

Tree

Non linear data structure, composed of nodes.
Where nodes have a parent , child relationship.

What is a tree

Binary Tree

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

In []:

1

Binary and n-ary trees

Skew Tree

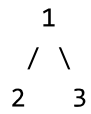
- Depth/height of skew tree
- Worst case complexity ?

Representation of a Tree

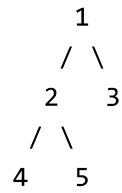
- Linked
- Array

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

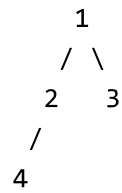
Y



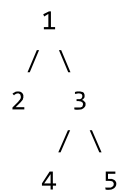
Y



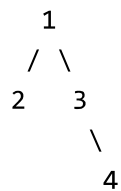
N



Y



N



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.

Y

```

    1
   / \
  2   3

```

Y

```

        1
       / \
      2   3
     / \
    4   5

```

Max Number of nodes in a tree

**Linked representation of tree **

C++

```

struct Node {
    int data;
    Node * left;
    Node * right;
};

```

JAVA

```

class Node {
    public int data;
    public Node left;
    public Node right;

    public Node(int data, Node left=null, Node right=null) {
        this.data= data;
        this.left = left;
        this.righ = right;
    }
}

```

PYTHON

```

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

```

Type *Markdown* and LaTeX: α^2

In []:

1

In []:

```

1  ```C++
2
3  struct Node {
4      int data;
5      Node *left;
6      Node *right;
7  };
8
9  Node* newNode(int data, Node* left=NULL, Node* right=NULL) {
10     Node *temp = new Node;
11
12     temp->data = data;
13     temp->left = left;
14     temp->right = right;
15
16     return temp;
17 }
18
19 // rLR - recursive
20 void printTree(Node *root) {
21     if (root == NULL) return;
22
23     cout << root->data << " ";
24     printTree(root->left);
25     printTree(root->right);
26 }
27
28 int main() {
29
30     Node *n1 = newNode(20);
31     //     cout << n1->data << " l=" << n1->left << " r=" << n1->right;
32
33     Node *n2 = newNode(30);
34     Node *n3 = newNode(10, n1, n2);
35
36     10
37     /  \
38     20  30
39
40
41     printTree(n3);
42 }
43
44 ```

```

```

1  **Pre-Order traversal of tree**
2  ```C++
3
4  struct Node {

```

```
5     int data;
6     Node *left;
7     Node *right;
8 };
9
10 Node* newNode(int data, Node* left=NULL, Node* right=NULL) {
11     Node *temp = new Node;
12
13     temp->data = data;
14     temp->left = left;
15     temp->right = right;
16
17     return temp;
18 }
19
20 // rLR - recursive
21 void preOrder(Node *root) {
22     if (root == NULL) return;
23
24     cout << root->data << " ";
25     preOrder(root->left);
26     preOrder(root->right);
27 }
28
29 // LrR - recursive
30 void inOrder(Node *root) {
31     if (root == NULL) return;
32
33     inOrder(root->left);
34     cout << root->data << " ";
35     inOrder(root->right);
36 }
37
38 // LRr - recursive
39 void postOrder(Node *root) {
40     if (root == NULL) return;
41
42     postOrder(root->left);
43     postOrder(root->right);
44     cout << root->data << " ";
45 }
46
47 int main() {
48     // LRr
49     //      10
50     //    /  \
51     //   20  30
52     //  /  \
53     // 40  50
54     //      \
55     //       60
56
57     // inOrder
58     // postOrder
59     Node *root = newNode(10, newNode(20, newNode(40), newNode(50, NULL,
60     newNode(60))), newNode(30));
61     preOrder(root);
```

```

61 }
62 /*
63                                     preOrder(10)
64                                     /      \
65                               preOrder(20)
66                               /      \
67         preOrder(30)                preOrder(50)
68         /      \                    /      \
69     preOrder(NULL) preOrder(NULL) preOrder(NULL) preOrder(60)
70                 /      \                /      \
71             preOrder(NULL) preOrder(NULL) preOrder(NULL)
72
73     pre - 10 20 40 50 60 30
74     in  - 40 20 50 60 10 30
75     post- 40 60 50 20 30 10
76 */
77 ...

```

In []:

1

Types of traversal of a tree

Depth order:

- rLR : pre order
- LrR : in order
- LRR : post order

TC: $O(N)$ SC : $O(\text{depth of tree})$

Level Order

- L->R (default)
- R->L

TC: $O(N)$

SC: ?

In []:

```

1 DFS = Depth First Search -> Stack
2 BFS = Breadth First Search -> Queue

```

In []:

1

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/>)

DIY

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

In []:

1

In []:

1

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 []:

1

Operations in a Tree

- Add
- Remove
- Traverse
- Search

In []:

1

Traversal of a tree

Depth First Traversal, DFS = Stack

- rLR
- LrR
- LRR

Recursive implementation of DFS

In []:

1

Breadth First / Level Order Traversal, BFS = Queue

In []:

1

In []:

1

In []:

1