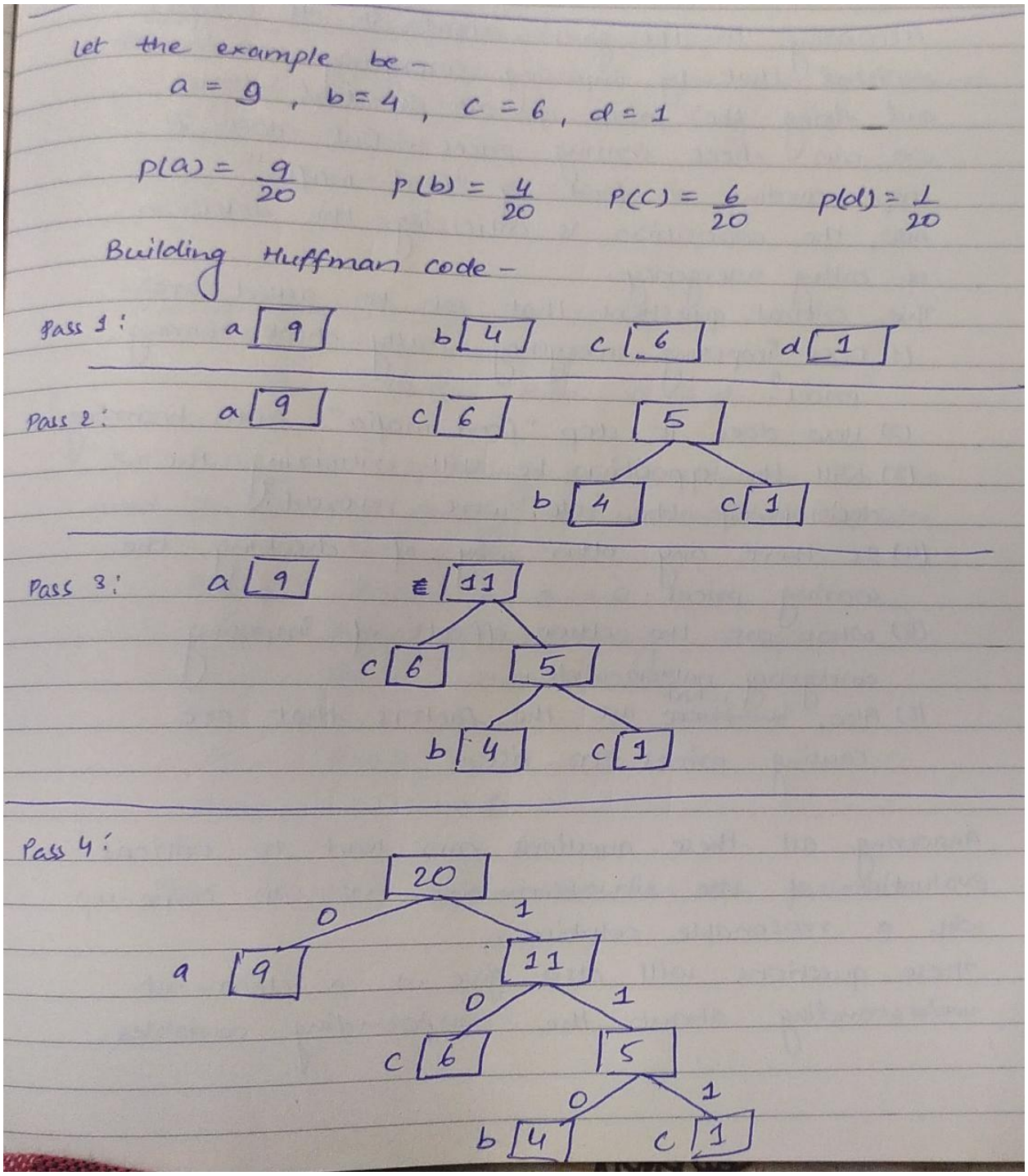
Tutorial 7 Solutions

1. (Sachin) In an Huffman code instance, show that if there is a character with frequency greater than ⅖ then there is a codeword of length 1. Show that if all frequencies are less than ⅓ then there is no codeword of length 1.

When one character is having frequency greater than ⅖



Huffman codes are –

a = 0
b = 110
c = 10
d = 111

} Here a has a codeword of length 1.

When all the character are having frequency less than ⅓

Consider the Huffman coding instance –

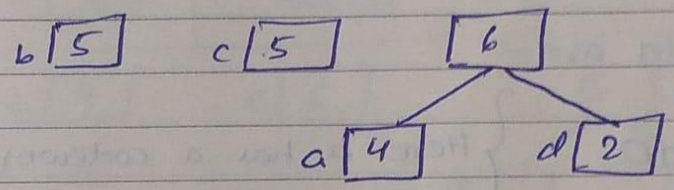
a = 4, b = 5, c = 5, d = 2

$p(a) = \frac{4}{16}$, $p(b) = \frac{5}{16}$ $p(c) = \frac{5}{16}$ $p(d) = \frac{2}{16}$

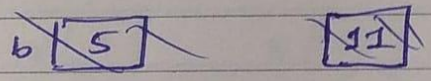
Huffman Tree –

Pass 1: a[4] b[5] c[5] d[2]

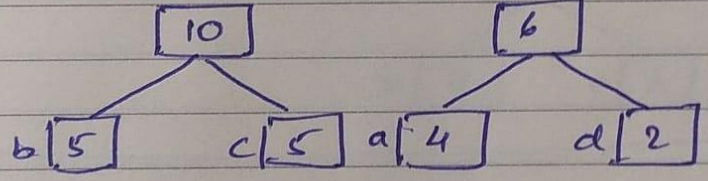
Pass 2:

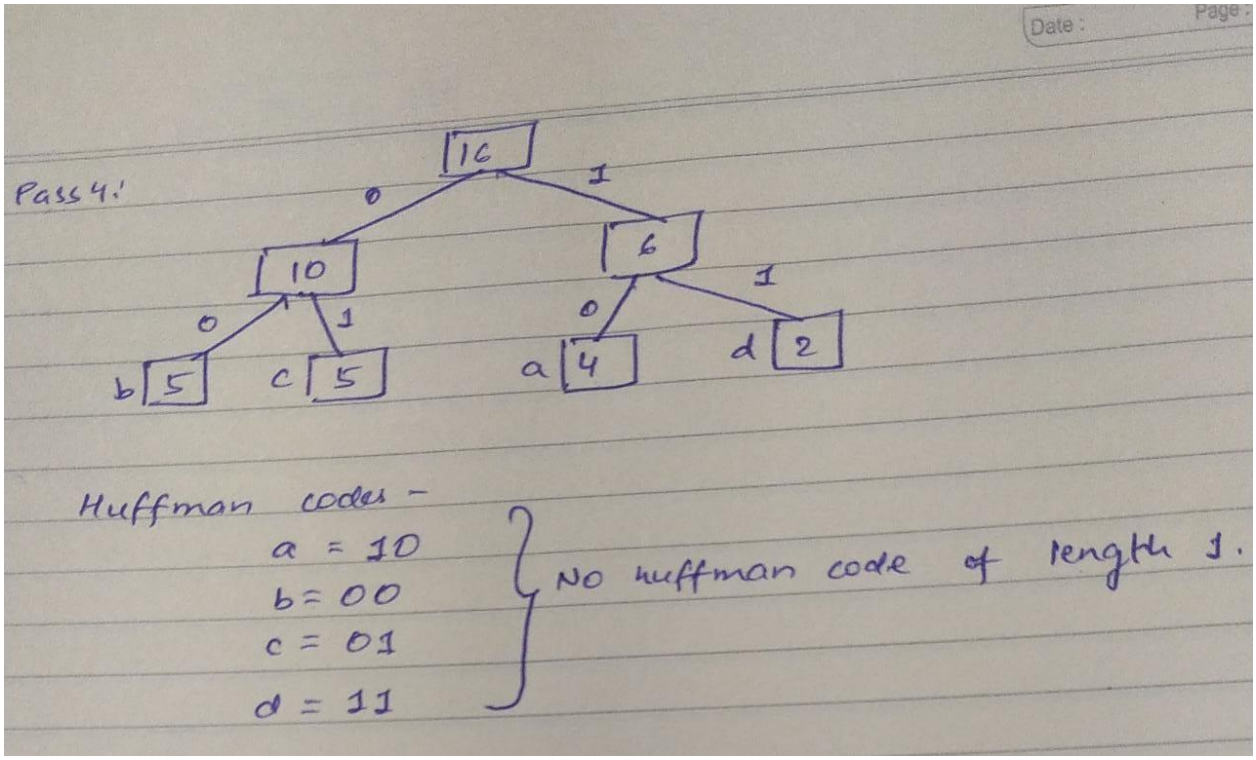


Pass 3:



Pass 3:





General Answer Part 1: Suppose that x is the letter with weight p >0.4. If the code for x is not a single letter, then during the execution of the algorithm at some point it was merged with some group of letters, say Y. Since the last two groups get merged, at the point, the situation could be A,,Y,x or A,x,Y, or A,B,...,Y,x etcetc (in decreasing order of frequency). Now since p(x)=0.4, there can't be two groups ahead of it. So, the situation must be A,x,Y. This means that p(A)>0.4 and thus p(Y)<0.2. Now A must have come as a join of C1 and C2. Since p(A)>0.4, one of them, say C1, is such that p(C1)>0.2. In which case, why was Y not picked in the merger of A? Contradiction.

The second part is similar and easier.

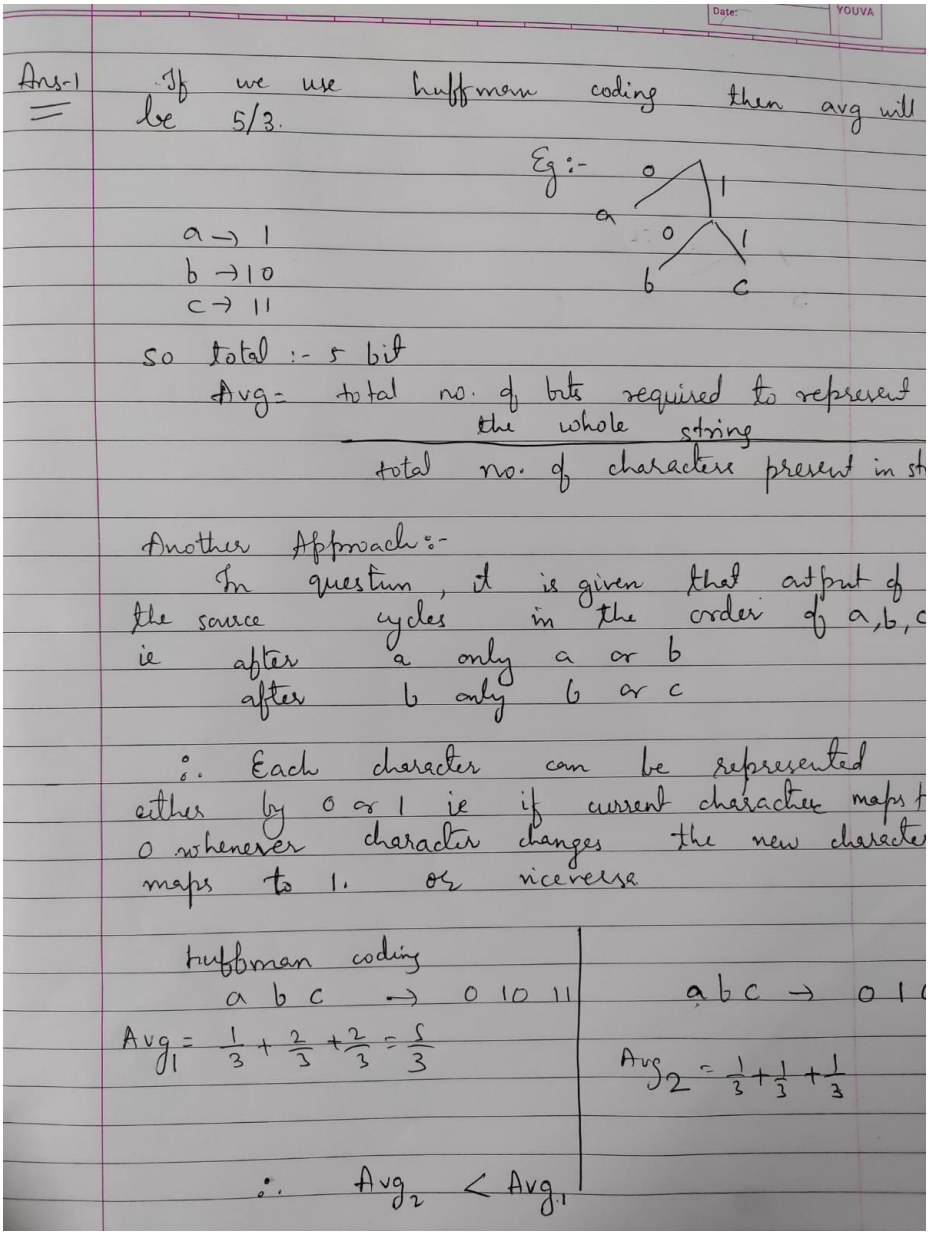
General Answer Part 2: Suppose that all the characters are having weight less than 0.33 and there is at least one character, say x having huffman code of length 1. So, at the final step, we must have (A,x) or (x,A). The second is not possible since that would put p(A)<p(x)<⅓ and p(A)+p(x)=1. So it must be (A,x) and p(A)>2/3. Now, in the previous step, A=C1+C2, where one, say, p(C1)>⅓. In which case, the merge should have been C2 and x! Contradiction.

---

2. (Pooja) Suppose that there is a source which has three characters {a,b,c}. The output of the source cycles in the order of a,b,c. In other words, if the last output was a b, then the next output will either be a b or a c. Each letter is equally probable. Is the Huffman code

the best possible encoding? Are there any other possibilities? What would be the pros and cons of this?

Ans



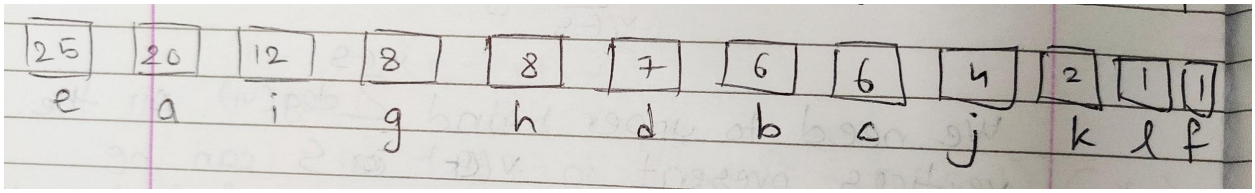
Constraint: We need to give information about the first character whether it is a or b or c. This can be generalized. For every current letter x, we have a Huffman coding tree Tx for the next letter. Thus, if we have transmitted y before, we transmit using Ty. If the next letter is x, the code for x is chosen using Ty. For the next letter, the tree Ty is used. And so on. The key input is a table: P(x|y): probability that the next letter is x, given that the previous letter was y.

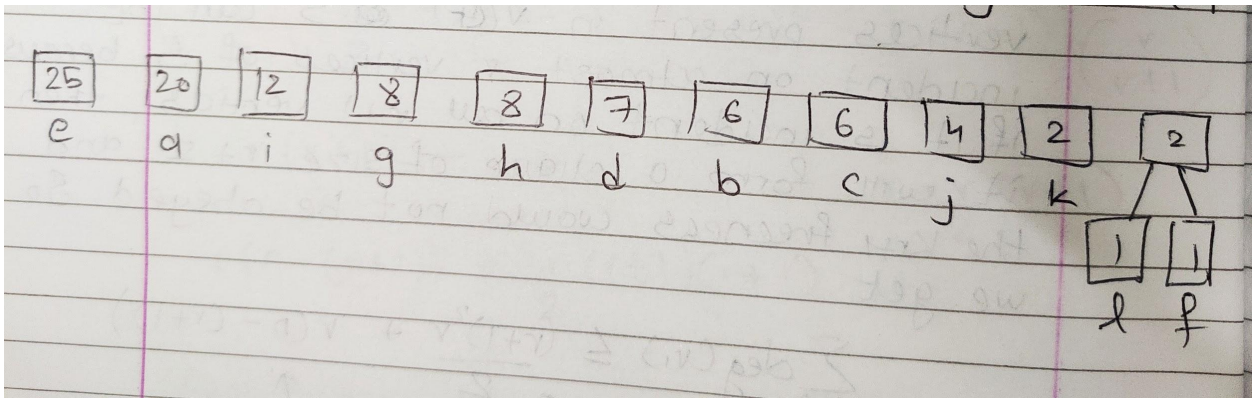
Next Prev	a	b	c	d
a		0.1	0.5	0.4
b	0.5	0.3		
c	0.4	0.6	0.5	0.4
d	0.1			0.2

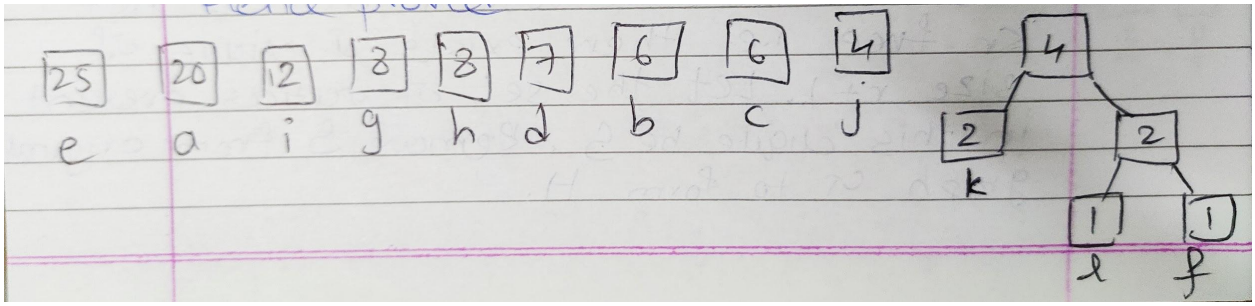
This needs several trees as a part of the code. Moreover, if we make a mistake due to transmission, then the error will continue. So, we need to periodically check.

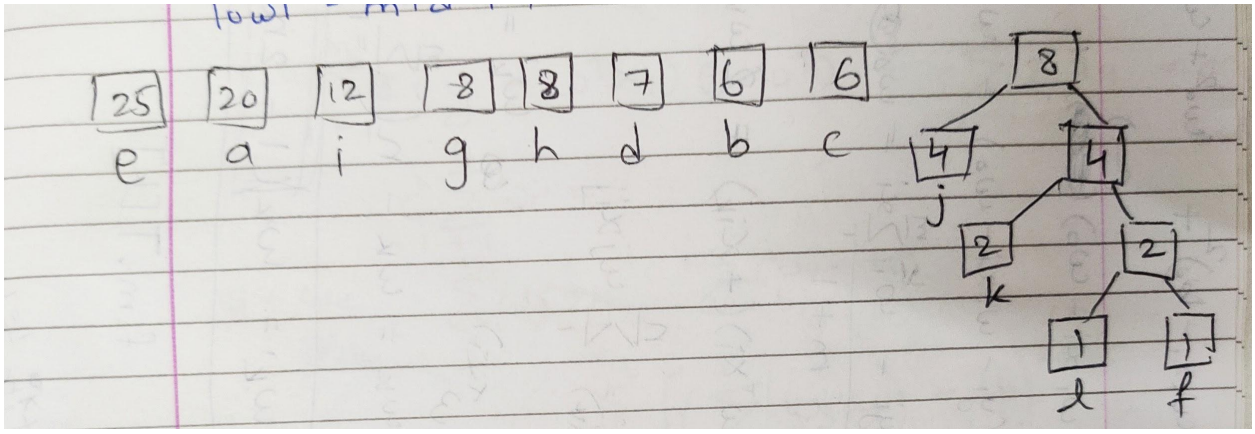
3. (Jalay) Consider the following table of letters and frequency. Design a Huffman code tree.

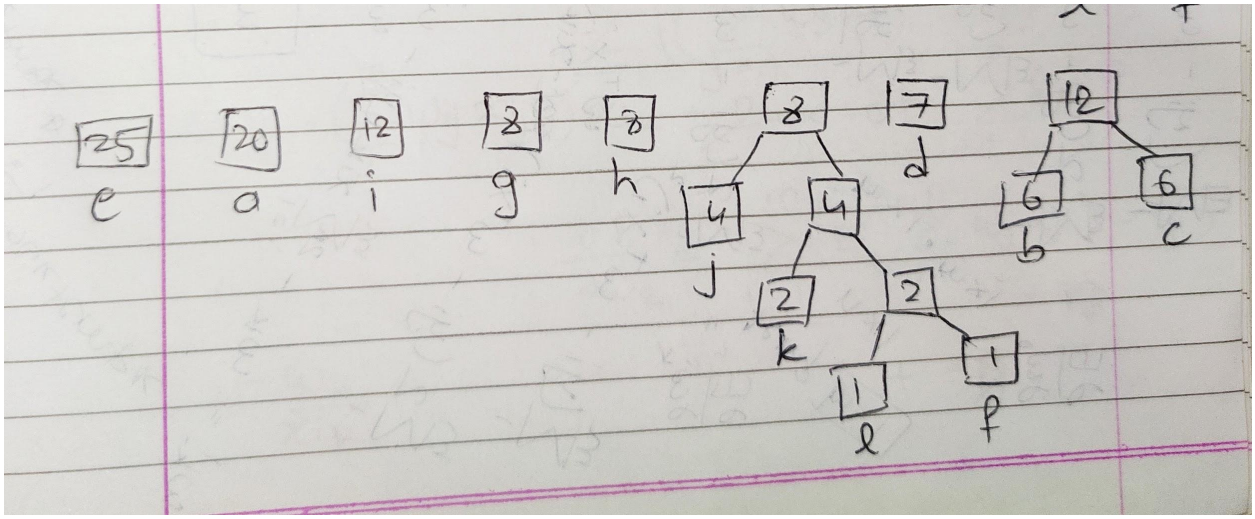
a	20	d	7	g	8	j	4
b	6	e	25	h	8	k	2
c	6	f	1	i	12	l	1

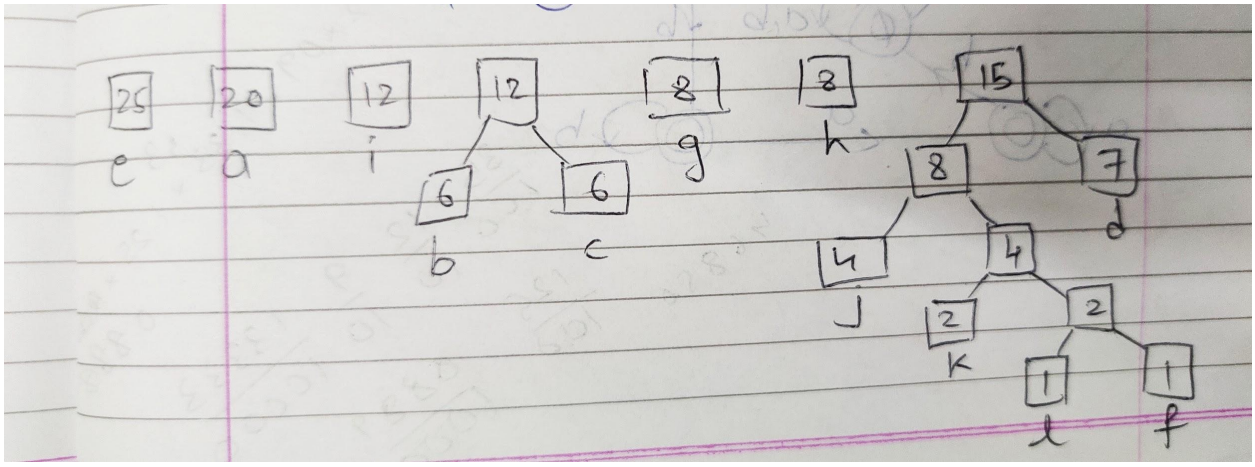


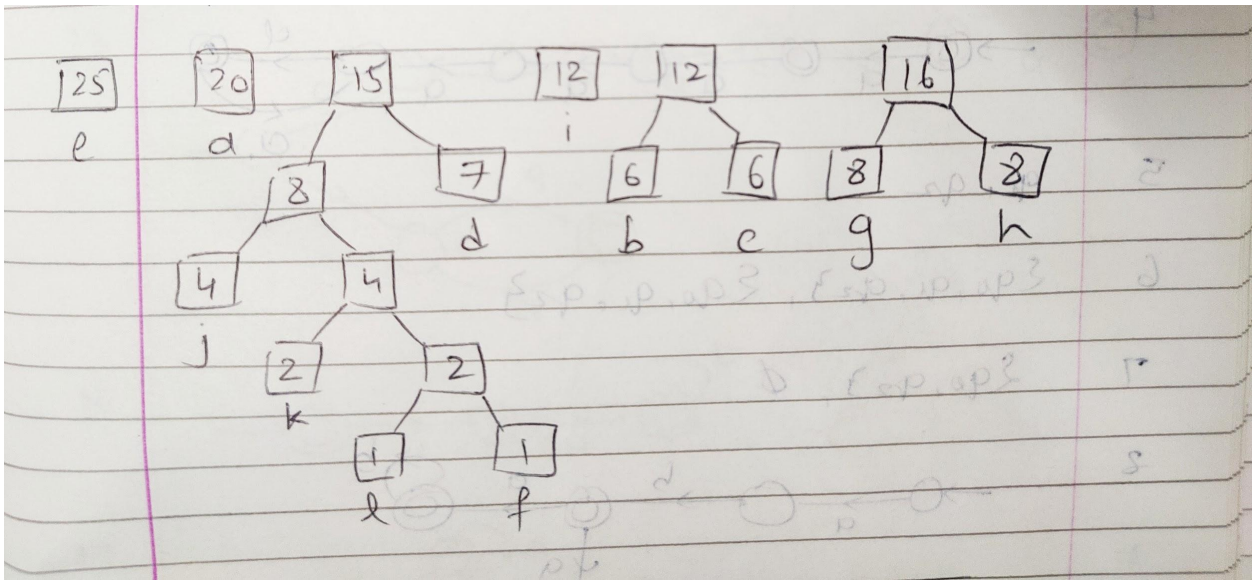


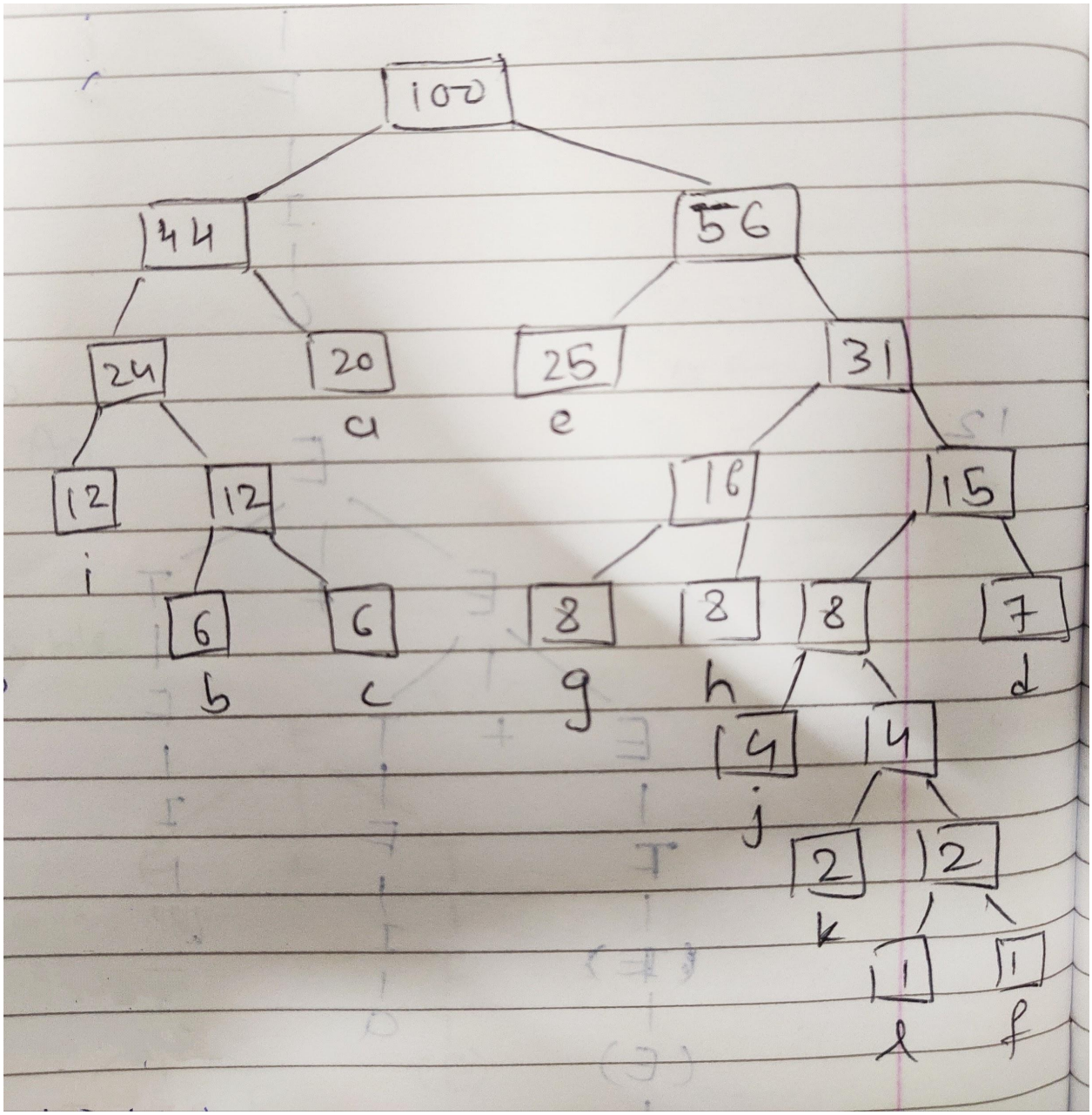

25 20 12 8 8 7 6 6 4
e a i g h d b c j
4
2 2
k
1 1
l f


25 20 12 8 8 7 6 6
e a i g h d b c
8
4 4
j
2 2
k
1 1
l f


25 20 12 8 8 8 7 12
e a i g h d
4 4 6 6
j b c
2 2
k
1 1
l f


25
e
20
a
12
i
12
6
b
6
c
8
g
8
h
15
8
7
d
4
j
4
2
k
2
1
l
1
f


25
e
20
a
15
8
7
d
4
j
4
2
k
2
1
l
1
f
12
i
12
6
b
6
c
16
8
g
8
h



4. (Debajyoti) Can a Priority Queue be implemented as an AVL tree? What advantages does a Heap implementation have over an AVL tree implementation?

Ans:-A typical Priority Queue requires following operations to be efficient:-

1> Get max or min element

2> Delete max, min

3> Insert element

4> Increase or decrease key

The time complexities for doing the above operations by binary heap is as follows:-

1>O(1)

2>O(logn)

3>O(logn)

4>O(logn)

The time complexities for doing the above operations by avl tree is as follows:-

1>O(1) by maintaining an extra pointer for maximum or minimum element

2>O(logn)

3>O(logn)

4>O(logn)

So a priority queue can be implemented as avl tree but by implementing as binary heap we get the following advantages:-

1>Since Binary Heap is implemented using arrays, there is always better locality of reference and operations are more cache friendly.

2>We can build a Binary Heap in O(n) time. Avl tree require O(nLogn) time to construct.

3>Binary Heap doesn't require extra space for pointers.

4>Although operations are of same time complexity, constants in Avl tree are higher.

5>There are variations of Binary Heap like Fibonacci Heap that can support insert and decrease-key in Θ(1) time

5. (Ankit Mtech 1) The Heap implementation needs us to find the “last’ element in the heap. Write a code snippet to maintain the last element. Suppose we maintain a pointer to the last element, write a code snippet top go to the previous one. This will be useful if there are a sequence of deletes.What is the worst and the average time complexity of locating the previous? What happens if we do not maintain the last element? How do we locate the last element?

   The prev and the next elements are very important to maintain the heap. Let us assume that we have the parent pointer to every node v. The basic logic is as follows:

   1. function [x,jump]=prev(v) // jump to tell whether level has changed
   2. w=parent(v); if w==null x=null; jump=0; return; // root
   3. If v==w.rightchild x=w.leftchild; jump=0; return; // easy case
   4. If w.parent==null x=w; jump=1; return; // this is when the level changes
   5. [x,jump]=prev(w).rightchild; return; // see how this takes care of level change
   6. endfunction;

   Let us make a a table of the recursive calls:

node	call, return	return		node	call, return	return
1	-	null,0		2	-	1,1
3	-	2,0		4	2,1,1	3,1
5	-	4,0		6	(3,2,0)	5

6. (Shivam) Suppose we have a 2D array where we maintain the following conditions: for every (i,j), we have A(i,j)<=A(i+1,j) and A(i,j)<=A(i,j+1). Can this be used to implement a priority queue?

i j	1	2	3	4	5	6
1	4	5	5	5	7	
2	4	5	7	8		
3	4	6	7			
4	4	7	Fill			
5	7	9 Last				

6	8					

[NewFill,NewLast]l=Bubble(Fill,X) // adds the value X in Fill and bubbles its up and
// Computes new fill and last
A(Fill(1),Fill(2)=X;

computes NewFill.
ii=Fill(1), jj=Fill(2).
// computing NewFill

If ii>1
  NewFill(1)=ii-1; NewFill(2)=jj+1; NewLast=Fill; // good case
Else
  NewFill(1)=ii+jj+1; NewFill(2)=1; NewLast=Fill;
EndIf

//Bubbling

donei=0; if (ii==1) OR (A(ii-1,jj)<=A(ii,jj)) donei=1; // looking UP
donej=0 if (jj==1) OR (A(ii,jj-1)<=A(ii,jj)) donej=1; // looking LEFT
While donei*donej !=1 do
 Locate max, interchange, reassign ii,jj, recompute donei, donej
Endwhile;
Endfunction;

[[X,NewLast,NewFill]=Vacate(Last): X=A(1,1) is vacated, the element A(Last) is inserted in the place vacated and bubbled up.

7. (Vinayak) Here is an interesting problem from Prof. Abhiram Ranade. A piecewise linear function on [0,1] may be represented by a sequence of special x and y values such as the table below:

x	0	0.3	0.5	1
f(x)	1.3	1.2	4.1	0.3

Thus, a good representation is f.NumberOfSegments, f.x (array of x values), f.y (array of y values). Now given two functions f and g, let h be the minimum of f and g. Clearly, it is also a piecewise linear function. Compute the representation of h.