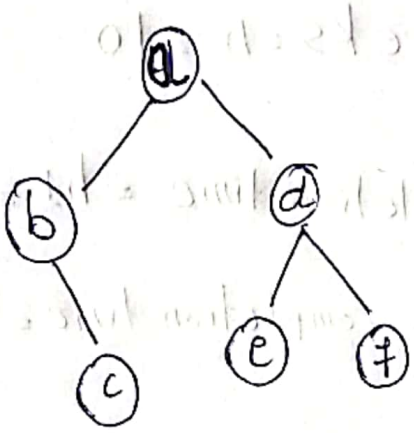
Name Fazeel khalid

Roll # L 19-1021

Section CS-4D

Question 1

Greedy algorithm is not a good approach it give us a rather poor approximation, because The inherent greedy algorithm for the vertex is to repeatedly choose the highest scoring vertex. When we put the vertex in the cover, we will remove its all adjacent node. But it's not work in this case for example



in this example node **d** will selected and after this node **c, e, f** will be selected only then we can meet the conditions but in this way total # of vertex are '4' that are a, c, e, f

But vertex {b, d} will work fine so no. of vertex is 2 so it's an optimal solution.

Question 2

of weeks = n

# of task = n = $t_1, t_2, t_3 ---- t_n$

Each task completion time h_i hours = i

So we will use greedy algorithm for this job. for maximize earning.

Let assume

# of weeks = n = 10

t_1 tasks completion time = h1 = 5

t_2 another tasks completion time = h2 = 15

So

$h_i * (n-j)\$$

$h_1 * (n-j)\$$

$5 \times (10-0)\$$

50 $

$h_2 \times (n-j)$

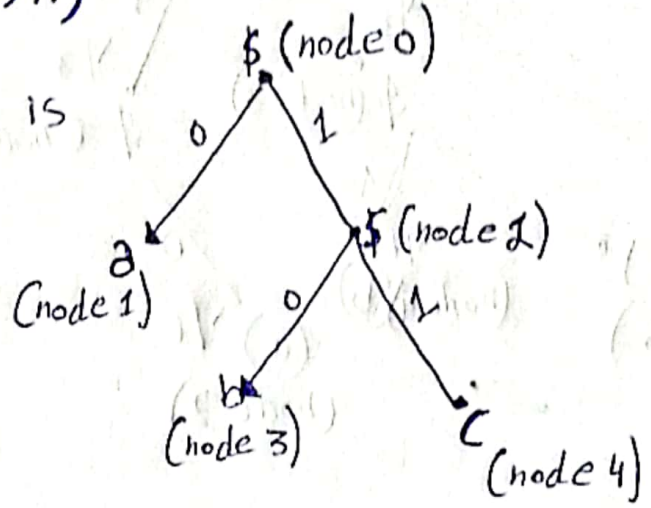
$15 \times (10-0)$

150 $

Question:3

a) Code (0, 10, 11)

Huffman tree is



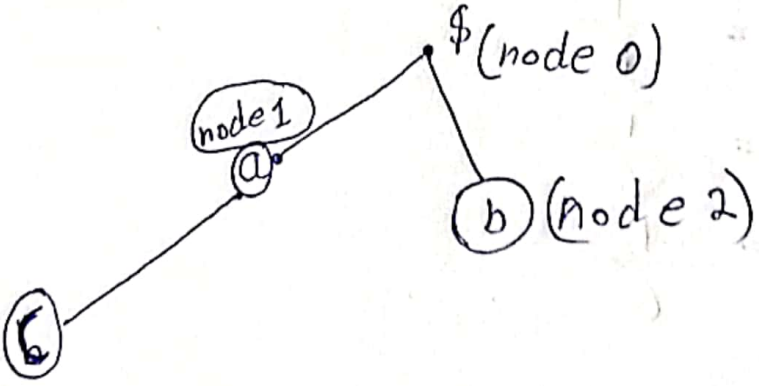
The frequency of node 1 is = $\frac{2}{3}$ = f_a

frequency of node 2 = $\frac{1}{3}$

frequency of node 3 = $\frac{1}{6}$ = f_b

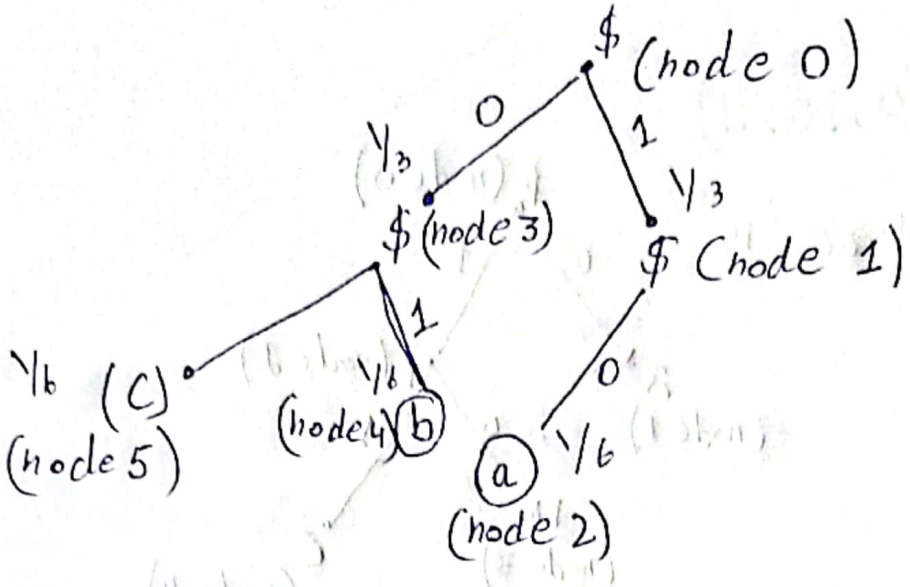
frequency of node 4 = $\frac{1}{6}$ = f_c

b) Code (0, 1, 00)



This huffman encoding is not possible because code of **a** is a prefix of **c**

(c) Code (10, 01, 00)



This is not an optimal solution even not a valid huffman encoding because.

for 2bit numbers there should be 4 leaf node but here we have only 3 nodes

If we talk about frequency

$$f_a = \frac{1}{6}$$

$$f_b = \frac{1}{6}$$

$$f_c = \frac{1}{6}$$

sum of all these frequencies are = $f_a + f_b + f_c$

$$= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{3}{6} = \boxed{\frac{1}{2}}$$

(3)

As total frequency is not equal to 1 so it also conclude that given code is not a Huffman encoding.

Question 4

Steps for Ternary Huffman encoding.

- Count the frequency of all latters & symbol and store it is ~~array~~ list
- Sort that list in assending order
- Remove all dublicates symbols and latters.
- Pick three frequencies from start of the list
- do the last step until only one element remain in the list.
- mark left node as zero
- mark right node as two
- mark middle node as one.
- As ternary tree use odd # of nodes for example zero ore three if we dont have odd # of nodes for this purpose we should need to add a node with frequency zero. By adding this if the # of nodes remain even then subtract two nodes from the tree otherwise carry on / move forward.

c) If a Ternary ~~tree~~ does not consist of three child at any node ~~level~~ and 3^n node at any level then that parent node will be the second last node and next of it will be leaf node, these child node can be deleted to get a full ternery tree. All this will start from the root node and then we will move towawrd downward ~~This alg~~

Conclusion =>

This algorithm proves that this approach should be a right one because ternary tree time complexity is $O(n \log_3 n)$