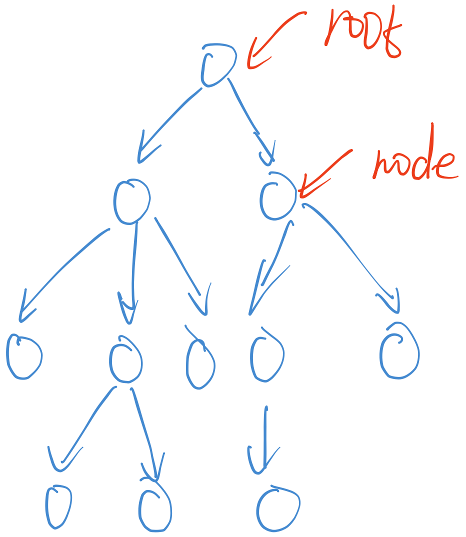
Introduction to Tree

- often to show hierarchy

- Defination: a collection of entities called nodes



root → no parent

children

parent

sibling → same parent

leaf → no children

↕(oppsite)

internal node → has children

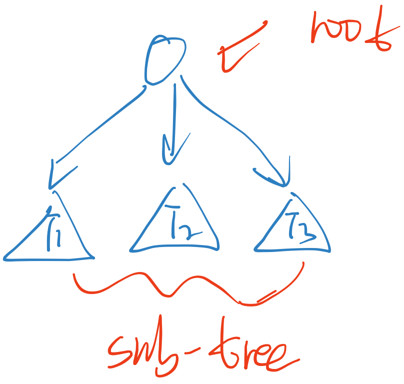
A is ancestor of B

B is descendent of A

<=>

could go from node A to node B.

- Recursive data structure



N nodes

N-1 edges

Depth: number of edges in path from root to $\underline{x}$

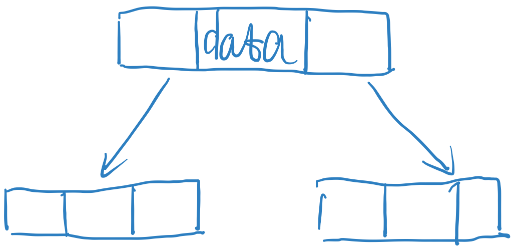
Height: number of edges in longest path from x to a leaf

Height of tree: Height of root node

Height of an empty tree = -1

Heights of tree with 1 node = 0

Binary Tree: a tree in which each node can have at most 2 children



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

Strict/Proper binary tree

↳ each node can have either 2 or 0 children

Complete Binary tree

↳ all levels except possibly the last are completely filled and all nodes are as left as possible

Perfect Binary Tree

↳ maximum nodes = $2^{h+1} - 1$

Height of PBT $= \log_2(n+1) - 1$

Height of CBT $= \lfloor \log_2 n \rfloor$

Balanced binary Tree

↳ Difference between height of left and right subtree for every node is not more than k (mostly 1)

$$\text{Diff} = |h_{left} - h_{right}|$$

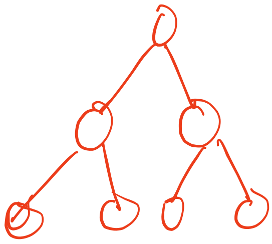
Zmplement :

a) dynamically created nodes

b) array (particularly use in CBT)

↳

L1	L2L	L2r	L3L	L3r	L3L	L3r
0	1	2	3	4	5	6



for node at index i:

left-chirdren - index = 2i+1

right ... = 2i+2

BST - Implement in C/C++

Define tree structure: int data node* left&right nodes

Insert

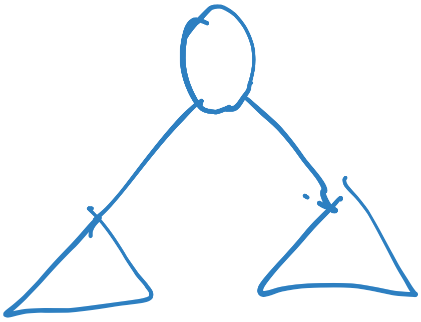
Search

Binary Search Tree

	Array (unsorted)	Linked List	Array (sorted)	BST (balanced)
search(x)	O(n)	O(n)	O(logn)	O(logn)
insert(x)	O(1)	O(1)	O(n)	O(logn)
Remove(x)	O(n)	O(n)	O(n)	O(logn)

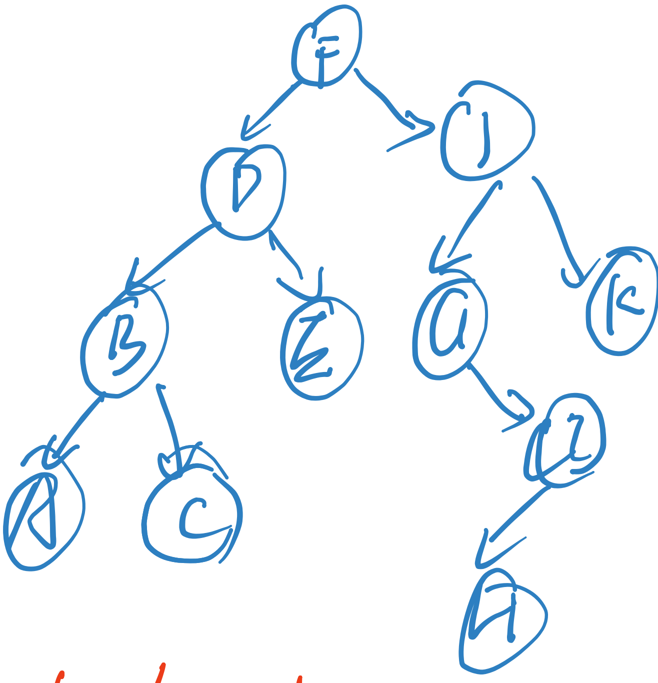
BTS

↳ a binary tree in which for each node, value of all the nodes in left subtree is lesser and value of all the nodes in right subtree is greater



$\{x|x \in left\} < Node < \{x|x \in Right\}$

Tree Traversal



Breadth-first level-order

F, D, J, B, E, G, K, A, C, I, H

Depth-first

Data left right

<root><left><right> — Preorder DLR

<left><root><right> Inorder LDR

<left><right><root> postorder LRD

for BST, Inorder traversal will get a sorted list.