Level Order Traversal

* It is a BFS Traversal
  ↳ Breadth First Search (goes level By level)
  ⇓
  from left to right

* Uses queue data structure to keep track of nodes

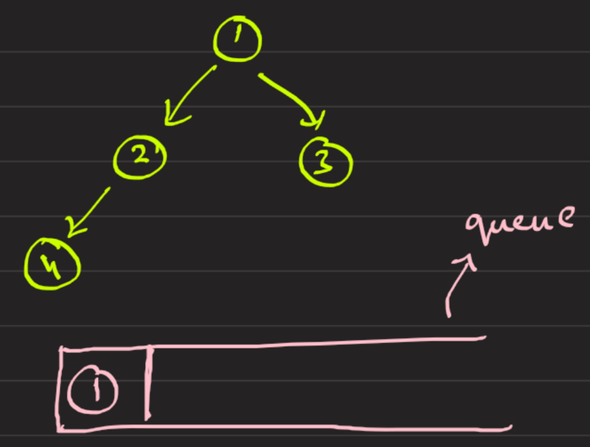
Steps Involved :

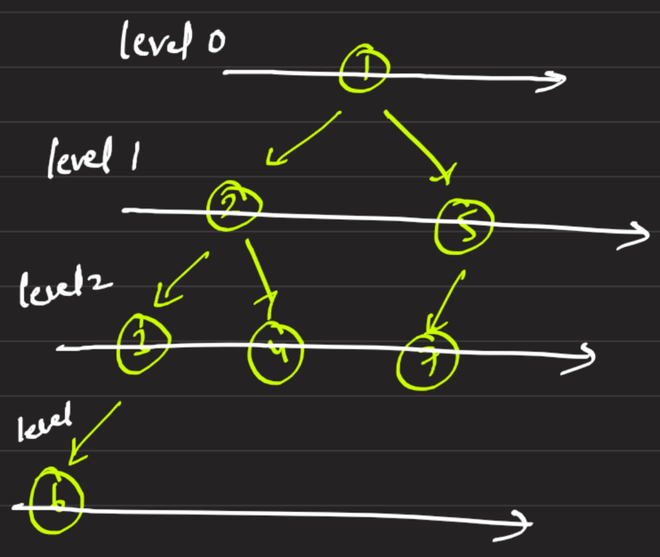
① add root to the queue

② Iterate till queue is not Empty

  ②-① while iterating

  * Remove node from queue
  * Print the node Value
  * Check if (left) Subtree is there for root
    ↳ if yes, add that to queue
  * Check if (right) Subtree is there for root
    ↳ if yes, add that to queue



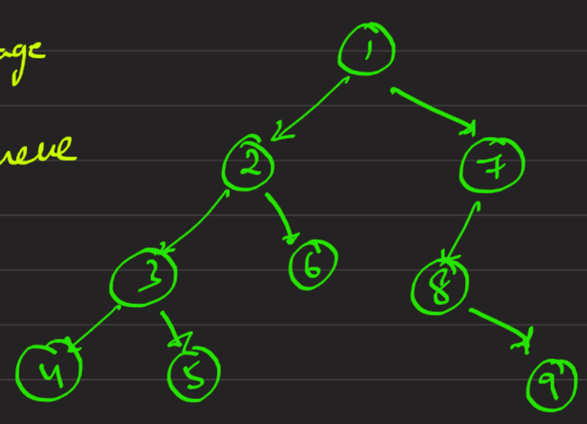


Output : 1 2 5 3 4 7 6

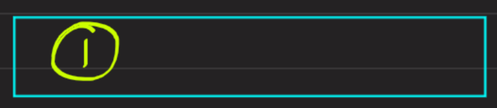
Dry Run

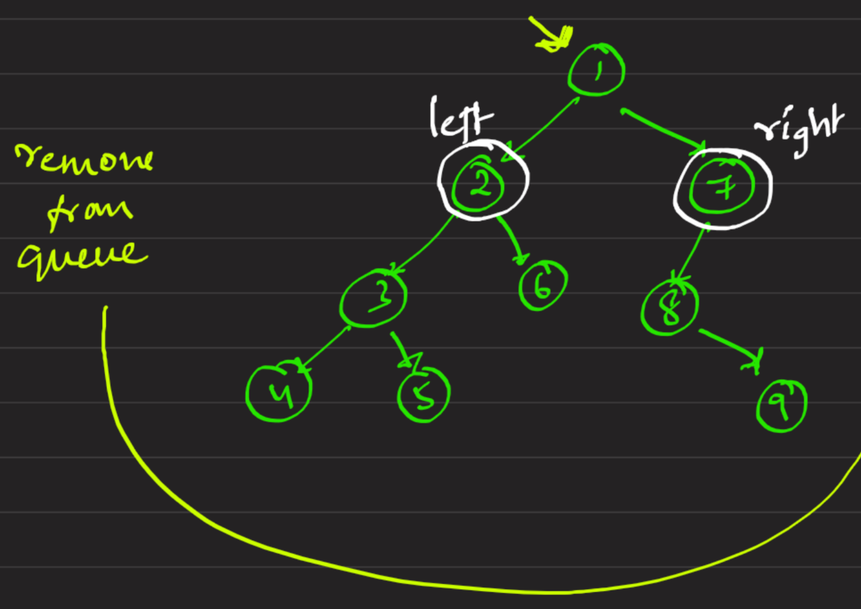
Initial Stage

Add to queue

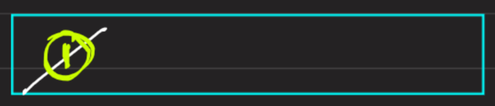


queue





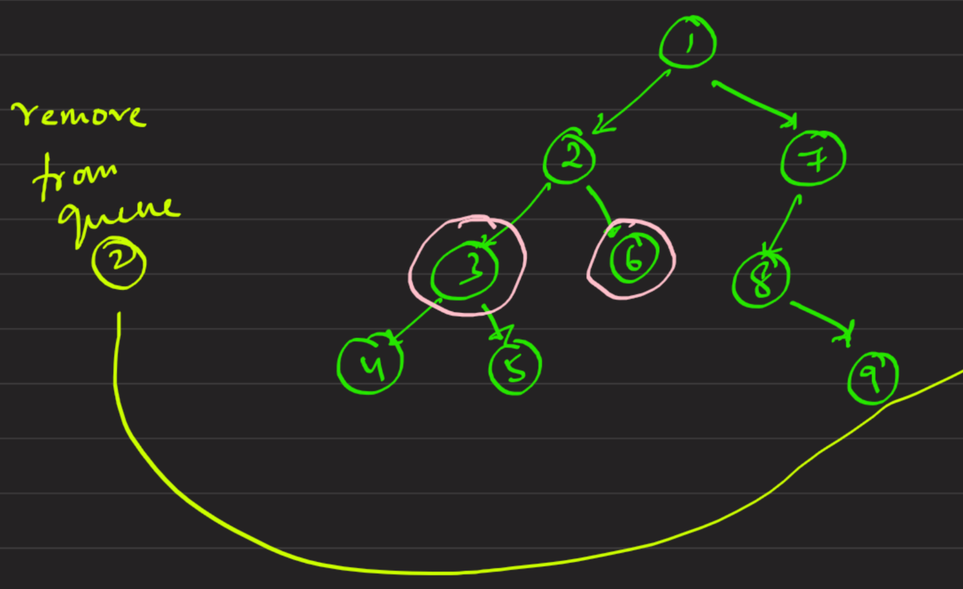
queue



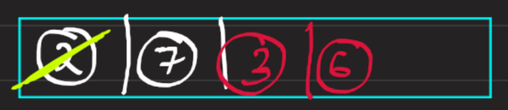
then, iterate till is not Empty

* if node ① has left, add to queue

* if node ① has right, add to queue

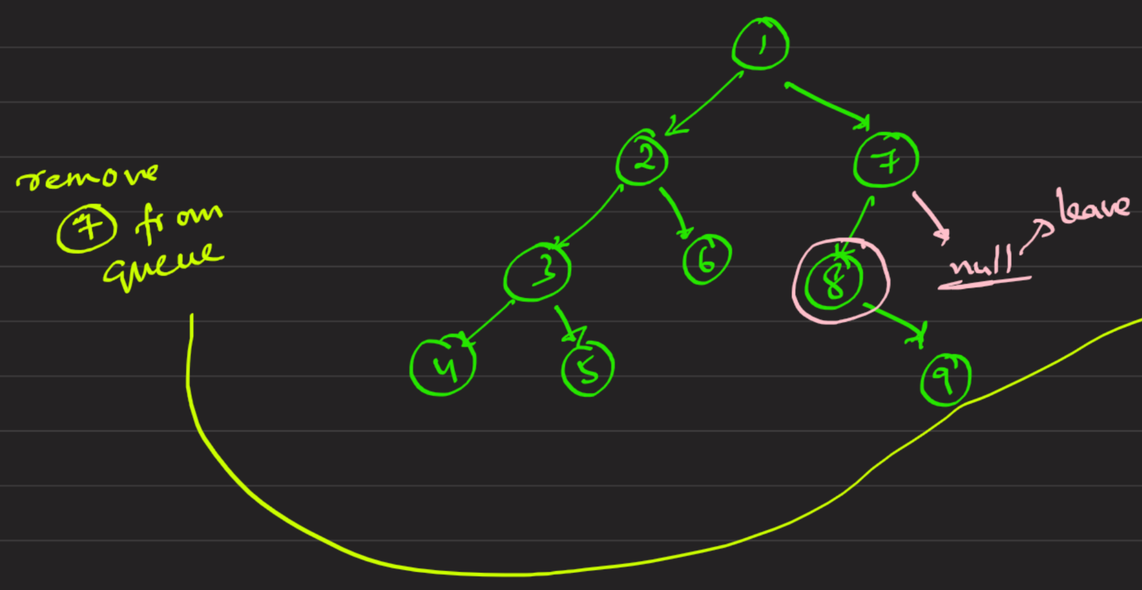


queue



Add left of ②, if contains

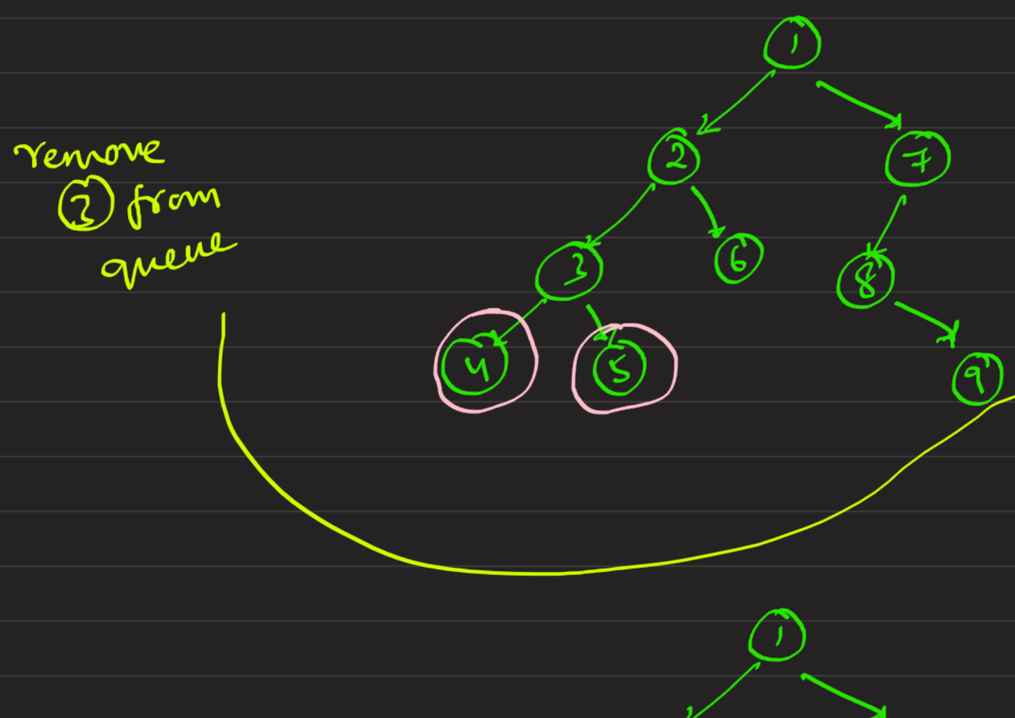
Add right of ②, if contains



queue

Add left of ⑦, if contains

Add right of ⑦, if contains

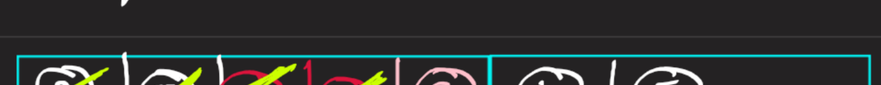


queue

Add left of ③, if contains

Add right of ③, if contains

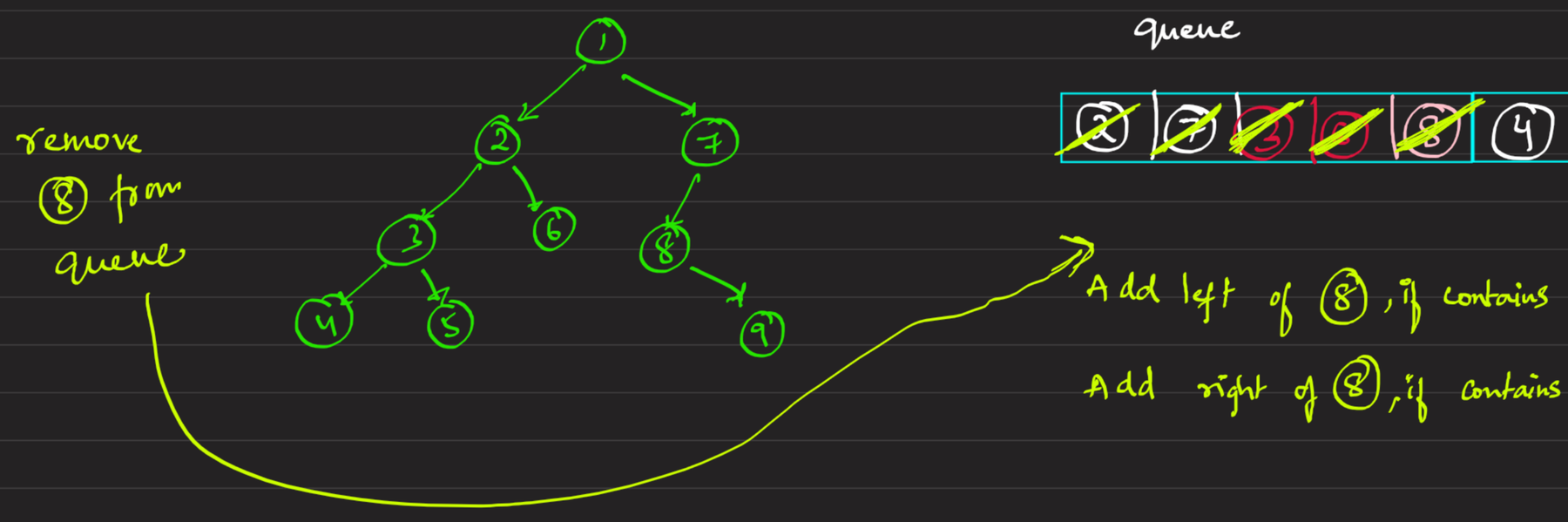
queue



Add left of ⑥, if contains → null

Add right of ⑥, if contains → null

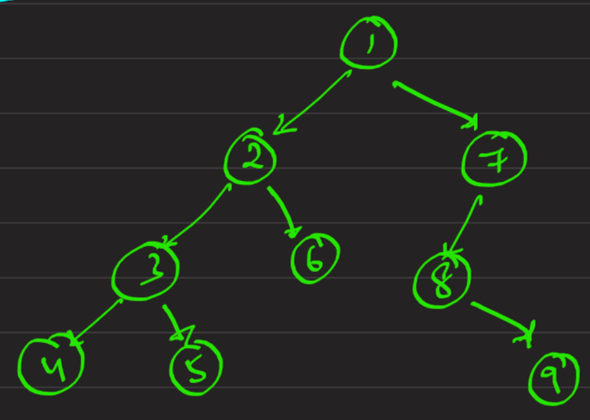
Skip

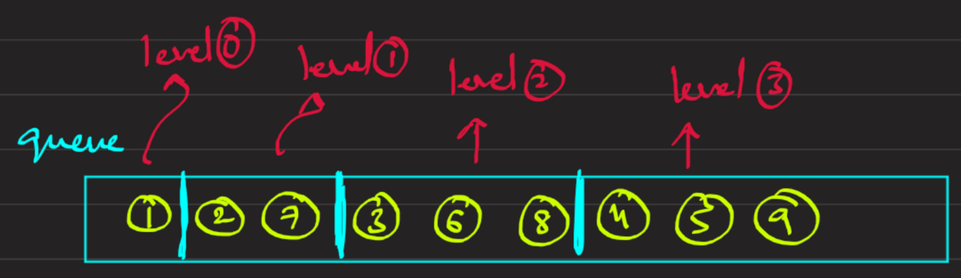


* from hereon, ④, ⑤, ⑨ will remove from queue one by one
but, they don't have have childrens, So queue never going to increase

New addings won't happen

If you observe,





* go back & check every iteration,

This will be Our Sequences of nodes

* This happened because of level wise traversal
* We are dealing level-By-level, so we are all Same level nodes at one iteration

Pseudo Code

```
Void levelOrder (Node root)
{
    if (root == null) return;          // we add nodes, not values, then only we can access, left & right nodes
    Queue<Node> q = new LinkedList<>();
    q.add(root);
    while(q.size() >0)                 // Assume, rem is our removed node from queue
    {
        Node rem = q.remove();
        S.O.p(rem.val + " ");
        if (rem.left != null) q.add(rem.left);
        if (rem.right != null) q.add(rem.right);   // if Contains add to queue check
    }
}
```

And, for above code, we get Output as:

output : 1 2 7 3 6 8 4 5 9

But, we want it to be in:

1

2 7

3 6 8

4 5 9

level - By - level → How to print like this??

* just observe a case,

At first, we have ① → we only remove 1st level nodes

then we add ②,⑦ → we only remove 2nd level nodes

then, we add ④,⑤,⑨ → we only remove 3rd level nodes

like wise...,

* if you modify our previous code to:

Pseudo Code

```
void levelOrder (Node root)
{
    if (root == null) return;
    Queue<Node> q = new LinkedList<>();
    q.add(root);
    while (q.size() > 0)
    {
        int n = q.size();
        for (int i=0; i<n; i++)
        {
            Node rem = q.remove();
            S.O.p(rem.val + " ");
            if (rem.left != null) q.add(rem.left);
            if (rem.right != null) q.add(rem.right);
        }
    }
}
```

Initially, root node ①, for that n=1 → inner loop runs → 1 time ⇒ level ①

then 2 nodes added → q.size() = 2, for that n = 2 → inner loop runs → 2 times ⇒ level ②

↓

goes on level by level