FAKULTAS
TEKNOLOGI
INFORMASI

Universitas Advent Indonesia

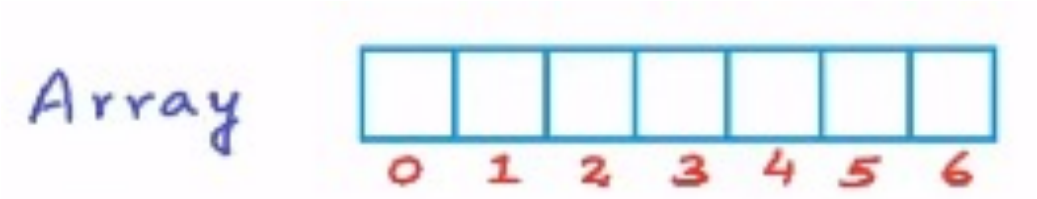
Lecture 13 – Tree Data Structure

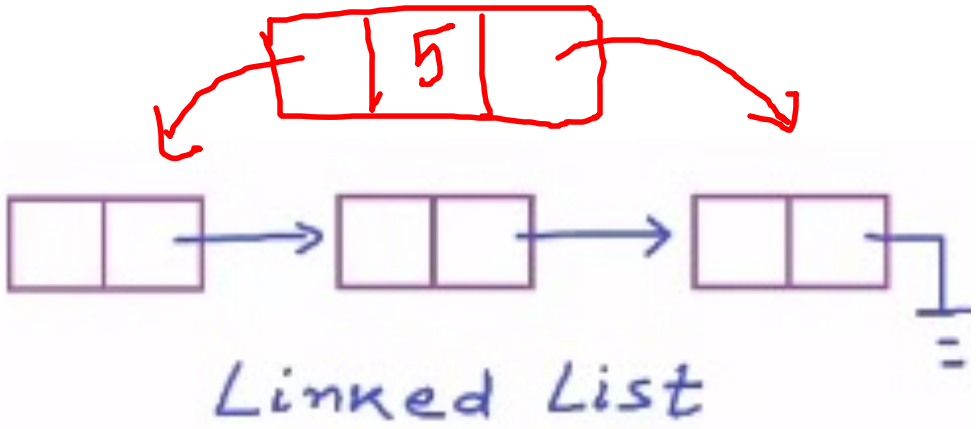
Friday, December 1, 2023

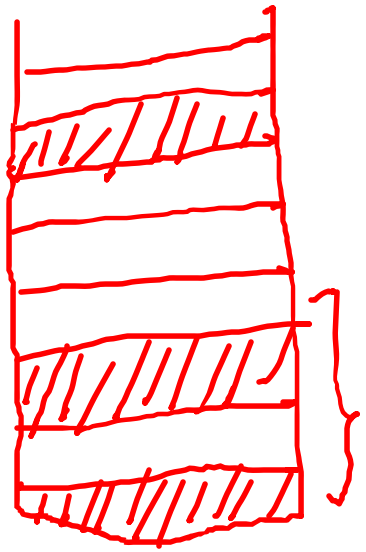
Data Structure

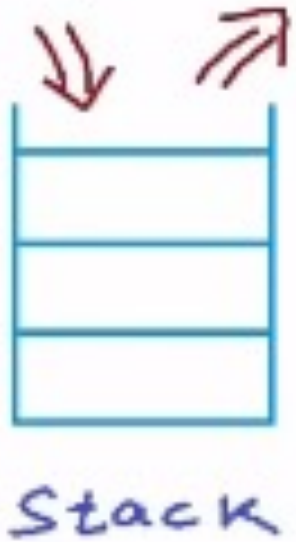
fti.unai.edu

Introduction









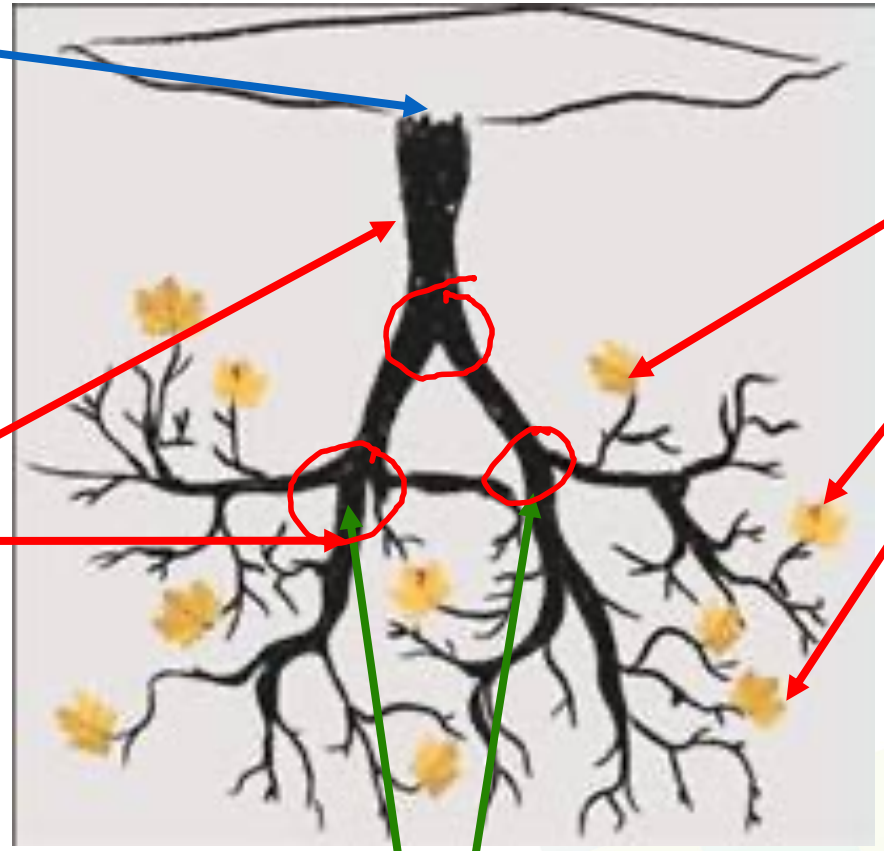
FAKULTAS TEKNOLOGI INFORMASI
Universitas Advent Indonesia

Introduction

- Data structure → ways to store and organize data in computers.
- How should I decide which data structure to use?
  - Depends upon a number of factors
    - What needs to be stored?
    - Cost of operations
    - Memory usage
    - Ease of implementation

Tree in Real World

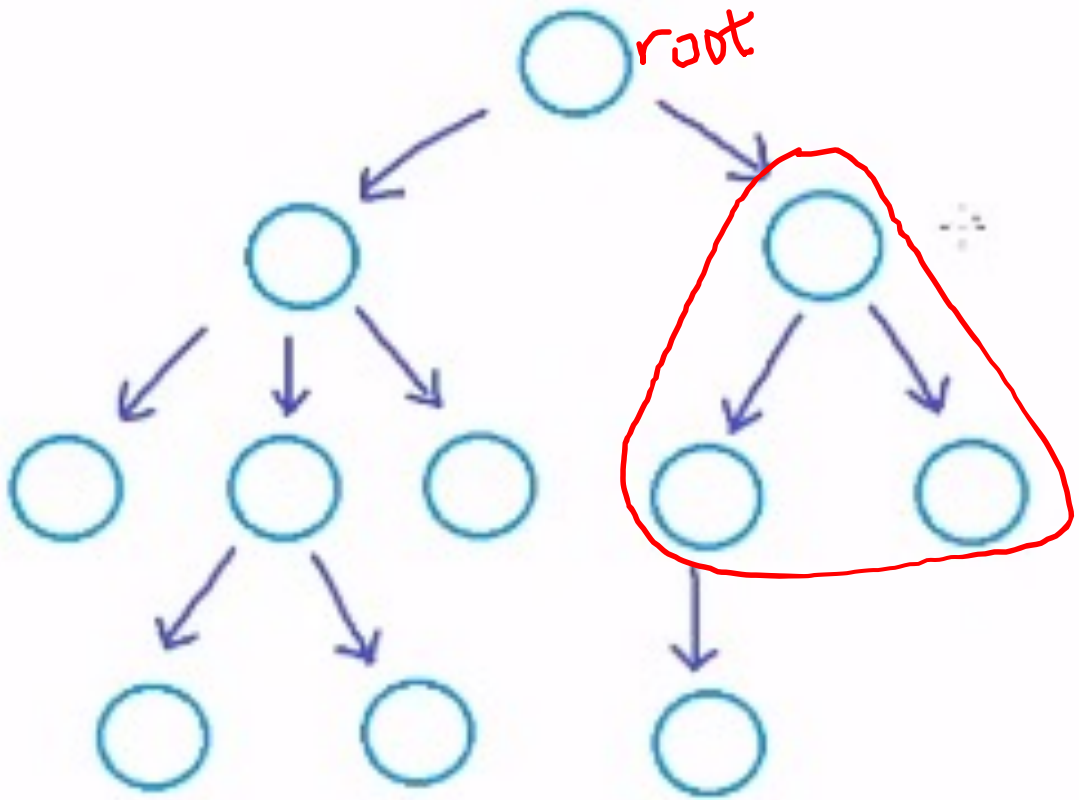
Computer Scientist's View



Introduction to Tree Data Structure

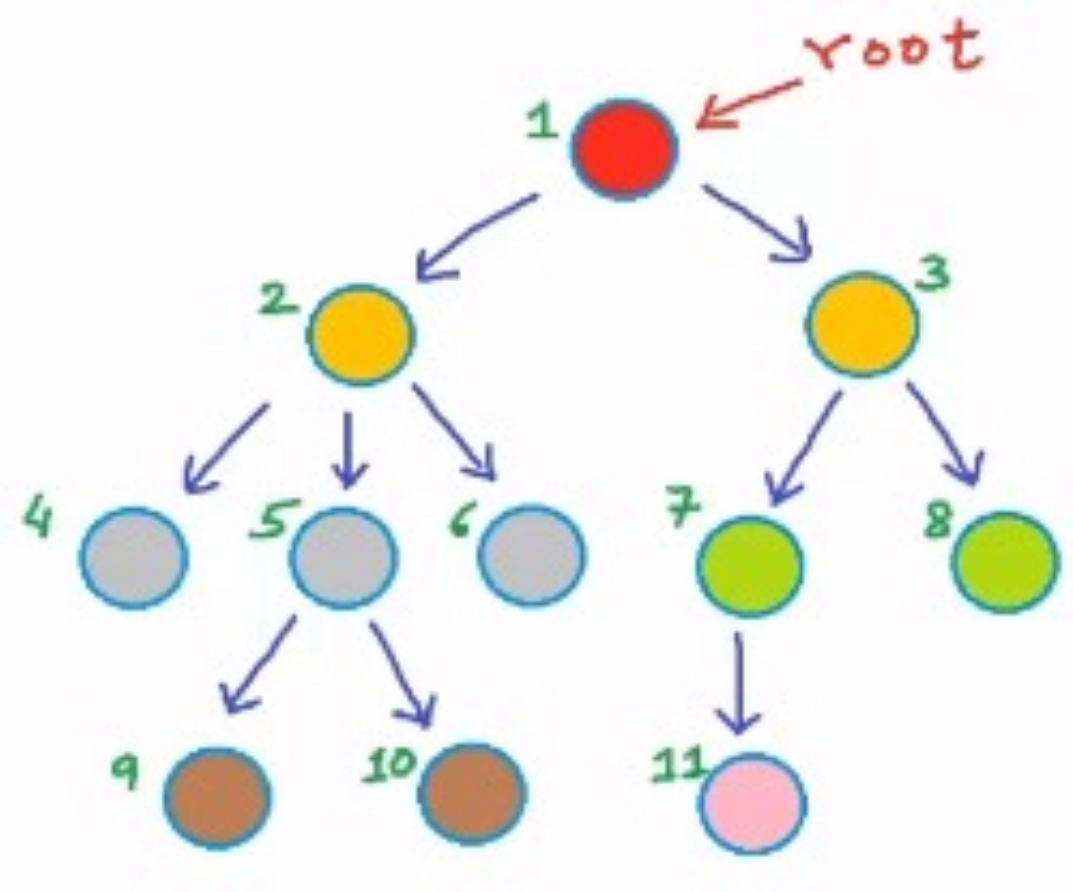
- Tree data structure can be defined as a collection of entities called **nodes** linked together to simulate hierarchy.
- Tree is a non-linear data structure. It is a hierarchical structure.
- The top most node is called root.
- Each node in a tree will contain some data. This can be data of any type, and may contain link or reference to some other nodes that can be called its children.

Tree Terminologies



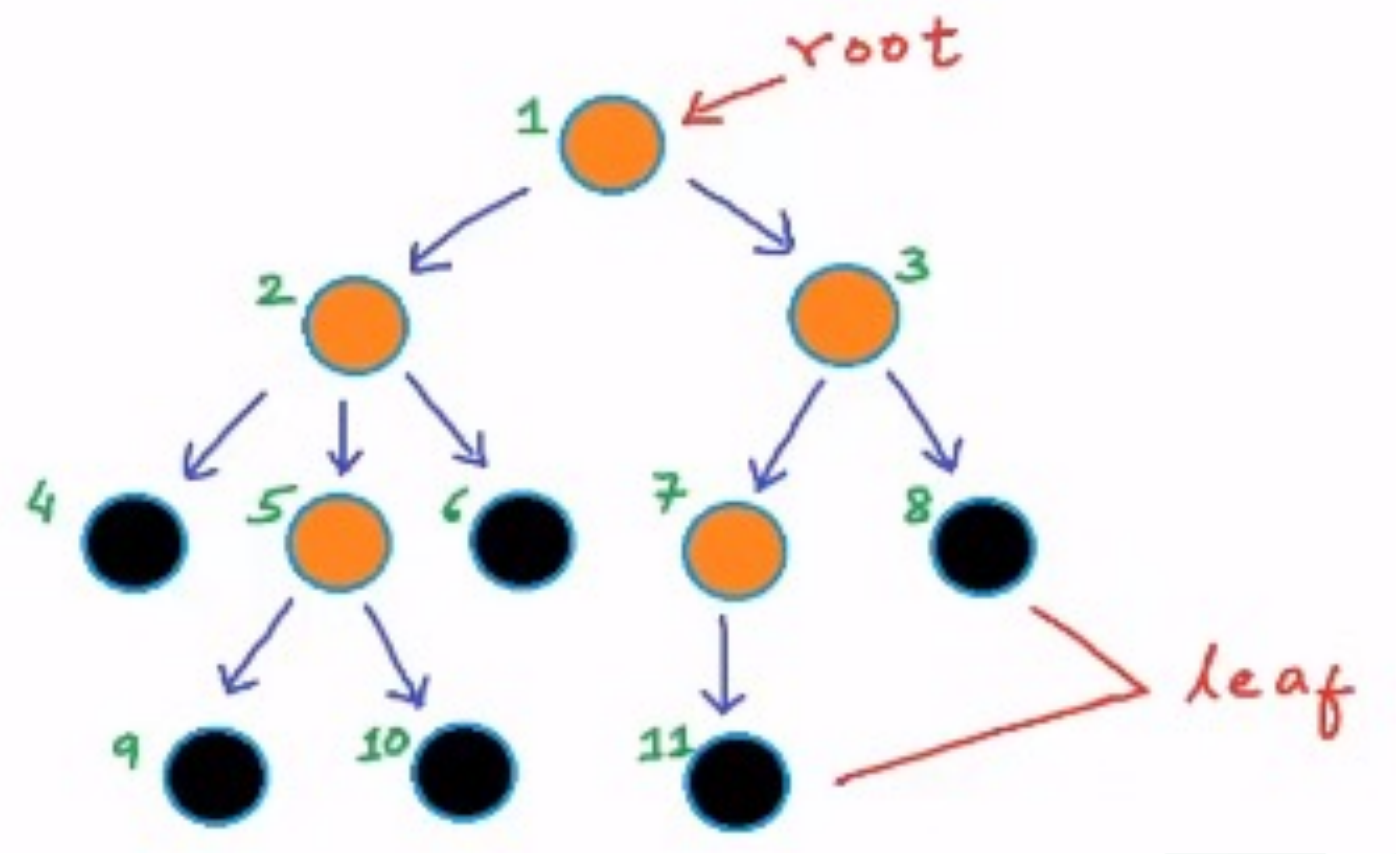
- Tree is an efficient way of storing and organizing data that is naturally hierarchical.

Tree Terminologies



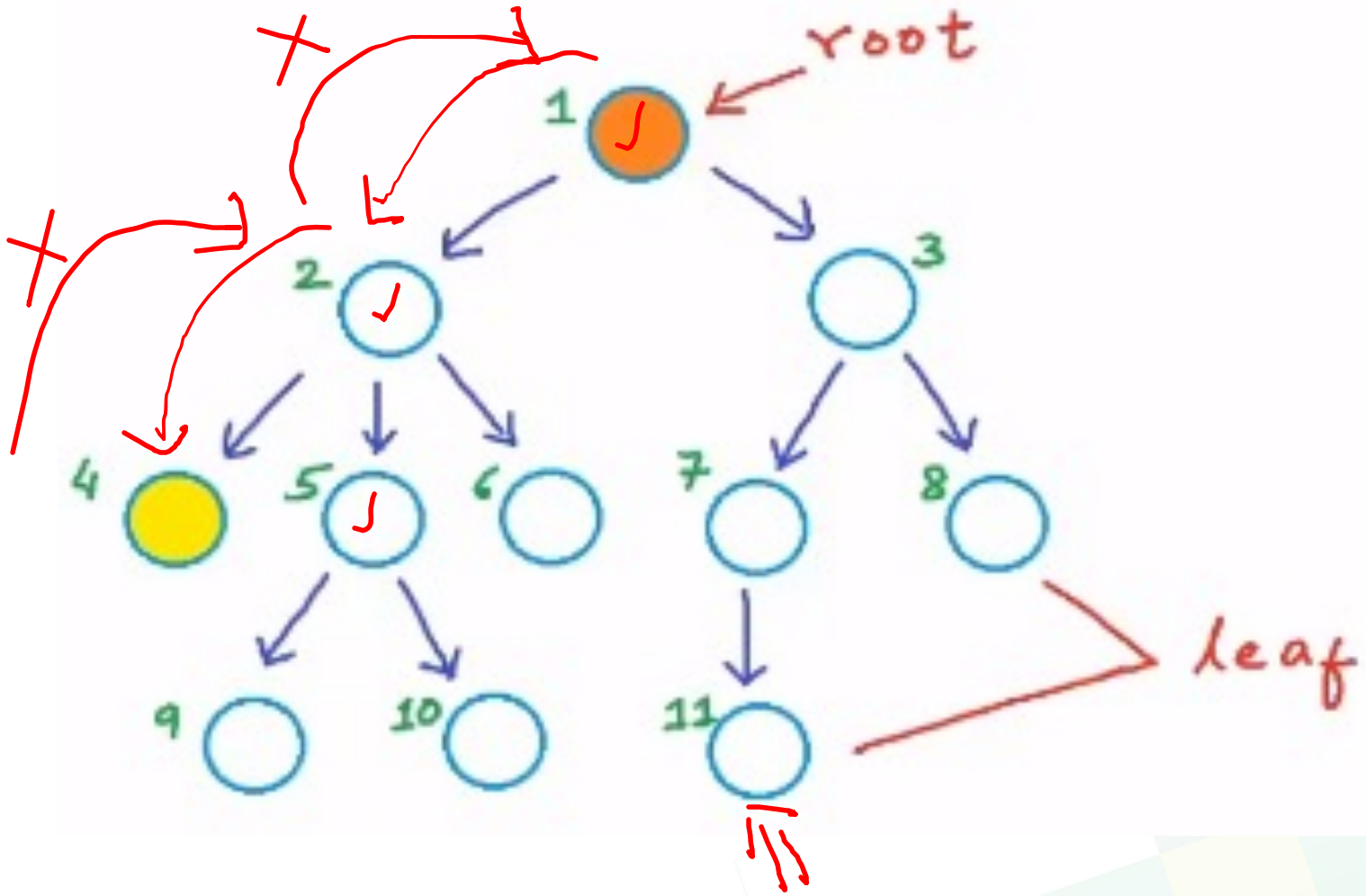
- Root
- Parent
- Children
- Sibling

Tree Terminologies



- Root
- Parent
- Children
- Sibling
- Leaf
- Internal nodes

Tree Terminologies



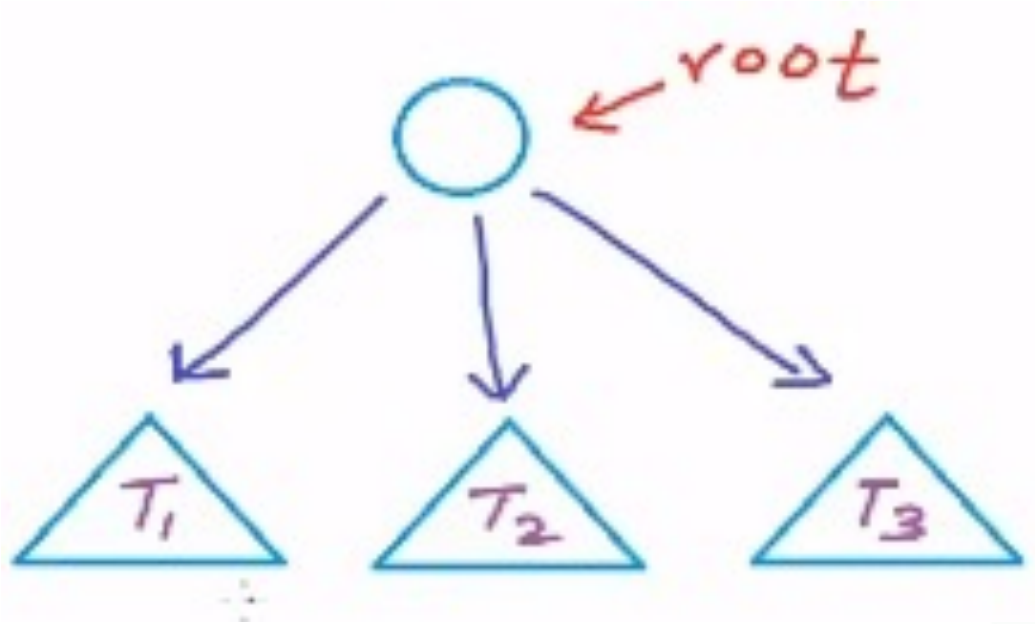
- Link in tree is not bidirectional.
- If we can go from A to B :
  - A is an ancestor of B
  - B is descendent of A
  - Example: 1, 2, 5 are all ancestors of 10 and 10 is a descendant of all of these nodes.

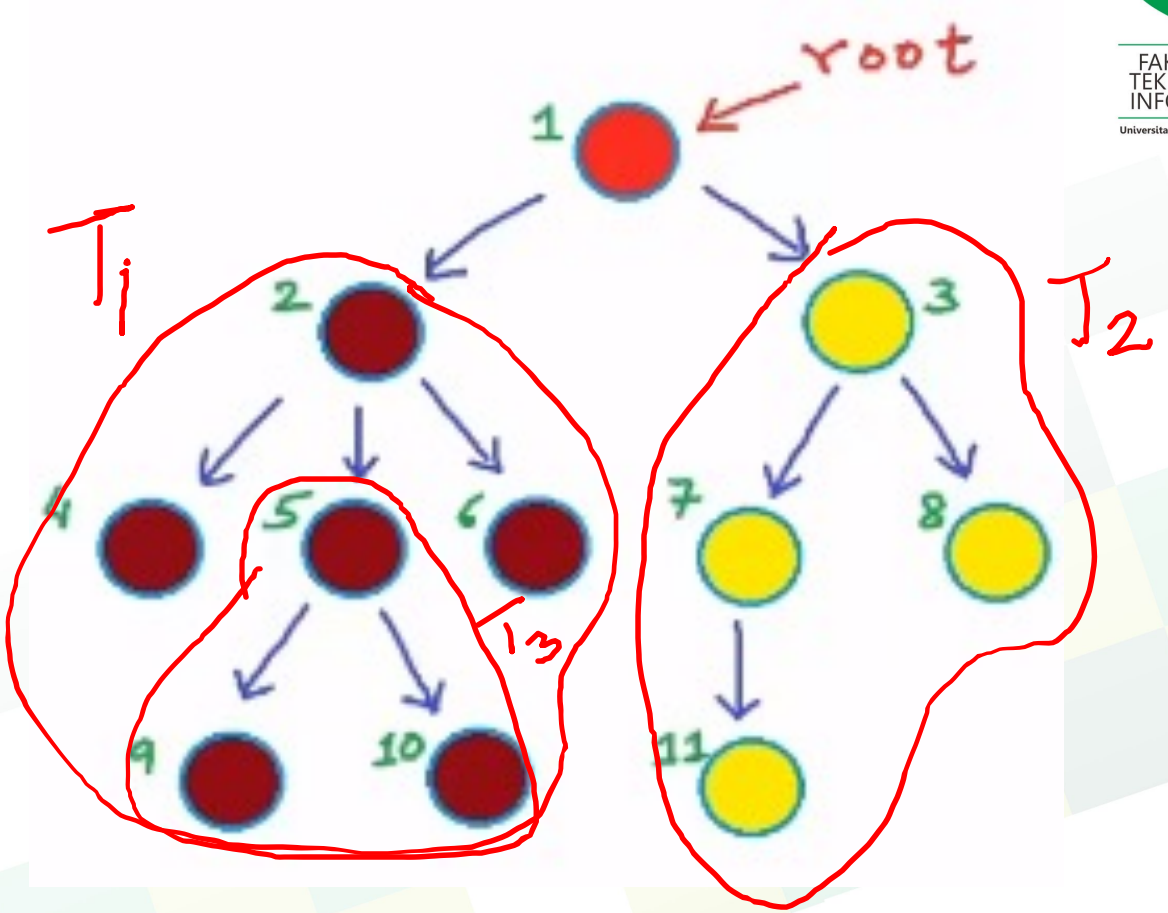
Tree Property

- Tree → recursive data structure

- We can define tree recursively as a structure that consists of a distinguished node called root and some sub-trees and the arrangement is such that root of the tree contains link to roots of all the sub-trees.

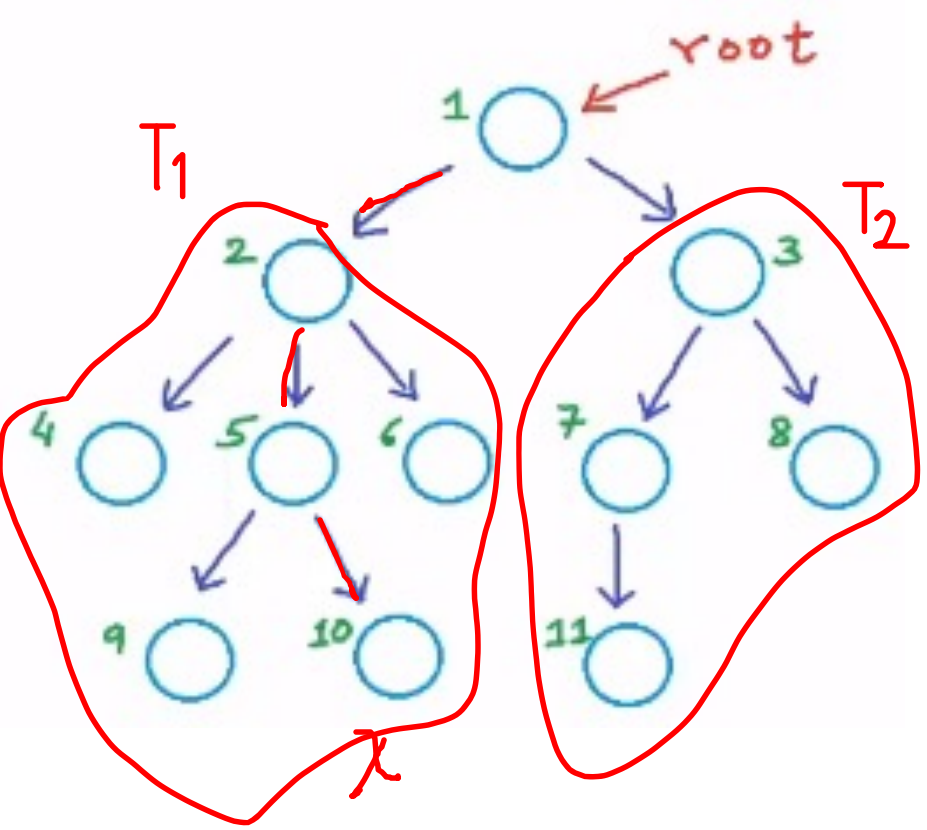
Tree Property





FAKULTAS TEKNOLOGI INFORMASI
Universitas Advent Indonesia

Tree Property



- Tree → N nodes will have N-1 edges (links).
- Depth and Height
  - Depth of x = length of path from root to x or number of edges in path from root to x. For example the depth of node 5 is 2.
  - Height of x = number of edges in longest path from x to a leaf. For example, for node 3, the longest path from this node to any leaf is 2.
  - Height of tree = height of root node

Tree Traversal

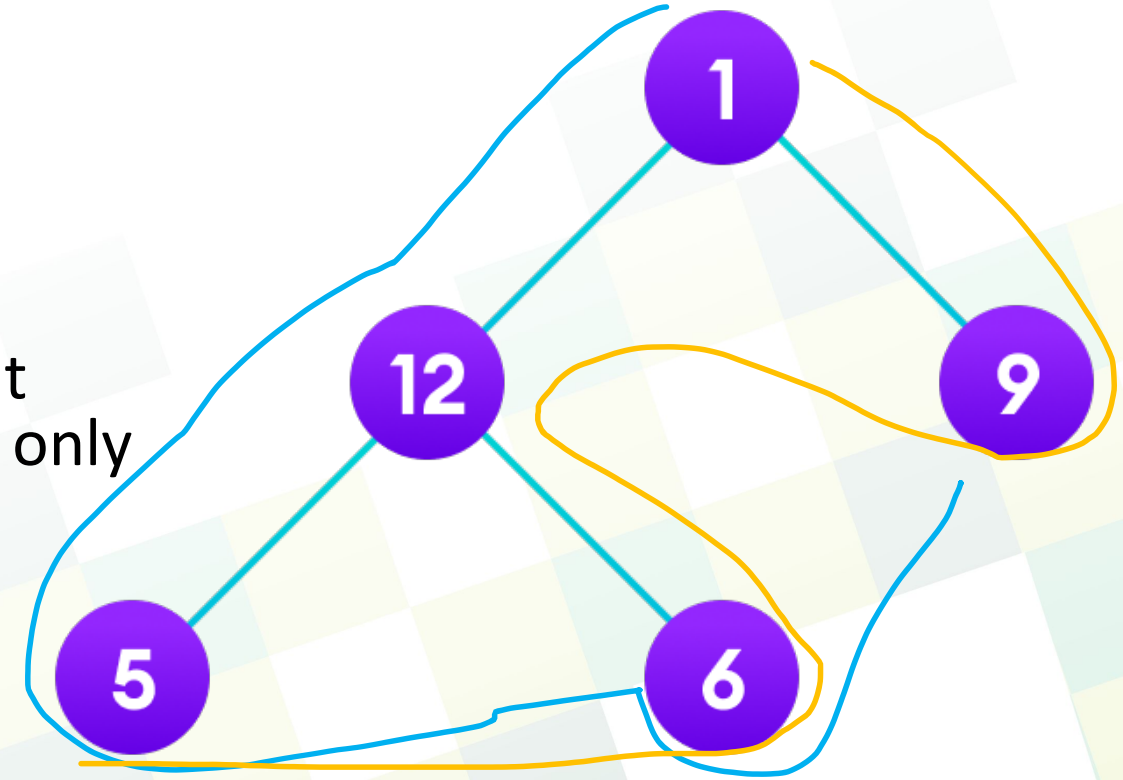
- Let's think about how we can read the elements of the tree in the image shown above.
- Starting from top, Left to right

1 -> 12 -> 5 -> 6 -> 9

- Starting from bottom, Left to right

5 -> 6 -> 12 -> 9 -> 1

- Although this process is somewhat easy, it doesn't respect the hierarchy of the tree, only the depth of the nodes.

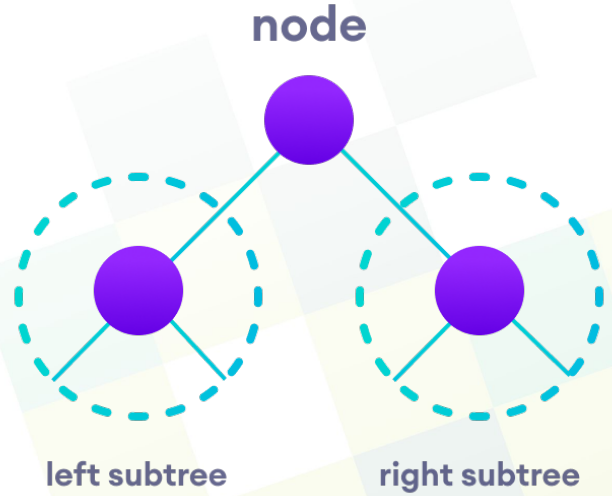


Tree Traversal

- We use traversal methods that take into account the basic structure of a tree

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

- The struct node pointed to by left and right might have other left and right children so we should think of them as sub-trees instead of sub-nodes.
- According to this structure, every tree is a combination of:
  - A node carrying data
  - Two subtrees



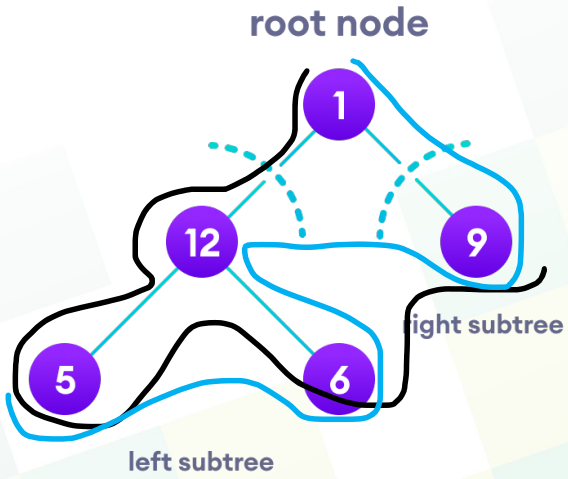
*Remember that our goal is **to visit each node**, so we need to visit all the nodes in the subtree, visit the root node and visit all the nodes in the right subtree as well.*

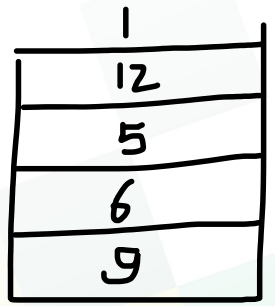
Tree Traversal

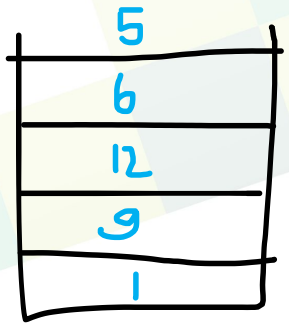
Depending on the order in which we do this, there can be three types of traversal.

- Inorder traversal
  - First, visit all the nodes in the left subtree
  - Then the root node
  - Visit all the nodes in the right subtree
- Preorder traversal
  - Visit root node
  - Visit all the nodes in the left subtree
  - Visit all the nodes in the right subtree
- Postorder traversal
  - Visit all the nodes in the left subtree
  - Visit all the nodes in the right subtree
  - Visit the root node

L → R → Rg

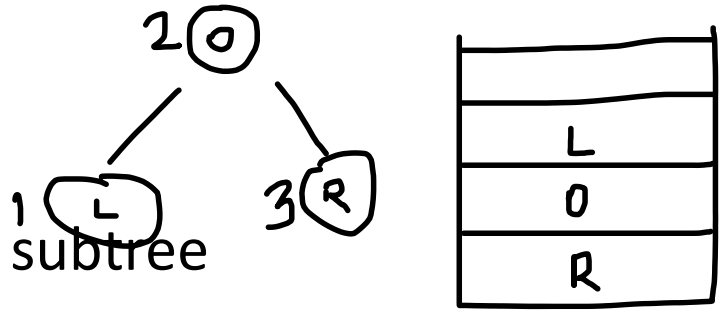


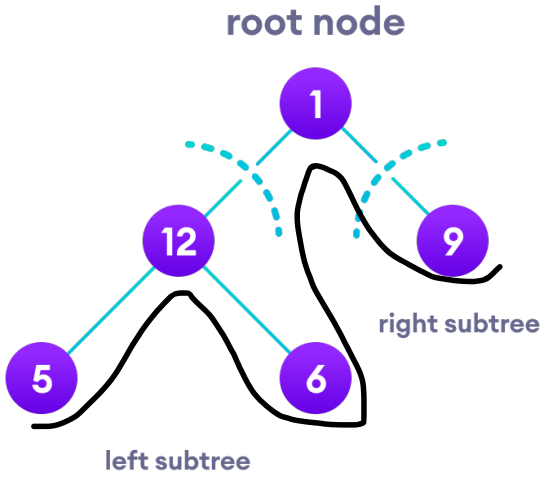




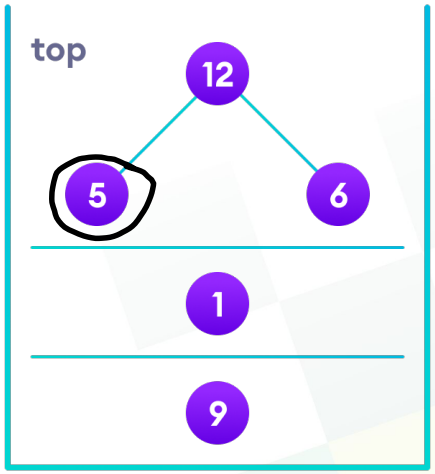
Tree Traversal Visualization

- Inorder traversal
  - First, visit all the nodes in the left subtree
  - Then the root node
  - Visit all the nodes in the right subtree

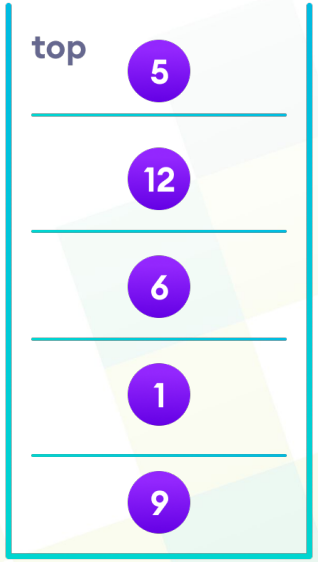




We traverse the left subtree first. We also need to remember to visit the root node and the right subtree when this tree is done.
Let's put all this in a stack so that we remember.



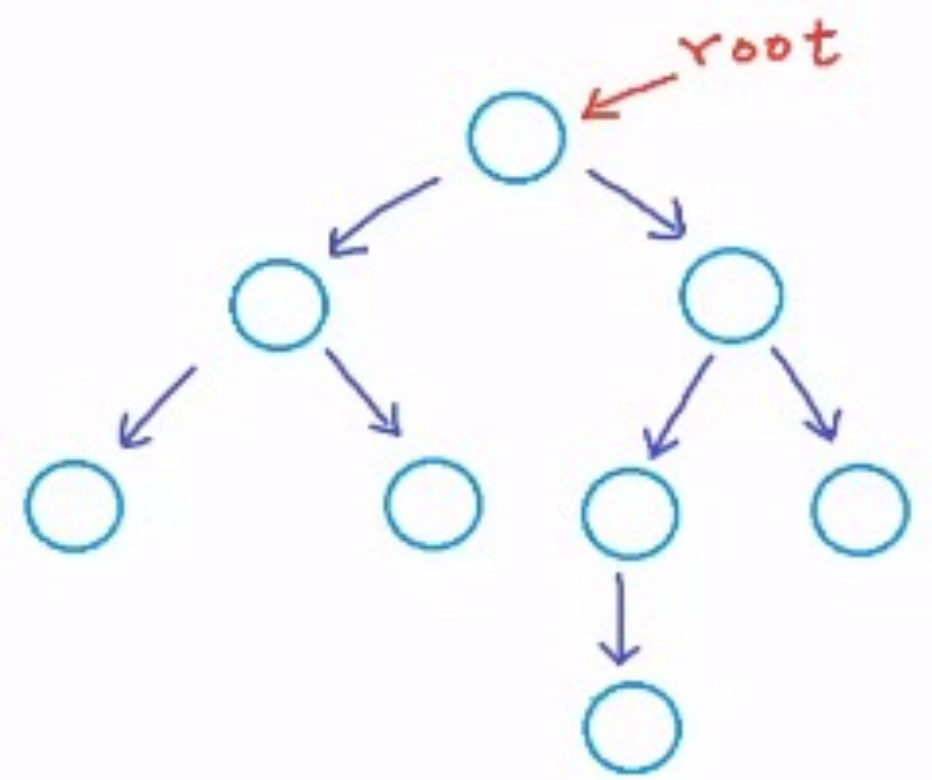
Now we traverse to the subtree pointed on the TOP of the stack. Again, we follow the same rule of inorder:
Left subtree -> root -> right subtree



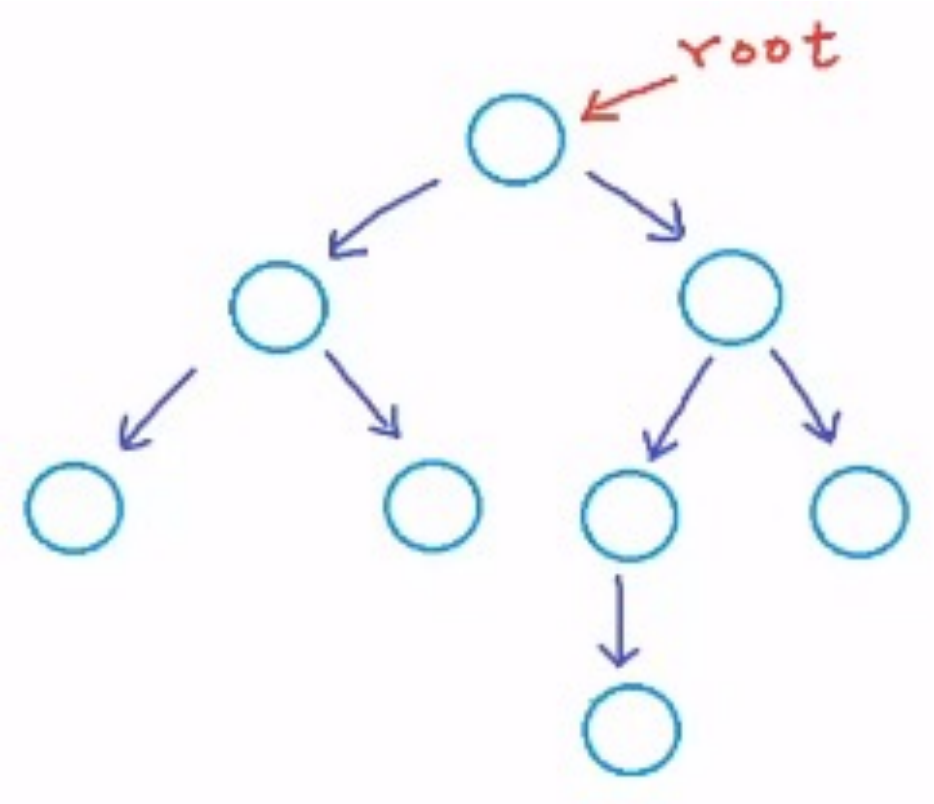
Since the node "5" doesn't have any subtrees, we print it directly. After that we print its parent "12" and then the right child "6". Putting everything on a stack was helpful because now that the left-subtree of the root node has been traversed, we can print it and go to the right subtree.
After going through all the elements, we get the inorder traversal as:
5 -> 12 -> 6 -> 1 -> 9

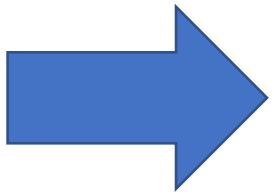
Binary Tree

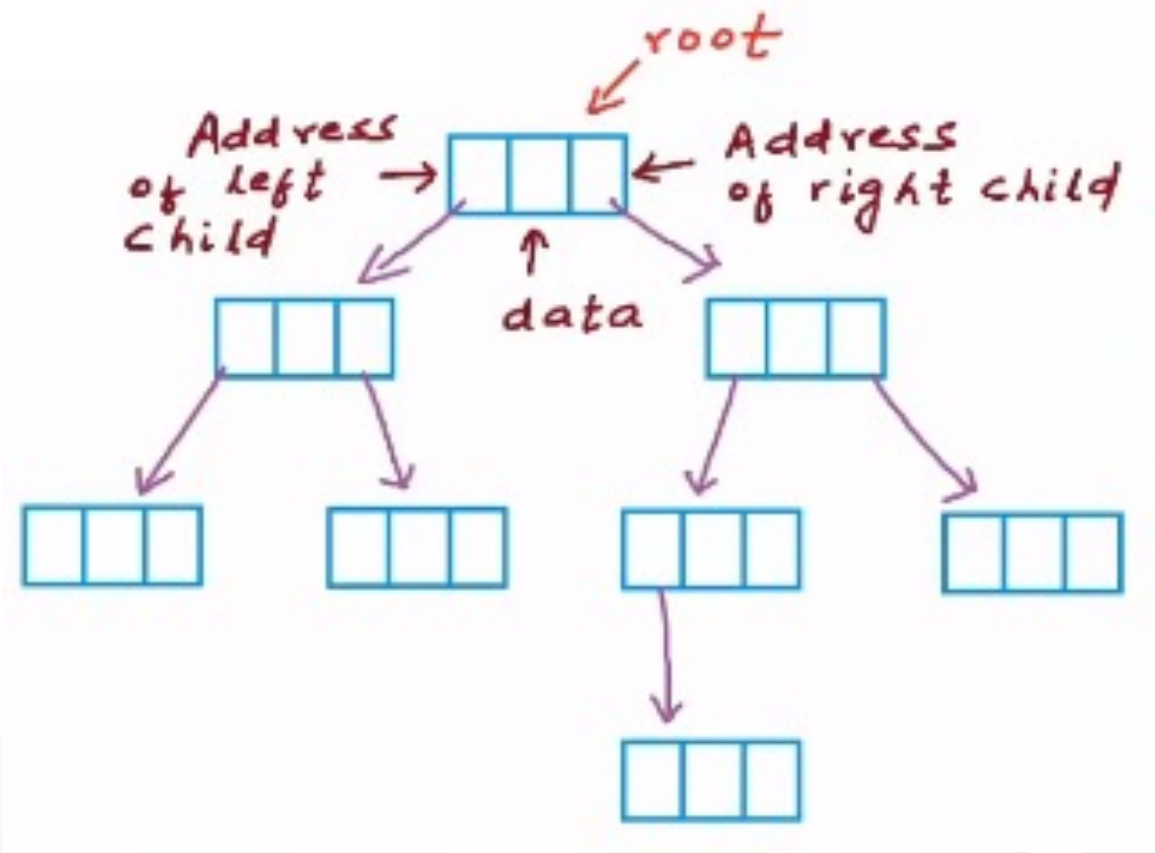
- Based on these properties, trees are classified into various categories.
- Simplest and most common kind of tree is a tree with this property that any node can have at most 2 children.
- Binary Tree → a tree in which each node can have at most 2 children
- The most common way of implementing tree is dynamically created nodes linked using pointers or references just like linked list.



Binary Tree







Tree Applications

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.