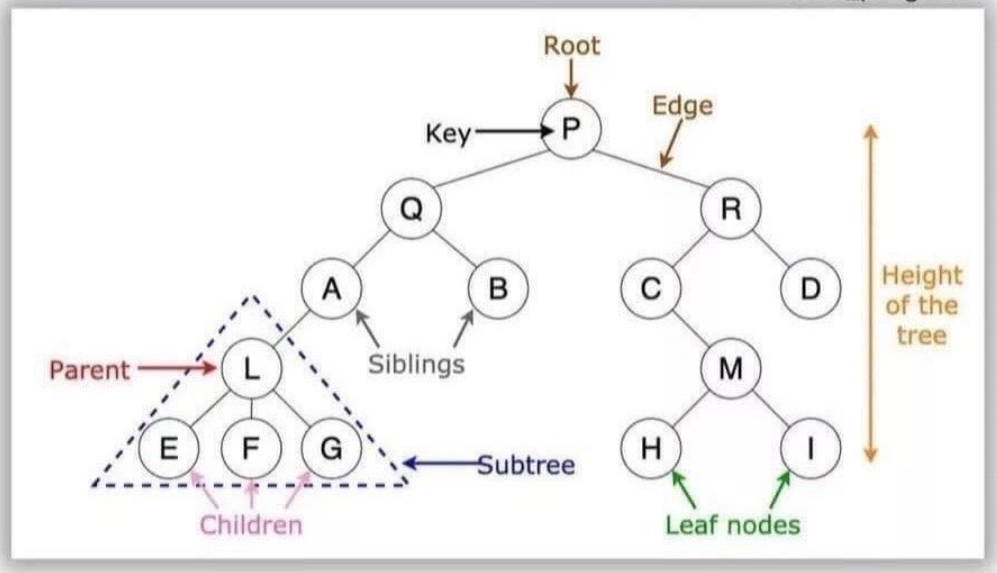
Tree

Binary Tree : • It don't obeys the sort of the Numbers, Just insert like comes

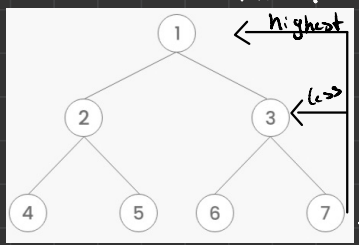
A Tree is a non-linear hierarchical data structure that consists of nodes connected by edges. A node is an entity that contains a key or value and pointers to its child nodes.

Complete / incomplete ?

learn_programo



Hierarchical ?

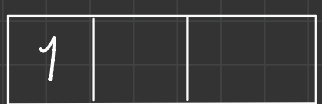


- Elements are organized in levels
- Each level represent a different degree of importance
- height: The longest path from the root to a leaf

In the program we use a queue: to dont loose track of the next nodes to Insert its childs if the user want

1) P = Root

InQueue (P)



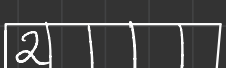
P = Dequeue; ← Parent Node

P -> 1

2) Insert left child

InQueue (t)

temporal pointer to a tree node



P -> LeftChild = t

3) Insert RightChild

InQueue (t)

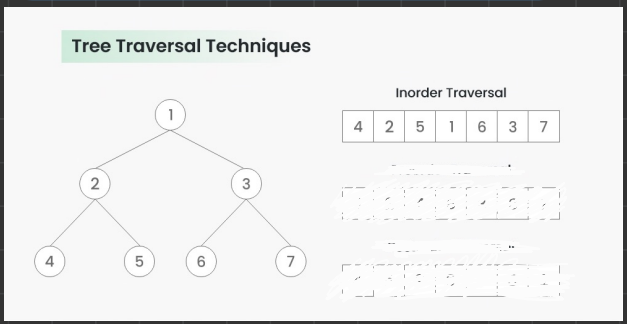
P -> RightChild = t

Repeat

P = Dequeue; ← Parent Node

2

In order Traversal:



• Left • Data • Right

- goes to the left most node
  - Prints its data
- go to right

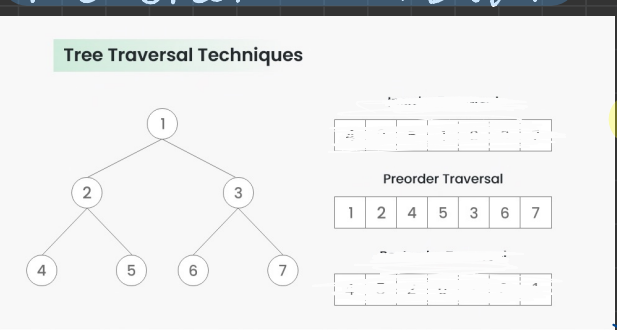
left - Right

You process child Node Child

• Prints in ascending order if is a BST
• the Root is displayed in the middle
• Can be used to check if is sorted the tree

[Implicity we are using the stack because we are traversing through recursive calls]

Pre-order Traversal:

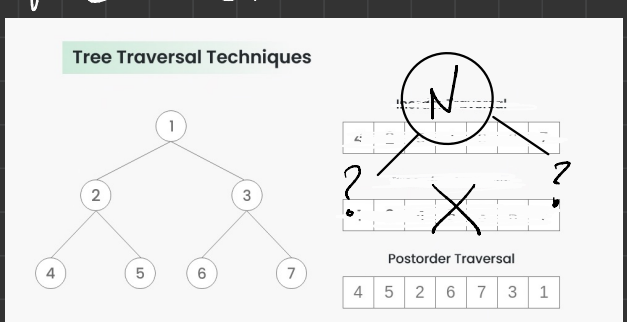


• Data • Left • Right

Is commonly used to Copy a tree since you visit the Root Node and then, the childs

[You process first node child child]

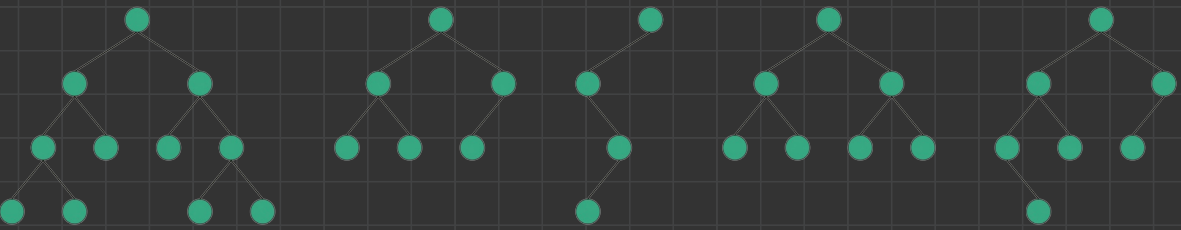
Post-order Traversal



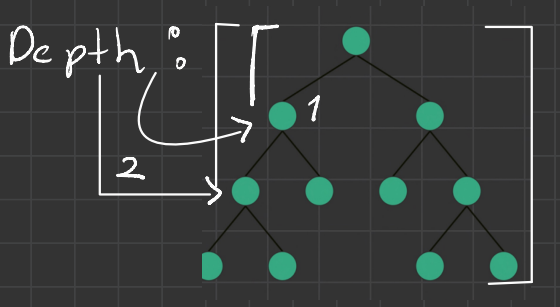
• left • Right • Data

• Commonly used to delete the tree, since the last Node that visit is the Root

Process: Left - Right - Node



Full | Complete | Degenerate | Perfect | Balanced



Level 0
1
⋮
height

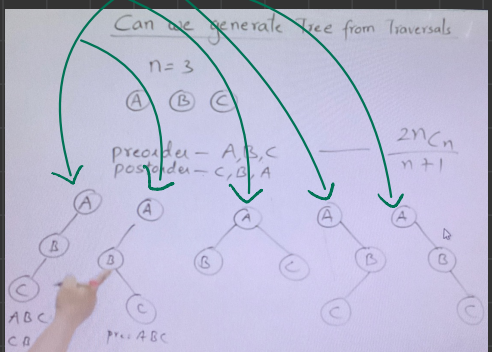
Degree: The number of childrens a node has

Generate a tree from traversals

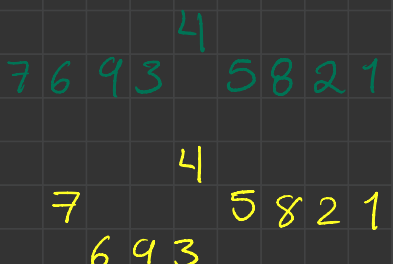
We cannot generate a tree from just one preorder – Inorder – Post order, if we just have one reference, many possible trees can be generated

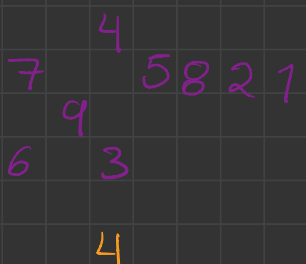
Preorder: 4 7 9 6 3 2 5 8 1

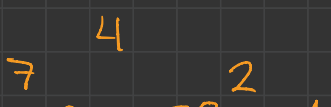
Inorder: 7 6 9 3 4 5 8 2 1

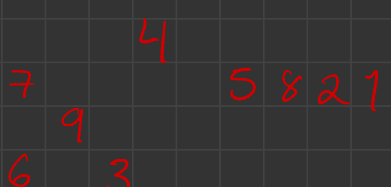


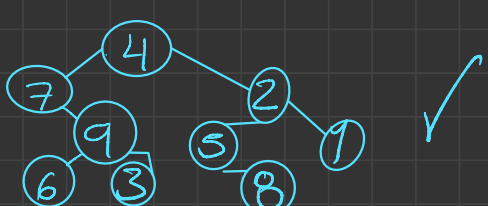
- Search the number in Inorder
- Make it a node
- Stalish the left part and Right part as childs











If you search and is already a single node don't do anything

Preorder: 4 7 9 6 3 2 5 8 1

Inorder: 7 6 9 3 4 5 8 2 1

-1 Means, there is nothing more

Preorder: 4 3 5

Inorder: 3 4 5

4

4 → Left = Create(Preorder, Inorder, begin, idx-1) (0, 1-1=0)

Recursive call

3 → Left = Create(Preorder, Inorder, begin, idx-1) (0, -1)

-1 end < begin

3 4

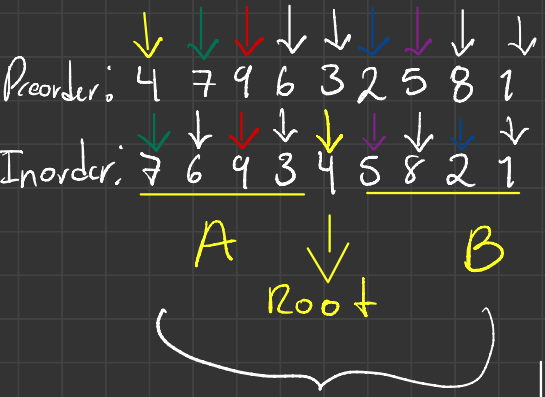
4 → Right = Create(preorder, Inorder, idx+1, end) (1+1=2, 2)

5 → Left = Create(Preorder, Inorder, begin, idx-1) (2, 2-1=1)

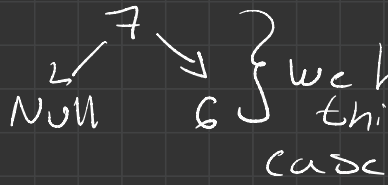
1 < 2

3 4 5

Return *P



If I take 7 as my Root



Preorder: 4 3 5

Inorder: 3 4 5

Main

Root = Create (Inorder, Preorder, start, end

Find ⟶ Peorder [0] // ← First Find, this is the Root

4 ⟶ leftchild (Preorder, Inorder, start, Idx-1) 1-1=0

3 Idx = 1 // Where was found 4

If (start == end) Left Node Return

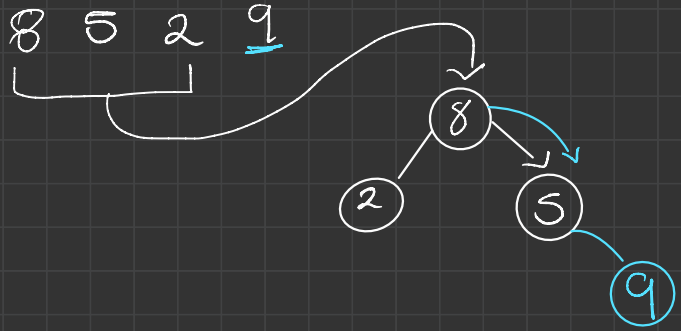
3 4 ⟶ Right (Preorder, Inorder, Idx+1, end) 1+1=2 2

5 If (start == end) Left Node Return

Return

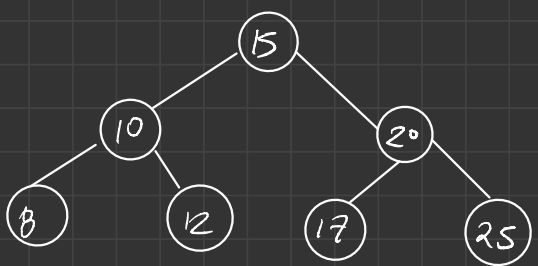
Binary Search tree:

the program was created depending on the inputs you introduce



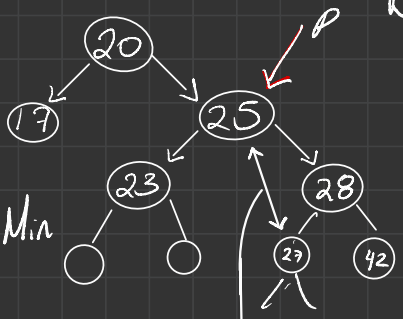
Rules:

- No Duplication
- In order gives you sorted order



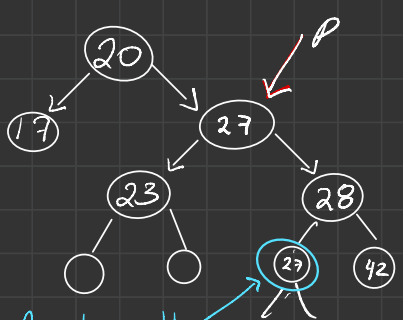
Worst case that's why we always try to balance the tree on AVL Tree

Deletion



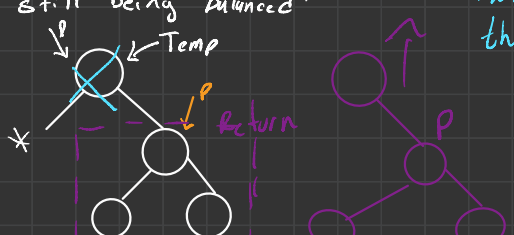
1) Find Min
P->R

by rule that number will acomplish the rule that Replacing "25" to "27", the tree will still being "balanced"

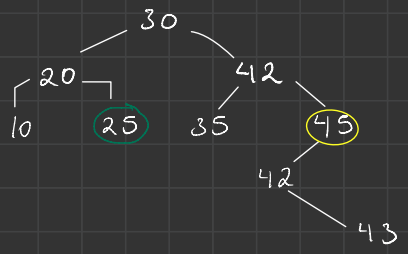


Acording with this case search how it will be deleted this node

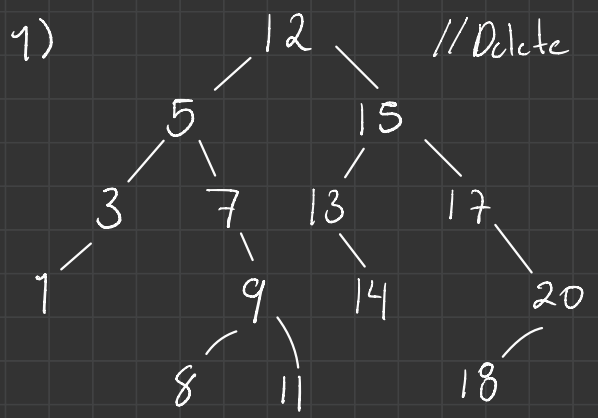
1) Left Node
2) Right Node
3) Lefth Node
4) Parent



Deletion of a node in tree:

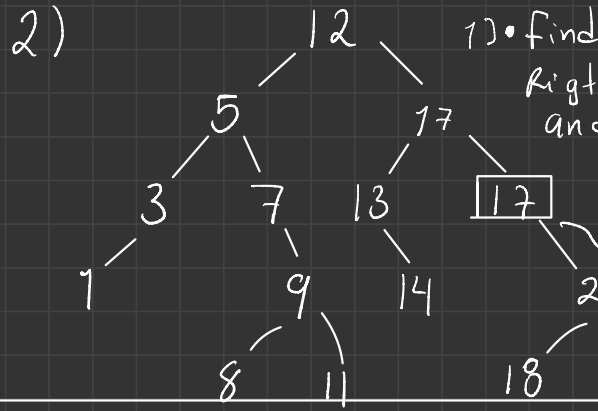


1) // Delete 15



Case 1: No child
Case 2: One child
Case 3: 2 children

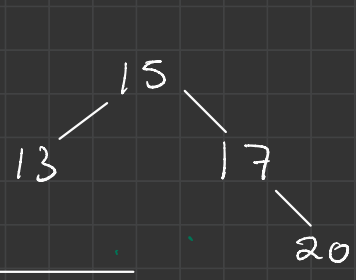
2)

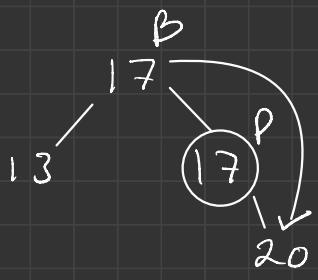


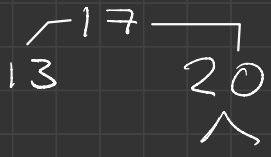
1) • find minimum of Rigth subtree and replace the value

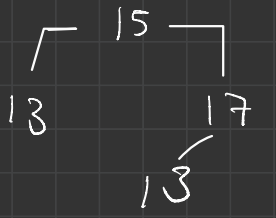
2) Solve the case for 17

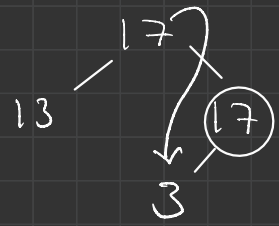
send this address and aply

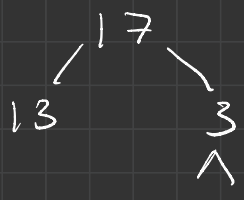


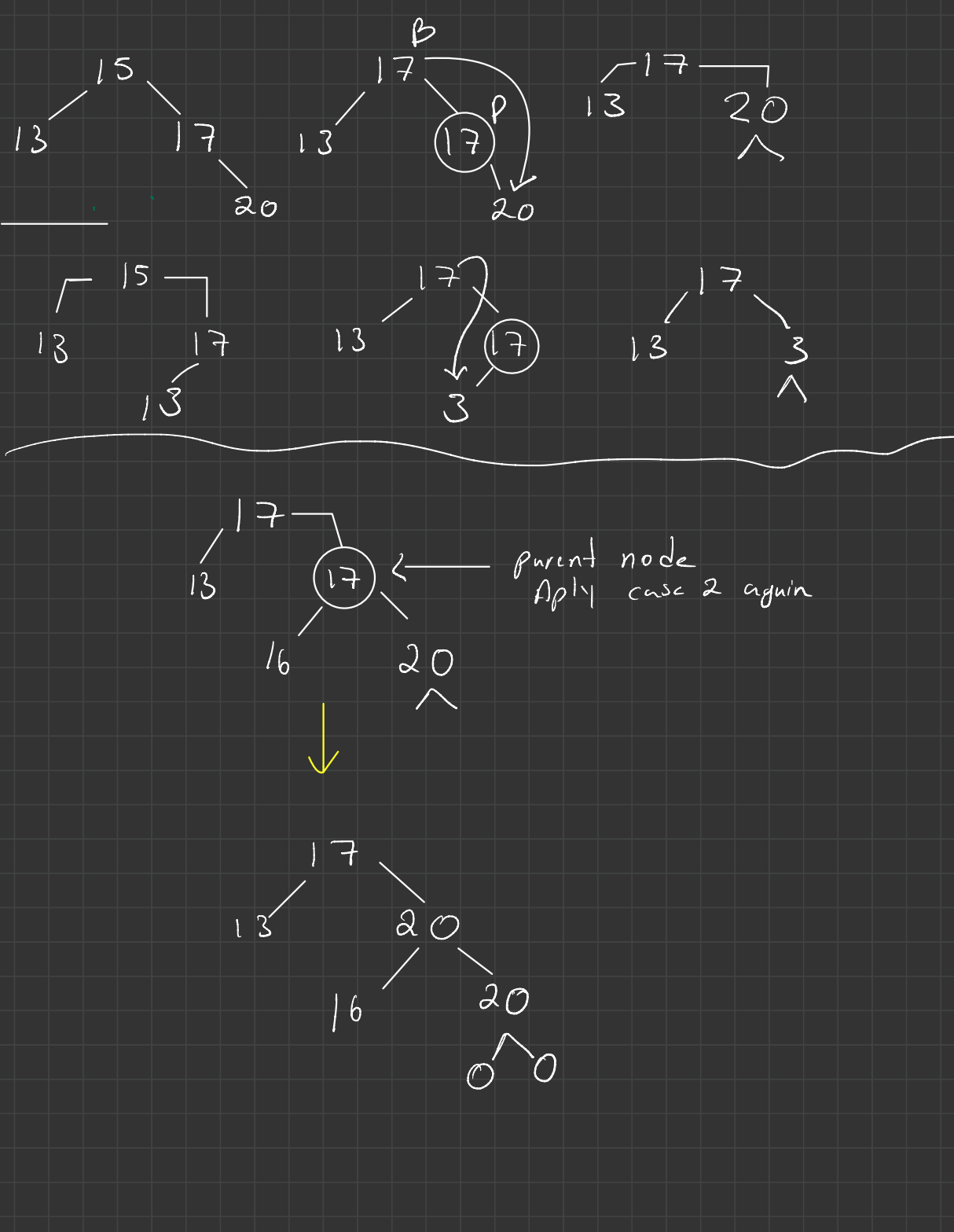




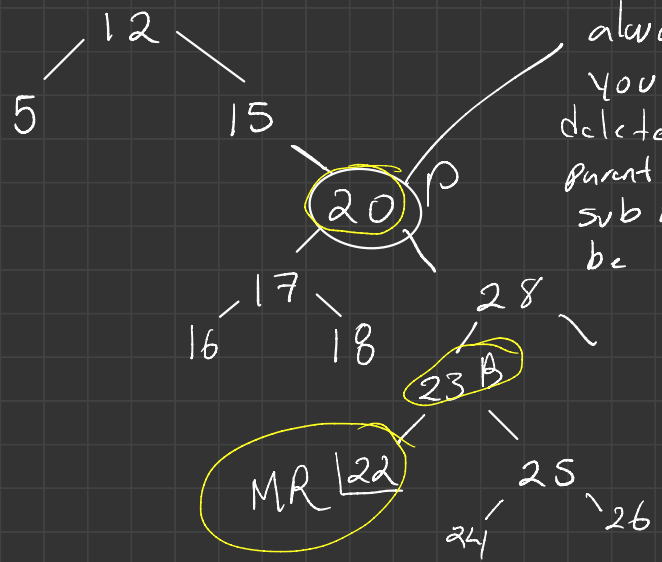






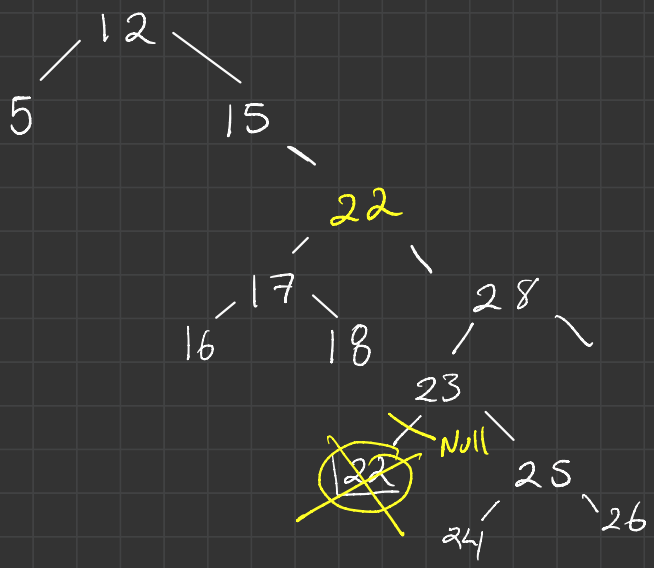

B
15
13
17
20
17
13
17
P
20
17
13
20
15
13
17
13
17
13
17
3
17
13
3
17
13
17
parent node
Apply case 2 again
16
20
17
13
20
16
20
0
0

1)

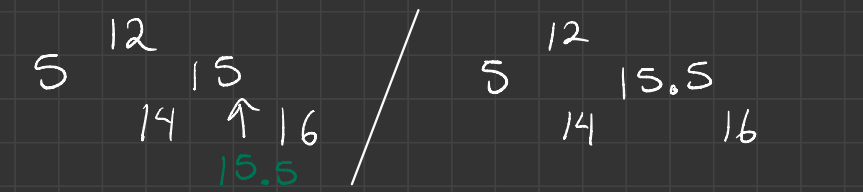


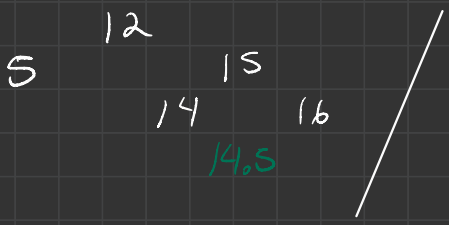
always when you try to delete a parent node, the sub routine will be a leaf node

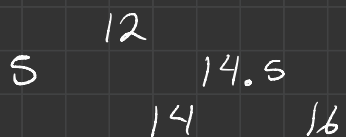
2)

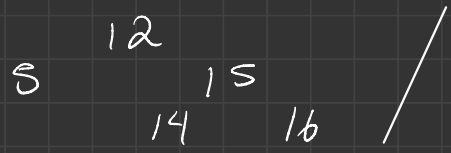


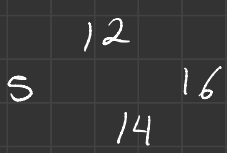
Deletion recursive form



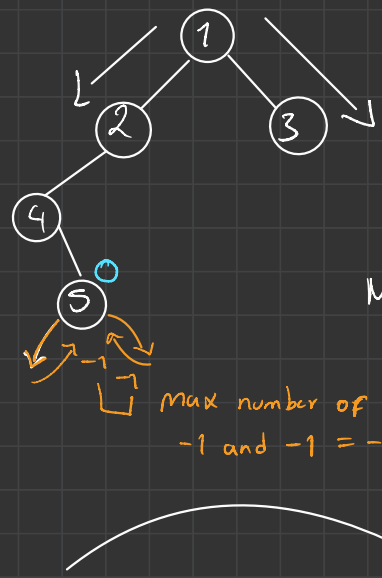








Get high of a tree



- hight of left subtree + Right subtree
* IF is equal to null return -1

Max number of
-1 and -1 = -1 + 1 = 0

Max of 2 Nodes (int n1, int n2){return n1 > n2 ? N1 : N2);}

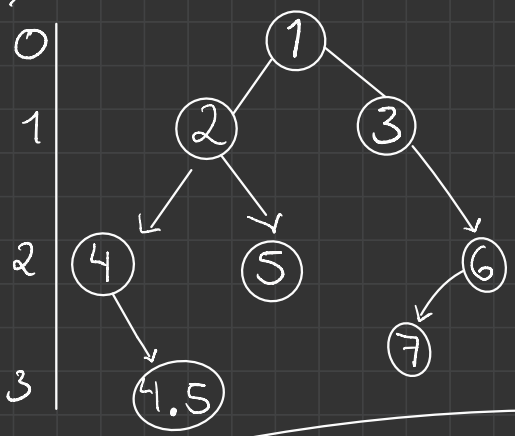
```
int GetHightTree ( Node *P ){
  If( P = Null ) { return -1; }
  else{
    Return 1 + max2numbers (GetHT(P->L), GetHT(P->R))
}
```

Node	Lhight	Rhight	
1	0	0	0+1 = 1
2	-1	-1	0
3	-1	-1	0

```
int height(Node *p){
  If(p == Null/pt.) return -1;

  int Lh = height ( P->Lchild);
  int Rh = height ( P->Rchild);

  return Lh > Rh ? Lh+1 ; Rh+1 ;
}
```

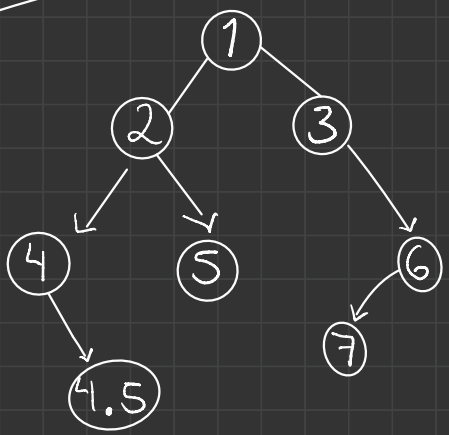
```
int height(Node *p){
  if(p==Null||ptr) return -1;

  int Lh = height(p->Lchild);
  int Rh = height(p->Rchild);

  return Lh > Rh ? Lh+1 : Rh+1;
}
```

Node	Lhight	Rhight	
1	2	2	3
2	1	0	2
4	-1	0	1
4.5	-1	-1	0
5	-1	-1	0
3	-1	1	2
6	0	-1	1
7	-1	-1	0



If is a no child node returns; 0
If is a one child node returns; 1 and two

Queue



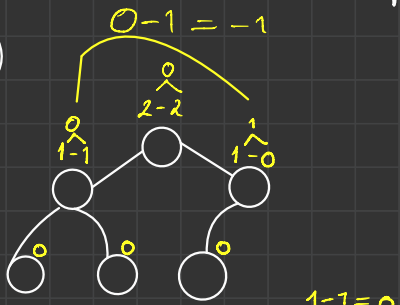
1
2
3

Display by level

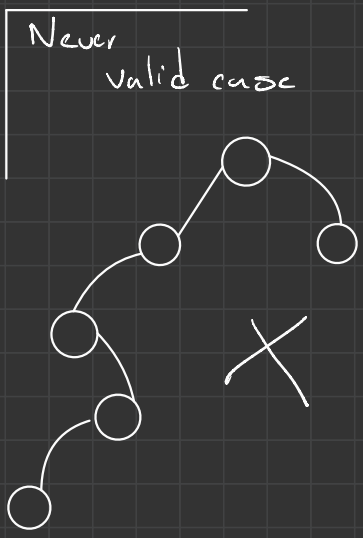
AVL Tree

Balance factor: height left subtree
height right subtree

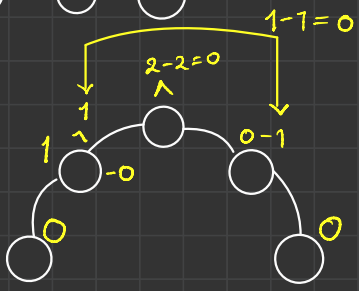
1)



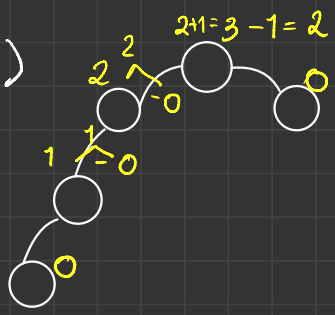
Never
valid case



2)



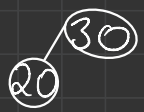
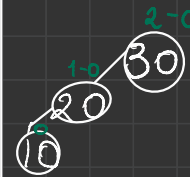
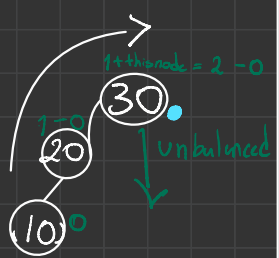
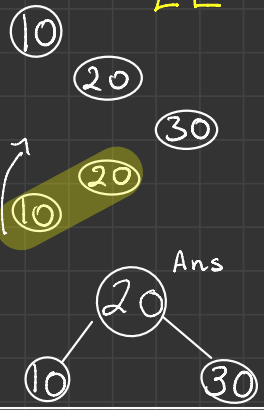
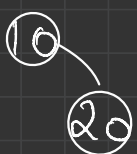
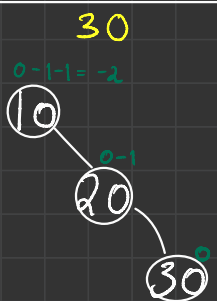
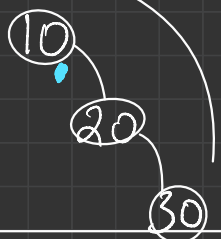
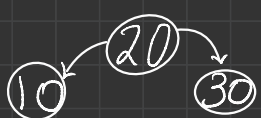
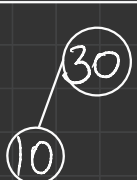
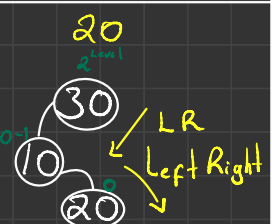
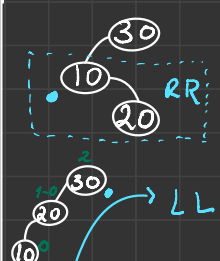
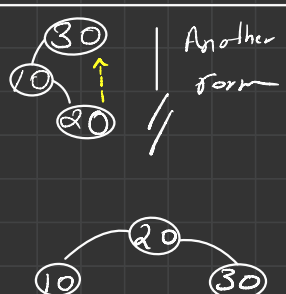
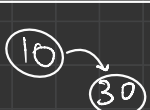
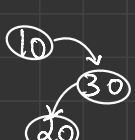
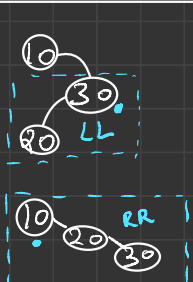
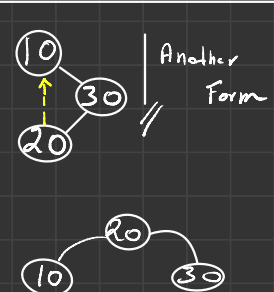
3)



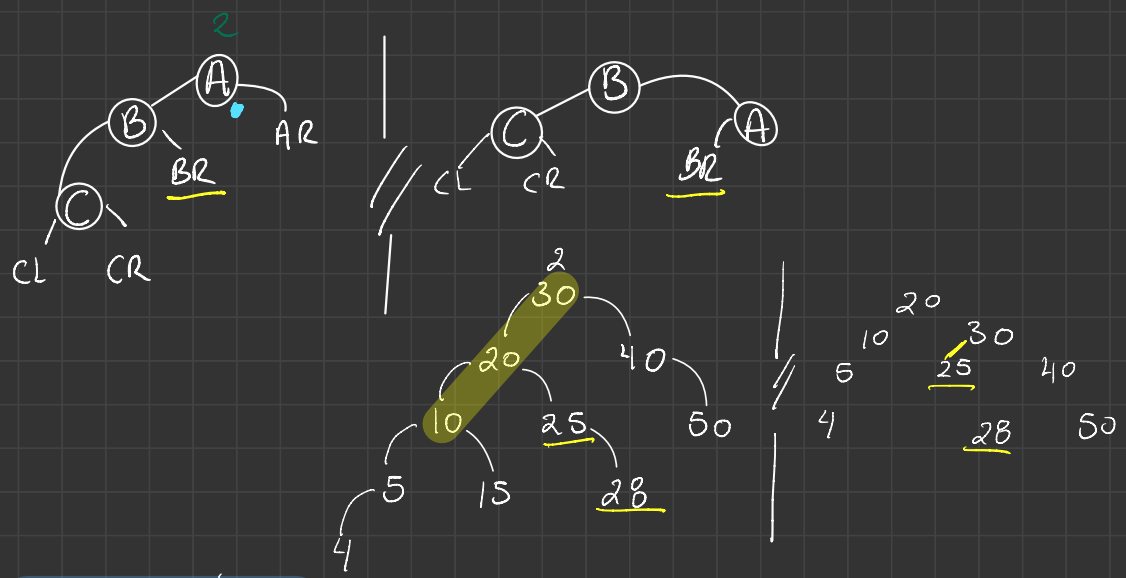
BF = Balance factor = 0, 1 or -1 Balanced
> 1 Un balanced

* Count high or level of the tree -2 or 2 Unbalanced

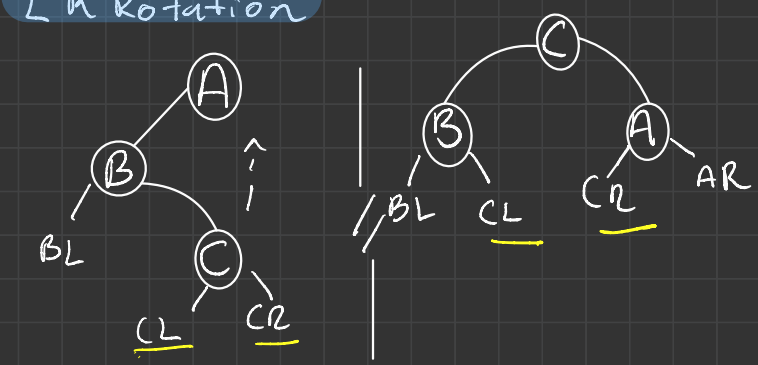
AVL Tree

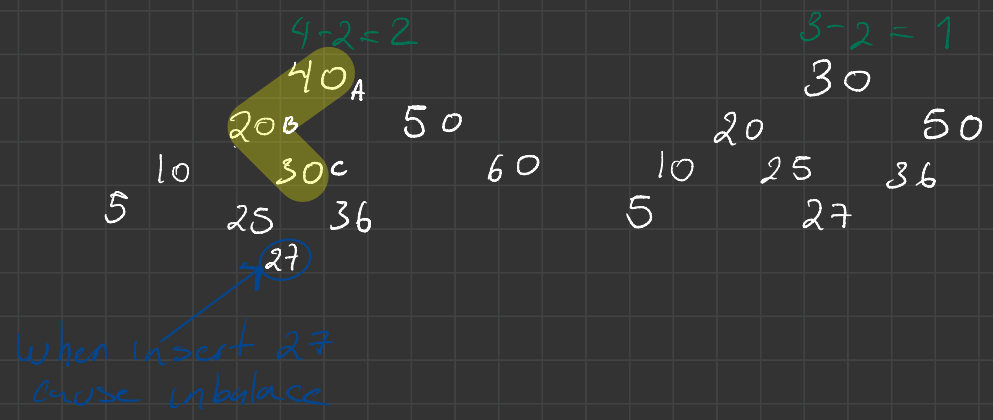
Initials	After Insert	rotation	After
	10 	LL 	LL 
	30 	RR 	
	20 	LR 	
	20 RL 		

LL Rotation



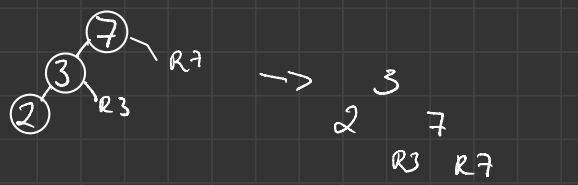
LR Rotation



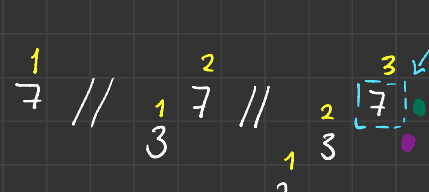


When insert 27 cause unbalace

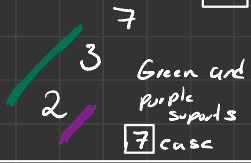
Programm:



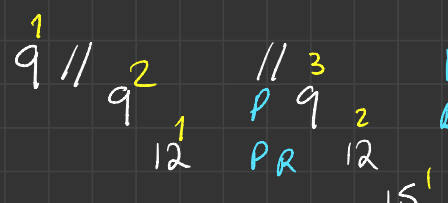
LL Rotation case



If 2 and 1 LL case

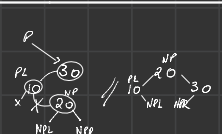


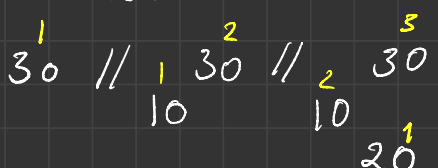
RR Rotation case



If (−2 and −1) RR case

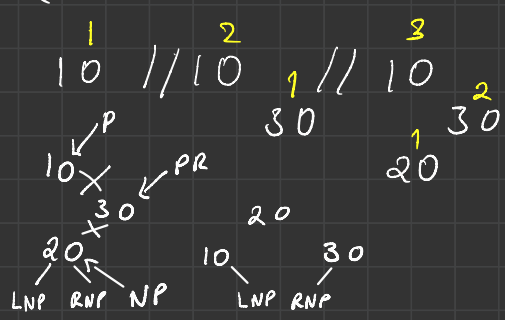
LR Rotation case





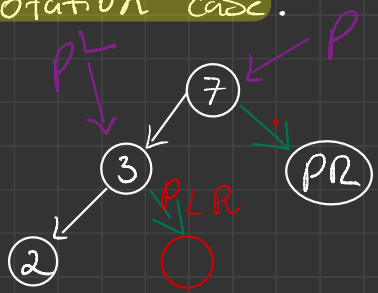
If (2 and −1) LR case

RL Rotation case



If (−2 and 1) RL case

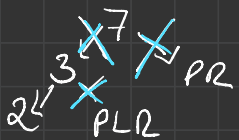
LL Rotation case:



Description

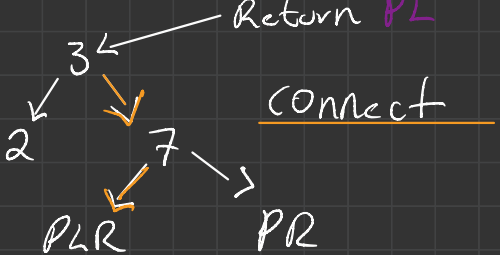
Step

Disconnect



Step

Return PL



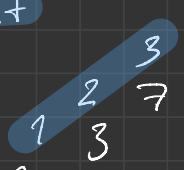
connect

Return PL

---

height

Rotation code

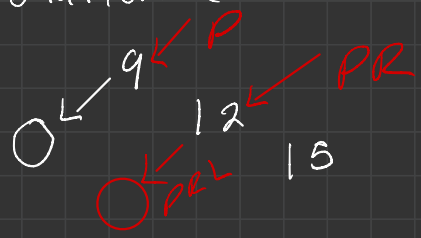


Balance Factor (7) = 2 - 0 = 2

Balance Factor (3) = 1 - 0 = 1

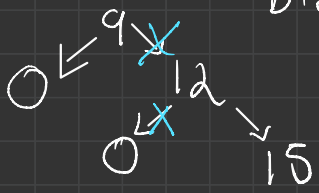
IF we get (2 and 1) we need to perform LL Rotation

RR rotation case



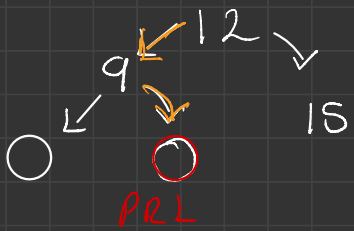
Description

Step



Disconnect

Step

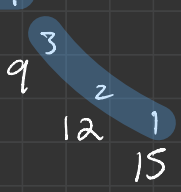


Connect

Return PRL

---

height



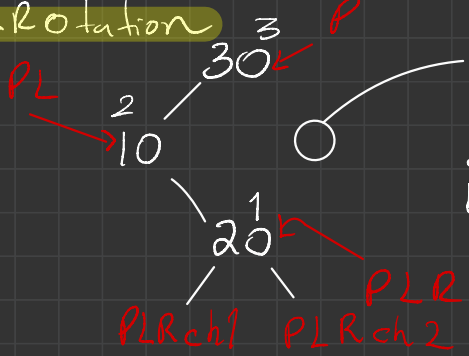
Rotation Code

Balance Factor (9) = 0 - 2 = -2

Balance Factor (12) = 0 - 1 = -1

IF we get (-2 and -1) we need to perform R Rotation

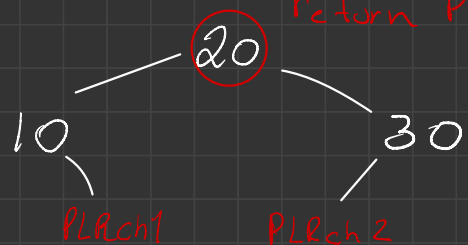
LRotation



balance factor (30) = 2
balance factor (10) = -1

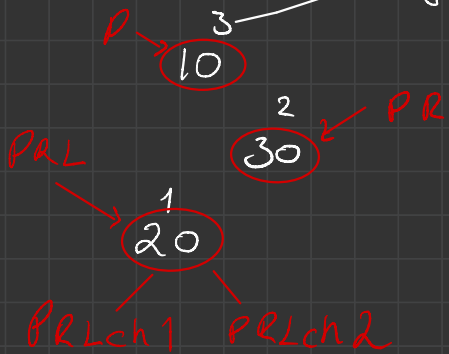
Quote: by default, if you get a 2, -1, case the node that holds "10", doesn't have Left child

return PLR



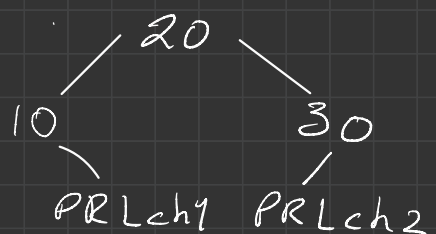
RLRotation

height

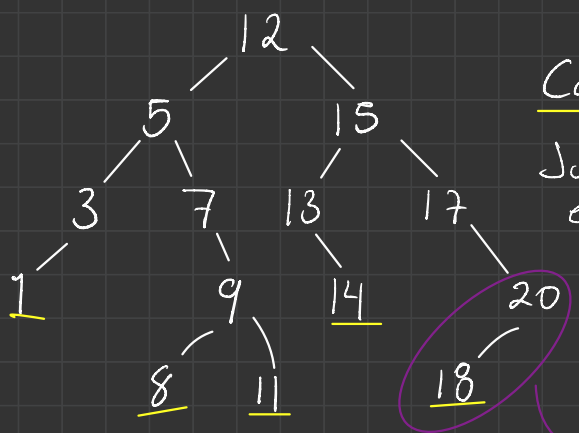


Balance Factor (10) = -2

Balance factor (30) = 1

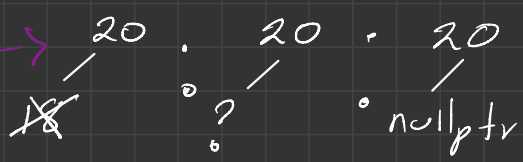


Deletion of a node in the tree



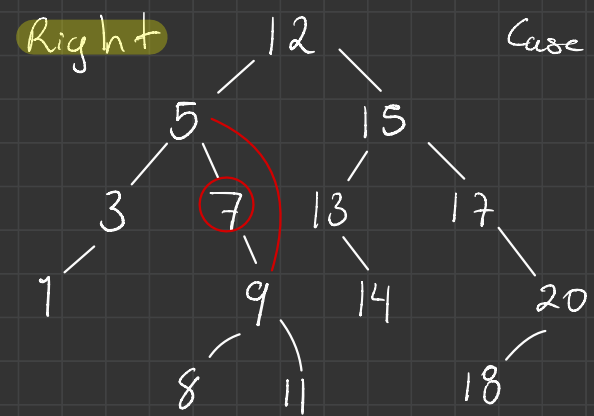
Case 0) Leaf Node

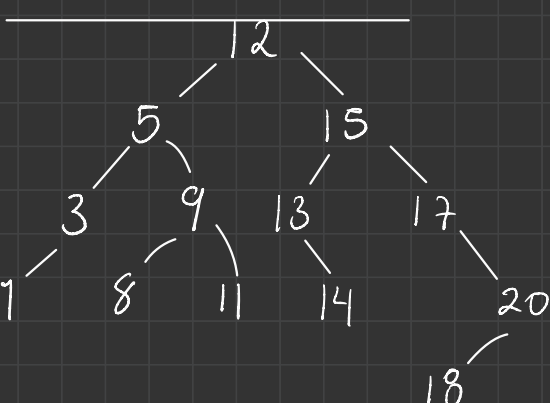
Just delete the node, but ensure that the pointer to that node is now (nullptr) to avoid unexpected behaviors



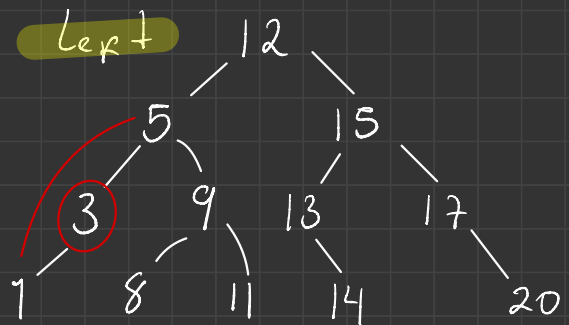
Case 1) the node to delete has one child : Right or Left

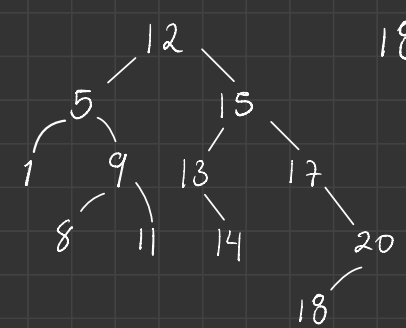
Right



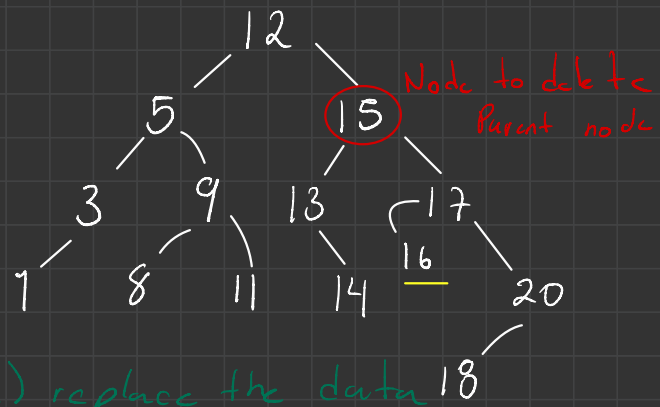


Left





Cuse 2, when is a parent node:



soluing:

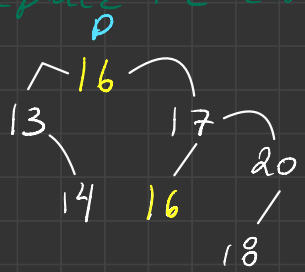
Find the min of the right node to delete

1) From the node to delete:

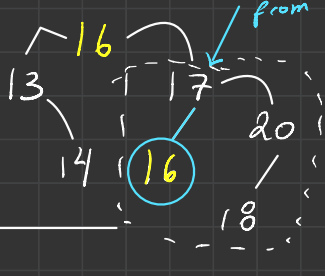
Right - left

bill reach to the last node

2) replace the data

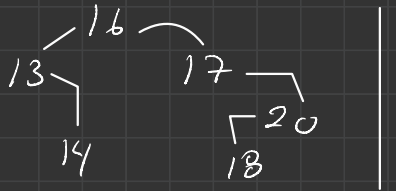


3) from P->Rchild, search and delete the node that you have replaced its data



the data is found and perform a deletion of a leaf node

Since many things could happend when we are deleting a node any result that returs with recursion power, we need to be sure to P(16)->Rchilds, points to the subtree generated by the recursion call.

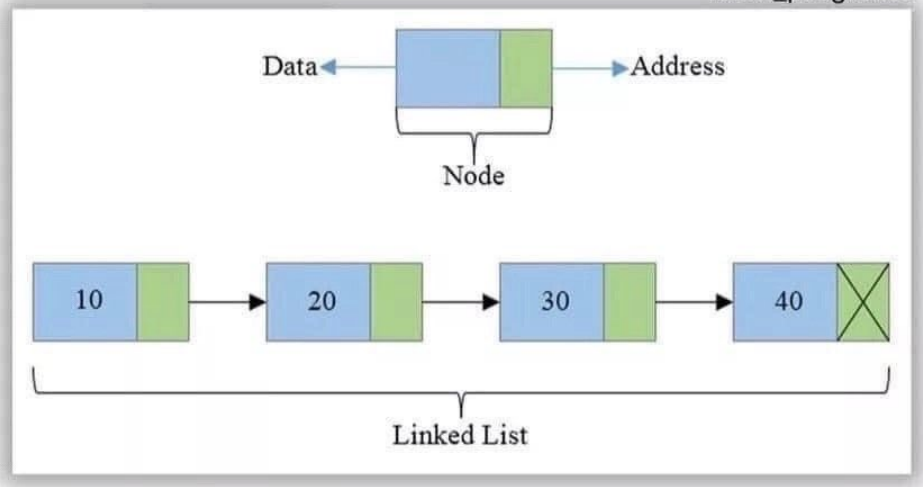


final sub tree

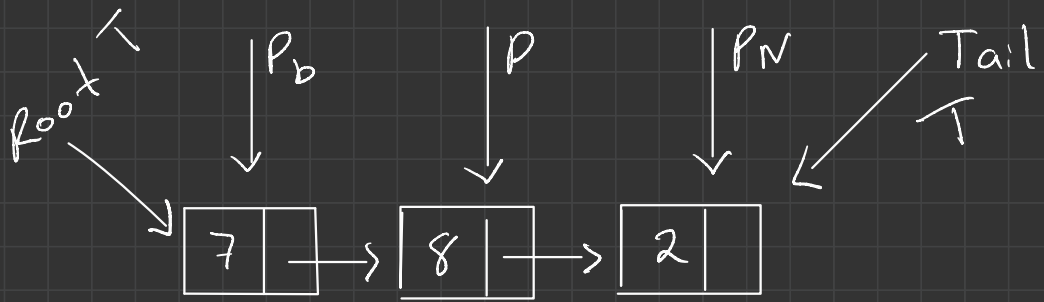
Linked List

A Linked List is a linear data structure, in which the elements are not stored at contiguous memory locations. It consist of nodes where each node contains a data field and a link to the next node in the list.

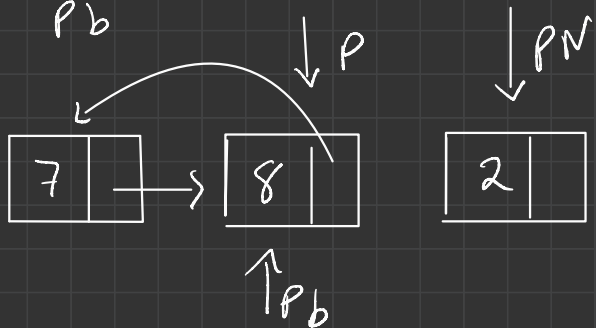
learn_programo



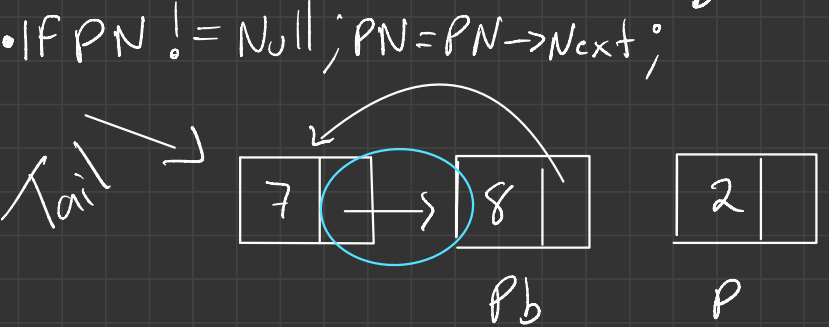
Reverse a linked list:



- Make P point to Pb
- Pb point to P



- P = PN
- IF PN != Null; PN = PN->Next;

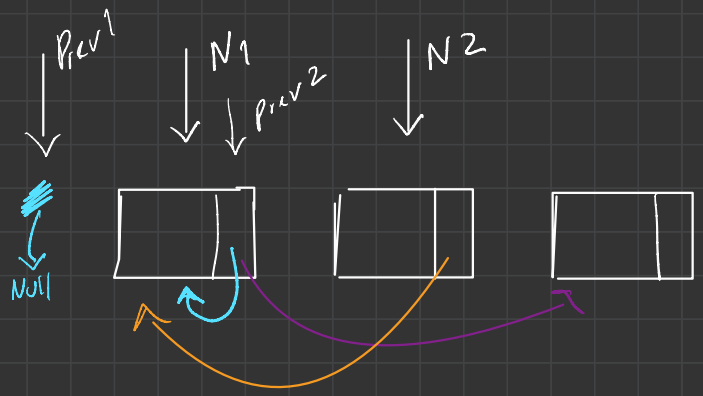



```
While (P != Null ptr){
    P->Next = Pb
    Pb = P;
    P = PNext
    If (PNext != Null){PNext = PNext->Next;}
Tail = T.Root;   Tail->Next = Null
Root = Pb
```

In some point you'll do a step where is null asure to do just one time Null asignation

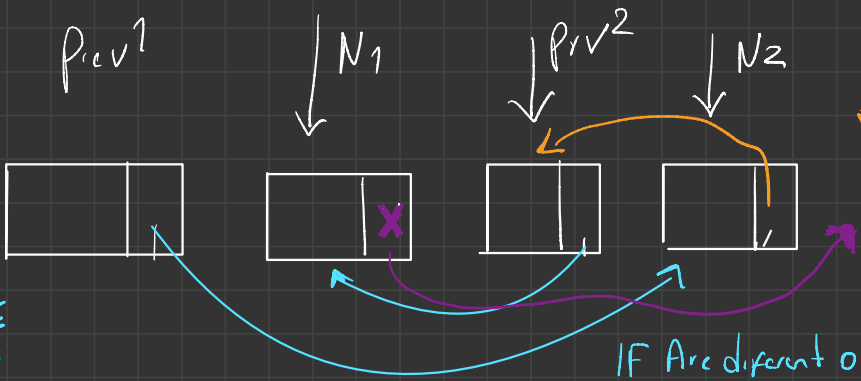
Swap two Nodes.

// Case when the nodes are beside the other



tem = N1

```
// Link previous of node1 with node2
if (prev1 != NULL)
    prev1->next = node2;

// Link previous of node2 with node1
if (prev2 != NULL)
    prev2->next = node1;

// Swap node1 and node2 by swapping their
// next node links
temp        = node1->next;
node1->next = node2->next;
node2->next = temp;

// Make sure to swap head node when swapping
// first element.
if (prev1 == NULL)
    head = node2;
else if (prev2 == NULL)
    head = node1;
```



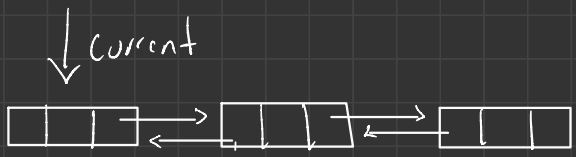
temp = Prv2

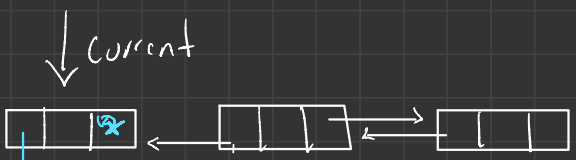
IF Are diferent of null you can link

IF (N1 == Root) Root = N2;

If (N2 == Tail) Tail = N1;

Doubly linked list: Reverse

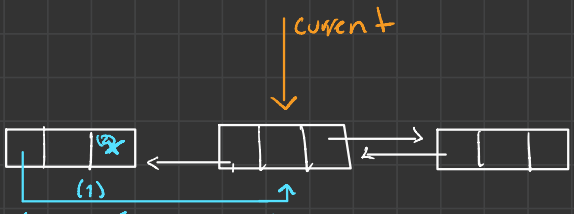




Swap the pointers

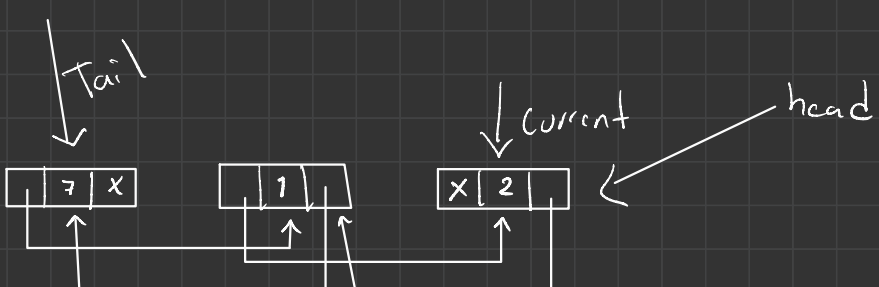
Reverse the pointers of the current Node

Current = current->prev;

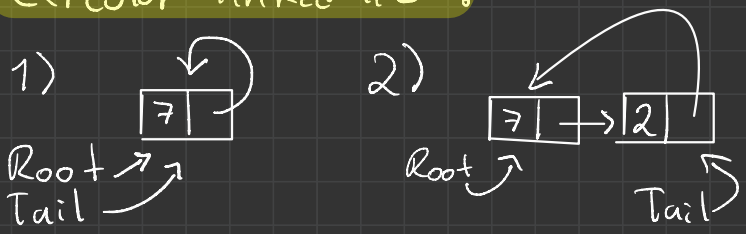


Swap the pointers

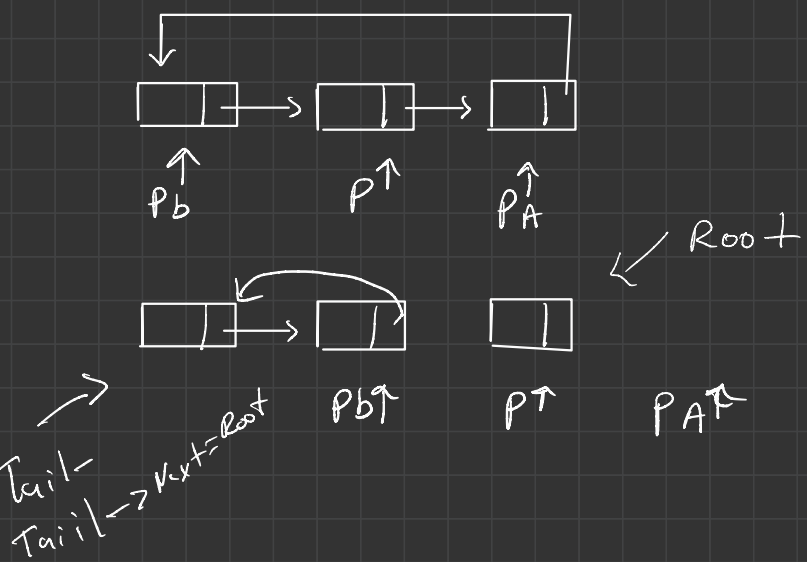
- Repeat until current = Null
- Update head and Tail



Circular linked list;



Reverse list;



- Make the case when are only two nodes to reverse