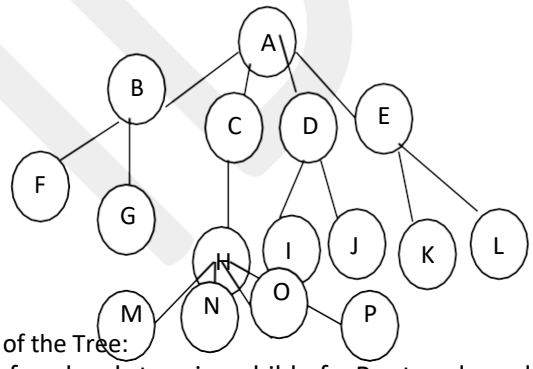
Unit 5: Chapter 10
Trees

Unit Structure

10.0 Trees: A Tree is a collection of nodes. The collection can be empty also. There is a specially designated node called Root Node. The Remaining nodes are partitioned in sub-trees like T1, T2,…Tn. The tree will have unique path from root node to its children or leaf node. The tree does not have cycle.

Dig 1.



H(O)=0; H(A)=3

D(P)==3

H(I)=2; H(E)=1;

D(H)=2; D(K)=2;

10.1 Properties of the Tree:

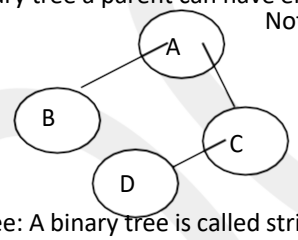
1. The root of each sub-tree is a child of a Root node and root R is the parent of each root of the sub-tree.
2. Every node except the root node has one parent node and all the parent nodes have at least one child. The parent can have one or more children except the leaf node.
3. The node which has no child called leaf node or terminal nodes. A tree can have one or more leaf nodes or terminal nodes.
4. The nodes with the same parent are called siblings.

5. The depth of a node n is the unique path from root node to the node n. The depth of the tree is the unique path from root node to the deepest leaf node.
6. The height of a node n is the unique path from the node n to the root node. The height of the tree is the unique path from deepest leaf node to the root node.
7. The height of the tree must be equal to the depth of the tree.
   Therefore H(T)= D(T). Where H represents the Height, D represents the Depth and T represents the Tree.
8. All the leaves at height zero and the depth of the root is zero.
9. If there is a direct path from n1 to n2 then n1 is an ancestor of n2 and n2 is descendant of n1.
10. A tree should not have multiple paths from node n1 to node n2.

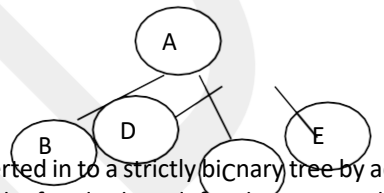
The above tree, height and depth of the tree: 3
Height of root node: 3 ; Depth of all the leaf nodes: 3
Depth of Root node: 0 ; Height of all the leaves : 0

10.2.1 Binary Tree: A tree is called binary tree if all the nodes in the tree has at the most two children. In a binary tree a parent can have either zero child or one child or two children.

Note: All the nodes have at the most two children.



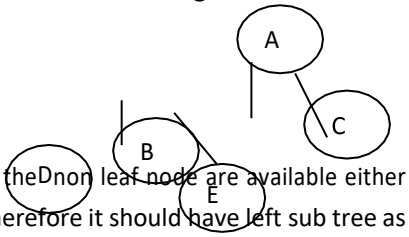
10.2.2 Strictly Binary Tree: A binary tree is called strictly binary tree if every non leaf node in a binary tree has non empty left sub-tree and right sub-tree.



Any binary tree can be converted in to a strictly binary tree by adding left sub-tree or right sub-tree.
In the above tree all the non leaf nodes have left sub tree as well as right sub tree also. A is a non leaf node has left and right sub tree similarly C is a non leaf node has left and right sub tree. Therefore the above tree is a strictly binary tree.

10.2.3 Almost complete Binary Tree: A binary tree is called almost complete binary tree if it satisfys the following two properties:

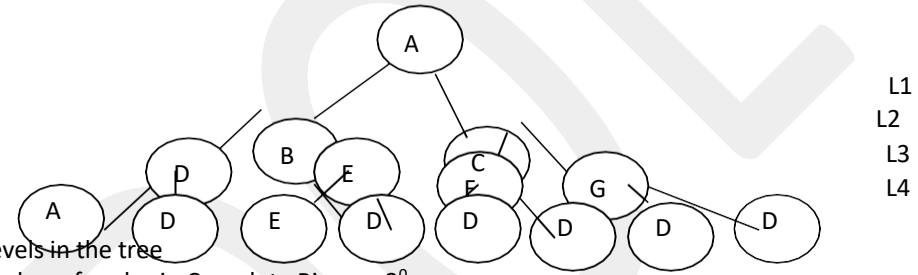
1. All the leaf nodes should be present either at level d or d-1.
2. Any non leaf none if it has right sub-tree then there should be left sub-tree also.



In this tree C, D, E are the non leaf node are available either at depth d or d-1.
B has right sub tree therefore it should have left sub tree as well. The node B has both the sub trees hence it is an almost complete binary tree.

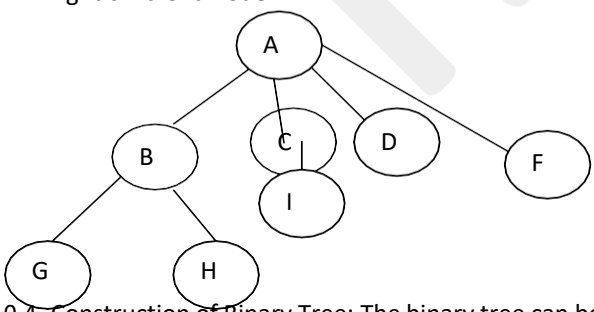
10.2.4 Complete Binary Tree: A binary tree is called complete binary tree if it satisfies the following properties:

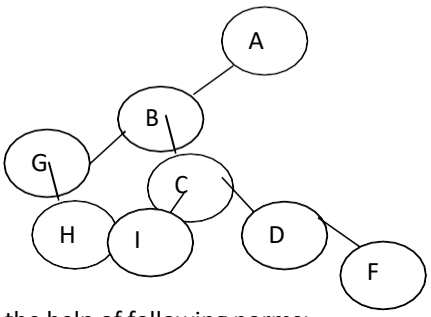
1. Each node in tree must have at the most two children.
2. In a complete binary tree with n node at position i , the left child node must be present at position 2i such that 2i<=N where N is the total number of nodes in a tree and the right child node must be present at position 2i+1 such that 2i<=N where N is the total number of nodes in a tree.
3. The parent of left child node must be present at position i/2 such that i/2<N where N is the total number of nodes in a tree and the parent of right child node must be present at position (i-1)/2 such that (i-1)/2< N where N is the total number of nodes in a tree.
4. The complete binary tree at depth d is a strictly binary tree in which all the leaves are present at depth d.
5. The complete binary tree of level l contains 2^l at any level of a complete binary tree.
6. The total number of nodes in a complete binary tree will be calculated E(l=0 to d) 2^l.



There are 4 levels in the tree
Each level number of nodes in Complete Binary : 2^0
Total Number of nodes at level 0: 2^0 =1
Total Number of nodes at level 1: 2^1 =2
Total Number of nodes at level 2: 2^2 =4
Total Number of nodes at level 2: 2^3 =8
Total Number of nodes in the tree: L0 + L1 + L2 + L3= 1+2+4+8= 15

10.3 Conversion of Tree to Binary tree: A tree can be converted to a binary tree with the help of following statement:
The left child of a node n will remain the left child of a node n and the right sibling will become the right child of a node n.

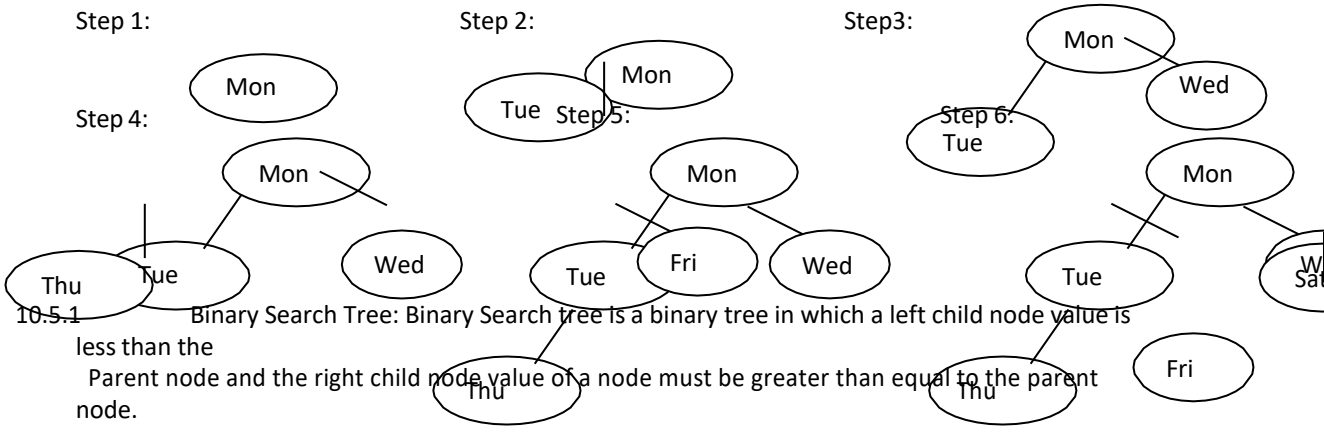




10.4 Construction of Binary Tree: The binary tree can be constructed with the help of following norms:

1. Start with the first value, become the root node.
2. The next value will be added at the last position of the tree.
3. Always first add the left child of the parent then the right child to the parent.

Problem: Construct Binary tree of Mon, Tue, Wed, Thu, Fri, Sat

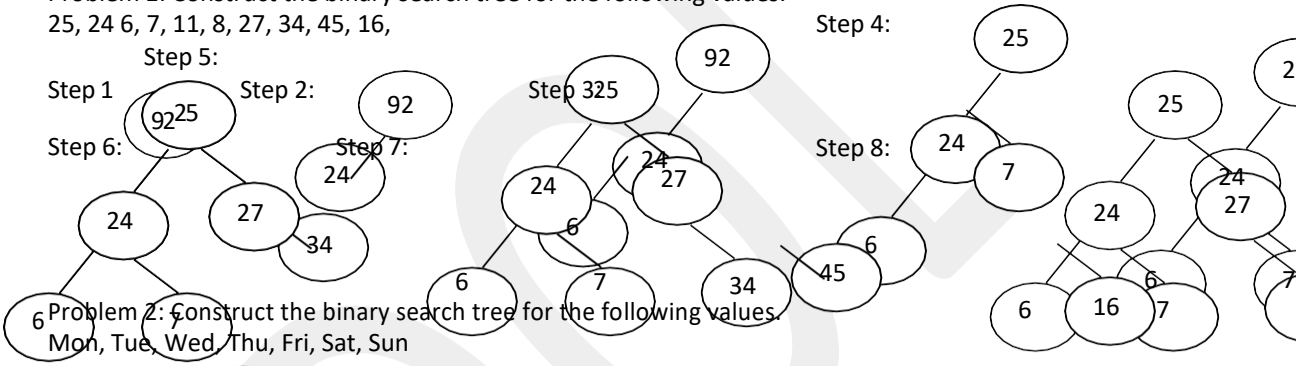


10.5.1 Binary Search Tree: Binary Search tree is a binary tree in which a left child node value is less than the

Parent node and the right child node value of a node must be greater than equal to the parent node.

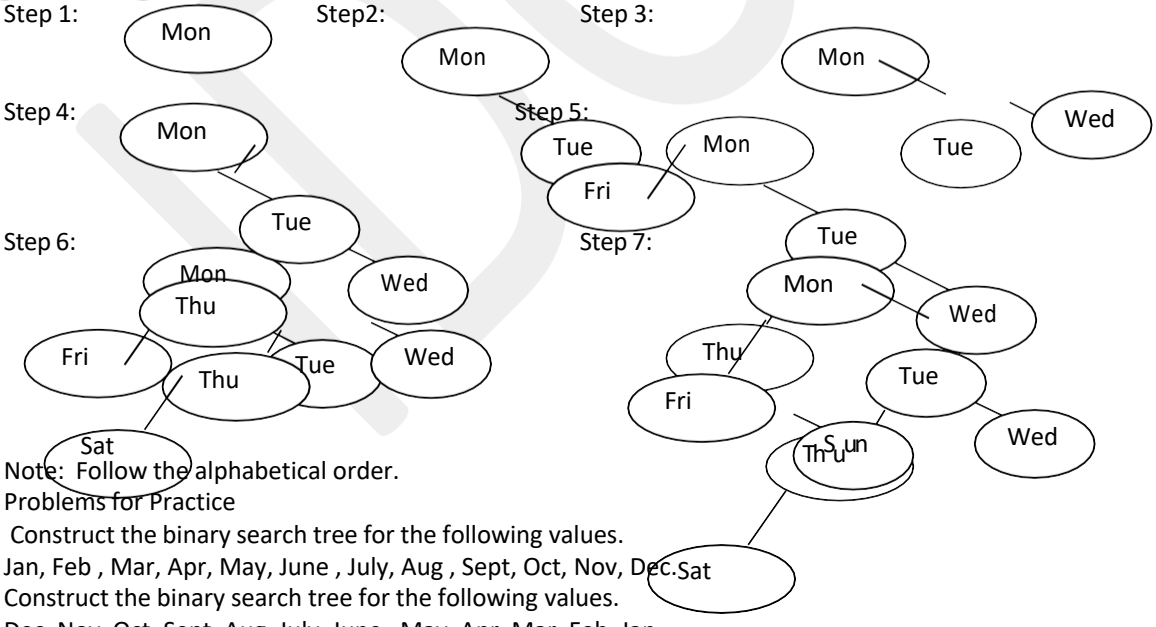
Problem 1: Construct the binary search tree for the following values.

25, 24 6, 7, 11, 8, 27, 34, 45, 16,



Problem 2: Construct the binary search tree for the following values.

Mon, Tue, Wed, Thu, Fri, Sat, Sun



Note: Follow the alphabetical order.

Problems for Practice

Construct the binary search tree for the following values.

Jan, Feb , Mar, Apr, May, June , July, Aug , Sept, Oct, Nov, Dec.

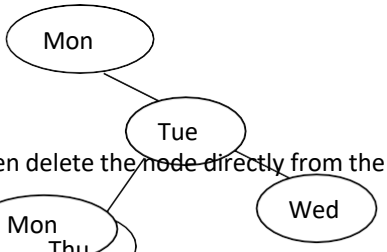
Construct the binary search tree for the following values.

Dec, Nov, Oct, Sept, Aug, July, June , May, Apr, Mar, Feb, Jan

10.5.2 Operations on Binary Search Tree: The following operations can be performed on Binary Search.

How to insert a node in a binary search tree: If the new node value is less than the parent node value then the new node value will be the part of left sub-tree otherwise right sub-tree.

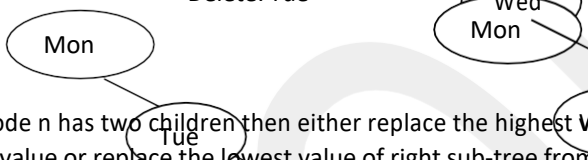
How to delete a node from a binary search tree: Any node can be deleted from the bellow binary search tree.



If the deleted node is a leaf node then delete the node directly from the binary tree.
Delete Thu :

If the node to be deleted has only one child then the child of node n will be connected with the parent of the node n and the node n can be removed.

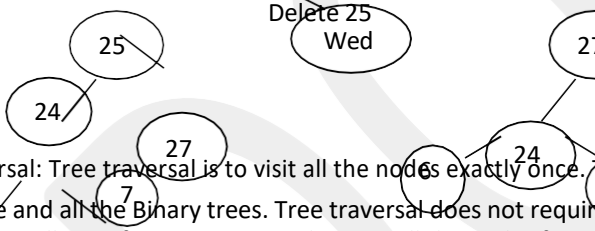
Delete: Tue



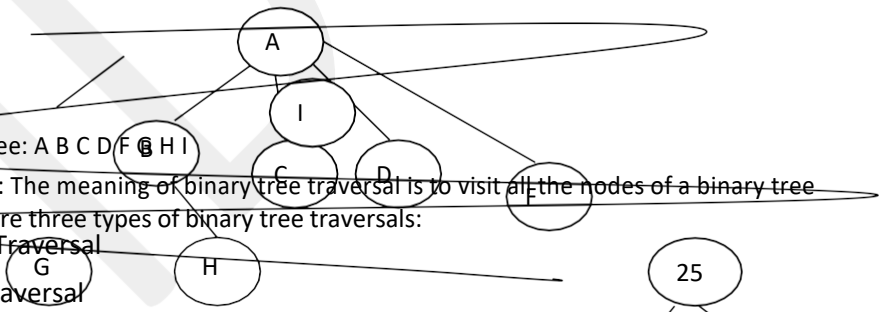
If a deleted node n has two children then either replace the highest value of left sub-tree from the deleted node value or replace the lowest value of right sub-tree from the deleted node value.

Delete 25

Replaced by lowest value from RST



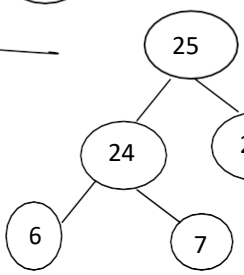
10.6 Tree Traversal: Tree traversal is to visit all the nodes exactly once. Tree traversal is possible only for Simple tree and all the Binary trees. Tree traversal does not require algorithm. In case of Binary tree the traverse will start from root not and covers all the nodes from left to right.



Traversal of above tree: A B C D F G H I

Binary Tree Traversal: The meaning of binary tree traversal is to visit all the nodes of a binary tree exactly once. There are three types of binary tree traversals:

1. Pre-Order Traversal
2. In-Order Traversal
3. Post-Order Traversal



Dig 8

The Algorithms of Binary Tree Traversals:
Pre-Order Traversal

1. Visit the root node.
2. Traverse the left sub-tree in Pre-Order.
3. Traverse the Right sub-tree in Pre-Order.

Pre Order Traversal of tree in dig 8: 25, 24, 6, 7, 27

In-Order Traversal

1. Traverse the left sub-tree in In-Order.
2. Visit the root node.
3. Traverse the Right sub-tree in In-Order.

In Order Traversal of tree in dig 8: 6, 24, 7, 25, 27

Post-Order Traversal

1. Traverse the left sub-tree in In-Order.
2. Traverse the right sub-tree in In-Order.
3. Visit the root node.

Post Order Traversal of tree in dig 8: 6, 7, 24 , 27, 25

Problems for Practice:

Problem 1: Construct and Traverse the binary tree in Pre-Order, In-Order and Post-Order .

92, 24 6,7,11,8,22,4,5,16,19,20,78

Problem 2: Construct and Traverse the binary tree in Pre-Order, In-Order and Post-Order .

Mon, Tue, Wed, Thu, Fri, Sat, Sun

Problem 3: Construct and Traverse the binary tree in Pre-Order, In-Order and Post-Order .

Jan, Feb , Mar, Apr, May, June , July, Aug , Sept, Oct, Nov, Dec.

Problem 4: Construct and Traverse the binary tree in Pre-Order, In-Order and Post-Order .

Dec, Nov, Oct, Sept, Aug, July, June , May, Apr, Mar, Feb, Jan

10.7 Construction of Binary Tree from the Traversal of the tree:

If any of the traversal pre-order or post-order is given the tree can be constructed using following method:

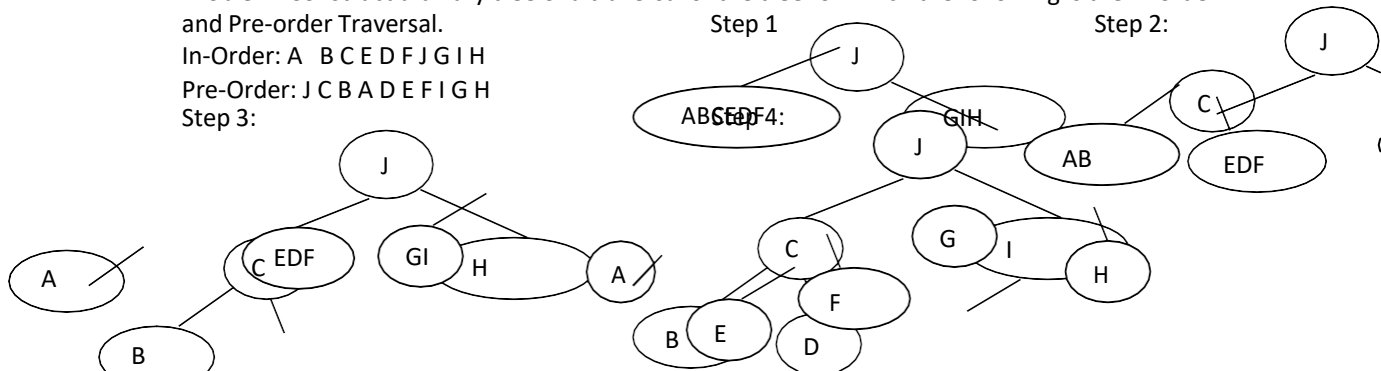
Pre- Oder and In-Order

1. The first value of pre-order traversal becomes the root node value.
2. All the values lying left to the same value in in-order will be the part of left sub-tree and the values which are right to the in-order of the same value will be part of right sub-tree.

Problem: Construct a binary tree of a traversal of the tree for which the following is the in-order and Pre-order Traversal.

In-Order: A B C E D F J G I H

Pre-Order: J C B A D E F I G H

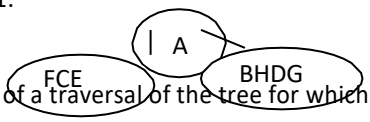


Problem: Construct a binary tree of a traversal of the tree for which the following is the in-order and Post-order Traversal.

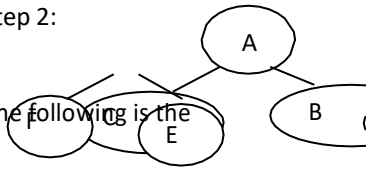
Post-Order: F E C H G D B A

In-Order: F C E A B H D G

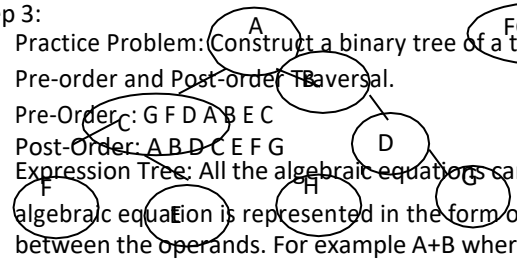
Step 1:



Step 2:



Step 3:



Practice Problem: Construct a binary tree of a traversal of the tree for which the following is the Pre-order and Post-order Traversal.

Pre-Order: G F D A B E C

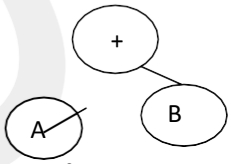
Post-Order: A B D C E F G

10.8 Expression Tree: All the algebraic equations can be represented in the form of binary tree. An algebraic equation is represented in the form of infix notation in which the operator is coming between the operands. For example A+B where A and B are the operands and + is an operator which is coming between operands A and B. While representing an algebraic equation in the form of binary tree, the entire operator will be either root node or internal nodes and all the operands will be the leaf nodes.

Expression tree satisfy following properties:

1. It represents an algebraic equation in the form of binary tree.
2. All the operators will be either root node or an internal node and all the operands will always be the leaf nodes.
3. Expression is an infix notation of a binary tree.
4. If you traverse the expression tree in an in-order then it is converted in to an algebraic equation.

Algebraic equation A+B can be represented as



Problem:

How to construct an expression Tree:

Step1: Convert the algebraic equation either in to pre fix notation or postfix notation.

Step 2: By Prefix / Postfix notation identify the root.

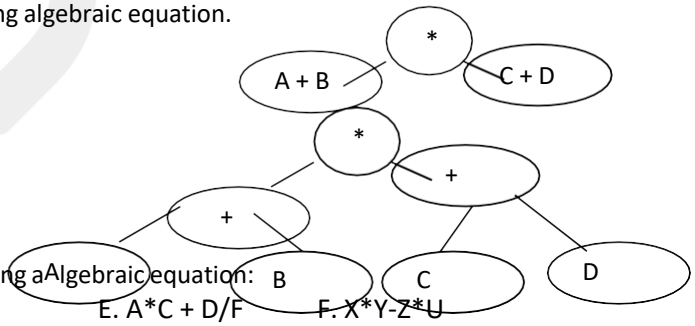
Step 3: All the values which comes left to prefix value in infix notation will be part of left sub tree and the values come to right to the prefix value in infix notation will be the part of right sub tree.

Construct an expression tree of the following algebraic equation.

A + B * C+ D

Infix Notation: A + B * C+ D

Prefix Notation: *+AB+CD



Construct an expression Tree for the following algebraic equation:

A. A+B*C B. (A+B)*(C-D) E. A*C + D/F F. X*Y-Z*U

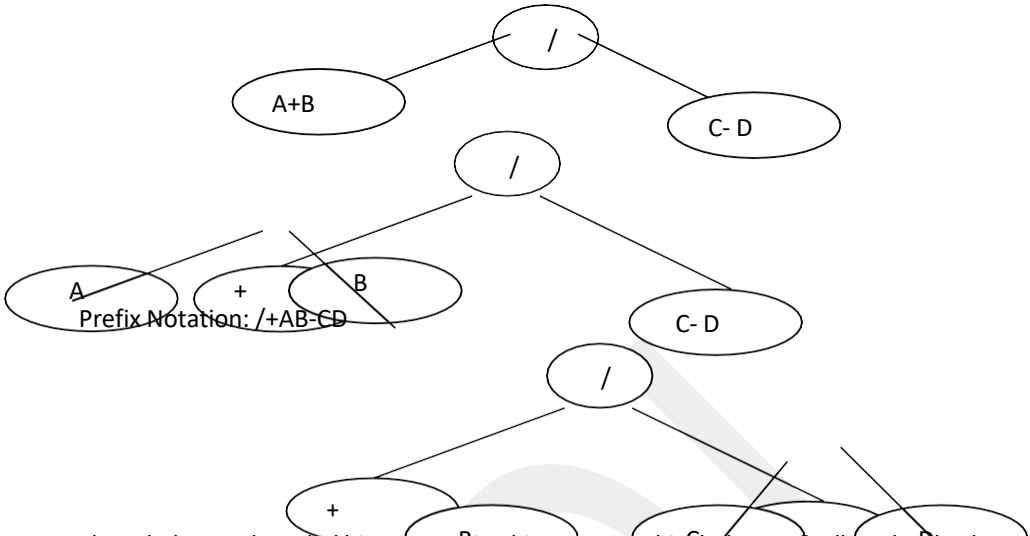
Construct an expression Tree for the following algebraic equation:

A. 2+P+Q*C B. (A-B)/(C+D) E. A*C * D/F F. P/Q+Z-U

Convert the expression in prefix notation and post fix notation and then construct the tree.

A. A/B*C B. (A+B*C)/(E-D) E. A*C * D/F F. X/Y-Z*U

Example: Infix Notation: (A+B)/(C-D)



Prefix Notation: /+AB-CD

10.9 Threaded tree: Threaded binary tree is a binary tree, which does not allow the back tracking method. Threaded Binary Tree is a solution of traversing the tree without using back tracking method.

If a binary search tree is constructed and traverse the same tree in in-order then to visit the next node some it passes through a previous node value. Threaded binary tree is a solution of back tracking method in which while traversing the tree in in-order, it will not pass through the previous node value.

Process: Traverse the binary tree in in-order and mark all the successor node of all the leaf node of in-order traversal tree of the same tree. These leaf nodes will be connected to their successor node value by a thread which is redirecting the leaf node value to connect with successor node and avoid going to previous node value.

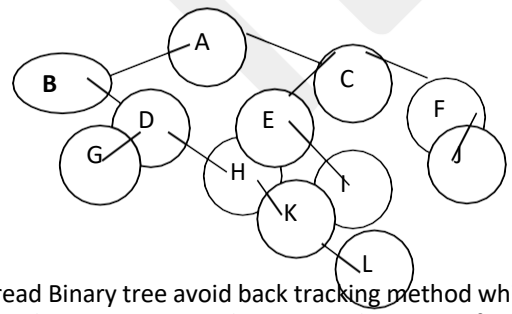
In-order: B G D H K L A E I C J F

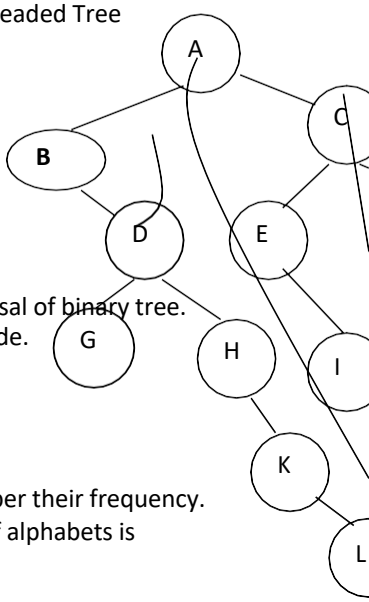
Leaf Nodes: G L I J

Successor Leaf nodes : G-> D; L->A I-> C J->F

Therefore G will be connected with D by a threaded. Similarly L by A , I by C and J by F.

Binary Threaded Tree





Thread Binary tree avoid back tracking method which is a drawback of traversal of binary tree. Threads are proving an alternate path to move from one node to another node.

10.10 Huffman tree: Huffman Tree is a representation of alphabets or numbers as per their frequency. Many times we see an alphabet is repeated multiple times or combination of alphabets is

repeated in the same combination. Huffman coding is a lossless data compression technique. In this all the alphabets are assigned variable length unique code by forming the tree. This tree is called Huffman Tree.

How to construct a Huffman Tree:

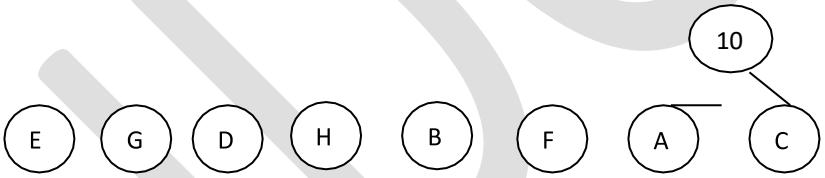
1. Arrange all the alphabets in ascending or descending order and place them a leaf node of a tree.
2. Add two lowest frequency nodes and form a new node which will be one level up. The value of this new node will be the addition of both the nodes which are added.
3. Keep adding two lowest frequency nodes which are not yet added.
4. Repeat step number three un till a root is formed.
5. Assign 0 to all the left branch of the tree and 1 to all the right branch of the tree.
6. A code of each alphabet is identified by reading the branch values from top to bottom which is a unique path from root to the leaf node.

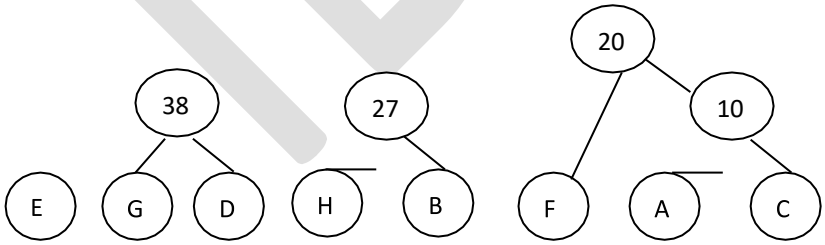
Problem: Construct Huffman Tree for the following data values:

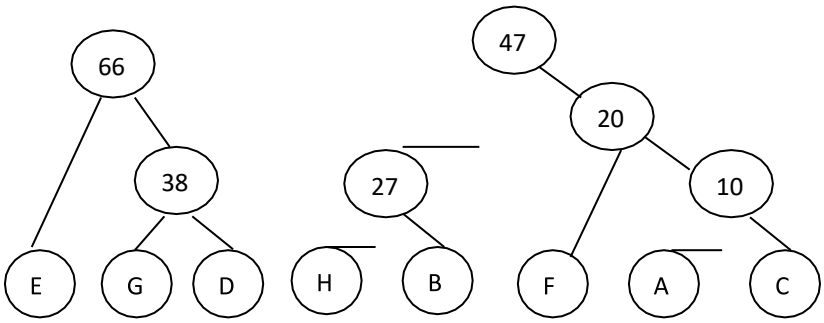
Symbol	A	B	C	D	E	F	G	H
Frequency	6	13	4	18	28	10	20	14

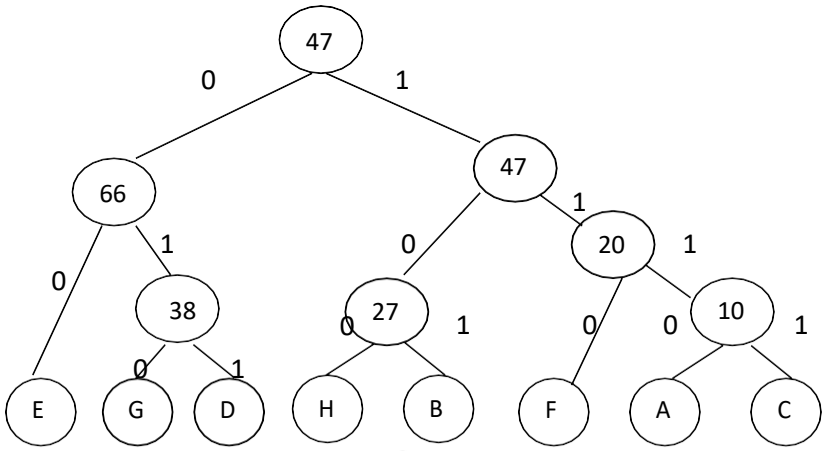
Arrange all the alphabets in descending order as per their frequency:

E	G	D	H	B	F	A	C
28	20	18	14	13	10	6	4









Huffman Code:

E: 00	D: 011	B: 101	A: 1110
G: 010	H:100	F: 110	C:1111

10.11 AVL tree: AVL Tree is identified by Adelson Velskii and Landis. That is why it is called as AVL Tree.

AVL tree is a solution of a binary search tree which is constructed for either all the values of increasing order or decreasing order. It has following problems:

1. When a binary search tree is constructed of ascending or descending values it will go one side either left hand side or right hand side.
2. The tree will not look like a tree at the same time and it will not be balanced.

Therefore AVL Tree is a solution to overcome from both the above problems.

AVL Tree is a binary search tree in which the difference between the height of left sub tree and the height of right sub tree must be less than equal to one. The condition can be represented by following equation.

$$H_L \sim H_R <= 1$$

H_L is Height of Left sub tree
H_R is Height of Right sub-tree.
Since the tree will be balanced therefore it is called as Height Balance tree or Balanced Binary Search Tree.
AVL Tree must satisfy the following properties:

1. The difference between the height of left sub tree and the height of right sub tree must be less than equal to one.
2. A new node can be added to the AVL tree only if the tree is balanced.
3. All the nodes must have balance factor either 0 or 1.

How to Balance Binary search tree: Use rotation by finding the unbalancing is due to which sub tree.

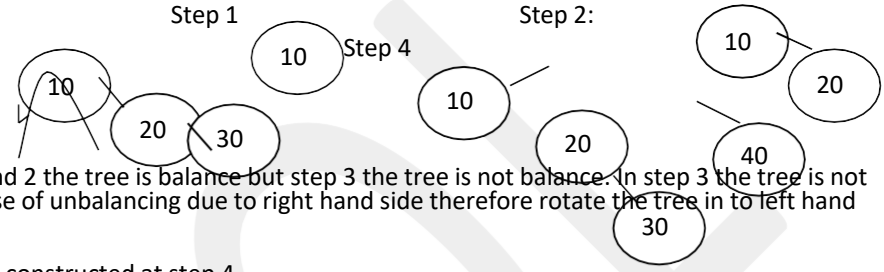
1. If the tree is unbalance due to the height of left sub tree then rotate the tree in to right hand side. Dig

2. If the tree is unbalance due to the height of right sub tree then rotate the tree in to left hand side. Dig
3. If the tree is unbalance due to the right sub tree of left sub tree then rotate the tree first left then right hand side. [In this case there will be multiple rotations to balance the tree.]
4. If the tree is unbalance due to the left sub tree of right sub tree then rotate the tree first right then left hand side. [In this case there will be multiple rotations to balance the tree.]
5. Dig 14.1

Construct AVL Tree for the following values:

10 20 30 40 Step 1 Step 2:

Step 3: Step 4



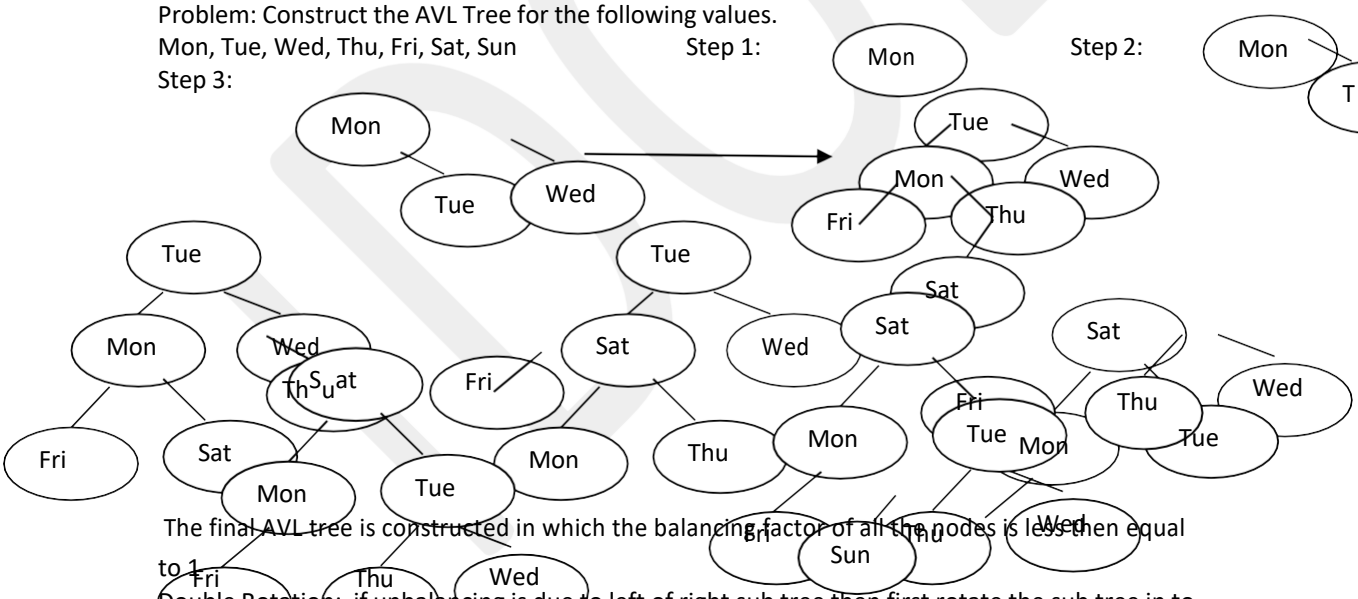
Note: Step 1 and 2 the tree is balance but step 3 the tree is not balance. In step 3 the tree is not balance because of unbalancing due to right hand side therefore rotate the tree in to left hand side.

The AVL Tree is constructed at step 4.

Problem: Construct the AVL Tree for the following values.

Mon, Tue, Wed, Thu, Fri, Sat, Sun Step 1: Step 2:

Step 3:



The final AVL tree is constructed in which the balancing factor of all the nodes is less then equal to 1.

Double Rotation: if unbalancing is due to left of right sub tree then first rotate the sub tree in to right side than rotate the sub tree in to left side and if unbalancing is due to right of left sub tree then first rotate the sub tree in to left side than rotate the sub tree in to right.

Practice Problems:

- Construct and AVL Tree for the following values.

  92, 24 6,7,11,8,22,4,5,16,19,20,78
- Construct and AVL Tree for the following values.

  Sun, Mon, Tue, Wed, Thu, Fri, Sat
- Construct and AVL Tree for the following values.

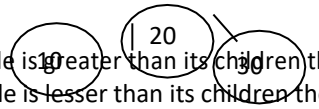
  Jan, Feb , Mar, Apr, May, June , July, Aug , Sept, Oct, Nov, Dec.

- Construct and AVL Tree for the following values.

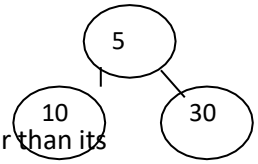
  Dec, Nov, Oct, Sept, Aug, July, June , May, Apr, Mar, Feb, Jan

10.12.1 Heap: Heap Tree is binary tree in which parent node is either greater than its children or lesser than its children. There are two types of Heap Tree.

Max Heap



Min Heap



If the parent node is greater than its children then it is called as Max Heap.
If the parent node is lesser than its children then it is called as Min Heap.

1. Max Heap: Max Heap Tree is binary tree in which parent node is greater than its children. Using max heap tree the values will be coming in descending order if the same heap tree is used for shorting.
2. Min Heap: Min Heap Tree is binary tree in which parent node is greater than its children. Using min heap tree the values will be coming in ascending order if the same heap tree is used for shorting.

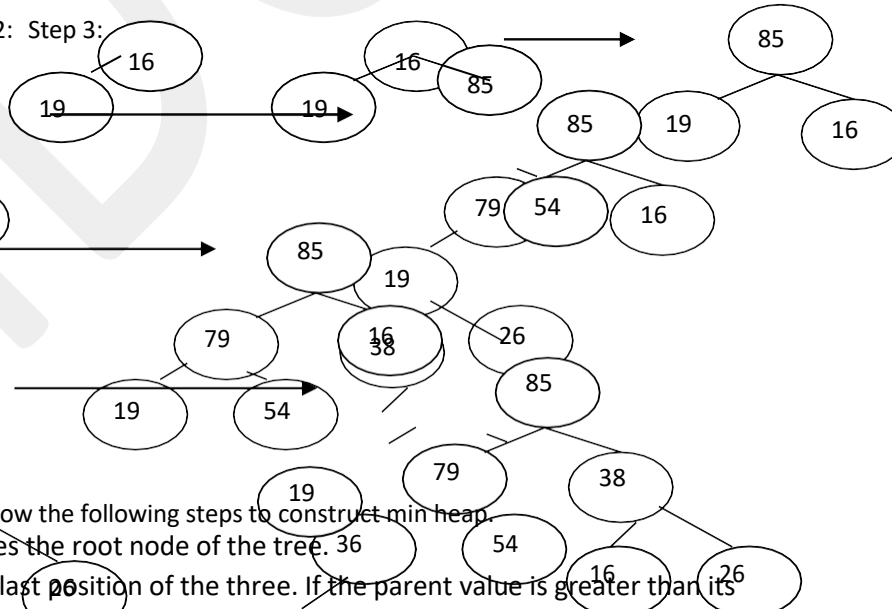
10.12.2 How to construct a Heap Tree

Steps to construct Max Heap: Follow the following steps to construct max heap.

1. The first value becomes the root node of the tree.
2. Add next value at the last position of the three. If the parent value is lesser than its children the replace the parent node value to the greater child value and the tree is converted in to heap.
3. Repeat the step number 2 until all the values are added in to the heap tree.

Construct Max Heap for the following values:
16, 19, 85, 79, 54, 38, 26, 36

Step 1: Step 2: Step 3:



Step 4:

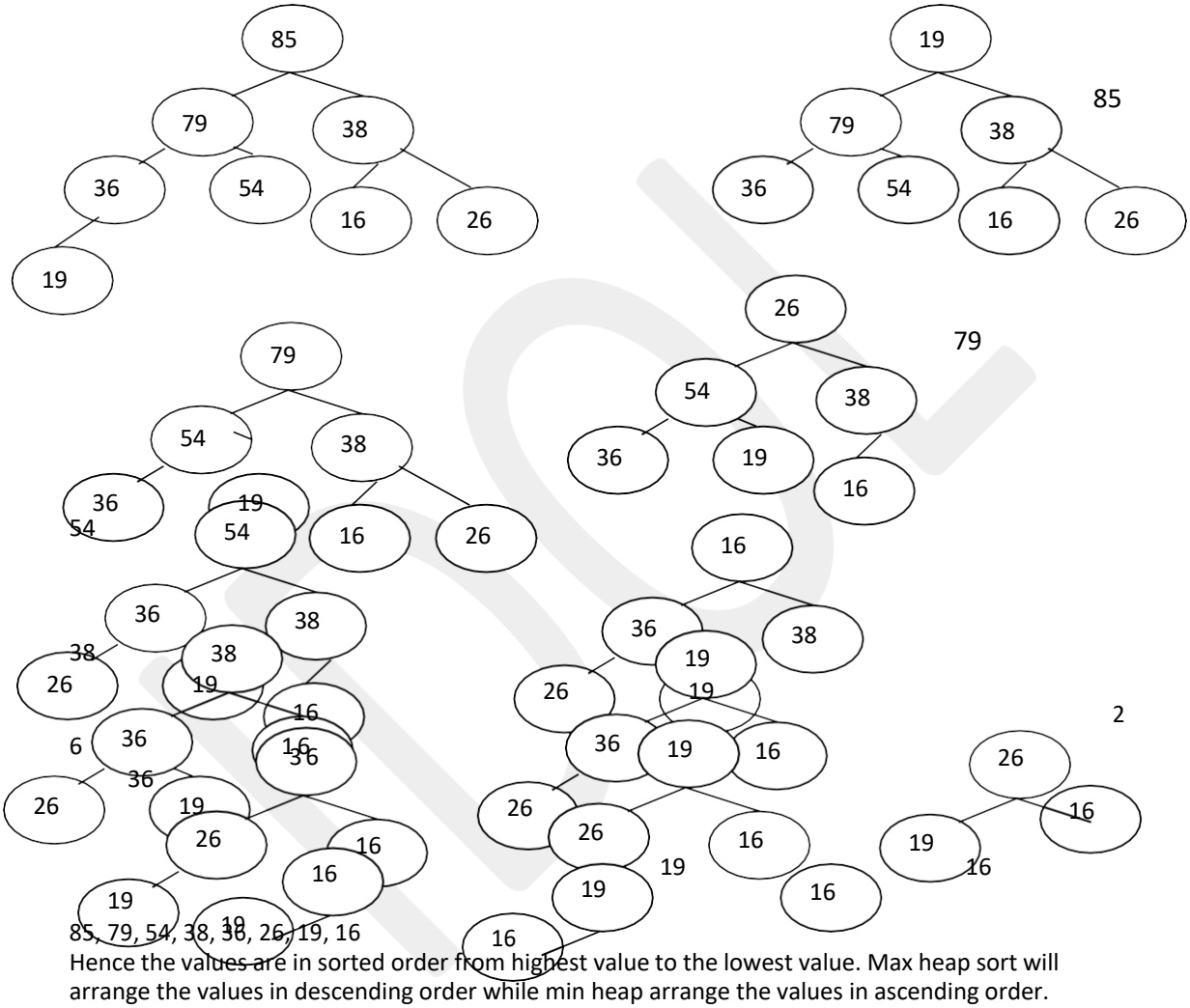
Steps to construct Min Heap: Follow the following steps to construct min heap.

1. The first value becomes the root node of the tree.
2. Add next value at the last position of the three. If the parent value is greater than its children the replace the parent node value to the lowest child value and the tree is converted in to heap.
3. Repeat the step number 2 until all the values are added in to the heap tree.

The way max heap is constructed; min heap can also be constructed similarly.

10.12.3 Steps to apply Heap Short:

1. Convert the tree in to heap.
2. Take out the value from the root.
3. Replace last node value from the root node value.
4. Go to step 1 until number of nodes are greater than one.



85, 79, 54, 38, 36, 26, 19, 16

Hence the values are in sorted order from highest value to the lowest value. Max heap sort will arrange the values in descending order while min heap arrange the values in ascending order.

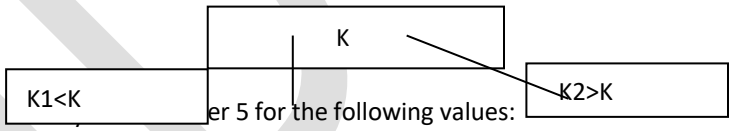
Reheap up and Reheap down: The concept of reheap up and reheap down is how the values are shifting upward direction or downward direction while constructing the heap tree. If the values are shifting upward direction then it is call Reheapup and if the values are shifting downward direction then it is call reheap down.

Unit 5: Chapter 11

M- Way Tree

11.1.1 M-Way Tree: M-way tree is a multi valued tree in which a node consists of multiple values and satisfy the following properties.

1. M-Way tree of order m will have maximum m pointers and m-1 key values in a node. All the keys must be placed in an increasing order or ascending order.
2. If a node has m-1 key values then the node is full. If a node is full then the new key value will be either left child value of the parent node if it is lesser then its parent node value or right pointer child value if it is greater than its parent value.
3. A new key value will be added at the leaf node value.
4. The leaves may not be at the same level.

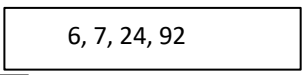


11.1.2 Construction of er 5 for the following values:

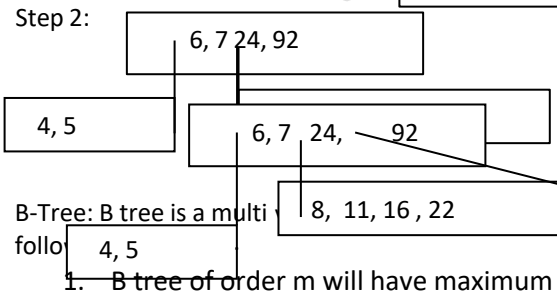
92, 24 6,7,11,8,22,4,5,16,29,40,78

Maximum Number of key values in a node: (m-1) = 5-1= 4

Step 1:



Step 2:



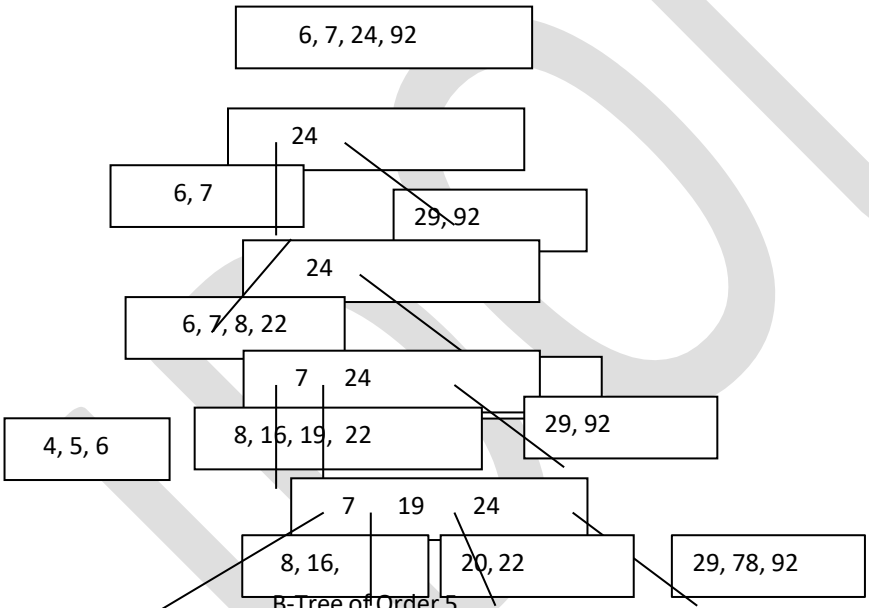
11.2.1 B-Tree: B tree is a multi v or satisfy the follo

1. B tree of order m will have maximum m pointers and m-1 key values in a node. All the keys must be placed in an increasing order or ascending order.
2. If a node has m-1 key values then the node is full. If a node is full then split the node. Select the median value which becomes the parent value. The values

coming left of median value will become the left child of parent value and the values coming to right to the median value will become the right child of the median value. The new key value will be inserted either left child of the parent node (if it is lesser then its parent node value) or right child value of the parent node (if it is greater than its parent value).

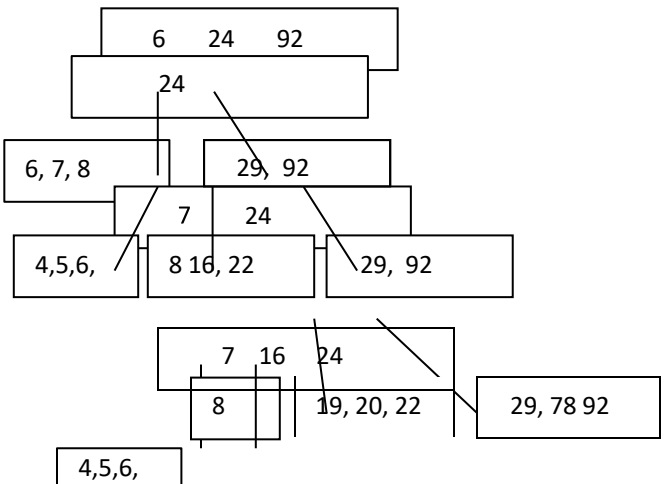
3. A new key value will be added at the leaf node value.
4. The leaves must be at the same level.
5. The height of B-tree will always be lesser than M-Way Tree.
6. The new value will be inserted in the leaf node.

11.2.2 Construction of B Tree: Construct B- tree of order 5 for the following values:
92, 24 6,7,29 ,8,22,4,5,16,19,20,78
Maximum Number of Key values in a node= m-1=5-1=4



B-Tree of Order 5

4, 5, 6 nstruct B- tree of order 4 for the following values:
,8,22,4,5,16,19,20,78
Maximum Number of Key values in a node= m-1=4-1=3

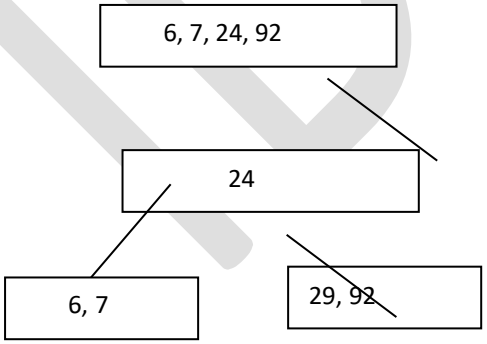


B Tree of Order 4

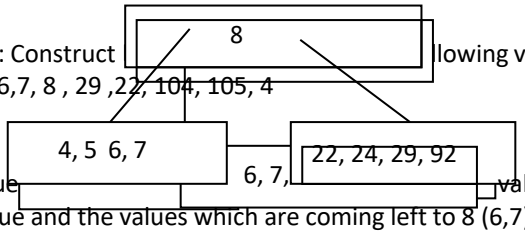
11.3.1 B*-Tree: B* tree is a multi valued tree in which a node consists of multiple values and satisfy the following properties.

1. B* tree of order m will have maximum m pointers and m-1 key values in a node. All the keys must be placed in an increasing order or ascending order.
2. If a node has m-1 key values then the node is full. If a node is full then split the node only if all the siblings are full. If all the siblings are full then only select the median value which becomes the parent value. The values coming left of median value will become the left child of parent value and the values coming to right to the median value will become the right child of the median value. The new key value will be inserted either left child of the parent node (if it is lesser then its parent node value) or right child value of the parent node (if it is greater than its parent value). If siblings are not full the rotate the values of the leaf node and make the place empty for the new key value.
3. A new key value will be added at the leaf node value.
4. The leaves must be at the same level.
5. The height of B*-tree will always be lesser than B Tree.
6. B* tree is referred for classification of topic. B* tree is also referred for the purpose of indexing.
7. The new value will be inserted in the leaf node.

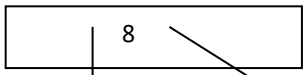
11.3.2 Construction of B* Tree: Construct B*- tree of order 5 for the following values:
92, 24 6,7,29 ,8,22,4,5
Maximum Number of Key values in a node= m-1=5-1=4



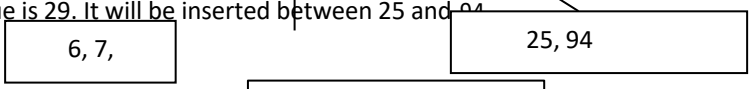
Problem: Construct [illegible] lowing values:
94, 25 6,7, 8 , 29 ,22, 104, 105, 4

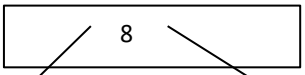


Next value [illegible] values 6, 7, 8, 24, 92. 8 will become the root node value and the values which are coming left to 8 (6,7) will become the left child of root node and the values which are coming right to 8 (24, 82) will become the right child of median value.

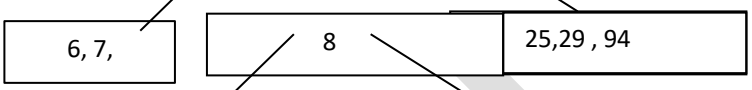


Next value is 29. It will be inserted between 25 and 94

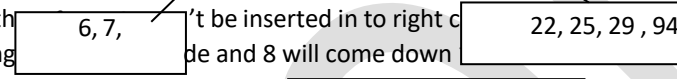




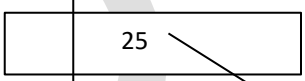
Next value is 22. It will be inserted before 25 as the node can consist of maximum 4 key values.



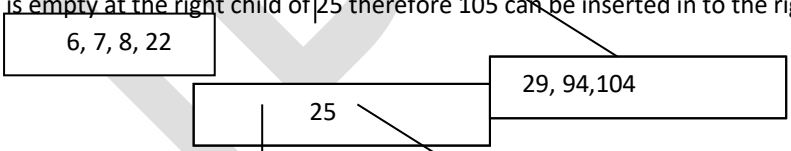
Next value is 104. It will be inserted after 94 as the node can consist of maximum 4 key values But the node is full th ’t be inserted in to right c can be inserted after rotating de and 8 will come down



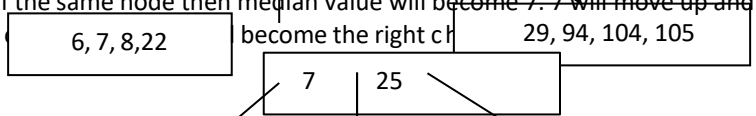
Once the key values ace is empty at rig
Next value is 105. It 4 as the node can ut the node is full therefore 105 can’t be inserted in to right child of 8. Hence new value can be inserted after rotating 22 as a root node and 8 will come down to the left hand side.



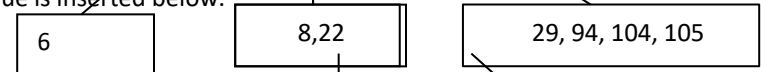
Now the place is empty at the right child of 25 therefore 105 can be inserted in to the right child of 25.

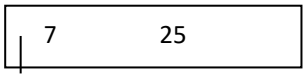


Next value is 4 but both the siblings are full therefore break the left node of 25. Since 4 is the left most value of the same node then median value will become 7. 7 will move up and 4,6 will become the left child become the right c



Here the new value is inserted below.



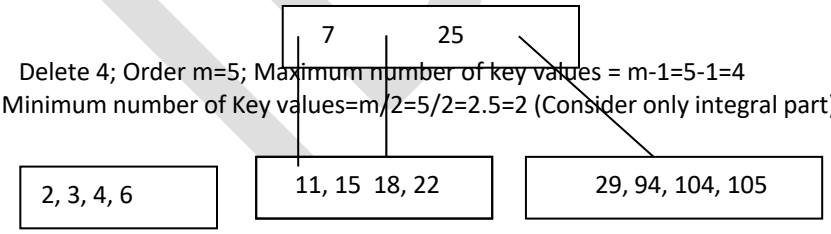


11.4 Deletion from B Tree:

Deletion from B tree is the deleting the key values from a node. Below are the rules to delete the key values from the node.

1. The key values are deleted from the leaf node only.
2. At a time only one key value is deleted.
3. The key value can't be deleted from root node and internal node.
4. Key value can't be deleted if the number of key values is less the m/2 key values.
5. If the key value which is available at leaf node is supposed to be deleted and the node have more than m/2 key values than delete the key value from the leaf node directly.
6. If the key value which is available at leaf node is supposed to be deleted and the node have m/2 key values or less than m/2 key values then merge the leaf node siblings and then delete the key value from the leaf node directly.
7. If the key value which is not available at leaf node is supposed to be deleted and the node have more than m/2 key values then move key value at the leaf node and then delete the key value from the leaf node directly.

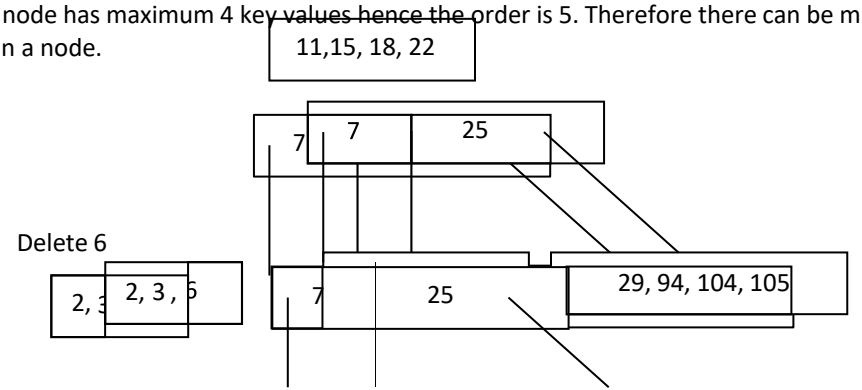
Delete the following key values from the below B-Tree.



Delete 4; Order m=5; Maximum number of key values = m-1=5-1=4
Minimum number of Key values=m/2=5/2=2.5=2 (Consider only integral part)

Four is deleted from the leaf node directly since it was available at the leaf node and the node has more than m/2 Key values.

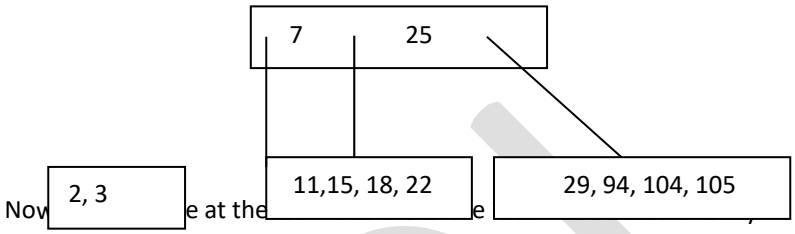
Since a node has maximum 4 key values hence the order is 5. Therefore there can be minimum 2 key values in a node.



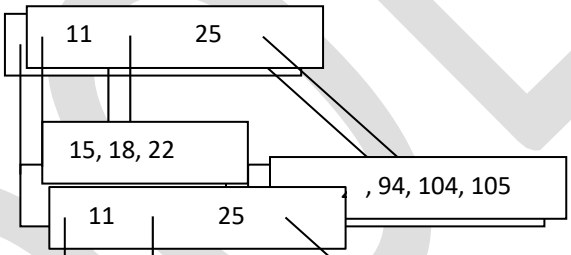
6 is deleted from the leaf node since it was available at the leaf node and number of key values in a node was more than m/2.

Delete 7

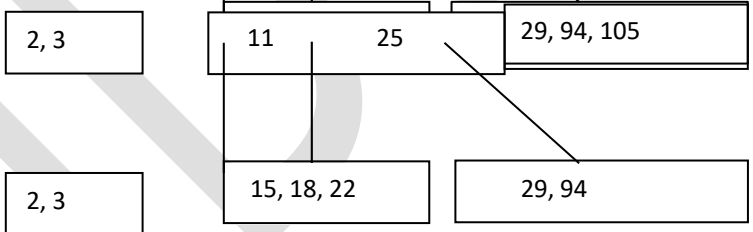
7 can't be deleted directly since it is not available at the leaf node therefore rotate the key values.



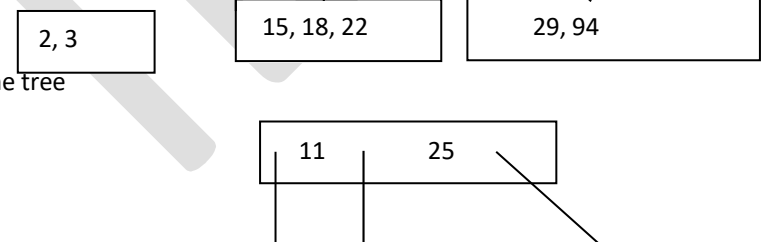
Now e at the e ,



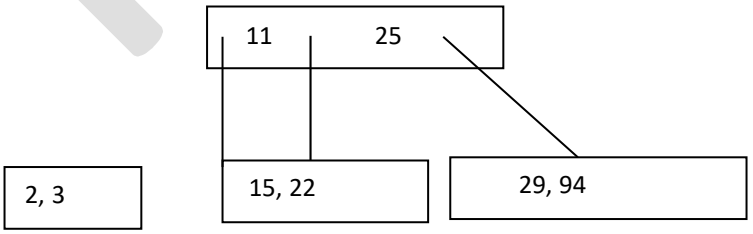
Delete 104 from the tree



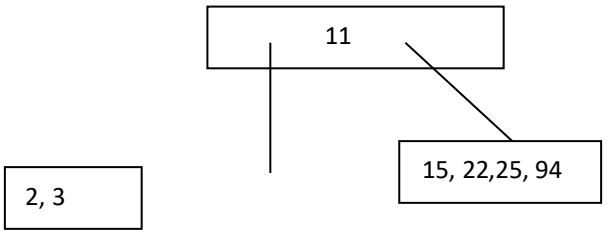
Delete 105 from the tree



Delete 18 from the tree

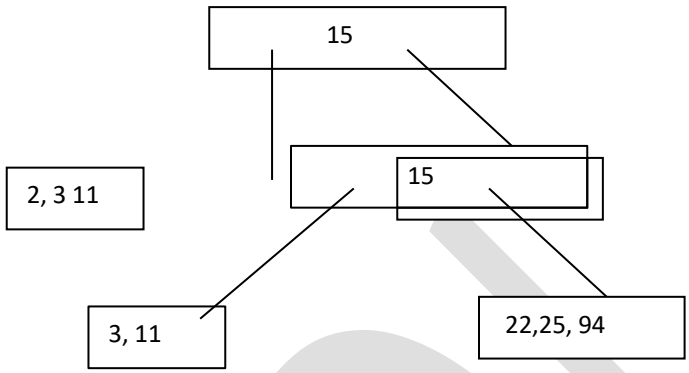


Delete 29 from the tree

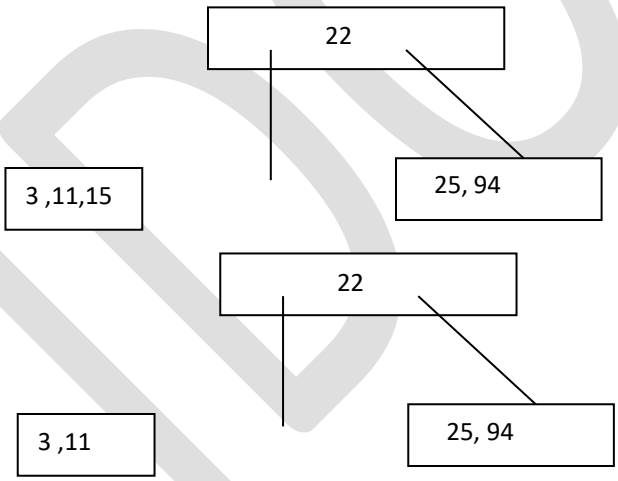


Since the node containing the value 29, will have less than m/2 key values therefore both the siblings will be merged.

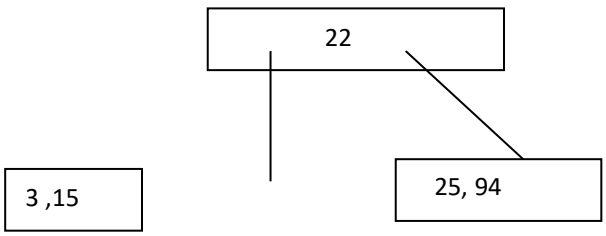
Delete 2 from the tree: First rotate the key values in which 11 will move down and 15 will move up.



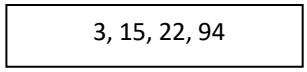
Delete 15 from the tree: 22 will move up and 15 will move down than 15 will be deleted from the leaf.



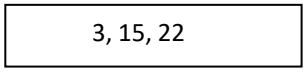
Delete 11 from the tree



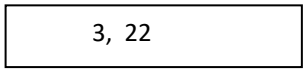
Delete 25 from the Tree: 25 is available at the leaf node but both siblings has m/2 key values therefore merger will take place.



Delete 94 from the Tree:



Delete 15 from the Tree:



Further values from the tree can't be deleted since the node is containing m/2 key values and neither rotation is possible nor merger is possible.

11.5 Similarities in B-Tree and B* Tree

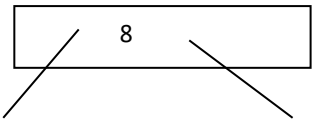
- Both have maximum m-1 key values in a node if the order of the tree is m.
- Both the trees can have maximum m pointers in a node if the order of the tree is m.
- Both the trees the key values in a node must be in increasing order.
- Both the trees leaves must be at the same level.
- At the time of deletion the rotation is possible in both the trees.

Difference in B-Tree and B* Tree

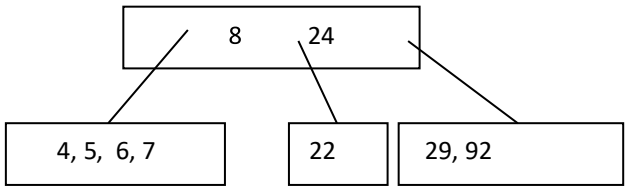
- The height of B*-Tree is either lesser or equal to the B-Tree for same number of key values and for the same order of the tree.
- In case of B* Tree, a node will have more number of key values compare to B Tree.
- B Tree, rotation of key values is not allowed while in B* Tree rotation of key values is allowed at the time of inserting the key values.
- We do not split the node in B*-Tree until sibling are full while B tree, we split the node once the node is full.

11.6 Practice Problems on B Tree and B* Tree

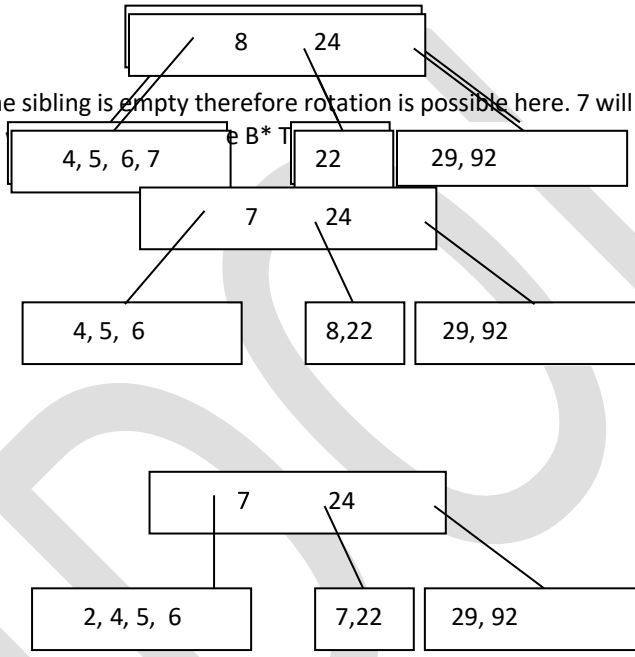
Problem: Insert 107 in the following B-tree of order 5.



Solution: The value 107 can't be inserted directly in to the B tree of order 5 and a node can contain maximum 4 values in this case the node. Therefore split the right child node.



Problem: Insert 2 in the following B* Tree of Order 5.



Solution: Since the sibling is empty therefore rotation is possible here. 7 will move up and 8 will come down and then 2 [illegible] e B* T