Lecture 7 : Splay Trees

↓

Most widely used

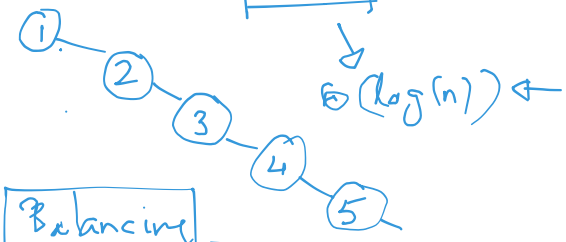
* Structurally identical to a BST
* Balanced data structure of choice in real world operations

All operations take $\Theta(\lg n)$ time on average.

Some operations can be very slow. ($\Theta(n)$)

BST → Worst case

Insert, delete, search ~ $\Theta(n)$



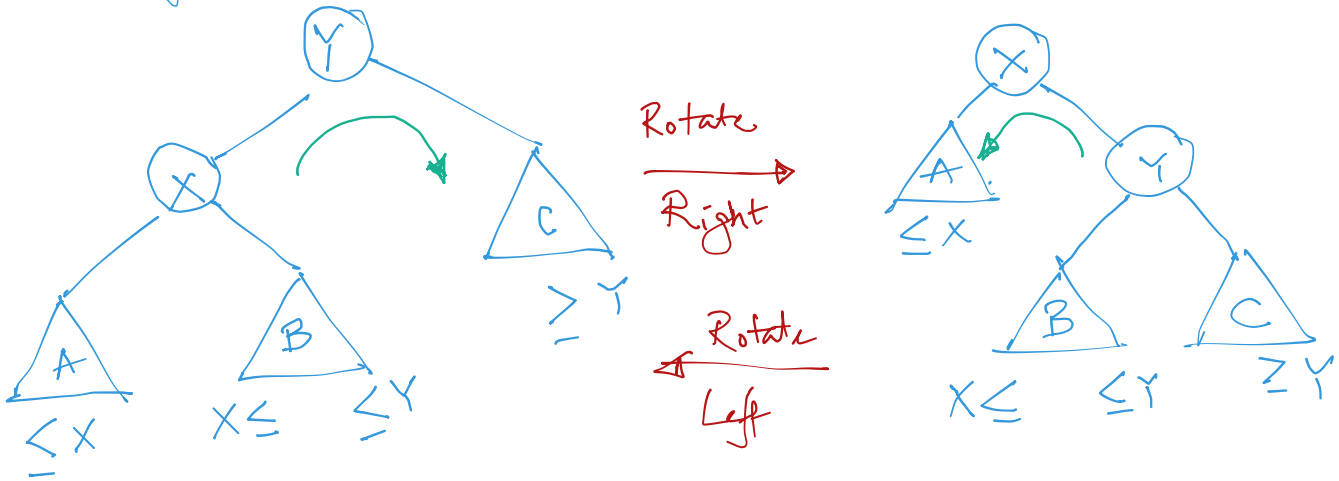
$\Theta(\log(n))$ ←

Balancing

Eg. AVL, Red Black, Splay trees

Any sequence of k splay tree operations starting from an empty tree, never exceeding n items in the tree, takes $\Theta(k \lg n)$ in the worst case.

Balancing of a tree → Tree Rotations



Splay trees : are not kept **perfectly balanced**.

① Find (key k)

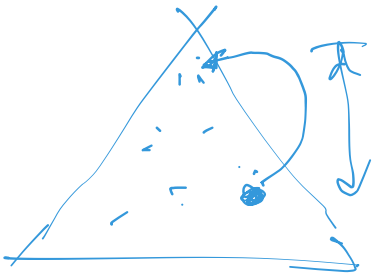
Begin by finding the key k [Like ordinary BST]

* Walk down in the tree to the node with key k, OR reach a nullptr.

* X → Node where the search ended, whether succeeded or not.

* Splay X up the tree, through a sequence

of rotations, so that X becomes the root.



Semi-balance the tree

~~in the process~~

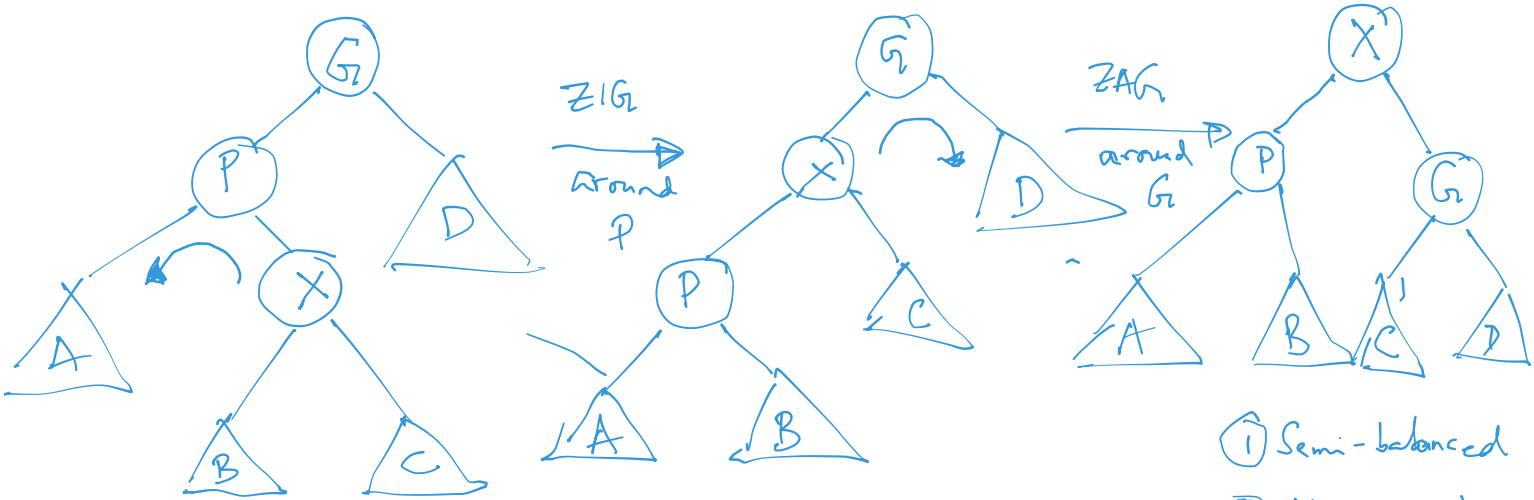
Opportunity to rebalance the tree.

* Same item is looked up again and again
→ Faster

* Do the right rotation at the right time.

3 cases

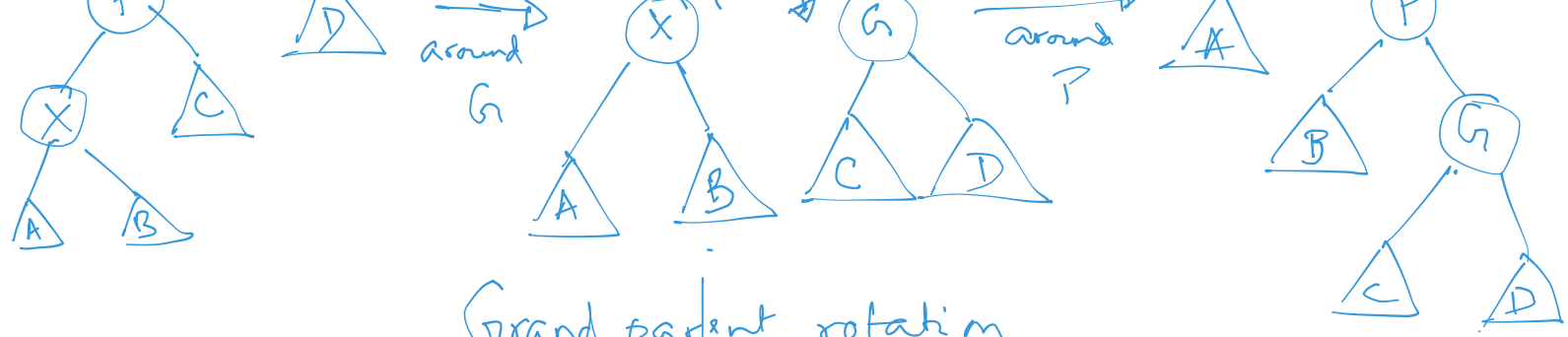
1. X is the left child of a right child
OR X is the right child of a left child.



ZIG-ZAG rotation

2. X is a left child of a left child, OR
X is a right child of a right child.

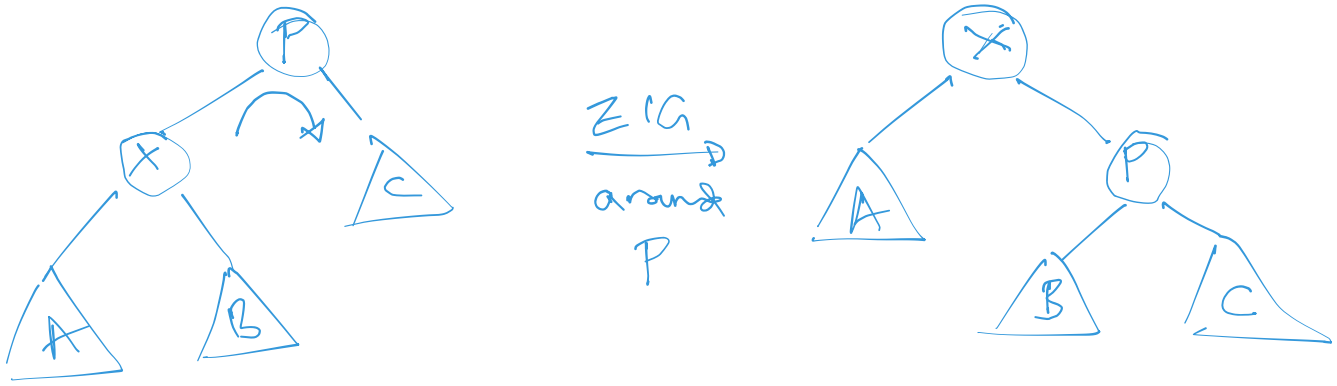
ZIG-ZIG rotation



Grand parent rotation
goes before the parent rotation

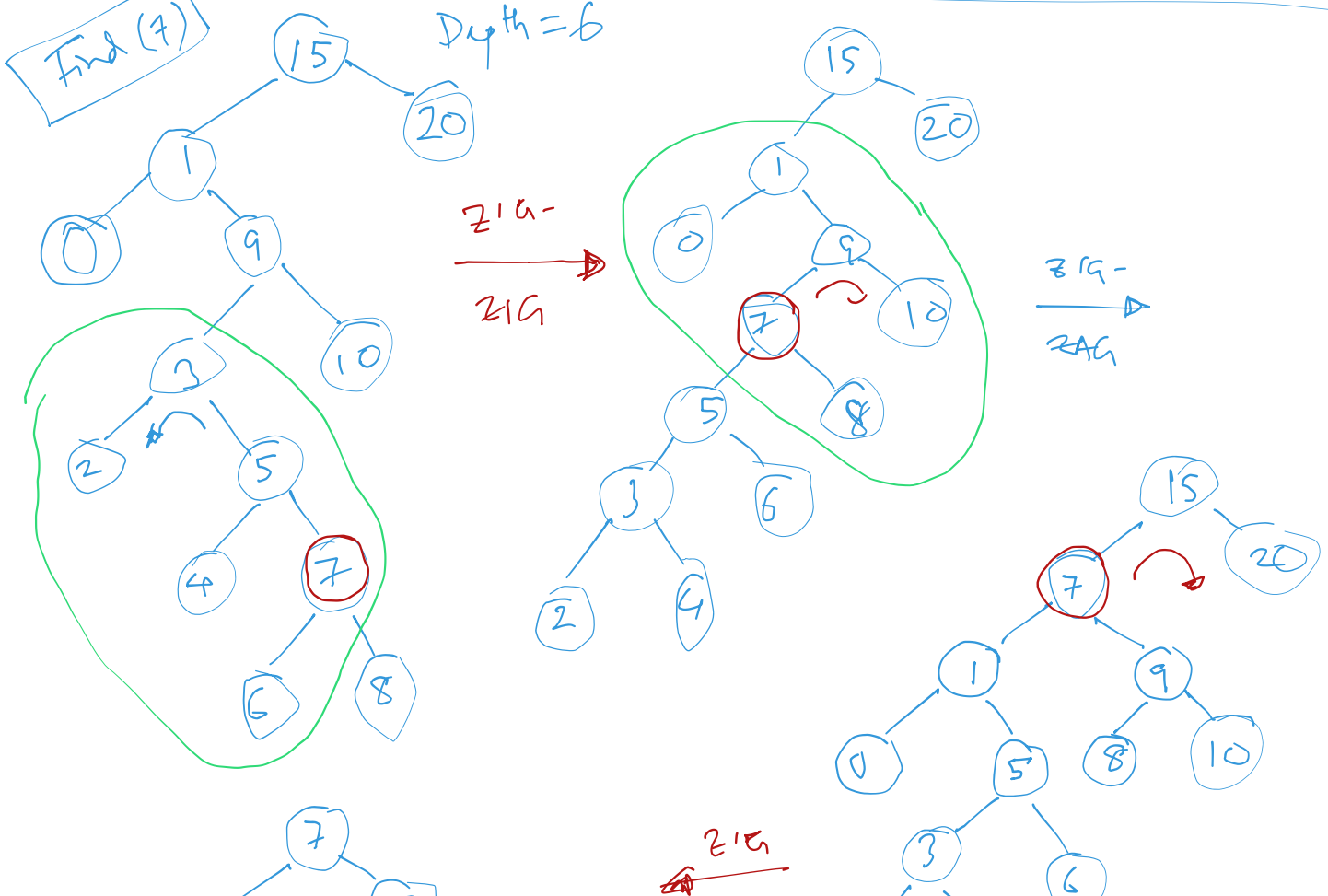
Repeat ① and ②

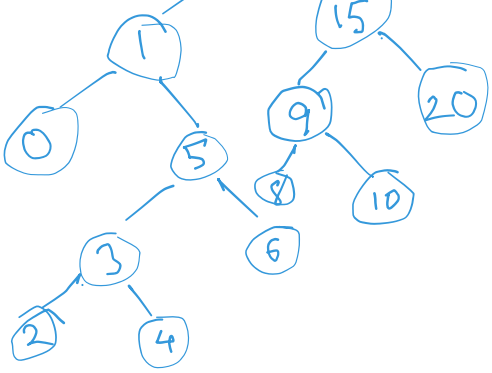
③ X is a child of the root.



Find (7)

Depth = 6





Depth = 4

A node initially at depth d from the access path from the root to X moves X to the final depth $\leq \left(3 + \frac{d}{2}\right)$

$\Theta(\lg n)$

(2) Insert (Key k, Value v)

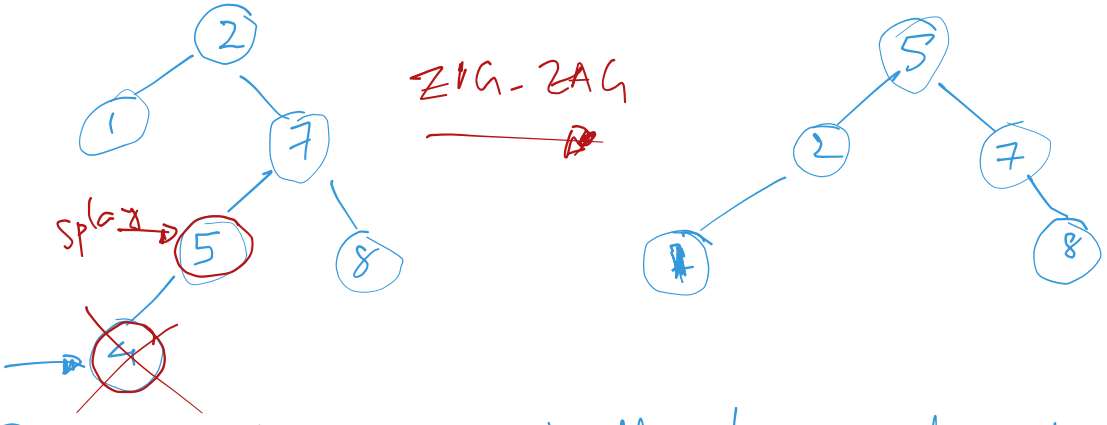
* Insert new entry (k, v) → BST

* Splay the new node to the root.

(3) Remove (key k)

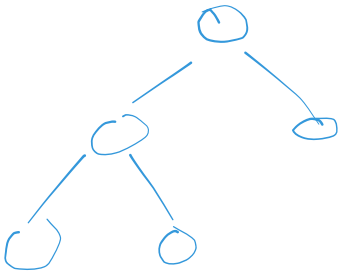
A node having key k is removed from the tree, as with ordinary BST.

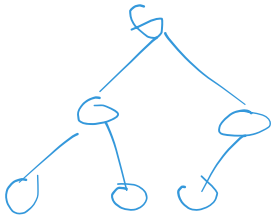
Let X be the node removed from the tree. Splay X's parent to the root.



If key k is not in the tree, splay the last node visited where the search ended (to the root)

Use Operations (find / remove / insert) as a pretest for performing Rotations (splay)



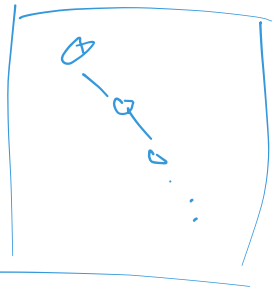


n

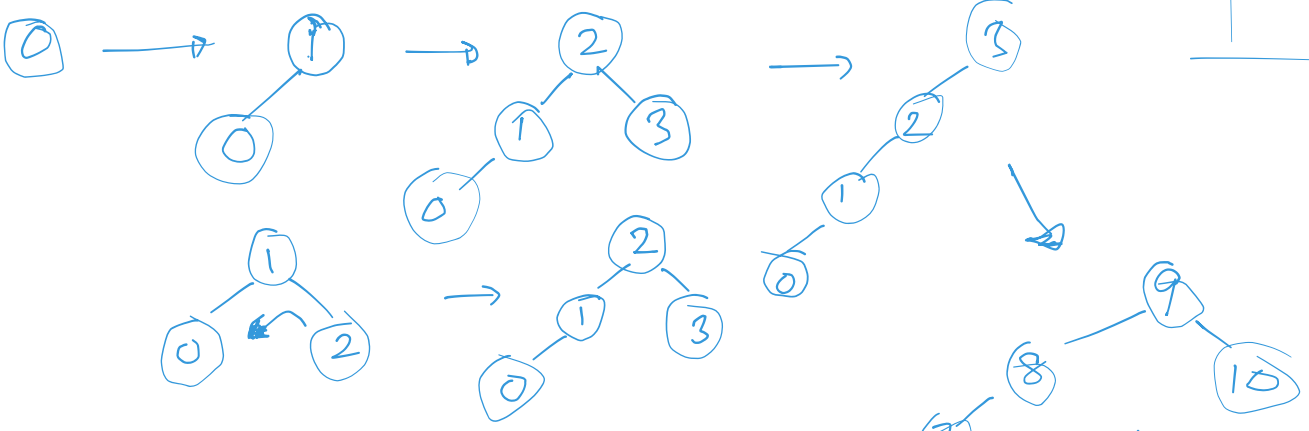
$(\log_2 n)$

$\Theta(\lg n)$

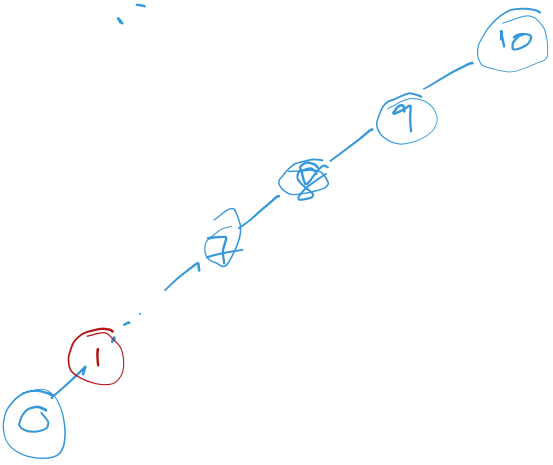
Why rotate around the grandparent first for ZIG-ZIG operation?

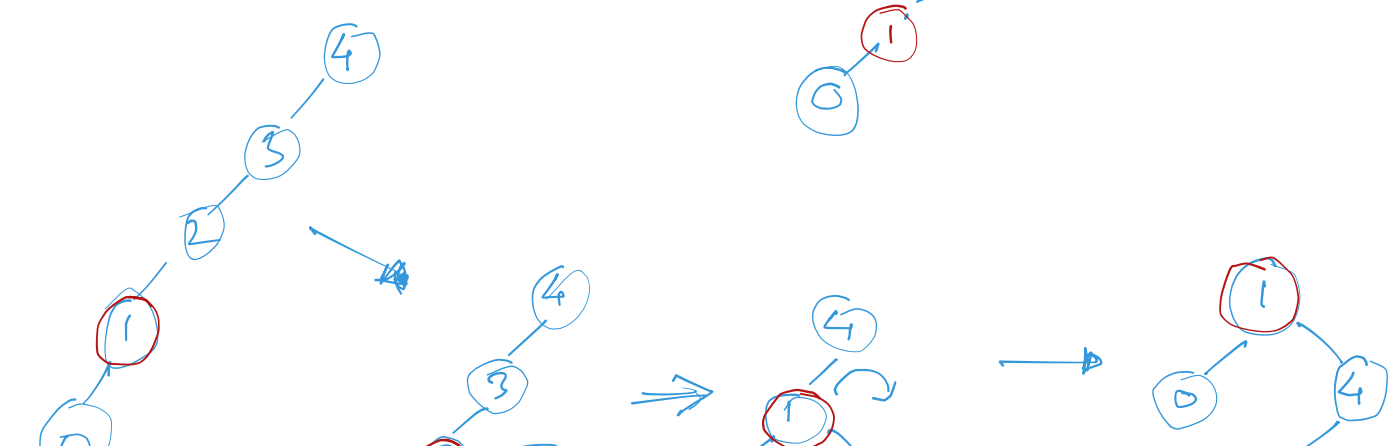


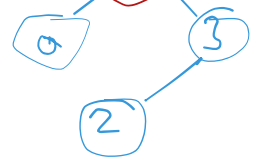
0, 1, 2, 3, 4, ..., 9, 10

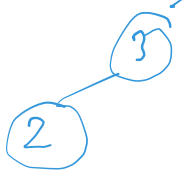


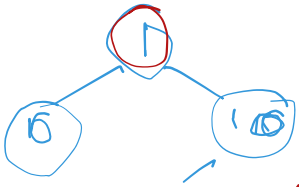
Find (1)

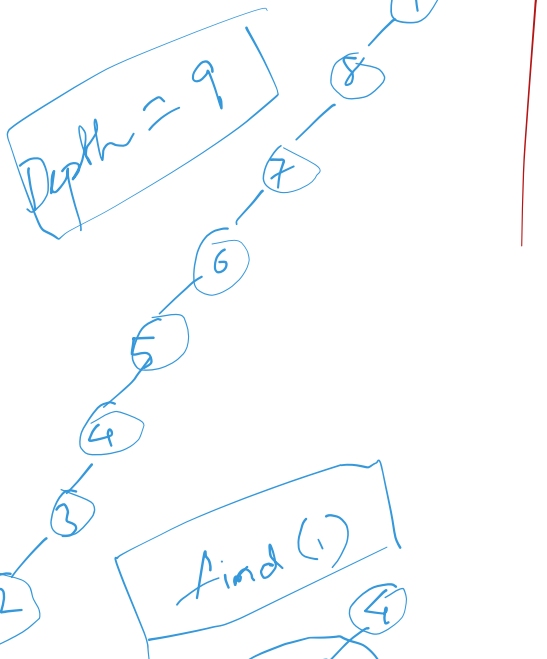








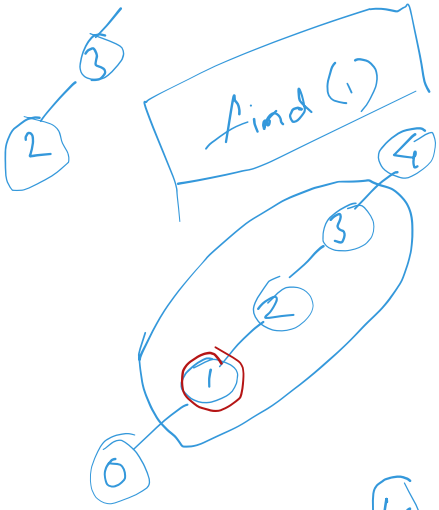




Depth = 9

Rotated parent first and then the grandparent.

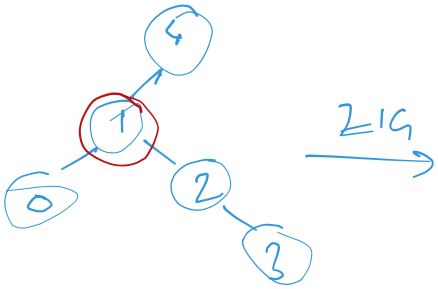
Grandparent first



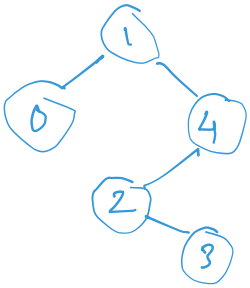
Find (1)

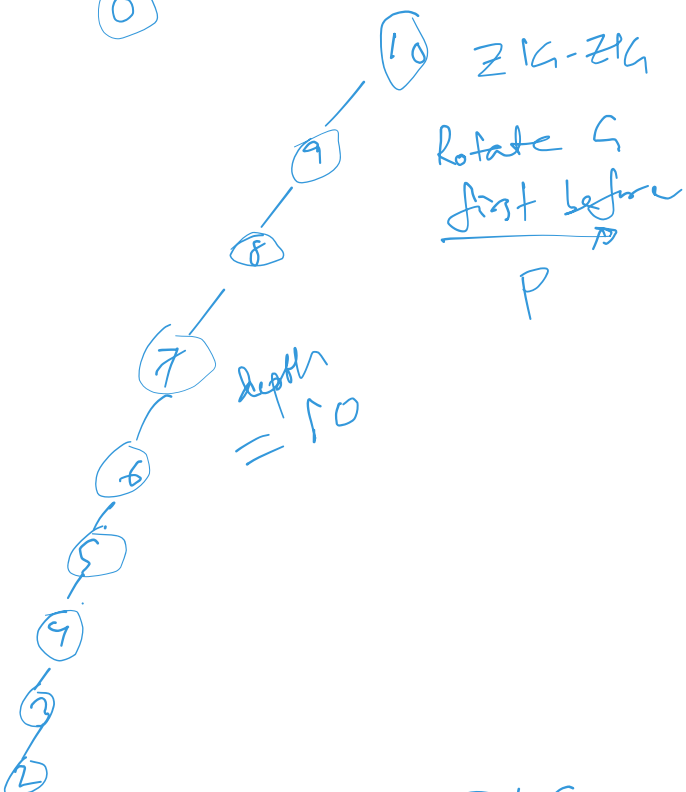
ZIG

ZIG



ZIG

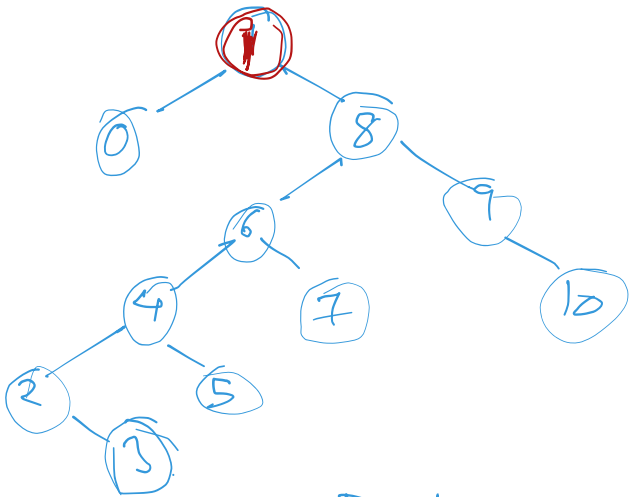




ZIG-ZIG

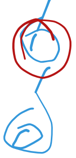
Rotate G first before P

depth = 10



Depth = 5

Rotate around.



Zig - Zig → Grandparent first.