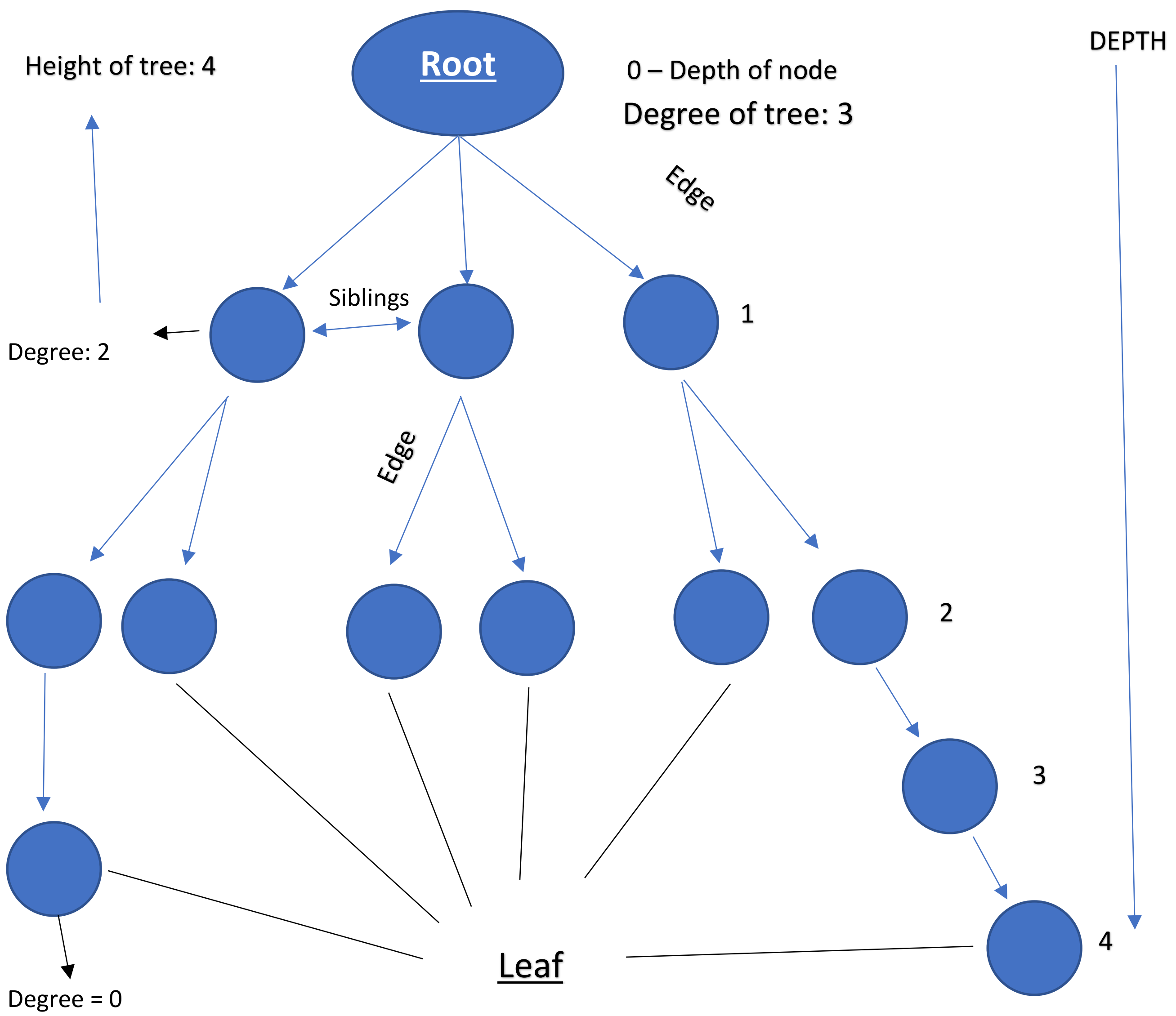
TREE

- Combition of nodes through edges(used to represent Hierarchical data)



Root: Node which do not have any parent node.

Parent: Any node which have a node below is known as parent.

Child: Any node which have a node above is known as child.

Siblings: Nodes with same parent.

Leaf/ External Node: Any node which do not have any child is known as LEAF or External Node

Internal Node: Any node which have a child is known as Internal node

Depth: No. of edges above a node is known as its depth.

Height: No. of edges below a node is known as its height.

***** Height of tree: No. of edges in longest branch of tree. *****

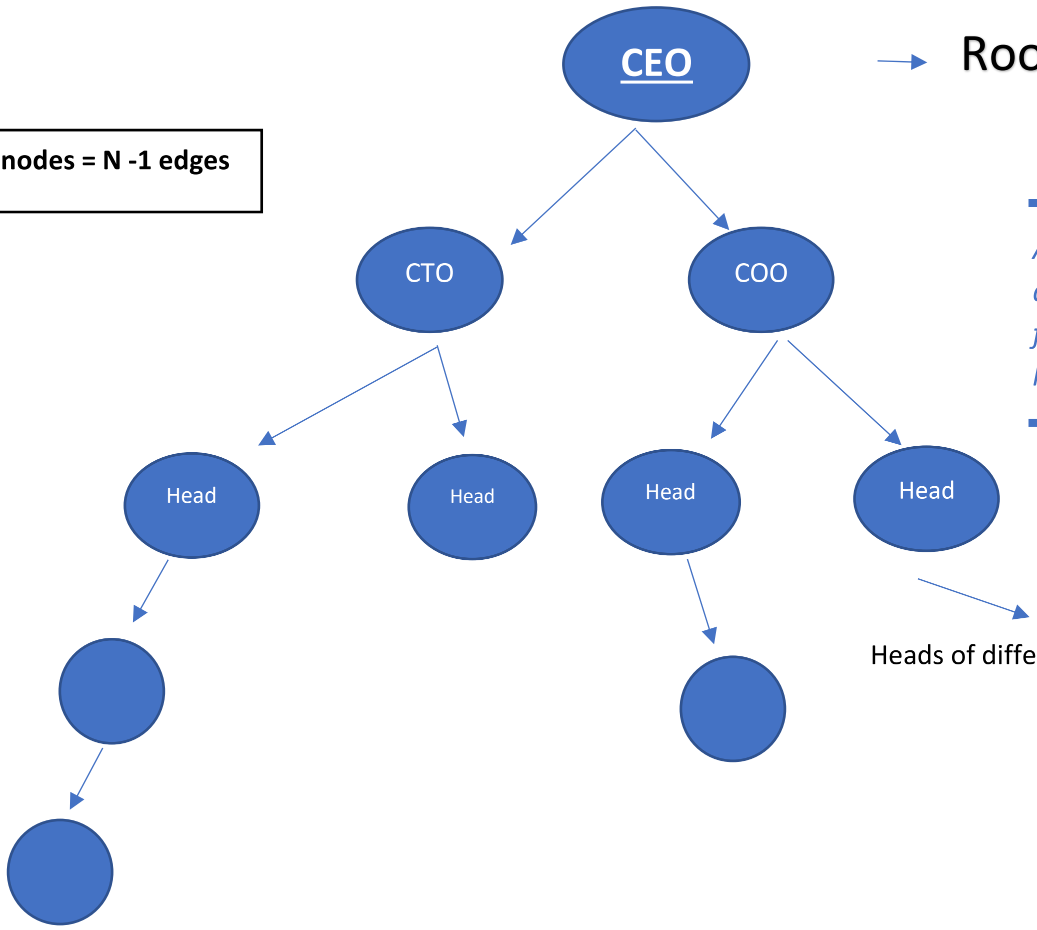
Ancestors: nodes before a node are called its ancestors.

Descendants: Nodes after a node are called its Descendants.

Degree: No. of Childs a node have is known as its degree.

***** Degree of tree: The highest degree of all nodes is known as degree of tree *****

Binary Tree

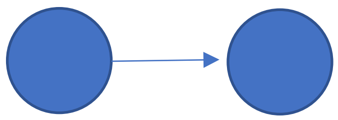


Root(Example)

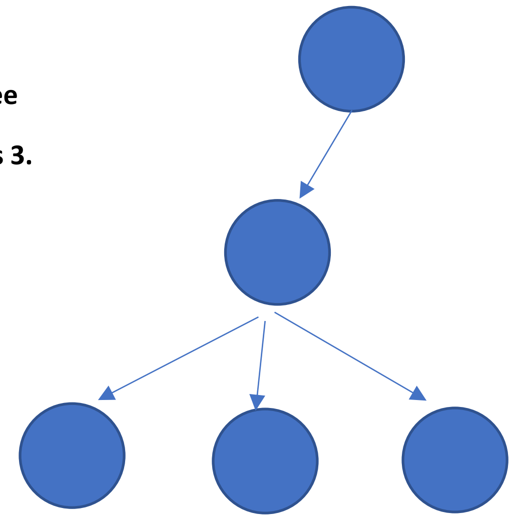
N nodes = N -1 edges

A Tree with at most 2 degree is known as Binary Tree. A binary tree always follows, hum 2 humaare 2. (1 aur 0 bhi ho skta h.)

Heads of different departments



This is also a binary tree with degree 1

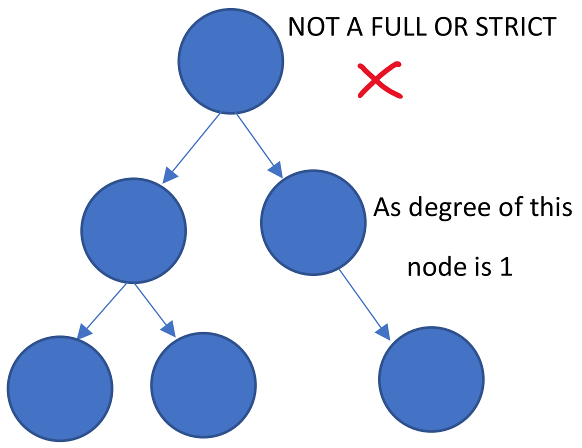


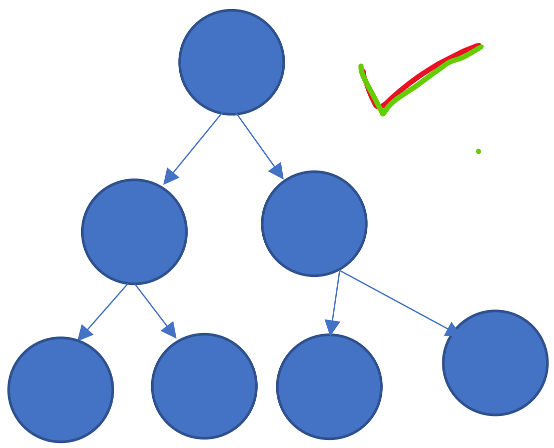
It's not a Binary Tree

As its tree degree is 3.

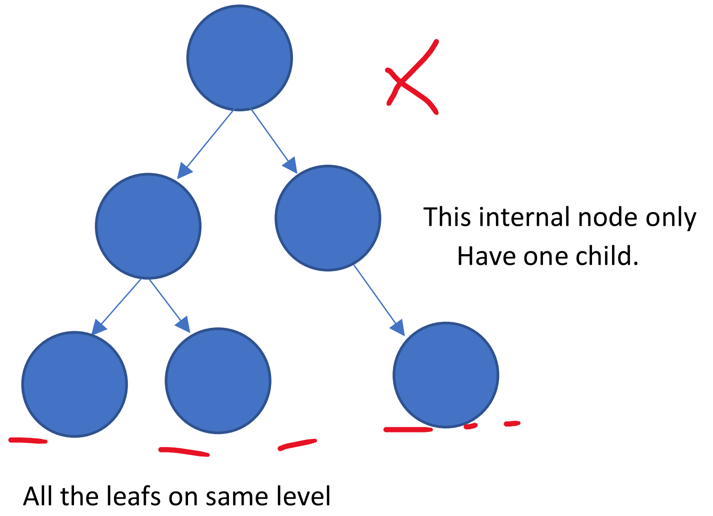
Types of Binary Trees

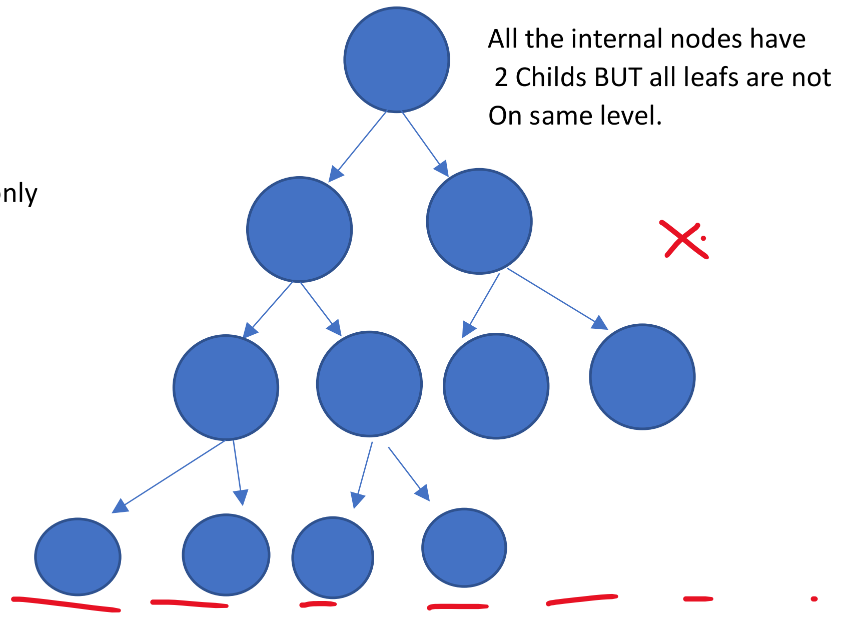
1. **Full or Strict Binary Tree:** The Binary tree whose all nodes, either have 0 or 2 degree is known as full or strict binary tree.

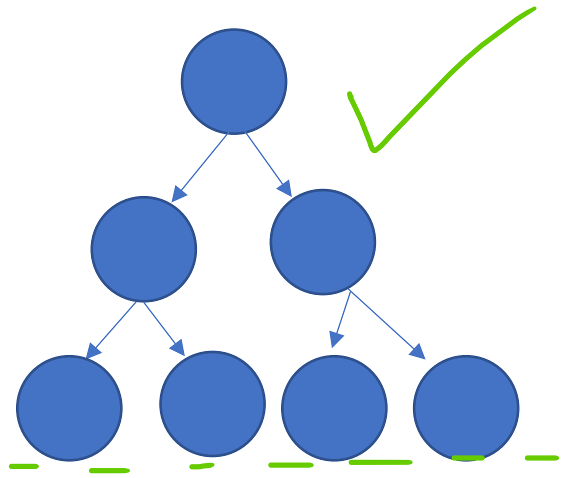




2. **Perfect Binary Tree:** The binary tree who's all the internal nodes have degree 2 and all the leafs are on same level. (Ab leaf h toh degree toh zero hi hogi, 1 ya 2 hui toh leaf hi nai reh jayega)

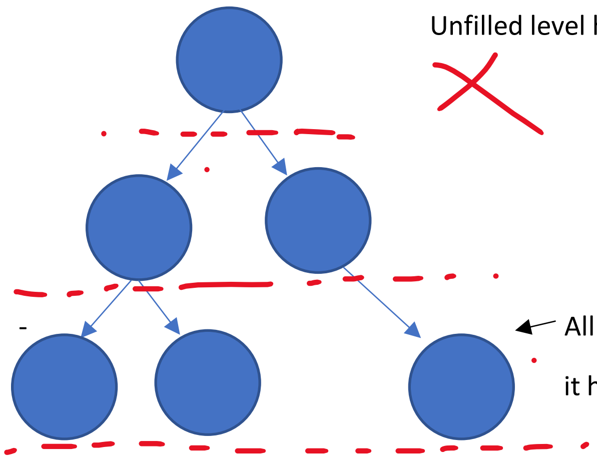




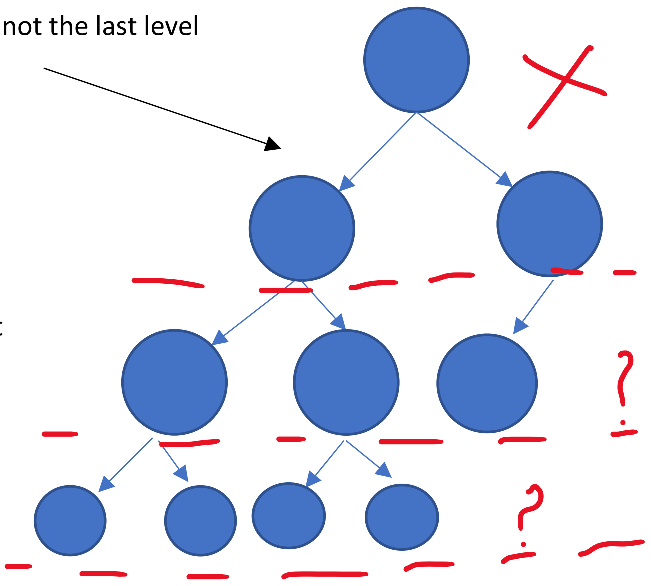


Now in this, all the internal nodes are with degree 2 and all leafs are on Same level.

3. **Complete Binary Tree:** The binary tree who's all levels are completely filed except possibly the last level BUT it must have its key on the left.

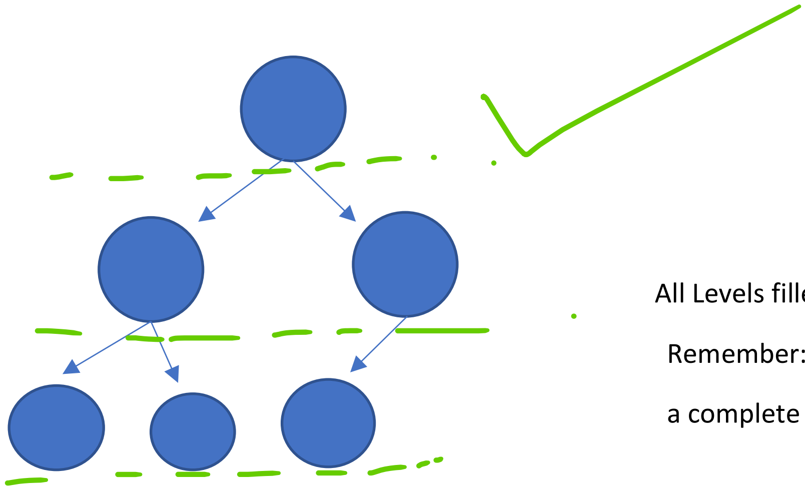


Unfilled level has its keys on left BUT it's not the last level



All levels filled except last but

it has its unfilled key on right.

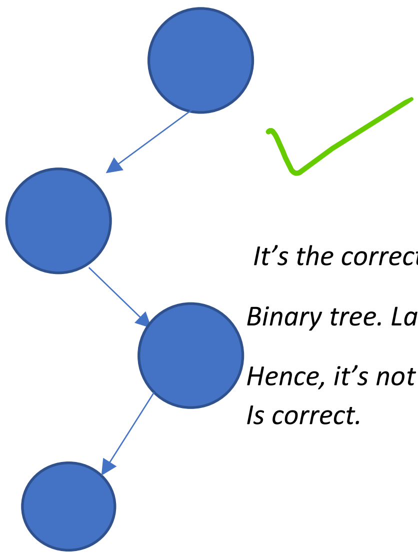


All Levels filled except last that too all keys on as left as possible

Remember: Agar Right left dono ho, mtlb perfect binary tree is also

a complete binary tree.

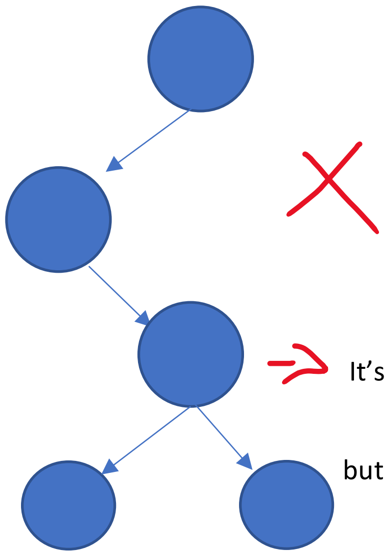
4. **Degenerate Binary Tree: The binary tree who's all the parents have 1 child.**



It's the correct degenerate

Binary tree. Last node has 0 child,

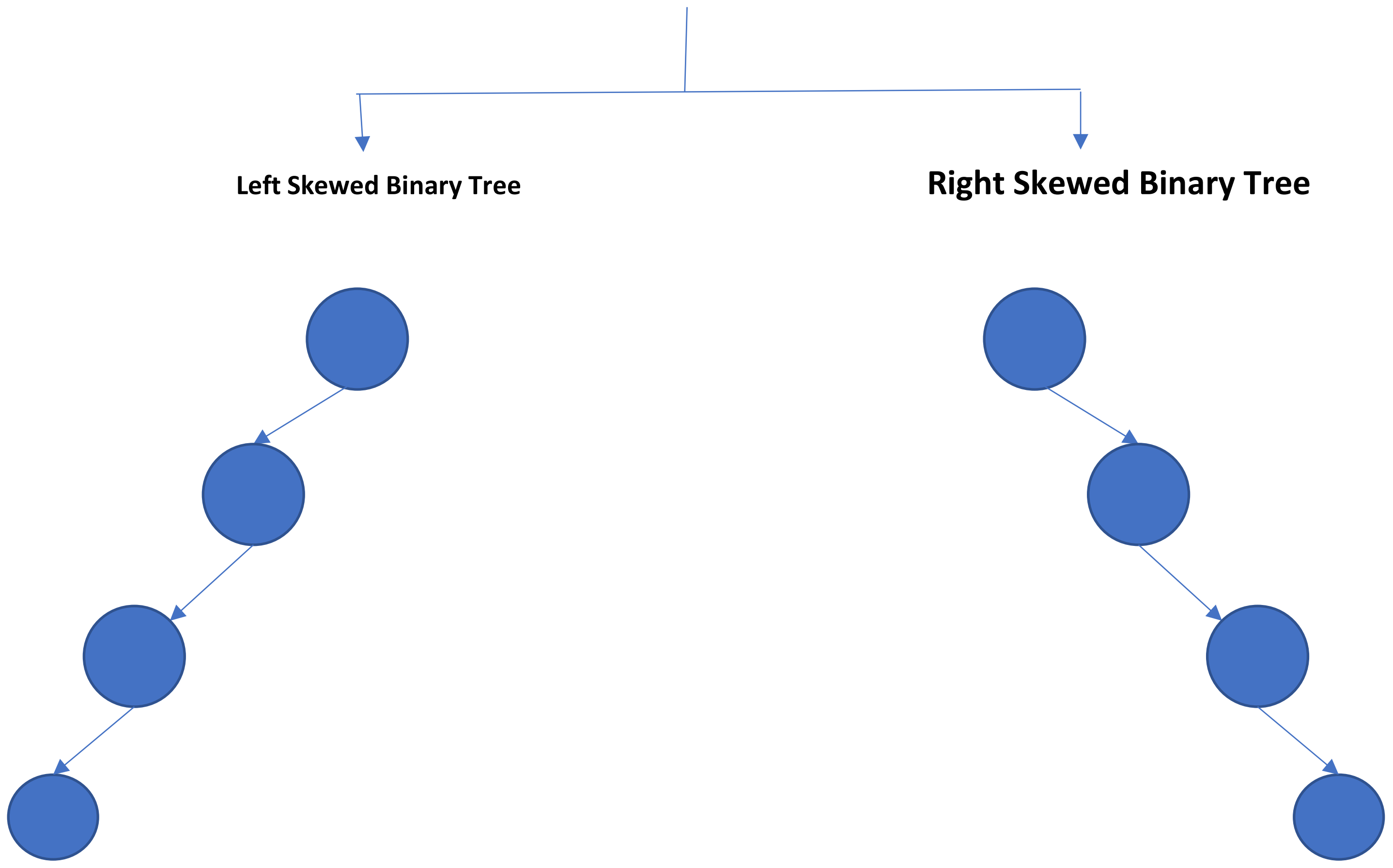
Hence, it's not a parent either hence definition
Is correct.



It's also a parent

but have 2 Childs

5. **Skewed Trees:** The tree's who's all the parent have only 1 child and all the keys are on either side.



MAXIMUM HEIGHT OF A TREE: n – 1
MINIMUM HEIGHT OF A TREE: log n

(Minimum height is floor of Log n, issiliye peeche mt padna ki decimal ka kya???)

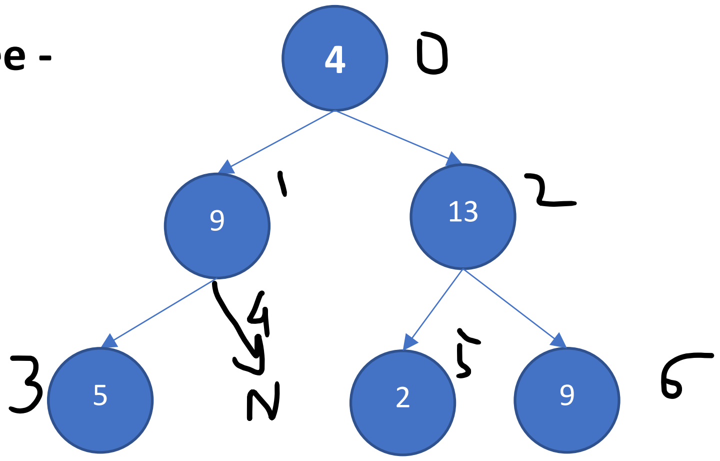
Representation of BT

1. Array Representation of Binary Tree -

Array size required: $2^{\text{No. of levels}} - 1$

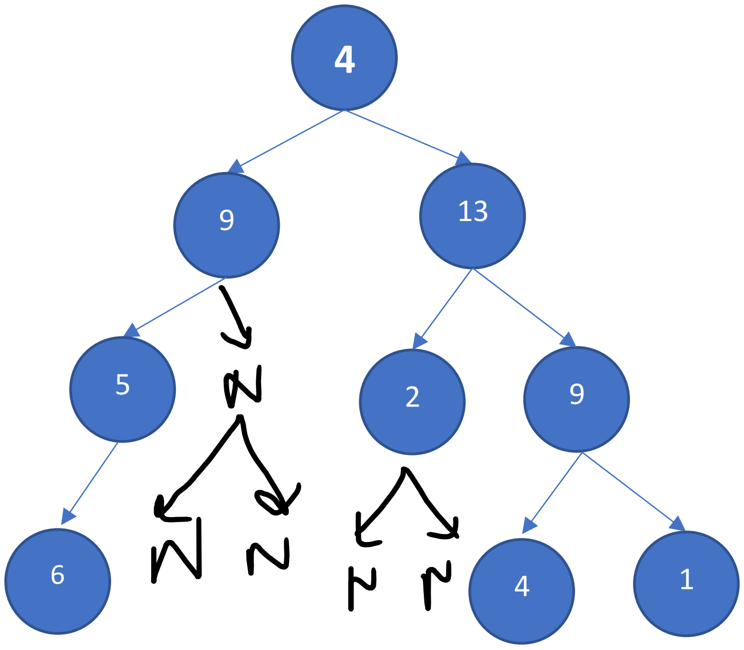
Example: size = $2^3 - 1 = 7$

**** Levels = Depth = Height ****



4	9	13	5	Null	2	9
0	1	2	3	4	5	6

It follow's left to right, if any position is missing, then null will be placed instead of it.



Size Required: $2^{\text{depth}} - 1$

$= 2^4 - 1$

$= 15$

4	9	13	5	N	2	9	6	N	N	N	N	N	4	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Array Representation is not good, as to extend the tree, we need to create a new array and then copy all the elements, also to add any branch (assume after 9(index 1)) then whole array needs to be incremented.

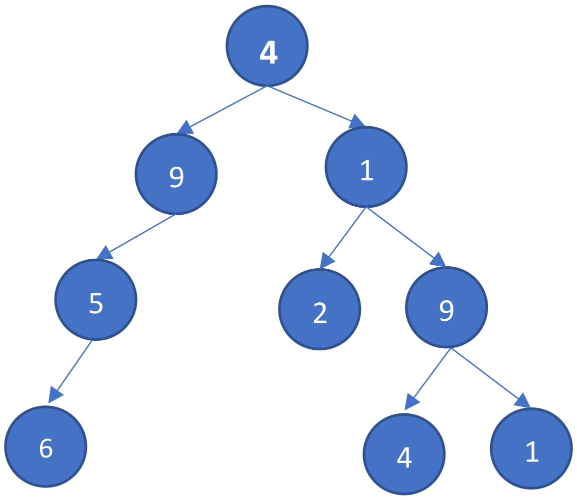
2. Linked Representation of binary tree: -

Doubly linked list is used to represent a binary tree in linked representation.

** It’s linked representation, do not call it linked list representation as Linked list is a linear DS whereas Binary tree is a Non -Linear DS. **

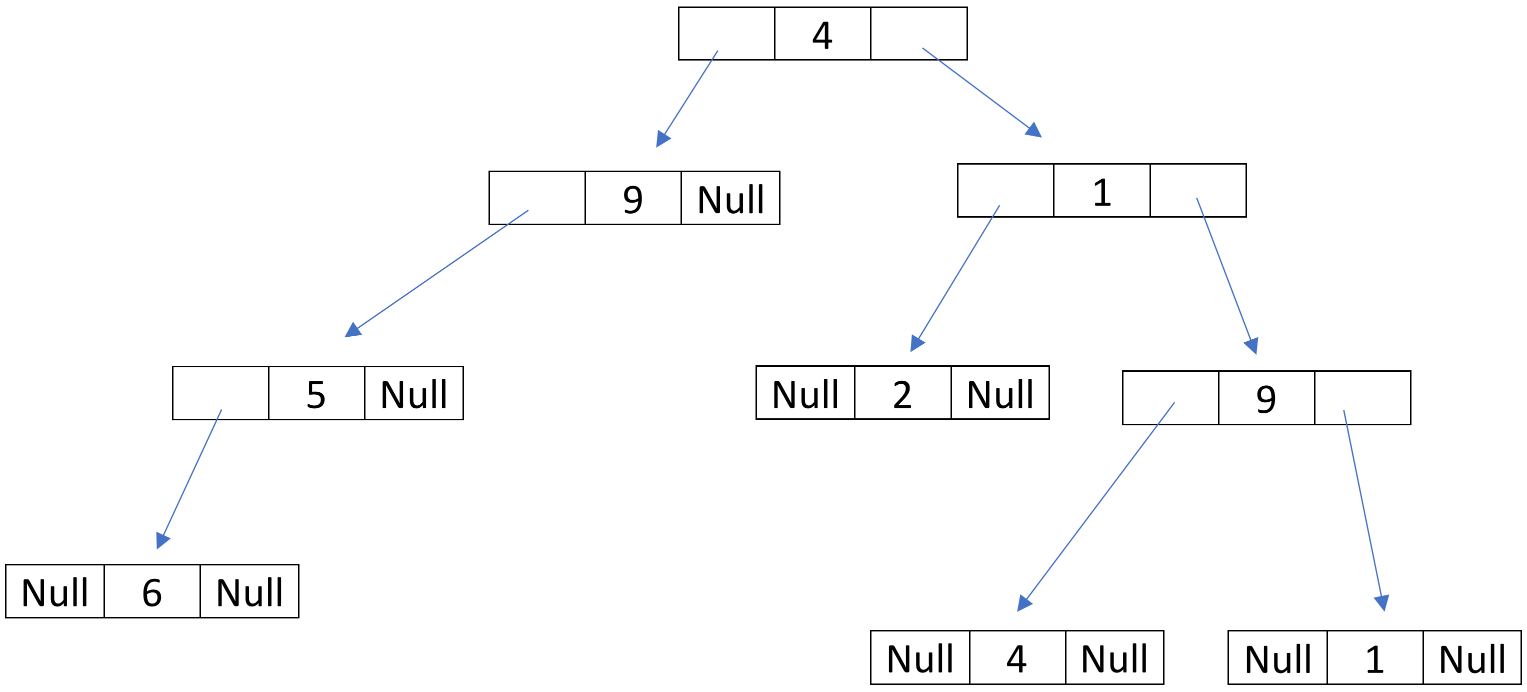
Node Representation:

Pointer to Left	DATA	Pointer to Right



In Linked Representation, a node has 3 fields, pointer to left key(node), data, pointer to right(key), Pointer holds the address of next node (whole node, not just data). If any parent has only 1 child or no child then the pointers will be NULL.

Below is the linked rep. of this.



Transversal of Binary Tree

** Left hamesa left mai rahega and Right hamesa right mai rahega. Sirf root ki position badlegi vo bhi name ke according. **

1. **Pre-Order Transversal:** First root, then left sub tree and then right sub tree.
   Root –> Left –> Right

2. **Post-Order Transversal:** As name suggest, Root at post(LAST) and left and right as usual, on left and right (First left sub tree, then right sub tree, and finally root)
   Left –> Right –> ROOT

3. **In- Order Transversal:** Again, as of name, Root in between and left on left and right on right. (First left sub tree, then root, then right sub tree)
   Left –> ROOT –> Right

Pre: root[left][right]

1 [2[4][5]] [3[][6]] //now 2 is root and have left and right sub trees

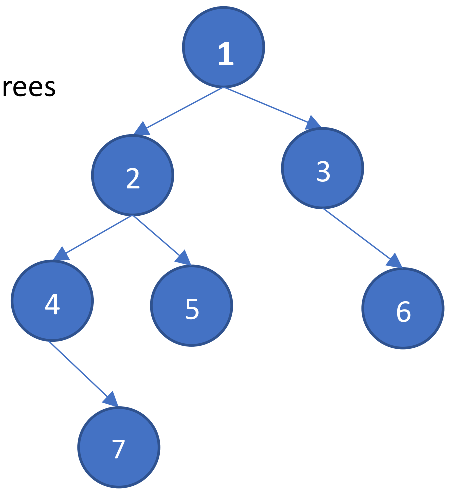
1 [2 [4 [][7]]] [3[][6]]

Pre-Order Transversal Result: 1247536

Post-Order Transversal Result: 7452631

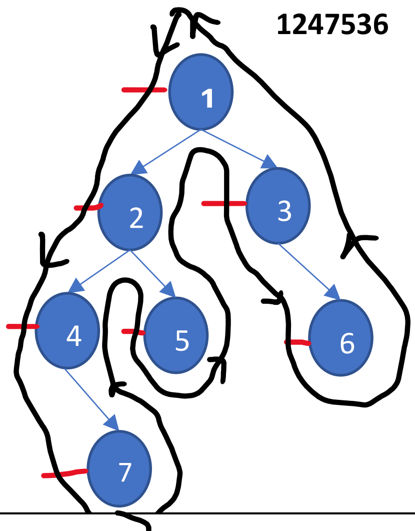
In-Order Transversal Result: 4725136

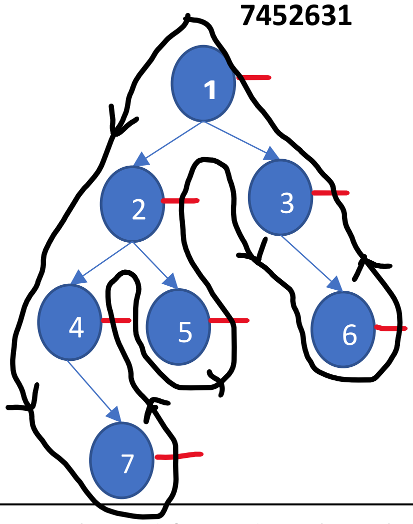
TIME COMPLEXITY OF ALL: O(n)

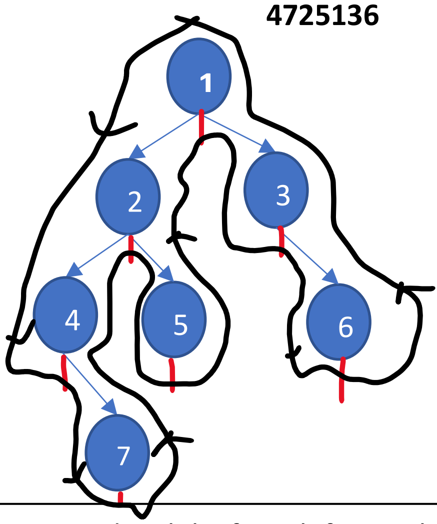


Above Tree is used for C code

Quick Truck to check results:







For pre, draw lines on left for post draw lines on right and for in draw lines below. Now start riding bike from left to right (as in black line) and as you cross red line, write it's value.

Binary Search Tree

Binary search tree is a binary tree which satisfies the following properties:

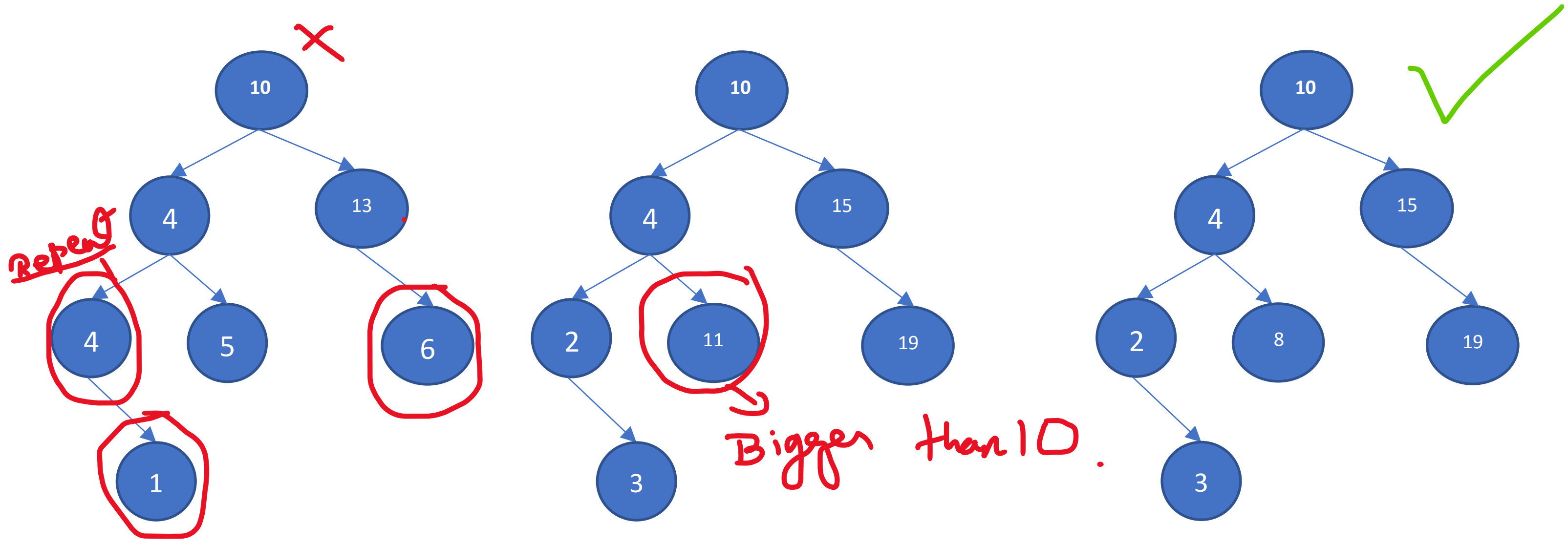
1. **All the elements of left subtree are smaller than root element. (Not Equal too)**
   (The value of root element must be greater than all the values of Left sub tree)

2. **All the elements of right subtree are greater than root element. (Not Equal too))**
   (The value of root element must be smaller than all the values of Right sub tree)

3. **Any element is not repeated in whole Binary Tree.**

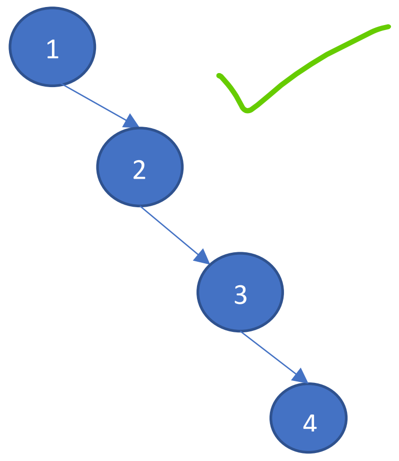
4. **Left and right sub trees are also the Binary search trees.**

5. **In-Order Transversal of a binary tree gives an ascending order Sorted Array.**



In-Order Transversal: 2 3 4 8 10 15 19

(Perfect Sorted Array)



Searching.....

Why Binary Search Tree?

Binary search tree makes it extremely simple and efficient for an element to search.

Time complexity of normal tree: O(n)

(We need to transverse through all the elements.)

Time complexity of Binary Search Tree:

In worst case (may need to transverse through all elements)**: O(n)**

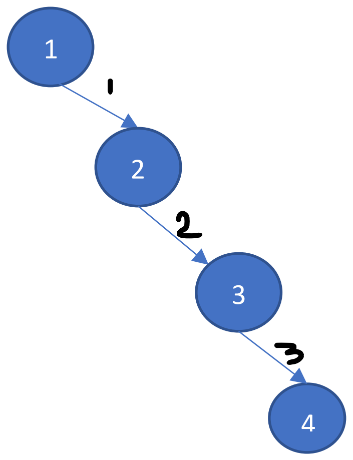
In General: O(h)

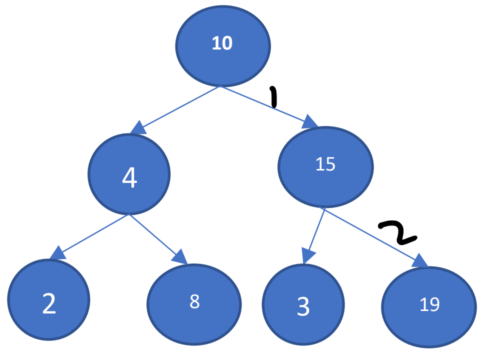
(h = height of tree)

****Minimum Height of a Binary Tree, h = log n, Hence,**

Best Case: O(log n)

log n <= h <= n



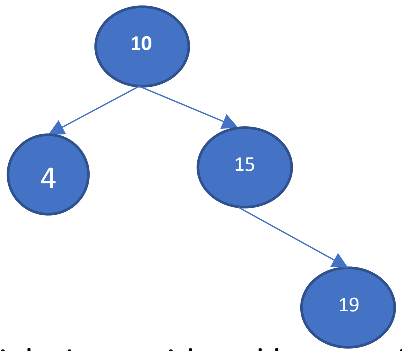


Worst Case of BST, h = n = 3

BEST CASE OF BST, h = log 7 = 2

MAXIMUM HEIGHT OF A TREE: n – 1

MINIMUM HEIGHT OF A TREE: log n



Actually, its floor(log n)

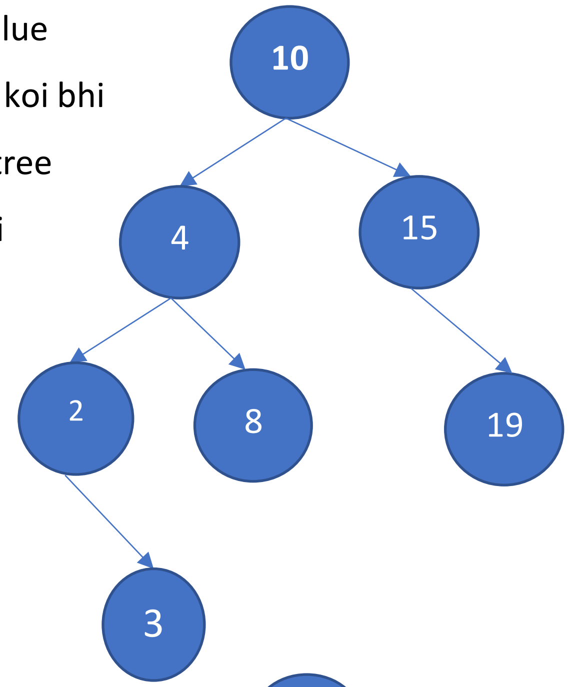
Hence, decimal is not considered

The Height is considered log n as for now

Also, height is 2

Insertion in BST

Toh, dekho insertion in binary search tree ka aisa h ki har ek value Ki position fix rehti h, Apan ko koi bhi element insert karna ho, koi bhi Vo element kisi ki bhi leaf zarur ban skta h jo ek binary search tree banaye. Maine bahut kosish ki, ki koi ek aise condition le aau ki beech mai hi insert karna pade bhut har ek element kahi na kahi leaf ban hi skta h. (For detail, see code, usme detail mai explained h). **Element must not be already present in BST.**



Chalo ek baar issi tree mai dekh lete h. Let's insert 12.

Lo, Bs transverse karte jaao jb tk null na aa jaye.

Agar element node value se bada h toh right taraf.

And agar chota h toh left taraf.

In above example,

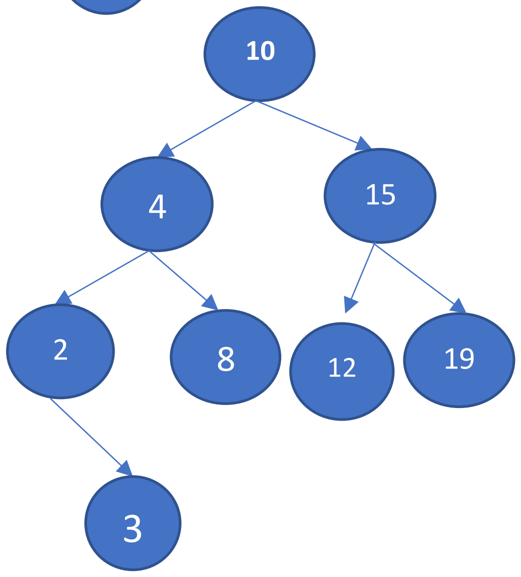
(initially in 10)

10 < 12 hence right node (now in 15)

15 > 10 hence left node (now in Null as left of 15 is null)

Ab null mai aa gaye h toh insert kar do New node.

Aur Lo, ho gaya insert.

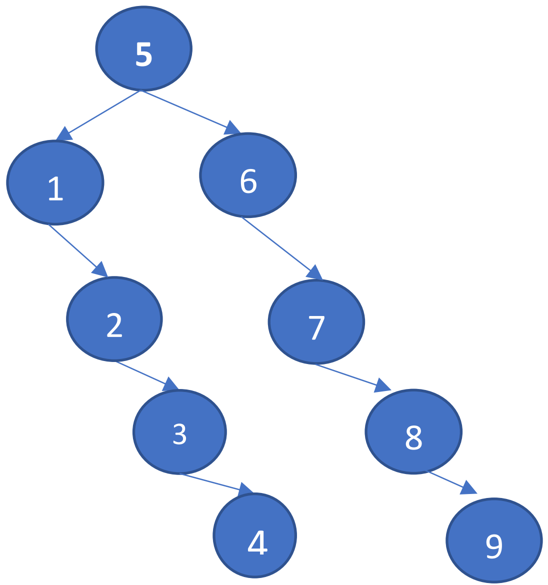


Accha, apan insert kr ke hi poora binary tree bhi bana skte h, lets

See agar 1 to 10 insert kare toh kaisa Banega tree.(sbse pehle 5 karte h insert taaki root ban sake)

Insertion order:

512346789



DELETION IN BST

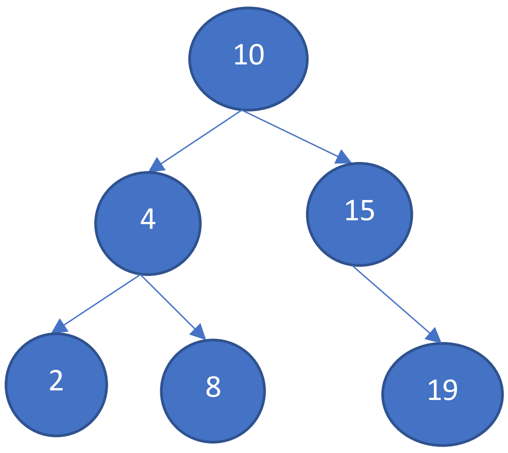
Leaf Node

Koi dikkat hi nai,

bs free kr do memory

ex: Delete 3.

(Consider above tree as base tree)



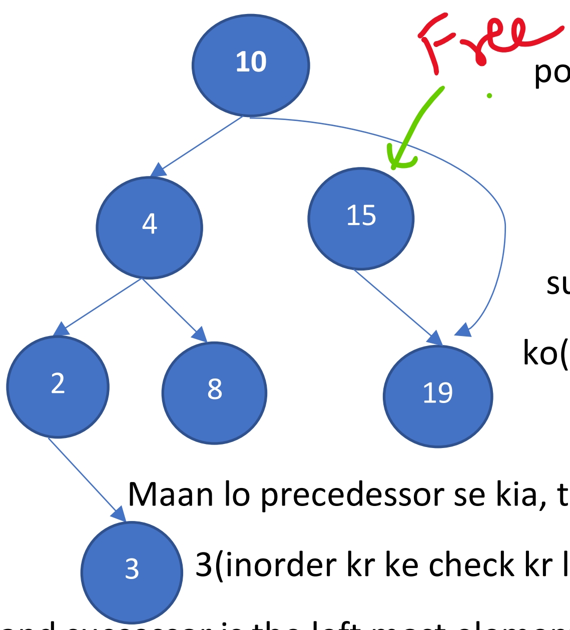
Single Child

Isme bhi koi dikaat nai,

Bs us node ko free karo aur previous node ke Left ya right ko is node ke left ya right pe point kr do.

Ex: Delete 15

(19 ke neeche bada tree hota toh bhi ye hi hota)



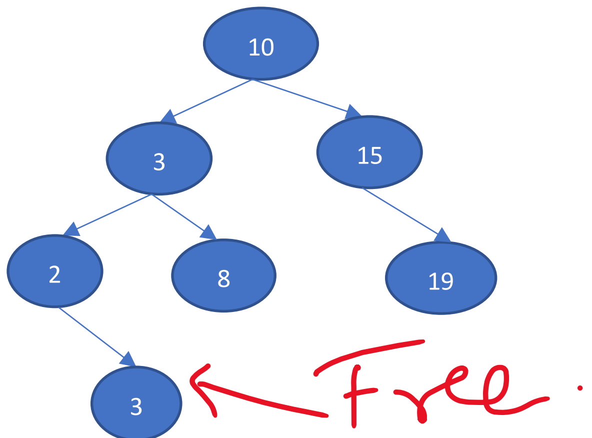
Pita of 2

(2 Child)

Ab ye h thodi dikkat waali baat, maan lo 4 delete karna h, Ab 4 ko free toh kr nai skte, Na hi 10 ke left pointer ko dono(2 and 8) ke saath point kr skte. Toh kya krte h ki inorder precedessor ya inorder successor of 4 se replace krte h 4 ko(taaki BST maintain rahe) kisi se bhi kr skte ho tum code mai.

Maan lo precedessor se kia, toh 4 ka precedasoor kya hoga…

3(inorder kr ke check kr lo) Vese precedessor is the right most element of left sub tree and successor is the left most element of right sub tree. Ab 4 ki jagah 3 likh diya… Ab neece se 3 ko delete krna h, toh delete function 3 ke liye chalayenge left sub tree ko root maan ke(agar successor use kia hota toh right sub tree ko root maante) Ab 3 toh leaf tha, issiliye simply free ho jayega aur null return karega.



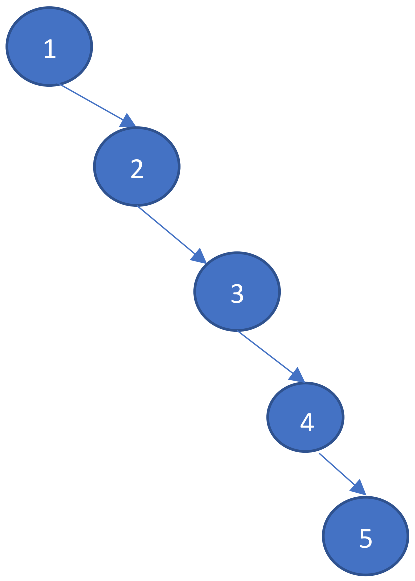
** Remember: jis node ki value 4 thi, uski value bs update ki h, us node ko free wagera nai kia h.

**But last mai ek node free zarur hoga, jb deep mai koi leaf ya single Child milega.

AVL Trees

(Full form main a jaao, scientists ke naam h jinne ye concept discover kia h)

If I tell you to create a BST from 1 to 5, and you are in hurry then you may create it like this (on the right). Which….Logically is correct But, in this binary search tree, searching an element would take O(n) Time complexity. As searching of an BST is given as O(h) {h = height}, Its very important to have a minimum height to make it efficient. Well, the only way to have a minimum height is through balanced BST. And that's what the AVL tree is.



AVL tree is the self balancing BST, where the difference between height of left sub tree and the height of right sub trees can not be more than 1 for all nodes.

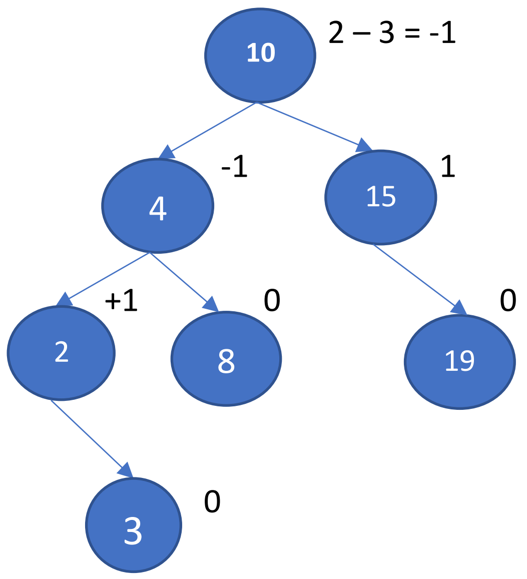
And that's what BF is (not Boyfriend):

Balancing Factor: height of right sub tree – height of left sub tree.

(You may also see left – right in some places, but that really dosen't matter, it's the mod which matters.)

Desi Bhasa mai, AVL tree vo BST h jisme har node ka balancing factor, -1, 0 ya 1 ho. Bs,

Maths mai bole toh where, |BF| <= 1



Now as you can see on your left is an AVL tree as all nodes follow the rule of |BF| <= 1, I know we Have been using it in our all above examples, and they are also AVL trees, it's just with the labeled BF on each respective node.

As you can see, for 10 height of right sub tree is 2 and that of left sub tree is 3 hence BF = -1, same that of 4 is 1-2 = -1 for All the leaf nodes its 0 – 0 = 0 and so on……

** Pro tip: zyada right – left dhyaan mai mt rakhna, jaisa mn kare kr dena, fi mod maine rakhta h Vo bs 1 se bada nai hona chahiye nai toh AVL nai reh jayega. **