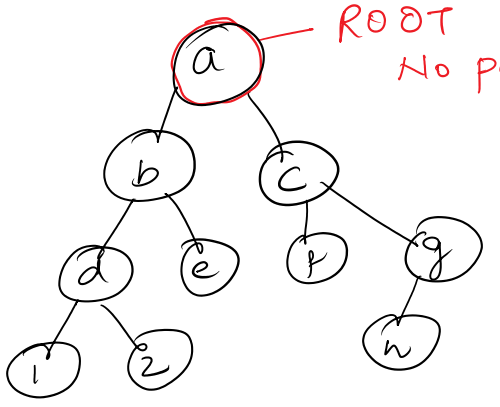
Trees — Fundamental DS

Specific Graph SIMPLER

- Vertices: Node Information
- Edges: Link b/w nodes

#g.



ROOT
No parent

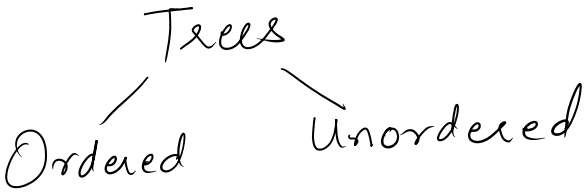
'a' is the parent of 'b'

'b' is the child of 'a'

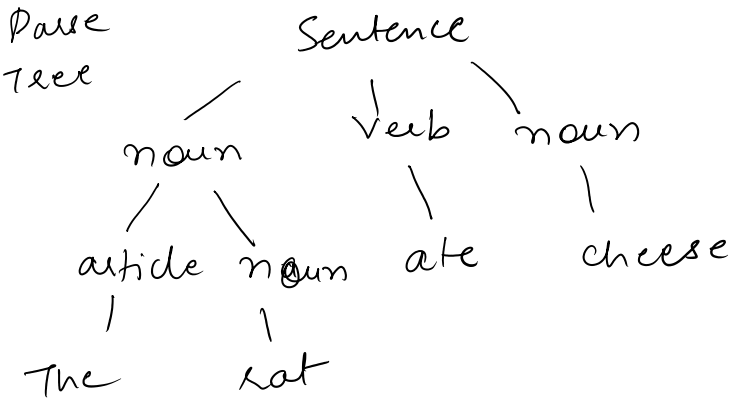
'b' and 'c' are siblings (common parent)

1. Path - List of vertices
   a → 2 = a b d 2
2. Leaf - Node with no children / external / terminal
3. Non-Leaf - Nodes with atleast 1 child

- Depth/Height - Length of the longest path from root to any leaf.
- Subtree - Any given node with all its children



Parse Tree



Specific order for the children

no criteria for ordering

- M-ary tree - This must have specific number of children (m) in a specific order

Binary Tree - Atmost 2 children

- A node has

atmost one edge leading to it

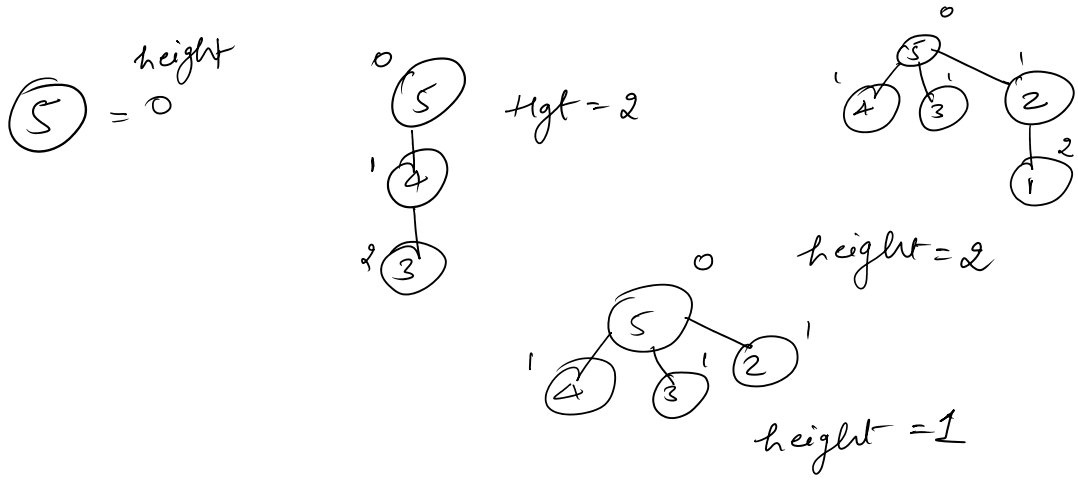
2. A node has exactly 1 parent
3. Atmost one path between any 2 nodes
4. Exactly one path from root to a leaf.

1. Level - root = 0

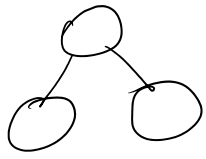
Level(Node) = Level(parent) + 1

Height / Depth = Maximum level of tree's nodes

#

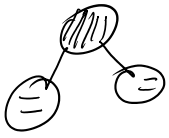


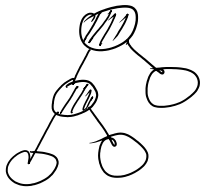
BINARY TREES

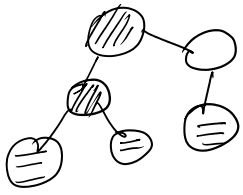


- Internal Node - Node has 2 children

- External Node - Node has no children
  Ni /// Ne ≡

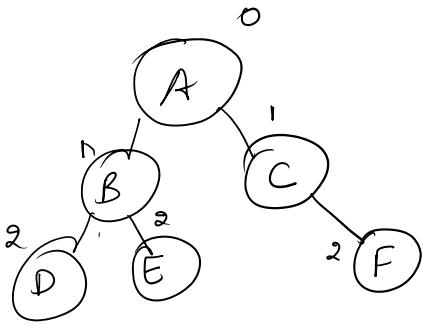
#g  Ni = 1, Ne = 2

 Ni = 2, Ne = 3.

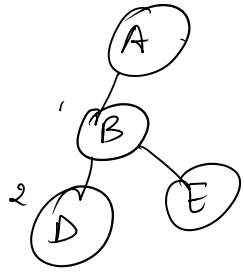
 Ni = 2, Ne = 3

N - internal nodes
N + 1 — external nodes

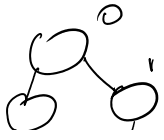
- Balanced Binary Tree
  Each node the difference between depth of left & right subtree is maximum 1.

#g 

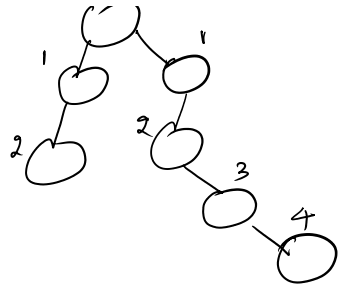
balanced



Unbalanced

#g  height = 4

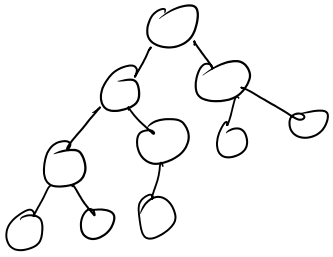
#g



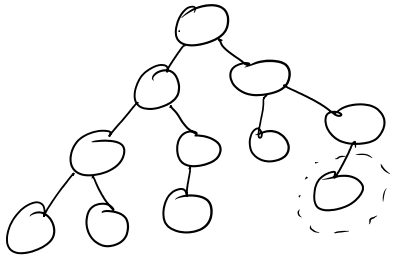
height = 4

- Complete Binary Tree
  All leaves must be placed as far to the left as possible. Leaves should be filled from left to right.

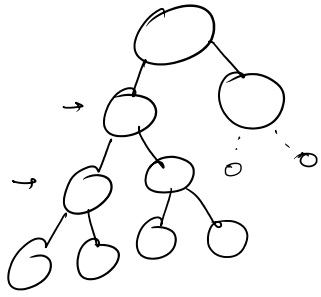
#g



C

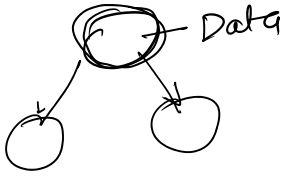


IC

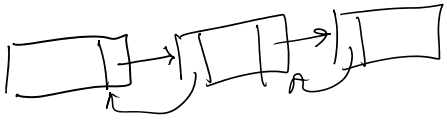


IC

Binary Trees




```
Struct TreeNode {
  TreeNode * left;
  TreeNode * right;
  Data * data
}
```



```
StructNode LLNode {
  LLnode * next;
}
```

```
    LLnode * prev;
    Data * data
};
```

Binary Tree — Recursive Algorithms

Base Recursive Call

```
Struct Node {
    Node * left;
    Node * right;
    int data
};
```

```
Node * MakeNode (int D)
{
    Node * node = new Node();
    node -> data = D;
    node -> left = NULL;
    node -> right = NULL;
    return node
}
```


```
typedef Node * Tree
```

```
void FreeNode (Node * node)
{
    delete node;
}
```

1. Number of nodes

```
int NodeCount (Tree -> Node tree)
{
    if (tree == NULL) return 0;
```

```
else
    return  1 + NodeCount (tree -> left)
              + NodeCount (tree -> right)
}
```

2. Height of the tree

```
int Height ( Tree tree)
{
    if (tree == 0) return -1;

    int HL = Height ( tree -> left) // left subtree
    int HR = Height (tree -> right) // right subtree

    return 1 + max ( HL, HR)
}
```

3. Binary Tree Traversal

PreOrder

Visit the Node

Traverse Left Subtree

Traverse Right Subtree

InOrder

Traverse LeftSubtree

Visit the Node

Traverse Right Subtree

Post Order

Traverse Left Subtree

Traverse Right Subtree

Visit the node