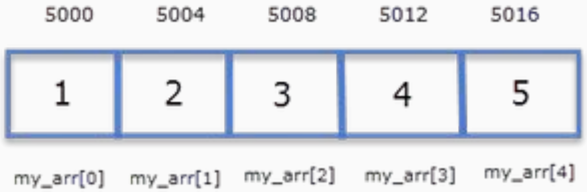
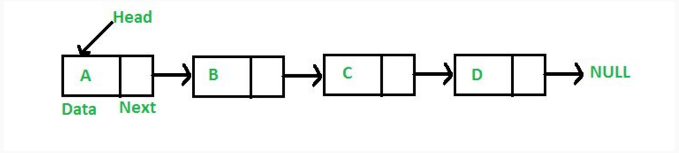
Binary Trees

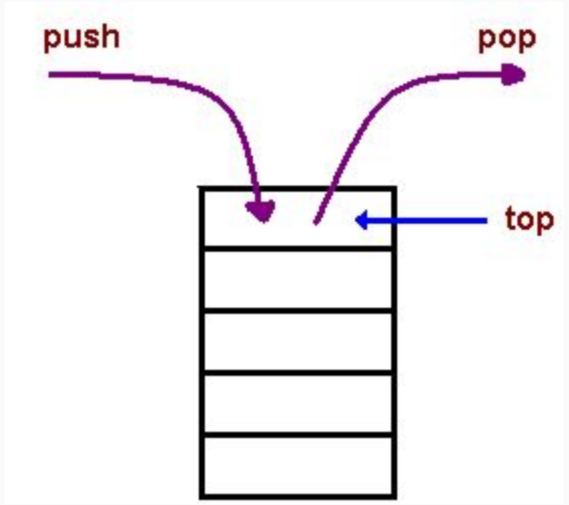
By: Aditya Sharma

Linear Data Structures

- Arrays
- Linked Lists
- Stacks
- Queues







Introduction to Trees

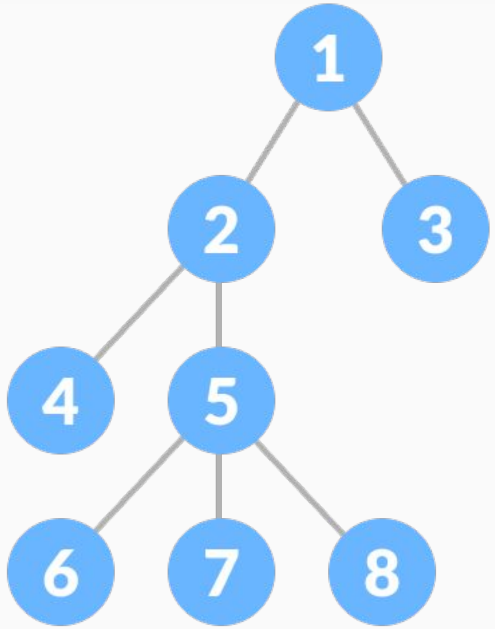
When analyzing Data Structures, often consider:

1. What needs to be stored?
2. Cost of Operation
3. Memory Usage
4. Ease of Implementation

Introduction to Trees

- **Definition:** A collection of nodes, linked together to simulate a hierarchy
- Form a Hierarchical Data Structure, organizational hierarchy
- Non-linear data structure.

Tree Structure Terminology

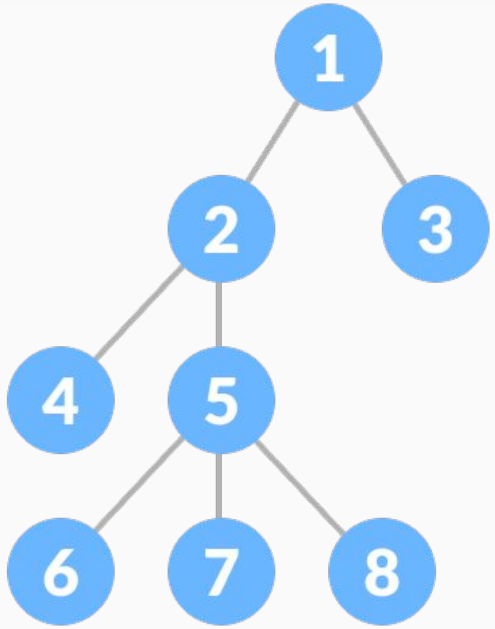


Each node will contain some data, of any type.

Nodes may contain links, or reference, to some other nodes

Root - topmost node in the tree. Root Node (1)
Parent - (1) is a parent of (2) and (3)
Children - (4) is a child of (2)
Sibling - Children of same parent (2) and (3) are siblings

Tree Structure Terminology



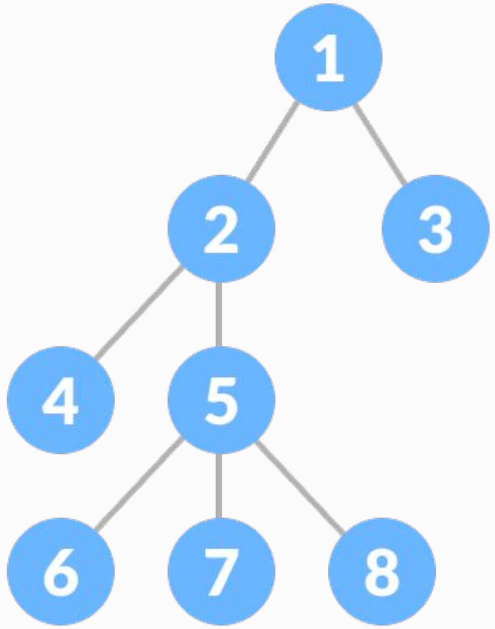
Root is the **<u>only</u>** node without a parent.

<u>Leaf Node</u> - any node in the tree without a child.
Ex: (3) (4) (6) (7) (8) are leaf nodes

Since we can go from (1) to (8).
- (8) is an ancestor of (1).
- (1) is a descendent of (8)

Find common ancestors of (4) and (6).

Subtrees



Each root node contains links to the **subtrees**.

Node (2) is the root of the left subtree.

A tree of n nodes has $n-1$ links/edges.

Depth - Number of edges in the path from root to a given node
Ex: depth(6) = 3. Travel 1 -> 2 -> 5 -> 6
Height - Number of edges in longest path from given node to a leaf. Generally for **root node**.

Binary Tree

- **Definition:** A tree in which each node can have at most 2 children.

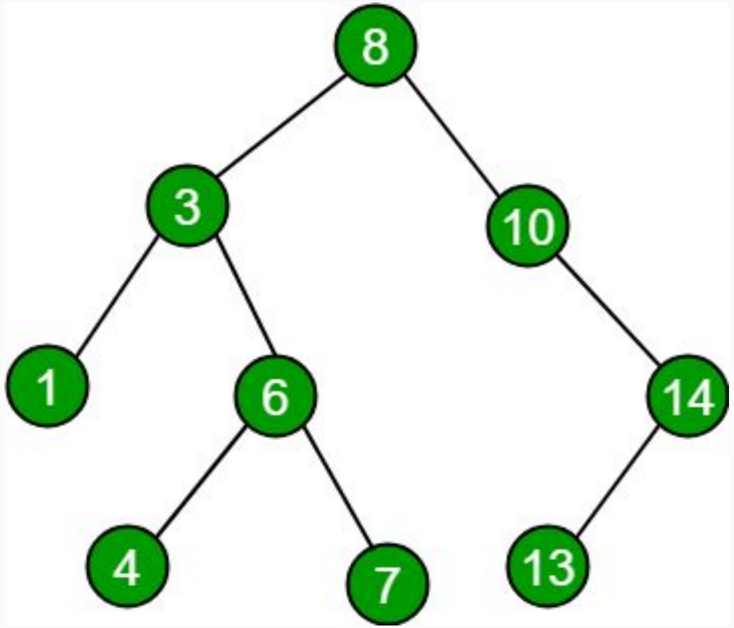
In the implementation of a Binary Tree, we have the pointer references to the left and right nodes and the data in the current node.

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

Binary Trees

- Nodes can have a left child + right child
  Ex: (8) {left child: (3), right child: (10)}
- Nodes can have only 1 child
  Ex: (14) {left child: (13), right child: NULL}
- Nodes (leaf nodes) have no children
  Ex: (13) {left & right child: NULL}

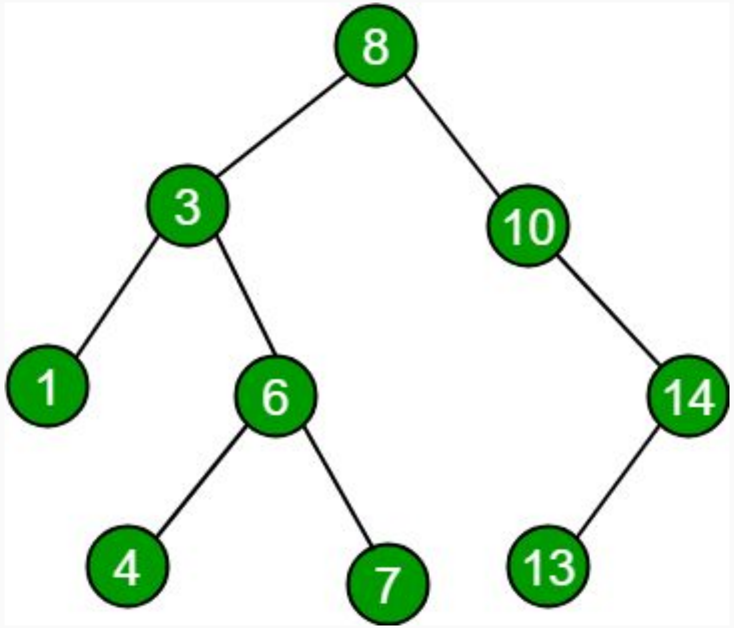
- Complete Binary Tree has 2^i nodes at a level i.



Binary Search Tree

Def: A binary tree in which for each node, the value of all the nodes in the left subtree are smaller than the root and the
value of all the nodes in the right subtree are greater than the root.

(8) - left subtree: $n < 8$
(3) - right subtree: $3 < n < 8$
(14) - left subtree: $10 < n < 14$

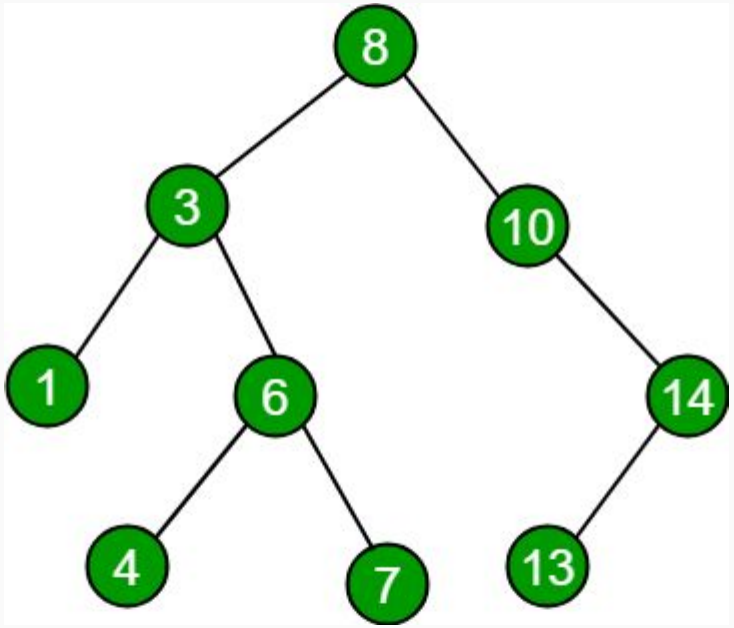


Searching

search(4).

- Start at root node (8): 4 < 8 {left subtree}
- Root node of left subtree (3): 4 > 3 {right}
- Root node of the right subtree (6): 4 < 6 {left}
- 4 = 4. So item is found

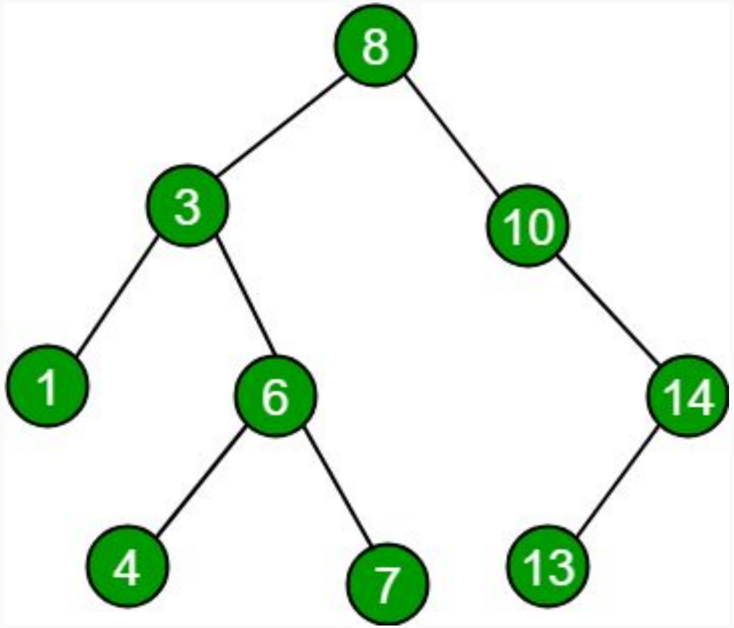
Time Complexity: Each time we search, ideally we reduce our searching to `n/2`. Binary Search
O(n) = $\log_2(n)$ [Balanced BST]
O(n) = n [Unbalanced BST: func like Linked list]



Insertion

insert(9).
- Root node (8): 9 > 8 {right subtree}
- (10): 9 < 10 {left}
- Left of (10) is NULL. so insert as left node.

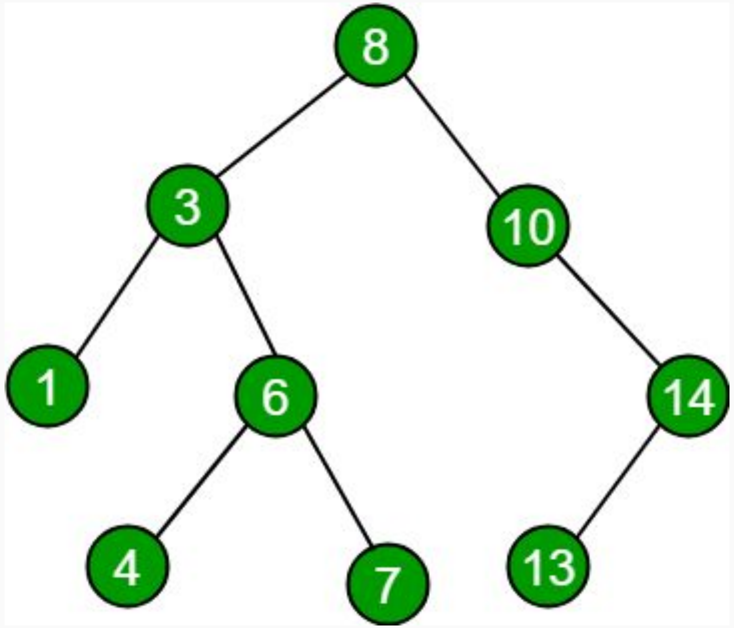
insert(11).
- Repeat same as (9).
- (14): 11 < 14. So left subtree
- (13): 11 < 13. So left subtree
- Left of (13) is NULL. so insert as left node



Insertion

Time Complexity: $O(n) = \log_2(n)$.

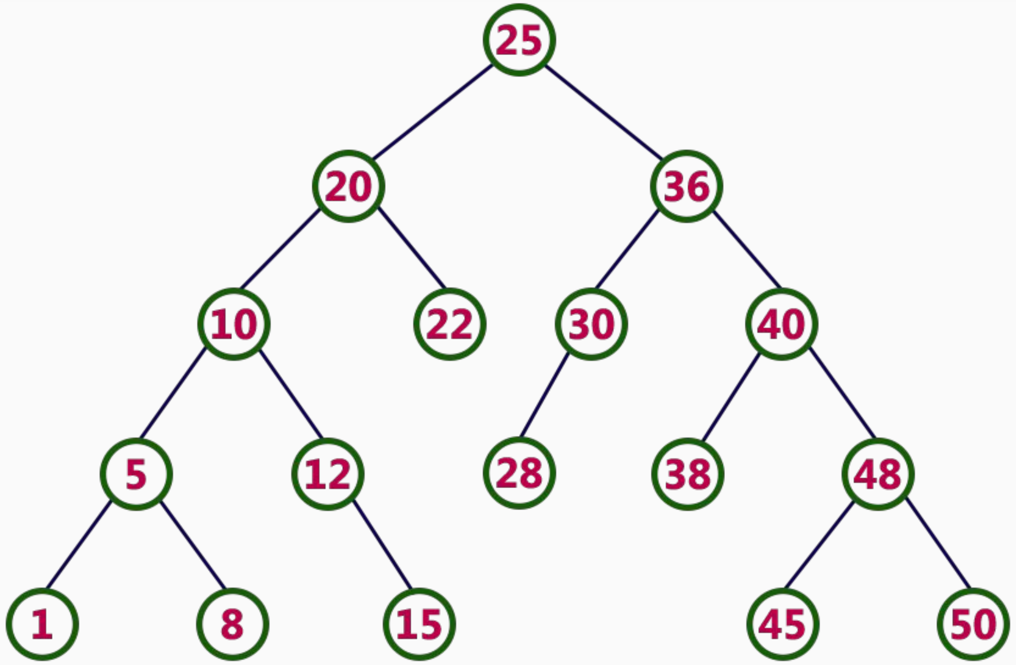
However, during insertion the binary tree may get unbalanced. Often times it may be useful to run an algorithm to re-balance the tree.



Tree Traversals

1. Breadth-first Search

2. Depth-first Search
   a. Preorder
   b. Inorder
   c. Postorder



Breadth-first Search

Goal: Go level-by-level starting from the root.

height(20) = height(36) = 1.

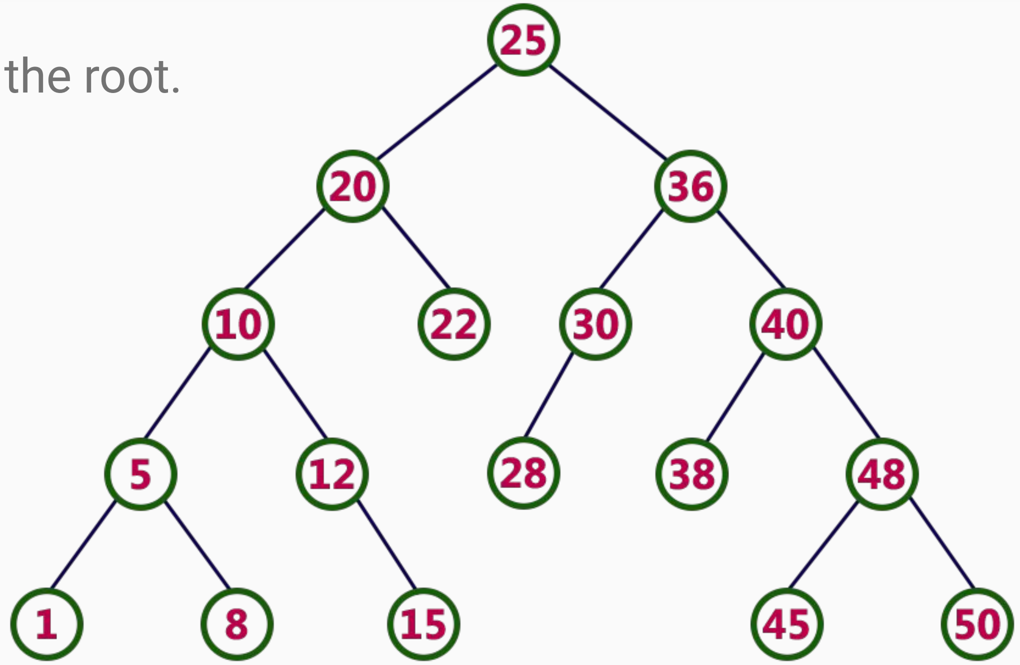
Traverse:

25

20, 36

10, 22, 30, 40

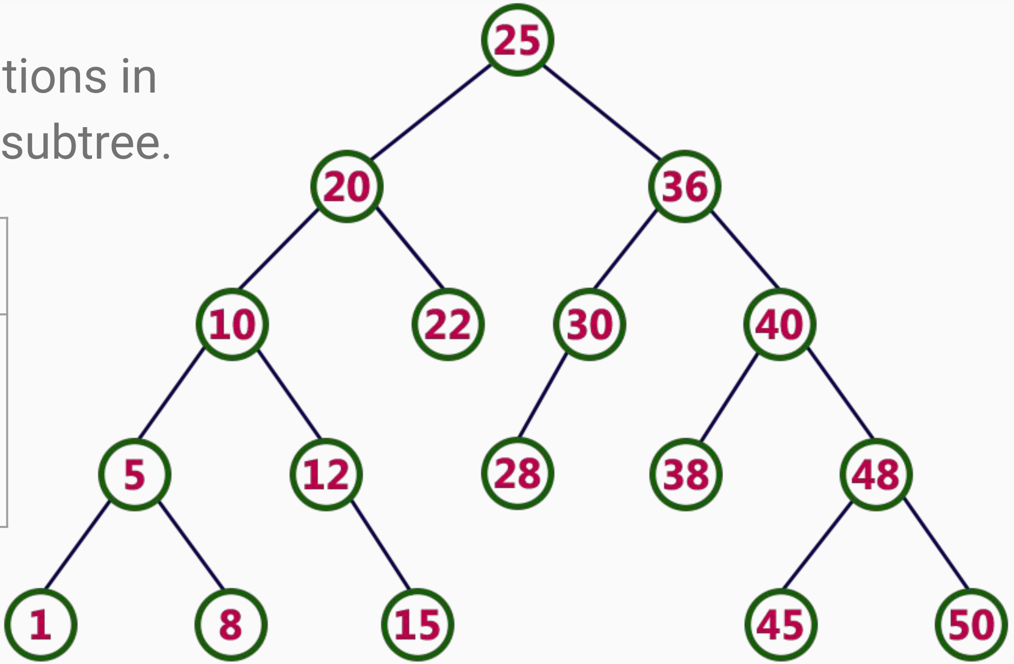
...



Depth-First Search

Goal: Visit root, then do all the operations in left subtree, then operations in right subtree.

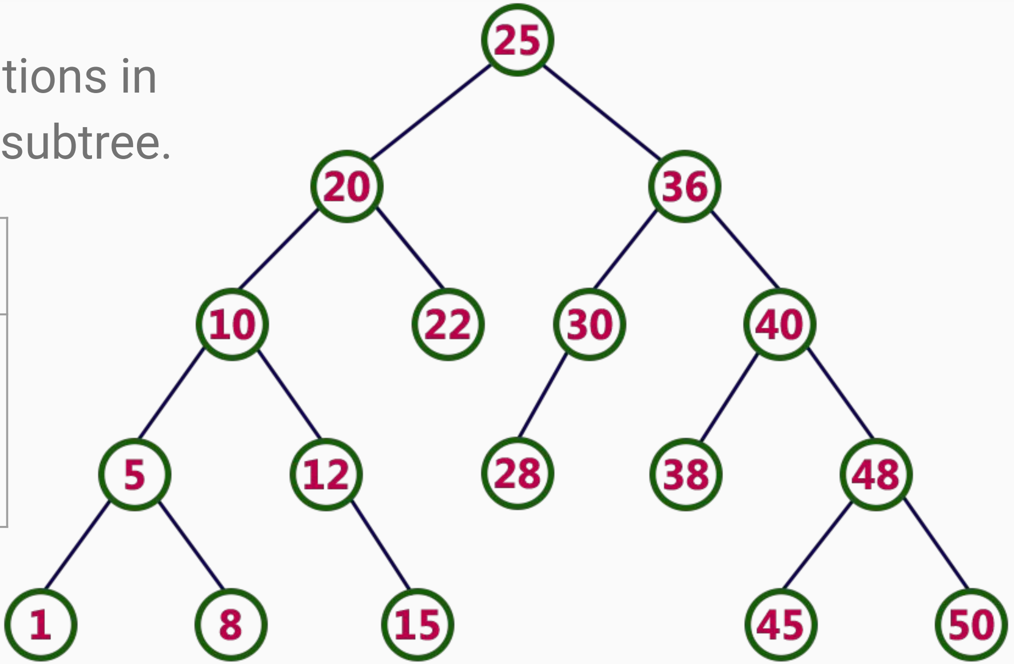
Pros	Cons
Easier to implement with recursive calls	Harder to follow visually



Depth-First Search

Goal: Visit root, then do all the operations in left subtree, then operations in right subtree.

Pros	Cons
Easier to implement with recursive calls	Harder to follow visually



Applications

- Storing hierarchical data. Ex: file system.
  Users -> Desktop -> CS 1A -> lab00

- Organize Data for quick search, insertion, deletion.
  The time complexity of tree traversals can be fast.
  BST operations are cost-effective.

Live Coding Demo