Huffman Coding

→ Those characters which have high freq. should have less storage space → Motive of Huffmann coding

Eg "a" in "aaabcd", so here 'a' shouldn't take 8 Bits / 1 Byte

☆ (←) [1 sb Pg]

This will never happen →

a → 00 g → 001

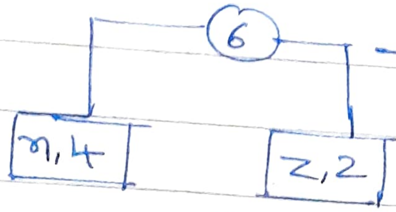
Code = $\underbrace{00}_{a}\underbrace{100}_{g\times}$ (but for ag you wanted 0001)

Not Possible since these are Prefix free Co...

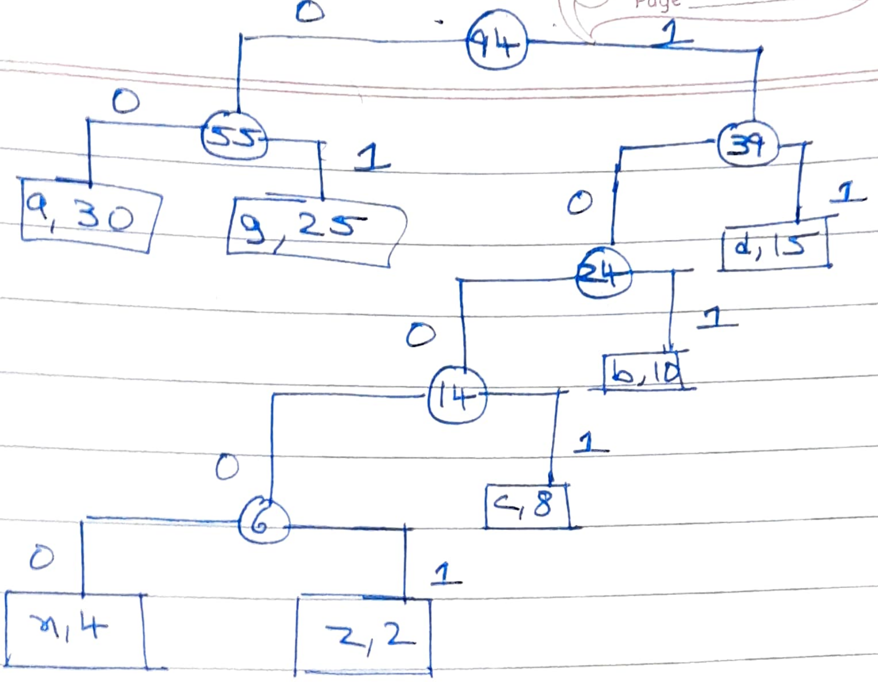
Eg

	Frequency
a	30
b	10
c	8
d	15
g	25
x	4
z	2

Pick those characters which have Min. frequency



→ Create a Node with freq. = Sum of freq. of these two and remove x, 4 and z, 2 from the frequency dictionary



This is a Tree ←

Now, to find Code for each character,

→ For every Left Node add 0

→ For every Right Node add 1 (1) →

∴ Code ⇒ a → 00

g → 01

d → 11

b → 101

c → 1001

x → 10000

z → 10001

⇒ abadg → 5 Bytes (1 Byte for each char) ⇒ m

code=
00101001101

1 Byte (8 Bits) 1 Byte] → 2 Bytes

⇓

[Space reduced compared to (1)]

So we need to stop adding 0/1 to Code once we reach a Leaf node or Root Becomes None

→ To store freq of each character, we use Hash Map.

→ At each step, You want those 2 characters which have Min. frequency → We use Min Heap

→ Tree like Struct. → We use Tree

→ To Store Code of each character → We use Hash Map (of Character, Bits)

- After Encoding ⇒

~~abca~~

~~↓~~

~~101111011100~~

~~↓~~

~~8 Bits~~

abca

↓

$\underbrace{10111011}_{8\ \text{Bits}}\underbrace{100}$

It'll convert this also into 8 Bits by adding zeroes

i.e. 10000000 → Aisa ho jayga

∴ While decoding, we don't know how many zeroes were present in the original code

Make sure length of Bits are Multiple of 8

To achieve this, You need to Pad this encoded text

How much part we have padded we will come to know through the first 8 bits

Eg $\underbrace{10111100}\boxed{101}$

Total length = 11

11 % 8 = 3 = length of □ portion

8 − (11 % 8) = 5 = No. of zeroes to be added after □

- Padding means adding zeroes at the end
- 1st 8 Bits Tell us how much we have padded

Eg If you have padded 3 zeroes, you'll store $000000\underbrace{11}_{\text{Binary form of 3}}$ in 1st 8 Bits

If — 4 zeroes, — $00000\underbrace{100}_{\text{Binary form of 4}}$ —

☆ ← (1st Pg)

Problem Solving Tips

① Try calling function first and then doing what you wanted to do

☆ Data structures req. for Huffman Coding:

① To store freq. of each char → Hash Map of chars and integers

② To Build a Tree → Tree

③ I want chars with Min. freq. → Min Heap

④ To store Code of each char → Hash Map ~~o~~ of chars and Bits

Eg a → 00

b → 101

- Decompression mean~~d~~s Decoding

Eg 00 means 'a', 101 means 'b'

Problem →

a → 00   g → 001

String = $\underbrace{00}_{a}\underbrace{100}_{x}$ OR $\underbrace{001}_{g}\underbrace{00}_{x}$ This will never happen (Codes are Prefix free) i.e. if a is 00 no ~~at~~ code of any other char (say g will start from 00) d)

Padding → $\underbrace{1\,0\,1\,1\,1\,1\,0\,0}_{8\text{ Bits}}\;\underbrace{1\,0\,1}_{\text{Add 5 '0s' to make 8 Bits}}$

⇒ $\underbrace{1\,0\,1\,1\,1\,1\,0\,0}_{8\text{ Bits}}\;\underbrace{1\,0\,1\,0\,0\,0\,0\,0}_{8\text{ Bits}}$

~~For me~~ I should know that I have added 5 '0s' to pad the text, so this padded info, I'll store in 1st 8 Bits. Here, I have padded 5 zeroes, Binary form of 5=101 I want to store it as a Binary No. consisting of 8 Bits

∴ $\underbrace{0\,0\,0\,0\,0\,1\,0\,1}_{\text{Padded\_info (Binary form of 5)}}\;\underbrace{1\,0\,1\,1\,1\,1\,0\,0}_{8\text{ Bits}}\;\underbrace{1\,0\,1\,0\,0\,0\,0\,0}_{8\text{ Bits}}$

To Remove Padding,

if original text = [10011100][10001]

⇒ Padded text = [10011100] [10001$\underbrace{000}_{\text{Padding (e=3)}}$]

∴ org.text = Padded text [: -1*e]