Tree

In [ ]:

What is a tree

Binary Tree

- Node, Root, Leaf
- Sub tree
- Calculate number of nodes in a perfect binary tree
- Calculate min height, given number of nodes

In [ ]:

Binary and n-ary trees

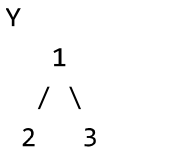
Skew Tree

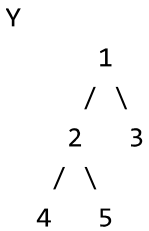
- Depth/height of skew tree
- Worst case complexity ?

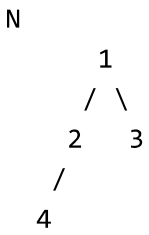
Representation of a Tree

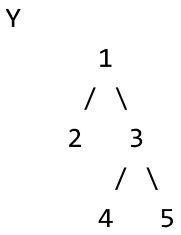
- Linked
- Array

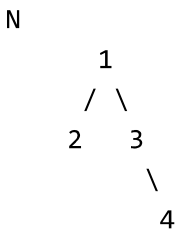
Full binary tree: every node has either 0 or 2 children



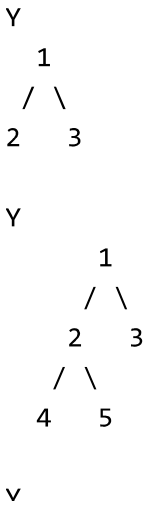








Complete binary tree: every level, except possibly the last is completely filled. Last level nodes are filled from the left to right. Can be represented using arrays. Ex: binary heap.



Max Number of nodes in a tree

**Linked representation of tree **

C++

Type *Markdown* and LaTeX: α^2

Height vs Depth

- Height: measured bottom up: Height of leaf node=0
- Depth: measured top to bottom: Depth of root node=0

Level=Depth (can start from 0/1)

Finding max depth of binary tree using recursion
https://leetcode.com/problems/maximum-depth-of-binary-tree/
(https://leetcode.com/problems/maximum-depth-of-binary-tree/)

In [ ]:
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (root == NULL) {
            return 0;
        }
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
};
```

In [ ]:
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

# BFS with dummy node
import queue
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if root is None:
            return 0

        count = 0
        q = queue.Queue()
        q.put(root)
        q.put(None)

        while not q.empty():
            curr  = q.get()

            if curr is None:
                count += 1
                if not q.empty():
                    q.put(None)
            else:
                if curr.left:
                    q.put(curr.left)
                if curr.right:
                    q.put(curr.right)

        return count
```

In [ ]:
```python
# BFS Without Sentinel or dummy node
import queue
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if root is None:
            return 0

        q = queue.Queue()
        q.put(root)

        count = 1
        levels = 0

        while count > 0:
            t = 0
            levels += 1

            for i in range(count):
                curr  = q.get()
                if curr.left:
                    q.put(curr.left)
                    t += 1
                if curr.right:
                    q.put(curr.right)
                    t += 1

            count = t

        return levels
```

DIY

https://leetcode.com/problems/minimum-depth-of-binary-tree/ (https://leetcode.com/problems/minimum-depth-of-binary-tree/)

In [ ]:
```java
class Solution {
    public int minDepth(TreeNode root) {

        if (root == null) {
            return 0;
        }
        if (root.left  null && root.right  null) {
            return 1;
        }
        if (root.left == null) {
            return 1 + minDepth(root.right);
        }
        if (root.right == null) {
            return 1 + minDepth(root.left);
        }
        return Math.min(minDepth(root.left), minDepth(root.right)) + 1;
    }
}
```

In [ ]:

https://leetcode.com/problems/maximum-width-of-binary-tree/ (https://leetcode.com/problems/maximum-width-of-binary-tree/)

In [ ]:
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

import queue
class Solution:
    def widthOfBinaryTree(self, root: Optional[TreeNode]) -> int:
        if root is None:
            return 0

        q = queue.Queue()
        q.put(root)

        count = 1
        max_width = 0
        has_next_level = True

        while has_next_level:
            t = 0
            width = 0

            has_node_at_current_level = False
            has_next_level = False
            for i in range(count):
                curr  = q.get()

                if curr is not None or has_node_at_current_level:
                    width += 1

                if curr is not None:
                    has_node_at_current_level = True
                    max_width = max([width, max_width])

                t += 2
                if curr is not None and curr.left:
                    has_next_level = True
                    q.put(curr.left)
                else:
                    q.put(None)

                if curr is not None and curr.right:
                    has_next_level = True
                    q.put(curr.right)
                else:
                    q.put(None)

            count = t

        return max_width
```

In [ ]:
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

import queue
class Solution:
    def widthOfBinaryTree(self, root: Optional[TreeNode]) -> int:

        depths = {}

        self.widthOfBinaryTreeUtil(root, 0, 1, depths)
        print(depths)

        max_width = 0
        for l, r in depths.values():
            max_width = max(max_width, (r-l+1))

        return max_width

    def widthOfBinaryTreeUtil(self, root, level, curr, depths):
        if root is None:
            return

        if level not in depths:
            depths[level] = [curr, curr]
        else:
            l, r = depths[level]
            depths[level] = [min(l, curr), max(l, curr)]

        self.widthOfBinaryTreeUtil(root.left, level+1, curr*2-1, depths)
        self.widthOfBinaryTreeUtil(root.right, level+1, curr*2, depths)
```

In [ ]:
```python

```

Question

https://leetcode.com/problems/symmetric-tree/ (https://leetcode.com/problems/symmetric-tree/)

DIY

https://leetcode.com/problems/same-tree/ (https://leetcode.com/problems/same-tree/)

In [ ]:
```python

```

Operations in a Tree

- Add
- Remove
- Traverse
- Search

In [ ]:

Traversal of a tree

Depth First Traversal, DFS = Stack

- rLR
- LrR
- LRr

Recursive implementation of DFS

In [ ]:

Breadth First / Level Order Traversal, BFS = Queue

In [ ]:

Question

https://leetcode.com/problems/path-sum-ii/ (https://leetcode.com/problems/path-sum-ii/)

In [ ]:

Right view:
https://leetcode.com/problems/binary-tree-right-side-view/
(https://leetcode.com/problems/binary-tree-right-side-view/)

Solution-1 BFS:

1. In BFS solution keep additional Node pointer
2. Update the pointer each time we pop from queue (update when not null)
3. When we get the level change marker i.e. NULL, update the result with the value in pointer.

Solution-2 DFS: 1. Preorder 2. Inorder 3. PostOrder

rRL

Top View
https://www.hackerrank.com/challenges/tree-top-view/problem
(https://www.hackerrank.com/challenges/tree-top-view/problem)

Bottom view
https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1
(https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1)

In [ ]:
```
https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/
```

In [ ]:

In [ ]:

In [4]:
```
import queue

class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right


def preorder(root):
    if root is None:
        return
    print(root.data)
    preorder(root.left)
    preorder(root.right)

def inorder(root):
    if root is None:
        return
    inorder(root.left)
    print(root.data)
    inorder(root.right)

def postorder(root):
    if root is None:
        return
    postorder(root.left)
    postorder(root.right)
    print(root.data)

def bfs(root):
    if root is None:
        return

    q = queue.Queue()
    q.put(root)

    while(not q.empty()):
        curr = q.get()
        print(curr.data)
        if curr.left:
            q.put(curr.left)
        if curr.right:
            q.put(curr.right)

#       1
#    2     3
#  4   5
#
#
n4 = Node(4)
n5 = Node(5)
n2 = Node(2, n4, n5)
n3 = Node(3)
n1 = Node(1, n2, n3)

preorder(n1)
```

```
print()
inorder(n1)
print()
postorder(n1)
print()
bfs(n1)
```

```
1
2
4
5
3

4
2
5
1
3

4
5
2
3
1

1
2
3
4
5
```