oms sairam   omsrivenkatesayanamaha   omsriganadhipatayanamaha   om namaha shivayaha

Greedy Algorithm - Continuation:-

1) Huffman Code :-

* To compress the information and use less memory, we encode characters.

* There are different types of encoding patterns (or) standards

Ex:- ASCII

* In ASCII (American Standard for Information Interchange), every character is given a hexadecimal - number.

Ex:- a - 0x40   a - z , A - Z

b - 0x41 } 8 bits

So, if we have 10,000 characters, we use 80000 bits.

* To compress the data, We can use "small length" codes for most repeating chars and "large length" codes for less frequent chars.

So that we can use less memory

→ For example, a, e → repeats more in english characters, so, assign bits of small length to a, e, and for less repeating characters assign bigger length bits.

e := 1

a := 01

z := 00000001111111

But, Decoding = ?

→ Coding & Decoding made easy in Prefix Codes:

Prefix Codes:-

Ex:- a, b

Code(a) cannot be a proper prefix of codeword(b)

(or) any character

(or) any character

Proper-Prefix = ?

b_1 : 0001

b_2 : 000

"b_2" is proper prefix of "b_1"

→ ASCII codes are prefix codes

Ex:-

x ? y ?

x : $\boxed{0000}$01

y : 0000

→ We can arrange code-words in the form of a tree in "Pre-fix" Codes

(a, b, c, d, e)

Ex:-

a : 01

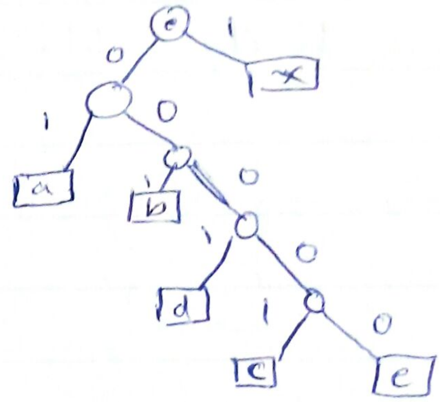
b : 001

c : 00001

d : 0001

e : 00000

decision tree



Ex:-

$\underbrace{00001}_{c}\ \underbrace{01}_{a}\ \underbrace{01}_{a}\ \underbrace{1}_{x}\ \underbrace{1}_{x}\ \underbrace{01}_{a}\ \underbrace{1}_{x}\ \underbrace{01}_{a}\ \underbrace{1}_{x}\ \underbrace{01}_{a}\ \underbrace{1}_{x}\ \underbrace{01}_{a}$

→ ASCII is a 8 bit prefix code

↓ can be stored in "Hash Table"

Optimal Pre-fix Code:-

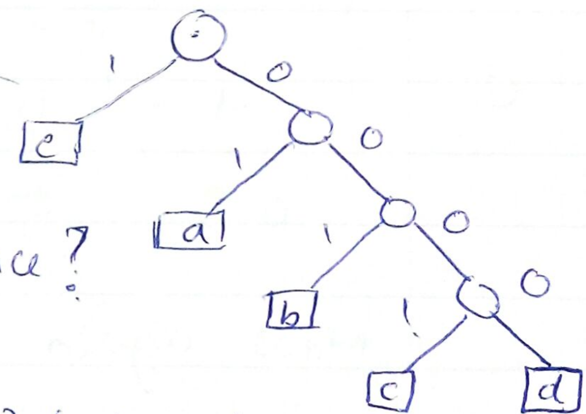
If we have document containing, 10^6 chars containing

a, b, c, d, e
0.3, 0.2, 0.1, 0.05, 0.35 → fractions of occurence

a : 000
b : 001
c : 010
d : 011
e : 100

} If we use '3' bits, we need 3×10^6 bits of space.

↳ considering above occurences,



using this code

10^6 bits document size requires how much space?

$= 0.35 * + 0.3 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.05 \times 4$

$= 2.15$ bits per character.

H.W: Design probabilities of chars, that gives even worse compression

Huffman Code :- (Going Bottom Up)

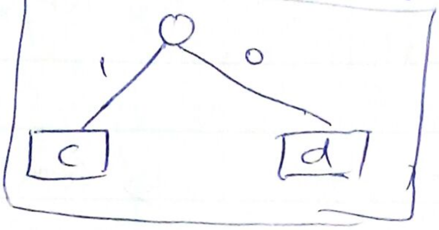
↳ Optimal

a	b	c	d	e
0.3	0.2	0.1	0.05	0.35

① Take the two least frequent chars & make a sub-tree ;

(combine two chars & make a composite character & assign frequency by adding freq. of two chars)

Tree -



→ composite char

"cd"

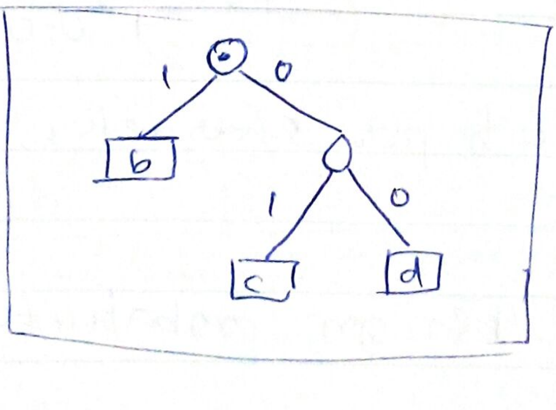
⇓

0.1 + 0.05 = 0.15

②

a	b	"cd"	e
0.3	0.2	0.15	0.35

Now again you take, two least frequently occuring char



"bcd"

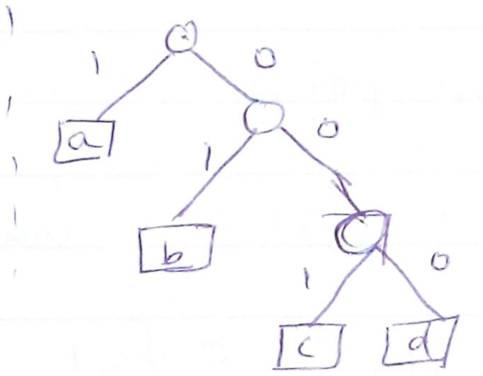
⇓

0.2 + 0.15

= 0.35

(3) a bcd e

0.3 0.35 0.35



"abcd"

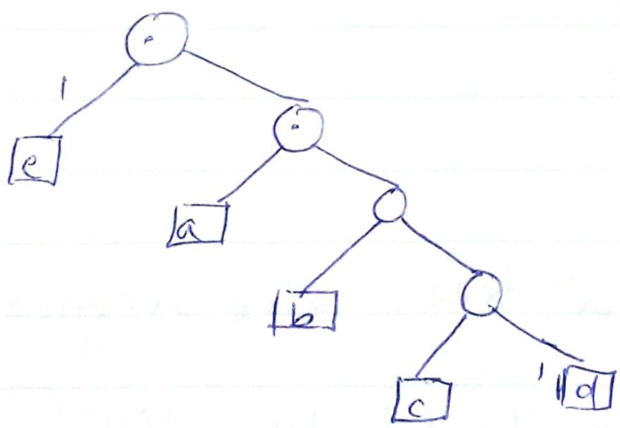
⇓

0.3 + 0.35

= 0.65

4) abcd e

0.65 0.35



⟶

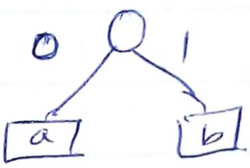
If you go from "Top to Down" (Shannon's Algo). It'll be Sub-optimal.

Why Huffman-Code Optimal?

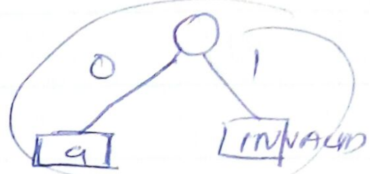
Proof of Optimality:-

Three main points

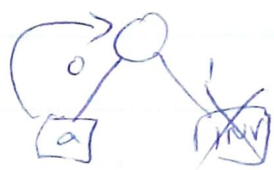
1) Deepest node always has a sibling. It should be like



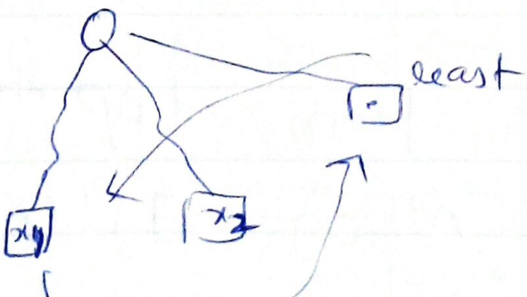
but cannot be



→ Huffman-code will not give invalid nodes



② In a tree, the deepest nodes, x_1 & x_2 are (or) should be least frequent characters

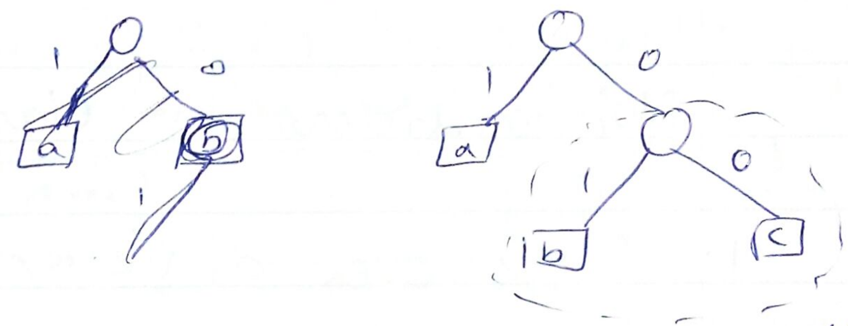


(or) If that is not the case, then exchange with the least frequent characters, node, which is up in the tree.

③ Optimal- Substructure

Transform the input-problem, ~~to~~ by taking, ~~to~~ '2' least frequently occurring characters & ~~mere~~ combining to 'one' character by adding probabilities (or) frequencies.,

then, the remaining tree must be optimal for the other problem



Sub-problem