

# Operations on Binary Tree - Part I

Course on C-Programming & Data Structures: GATE - 2024 & 2025

# Data Structure

## Tree 2

By: Vishvadeep Gothi

# Tree Terminologies

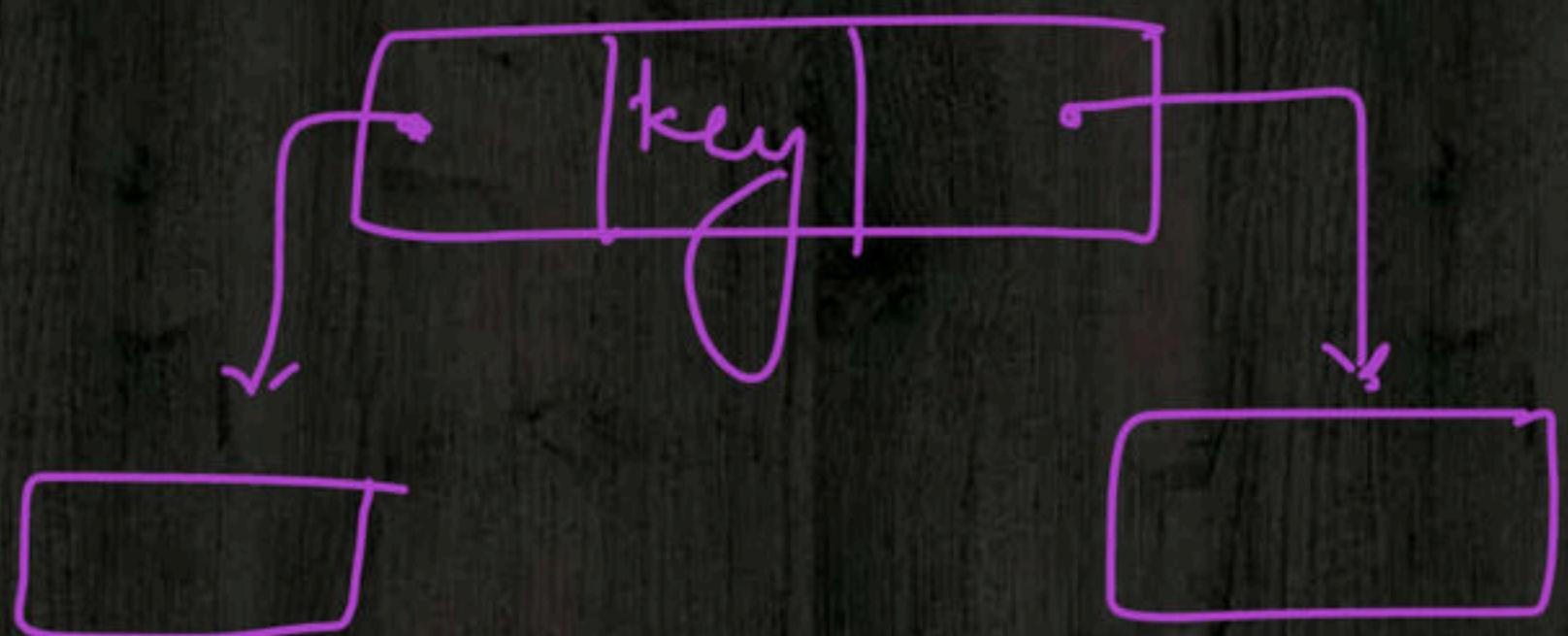
- Root
- Child
- Parent
- Sibling
- Subtree
- Internal Node
- External Node
- Level

# Binary Tree



every node => max 2 children

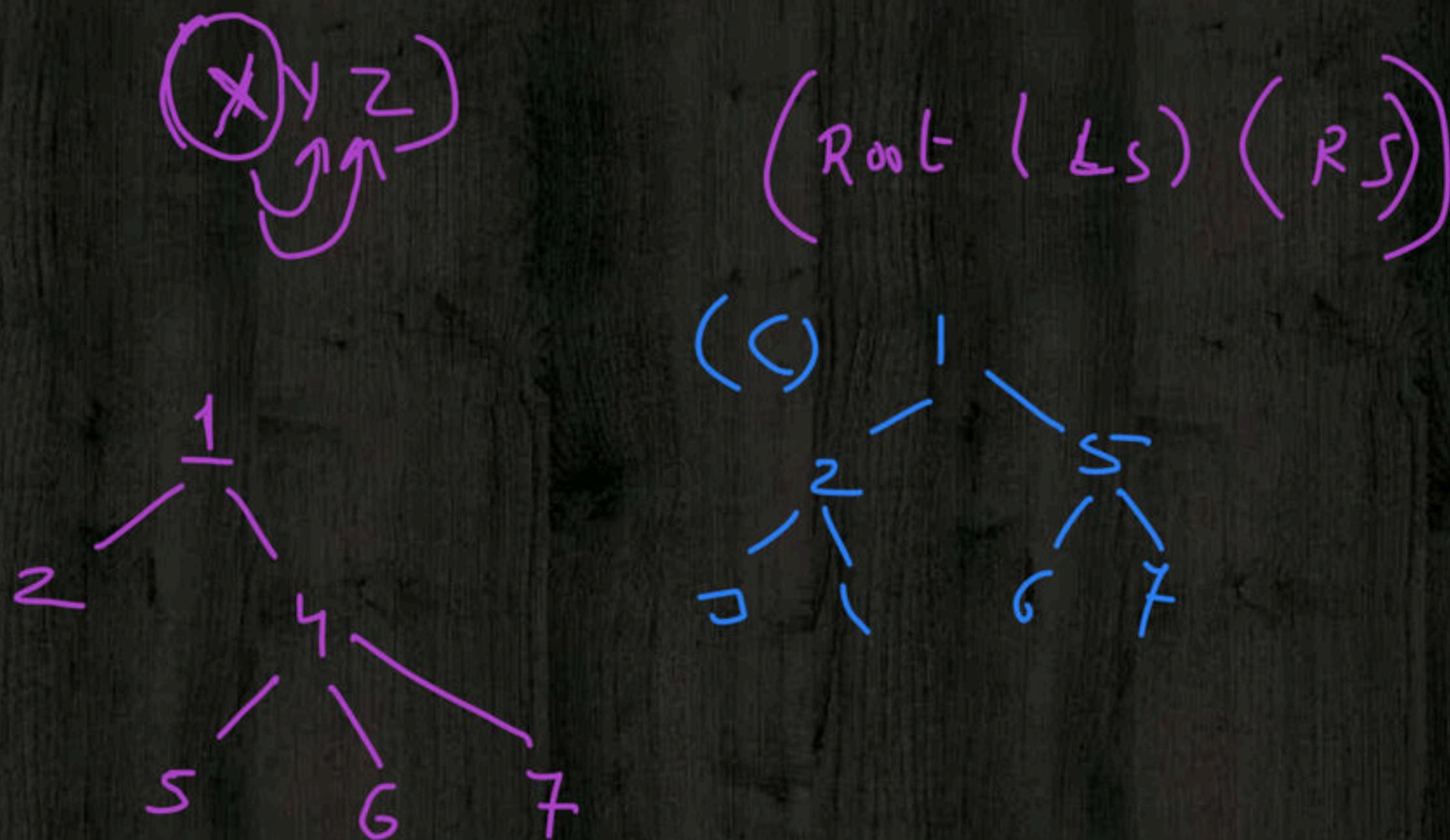
Linked representation



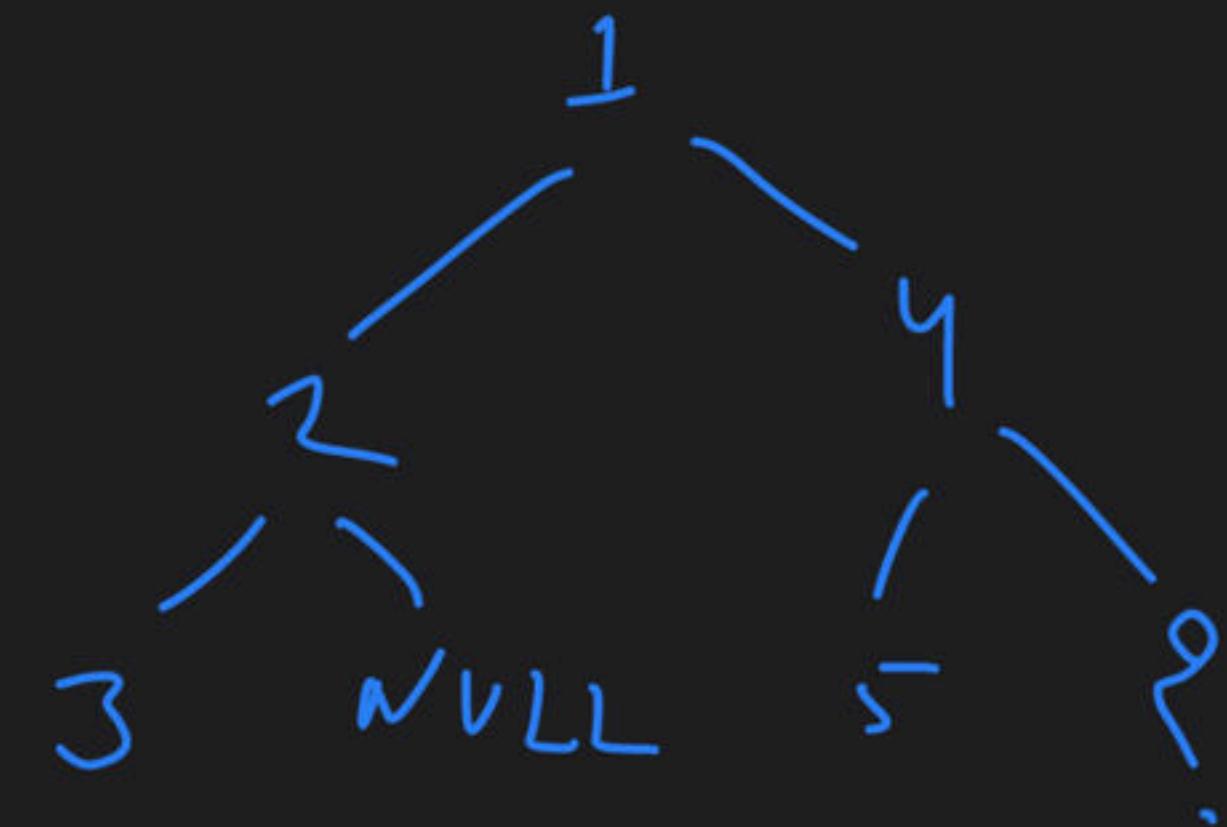
# Question GATE-2000

Consider the following nested representation of binary trees: (X Y Z) indicates Y and Z are the left and right subtrees respectively of node X. Note that Y and Z may be NULL or further nested. Which of the following represents a valid binary tree?

- (1 2 (4 5 6 7))
- (1 ((2 3 4) 5 6) 7)
- (1 (2 3 4)(5 6 7))
- (1 (2 3 NULL)(4 5))

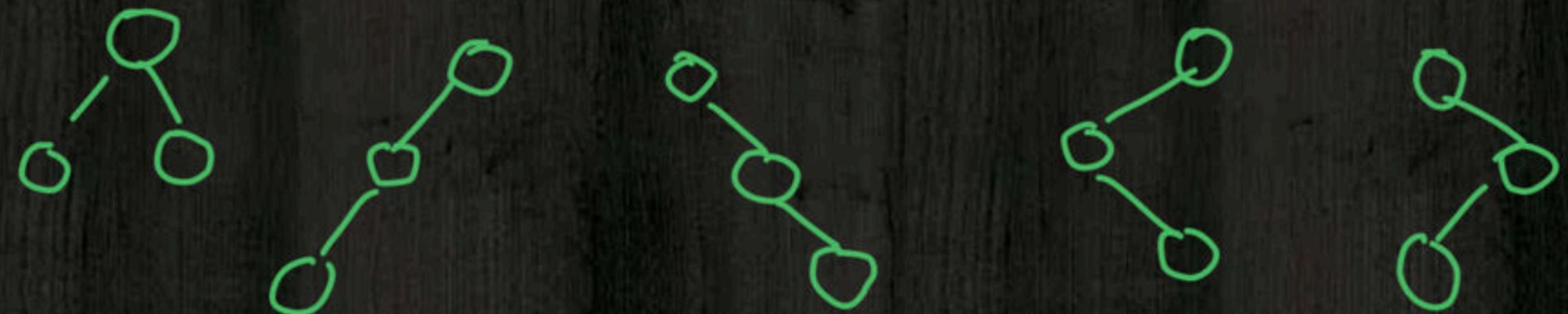


(2)



# Question

How many binary tree can be constructed using 3 unlabeled nodes?



Ans = 5

---

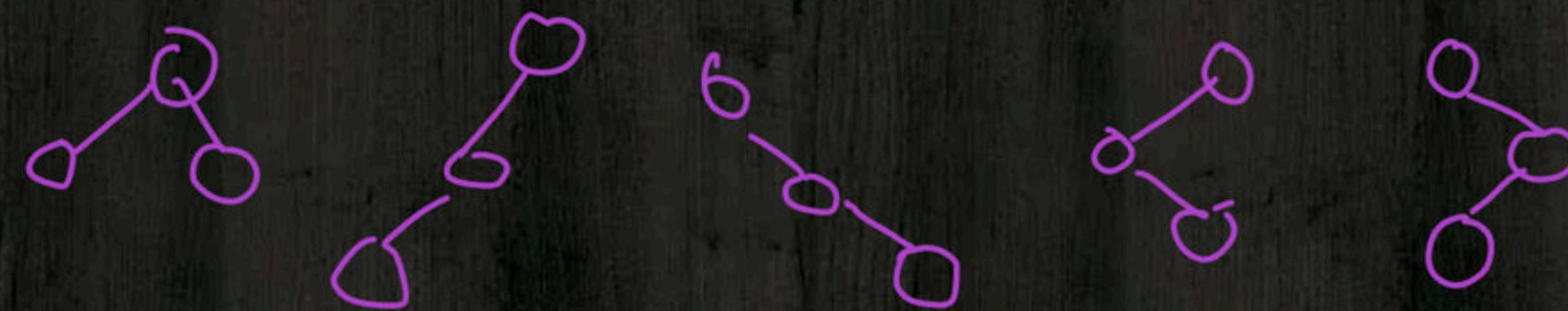
for  $n$  unlabeled nodes  $\Rightarrow$

$$\text{no of distinct BTs} = \frac{^{2n}C_n}{n+1}$$

# Question

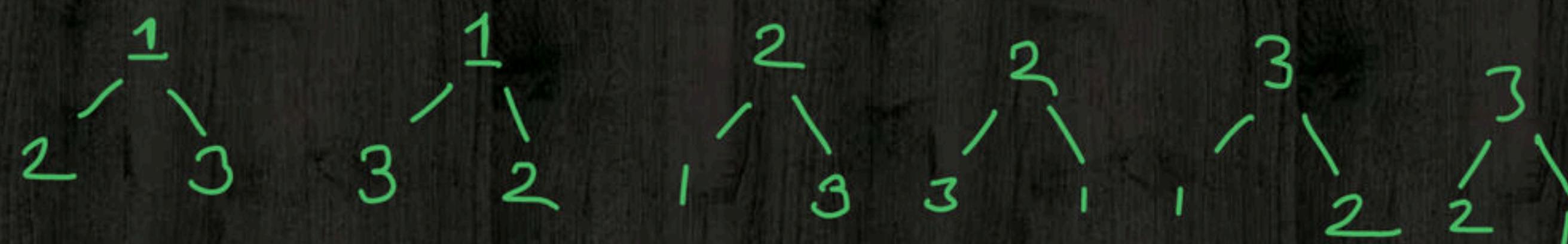
distinct

How many binary tree can be constructed using 3 distinct keys?



$$\begin{aligned} \text{Ans} &= 5 * 6 \\ &= 30 \end{aligned}$$

keys = 1, 2, 3



$n!$  ways to populate one BT with  $n$  keys

for  $n$  distinct keys,

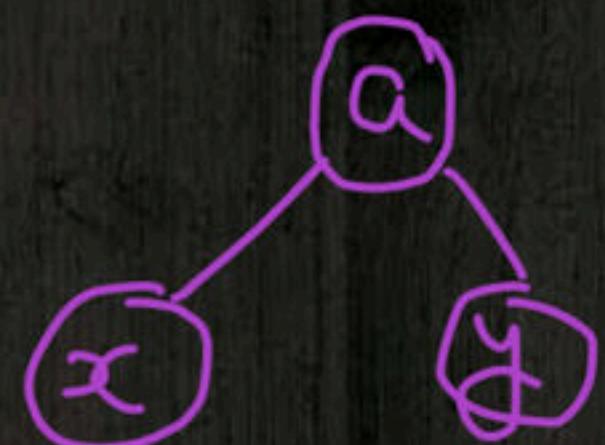
$$\text{no of distinct BST} \Rightarrow \frac{C_n}{n+1} * n!$$

# Tree Traversals

① Preorder traversal  $\Rightarrow nLR \Rightarrow axy$

② Inorder —————  $\Rightarrow L_nR \Rightarrow xay$

③ Postorder —————  $\Rightarrow LRn \Rightarrow xya$

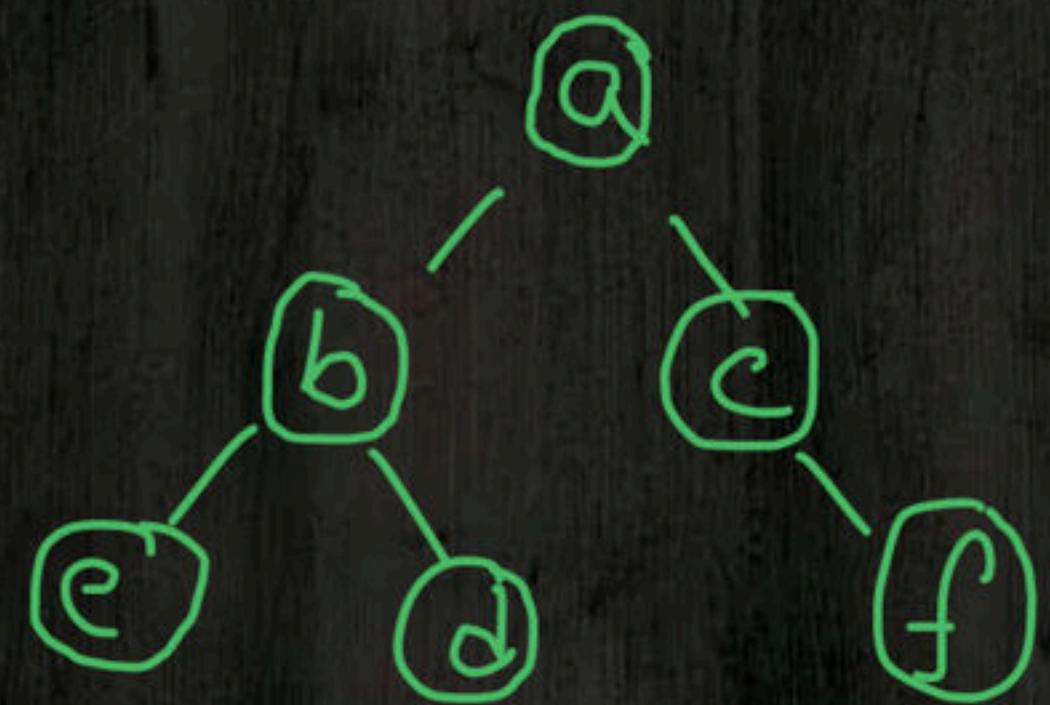


# Preorder

```
void Preorder(struct BTNode *t)
{
    if(t)
    {
        printf("%d", t->data);
        Preorder(t->LeftChild);
        Preorder(t->RightChild);
    }
}
```

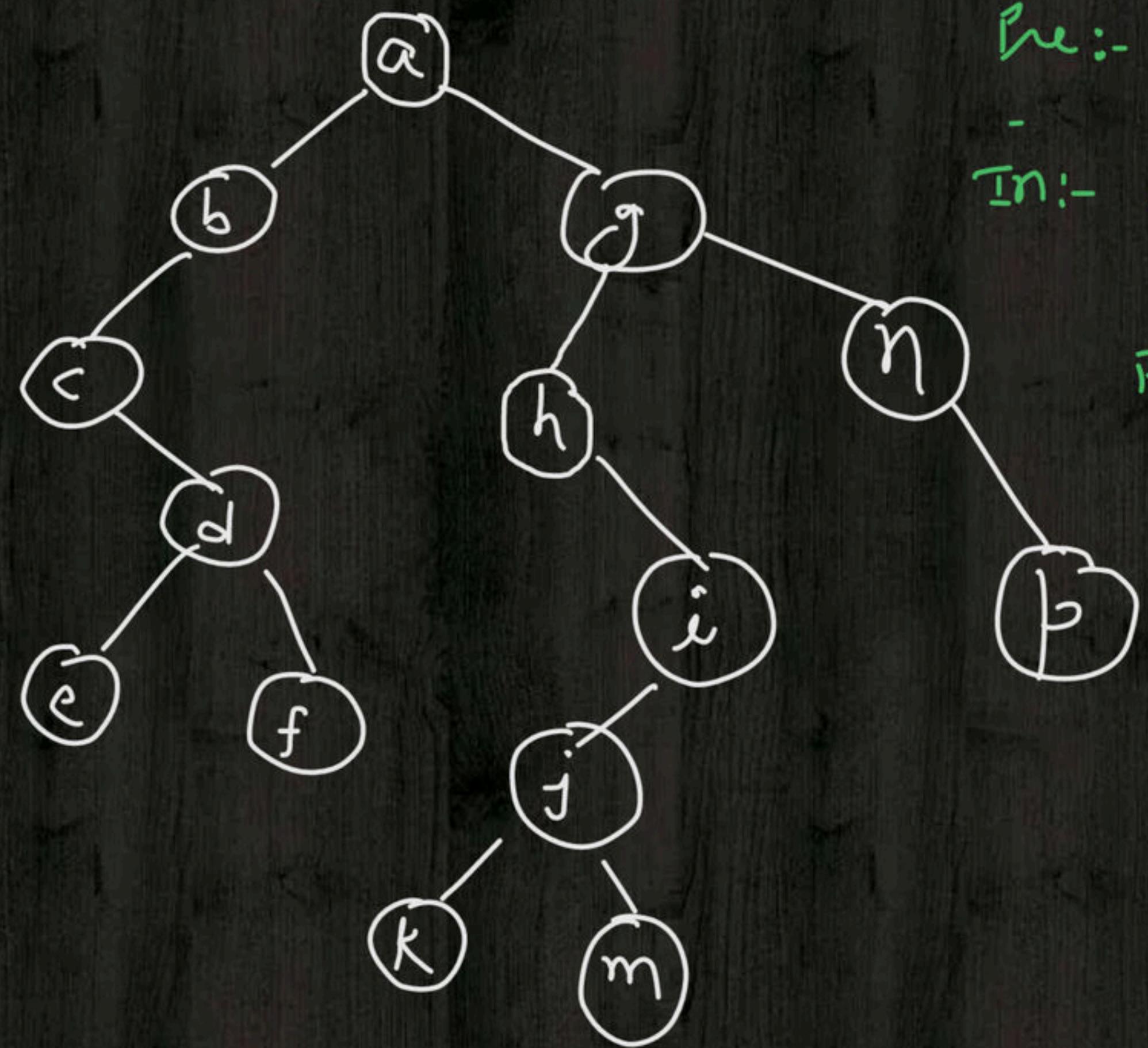


# Question



Pre:- abedcf  
—  
In:- ebdaacf  
—  
Post:- edbfcfa

# Question



Pre:- abcdefghijklmnþ

In:- cedfbahkjlmignþ

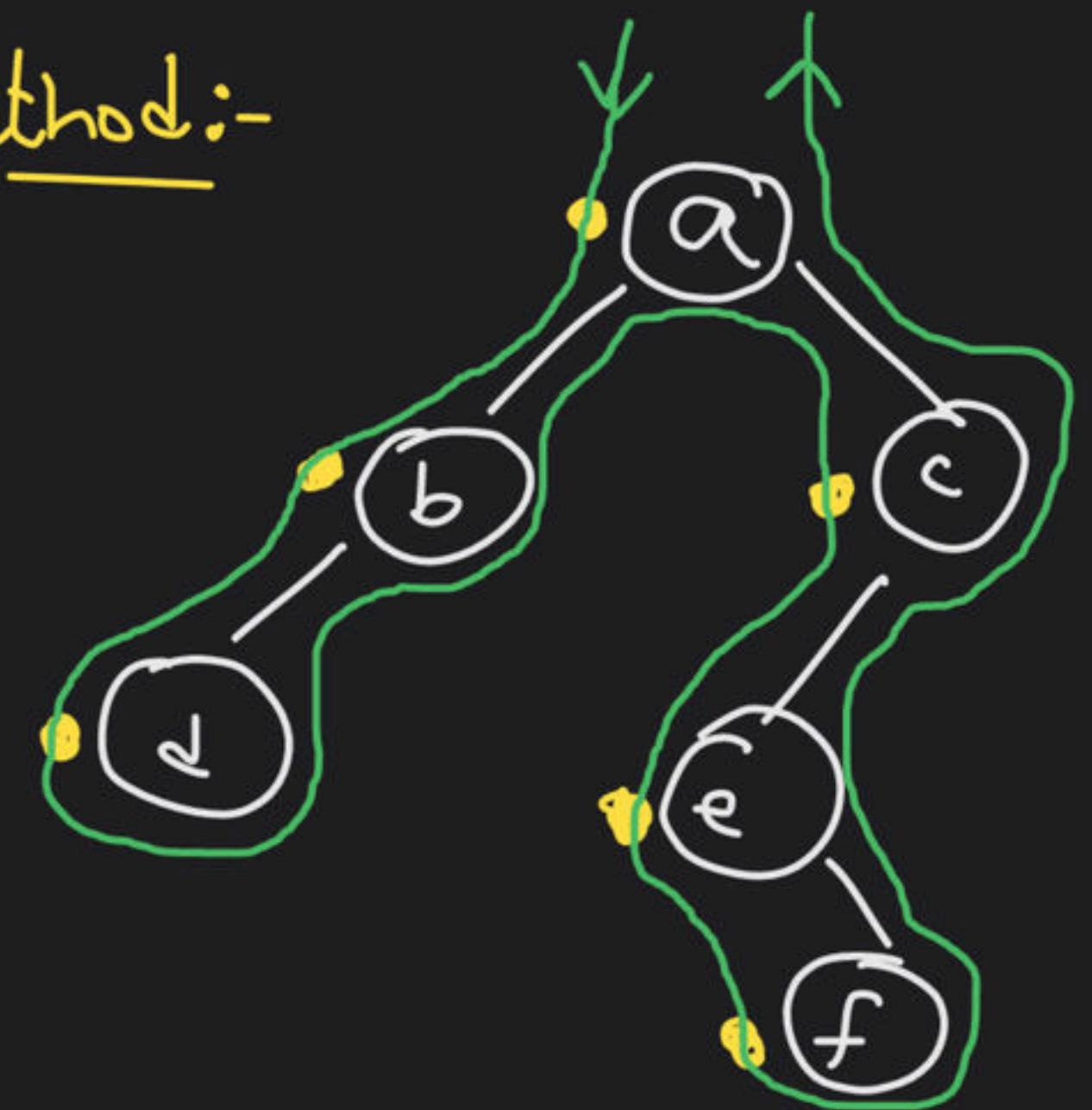
Post:- efcdcbkmnjilþngþ

Preorder:-

node traversed before left & right subtree

Traversal direction  $\Rightarrow$  left to right

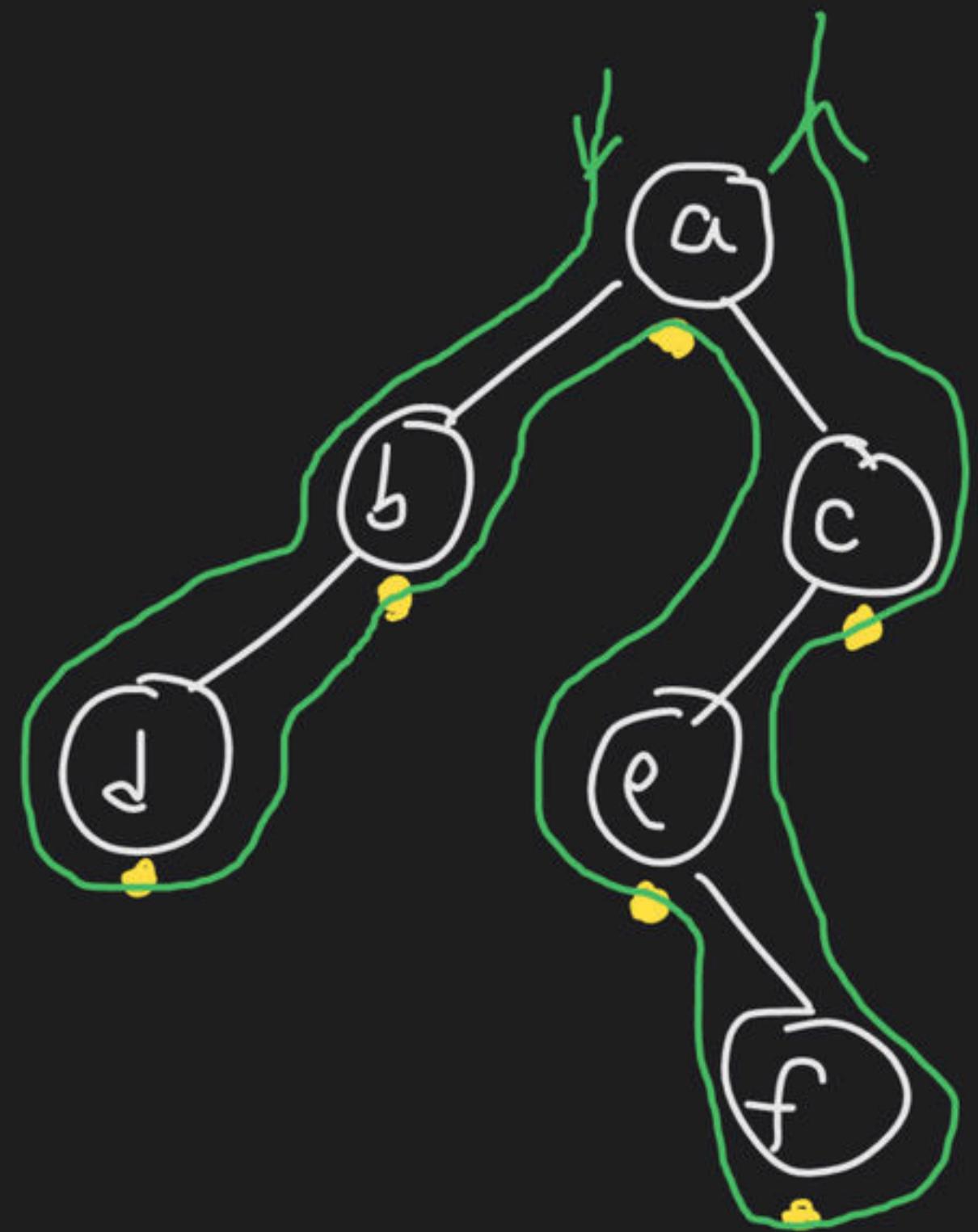
Dot method:-



abdcef

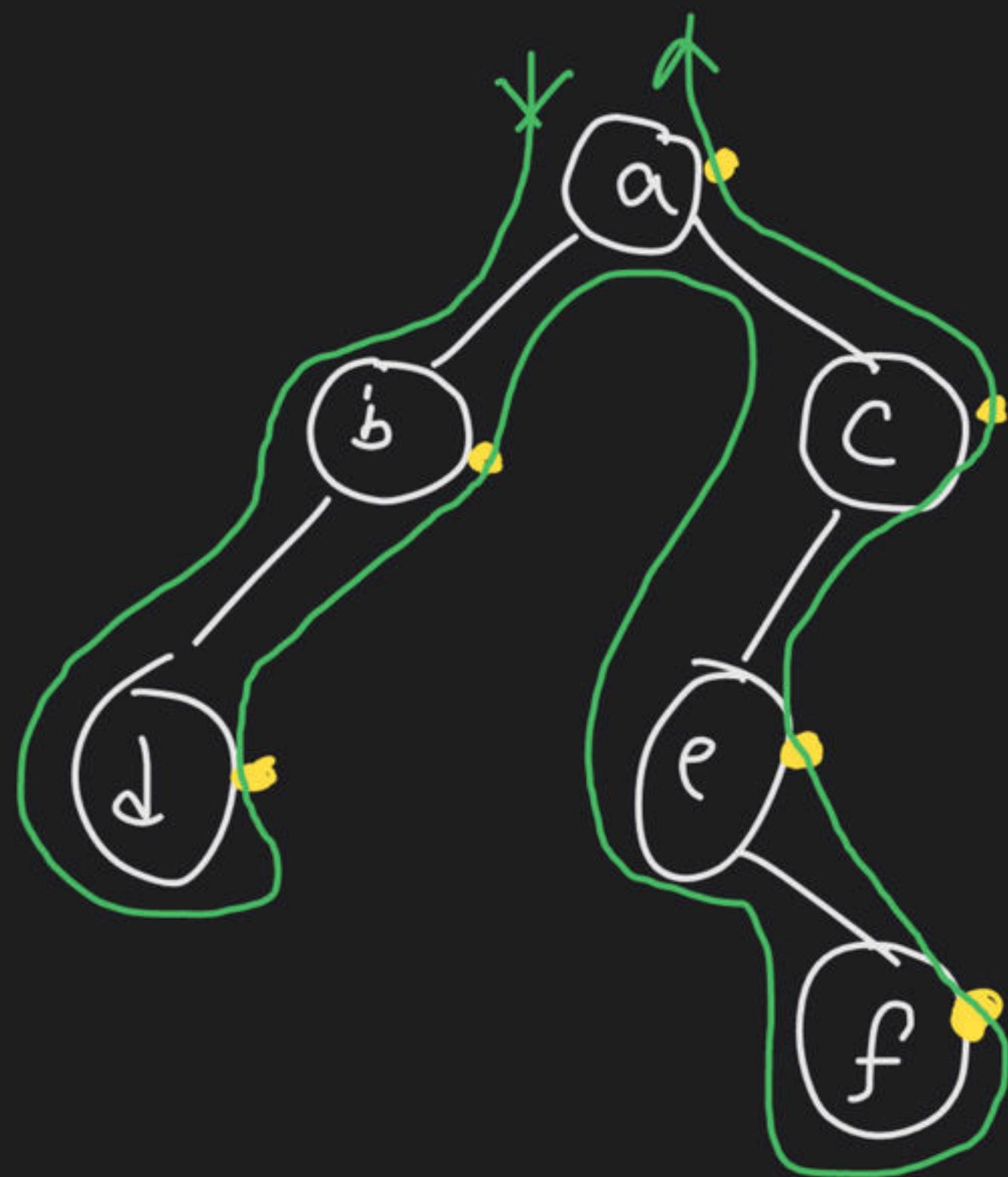
Inorder:-

node traversed  
in b/w left & right subtree



In:- d b g e f c

Postorder:- node is traversed after Left & Right subtree



dbfec a

# Observation from Traversal

$\Rightarrow$  first symbol of preorder traversal  $\Rightarrow$  Root of tree

$\Rightarrow$  Last — || — postorder — if  $\Rightarrow$  Root of tree

# Question

Identify the inorder, preorder and postorder traversal:

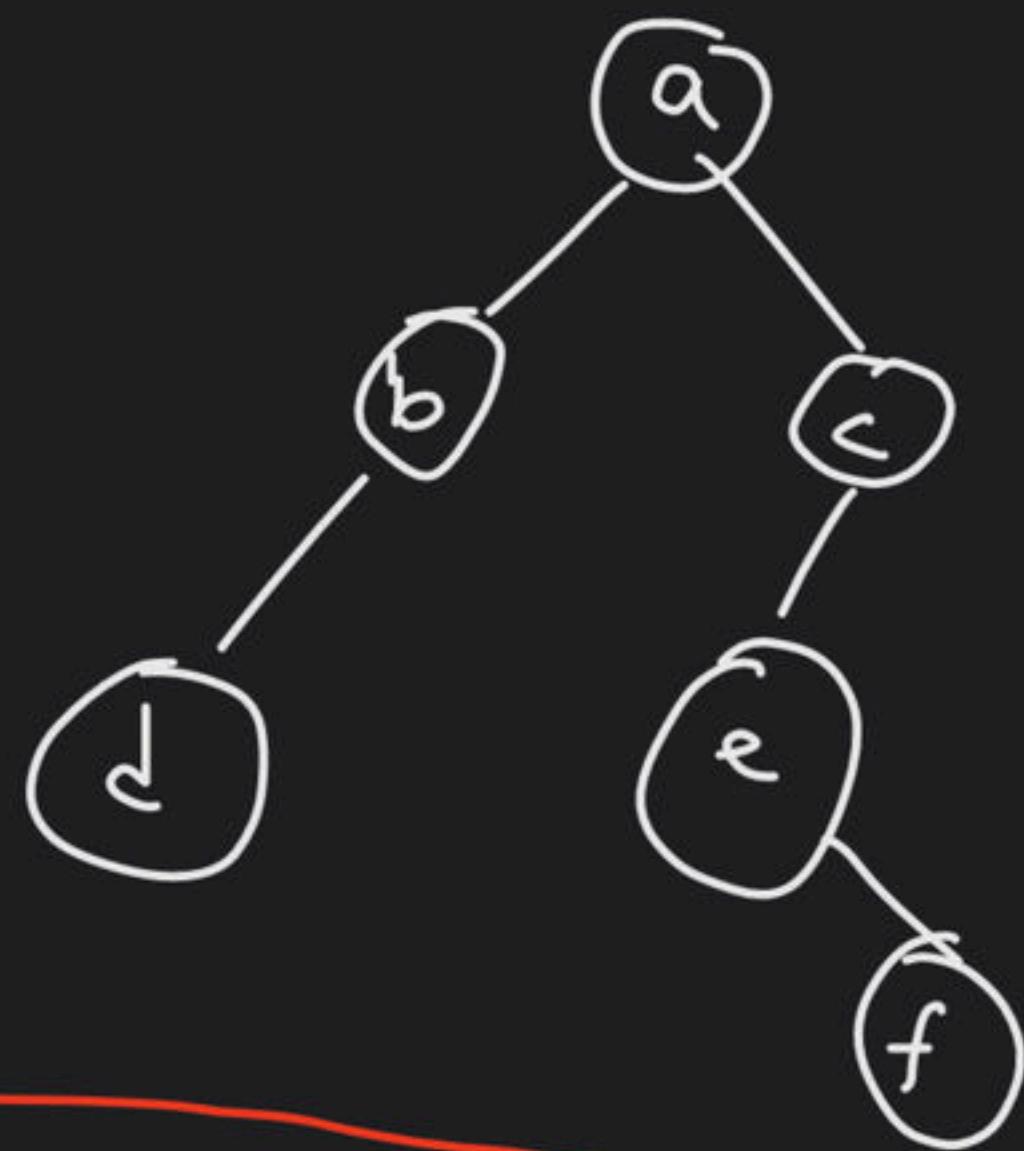
1. GHFEAPKDCBXM → Post
2. EGFHAPMCKDXB → In
3. MPAEFGHXCDKB → Pre

# Converse Order Traversal

Converse Preorder  $\Rightarrow$  nRL

Converse Inorder  $\Rightarrow$  RnL

Converse Postorder  $\Rightarrow$  RLn



Converse Preorder :- ac e f b d  
Converse Inorder :- c f e a b d  
Converse Postorder :- f e c d b a

conventional

Preorder :- a b d c e f  
Inorder :- d b a e f c  
Postorder :- d b f c e a

```
void ConversePreorder ( struct BTnode *t )  
{  
    if ( t )  
    {  
        printf( "%c", t->data );  
        conversepreorder ( t->Rightchild );  
        conversepreorder ( t->Leftchild );  
    }  
}
```

# Observation from Converse Order Traversal

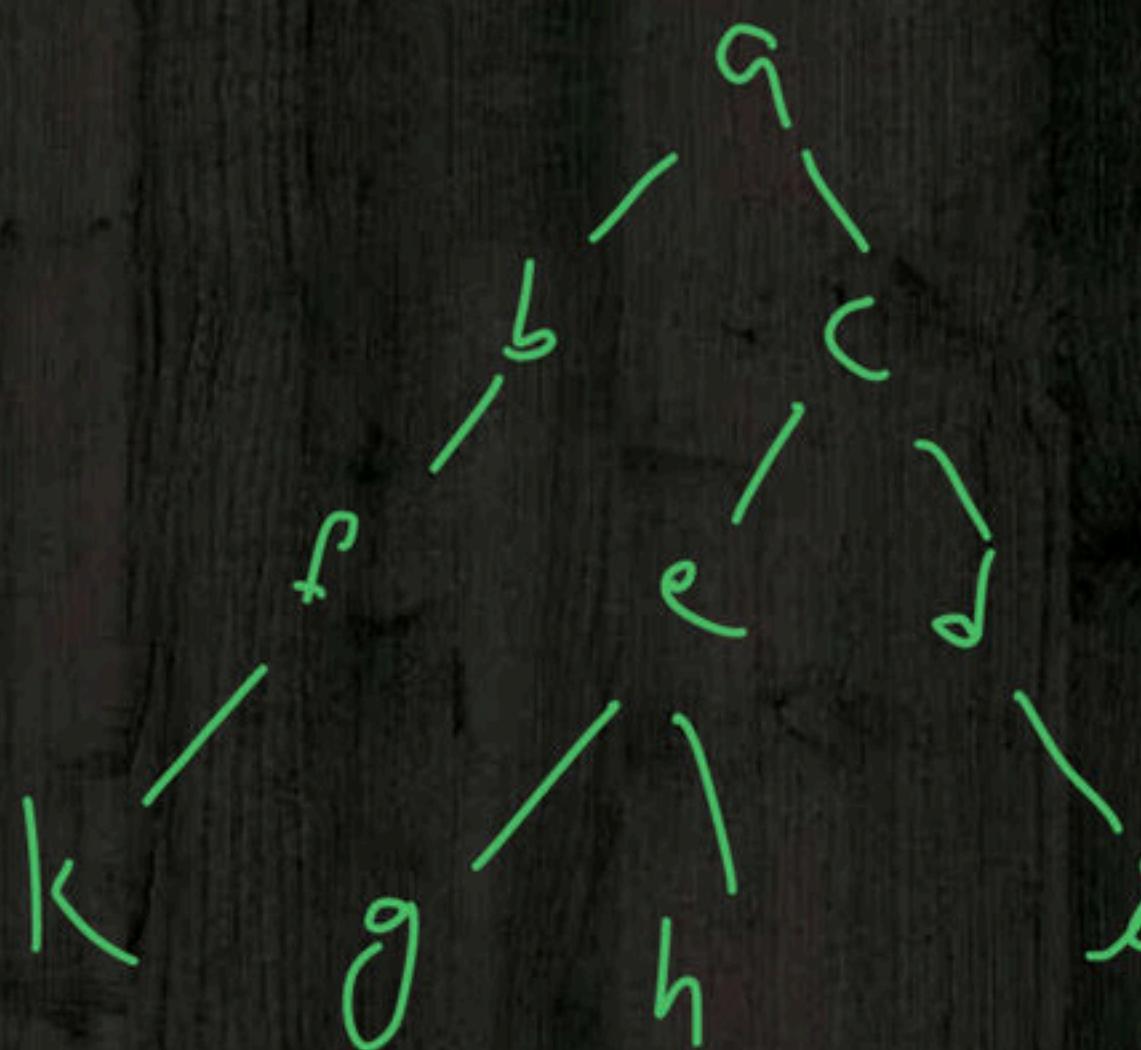
Reverse of converse pre-order traversal  $\Rightarrow$  conventional postorder

$\overline{\text{---}} \text{---} \overline{\text{---}}$  postorder  $\overline{\text{---}} \text{---} \Rightarrow \overline{\text{---}} \text{---} \overline{\text{---}}$  pre-order

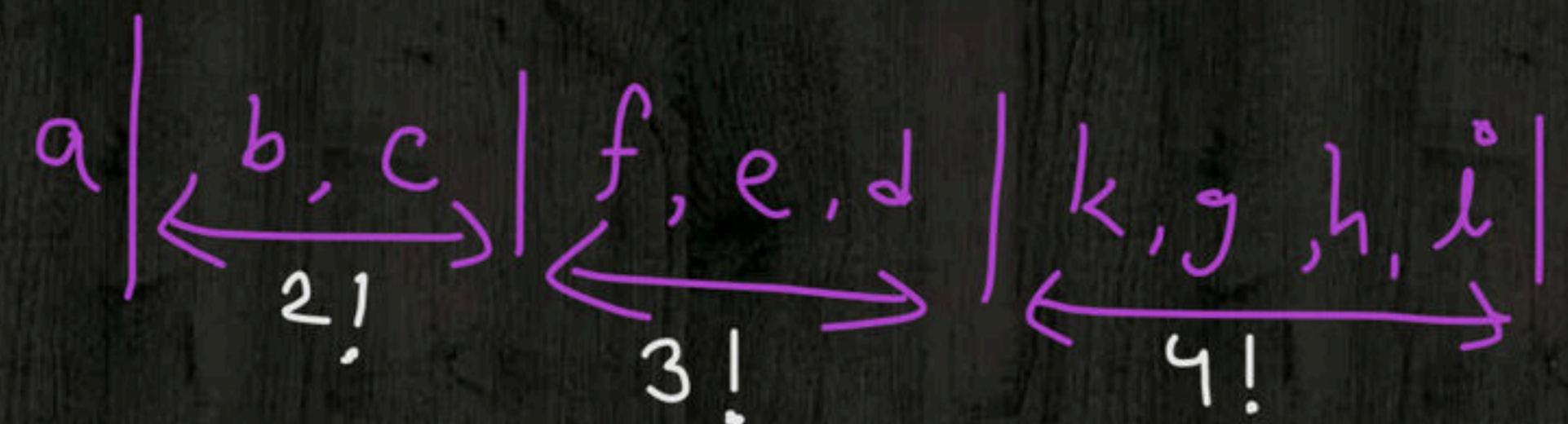
$\overline{\text{---}} \text{---} \overline{\text{---}}$  morde  $\overline{\text{---}} \text{---} \Rightarrow \overline{\text{---}} \text{---}$  Inorder

# Level Order Traversal

Starting from level 0, traverse all nodes of level no.  $L$  before traversing any node of higher level no.



a, b, c, f, e, d, k, g, h, i



no of distinct  
level order  
traversals possible =  $\frac{1+2!+3!}{4!} = 288$

# Data Structure: Tree 3

By: Vishvadeep Gothi



*Hello!*

# I am Vishvadeep Gothi

I am here because I love to teach

# Question

How many binary tree can be constructed using 3 unlabeled nodes?

# Question

How many binary tree can be constructed using 3 distinct keys?

# Conversion of General Tree to Binary



# Tree Traversals

# Preorder

```
void Preorder(struct BTNode *t)
{
    if(t)
    {
        printf("%d", t->data);
        Preorder(t->LeftChild);
        Preorder(t->RightChild);
    }
}
```

# Question

# Question

# Observation from Traversal

# Question

Identify the inorder, preorder and postorder traversal:

1. GHFEAPKDCBXM
2. EGFHAMCCKDXB
3. MPAEFGHXCDKB

# Converse Order Traversal

Converse Order Traversal is a traversal pattern that visits nodes in a tree in a specific order. It starts at the root node and traverses the tree by visiting the right child of each node before its left child. This results in a traversal order that is the converse of Inorder Traversal.

The traversal pattern can be implemented using a stack-based approach. The algorithm starts at the root node and pushes it onto the stack. Then, it enters a loop where it pops a node from the stack, prints its value, and pushes its left child onto the stack if it exists. If the left child does not exist, it pushes its right child onto the stack. This continues until the stack is empty.

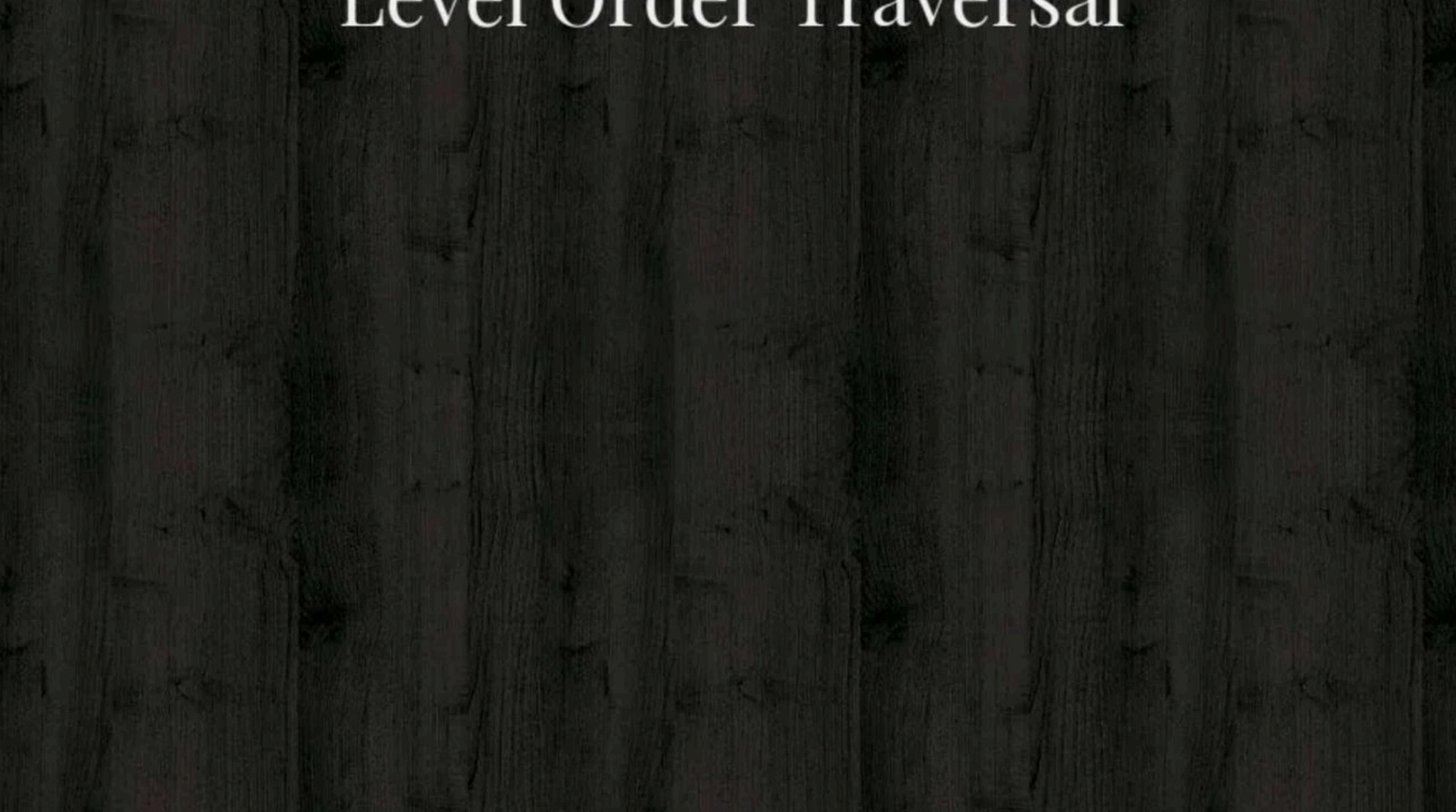
For example, consider a binary tree with root node A. The tree has left child B and right child C. Node B has left child D and right child E. Node C has left child F and right child G. The traversal order for Converse Order Traversal would be A, C, B, G, F, D, E.

Converse Order Traversal is useful for traversing trees in a specific order or for implementing certain search algorithms. It is also used in some compilers for generating code for tree structures.

In conclusion, Converse Order Traversal is a traversal pattern that visits nodes in a tree in a specific order. It starts at the root node and traverses the tree by visiting the right child of each node before its left child. This results in a traversal order that is the converse of Inorder Traversal. The traversal pattern can be implemented using a stack-based approach. It is useful for traversing trees in a specific order or for implementing certain search algorithms. It is also used in some compilers for generating code for tree structures.

# Observation from Converse Order Traversal

# Level Order Traversal



# Constructing the Tree Using Traversals

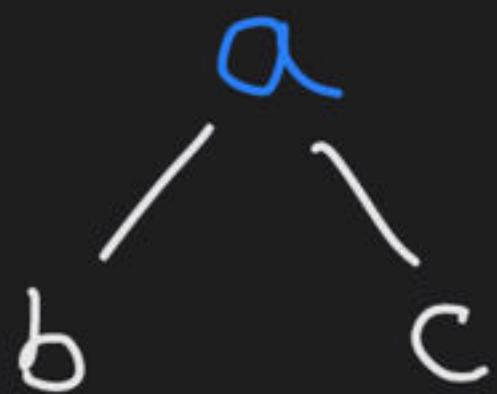
To construct unique BT from traversals

- ① At least 2 traversals are needed
- ② one should be inorder

- 
- ① Preorder or postorder => Identify root of tree (sybtree)
  - ② Inorder => Identify left & right subtree

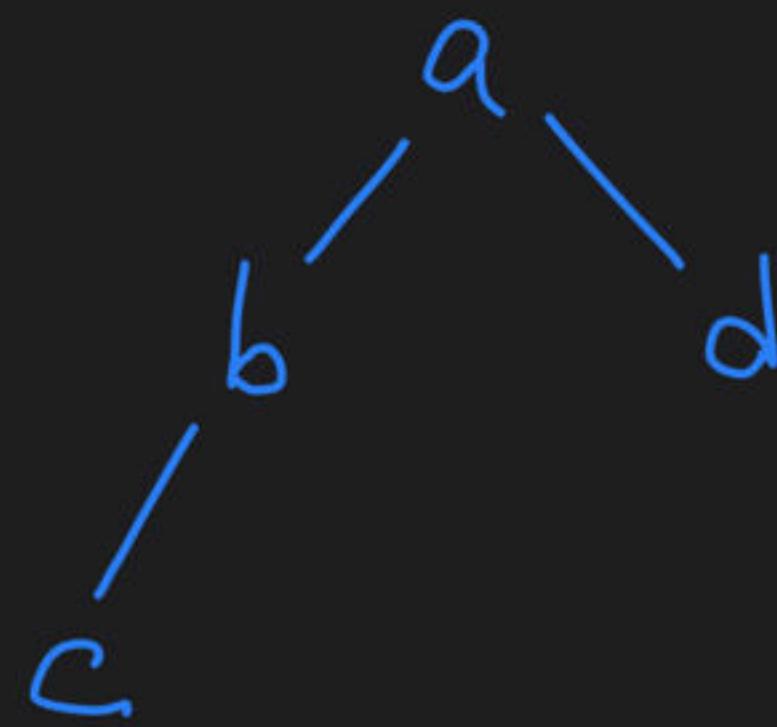
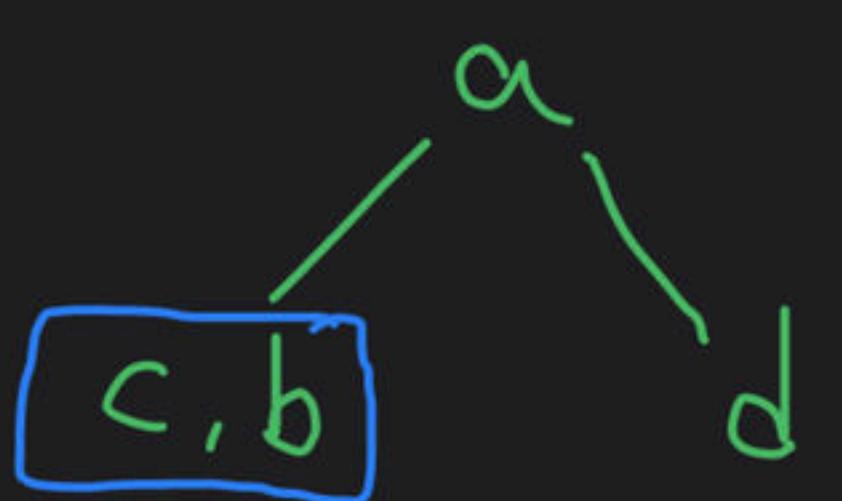
Pre:-      abc

In:-      bac



Pre :- abcd

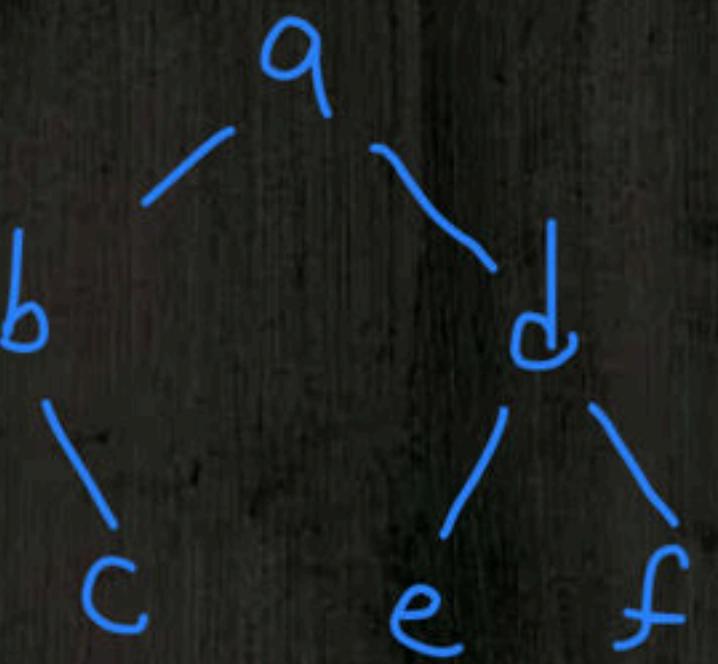
In :- cbad



# Question

PRE: abcdef

IN: bcaedf



# Question

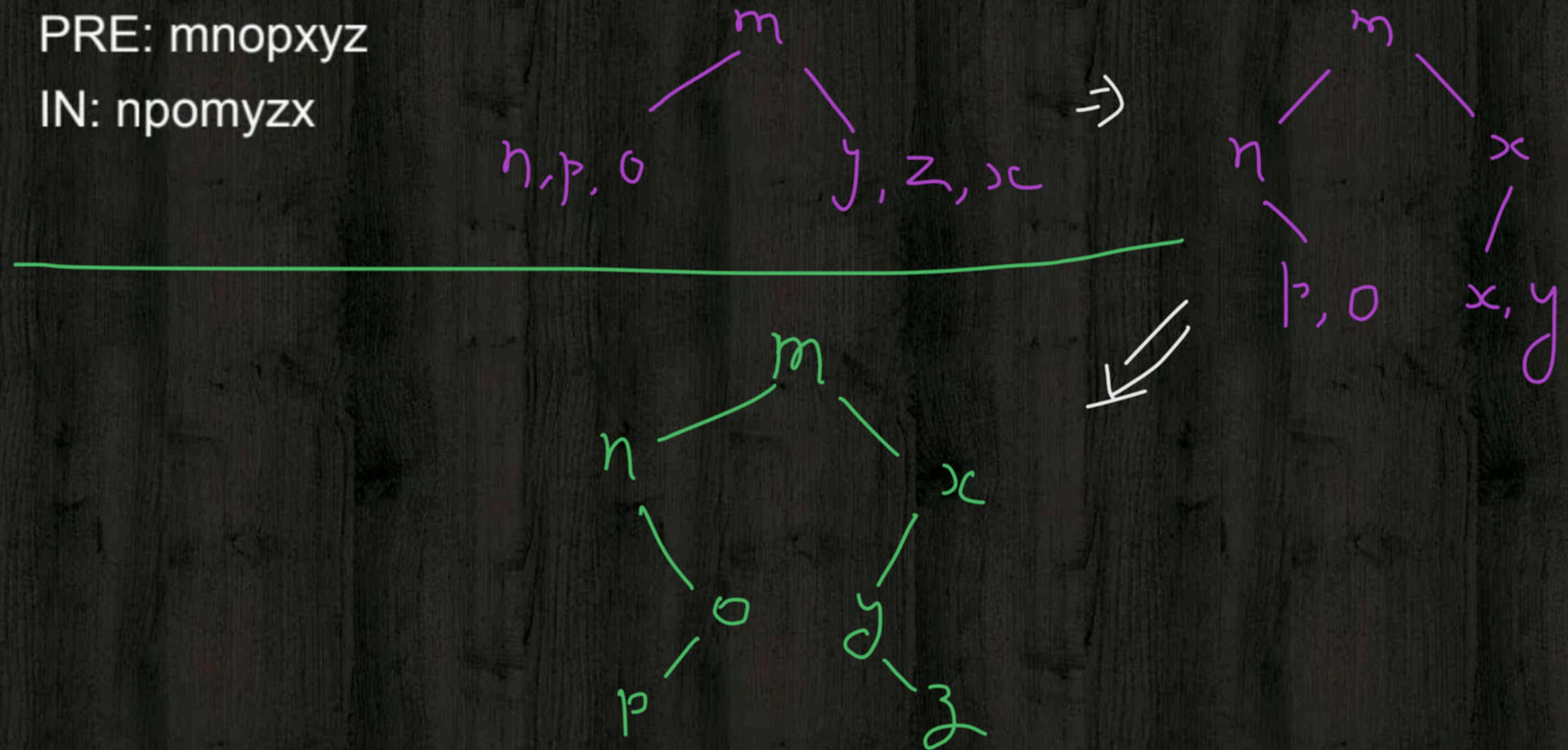
POST: efbgdca

IN: ebfadgc

# Question

PRE: mnopxyz

IN: npomyzx

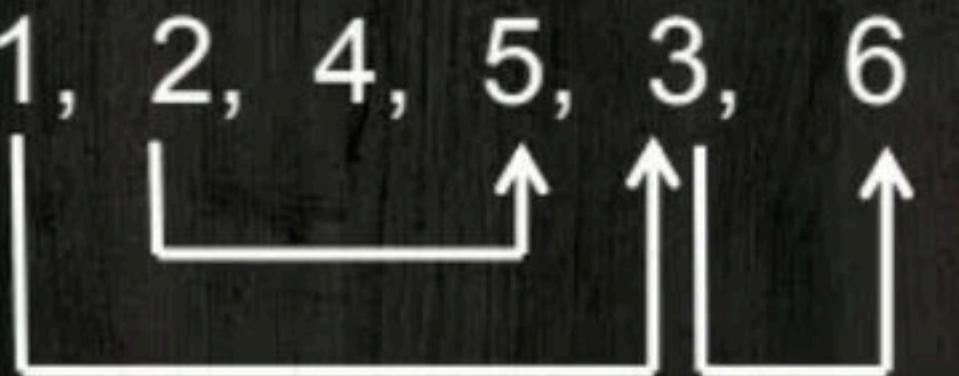


# Question

Preorder:

1, 2, 4, 5, 3, 6

RightPointer:



# Question

Postorder:

5, 6, 2, 7, 4, 3, 1

LeftPointer:



# Question

Postorder: d e b f g c a

Degree: 0 0 2 0 0 2 2

Get the Tree and Inorder/Preorder traversal

# Conversion of General Tree to Binary

General Tree to Binary Tree

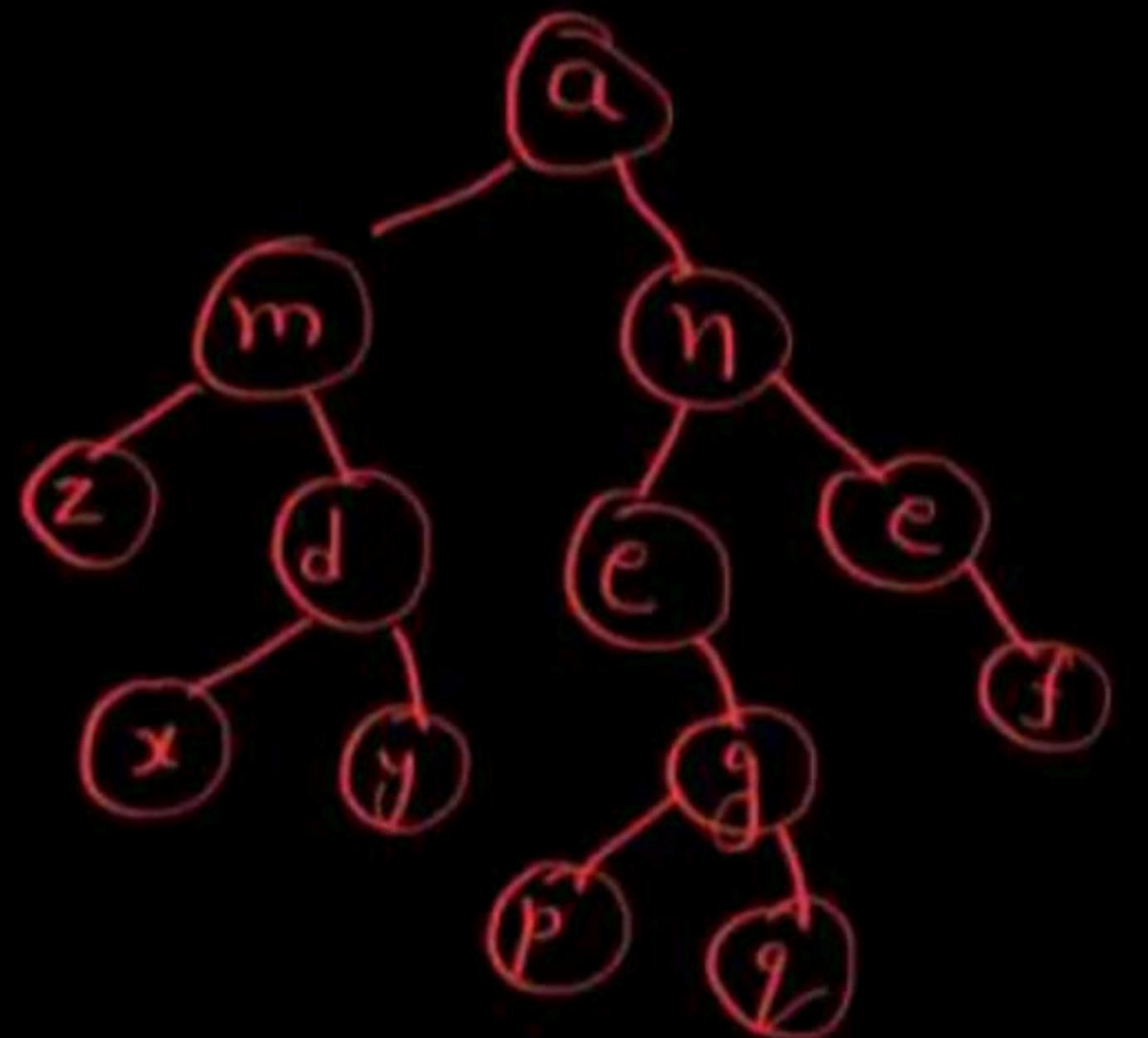


*DPP*

# Question

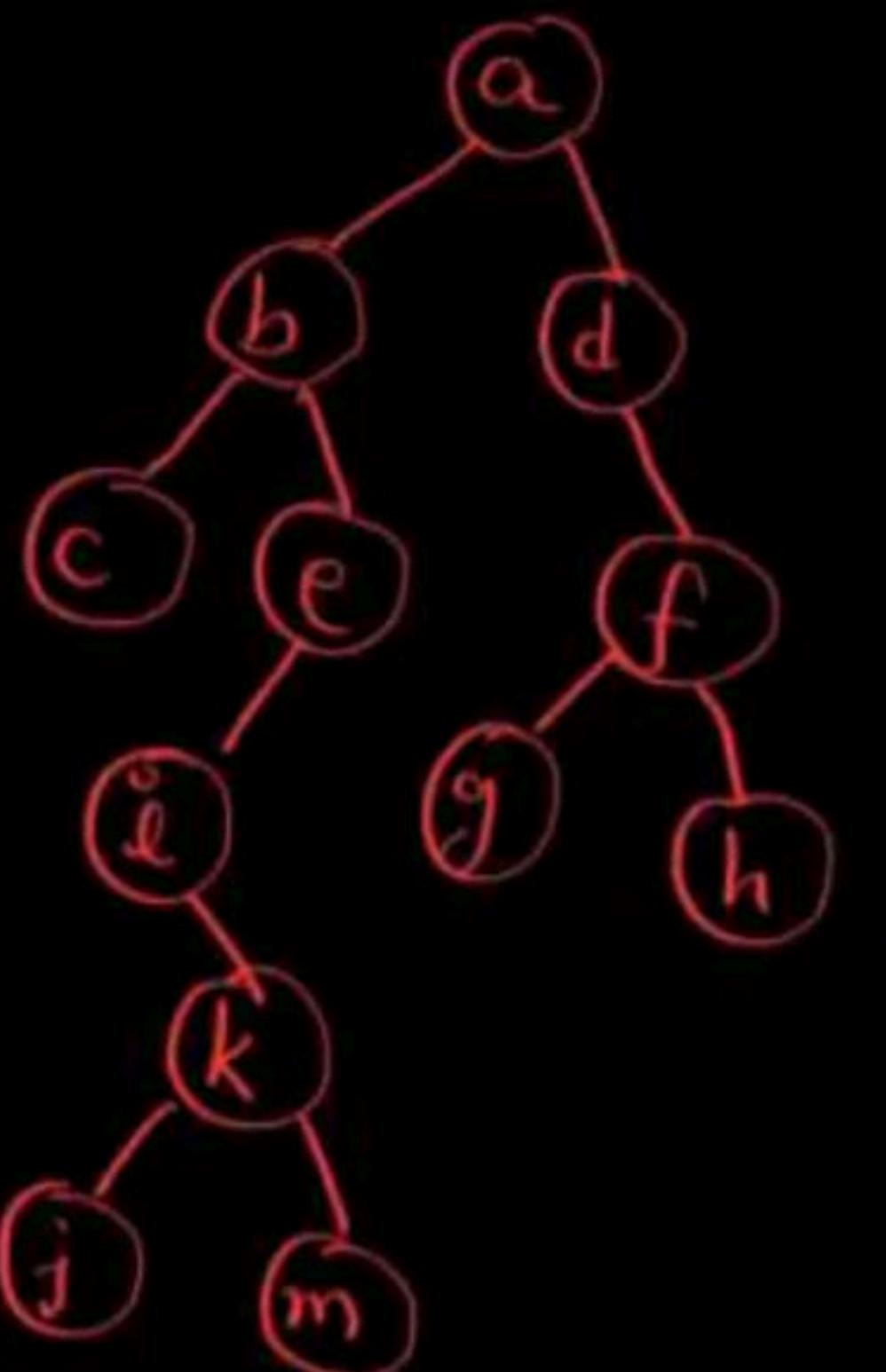
Find Traversals

1. Preorder
2. Inorder
3. Postorder



# Question

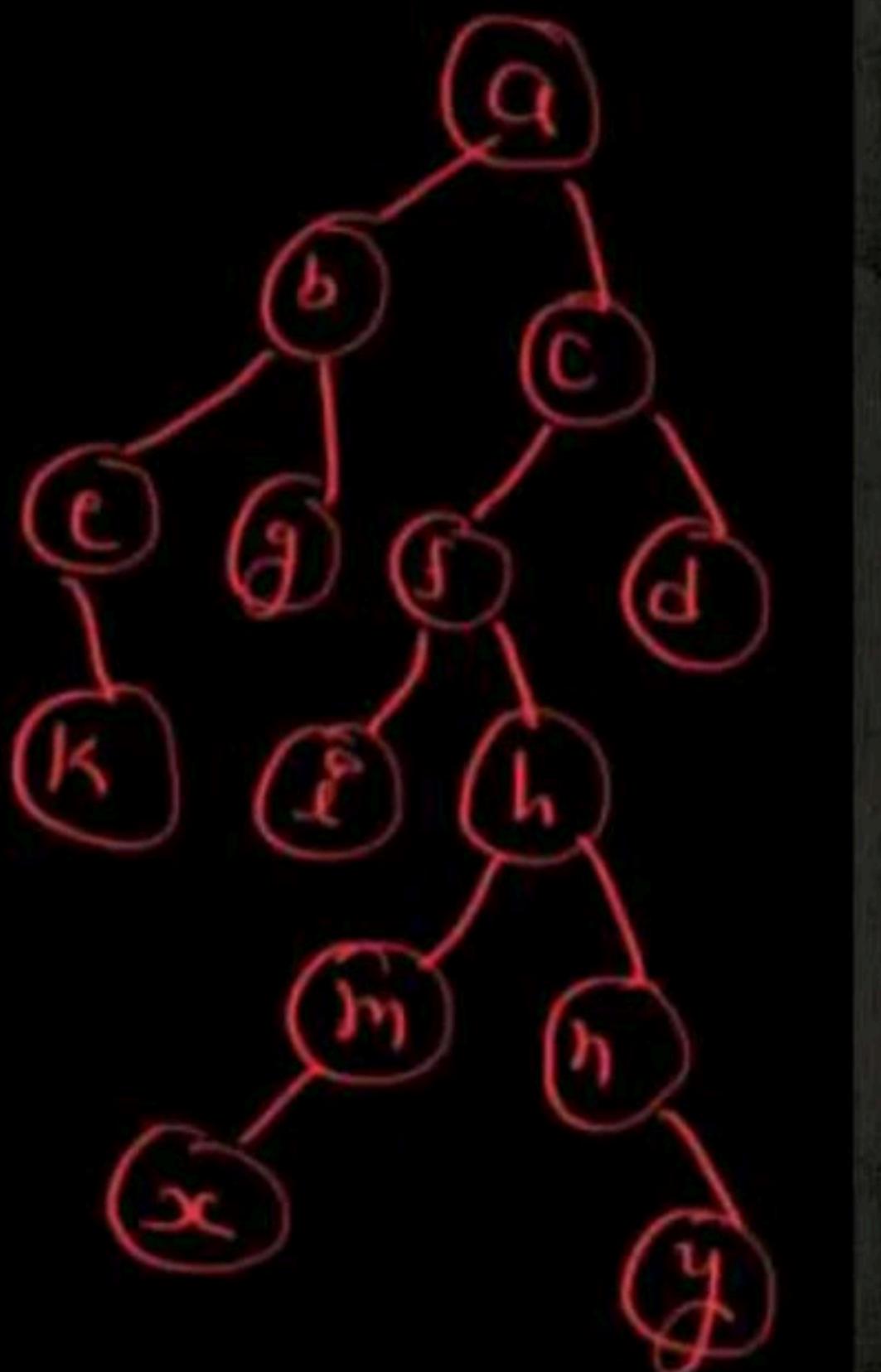
Find Traversals  
1. Preorder  
2. Inorder  
3. Postorder



# Question

Find Traversals

1. Converse Preorder
2. Converse Inorder
3. Converse Postorder



# Question

Draw the tree for:

Preorder: 1 2 3 4 5 6 7

Inorder: 2 4 3 1 6 7 5

# Question

Postorder: dxcbygfea

Inorder: bdcxayfge

# Question

Postorder: MBCAFHPYK

Inorder: MABCKYFPH

# Question

Converse Postorder: 9, 10, 8, 7, 3, 6, 5, 4, 2, 1

Converse Inorder: 7, 9, 8, 10, 3, 1, 6, 4, 5, 2

# Question

Converse Preorder: a, g, h, i, k, j, b, c, d, f, e

Inorder: c, e, d, f, b, a, h, j, i, k, g

# Question

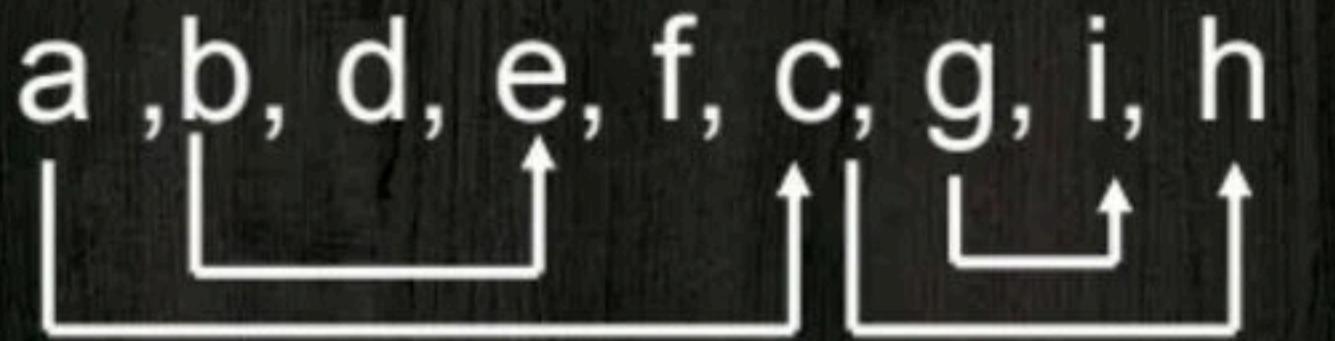
Preorder: 1, 2, 3, 4, 5, 12, 13, 6, 7, 8, 9, 11, 10

Converse Inorder: 10, 8, 11, 9, 7, 1, 2, 6, 4, 5, 13, 12, 3

# Question

Preorder:

RightPointer:



---

# Happy Learning



---

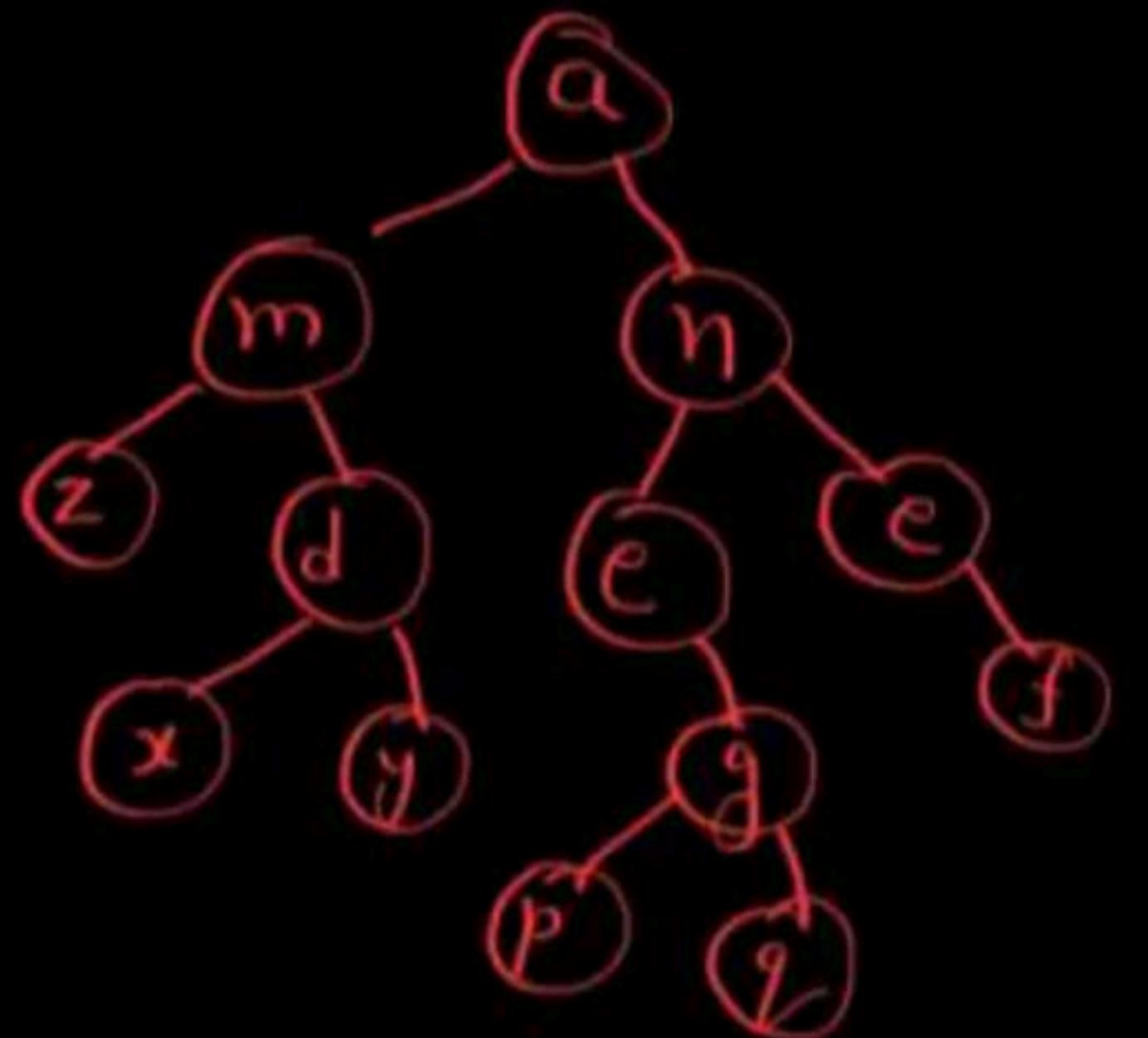


*DPP*

# Question

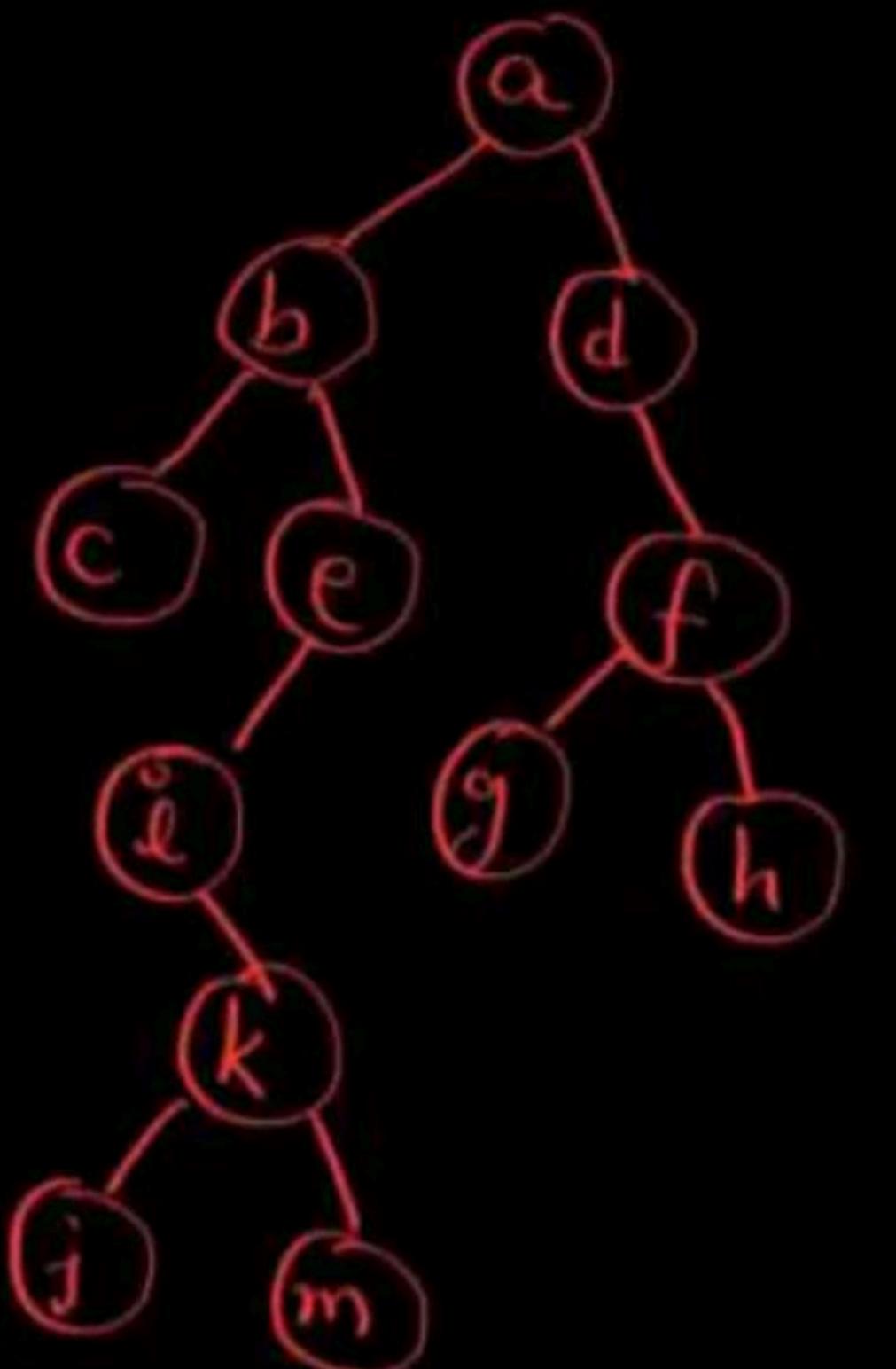
Find Traversals

1. Preorder
2. Inorder
3. Postorder



# Question

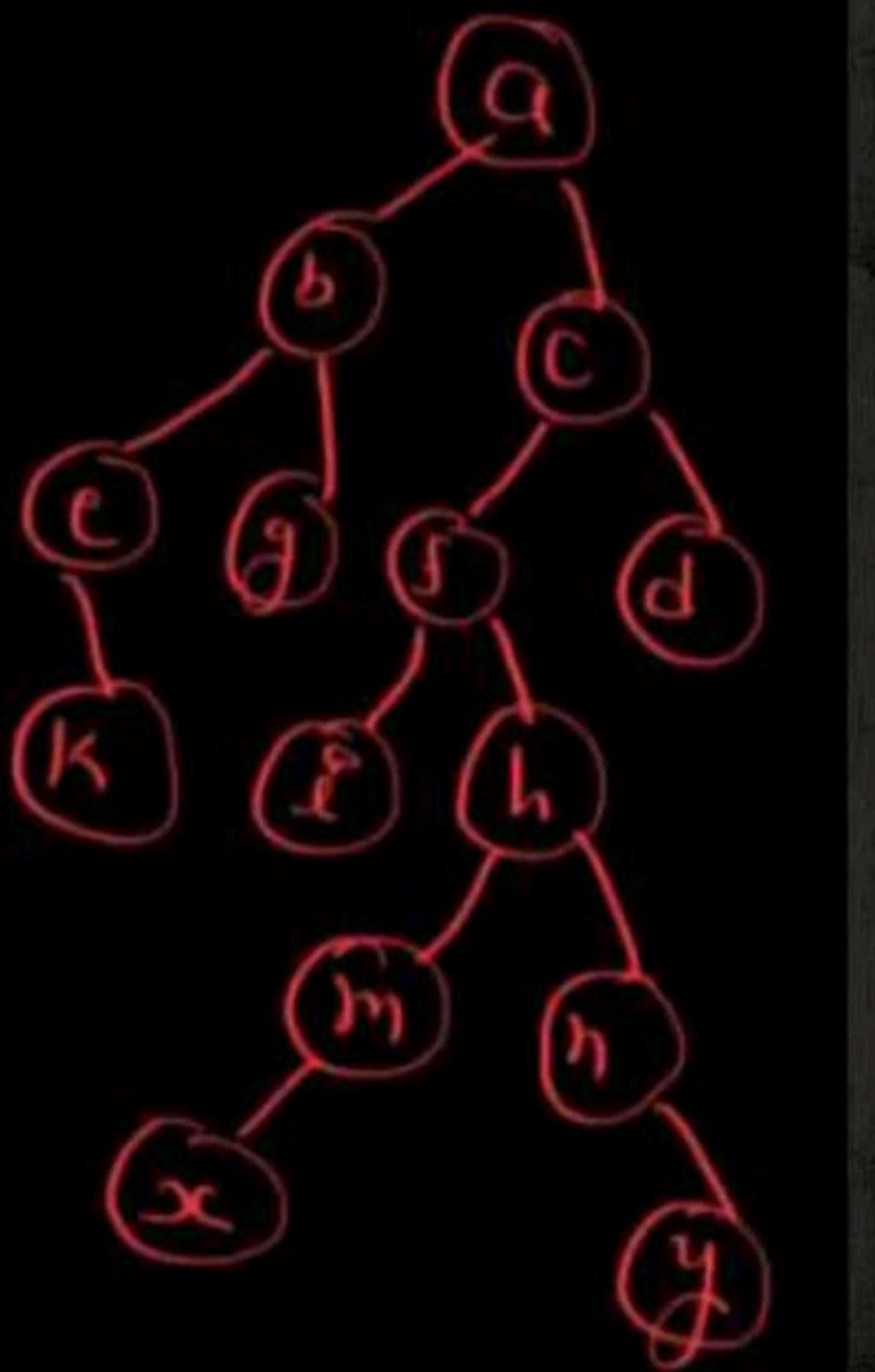
Find Traversals  
1. Preorder  
2. Inorder  
3. Postorder



# Question

Find Traversals

1. Converse Preorder
2. Converse Inorder
3. Converse Postorder



---

# Happy Learning



---