Recap: Binary Trees → Recursive codes (8)
→ CRUD → Read/Traversal : DFT → Preorder, Inorder, Postorder
BFT → Level order Traversal

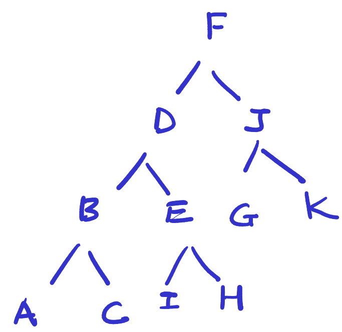
CRUD → Variation of Level Order Traversal (BFT)

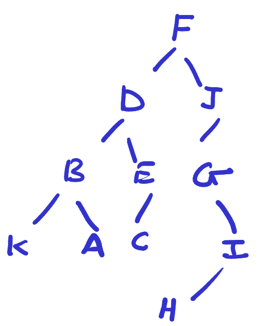
Tree → Traversal (any one of 4 : Pre, In, Post, level)

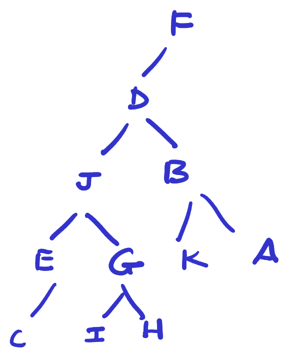
Traversal → Tree (construction of tree) → not coding test
but asked in interviews

1. level order

Traversal ⇒ F D J B E G K A C I H







multiple such trees | can't come up with a unique tree

2. In Pre Post → story remain same .

Non linear → linear

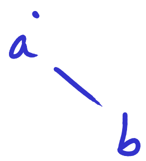
Conclusion : Can't come up unique tree with 1 traversal.
You need at least 2.

pre level ✗	pre in		
in level	post in		6 possibilities .
post level ✗	pre post. ✗		

Trivial case :

pre : [a b]
post : [b a]

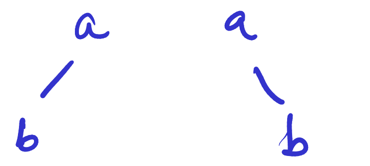
Same for both diff trees.



Conclusion : pre post → 2 traversal can be same for 2 different trees.

Similarly, same applies to combinations pre level & post-level.

*: You need out of 2 traversals, at least 1 to be inorder.



pre: [a b] . [a b]

post: [b a] [b a]

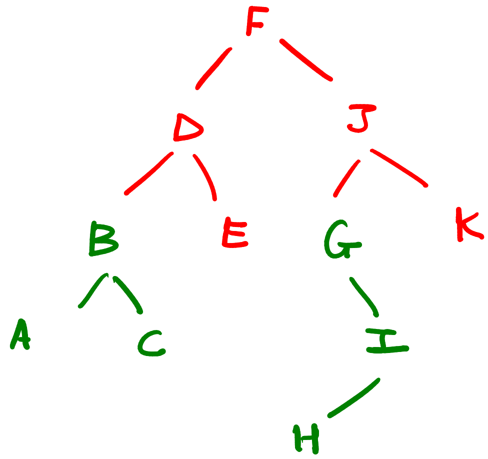
in: [b a] [a b] → diff

Build a tree:

pre: F D B A C E J G I H K

in: A B C D E F G H I J K

left right



1. Find the 1st element of pre or last of post ∵ that is root. → in inorder

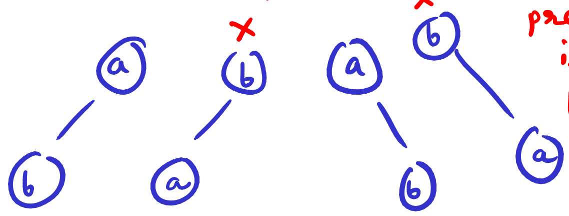
2. Left of this root in inorder will form left ST & others in RST.

Q. 1 traversal → many trees.

Idea: pre: [a b] 2 nodes

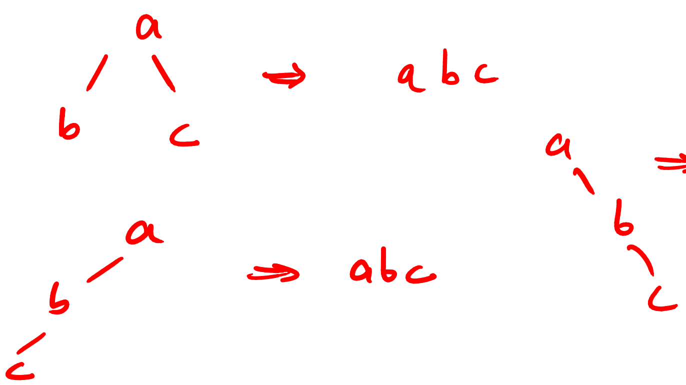
'n' nodes.

count how many such trees are possible?



their preorder is not [a, b]

Total number of BT = 4



→ abc

→ abc

→ abc

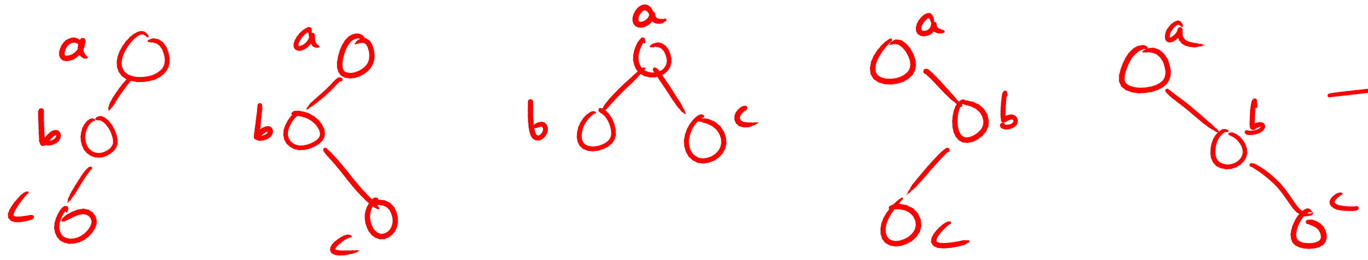
All trees possible → different structurally.

Structurally different: X_n

Now fill your values in such a way that preorder remains same.

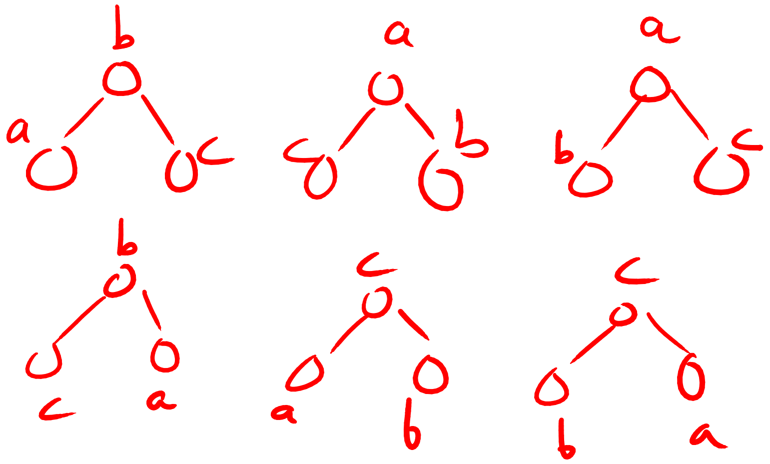
diff label = different

No. of trees = $n! \times X_n$



→ 5

per: [abc]



3!

⇒ same structure

\# diff trees = $X_n \cdot n!$

↓ different structures.

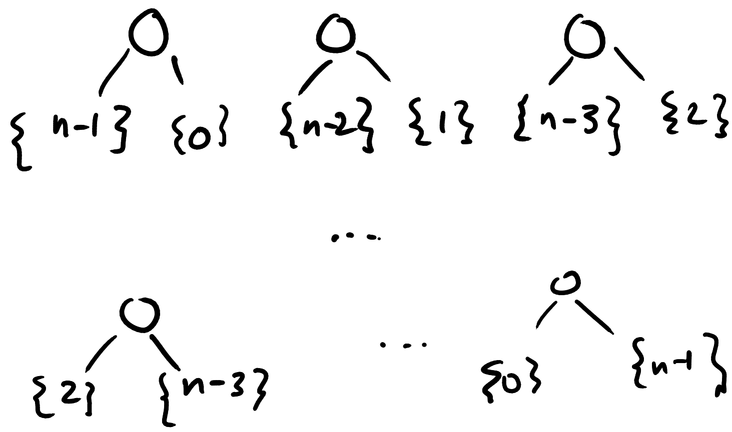
for $n=3$, $X_n = 5$.

6 different trees with same structure.

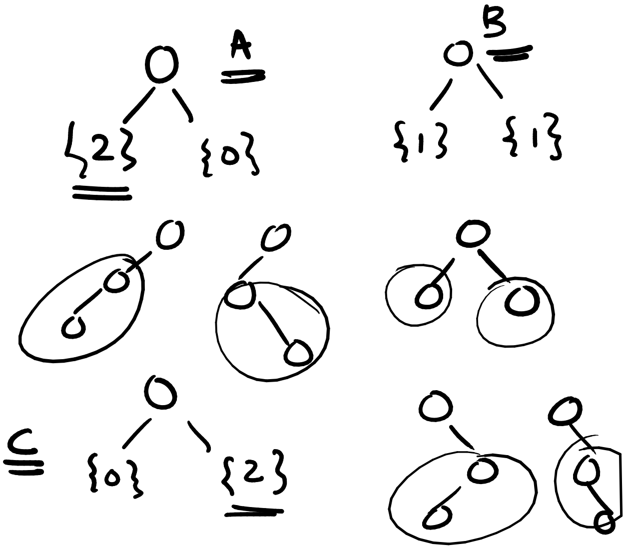
X_n → lets try to find. (Template → my fav) | Samsung MCM

n nodes.

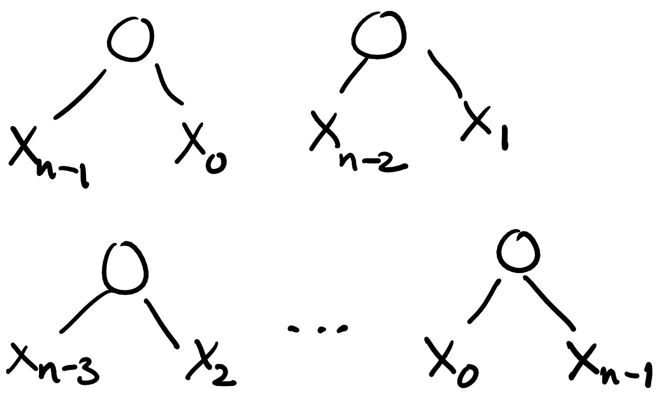
1 node surely has to be root.
n-1 nodes that I can arrange



n=3

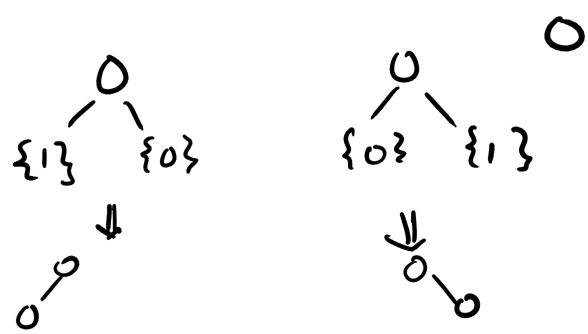


X_n ⇒

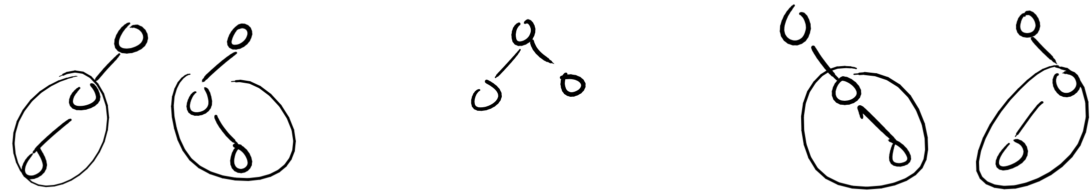


n=2

n=1

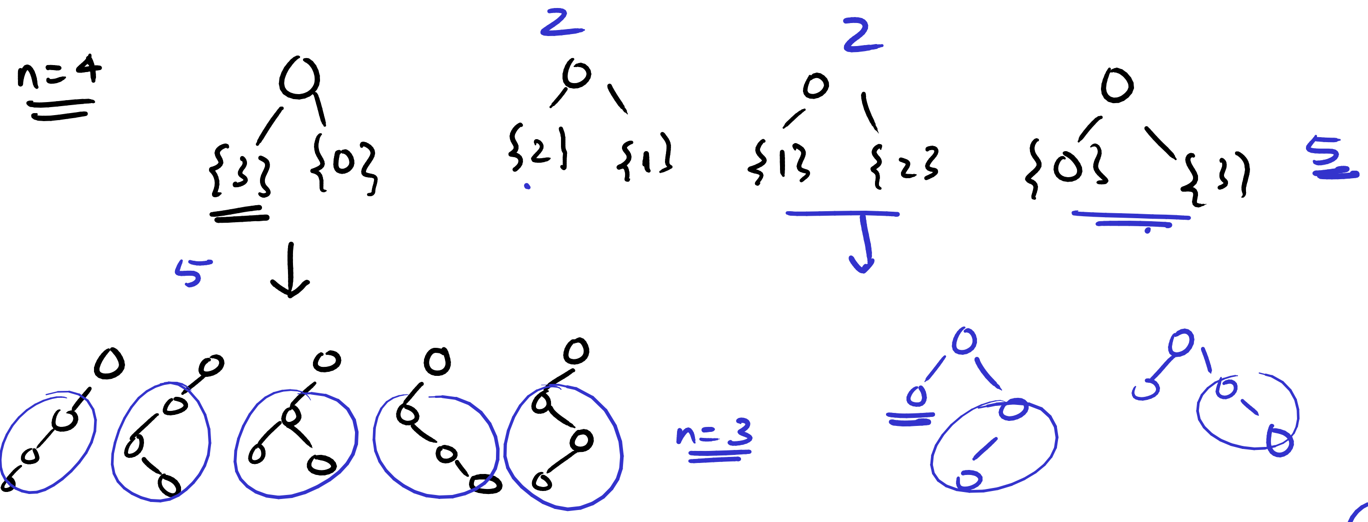


n=3   5

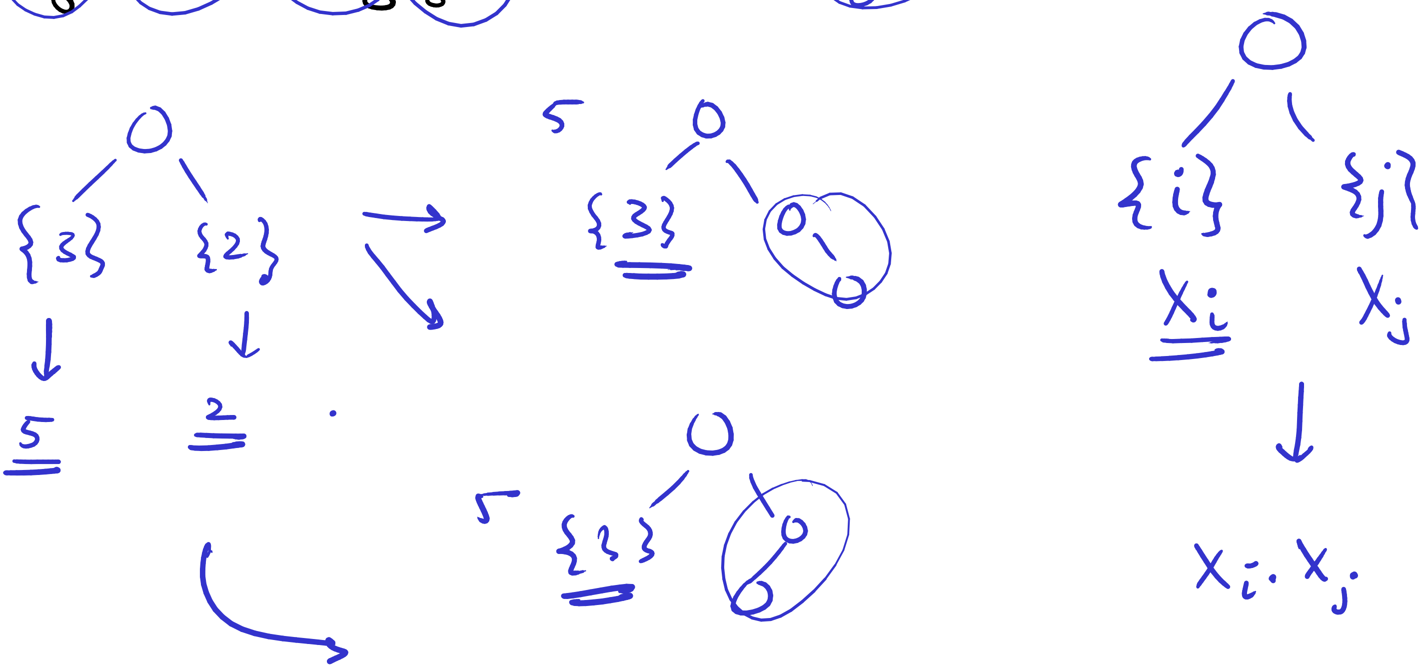


→ 5

n=4



$X_n = 14.$



Total $X_3 . X_2 = 5.2 = 10$

$$X_n = X_{n-1} \cdot X_0 + X_{n-2} \cdot X_1 + \cdots + X_1 X_{n-2} + X_0 X_{n-1}$$

$$\boxed{C_n = \sum_{i=0}^{n-1} X_{n-1-i} \cdot X_i} \Longrightarrow \text{Catalan Number.}$$

$C_0 = C_1 = 1$

$n!$ → for any 1 structure, the no. of ways you can arrange labels.

$n \Rightarrow \quad C_n\ [n+1] \qquad c[0] = c[1] = 1 \qquad n = 5$

for $i = 2, n-1, i{+}{+}$:

for $k = 0, i, k{+}{+}$:

$c[i] \mathrel{+}= c[k] * c[i-k-1]$

ans $= c[n]$ $\qquad$ T.C. $O(n^2)$

Dry Run $\longrightarrow$

1	1	2	5	14	

$C_2 \quad C_3$

$C_0C_1 \quad C_1C_0$

$1.1 + 1.1$

$i = 3$ $\quad C_0C_2 \quad C_1C_1 \quad C_2C_0$

$1.2 + 1.1 \quad 2.1$

$2 + 1 + 2$

$i = 4$ $\quad C_3C_0 \quad C_2C_1 \quad C_1C_2 \quad C_0C_3$

$5.1 \quad 2.1 \quad 1.2 \quad 1.5$

Variation #1 : n pairs of balanced parentheses, how many balanced expressions are possible?

Samsung.

$n = 2$ $\qquad$ (()) $\quad$ ()() $\qquad$ O/p: 2

n-1 pairs

Idea: $\quad (\{n-1\})\{0\} \qquad (\{n-2\})\{1\}$

$(\{n-3\})\{2\} \ \ldots \qquad (\{0\})\{n-1\}$

2 pairs.

Eg. $n = 3$

$(\{2\})\{0\} \rightarrow$ (()()) $\quad$ ((()))

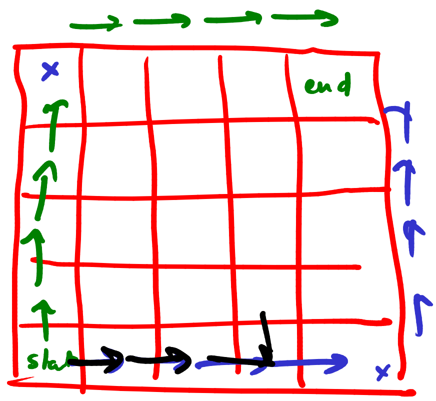
$(\{1\})\{1\} \rightarrow$ (())()

$(\{0\})\{2\} \rightarrow$ ()()() $\quad$ ()(())

Ans. C_n

Variation #2. square n x n grid.

How many ways? choices. n-1



right

top

$$C_n = \underline{C_{n-1}} \cdot C_0 + C_{n-2} \cdot C_1$$

right top

$$C_{n-3} \cdot C_2 + \cdots + C_0 C_{n-1}$$

Ans. C_{n-1}

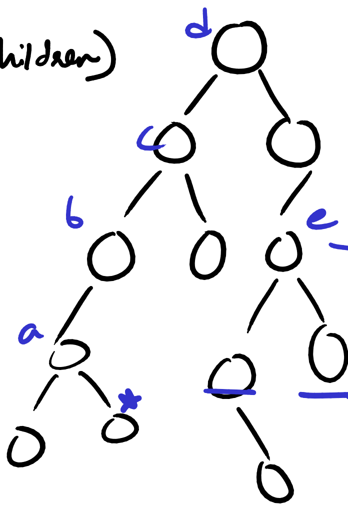
BT : Hierarchy. Family (1 child or 2 children)

Terminology : parent

parent all the way to start → ancestors.

siblings ⇒ same parent

cousins ⇒ same level b and e



ancestors

→ parent

→ children

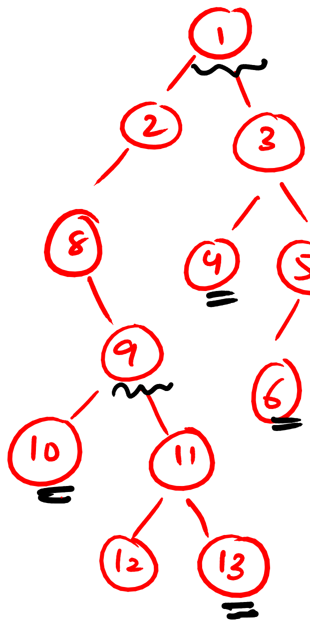
LCA (least common ancestor) → Highly asked Test/Interview

Super easy

LCA(6,7)	→	5
LCA(10,13)	→	9
LCA(4,6)	→	3
LCA(9,7)	⇒	1
LCA(8,9)	⇒	8
LCA(11,2)	⇒	2

intersection

edge cases



10,13

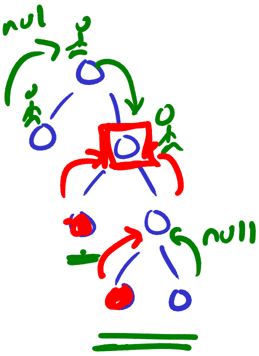
logic
↓
LCA(node1, node2)
root ↓
search for n1 & n2
if both belong to left ST, I will go left
Both right → right

One in left, one in right ⇒ you are on LCA!

```
Node find_lca (Node root, int n1, int n2):
    if root == null : return null ;          leaf node
    if (root.data == n1 || root.data == n2) : return root

    Node left_st = find_lca (root.left, n1, n2)
    Node right_st = find_lca (root.right, n1, n2)

    if left_st != null and right_st != null : return root
    if left_st != null : return left_st
    return right_st
```



null

O(n)

k^{th} ancestor :-

ancestors (10, 13) = 9, 8, 2, 1

1st ancestor

3rd ancestor

= 2 (3rd ancestor)

$n_1, n_2, k \rightarrow$ i/p :

5th ancestor $\Rightarrow$ −1 (doesn't exist) .

Toughest !!

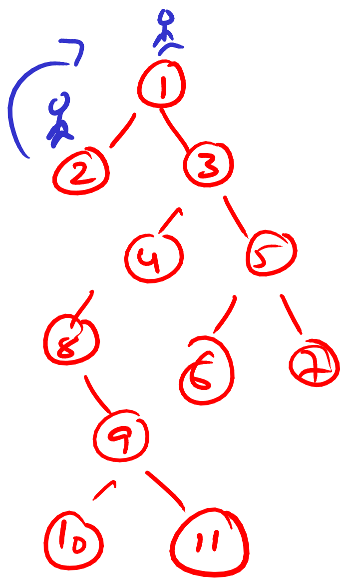
→ ① put all ancestors in the list. → not as easy !

any node → find root to node path.

find_path (root, 9) : [1, 3, 4, 8, 9]

find_path (root, lca.data)

key



Recall Backtracking !

path ← node.data when moving downward

when coming backwards → pop from path

if root.data == key : found an ans.

```
bool root_to_node (Node root, int key):                int path[ ].
    if root == null : return false
    if root.data == key : return true

    path.push (root.data)    // potential ans
    if (root_to_node (root.left, key) || root_to_node (root.right,
                                                      key))
        return true.

    path.pop ( )
    return false
```

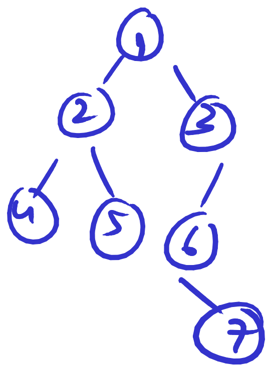
Root to all leaves : All branches.   2D paths[ ][ ];
                                                  global / pass it
                                                  an arg

```
                         ↙ temp. 1D array.
void root_to_leaves (root, path[ ]) :
    if root == null : return;
    Path.push (root.data)

    if root.left == null and root.right == null :
O(n)    all_paths.push (path)

    root_to_leaves (root.left, path)
    root_to_leaves (root.right, path)

    path.pop ( )
```



o/p :
[[1, 2, 4]
  [1, 2, 5]
  [1, 3, 6, 7]]

BT Concludes.