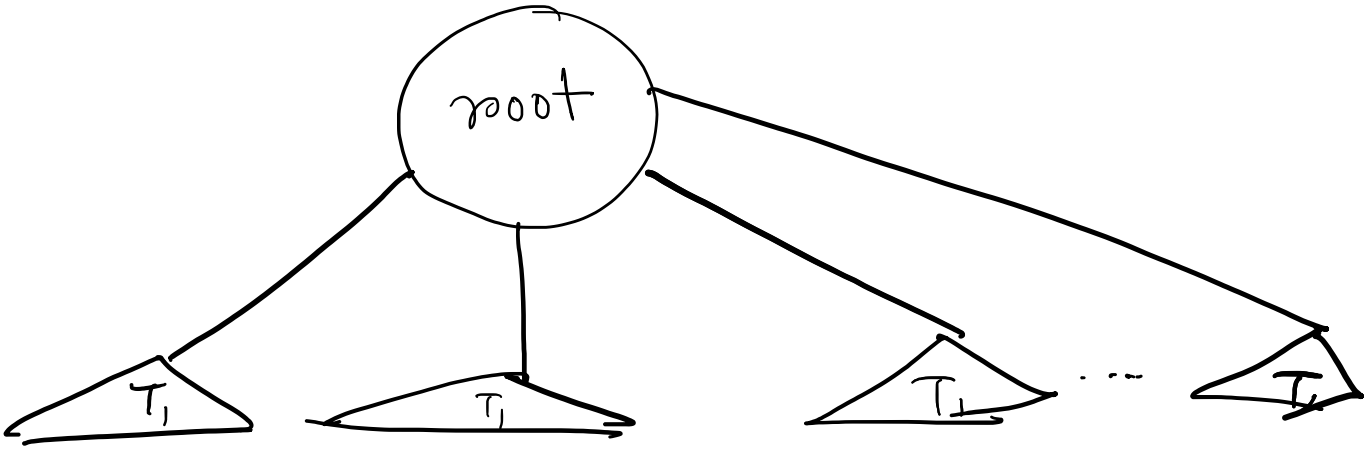
Chapter 4

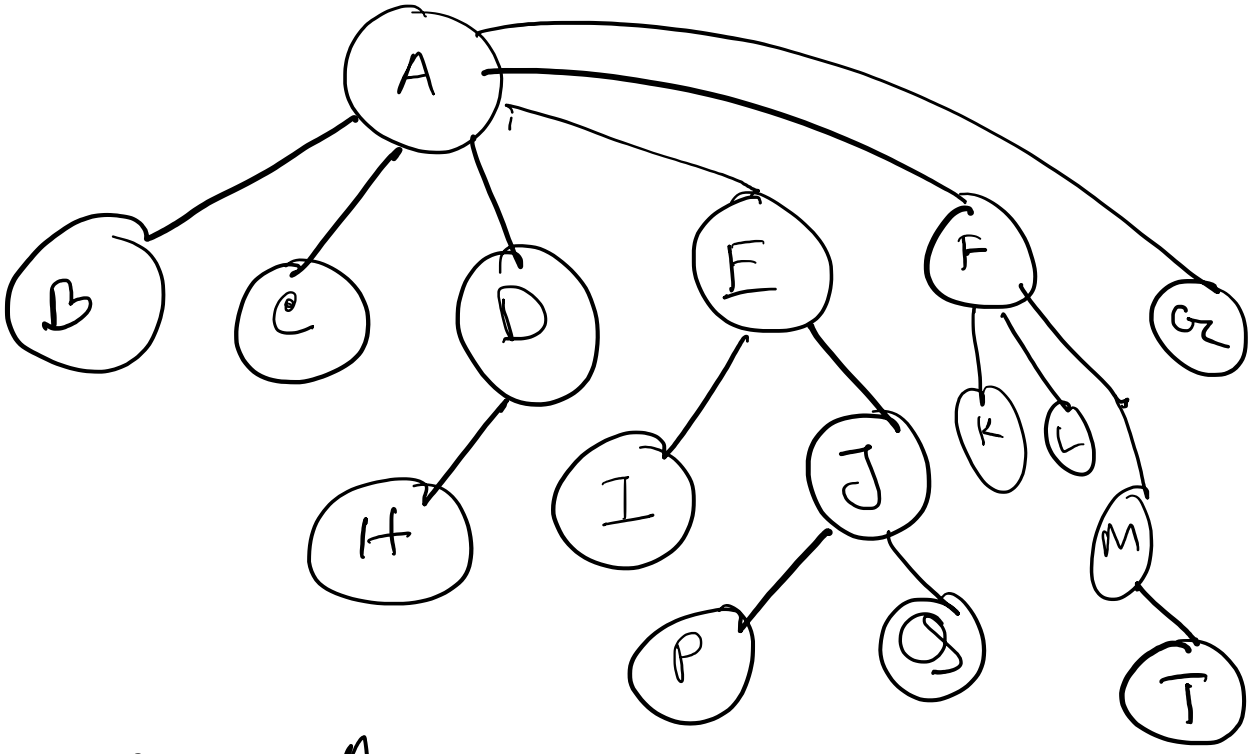
Tree



Collection of nodes and edges.

Recursively,

a root and its subtrees



Parent (D) = A

children (D) = {H}

children (E) = {I, J}

Ancestors (J) = {E, A}

Descendents (F) = {K, L, M, T}

Siblings (B) = {C, D, E, F, G}

Definitions

→ **Leaf node:** node having no children

→ **size:** total number of nodes in tree

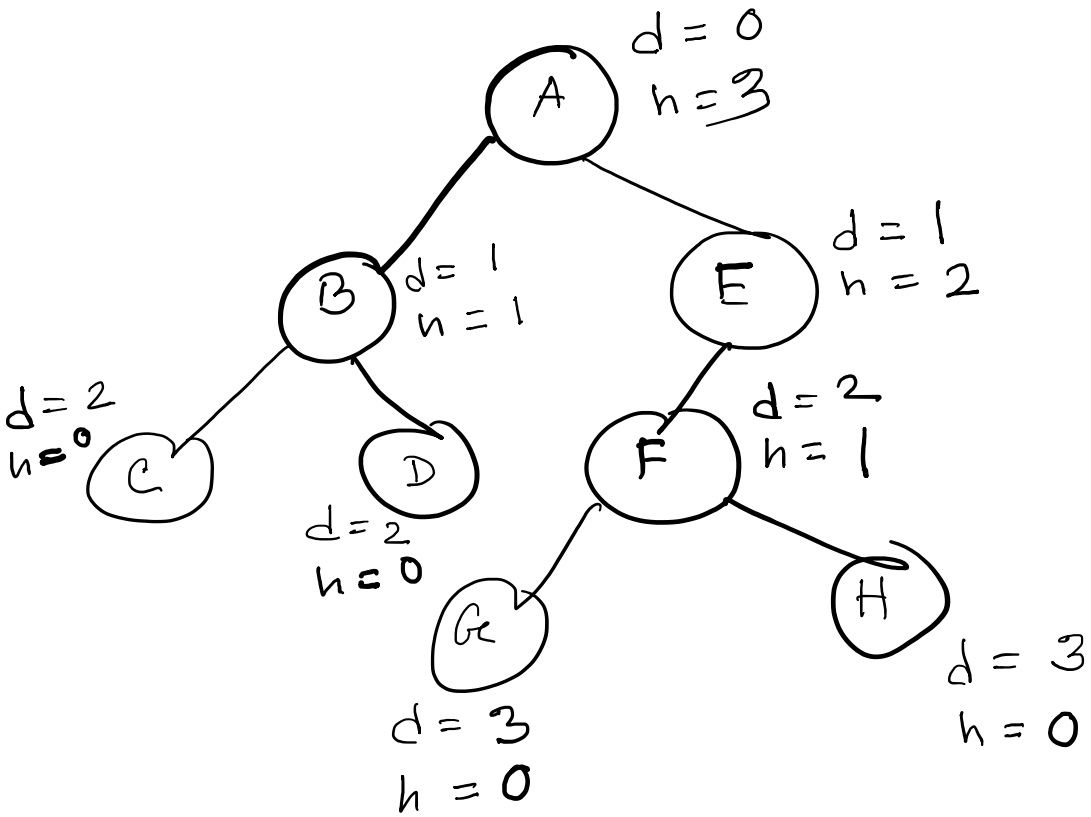
→ path$(s, d) = \{n_1, n_2, n_3 \ldots n_k : n_1 = s,$ $n_k = d$, parent$(n_{i+1}) = n_i$ for $1 \leq i \leq k\}$

→ Length of a path: is the number of edges along the path

→ Level: all nodes at a given depth share a level.

→ **Depth:** distance of node n from root in terms of the number of edges

→ **Height:** maximum distance of node n from its farthest leaf.



→ **Width:** number of nodes on the level containing the most nodes.

Binary Tree

min height = $\lfloor \log_2 n \rfloor$

max height = $n - 1$

n = total number of nodes

Array representation

parent (i) = $(i-1)/2$

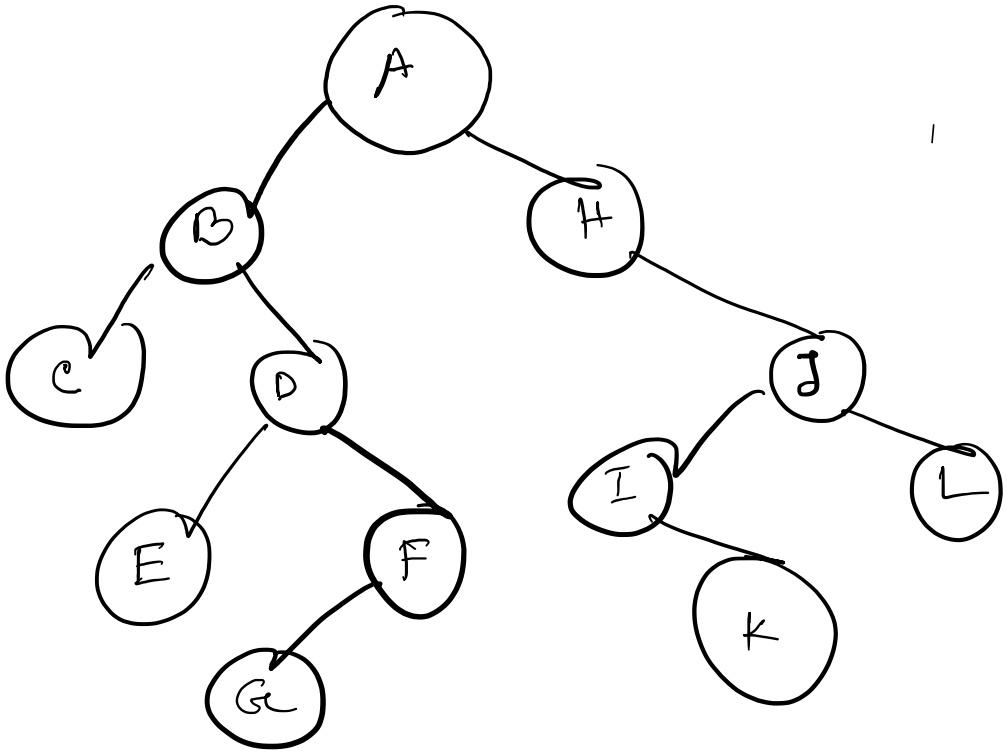
Left (i) = $2i + 1$

righ (i) = $2i + 2$

Tree traversal:

→ Breadth-first: visit nodes level by level, from left to right

→ Depth-first:
- → pre-order: root, left, right
- → in-order: Left, root, right
- → post-order: left, right, root



Breadth-first: A B H C D J E F I L G K

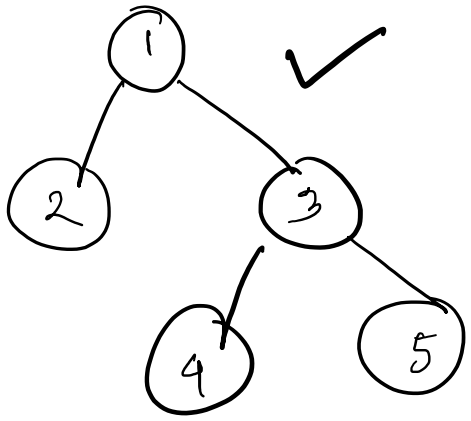
Pre-order: A B C D E F G H J I K L

In-order: C B E D G F A H I K J L

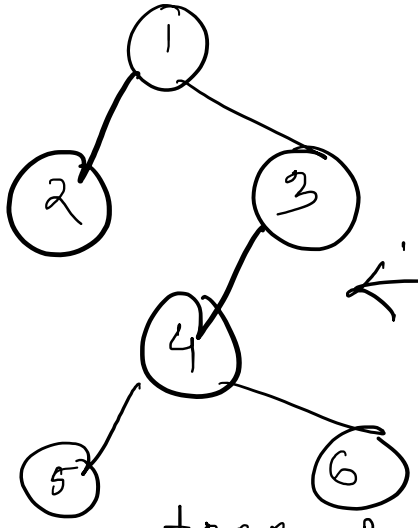
Post-order: C E G F D B K I L J H A

Full Binary Tree:

Each interior node (non-leaf) node has two children



tree A

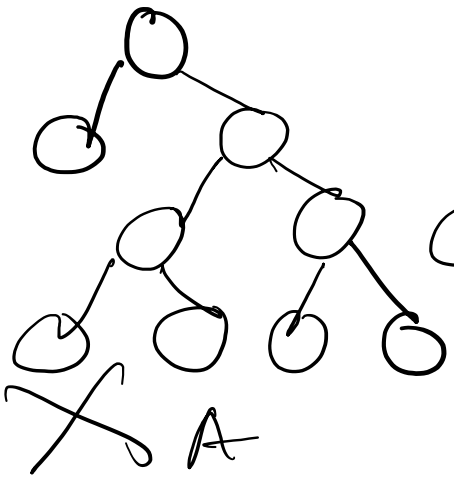


← not a Full B. Tree

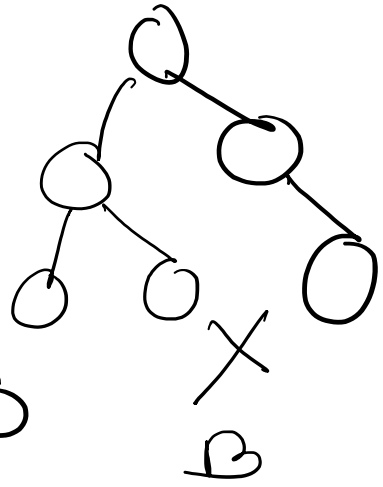
tree B

Perfect Binary Tree

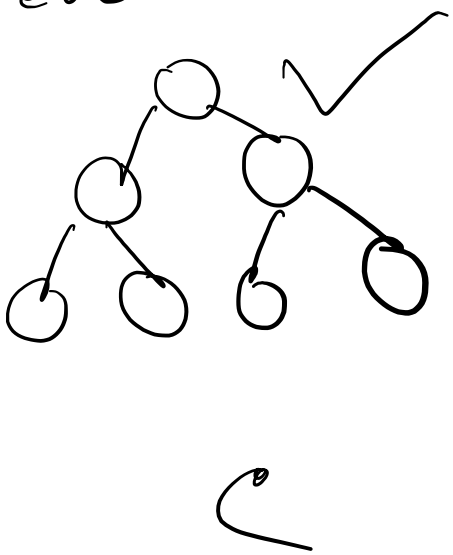
A full binary tree with all leaves at the same level.



A



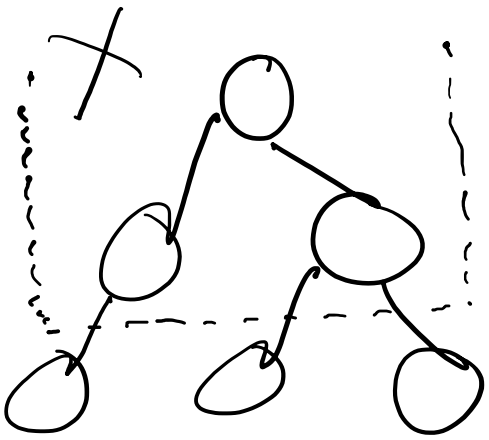
B



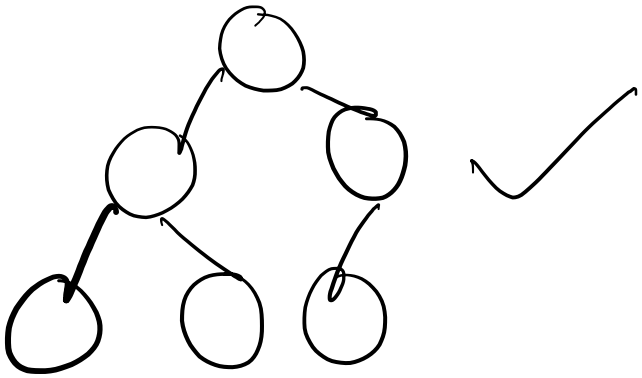
C

Complete Binary Tree

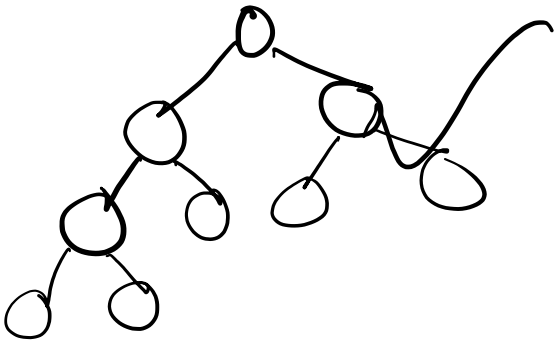
A binary tree of height h, where the tree is a perfect binary tree down to height h-1 and the nodes at the lowest level are filled from left to right



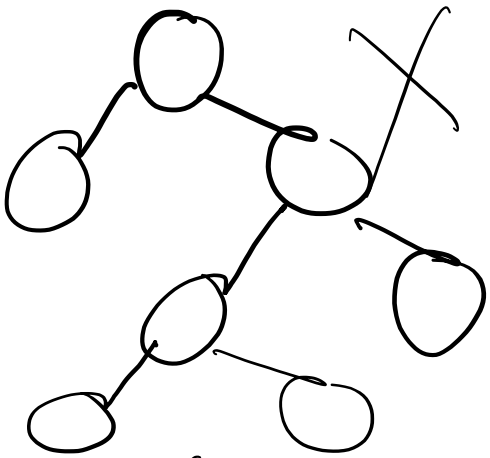
(A)



(B)



(C)



(D)

Binary Search Tree

Operations

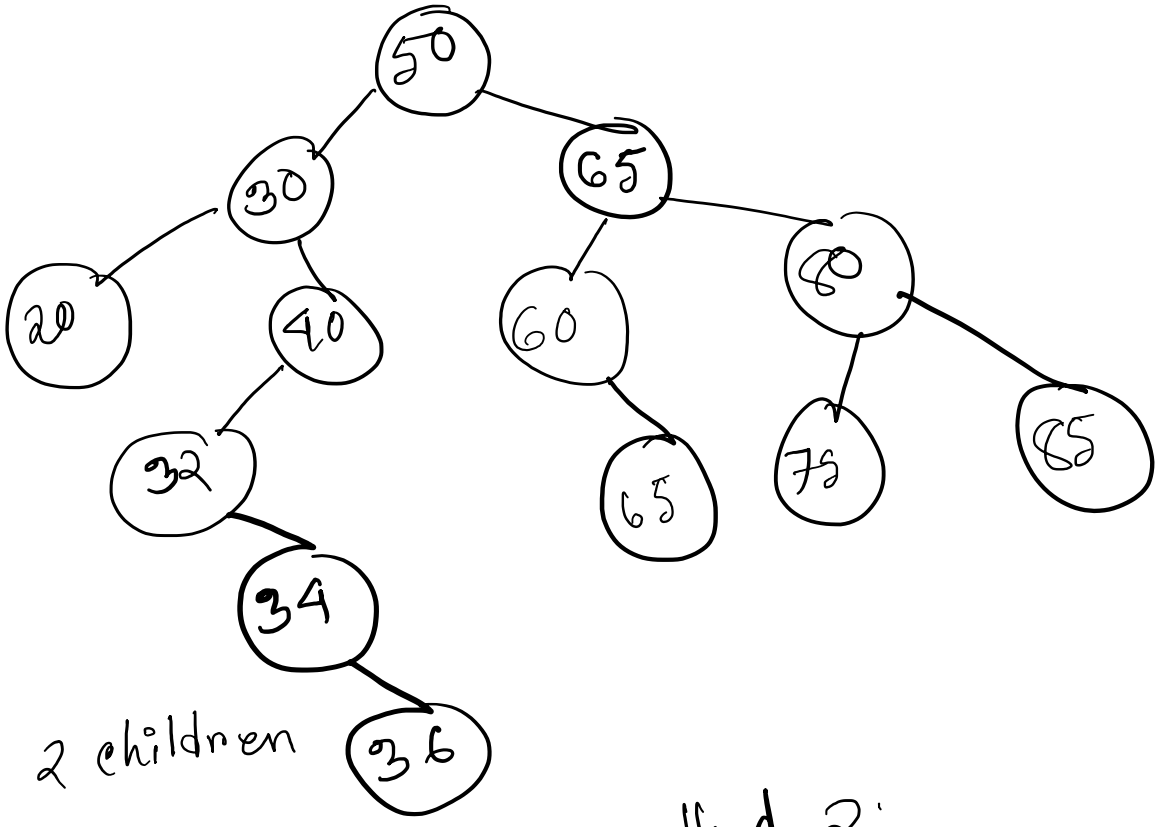
1. contains
   - ↳ boolean
2. findMin / findMax
   - ↳ for min keep going to the left
   - ↳ for max keep going to the right
3. insert
   - ↳ if x is not found in the tree (contain returns false), insert in appropriate path
4. delete

   case 1: is leaf node

   → delete the node

case 2: 1 child

↳ delete the node and connect with the parent of the deleted node.



Case 3: 2 children

method 1:

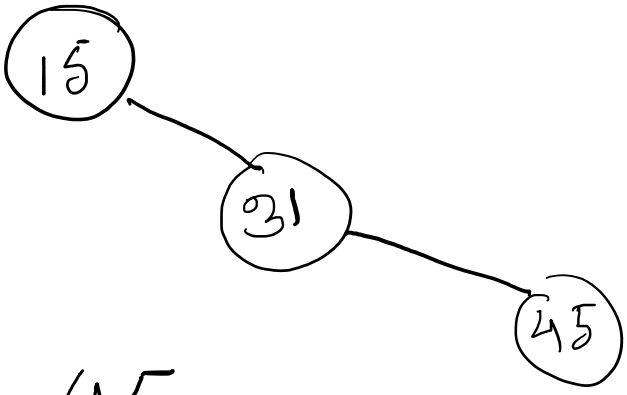
a) Find min of the right subtree
b) copy the value in targetted node
c) remove duplicate

method 2:

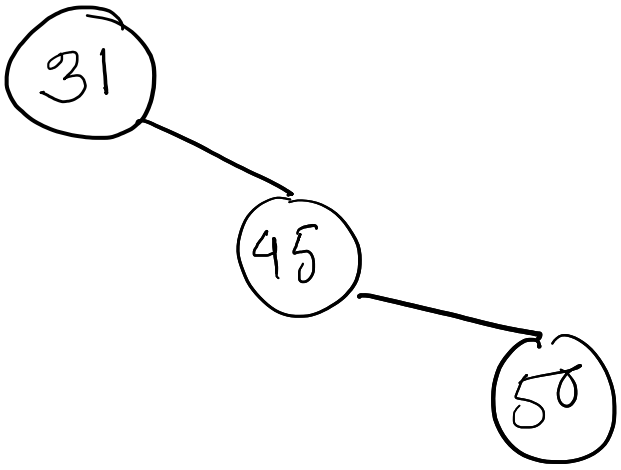
a) Find max of the left subtree
b) and c) same as method 1.

Balancing

15, 31, 45



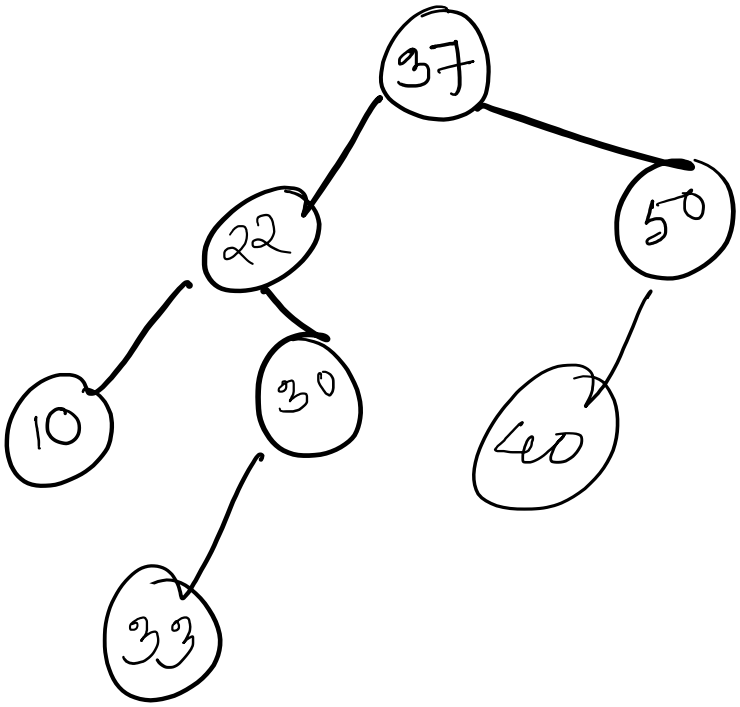
31, 15, 45



AVL Trees

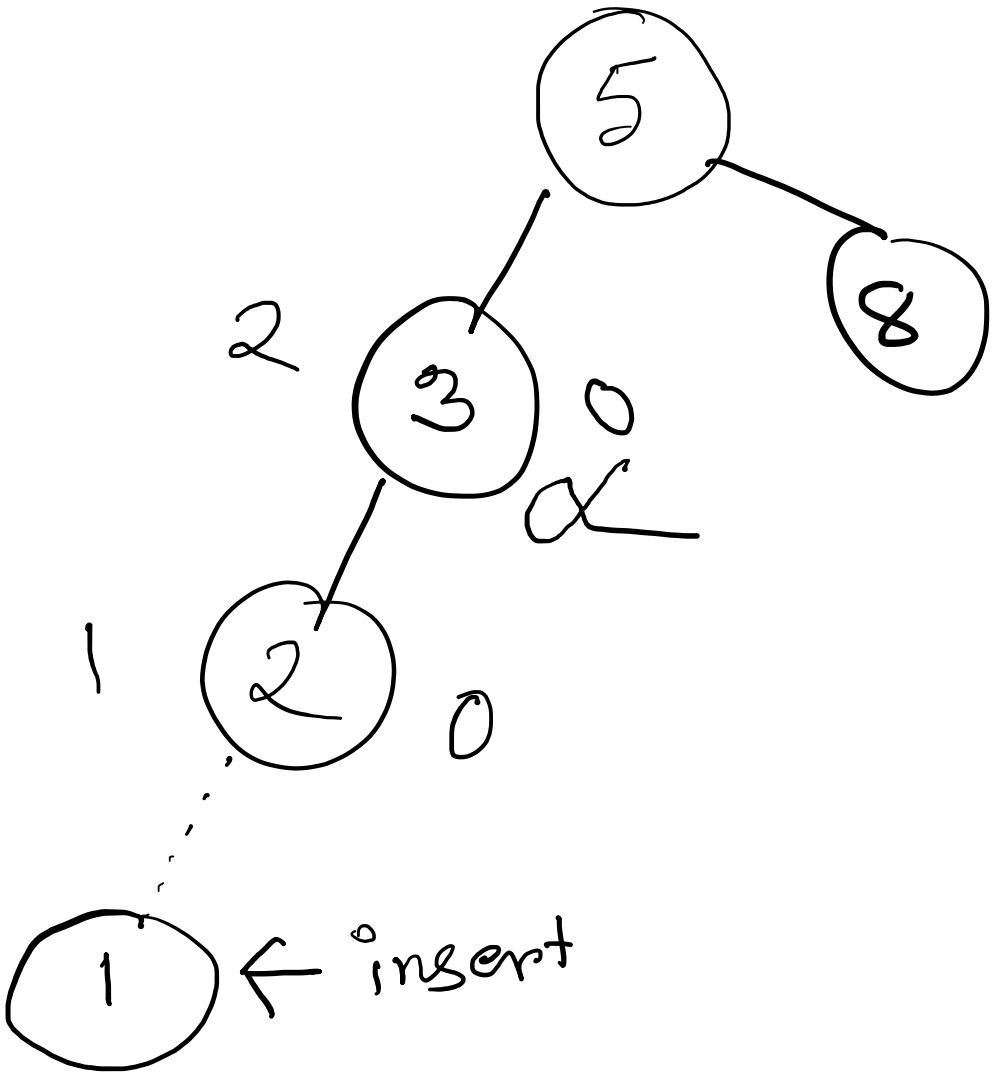
A binary search tree where for every node the height of the left and right subtrees can differ by at most 1.

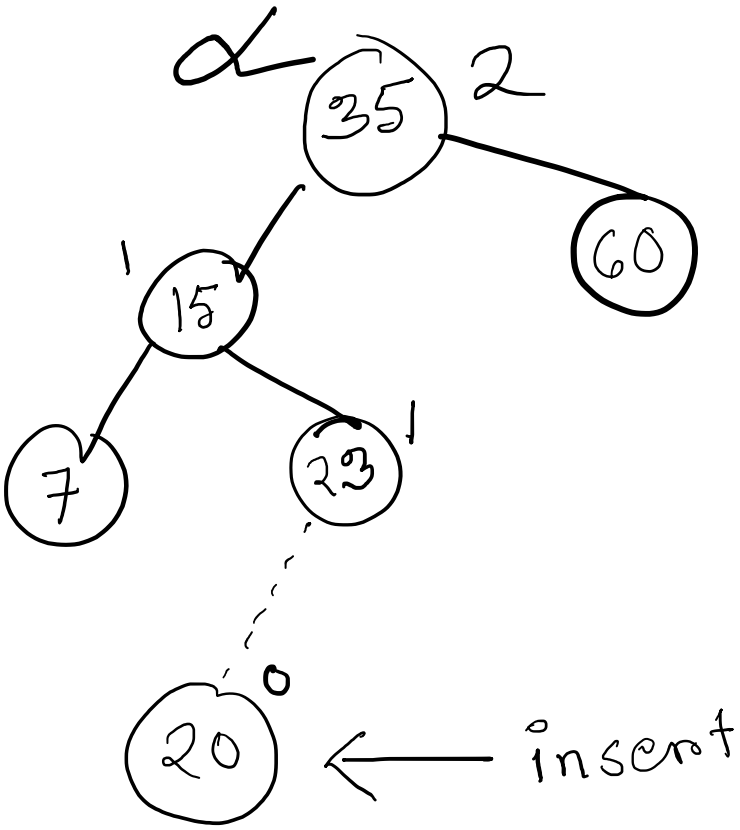
AVL = Adelson-Velskii and Landis



Four cases of unbalanced insertions

Let α be the node that become unblanced




α
35
2
60
1
15
7
23
1
0
20
insert