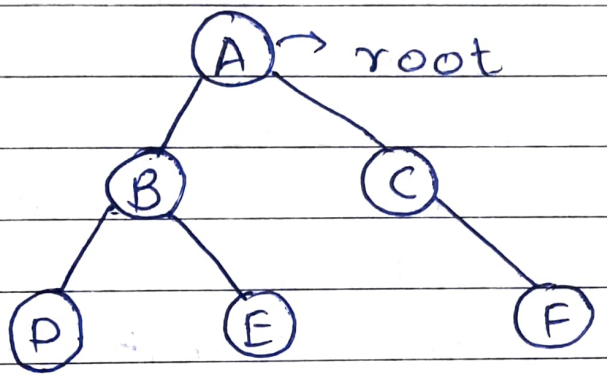
Introduction to Trees

* Trees:
- "Tree" can be defined as a collection of entities (nodes) linked together to simulate a hierarchy.



- It is a non-linear data structure.

- Some termenologies:
→ Root: The node having no parent i.e. first node in hierarchy. (A)
→ Nodes: Elements of tree containing some information and link to next node. (A, B, C, D, E, F)
→ Parent node: Immediate predecessor of any node. (A → B, C; D → None; B → D, E). Root doesn't have parent node.
→ Child node: Immediate successor of a node. (B, C → A; D, E → B; F → C)

→ Leaf node: nodes having no child also known as external nodes.
(D, E, F)

→ Non-leaf node: nodes having at-least one child. (A, B, C) i.e. all nodes other than leaf nodes i.e. internal nodes.

→ Edge: link between two nodes.

→ Path: sequence of consecutive edges from source node to destination node.

→ Ancestor: any predecessor node on the path from root to node.
(D ⇒ A, B; E ⇒ A, B; F ⇒ A, C)

→ Decendant: any successor node on the path from that node to leaf node.
(A ⇒ B, D; B, E; C, F;; C ⇒ F; B ⇒ D; E)

→ Subtree: a node inside tree containing all the desendants of that respective node.

→ Siblings: nodes having same parent are siblings.
[(B, C), (D, E)].

→ Degree: → number of children to a node is degree of that node.
→ maximum degree of any node from tree is degree of tree.

→ Depth of node: Length of path from root to that node i.e. number of edges between root to that node

→ Height of node ⇒ Number of edges in the longest path from that node to a leaf node.

→ Height of tree = height of root node.

→ Level of node: No. of edges from root to that node i.e. depth of a node = level of a node.

→ Level of tree: Level of a tree = height of a tree.

→ n - nodes ⇒ (n-1) edges.

V-50

* Binary trees and it's types:

- Each node can have at most 2 children i.e. 0, 1, 2 children.
- Maximum number of nodes possible at any level 'i' is $(2)^i$.
- Maximum number of nodes of height h $= 2^{h+1} - 1$.
- Minimum number of nodes of height h $= h + 1$.
- Maximum height given no. of nodes $n = n - 1$
- Minimum height given no. of nodes n $= [\log_2(n+1) - 1]$

Types of binary tree:

1. Full / Proper / Strict binary tree.
2. Complete binary tree.
3. Perfect binary tree.
4. Degenerate binary tree.

① Full / Proper / Strict binary tree:

→ Each node have either 0 or 2 children.
→ Every node has two children except the leaf nodes.
→ No. of leaf nodes = No. of internal nodes + 1.
→ Max. nodes = $2^{h+1} - 1$
→ Min. nodes = $2h + 1$.
→ Min. height = $\lceil \log_2(n+1) \rceil - 1$
→ Max. height = $(n-1)/2$.

② Complete binary tree:

→ All levels are completely filled (except possibly the last level) and the last level has nodes as left as possible.
→ Max. nodes = $2^{h+1} - 1$.
→ Min. nodes = 2^h.
→ Min. height = $\lceil \log_2(n+1) \rceil - 1$.
→ Max. height = $\log_2 n$.

③ Perfect binary tree:

→ All internal nodes have 2 children and all leaves are at same level.

→ Every perfect binary tree is a full binary tree and a complete binary tree but vice versa is not true.

④ Degerate binary tree:

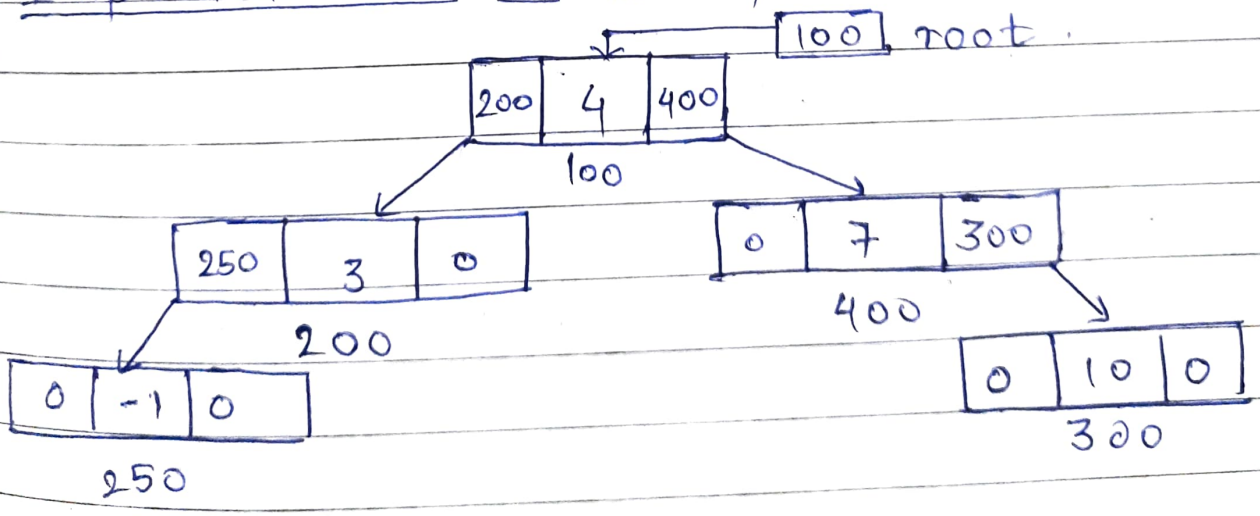
→ Each internal nodes have 1 children.

→ If internal nodes only contain left child then it's a "left skewed binary tree".

→ If internal nodes only contain right child then it's a "right skewed binary tree".

V-51.

* Implementation of binary tree:



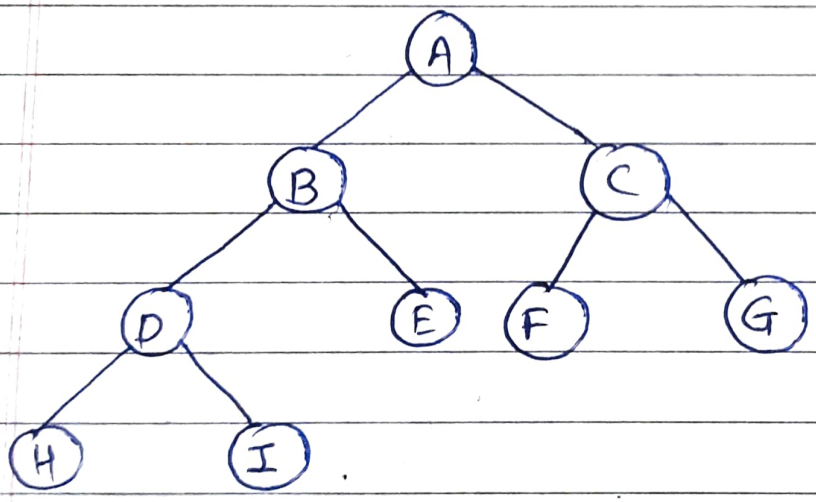
Code:

```
struct node
{ int data;
  struct node *left *right; };
```

remaining code in laptop;
check it; (Watch V-51 from jenny's playlist for better understanding)

V-52

* Array representation of binary tree:



- Case I:

A	B	C	D	E	F	G	H	I
0	1	2	3	4	5	6	7	8

→ If a node is at i^{th} index:
left child would be at: $2i+1$
right child would be at: $2i+2$.
parent would be at: $\left[\frac{(i-1)}{2}\right]$

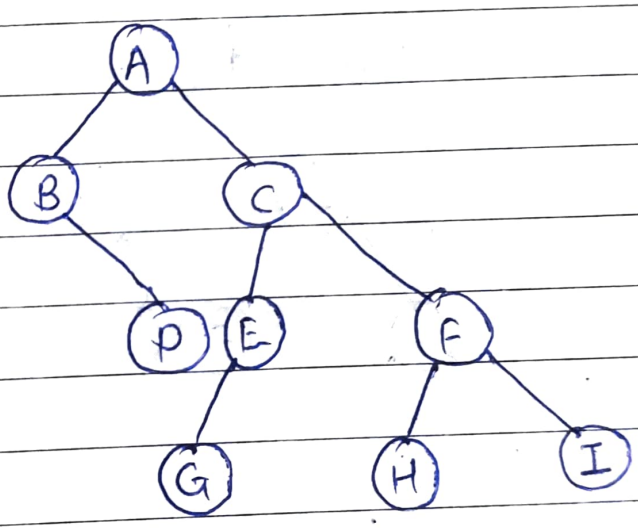
• Case II :

A	B	C	D	E	F	G	H	I
1	2	3	4	5	6	7	8	9

→ If node is at i^{th} index.
left child would be = $2i$.
right child would be = $2i+1$
parent would be = $\left[\frac{i}{2}\right]$

V-53

* Binary Tree traversal :



• Inorder : Left Root Right
• Preorder : Root Left Right.
• Postorder : Left Right Root.

• Inorder : B D A G E C H F I.
• Preorder : A B D C E G F H I.
• Postorder : D B G E H I F C A.

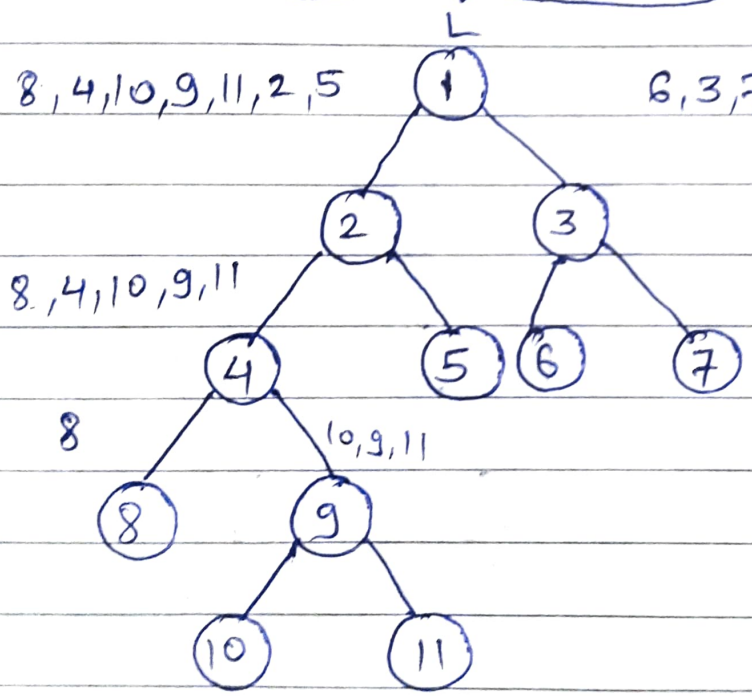
V-54

Implementation of preorder, postorder, inorder in C.
(Refer video-54 from jenni's playlist.

V-55

* Construct a binary tree from preorder and inorder:

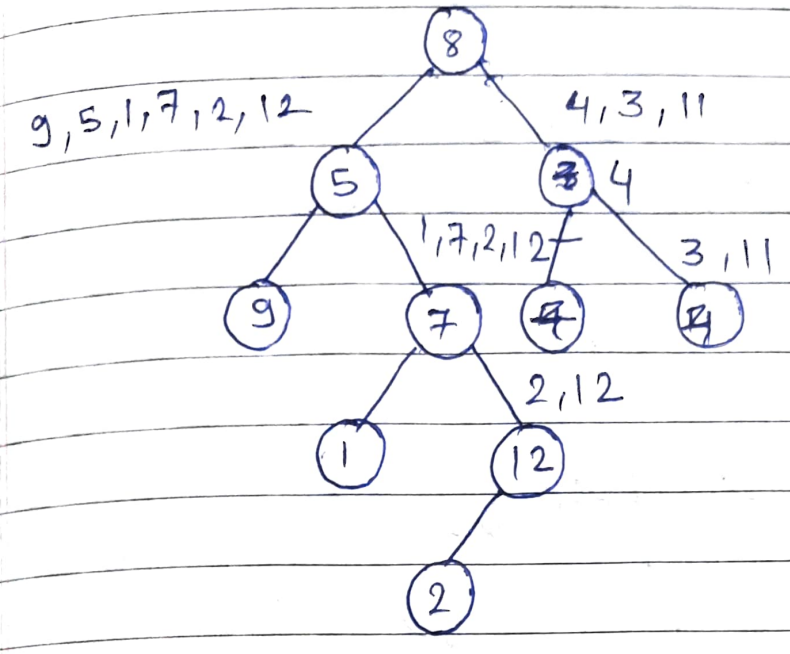
- Preorder: 1, 2, 4, 8, 9, 10, 11, 5, 3, 6, 7 (Root L R)
- Inorder: 8, 4, 10, 9, 11, 2, 5, 1, 6, 3, 7 (L Root R)



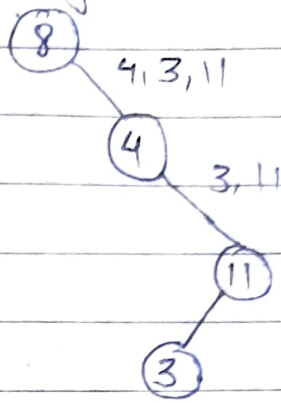
V-56

* Construct a binary tree from post-order and inorder:

- Postorder: 9, 1, 2, 12, 7, 5, 3, 11, 4, 8 (L R Root)
- Inorder: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11 (L Root R)



Right subtree :



V-57

* Construct a binary tree from preorder and postorder :

. There can't be just a unique binary tree i.e. from preorder and postorder only we can't build a unique binary tree but a unique full binary tree can be built; i.e. if a tree is full binary tree it can be built only given it's preorder and postorder.

V-55

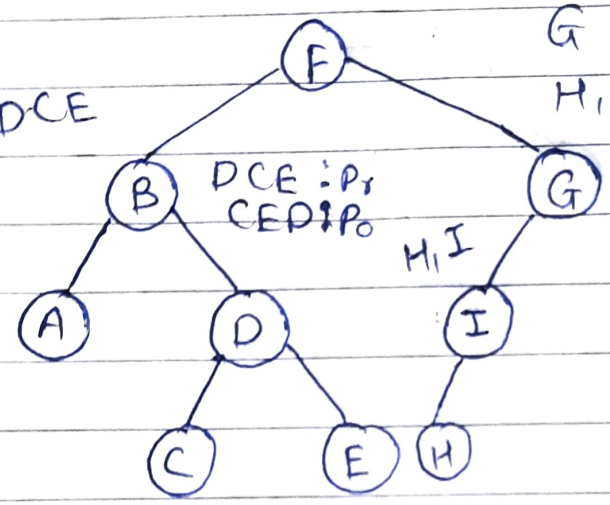
V-56 Concept

. If either of preorder or postorder is given along with inorder, we can track roots with the help of preorder or postorder and can find elements position i.e. whether they are in the right subtree or left subtree of respective element.

But here situation is different.

Ex.1)

- Preorder: F B A D C E G I H (Root L R)
- Postorder: A C E D B H I G F (L R Root)



BADCE

DCE: Pr
CED: Po

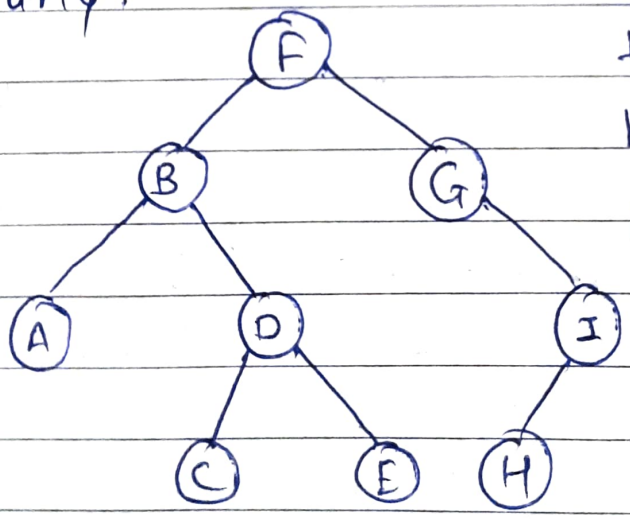
HI

G I H : Pre
H, I, G : Post

Pre: BADCE
Post: ~~BDF~~
ACEDB

Pre: I H
Post: H I

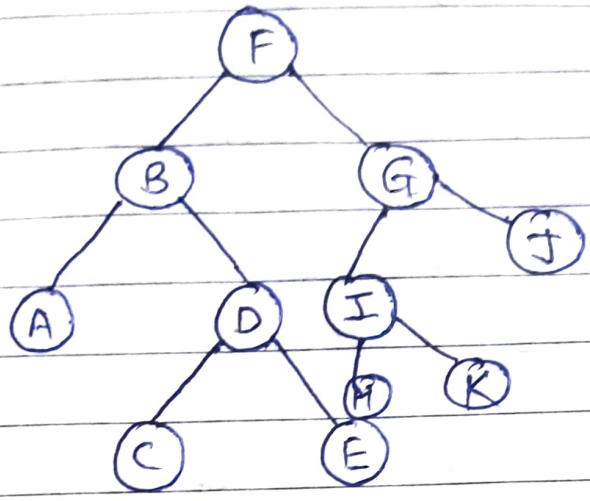
Similarly:



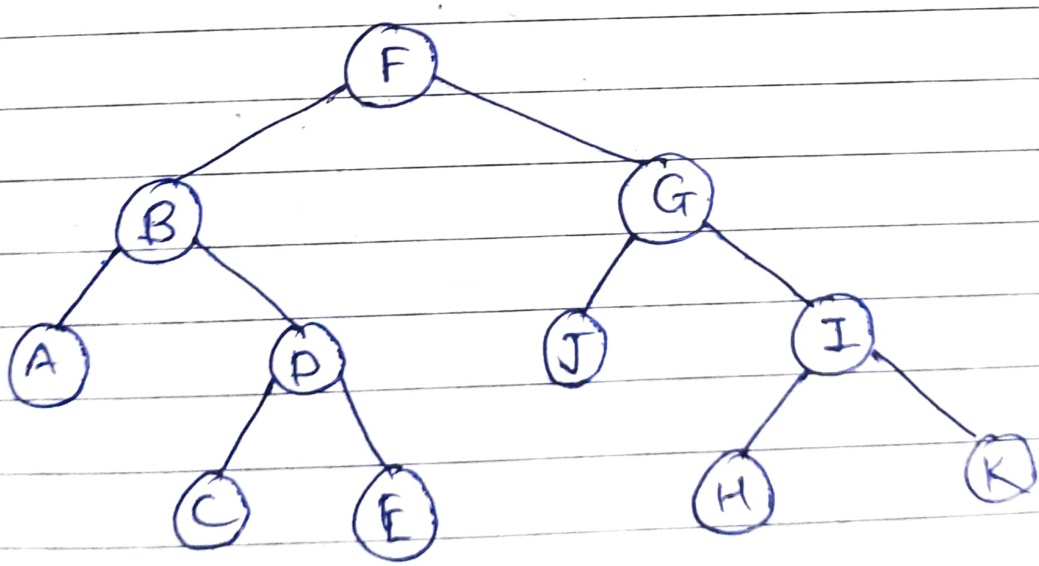
This binary tree also has same preorder and postorder.

- Here, we used recursive calls of the same function we can say we find preorder and postorders of subtrees.

2. Preorder : F B A D C E G I H K J (Root L R)
Postorder : A C E D B H K I J G F (L R Root).



3. Preorder : F B A D C E G J I H K
Postorder : A C E D B J H K I G F



For revision of trick refer V-57 from jenny's playlist.