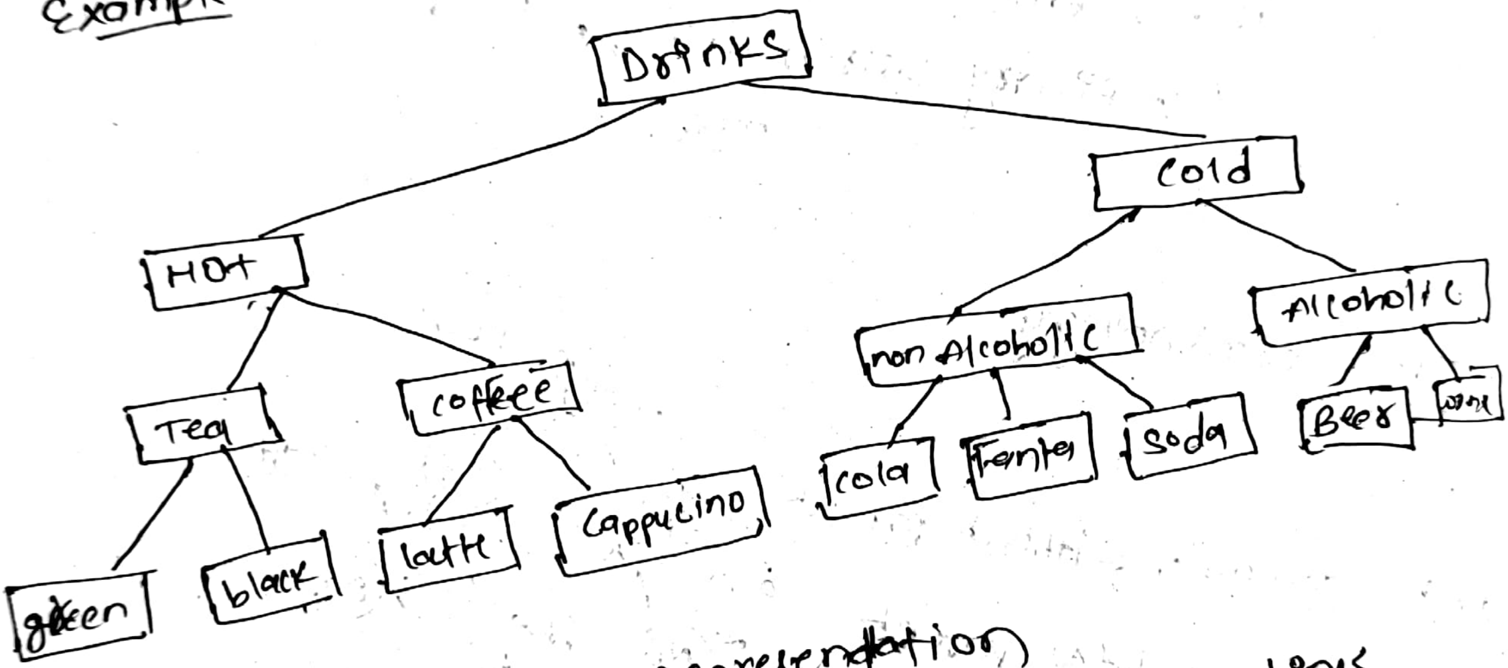
5. TREE

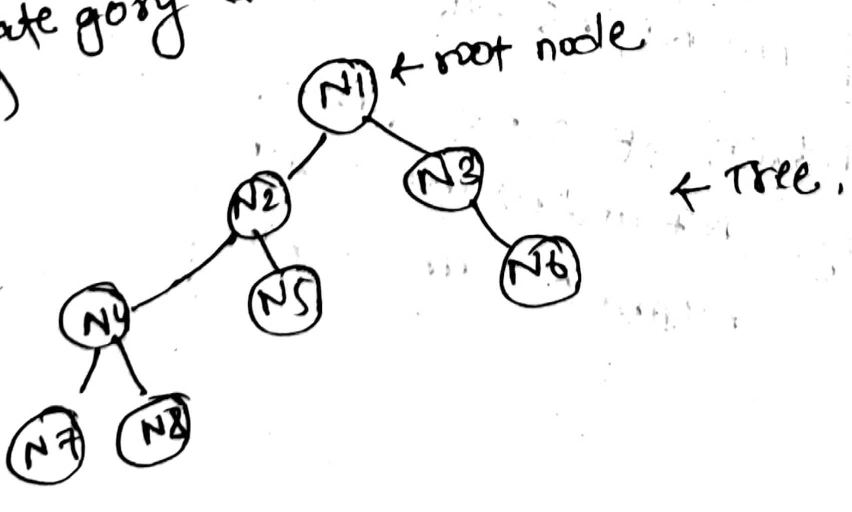
A tree is a non linear data structure with hierarchical relationships between its elements without having any cycle, it is basically reversed from a real life tree.

Example



- hierarchical form representation
- Each node has two components: data and a link to its sub category.

- Base category and sub categories under it. (root)

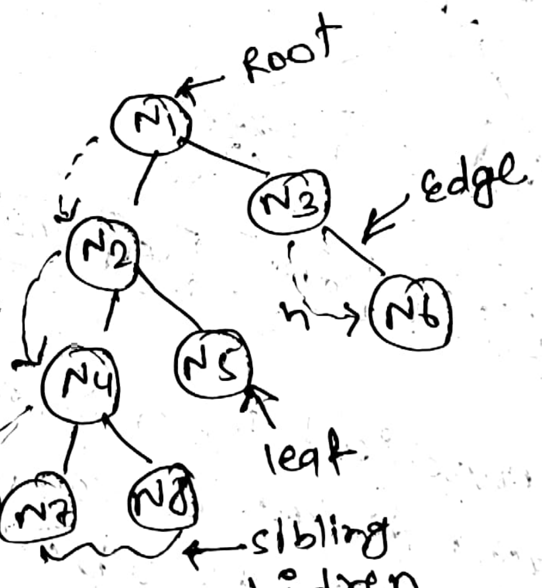


Why Tree?

- Quicker and easier access to data
- Store hierarchical data, like folder structure, organization structure, XML/HTML data.

- There are many different types of data structures which performs better in various situations :- Binary Search tree, AVL, Red Black Tree, Trie

Tree Terminology



(1) **Root** : top node without parent

(2) **Edge** : a link btw parent and a child

(3) **leaf** : a node which doesn't have children
Eg - N7, N8, N5, N6

(4) **Sibling** : children of same parent:
Eg :- N7 and N8 are sibling
N4 and N5 are sibling

(5) Ancestor : parent, grand parent, great grand parent of a node.

eg:- Ancestor of N7 → N4, N2, N1
" " N5 → N2, N1

(6) Depth of node : a length of path from root to node
eg:- Depth of N4 = 2

(7) Height of node : a length of the path from the node to the deepest node
eg:- Height of N3 = 1

(8) Depth of tree : depth of root node
eg:- Depth of tree = 0.

(9) Height of tree : height of root node
eg:- Height of tree = 3

* methods for creating Basic Tree is in code Section in PC.

Binary Tree

Binary trees are the data structures in which each node has at most two children, often refered to as the left and right children.

Binary tree is a family of data structure (BST, Heap tree, AVL, red black trees, syntax tree).

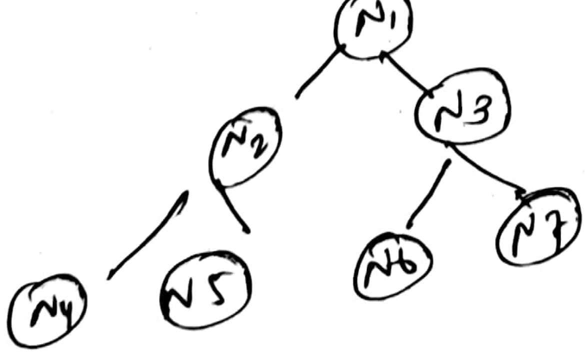
why need?

- Binary trees are a prerequisite for more advanced trees like BST, AVL, Red Black trees
- Huffman coding problem, heap priority problem and expression parsing problems can be solved efficiently using binary trees.

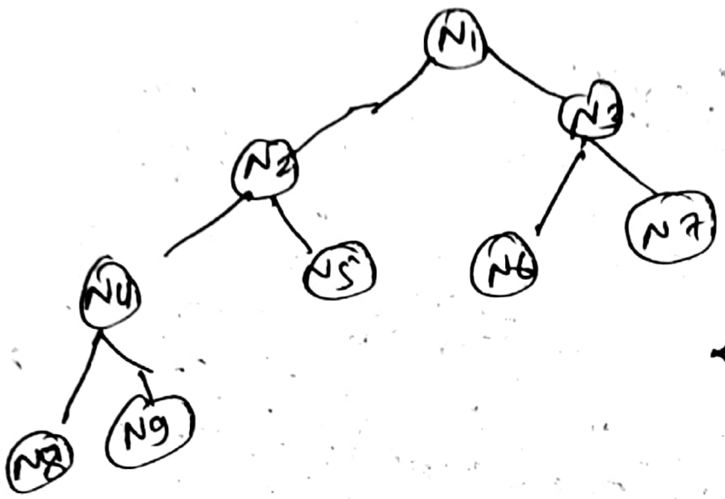
Types of Binary Tree

① Full Binary tree: Either two children or none

② perfect Binary tree: All non leaf nodes have two children and same depth

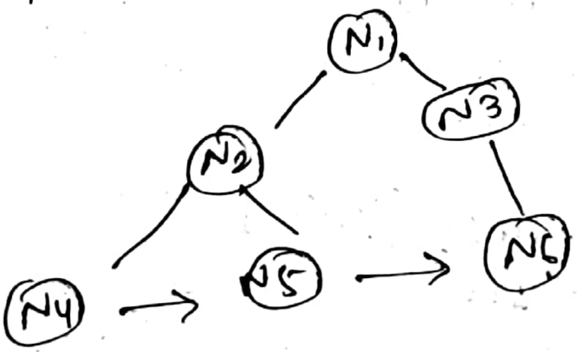


③ complete Binary Tree : have two children to full a level, except the last level



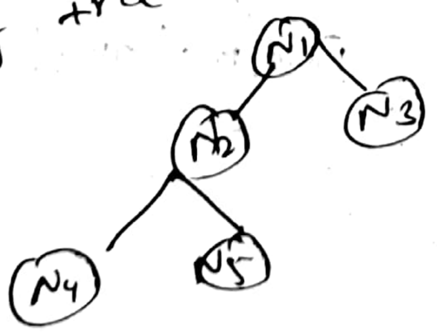
← this level is not filled.

④ Balance Binary tree : All leaf node is at same distance from the root node.



→ same distance from root node.

① full Binary tree

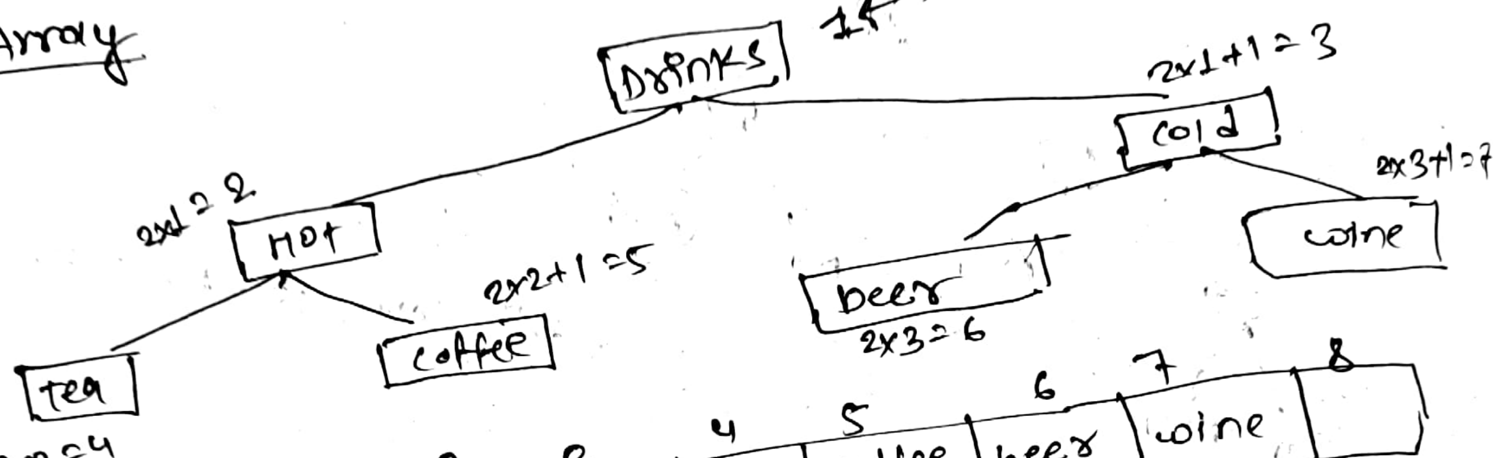


Not possible to have one children

Binary Tree

Representation

Array

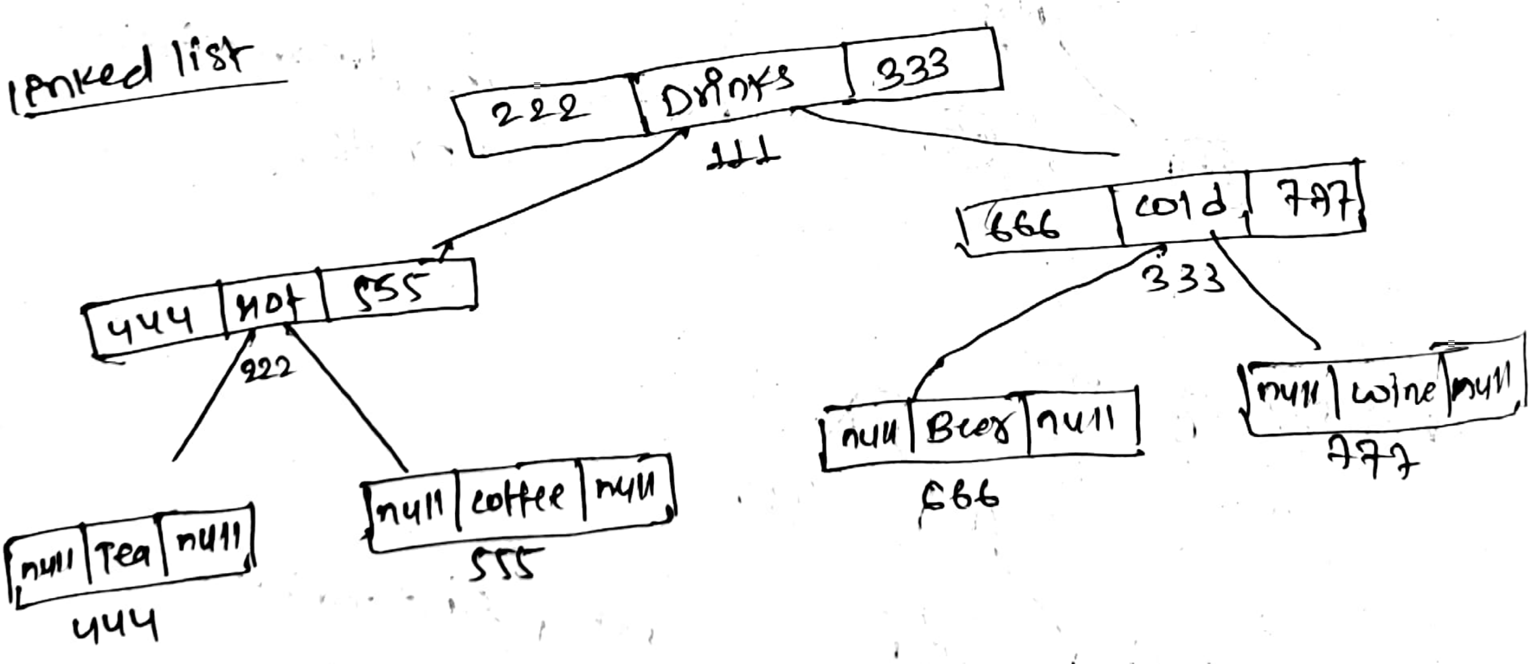


0	1	2	3	4	5	6	7	8
x	Drinks	Hot	cold	Tea	coffee	beer	wine	

left child = cell [2x] , x is index of parent
Right child = cell [2x+1]

* root node index is always 1

linked list



Binary Tree using linked list

① Creation of Tree

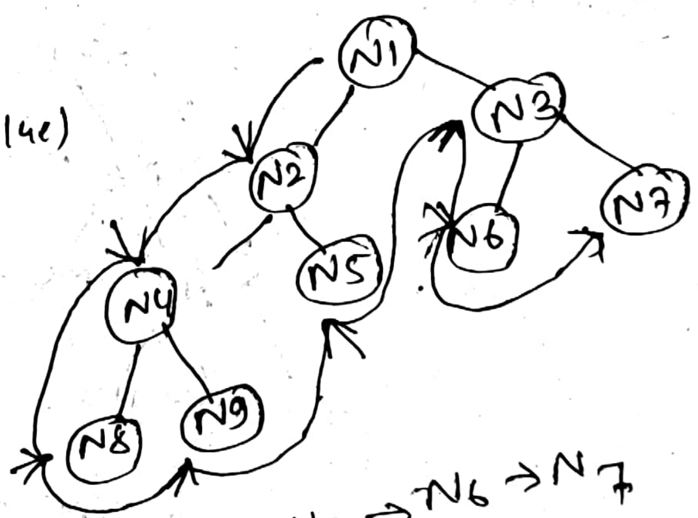
method in PC!

② Traversal of Binary Tree

Depth first search
- preorder traversal
- Inorder traversal
- postorder traversal

Breadth first search
- level order traversal

* preOrder traversal



Root Node (sout node.value)

↓

left subtree

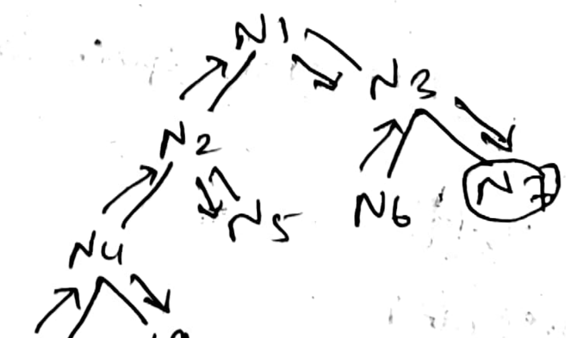
↓

Right subtree

⇒ N1 → N2 → N4 → N8 → N9 → N5 → N3 → N6 → N7

method in PC!

* InOrder traversal



left subtree

↓

Root Node (sout node.value)

↓

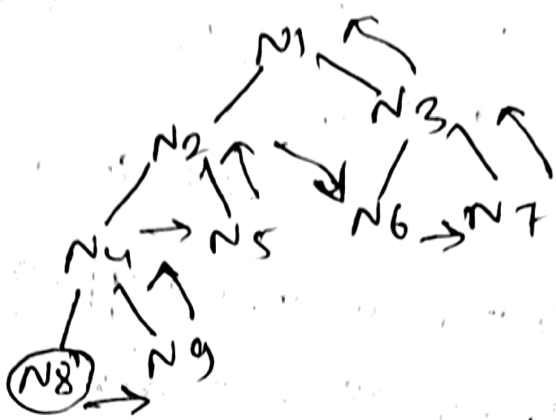
Right subtree

⇒ N8 → N4 → N9 → N2 → N5 → N1 → N6 → N3 → N7

method in PC!

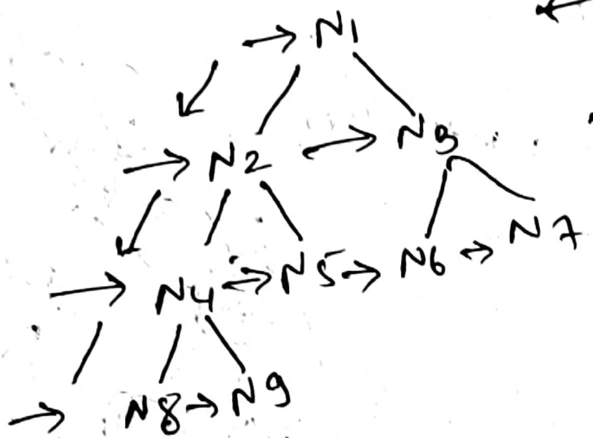
* Post Traversal

(left subtree)
↓
(right subtree)
↓
(Root node) (sout) node.value



⇒ N8 → N9 → N4 → N5 → N2 → N6 → N7 → N3 → N1.

* level order Traversal (VV I)



← level 1
← level 2
← level 3
← level 4

⇒ N1 → N2 → N3 → N4 → N5 → N6 → N7 → N8 → N9

method in pc!

Time and space complexity of different Traversals

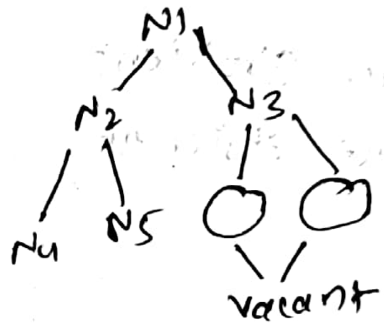
type	Time comp.	Space complexity
	O(N)	O(N)
preOrder	O(N)	O(N)
Inorder	O(N)	O(N)
postOrder	O(N)	O(N)
levelorder		

③ Searching In Binary Tree

use level order traversal for searching
↳ this uses Queue, other used stack. so its good

method in pc!
TC – O(N) , SC – O(N)

④ Insert a Node in Binary Tree

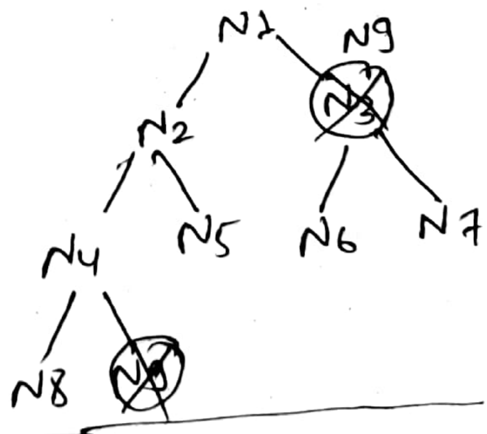


- A root node is null
- The tree exists and we have to look for a first vacant place

- using level order traversal.

method in pc!
TC – O(N)   SP – O(N)

⑤ Delete a Node in Binary Tree



→ level order traversal
→ to be deleted eg:- N3
step 1: find the Node
step 2: Find deepest Node
3: Set deepest Node's value to current node
4: Delete Deepest Node

method in pc!
TC – O(N)
SC – O(N)

⑥ Delete entire Binary tree

rootNode = Null;

TC - O(1)

SC - O(1)

	Linkedlist	Array
	✓	X
Space efficient	✓	✓
Time efficient		

Binary Tree using Array

① Create BT by Array

TC - O(1)   SC - O(N)

method in pc!

② Insert a Node in BT

- The Binary tree is full
- we have to look for a first vacant place

TC = O(1)   SC = O(1)

③ Traversal of BT

In context of notemaking, it is same as Binary Tree using linked list, logic is same, implementation is diff.

	TC	SC
preorder	O(N)	O(N)
inorder	O(N)	O(N)
postorder	O(N)	O(N)
levelorder	O(N)	O(1)

④ searching

TC - O(N)   , SC - O(1)

⑤ Delete Node

TC - O(N)   SC - O(1)

⑥ Delete Entire Binary Tree

arr = null;

TC = O(1)   SC = O(1)