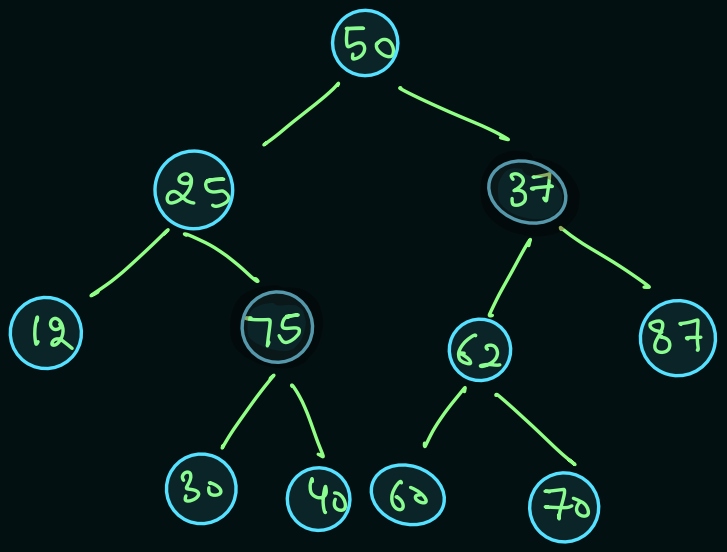
Recover BST :-



Fix faulty node in BST.

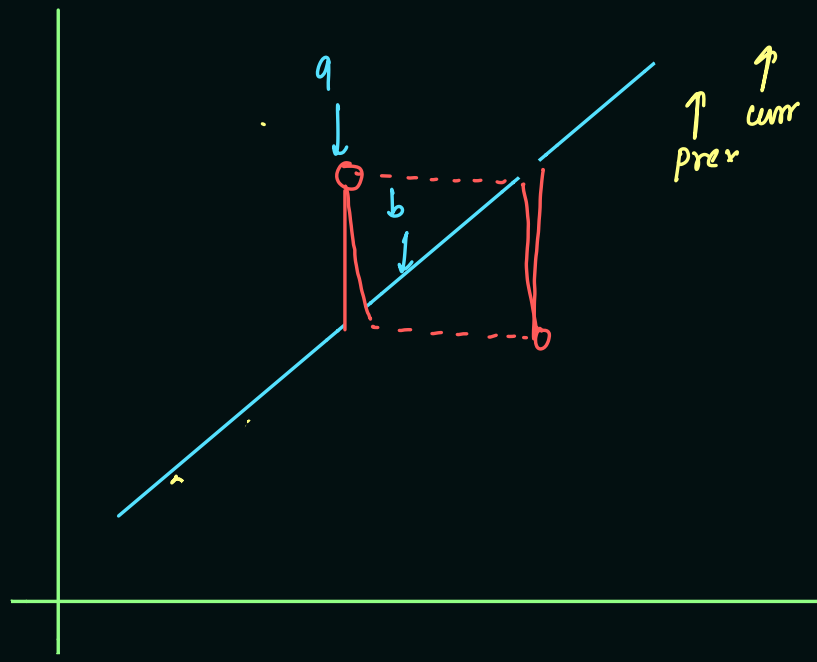
Initially tree was BST but one pair was somehow swapped. Now we have to fix this tree.

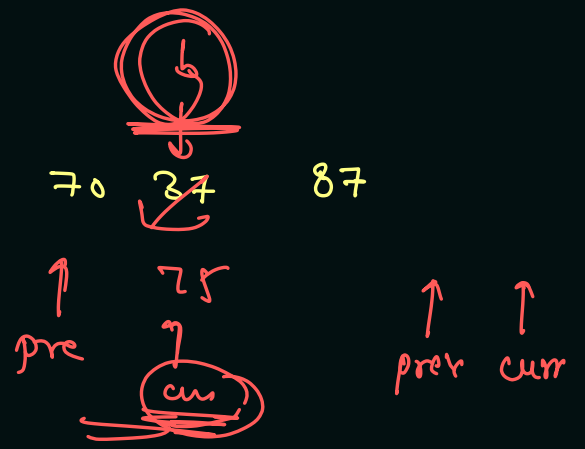
→ If Inorder is not sorted then BST is faulty.

Inorder of BST → 12 25 30 ~~75~~ 40 50 60 62 70 ~~37~~ 87

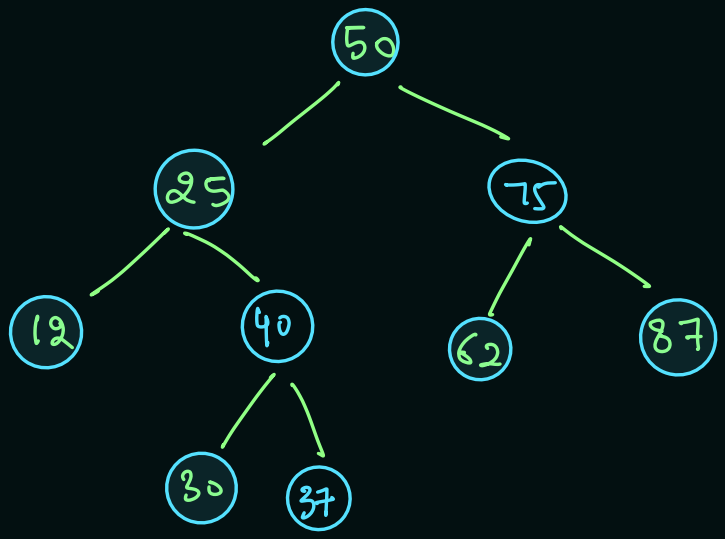
a → 75

b → 37



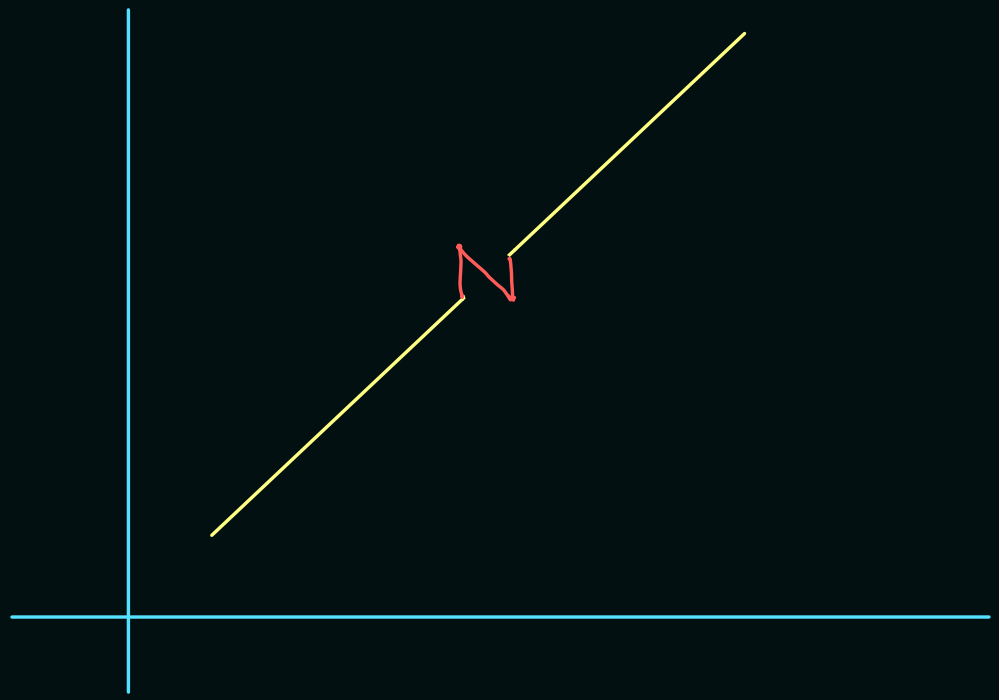


Inorder of BST

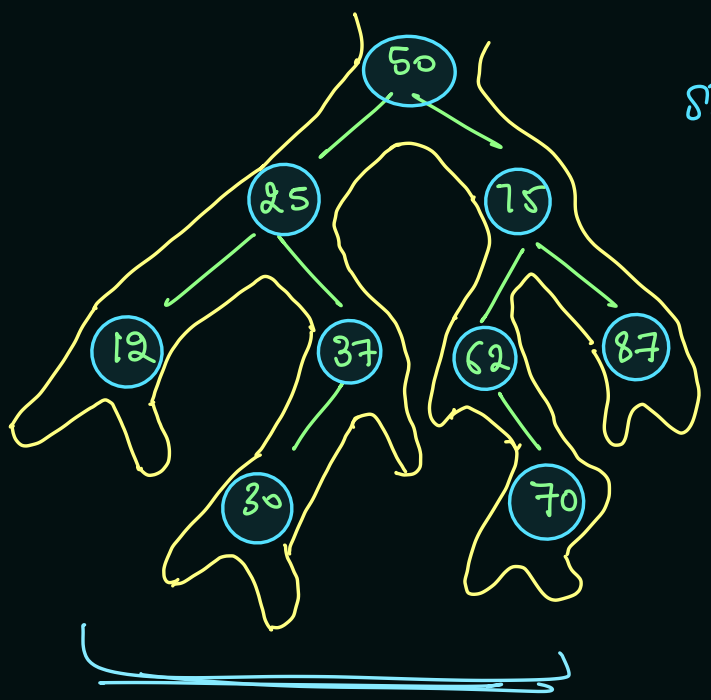


Inorder → 12 25 30 40 37 50 62 75 87

prev cur



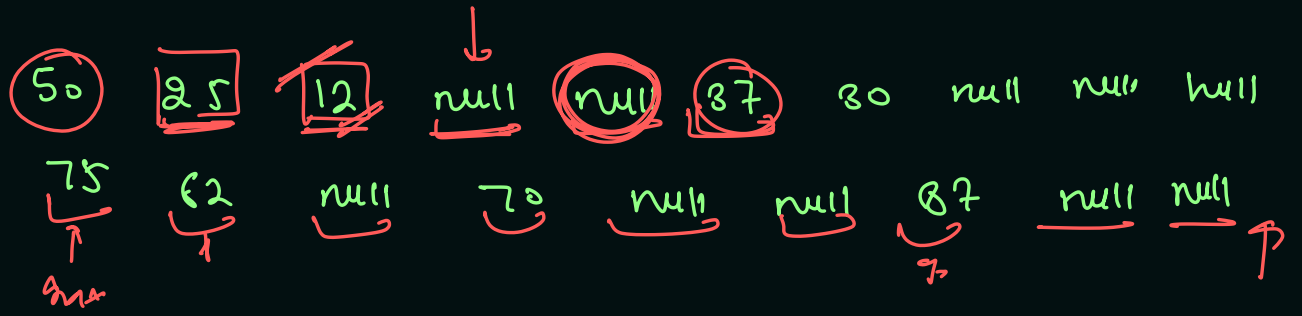
Serialize and deserialize

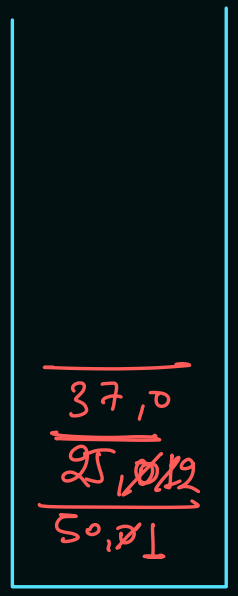


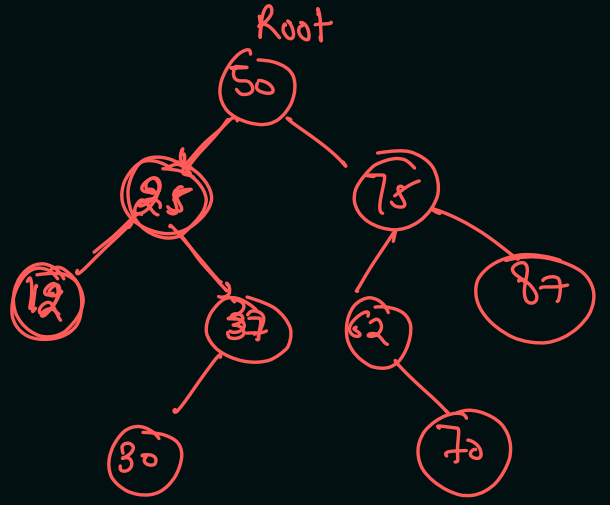
50 25 12 -1 -1 37 30 -1 -1 -1 75 62 -1
70 -1 -1 87 -1 -1

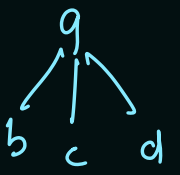
String → serialize :-

50 # 25 # 12 # null # null # 37 # 30 # null # null # null #
75 # 62 # null # 70 # null # null # 87 # null # null #

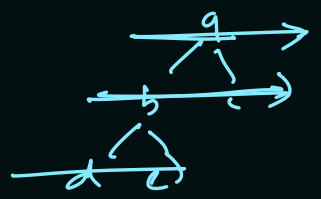




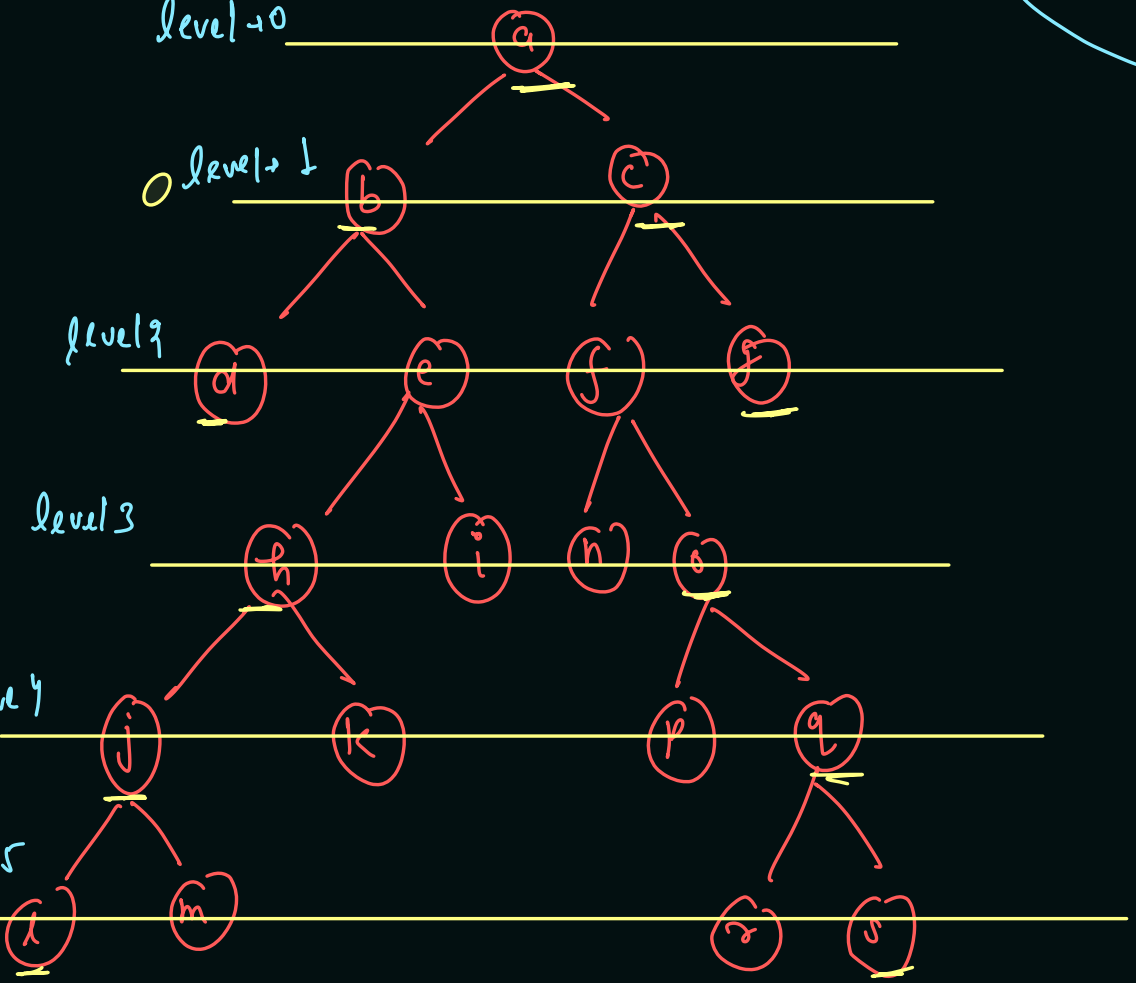




Left view and Right view :-



left view



a
b
d
h
j
l

left view

a
c
g
o
q
s

Right view

level order (1) →

level order (2) → [ ]

using null delimiter

using two queue

size based [while, while]

10 → a

11 → b c

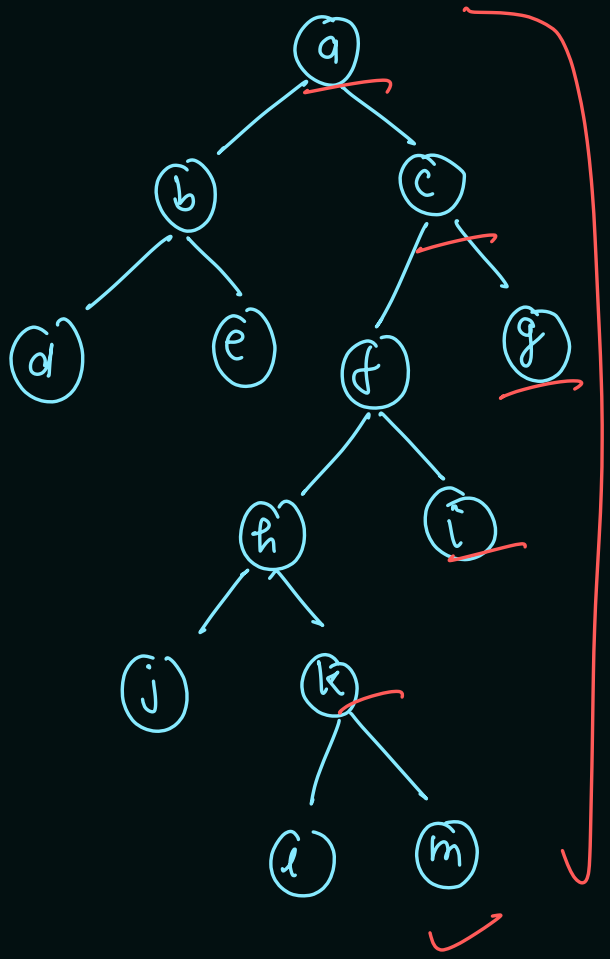
12 → d e f g

13 → h i n o ....

level order

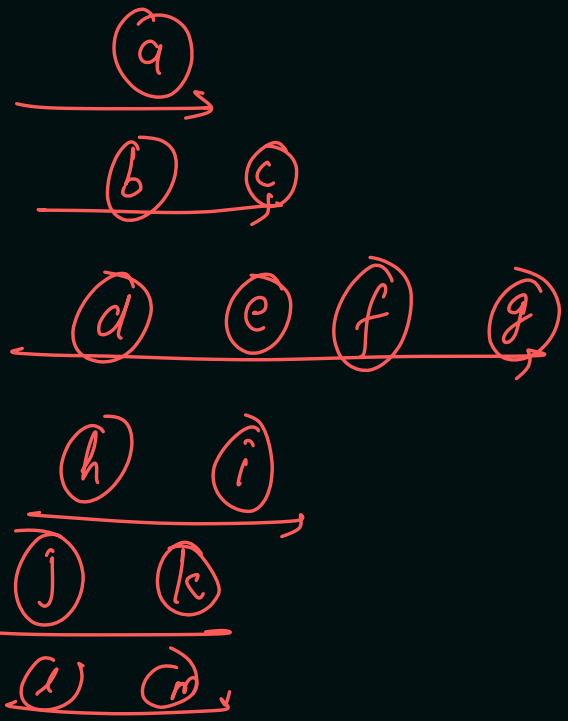
Left view → first node of level

Right view → last node of level



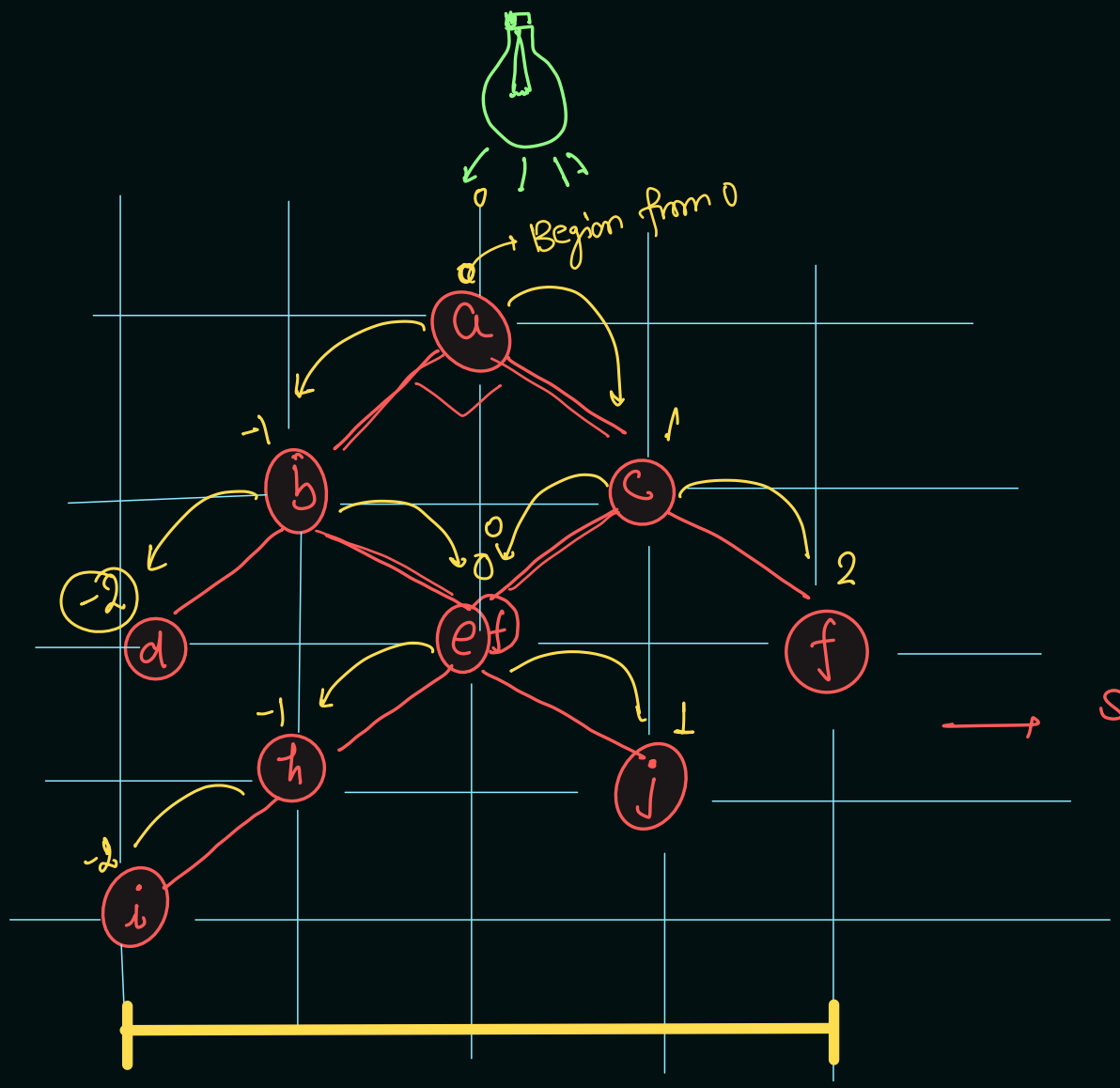
Right view

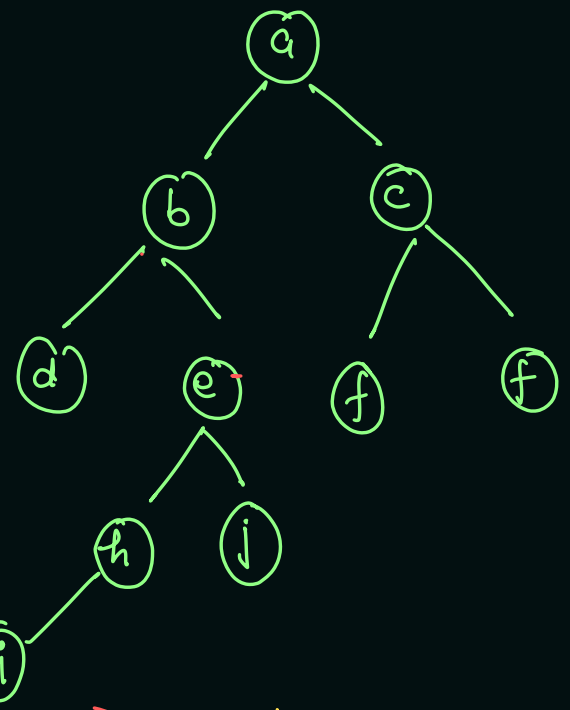
Size = ~~1~~ ~~2~~ 4 ~~2~~ ~~2~~ 2



a
c
g
i
k
n

Width of Shadow of Binary Tree:-





Shadow length = 4 on basis of End's.

right - left

left = -2

right = +2

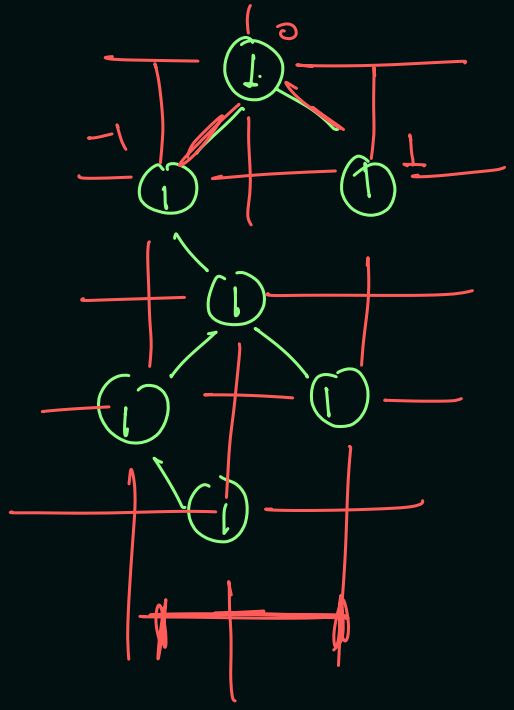
max left = |2|

max Right = 2

on the basis of node → right - left + 1

15

1
1
-1
1
1
-1
1
-1
-1
1
-1
-1
1
-1



on the basis of node

Result = Right - left

Vertical order → Top view, Bottom view

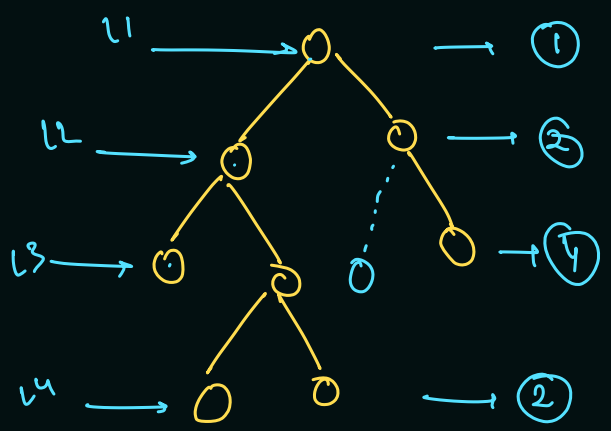
Mom's traversal → Pre, Post, Inorder

Any Node to any Node → Max sum

BST Iterator

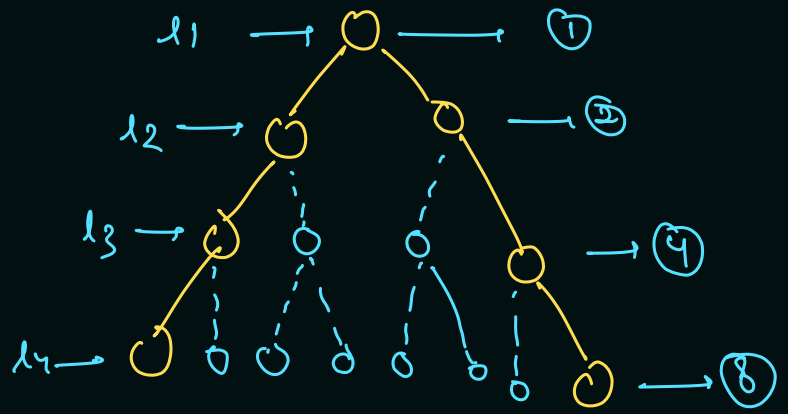
Max. width of Binary Tree :→

* On a particular level, leftmost node & Rightmost node should be not null
* space of node on a particular level is width of that level
* Max. width ??



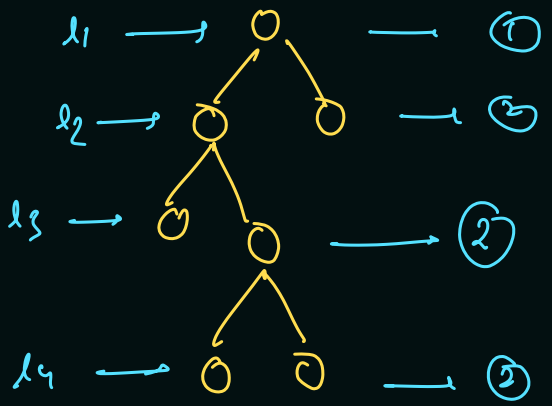
Max. width = 4

Shadow = 5



Max width = 8

Shadow = 7



Max = 2

Shadow = 4

Task :) Provide index to a complete Binary tree ??

rmi = right most index

lmi = left most index

Indexing –

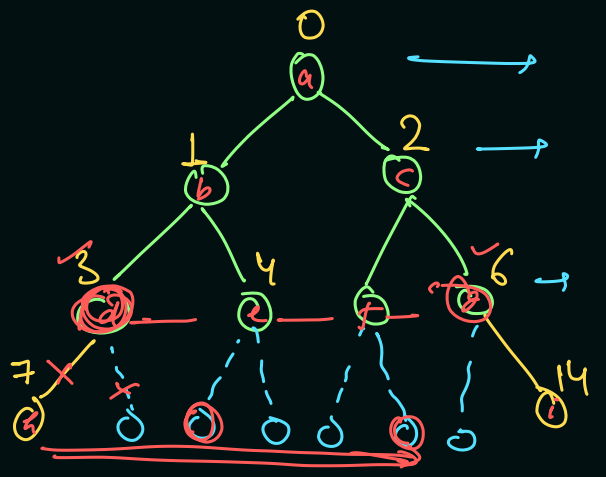
parent index = i

left child index = $2*i+1$

Right child index = $2*i+2$

width

$w = rmi - lmi + 1$



$0 - 0 + 1 = 1$

$2 - 1 + 1 = 2$

$6 - 3 + 1 = 4$

$14 - 7 + 1 = 8$

Task

1. level wise solve
2. parent index i

$lci = 2*i+1$

$rci = 2*i+2$

$min = lmi$

$max = rmi$

] during solution of a particular

3. width of particular level

$rmi - lmi + 1$

4. Maximise width.

~~(i,7)~~ ~~(i+1)~~

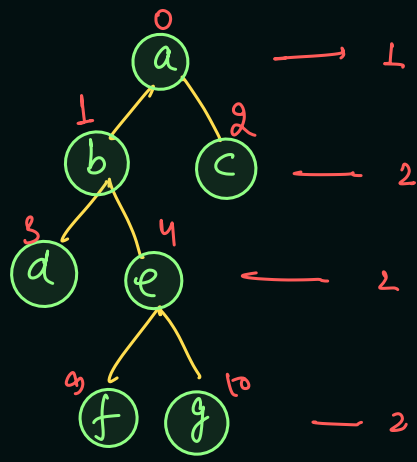
Size = ~~1~~ ~~2~~ ~~4~~ 2

lmi = min = ~~∞~~ ~~1~~ ~~3~~ 7

rmi = max = $-\infty$ ~~1~~ ~~3~~ ~~5~~ ~~6~~ ~~7~~ 14

⇒ 8

width = ~~1~~ ~~2~~ ~~4~~



Node
Index

Size = 1 2 2 2

Max width = 2

min index

lmi = 0 1 3 5

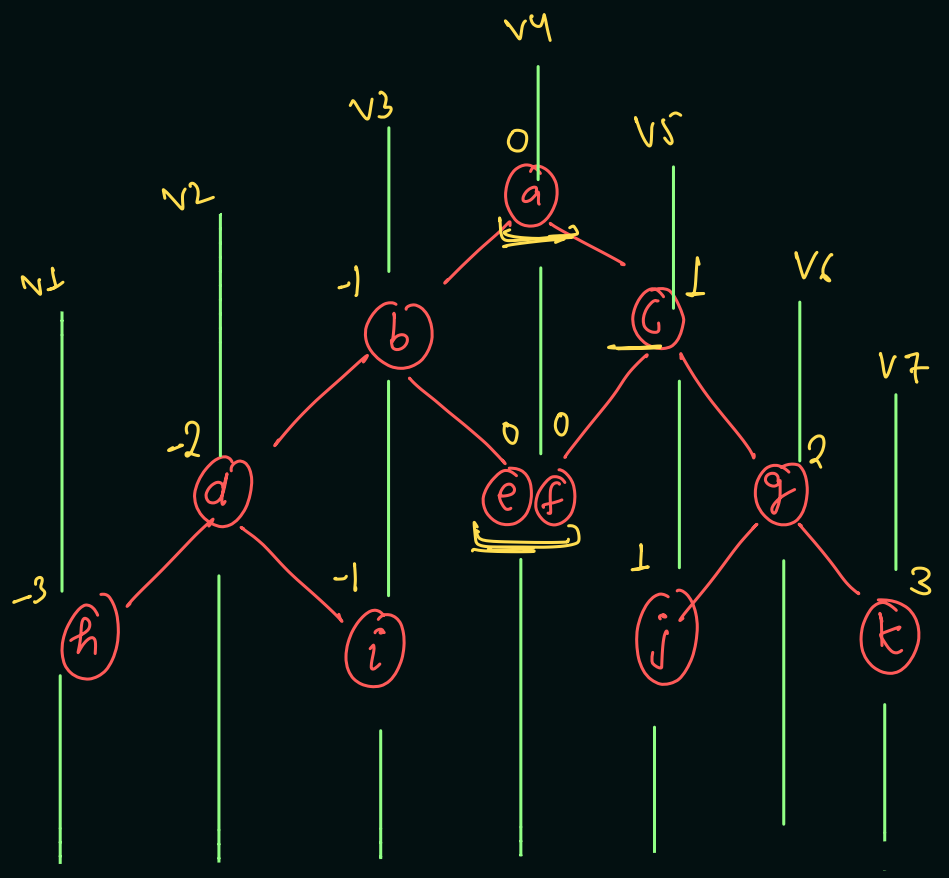
rmi = 0 1 2 3 4 5 6

max index

width = rmi - lmi + 1 = 1 2 2 2

Overall width = 0 1 2

vertical order traversal : →



V1 → -3 → h

V2 → -2 → d

V3 → -1 → b i

V4 → 0 → a, e, f

V5 → 1 → c j

V6 → 2 → g

V7 → 3 → k

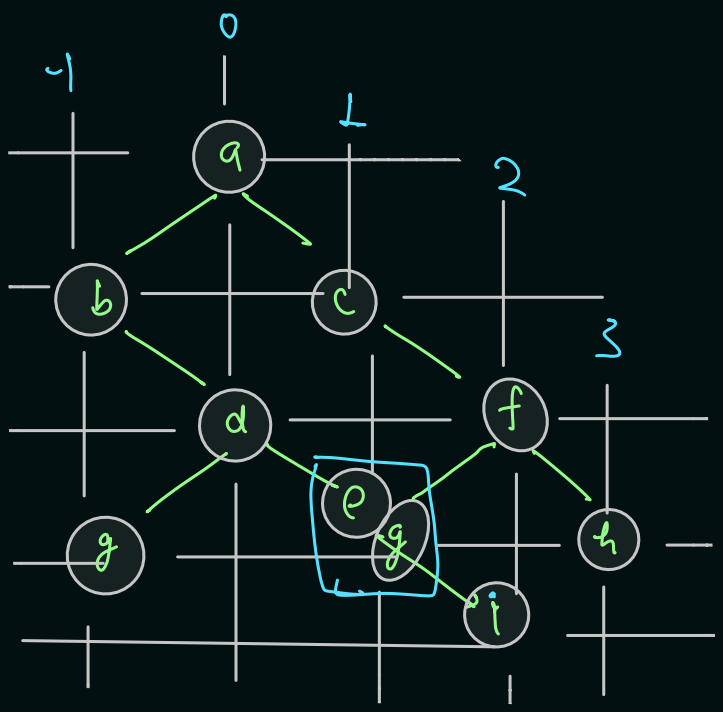
lh = -3

rh = 3

-3 to 3

HashMap < horizontal index, List<value> >

DFS:



Size = ~~1 2 2~~ 4 1

Hashmap:

-1 → b, g
0 → a, d
1 → c, e, g
2 → f, i
3 → h

vertical order (I)

lh = -1
rh = 3

-1 to 3 for every key. array ki + get

-1 → b, g
0 → a, d
1 → e, c, g
2 → i, f
3 → h

problem

DFS is not good we have to fill vertical order level wise.

$V_1 \rightarrow 10$

$V_2 \rightarrow 3$

$V_3 \rightarrow 7, (13), (15)$

$V_4 \rightarrow 2, 6, 8$

$V_5 \rightarrow 5, (1), (3)$

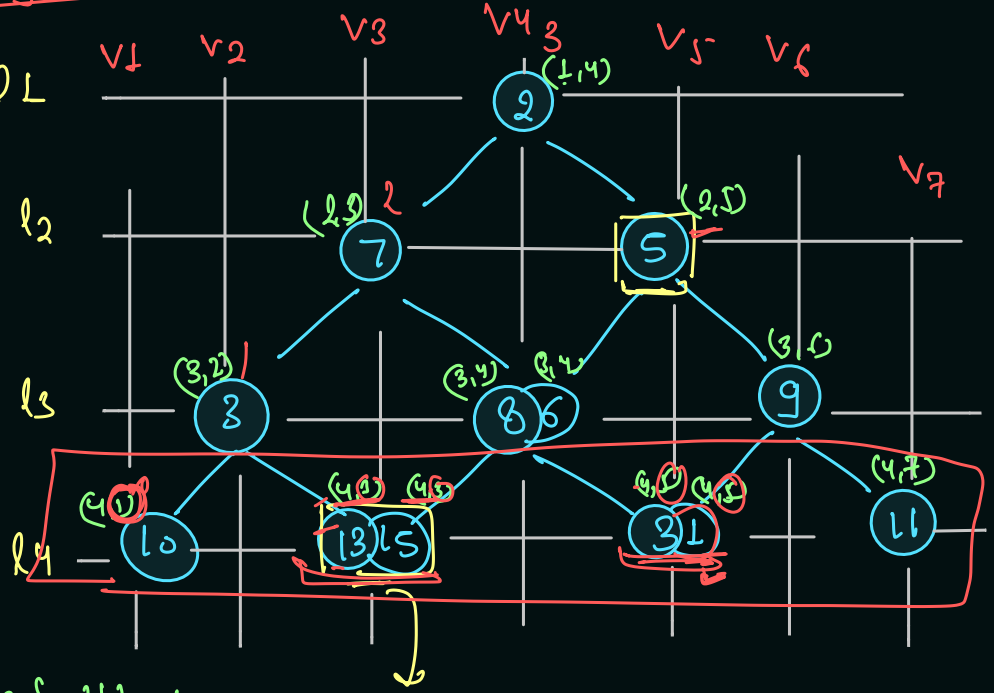
$V_6 \rightarrow 9$

$V_7 \rightarrow 11$

queue → priority Queue.

DL

RR = 3



By default → due to method of Iteration → level is prioritized.

on the basis of priority.

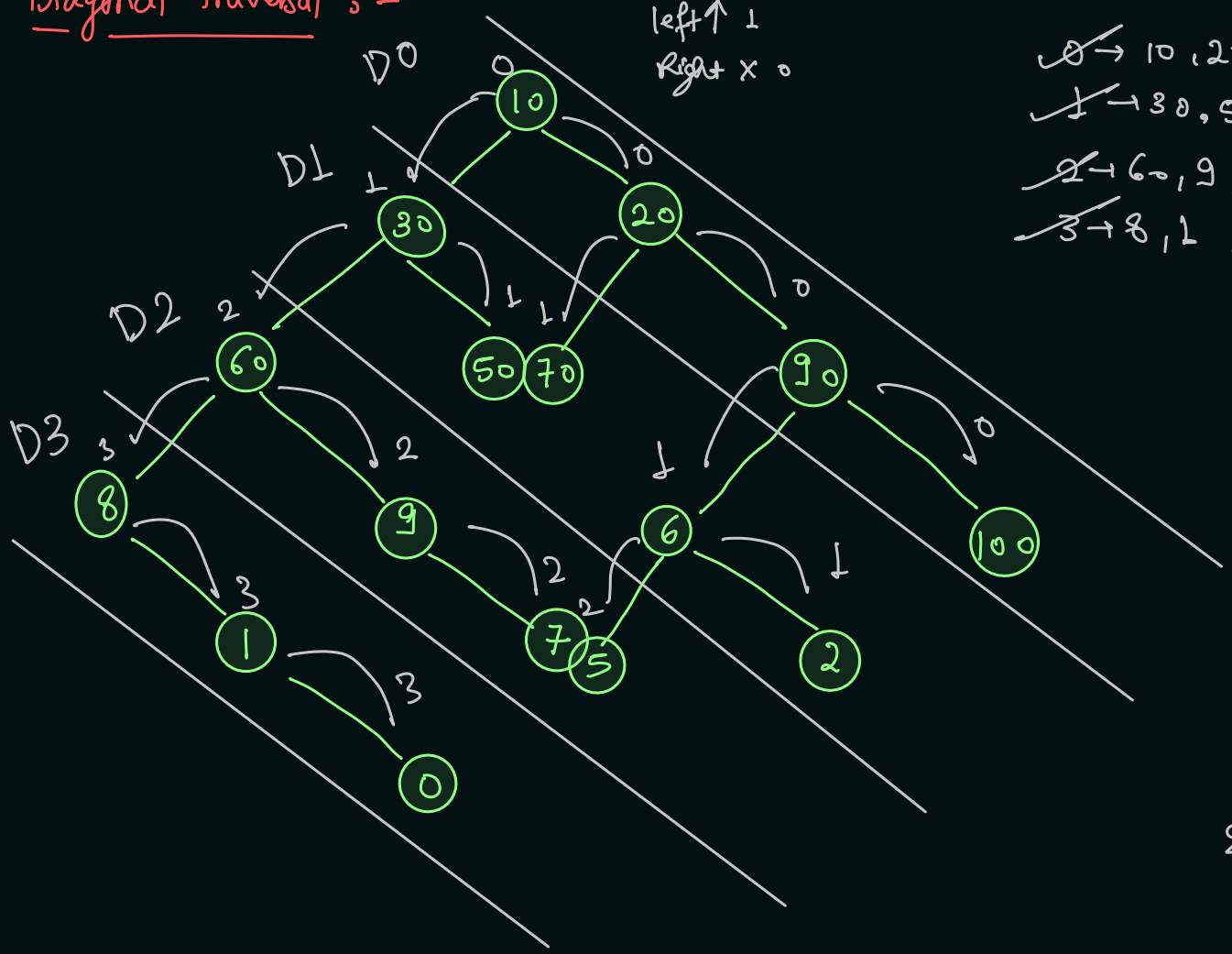
priority → Min.

For a particular level →

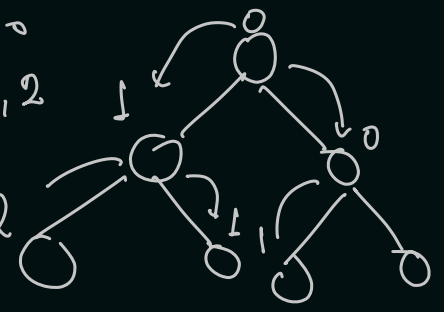
first priority → vertical index

If v.index is same then value is on priority.

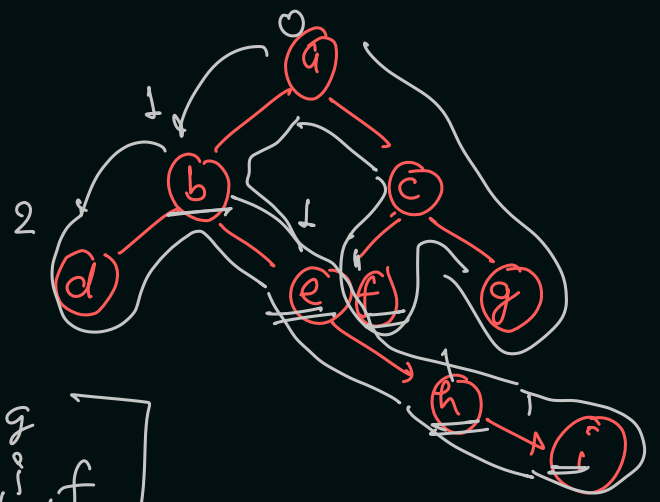
Diagonal traversal :-



0 → 10, 20, 90, 100
1 → 30, 50, 70, 6, 2
2 → 60, 9, 7, 5
3 → 8, 1, 0



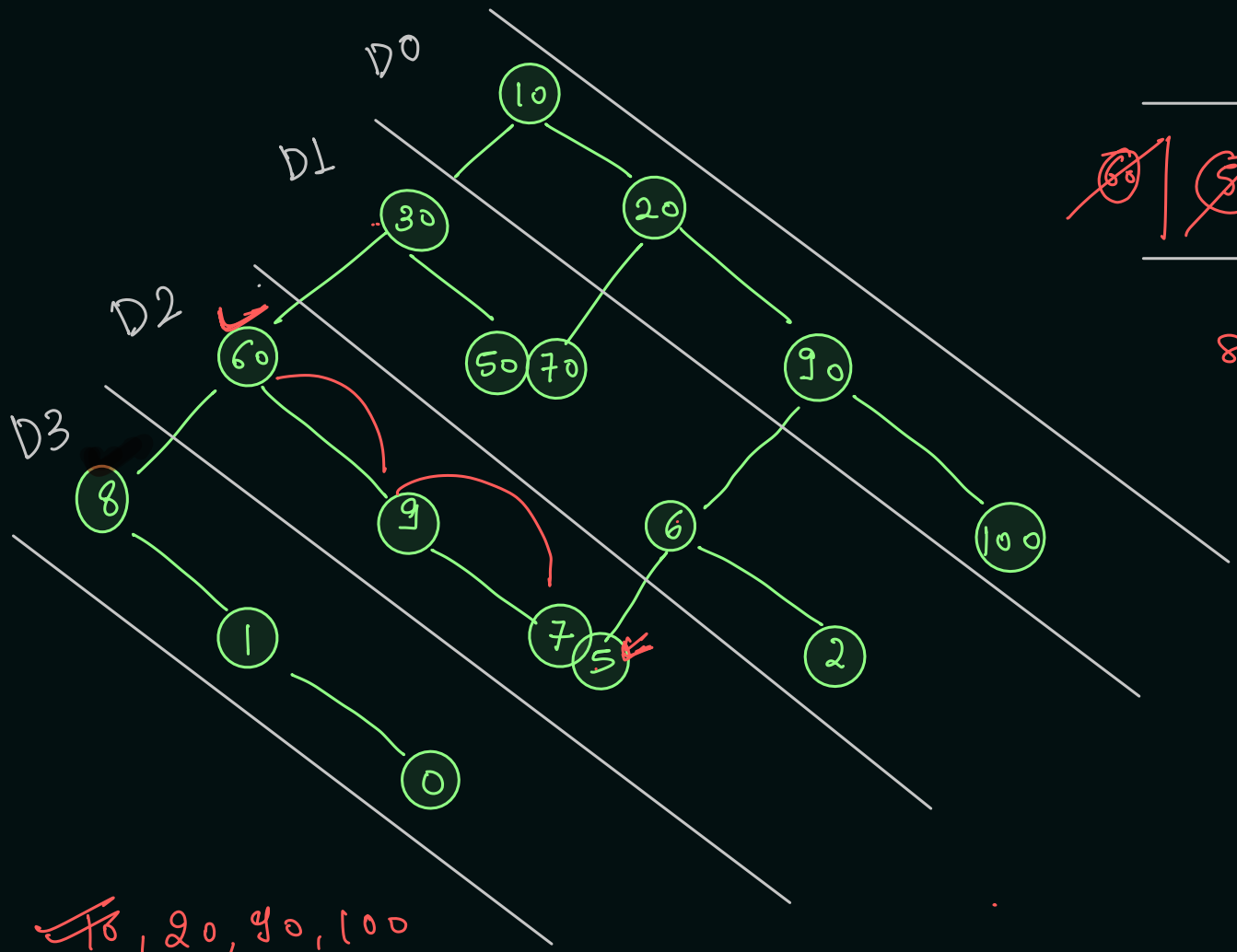
left (1) - increment
right no increment



0 → a, c, g
1 → b, e, h, i, f
2 → d

BFS + HashMap → Done

DFS doesn't work



~~10~~, 20, 90, 100

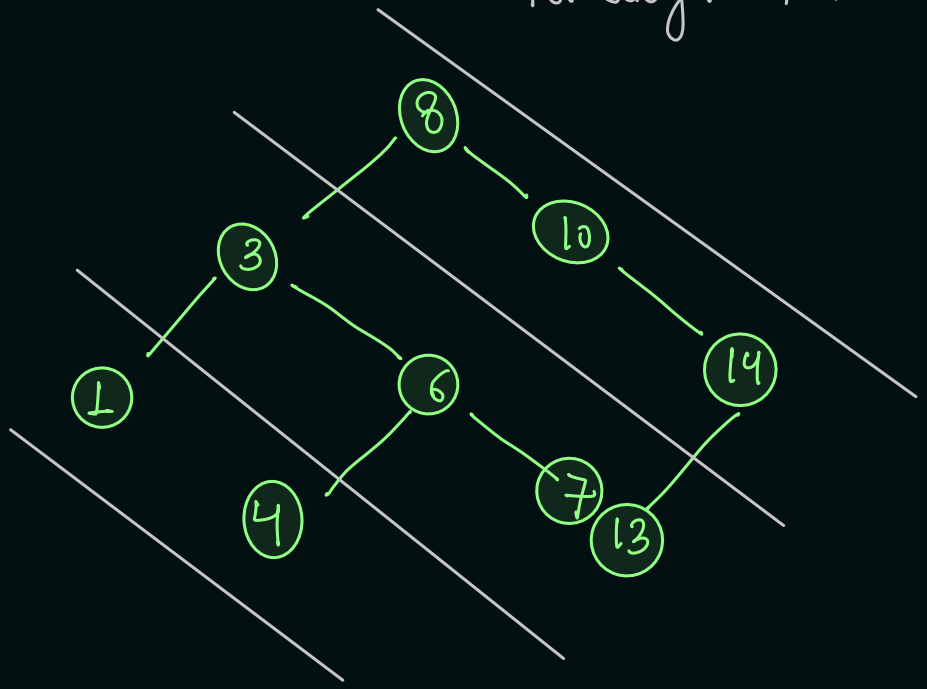
~~30~~, 50, 70, 6, 2

~~60~~, 9, 7, 5

~~60~~ ~~5~~ 8

size 2

For diagonal → move toward Right & add left in queue.




```java
public ArrayList<Integer> diagonal(Node root) {
    Queue<Node> que = new LinkedList<>();
    que.add(root);
    ArrayList<Integer> res = new ArrayList<>();
    while(que.size() > 0) {
        int sz = que.size();
        while(sz-- > 0) {
            Node rem = que.remove();
            while(rem != null) {
                res.add(rem.data);
                if(rem.left != null) que.add(rem.left);
                rem = rem.right;
            }
        }
    }
    return res;
}
```

1 4

sz = 2 1 1

Rem = 8 3

res = 8, 10, 14, 3, 6, 7, 13, 1, 4

Anti diagonal → How?? → move toward left, add Right Node in Que.

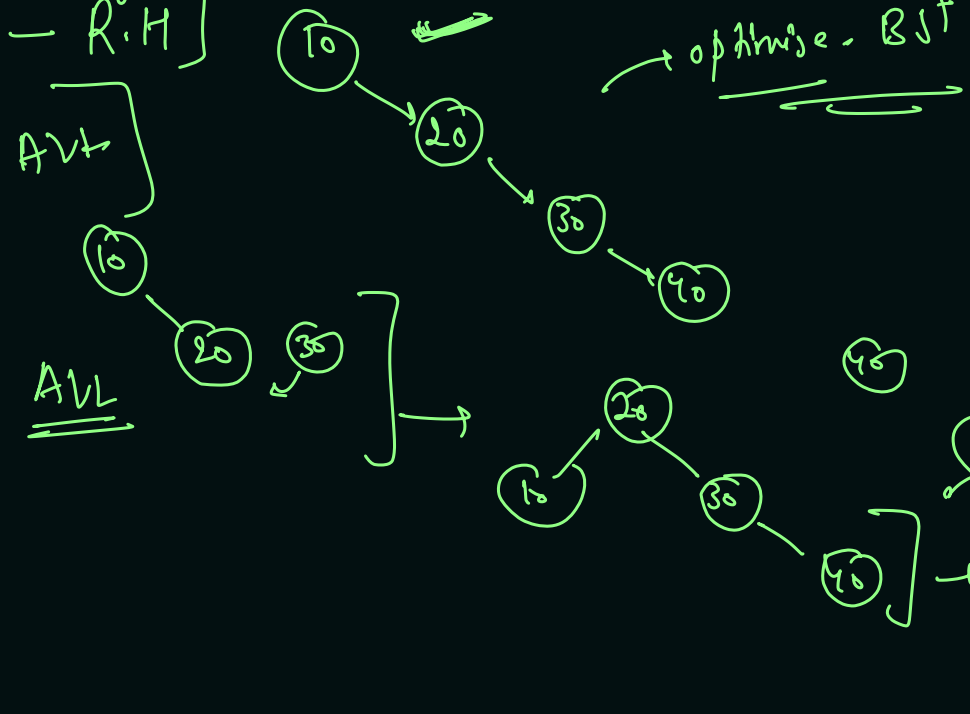
Morning → Tree Complete.

Evening →] → AVL

weekday →] → Thursday → AVL

left Right Height]

L.H. — R.H]



→ optimise. BST

creation.

1 class content

week days.

1. Burning Tree ① & ②
2. Max. path sum
3. Morris traversal → In & Pre
4. BST iterator

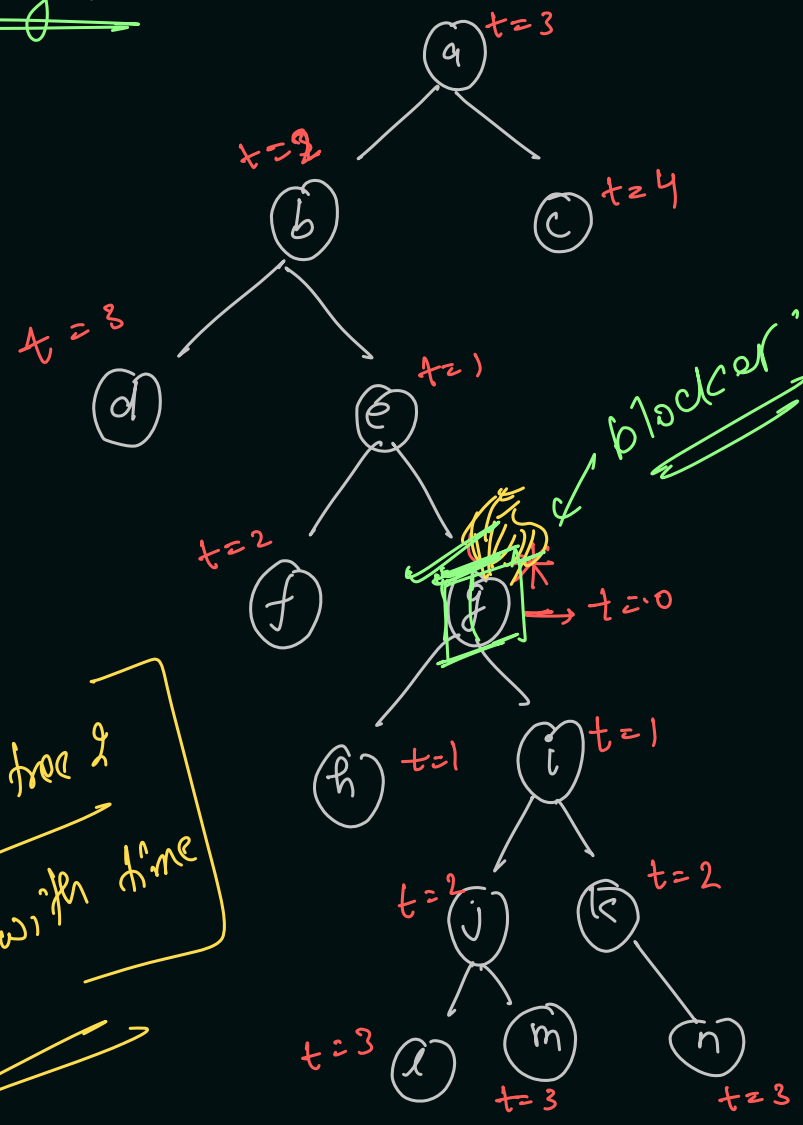
Thursday -

AVL → self Balancing Binary Search Tree

- LL Rotation
- RR Rotation → 2 days
- LR ''
- RL ''

add → remove

Burning Tree



Spreading Rate → Adjacent node per second.

Max. time Required to complete burning of tree.

Max time. = 4 sec

Burning tree & Node with time

Rotten Oranges
Fire in the city
Infection spreading

graph

but Not in tree

Node to Root path. →

max depth → begin from zero

g	e	b	a
0	1	2	3
(3)	(2)	(3)	(4)

Max = (4)

Soln