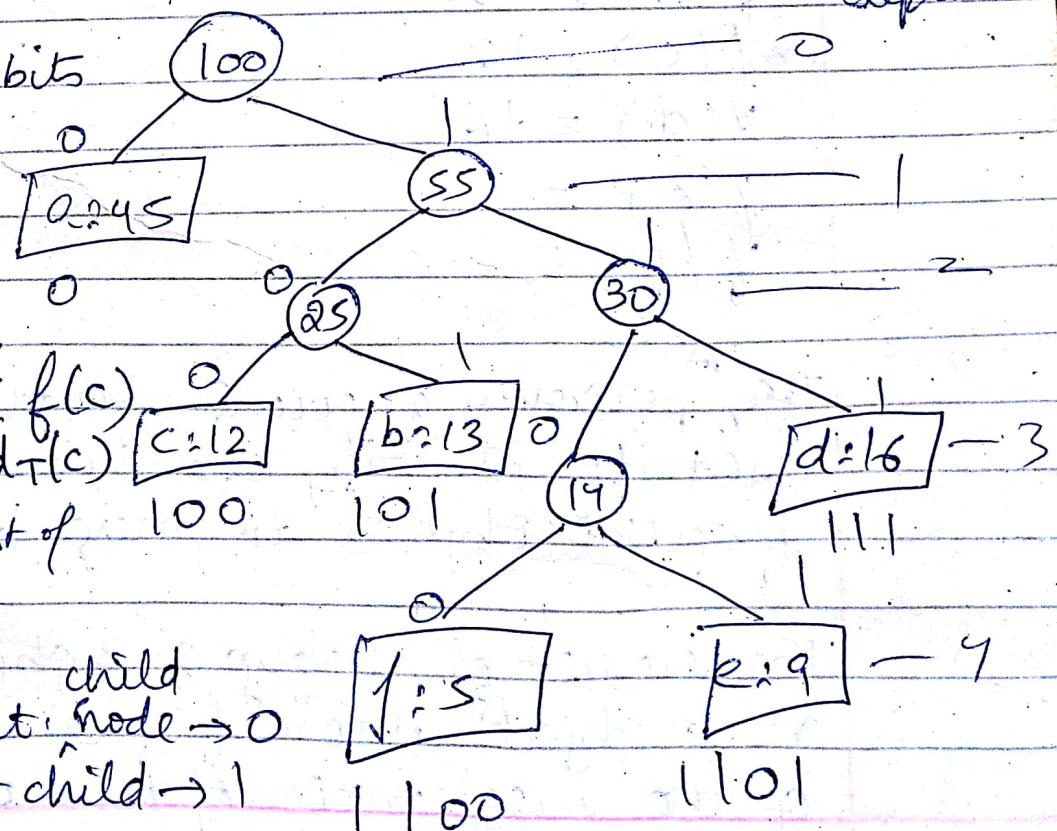
An optimal code for a file is always represented by a full binary tree in which every non leaf node has two children.

i.e. If we have 'n' type of characters to be decoded, then the binary tree contains 'n' leaf, one for each character and exactly (n-1) internal nodes.

The no. of bits reqd. to encode a file

$$B(T) = \sum_{c \in C} f(c)\, d_T(c)$$

This is cost of tree T.



Assign lt. child node → 0

" rt. child → 1

& then traverse for each char. from root node.

Construction of Huffman code:-

This algorithm builds the full binary tree T corresponding to the optimal code in a bottom up manner.

It begins with a set of |C| leaves and performs a sequence of |C|-1 merging operations to create the final tree where C is set from which characters are drawn.

In the following algo, we are assuming that C is the set of n characters and frequency of each character $x \in C$ is $f(x)$.

eg. $C = \{a, b, c, d, e, f\}$

$f(a) = 45$

$f(b) = 13$

$f(c) = 12$

$f(d) = 16$

$f(e) = 9$

$f(f) = 5$

A priority queue Q, keyed on f is used to identify the two least frequent objects to merge together.

The result of merging of 2 objects is a new object whose frequency is sum of the frequencies of two objects that were merged.

i=1

i=2

Running time

$\Rightarrow O(n \lg n)$

1 term (for loop) — no. of levels of binary tree

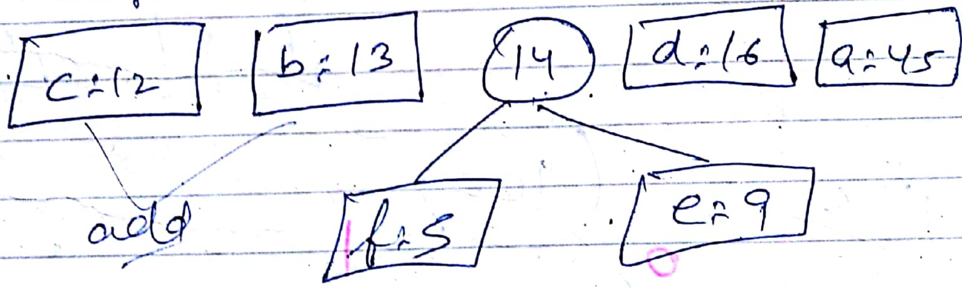
```
Huffman(C)
   n ← |C|
   Q ← C
for i ← 1 to n-1
do Z ← Allocate_Node()
     x ← left(z) ← Extract_Min(Q)
     y ← right(z) ← Extract_min(Q)
     f(z) ← f(x) + f(y)
     Insert(Q, z)
return Extract_Min(Q)
```

Arrange characters in ↑ing order of their frequencies

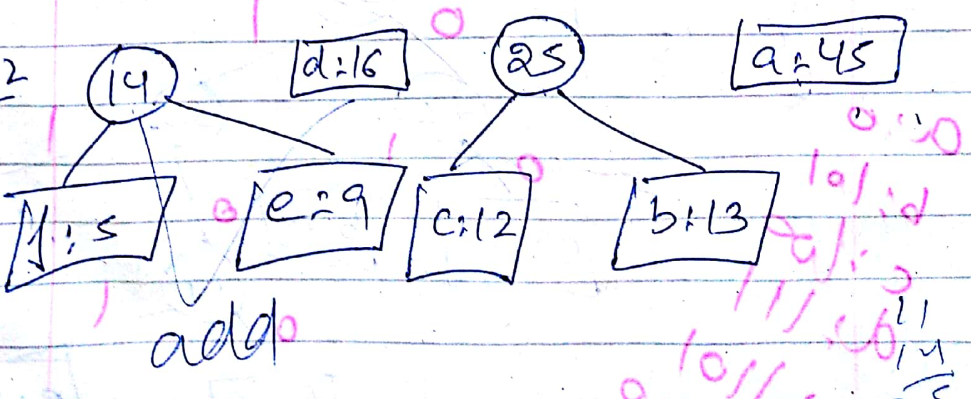
f:5	e:9	c:12	b:13	d:16	a:45

Now merge/add lowest 2 char. with lowest freq. which is Z and arrange this Z in proper posⁿ where Z has children which has that 2 char.
This goes on till n-1 times

i=1

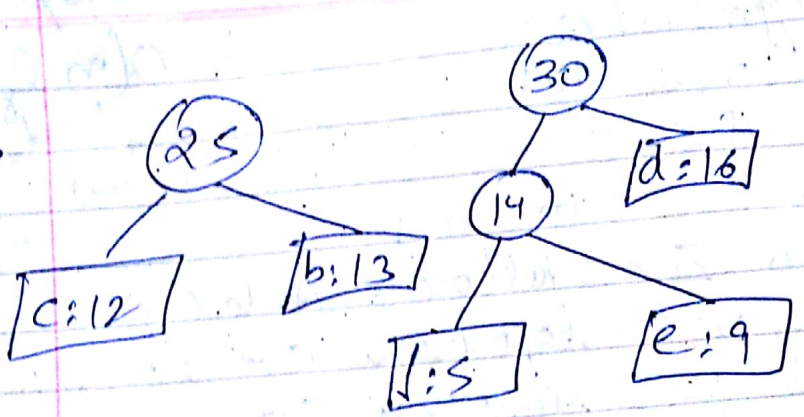


i=2

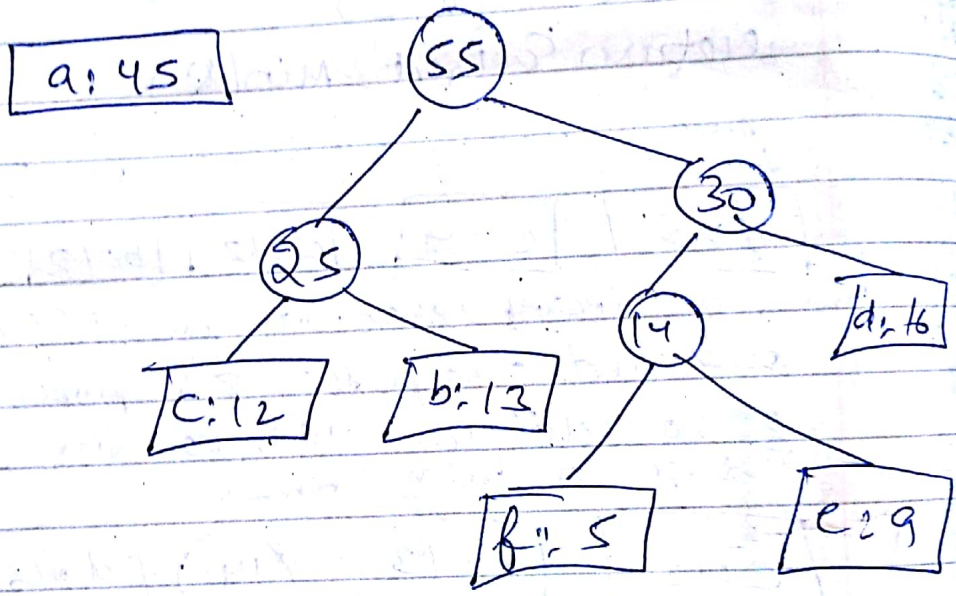


Q:

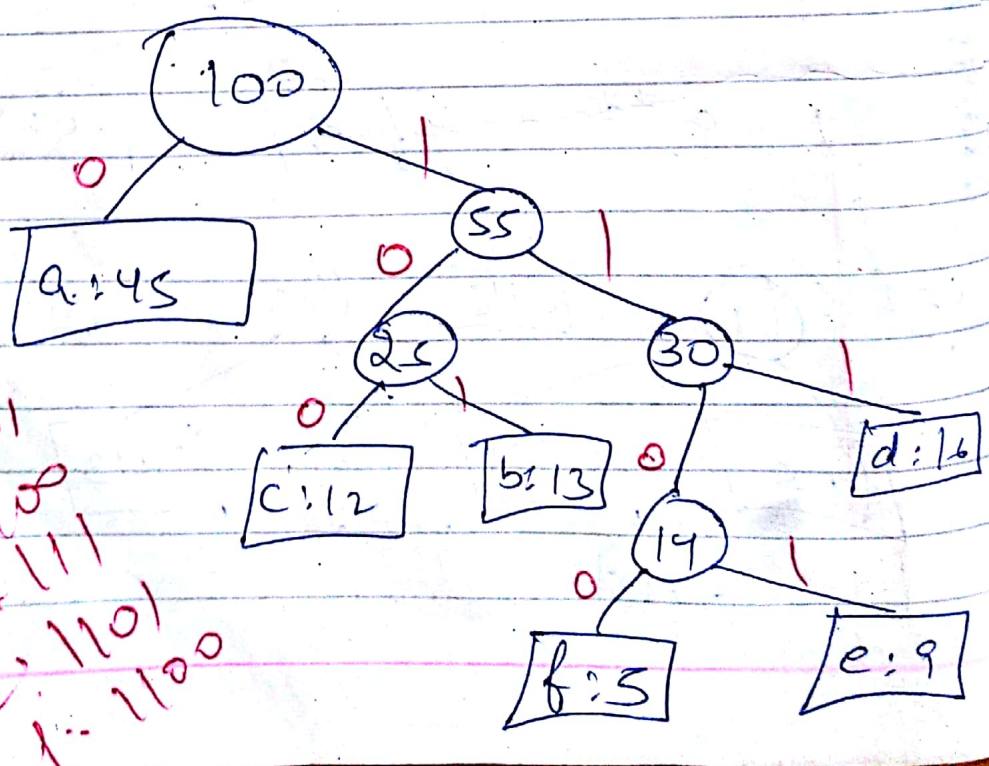
i=3



i=4



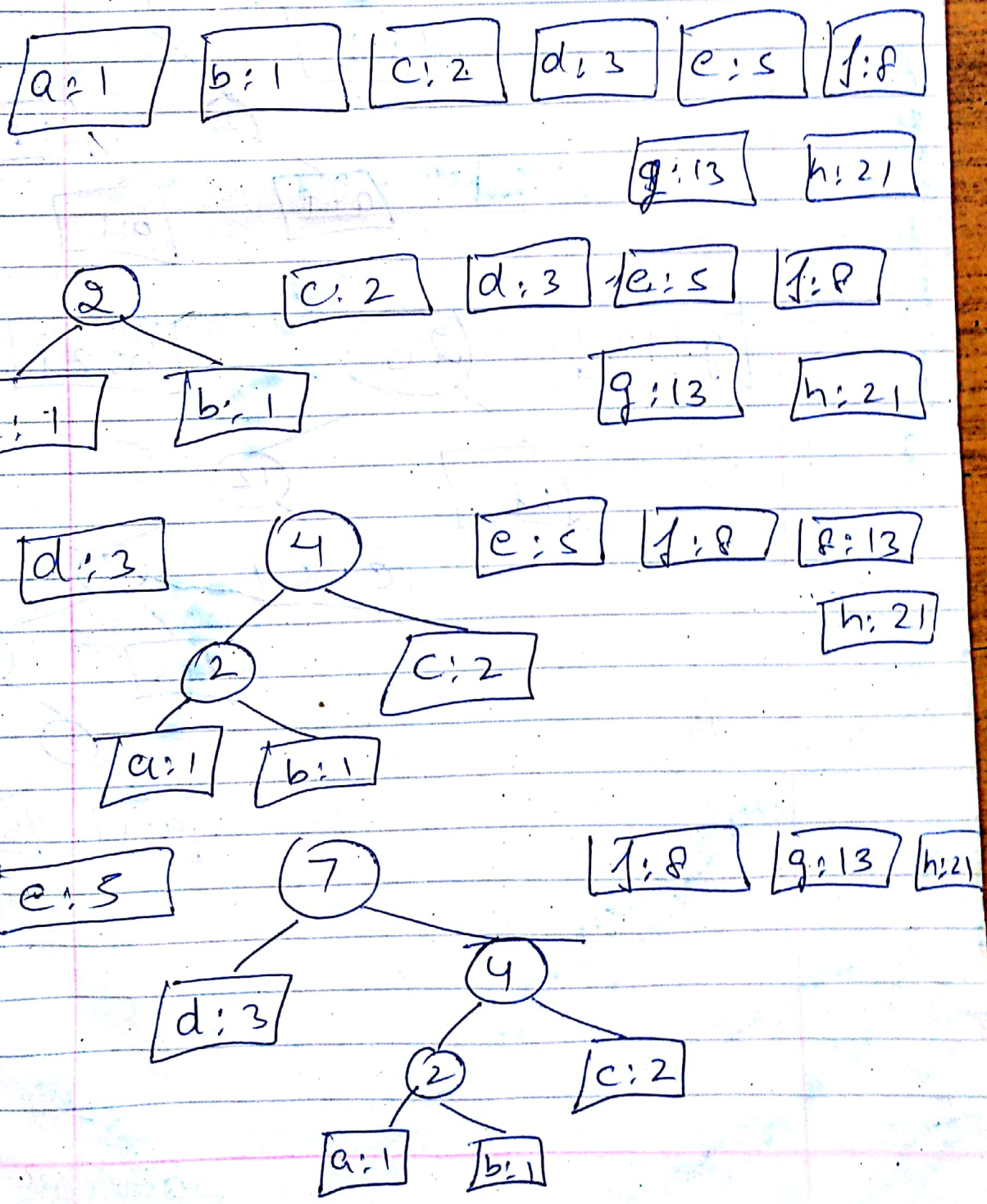
i=5

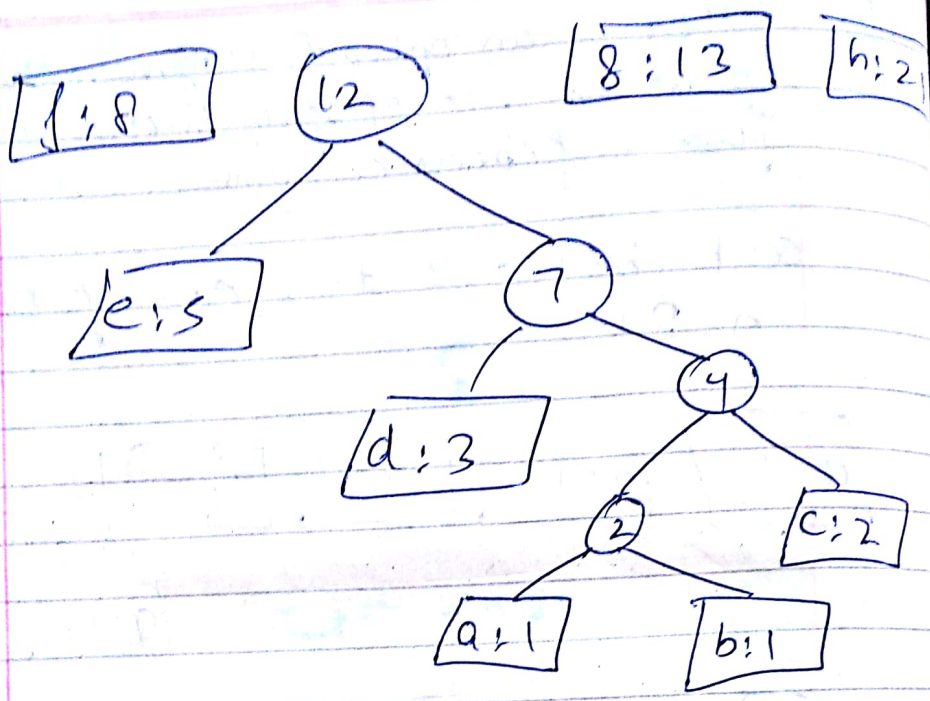


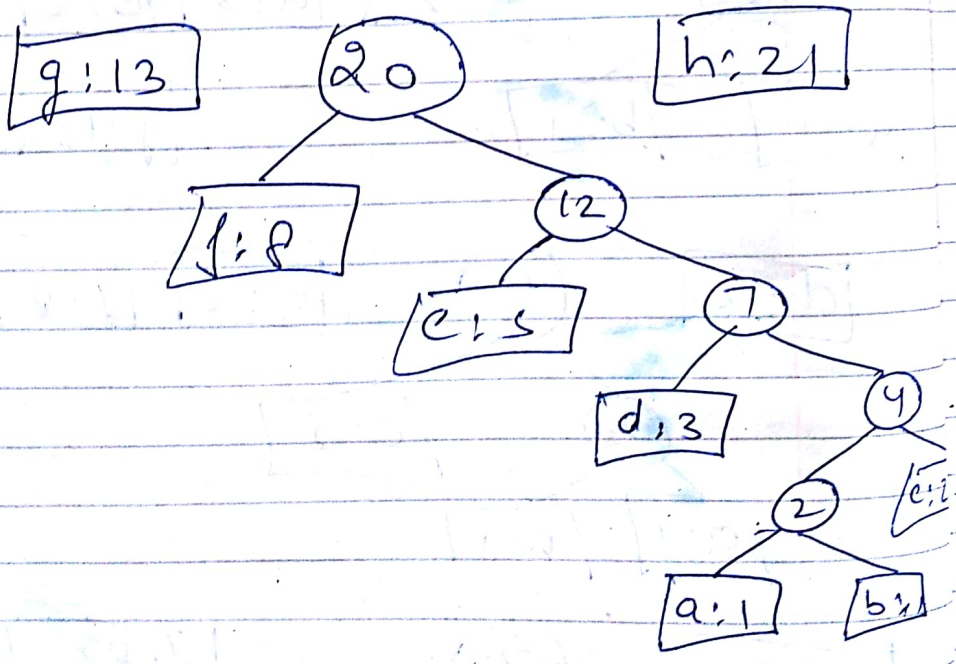
a: 0
b: 101
c: 100
d: 111
e: 1101
f: 1100

Q: What is an optimal Huffman code for following set of frequencies based on first 8 fibonacci numbers.

a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21



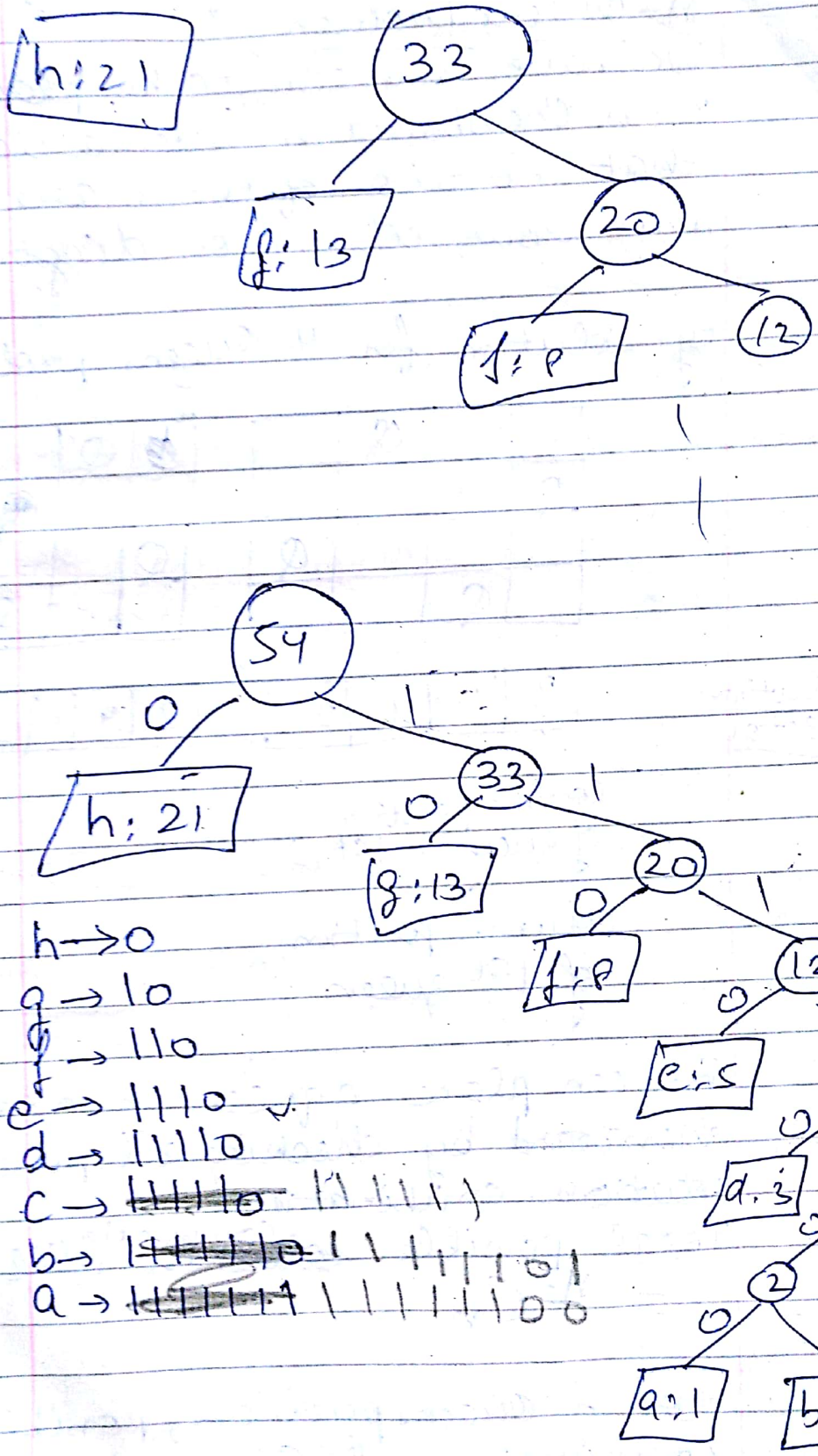




20
13
33

h
h→
g→
f→
e→
d→
c→
b→
a→

33
21
54



h → 0
a → 10
f → 110
e → 1110
d → 11110
c → 111110
b → 111111 0
a → 111111 1 0 0