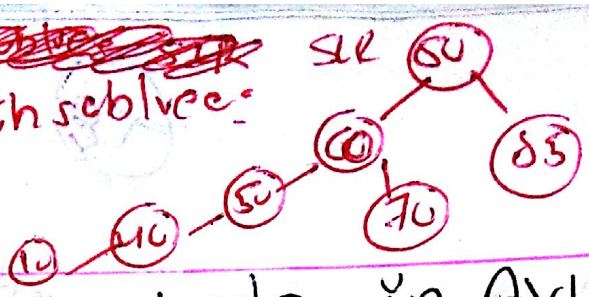
X leftchild, Rightsublree:

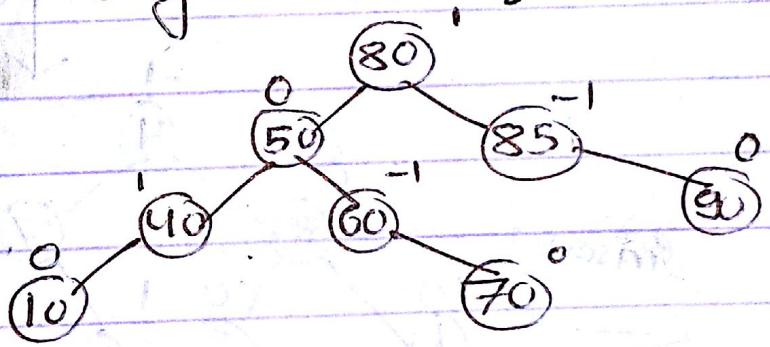


Deleting a Node in AVL tree

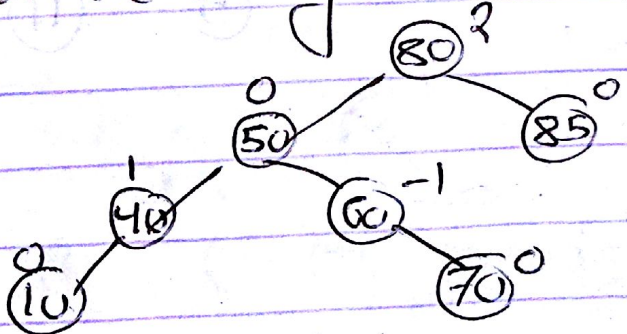
① Delete as in BST

② Check balance of node
if imbalance is detected then balance through rotation.

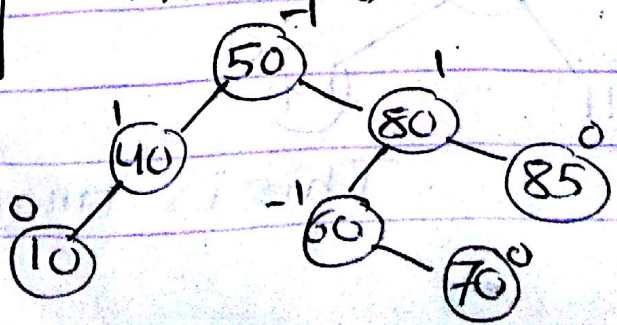
eg 1: Delete 90 from AVL search tree given below.

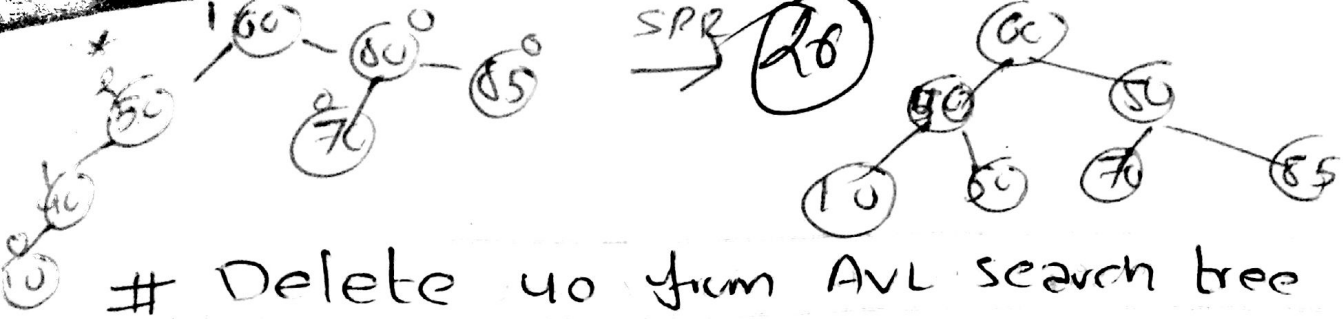


Soln: On deleting 90, we get

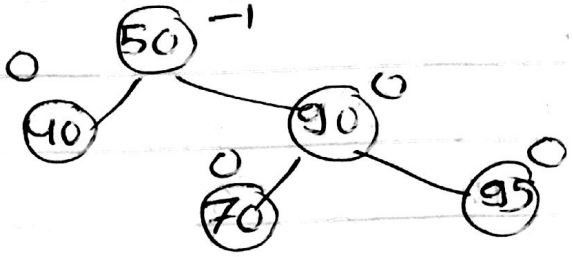


Imbalance occur at 80 because of left subtree of its left child. So Applying SRR we get

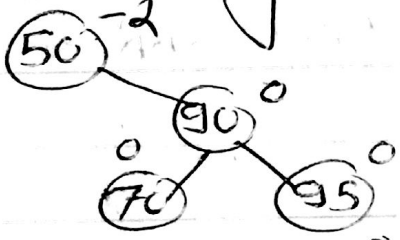




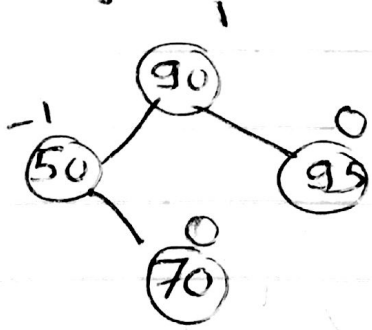
Delete 40 from AVL search tree



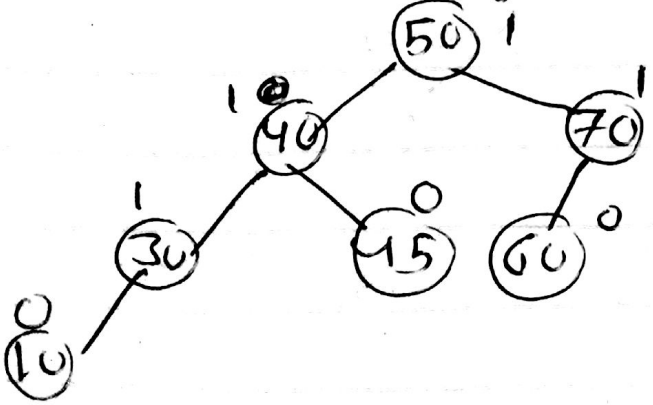
On deleting 40, we get



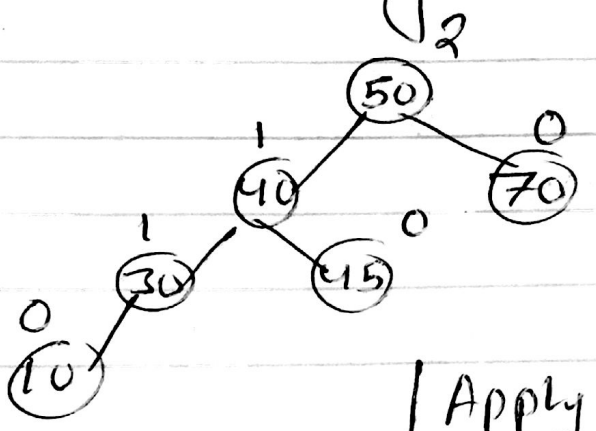
Imbalance occur at 50 because of right subtree of its right child. So applying SLR we get.



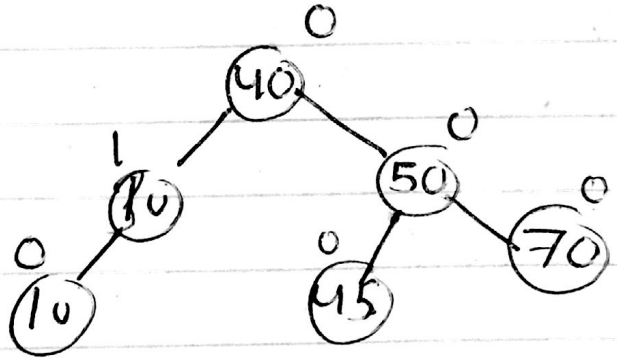
Delete 60 from AVL search tree



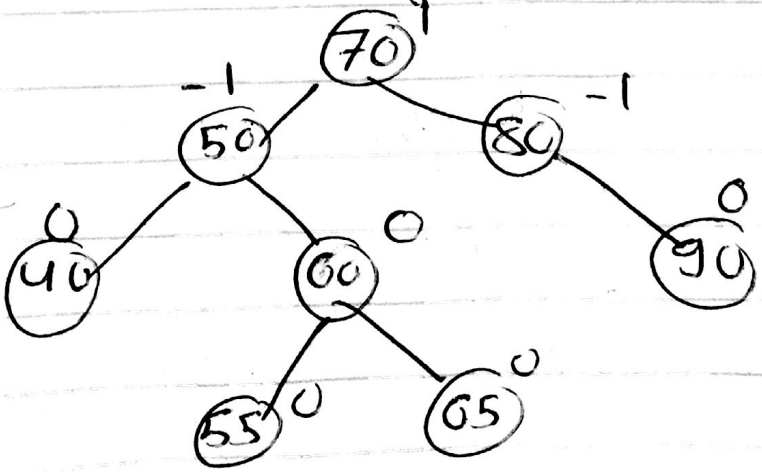
On deleting 60 we get,



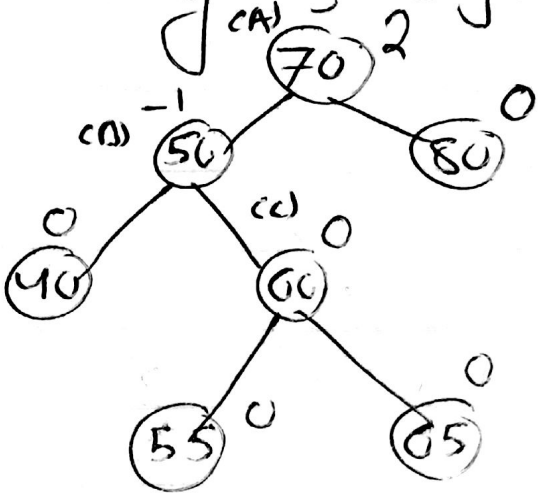
Apply SRR



Delete 90 from AVL search tree.



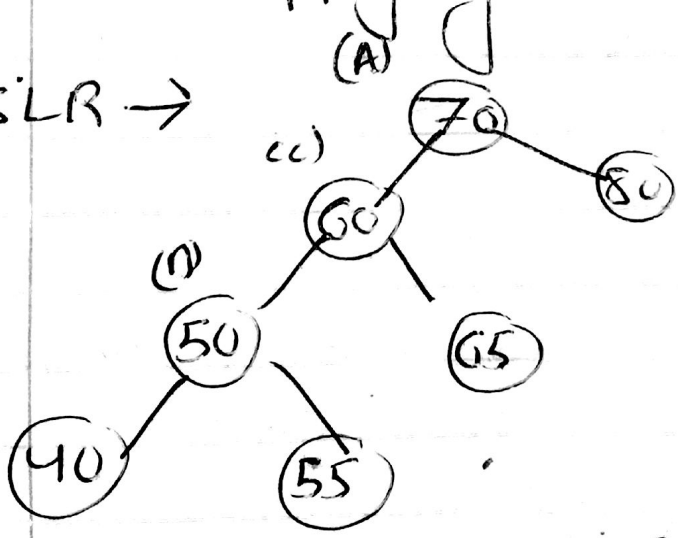
On deleting 90, we get



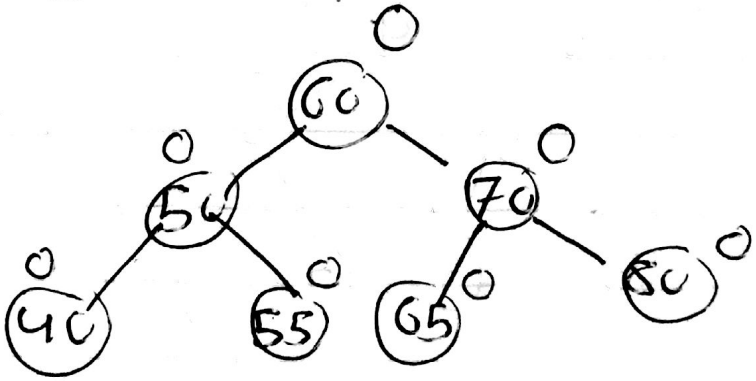
Imbalance occur at 70 because of right subtree of left child.

So applying DLR we get

SLR →



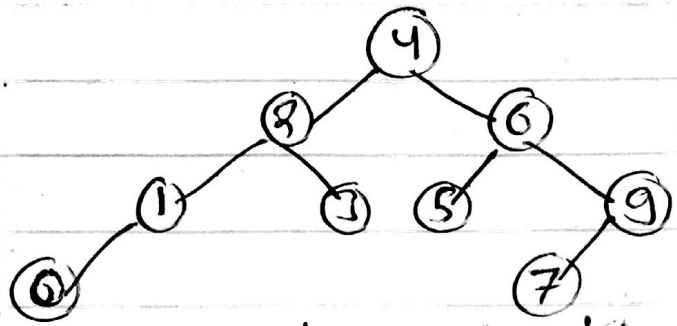
SRR →



Note:- In all above example, we performed the deletion of leaf node only.

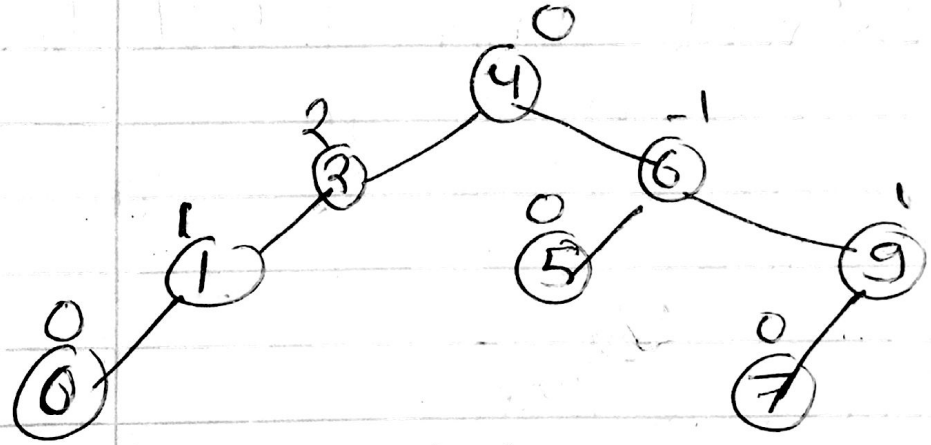
imp

Consider the following AVL tree.



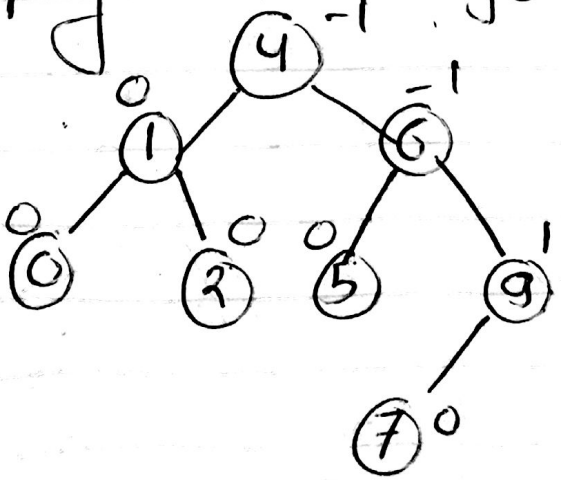
Perform deletion of 3, 5 and 4 in order.

Soln: Deletion of 3 (i.e leaf Node)



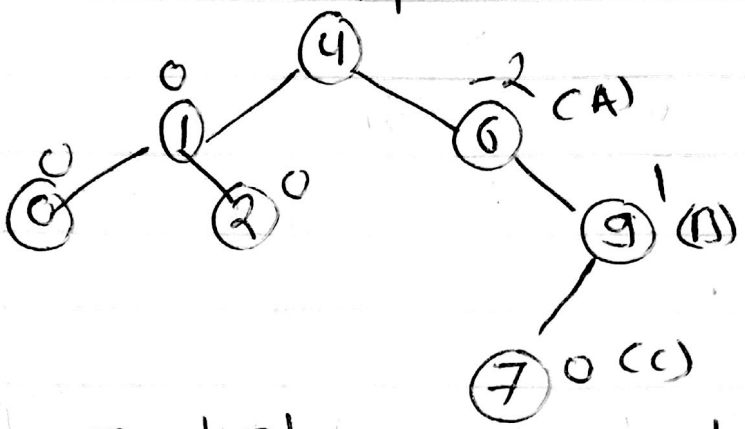
Imbalance occur at 2 because of left subtree of left child.

Apply SRR we get

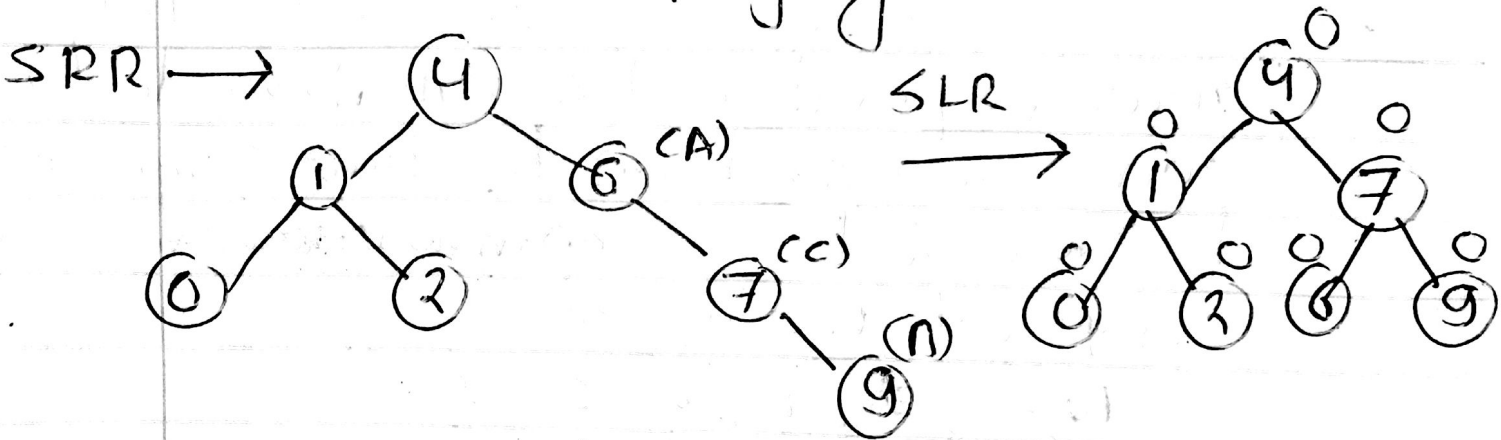


(28)

Now deletion of 5, which is again a leaf node :-

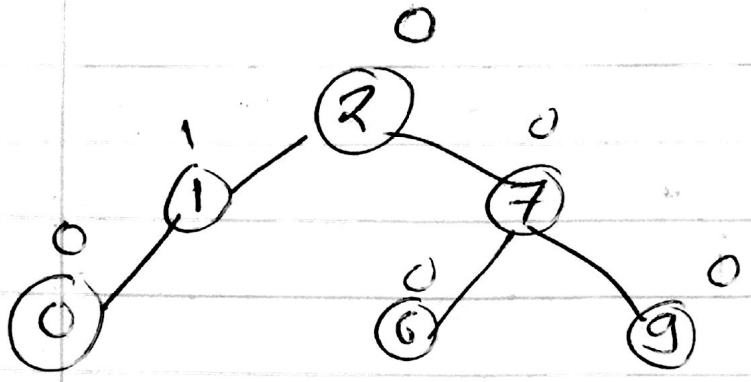


Imbalance occur at 6 because of left subtree of right child of pivot node A. So Applying DRR



Now deleting of 4, which contain two children. (* see Deletion of BST).
so Replacing 4 with 2 i.e 2 is the ~~b~~ rightmost node of left subtree. and deleting leaf 2 we get

(* or we can replace with 6)



This is the final AVL tree.

~~Multiway Search Tree (MST)~~

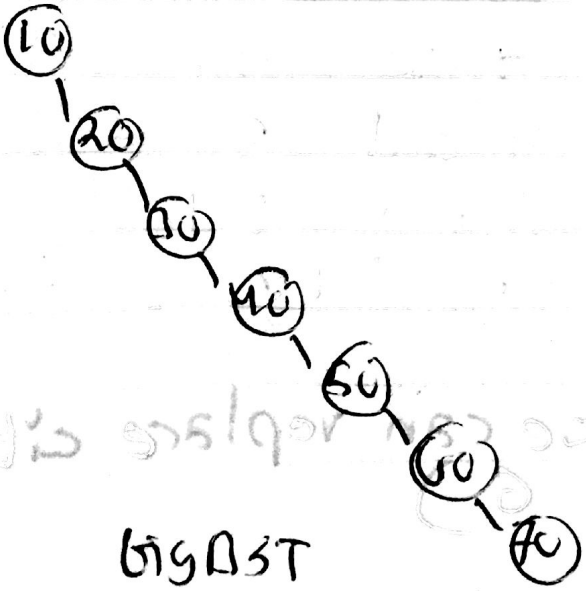
Advantage of AVL tree

- Since AVL trees are height balanced, operation like insertion, deletion and searching have low time complexity because of less comparision.

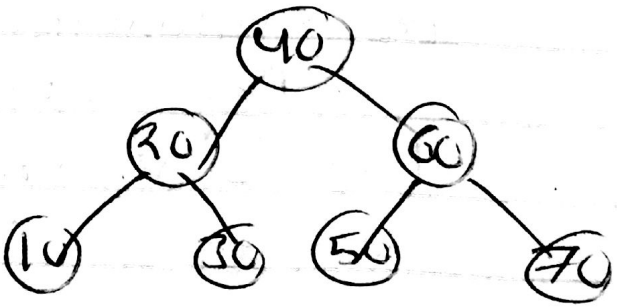
eg: Let the key be

10, 20, 30, 40, 50, 60, 70

Then BST and AVL tree are



bigBST



big: AVL

Q8

So to insert any node in BST, it requires 7 comparisions but in case of AVL tree it requires 3 comparisions. So AVL tree increases performance of programming.

Multiway Search Tree

The number of values that a particular node of BST or AVL tree is only one. To improve the efficiency of operation performed on a tree, we need to reduce the height of the tree. So multiway search tree can be used to improve the efficiency of tree operations. The main goal of m-way search tree is to minimize the accesses while retrieving a key from a file.

A multiway search tree of order n is a tree in which any node contain maximum n-1 values and n children.

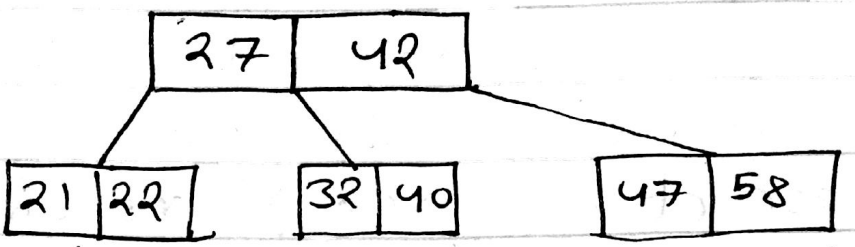


Fig: Multiway search tree of order = 3.

Disadvantage:

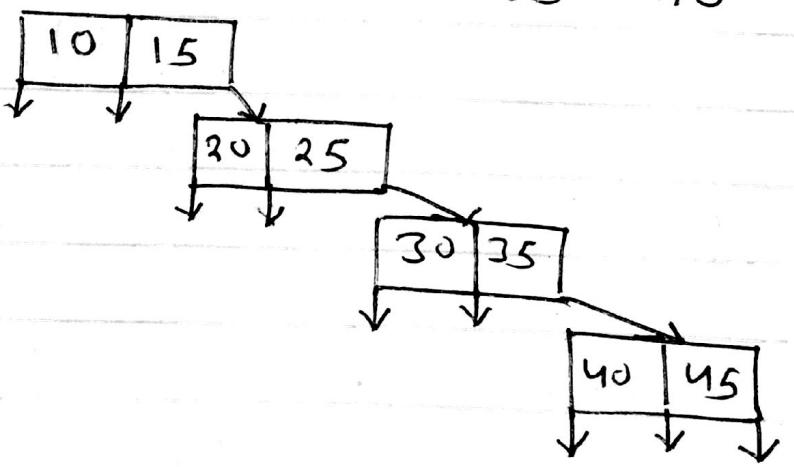
① The tree is not balanced

② Leaf Nodes are on different level.

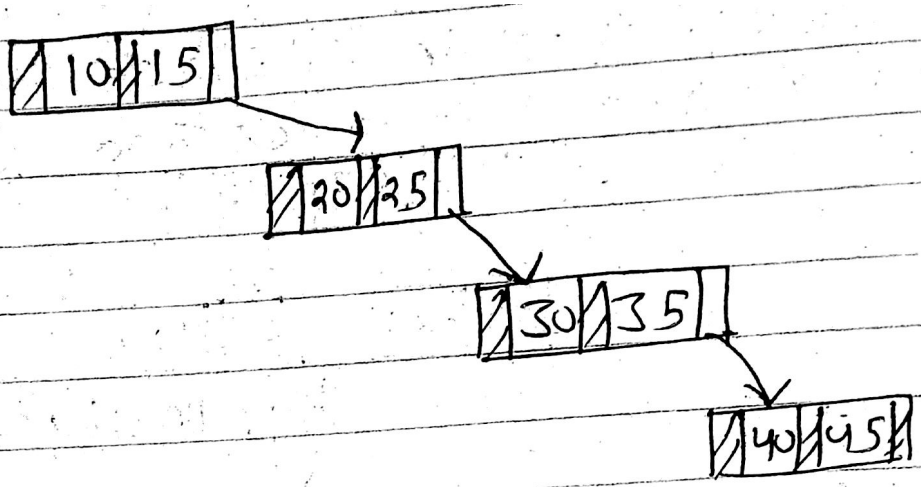
③ Bad space usage.

eg: Construct a 3 way search tree for the list of keys below.

i) 10 15 20 25 30 35 40 45



ii) 20, 35, 40, 10, 15, 25, 30, 45



Family of B-tree

① B tree (Balanced multiway Search tree)

— B-tree of order m is a multiway balanced search tree in which

a) All leaf node are at same level.

b) All the [redacted] node except root have atleast $\lceil m/2 \rceil$ children and at most m children.

[redacted] $\lceil m/2 \rceil$

c) All non leaf node has one fewer keys than the no of its children.

d) when a key (new) is to be inserted into a full node, it is split into two nodes and the key with the median value is inserted in the parent node.

e) All the values of a particular node are in increasing order.

f) All the node except root have atleast (m-1)/2 keys and at most (m-1) keys

The main application of B-tree is the organization of a huge collection of records into a file structure.

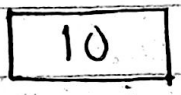
Insertion in B-tree

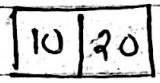
- Traverse the B-tree until we find leaf node suitable for insertion.
- we have two cases for inserting the key
  i) Node is Not full
     - insertion is done in the node in increasing order of keys.
  ii) Node is full
     - Split the node into two nodes at its median at same level, and push the median element up by one level in the parent node.
     - If the parent node is not full, accomodate the element otherwise

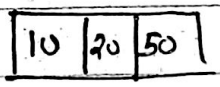
repeat the same procedure in case ii).

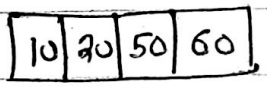
Construct B-tree of order 5 with keys below.
10, 20, 50, 60, 40, 80, 100, 70, 130, 90, 30, 120, 140, 25, 35, 160, 180

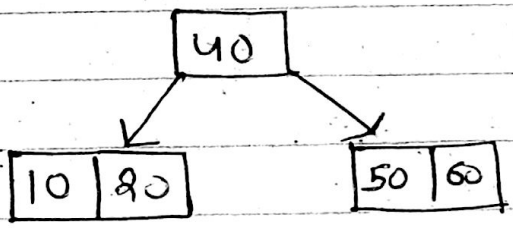
Soln:

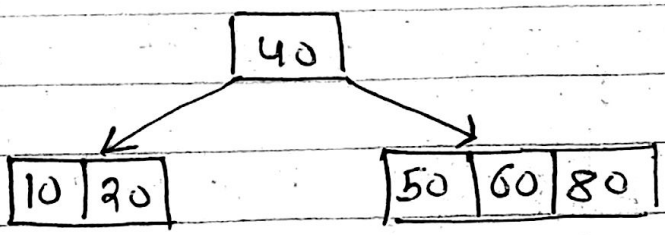
Insert 10 

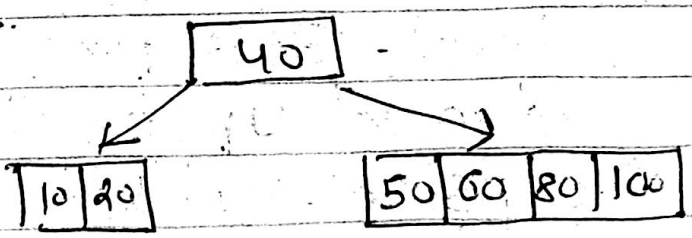
Insert 20 

Insert 50 

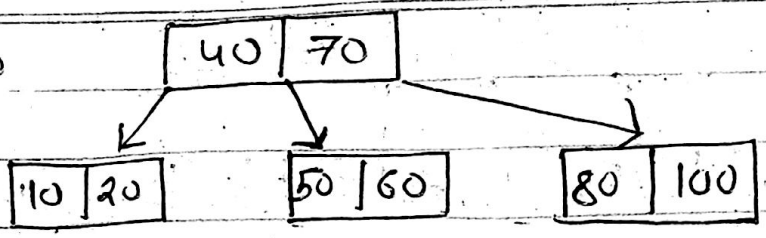
Insert 60 

Insert 40 

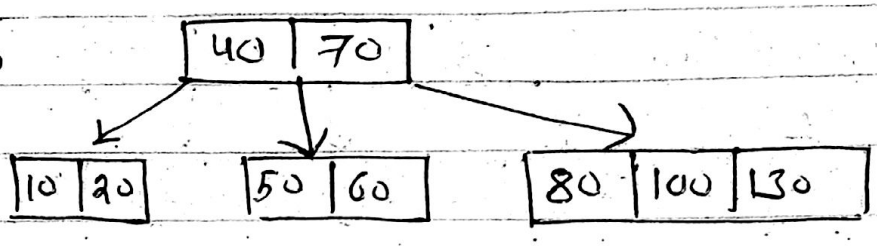
Insert 80 

Insert 100 

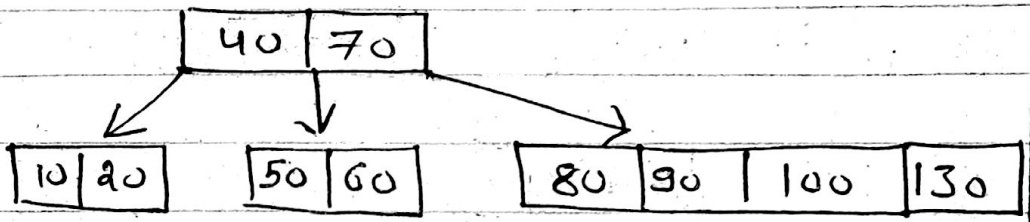
Insert 70



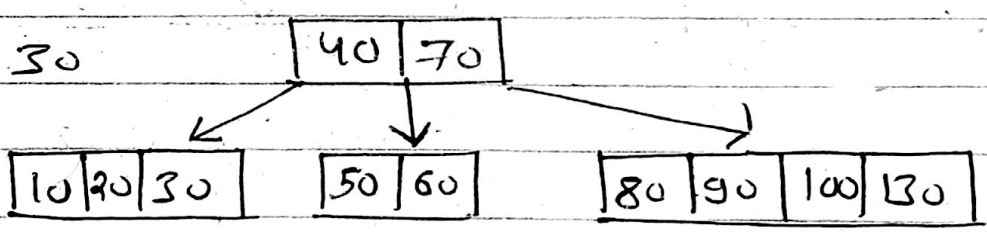
Insert 130



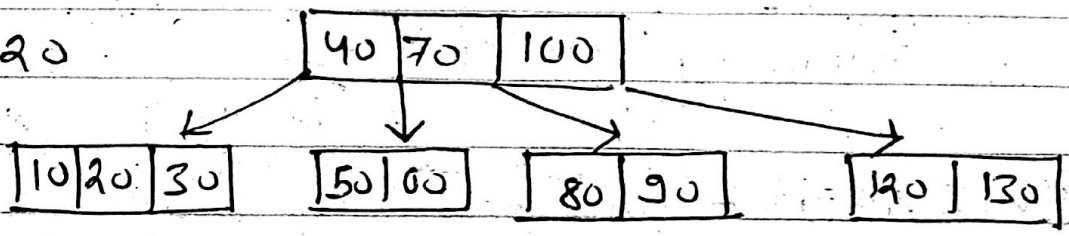
Insert 90



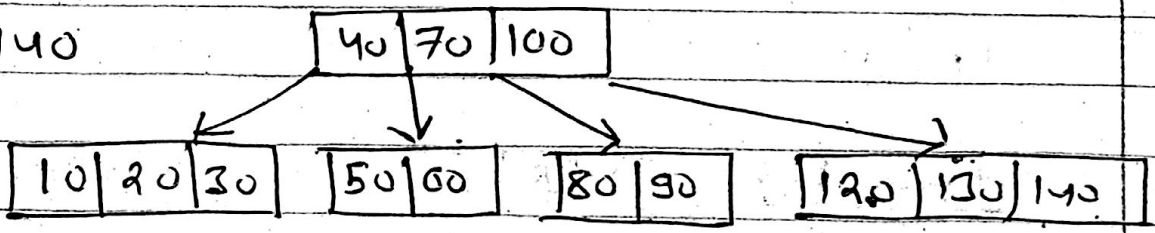
Insert 30



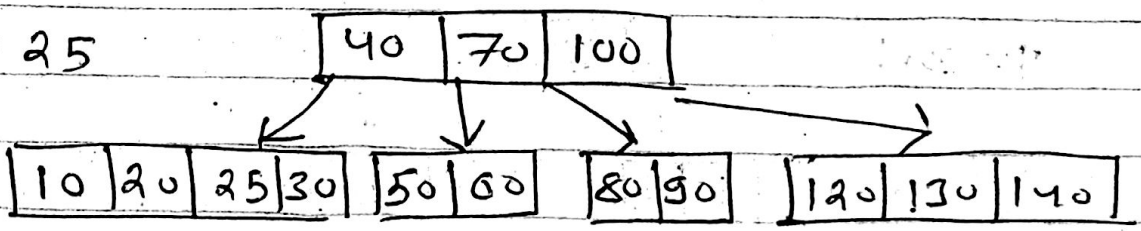
Insert 120



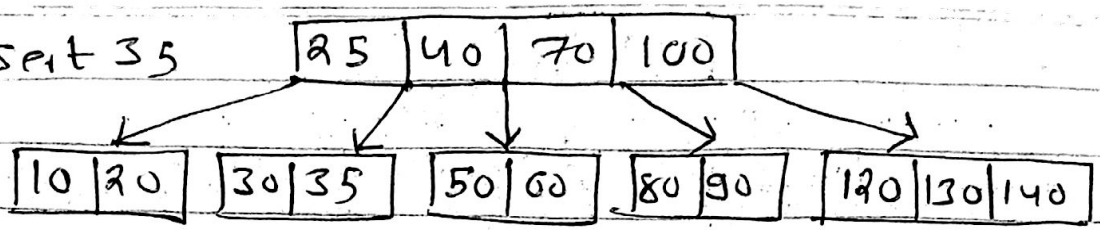
Insert 140



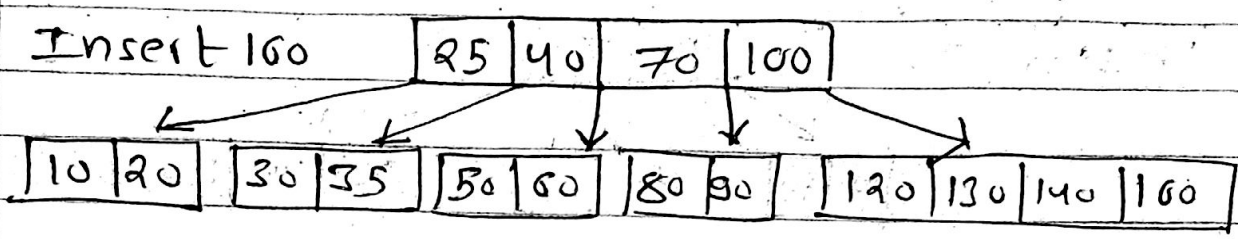
Insert 25



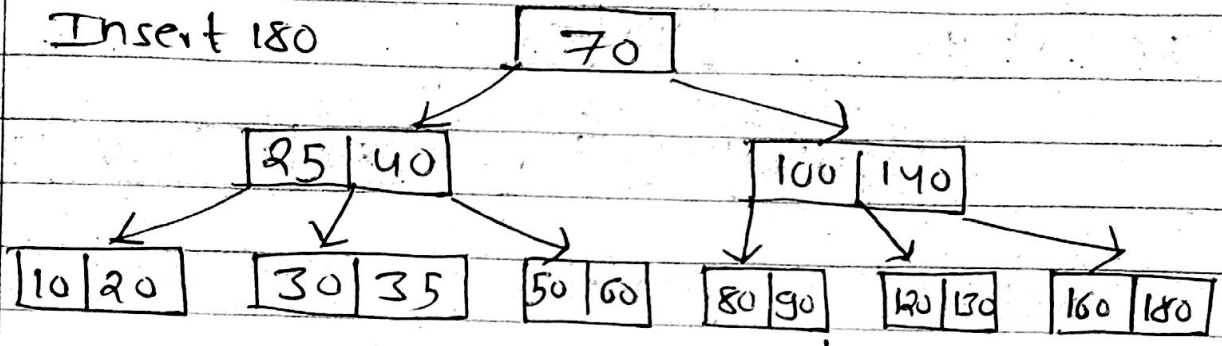
Insert 35



Insert 160



Insert 180

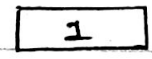


which is final B-tree

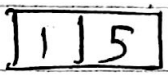
Construct B-tree of order 4 for the following set of keys

1, 5, 6, 2, 8, 11, 13, 18, 20, 7, 9

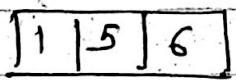
Soln: Insert 1



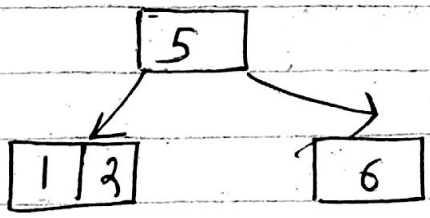
Insert 5



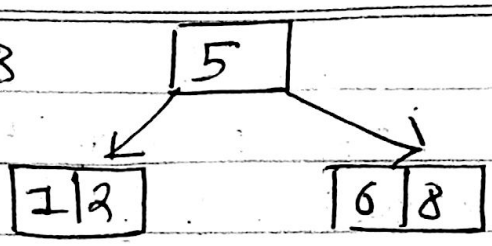
Insert 6



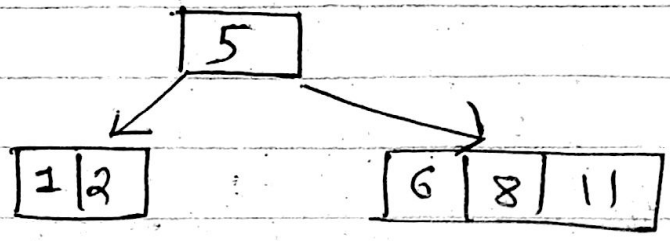
Insert 2



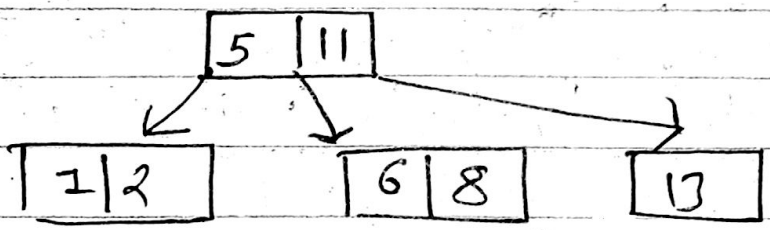
Insert 8



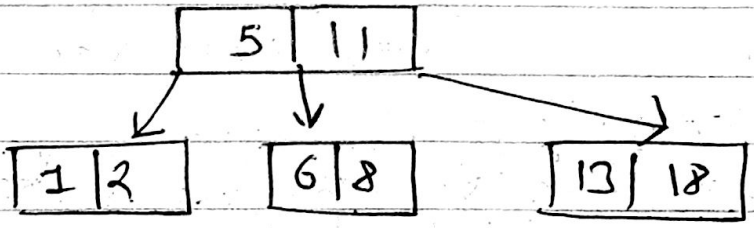
Insert 11



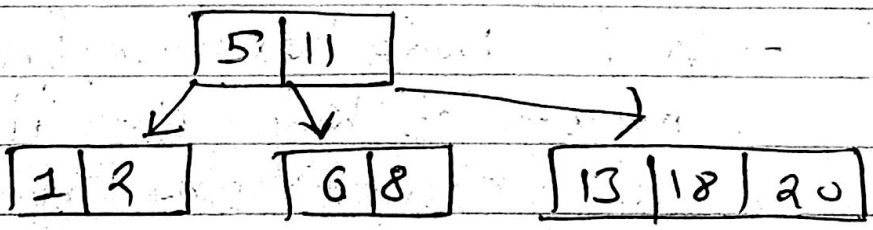
Insert 13



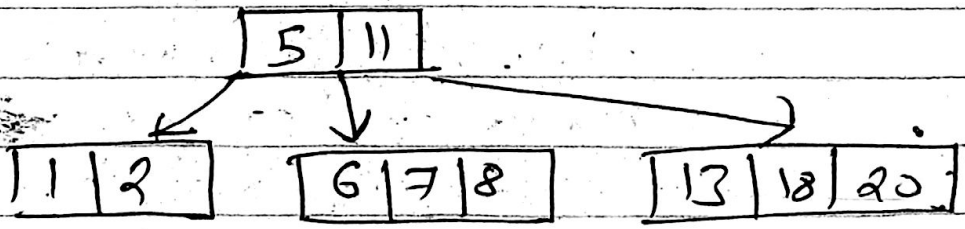
Insert 18



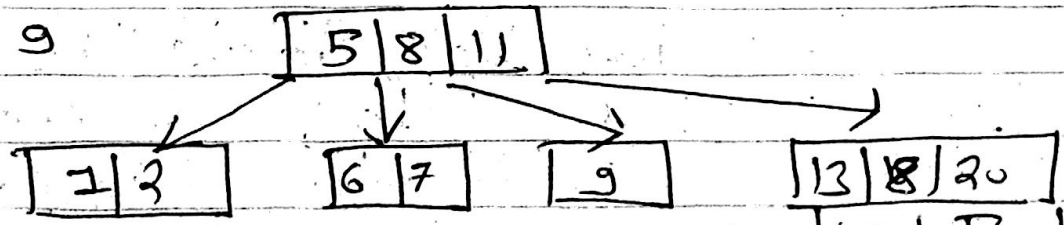
Insert 20



Insert 7



Insert 9



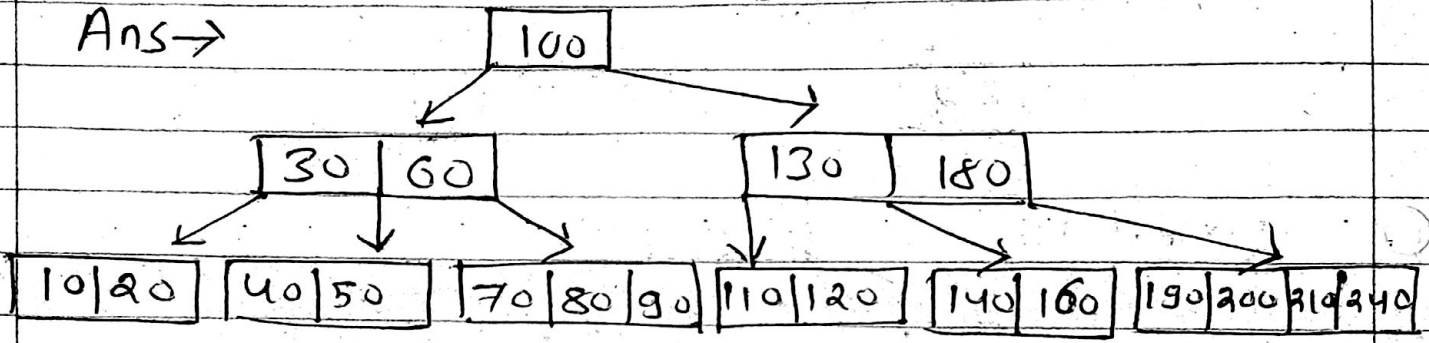
which is final B-tree.

Assignment

Construct B-tree of order 5

10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190, 90, 180, 240, 30, 120, 140, 200, 210, 168

Ans →



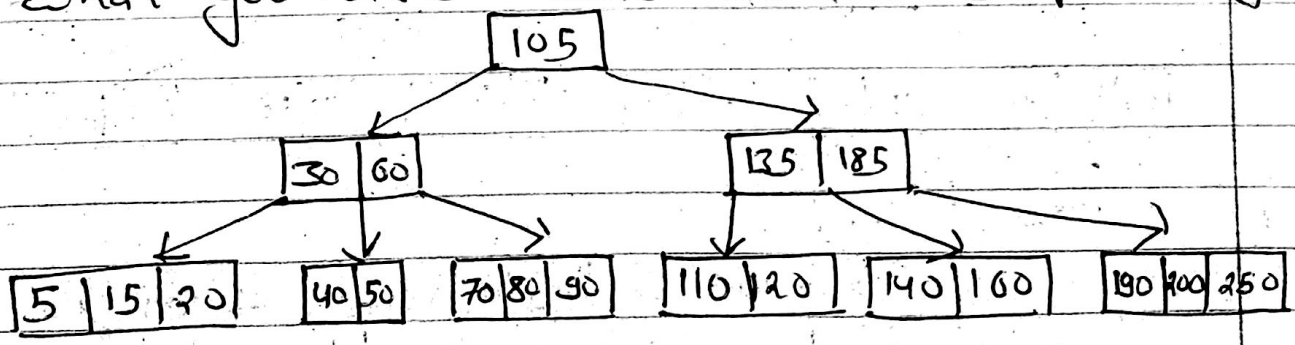
Deletion in B-tree

- Traverse the tree until reaching node containing key
- Now we have two cases for deletion
  - i Node is leaf Node
    - If the Node has more than minimum number of keys then it is simply deleted. and remaining keys are shifted left.
    - If the node has only minimum element, then first see the number of keys in left/Right adjacant node, if it has more than minimum number of keys then ~~last key~~, delete the node from leaf node then left/Right side element in parent node will come in leaf node and

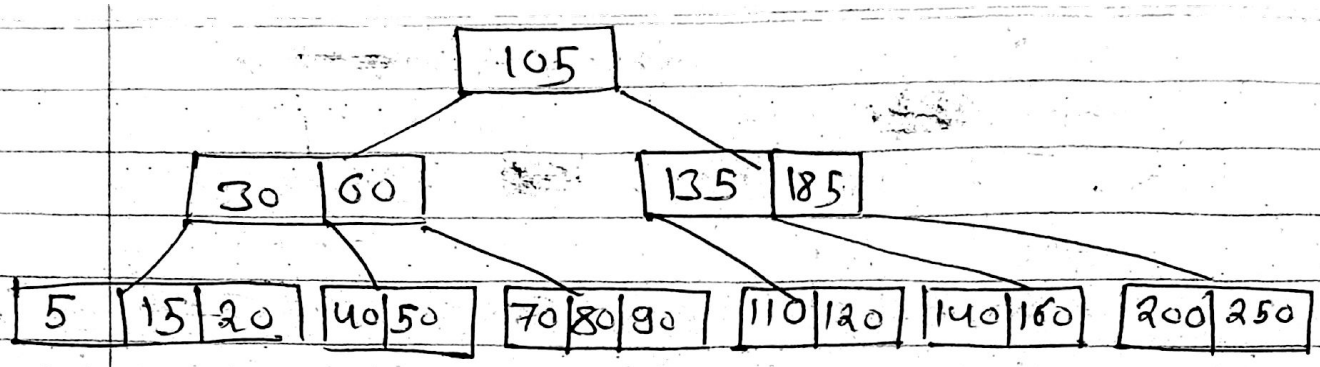
first
the last element of [redacted] adjacent [redacted] node will come in parent node. But if the [redacted] adjacent node i-e sibling also has minimum number of keys then leaf, and separator key and sibling are merged and the sibling node is discarded. Suppose now parent has also less than minimum number of keys then same thing will be repeated until it will get the node which has minimum number of keys.

ii) Node is non leaf node: see back page.

Let us take B-tree of order 5. Show the deletion of node 190, 60, 40, 140, explaining what you have done at each step.

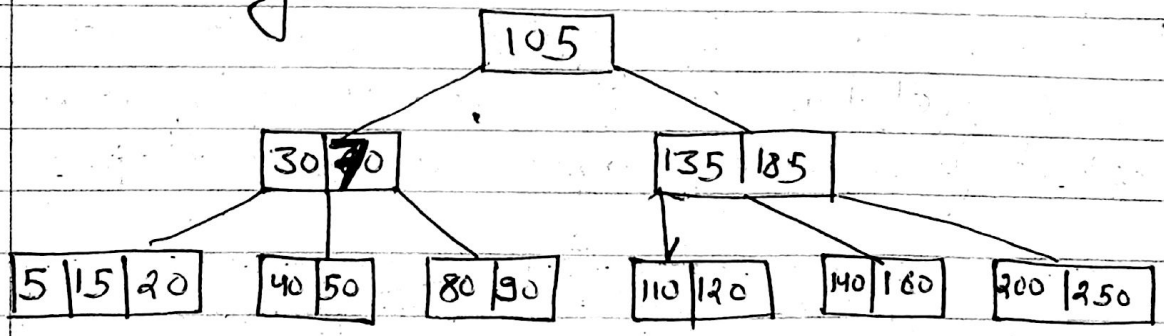


1) Delete 190,
Here 190 is leaf node, so it is simply deleted and all keys greater than 190 are shifted left.



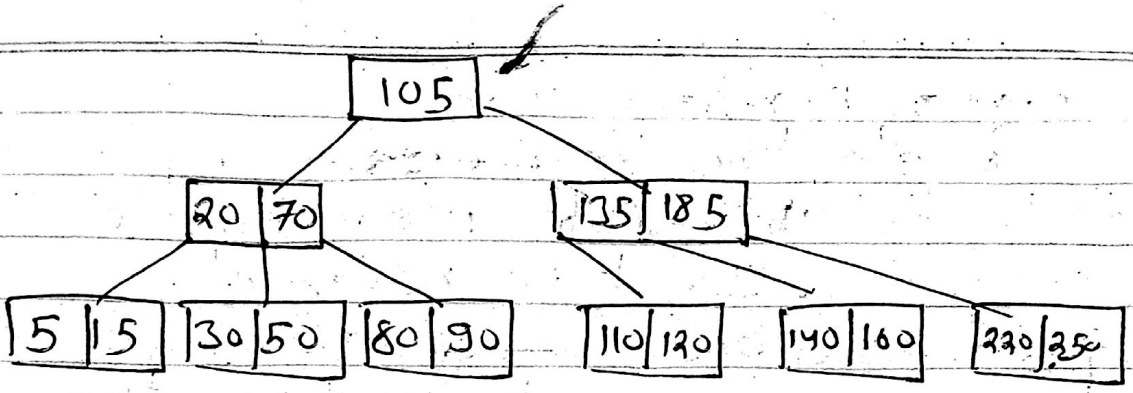
ii) Delete 60

Here 60 is non leaf node, so first it will be deleted from the node and the immediate successor is placed at that node. as the successor node has more than minimum no of keys.



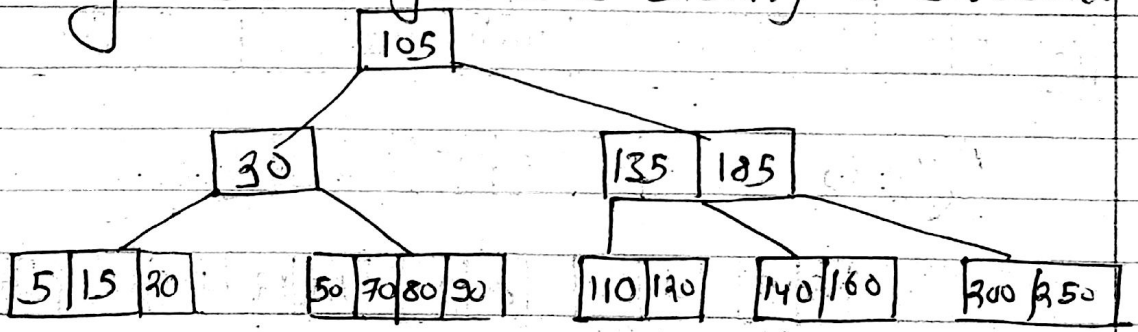
iii) Delete 40

Here 40 is leaf node, when 40 is deleted from leaf node then left side element in the parent node will come in leaf node and the last element of the left side node will come in parent node.

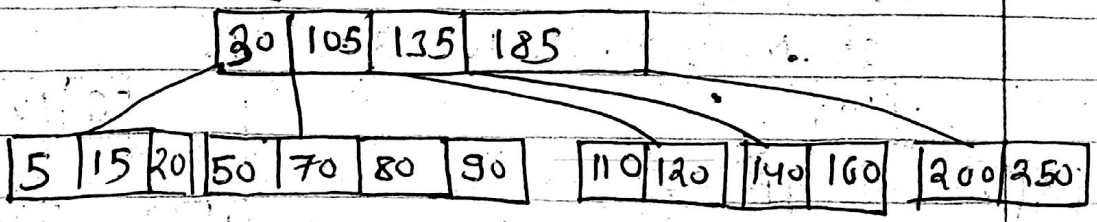


OR

Here 40 is leaf node, when 40 is deleted from leaf node then Right side element from the parent node will come in leaf node. But the right side node (sibling) has minimum number of keys so leaf, separator and sibling are merged and sibling is discarded.



Now parent has less then minimum number of keys so same process is repeated.

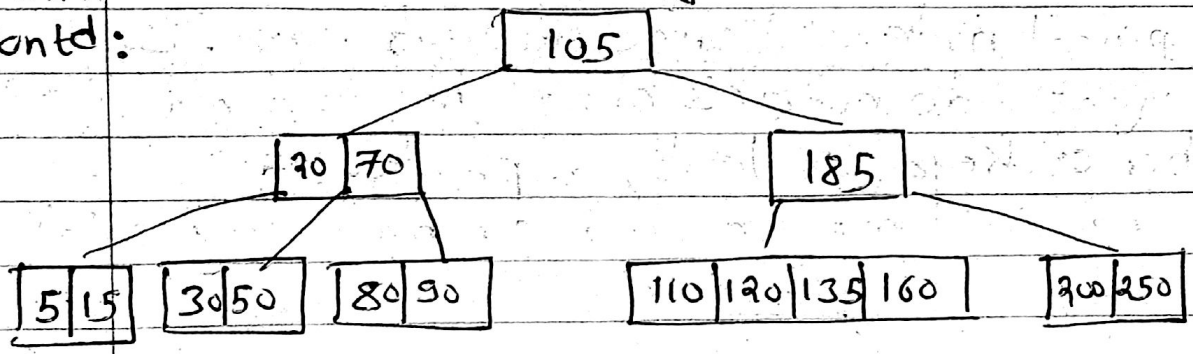


iv) Delete 140

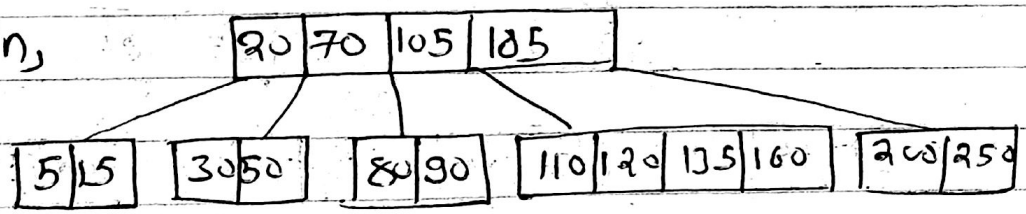
contd back. ***

(ii) Node is non-leaf node
— delete the key ~~and~~ by replacing with its immediate predecessor or successor. If both nodes of predecessor and ~~succer~~ successor have minimum number of keys then the nodes of predecessor and successor keys will be combined.

***
contd:



Again,



Assign #

Suppose a B-tree of order-5 as in the fig I. Construct final B-tree after successive deletion of node 8, 18, 16 and 4.

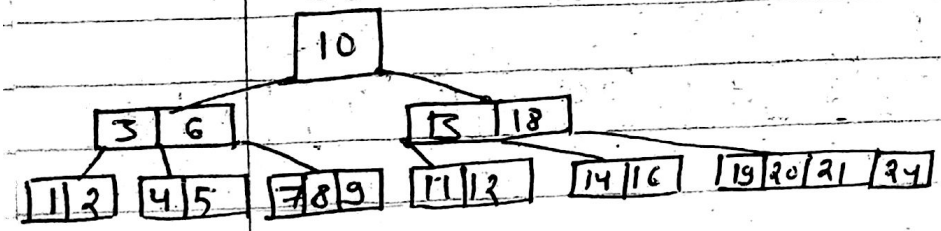
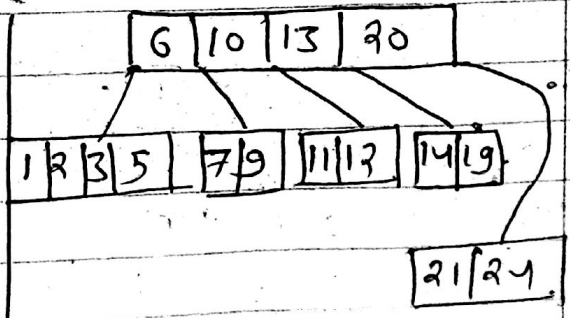


Fig I



Ans #

The Huffman Algorithm:-

~~Suppose that we~~

The Huffman algorithm is used to perform encoding and decoding of a long message consisting of a set of symbols. Message should be short and compact while sent over network.

Suppose that we have an alphabet of n symbol and a long message consisting of symbols from this alphabet. We wish to encode message as a long bit string (either 0 or 1) by asigning a bit string code to each symbol of the alphabet and concatenating the individual codes of the symbols making up the message to produce an encoding for the message. For example, the alphabet consist of 4 symbols A, B, C and D and codes are assigned to the symbol as follows.

Symbol	Codes
A	101
B	100
C	000
D	001

7.

The message ABACCDA would then be 1011001010000000001101.

Such encoding is inefficient because it uses fixed bit sequence (3 bit) for each symbol. So it requires 21 bits to encode the entire message.

Suppose we use 2 bits for assigning each symbols as follows

Symbols	Code
A	00
B	01
C	10
D	11

Then the message ABACCDA would be 00010010101100.

So it requires only 14 bits. However if want to reduce the bit further then we need ~~to use variable bit sequence~~ to generate a code of variable bit sequence. for this Huffman algorithm is used..

In the above message, each of B, C and D appears only

once in the message, but A appear 3 times. If a code is choosen so that letter A is assigned a shorter bit string than B and D the length of encoded message would be small. Consider following codes are assigned to symbol

Symbol	Code
A	0
B	110
C	10
D	111

Using this code, the message is encoded as 01100101011110 which require only 13 bits. In above the code but each symbol should be such that code of any character is not a prefix of code for other characters, otherwise it creates ambiguity in decoding process.

The decoding starts by scanning a string from left to right. If a 0 is encountered

(26).

as the first bit, the symbol is an A, otherwise it is B, C or D. Now the next bit is examined. If the second bit is 0, the symbol is C otherwise it must be B or D and the third bit must be examined. If the third bit is 0, the symbol is B. If it is 1, the symbol is D. As soon as the first symbol has been identified, the process is repeated, starting at the next bit to find the second symbol.

Building Huffman tree using Huffm an algorithm.

- A special type of tree called Huffman tree can be used to generate a code for variable length encoding scheme.

In Huffman tree, each leaf node is labelled with symbol. A binary code can be assigned to each symbol as follows

- Associate 0 to path from a node to its left child
- Associate 1 to the path from a node to its right child

Now the code for a symbol is obtained by following the path from the root to the leaf node containing the symbol.

Algorithm

1. Get the string.
2. Get the frequency of occurance of each symbols
3. Select the two symbols having the lowest probability occurance
4. Combine these two into a single.
5. Now repeat this till the final symbol is encountered.

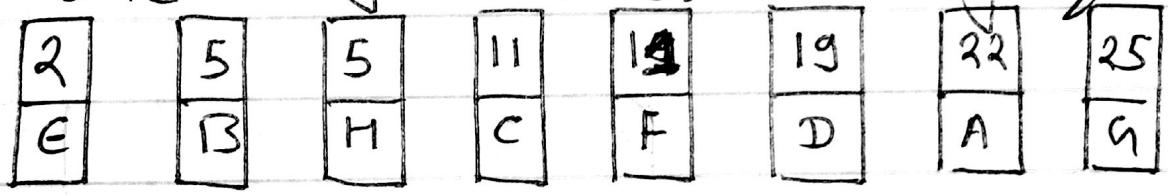
26

A, B, C, D, E, F, G, H are the data item and the assigned weight is given below. Generate Huffman code for each symbol.

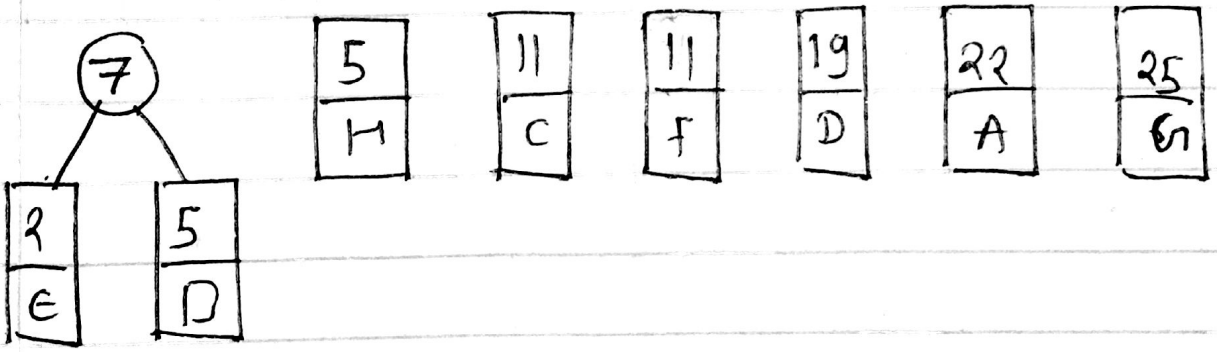
Symbol	A	B	C	D	E	F	G	H
Frequency	22	5	11	19	2	11	25	5

Soln: Arranging the ~~letters~~ symbols with the order of their frequency.

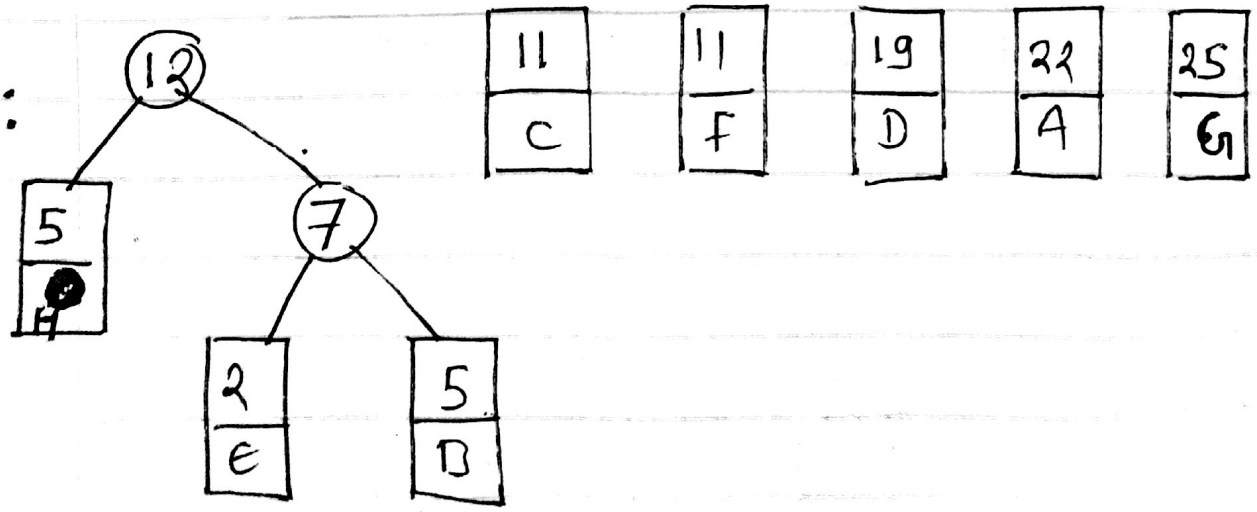
1:



2:

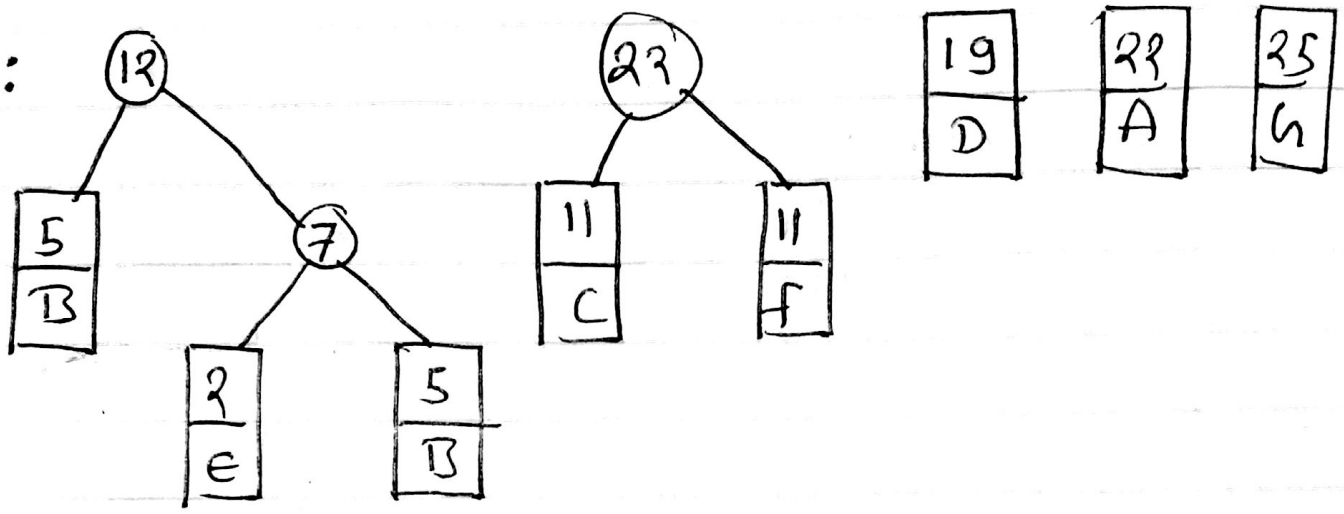


3:

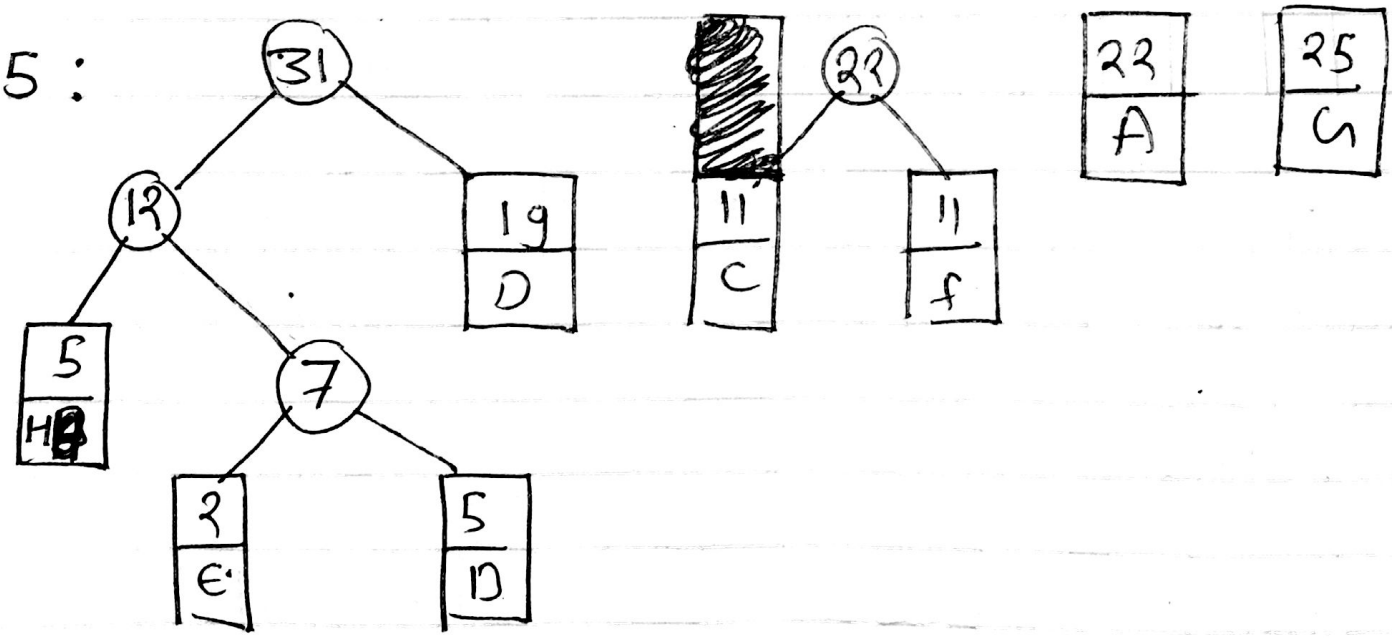


75

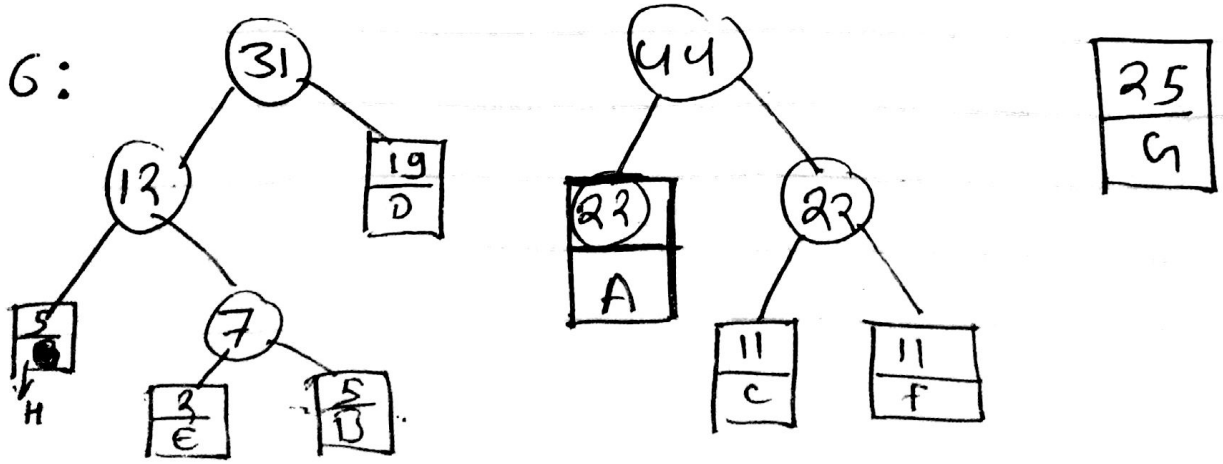
4 :



5 :

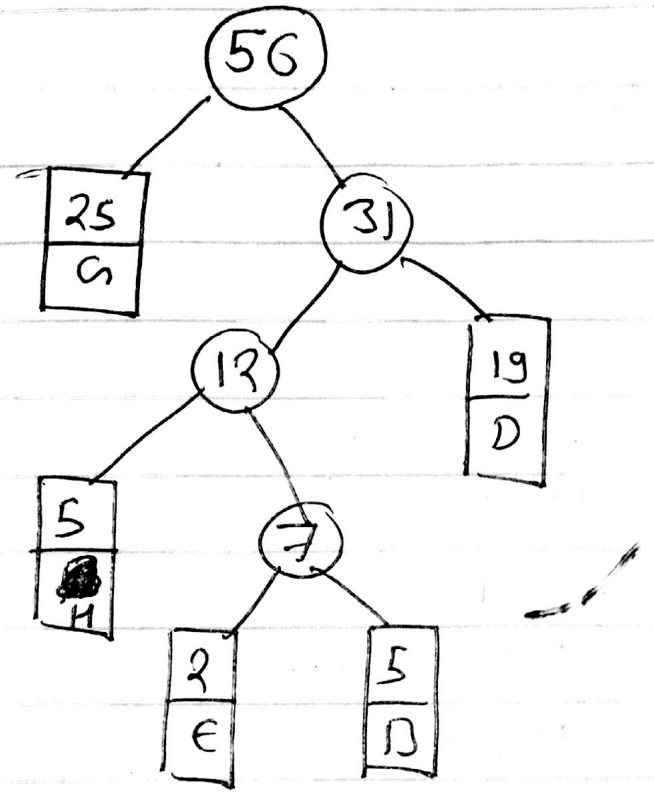


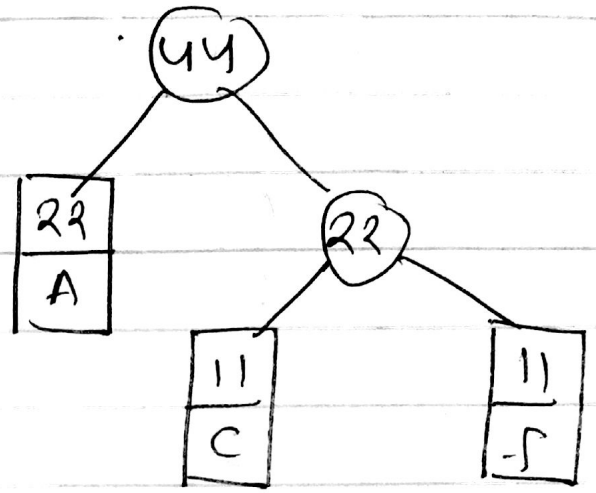
6 :



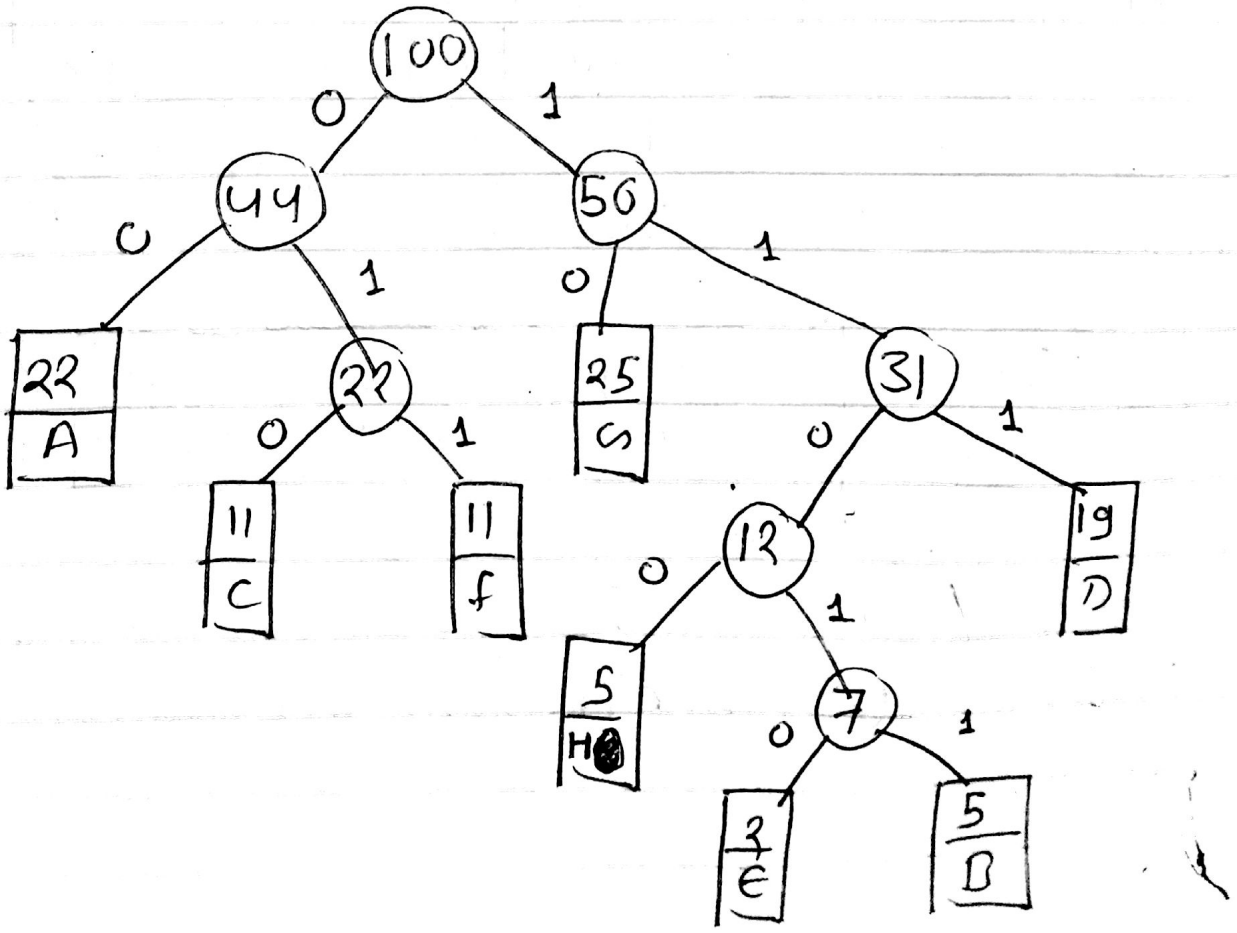
76

&C





7:



The Huffman code is given below

Symbol	code
A	00
B	11011
C	010
D	111
E	11010
F	011
G	10
H	1100

Construct Huffman tree and generate code for the following symbol/letter with their frequency given below.

Symbol	C	D	E	K	L	M	U	Z
Frequency	32	42	120	7	42	24	37	2

Soln: Arranging the symbol in order of their frequency

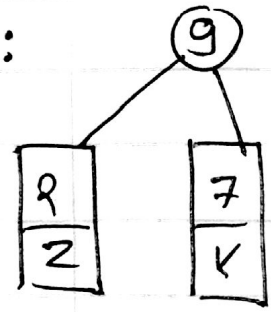
78

28

1:

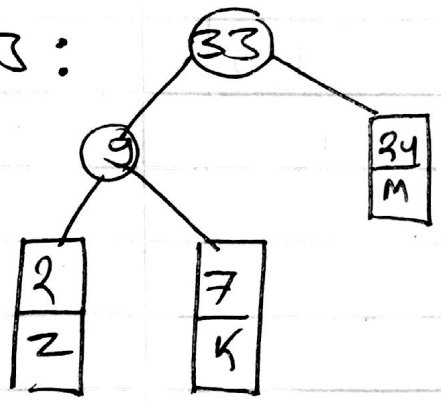
2	7	24	32	37	42	42	120
Z	K	m	C	U	D	L	€

2:



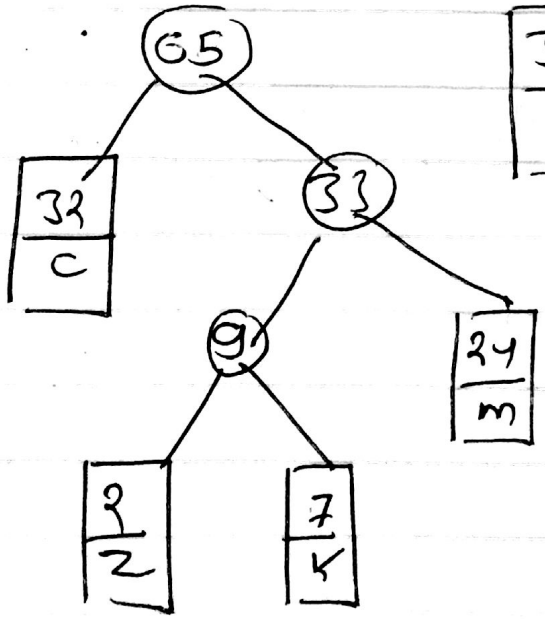
24	32	37	42	42	120
m	C	U	D	L	€

3:



32	37	42	42	120
C	U	D	L	€

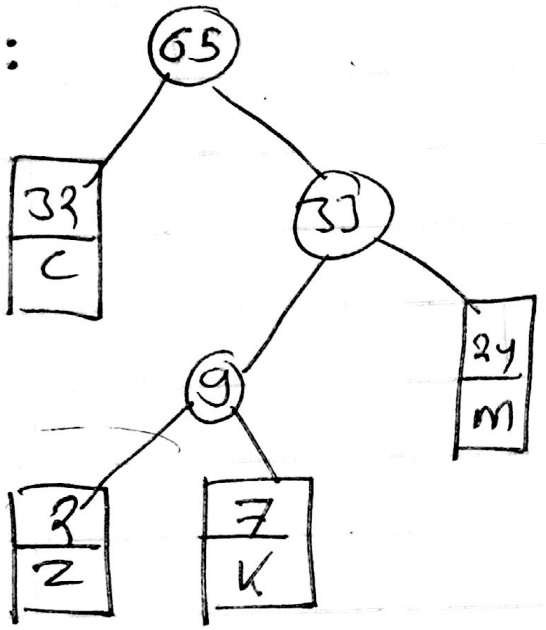
4:

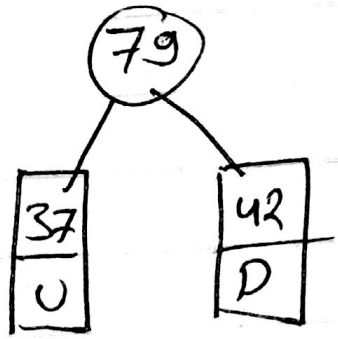


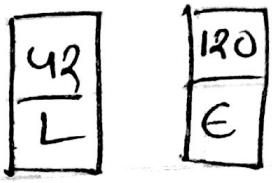
37	42	42	120
U	D	L	€

79

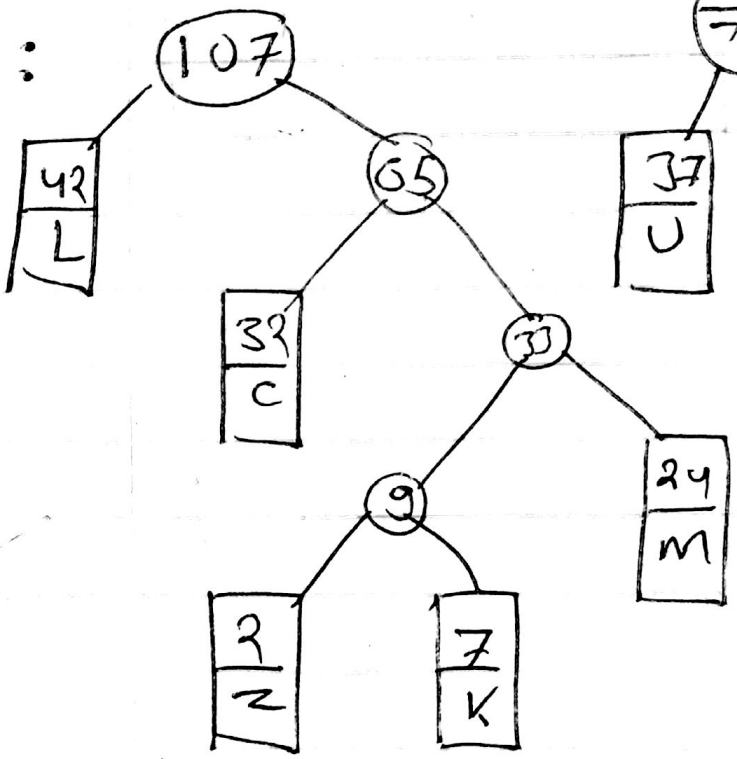
5:

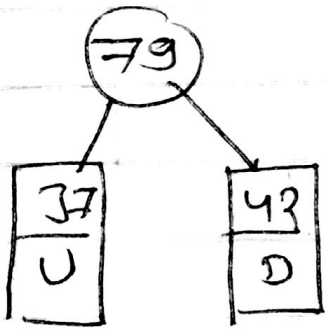


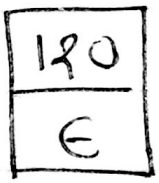




6:





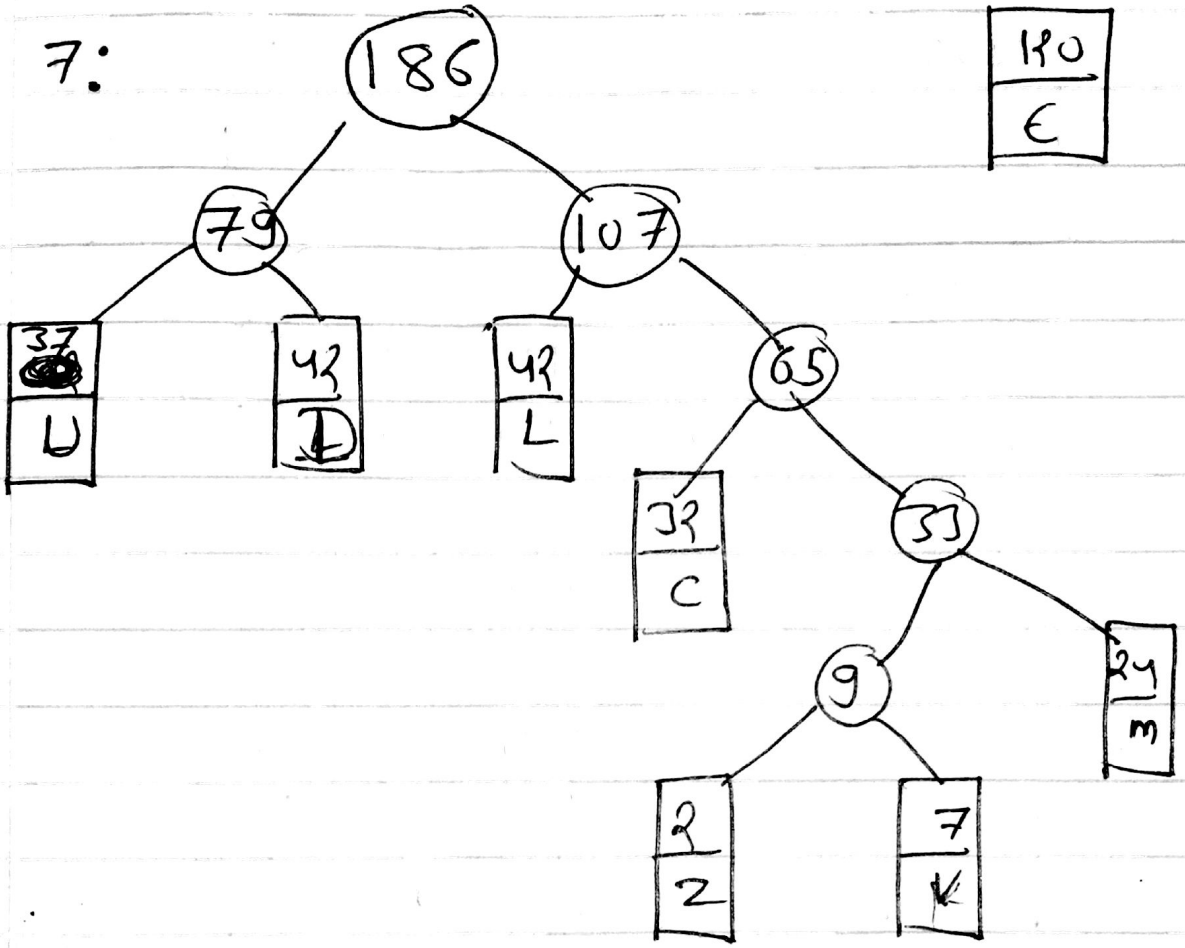


2

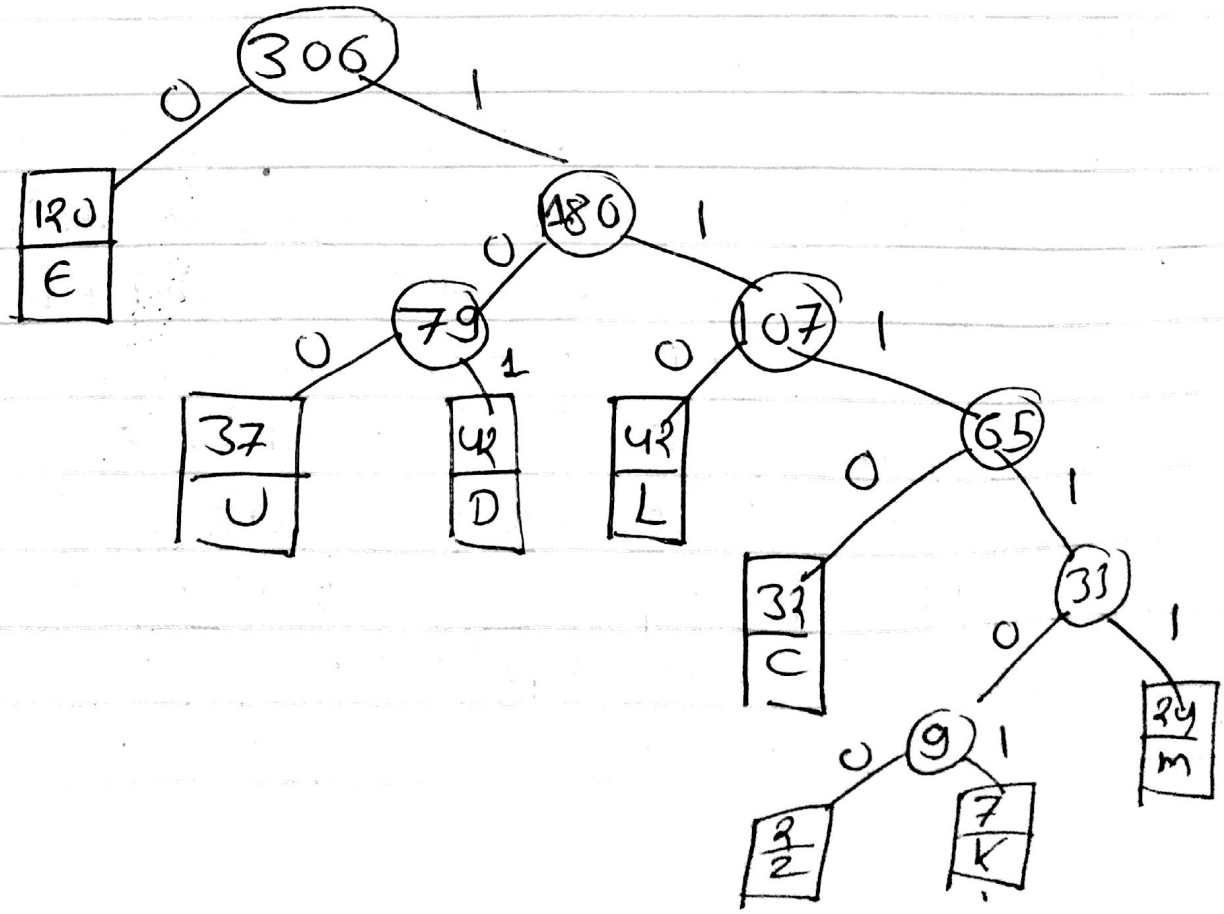
80

80

7:



8:



Now from the above Huffman tree, the Huffman code generated are

Symbol	Code
C	1110
D	101
E	0
K	111101
L	110
M	11111
U	100
Z	111100

Game Tree

A game tree is a name given for any regular tree that maps how a discrete game is played.

Let us take an example of game called Rocks. In this game we have two piles of rocks and the first pile consist two rocks and the second pile has one rock. However the number of piles and number of

~~Each~~ rocks in each pile may be increased as per our wish. The game has two player, who take turns taking one or more rocks from a single pile until one pile is left. When one pile is left, our goal is to force our opponent to remove the last rock. The person who remove the last rock looses.

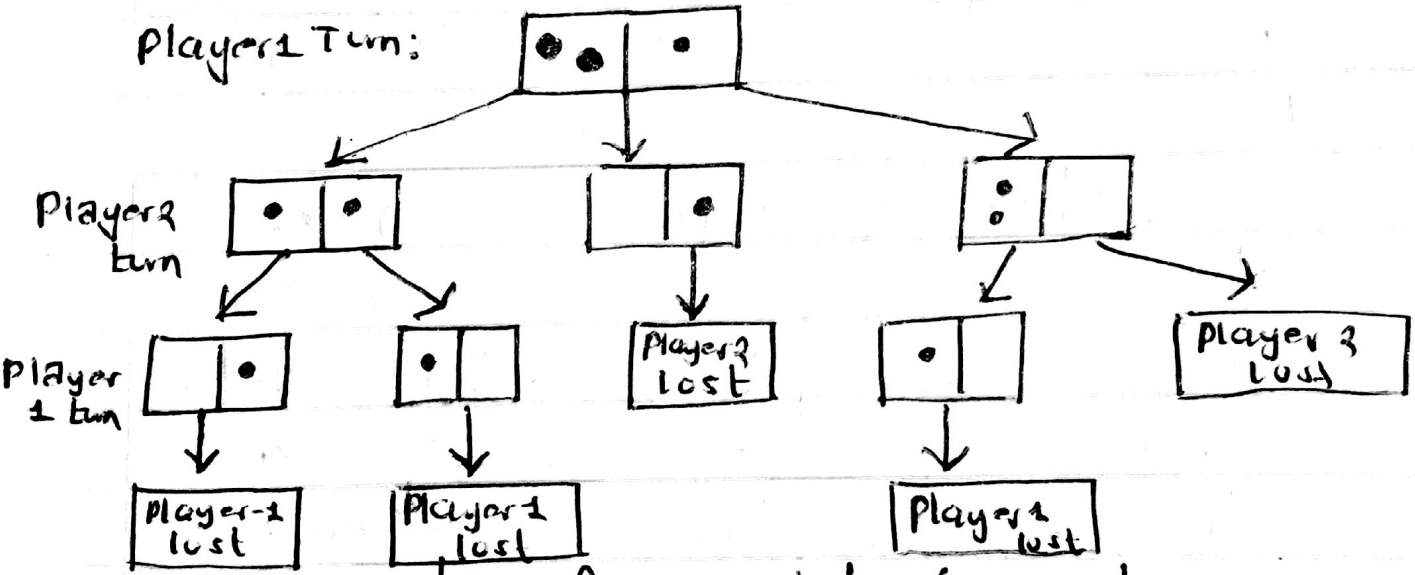


fig: A complete game tree.

Above figure is a complete game tree to represent all the possible moves that exist in the game. Initially, at level 0, the 1st pile has 2 rocks and 2nd pile has one. Now the first player has three choices.

① Remove one rock from pile1

② Remove two rock from pile 1.
③ Remove one rock from pile 2.

All The possible states of pile after the first player turn is shown by level 2 in game tree. After player 1, it's now player 2 turn. Player 2 choices is limited to current state of game. Let us suppose that player 1 has removed one rock from first pile. Now player 2 can remove either one rock from first pile or one rock from second pile. Whatever move he chooses, player 1 will be left with single rock still, as shown in level 2 of game tree, (subtree for choice 1). So player 1 loses if he make choice 1.

By our game tree, we see that player 2 lost for 2nd choice of player 1 and player 1 or player 2 might loose for player 1 3rd choice i.e if player 2 removes both rock then player 2 lost, if player 2 remove one rock then player 1 lost. The game is entirely completed by the time ~~fourth level~~ third level is reached.

Creation of Binary tree using Traversals.

- It is the process of constructing a binary tree using the traversal output

Two ways

① Preorder and inorder traversal
② Postorder and inorder traversal

① Creating a Binary tree from preorder and inorder traversal.

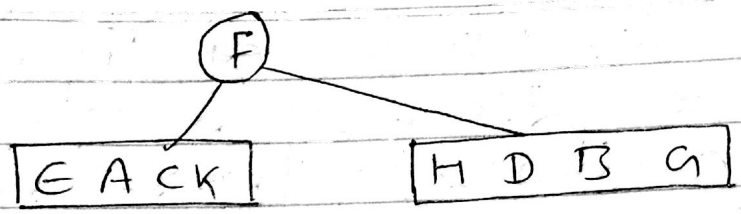
- Scan preorder traversal from left to right.
- For each node scanned locate its position in inorder traversal. Let the scanned node be x.
- The node precceding x in the inorder form its left subtree and node succedding it form a right subtree.
- Repeat above steps for each symbol in the preorder traversal.

Draw a binary tree.

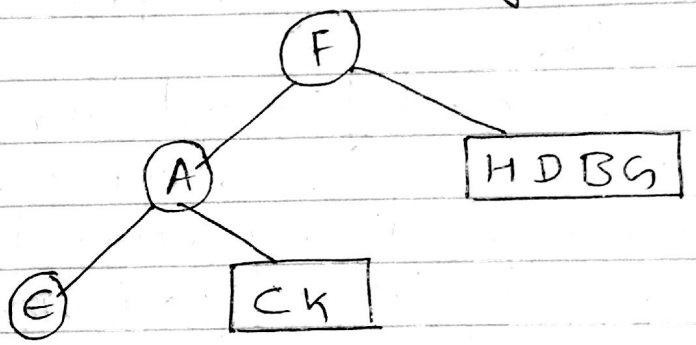
Preorder: F A E K C D H G B
Inorder: E A C K F H D B G

Soln: In preorder, root is the first node, so scanning the pre-order from first.

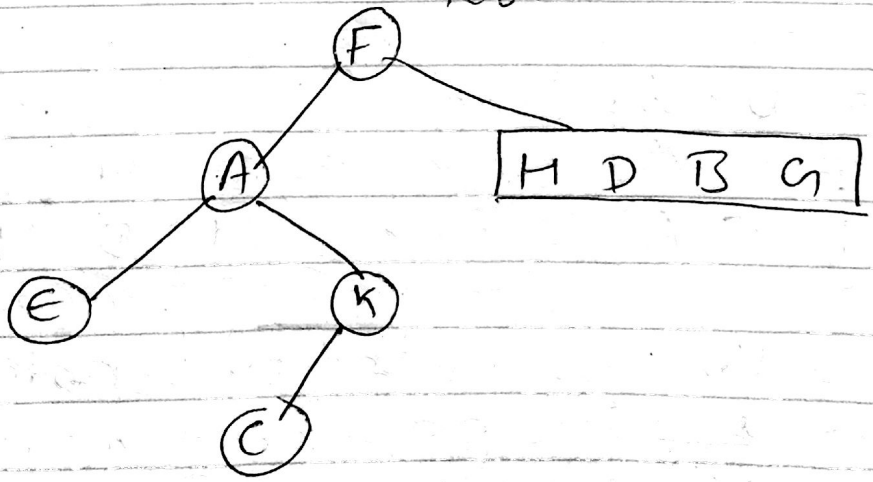
Pre. order: (F) A E K C D H G B
In. order: E A C K, (F) H D B G,
left, Right



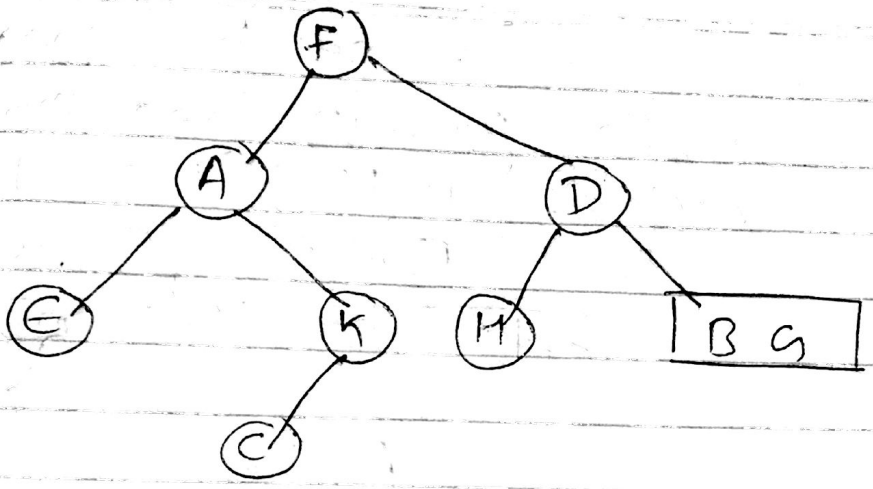
Again, Pre.order: F A ∈ K C D H G B
In.Order: ∈ A C K, F H D B G
left Right



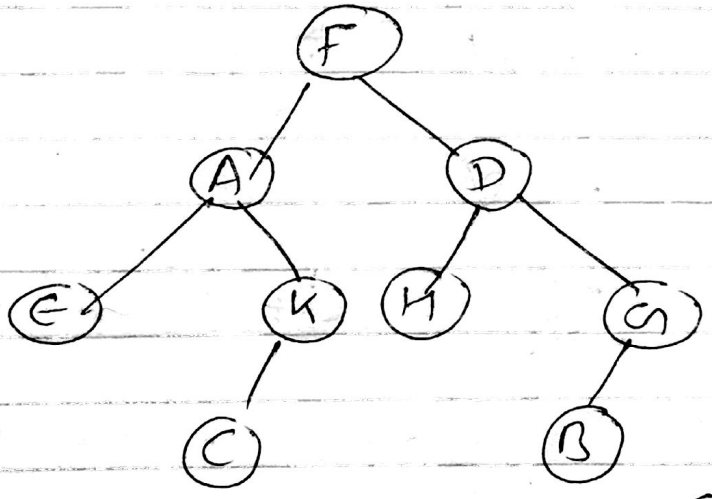
Again, Pre-order: F A ∈ K C D H G B
In-order: ∈ A C K F H D B G
left



Again, Pre-order: F A ∈ K C D H G B
In-order: ∈ A C K F H, D B G,
left Right



Again, Preorder: F A E K C D H G B
In-order: E A C K F H D B G
left



which is final Binary tree.

Construct binary tree for the given traversal.

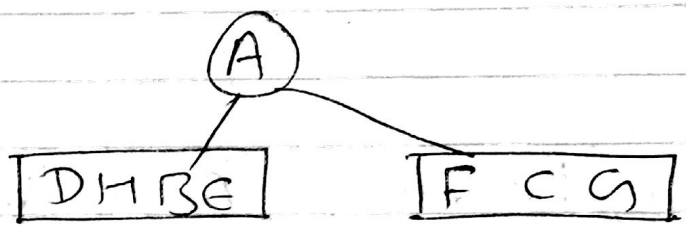
Preorder: A B D H E C F G
Inorder: D H B E A F C G

Soln: In preorder, root is the first node so scanning the preorder from first.

Preorder: (A) B D H E C F G

Inorder: D H B E (A) F C G

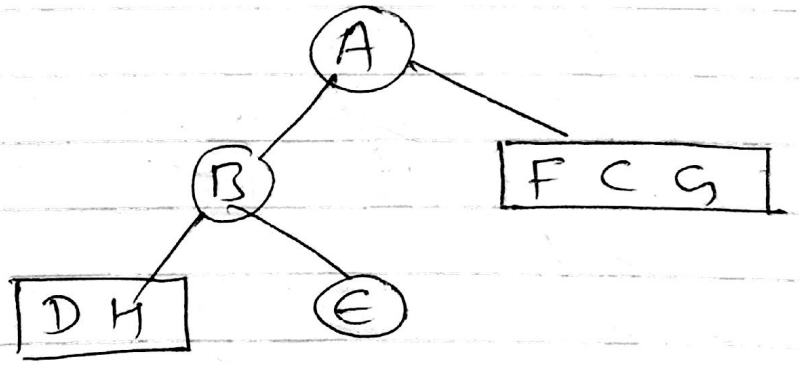
left | Right



Again, Preorder: A (B) D H E C F G

Inorder: D H (B) E A F C G

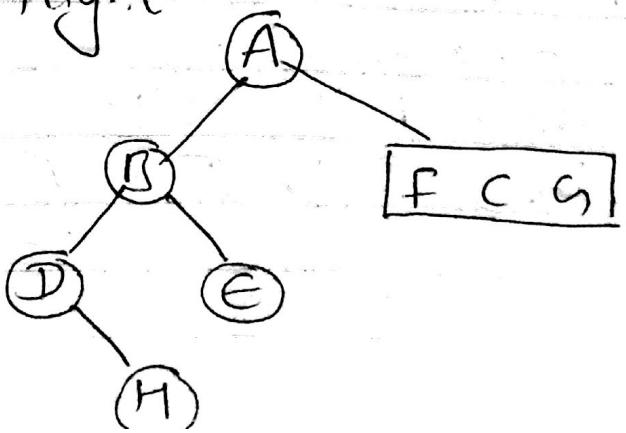
left | Right



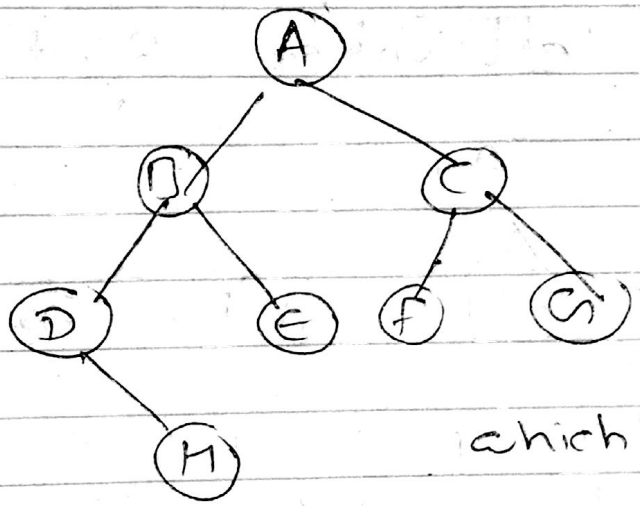
Again, Preorder: A B (D) H E C F G

Inorder: (D) H B E A F C G

Right



Again, Preorder: A B D H E C F G
Inorder: D H B E A F C G
left Right



which is final B-tree.

Creating a Binary tree from postorder and inorder traversal.

a) Scan the postorder traversal from Right to left.

b) For each node scanned, locate its position in inorder traversal. Let the scanned node be x.

c) The node preceeding x in inorder form its left subtree and nodes succeding it form a right subtree.

d) Repeat above steps for each symbol in postorder.

Draw binary tree.

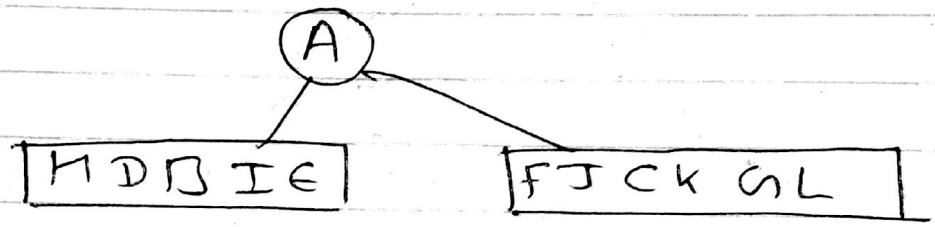
Postorder: H D I E B J F K L G C A
Inorder: H D B I E A F J C K G L

Soln: In postorder, root is the last node so scanning the postorder from right/last.

Postorder: H D I E B J F K L G C (A)

Inorder: H D B I E (A) F J C K G L

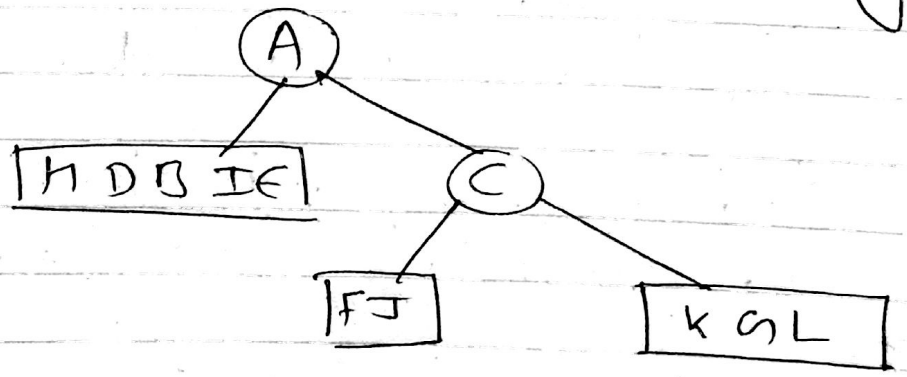
left subtree Right subtree



Postorder: H D I E B J F K L G (C) A

Inorder: H D B I E A F J (C) K G L

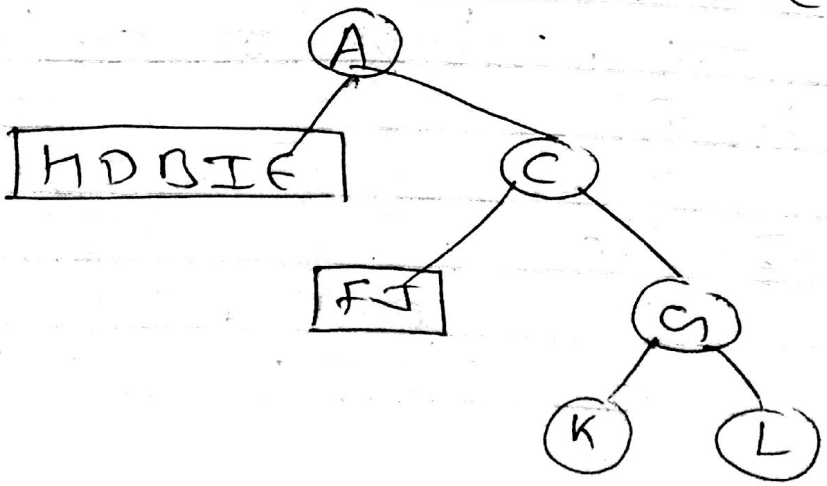
left Right



Postorder: H D I E B J F K L (G) C A

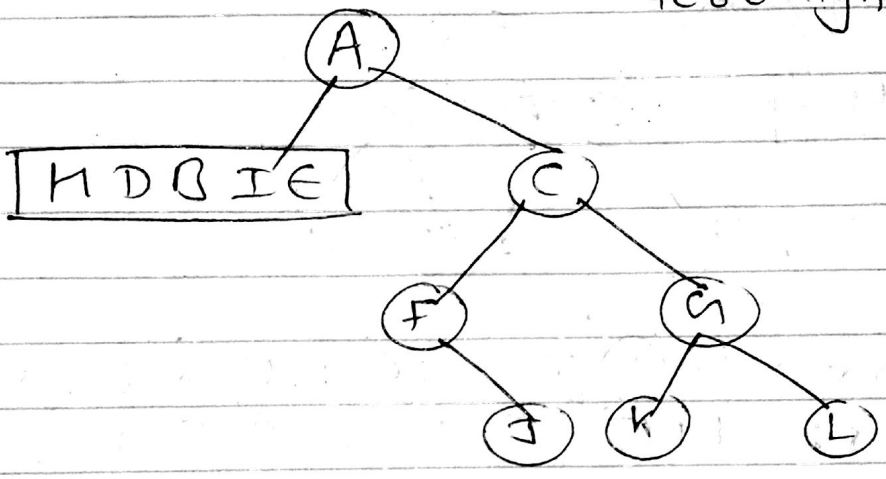
Inorder: H D B I E A F J C K G L

left Right



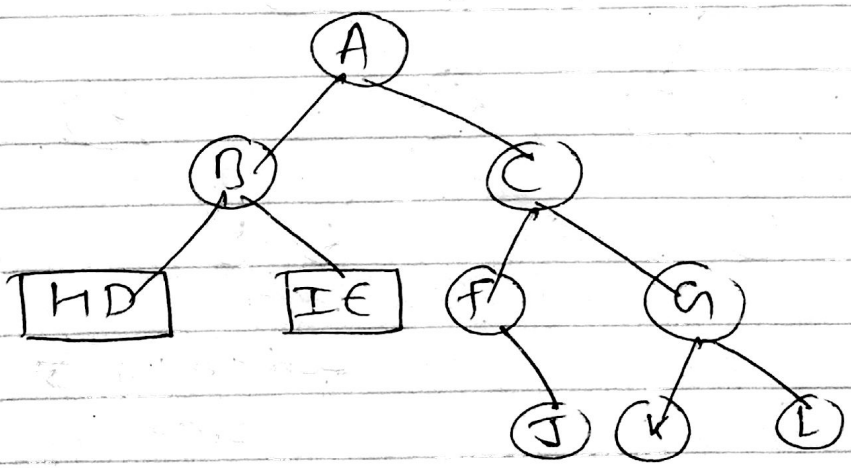
Postorder: H D I E B J F K L G C A
Inorder: H D B I E A F J C K G L

~~left~~ Right



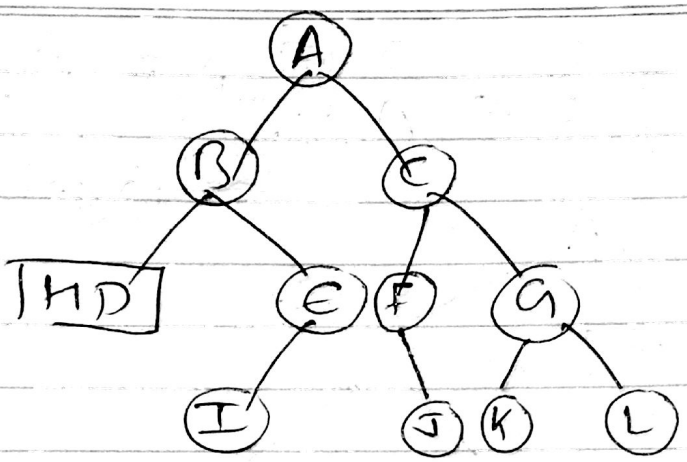
Postorder: H D I E B J F K L G C A
Inorder: H D B I E A F J C K G L

left Right



Postorder: H D I E B J F K L G C A
Inorder: H D B I E A F J C K G L

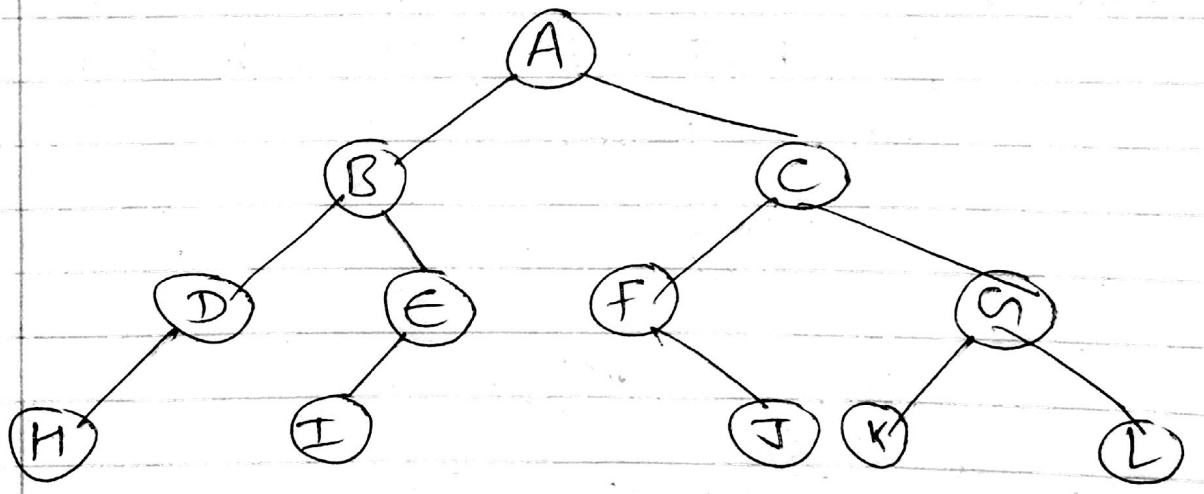
left



Postorder: H D I E B J F K L G C A

Inorder: H D B I E A F J C K G L

left

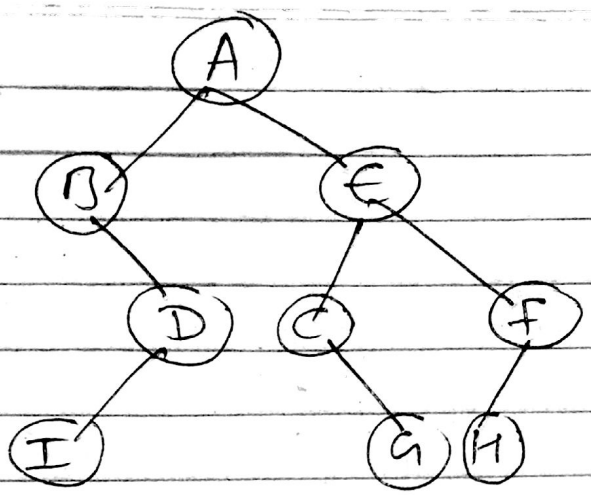


which is final Binary tree.

Construct Binary tree.

Postorder: I D B G C H F E A

Inorder: B I D A C G E H F

Ans → 

#

Polish Notation and Expression tree