Data structures:

Trees

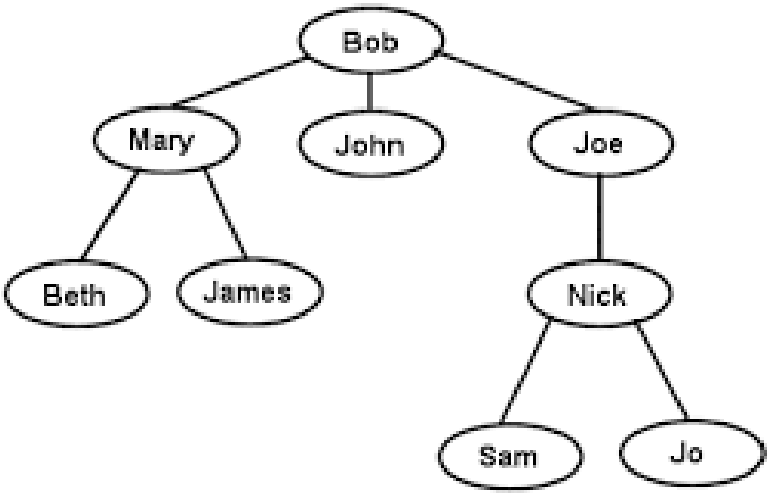
Dr. Sophea PRUM
sopheaprum@gmail.com

Data structures

- Linear: List (array, ArrayList, LinkedList), Stack, Queue
- Non-linear: Tree and Graph

Linear data structures

- Limitation
  - Each element has only one predecessor and/or one successor
  - Unable to represent hierarchical organization of information unless using a complex representation method
  - Example: family tree



Non-linear data structures: Tree

- Non-linear or hierarchical data structure
- Node in a tree can have **multiple successor**s, but it has just **one predecessor**
- Tree is a recursive data structure
  - many of the methods used to process trees are written as recursive methods

Tree terminology

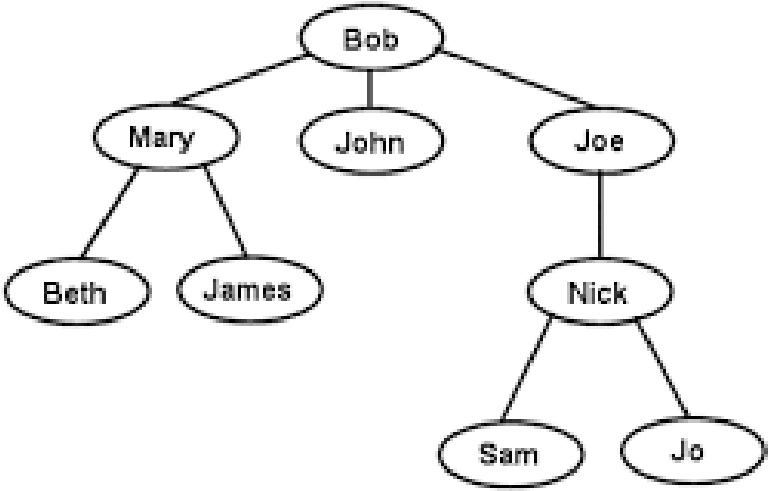
- Tree consists of a collection of elements or **nodes**
- Each node linked to its successors
- The node at the top of a tree is called its **root**
- The links from a node to its successors are called **branches**
- The successors of a node are called its **children**

Tree terminology

- The predecessor of a node is called its **parent**
- Each node in a tree has exactly one parent except for the root node
- Root node has no parent
- Nodes that have the same parent are siblings
- A node that has no children is a **leaf** node or **external** nodes
- Non-leaf nodes are known as **internal** nodes

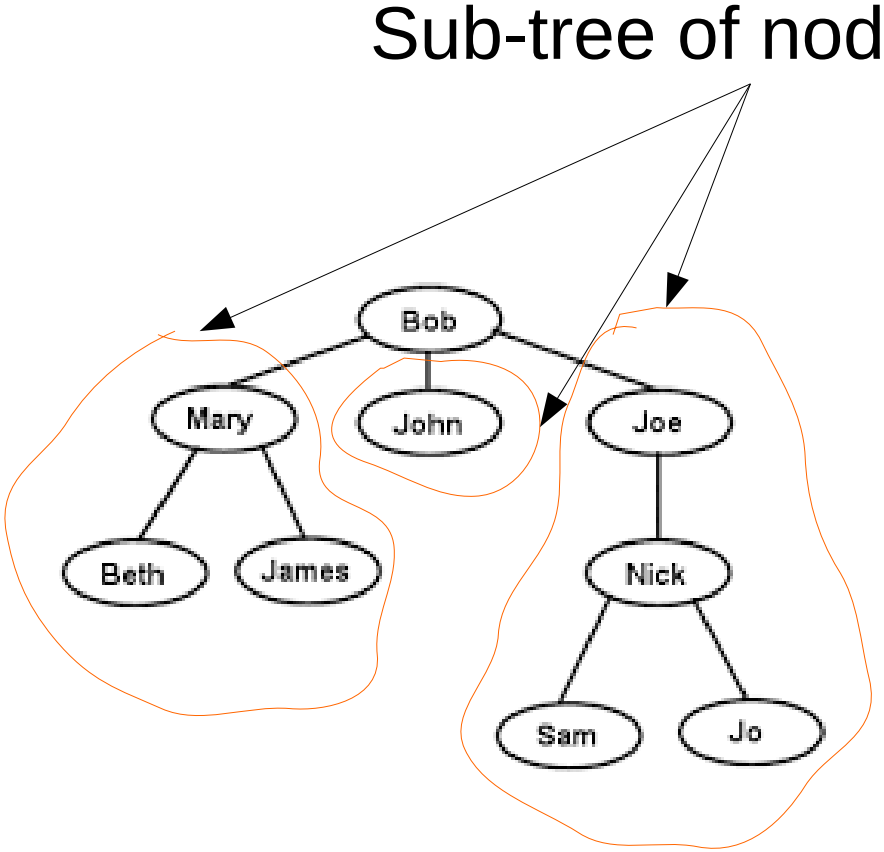
Tree terminology

- Example:
  - Node “Bob” is root node of the tree
  - Nodes “Mary”, “John” and “Joe” are children node of node “Bob”
  - Node “Mary” is the parent node of nodes “Beth” and “James”
  - Leaf nodes of external nodes: “Beth”, “James”, “Sam” and “Jo”
  - Non-leaf nodes or internal nodes: “Bob”, “Mary”, “John”, “Jeo” and “Nick”



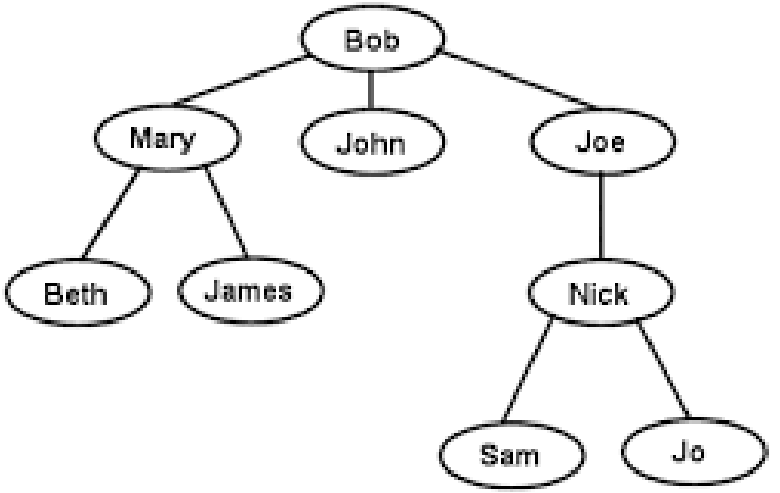
Tree terminology

- A **subtree** of a node is a tree whose root is a child of that node
  - Example:



Tree terminology

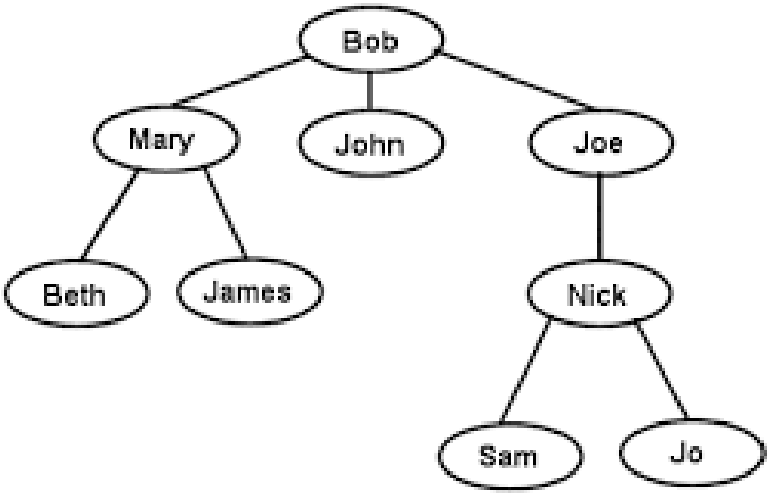
- **Level or depth of node**
  - If node **n** is the root of the tree, its level is **1**
  - If node **n** is not the root of the tree, its level is **1 + the level of its parent**
  - We sometimes use the term **depth** as an alternative term for level
  - Example:
    - Node “Bob” is at level 1
    - Node “Mary” is at level 2
    - Node “James” is at level 3
    - Or Node “James” is at level 1 + level of node “Marry



Tree terminology

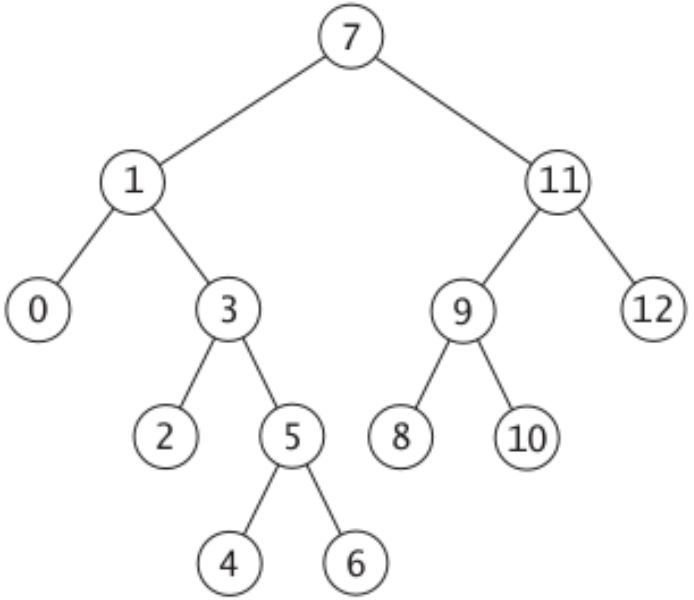
- **Height of a tree**
  - If T is empty, its height is 0
  - If T is not empty, its height is the maximum depth or level of its nodes.
    - Example: The height of this tree is 4

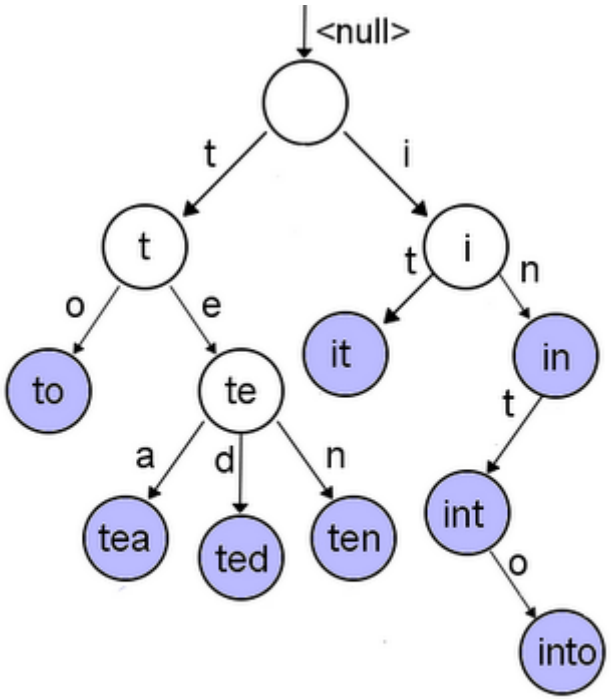
      (the longest path goes through the nodes : “Bob”, “Joe”, “Nick”, and “Sam”, or “Jo”)



Two type of Trees

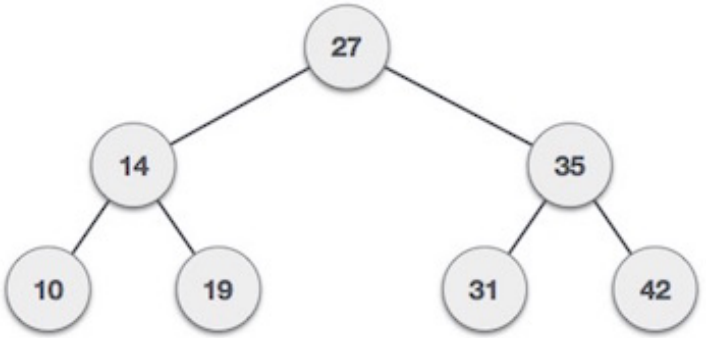
- Binary trees
  - each node has **at most** two successors
- General trees
  - No such constraint

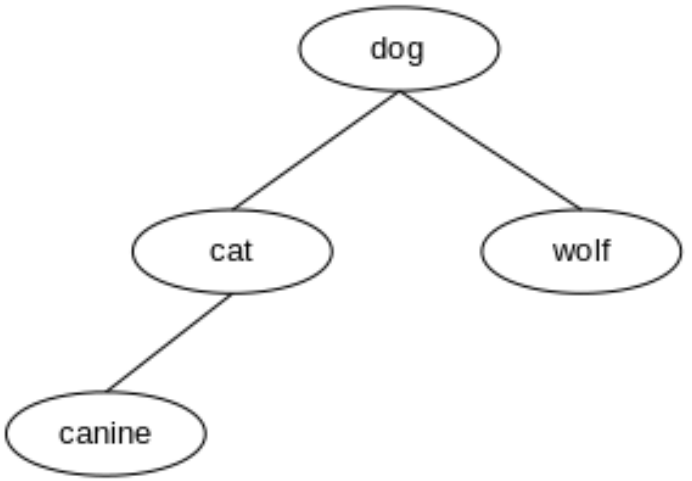




Binary trees

- Binary search tree
  - Typically created for binary search algorithm
  - The left sub-tree of a node has a value less than or equal to its parent node's value
  - The right sub-tree of a node has a value greater than to its parent node's value



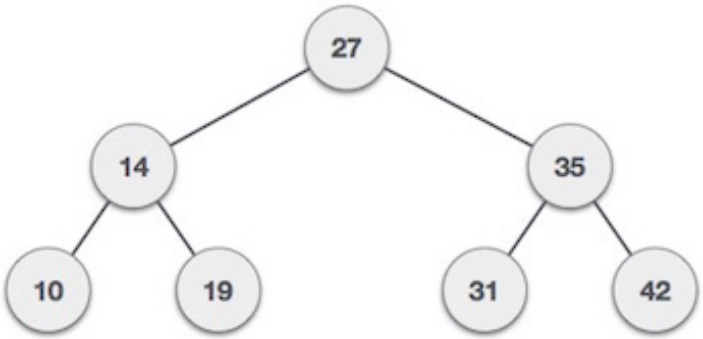


Binary trees

- Binary search tree
  - Traversals: Preorder traversal

1)If current node is empty
  - Return

2)Process current node
3)Preorder traversal the left subtree
4)Preorder traversal the right subtree

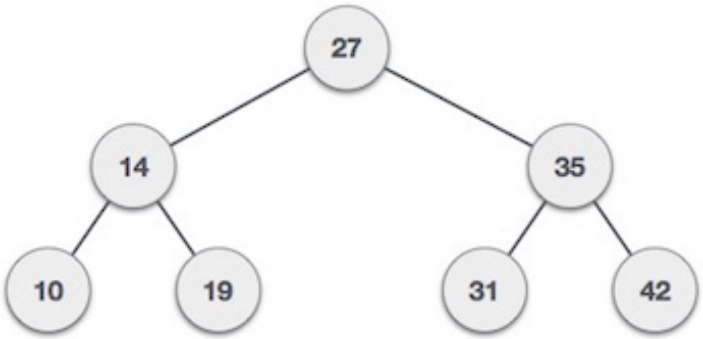


Example: view tree using preorder traversal
==> 27, 14, 10, 19, 35, 31, 42

Binary trees

- Binary search tree
  - Traversals: Inorder Traversal

1)If current node is empty
  1)Return
2)Inorder traversal the left subtree
3)Process current node
4)Inorder traversal the right subtree

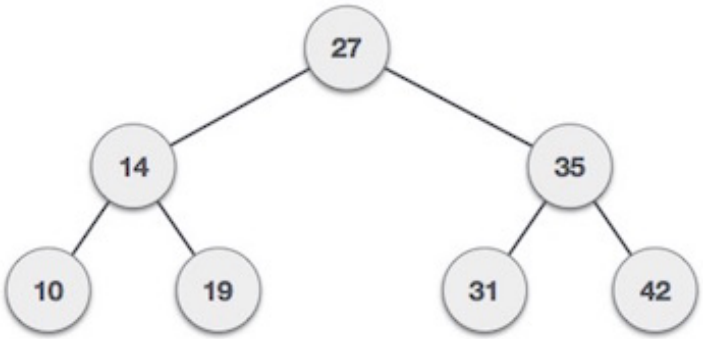


Example: view tree using inorder traversal
==> 10, 14, 19, 27, 31, 35, 42

Binary trees

- Binary search tree
  - Traversals: Postorder Traversal

1)If current node is empty
  1)Return
2)Postorder traversal the left subtree
3)Postorder traversal the right subtree
4)Process current node



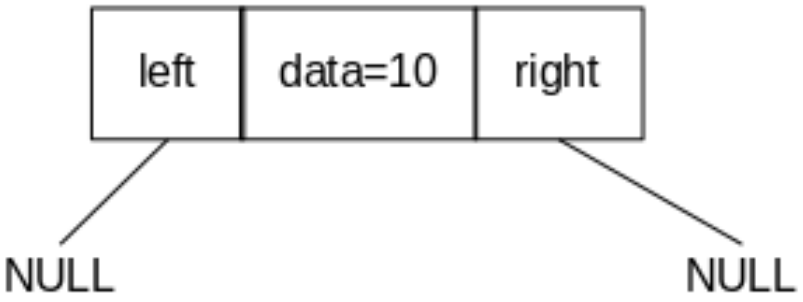
Example: view tree using Postorder traversal
==> 10, 19, 14, 31, 42, 35, 27

Binary trees

- Binary search tree
  - Create the Node class

```
class Node{
      Object data;
      Node left;
      Node right;
      public Node(Object data){
            this.data = data;
            left = null;
            right = null;
      }
}
```

Node node = new Node(10)



Binary trees

- Binary search tree
  - Create the BinarySearchTree class

```
class BinarySearchTree{
	Node root;
	public BinarySearchTree(Node root){
		this.root = root;
	}
}
```

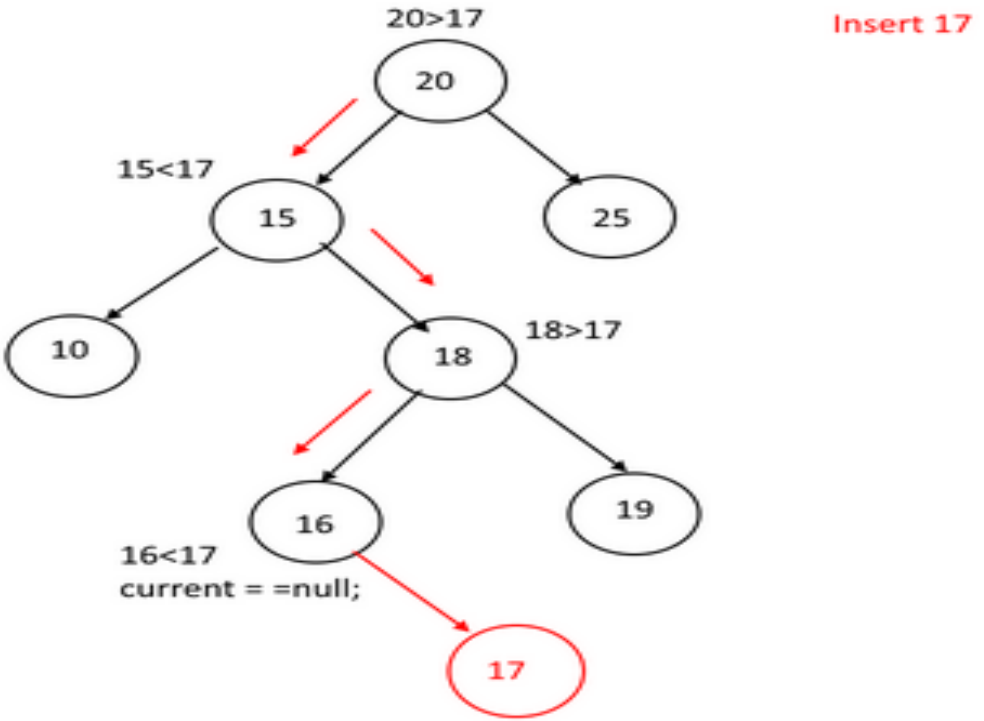
Methods:

- Insert a new node
- Find a node
- IsLeaf
- View the tree
- Delete a node

Binary trees

- Binary search tree
  - Insert a new node:

    void insert(Node current, Object newData)



Binary trees

- Binary search tree
  - Insert a new node:

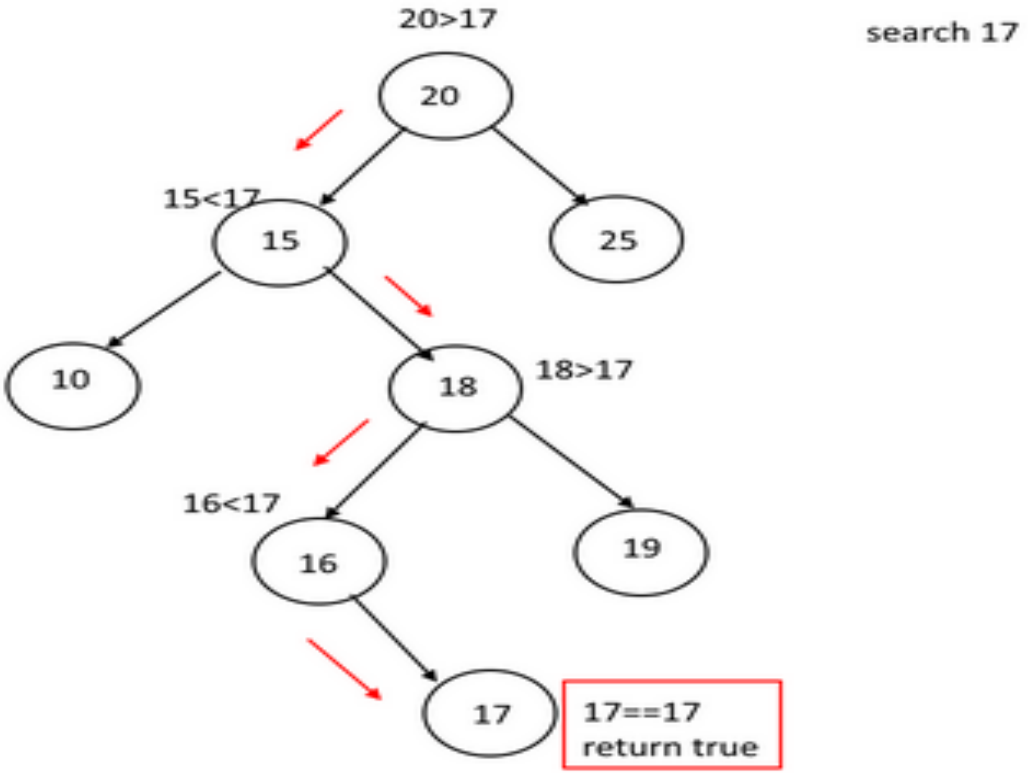
    void insert(Node current, Object newData)

    1) compare *current*.data with *newData*
    2) if *newData* is **greater** than *current.data* then insert in the **right subtree** (insert(current.right, newData))
    3) if *newData* is **smaller** than *current.data* then insert in the insert(current.left, newData) (insert(current.left, newData))
    4) if any point of time *current* is null that means we have reached to the leaf node, insert your node here

Binary trees

- Binary search tree
  - Find an object:

    Node find(Node current, Object object)



Binary trees

- Binary search tree
  - Find an object:

    Node find(Node current, Object object)

    1) If *current.data* is equal to the *object* then we have found the node, return true
    2) If *current is null*, we did not found the element, return false
    3) If *object* is **greater** than *current.data* then, find in the **right subtree** (find(current.right, object))
    4) If *object* is **smaller** than *current.data* then, find in the **left subtree** (find(current.left, object))

Binary trees

- Binary search tree
  - isLeaf an object:

    boolean isLeaf(Node current)

    1) If current.left == null & current.right==null
       - Return true
    2) else
       - Return false

Binary trees

- Binary search tree
  - View the tree

    void view(Node current)
    - 3 method to view the tree:
      - Preorder traversal
      - Inorder traversal
      - Postorder traversal

Binary trees

- Binary search tree
  - Delete one node

    void delete(Node deleteNode)

    - 3 cases to be considered

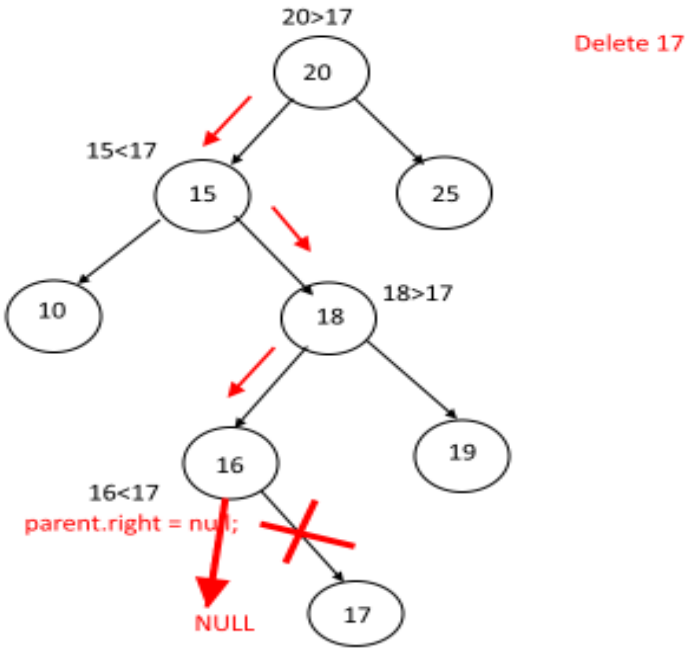
      1)If node to be deleted (deleteNode) is leaf
      2)If node to be deleted (deleteNode) has only one subtree
      3)If node to be deleted (deleteNode) has two subtree

Binary trees

- Binary search tree
  - Delete one node

    void delete(Node deleteNode)

    - If node to be deleted (deleteNode) is leaf



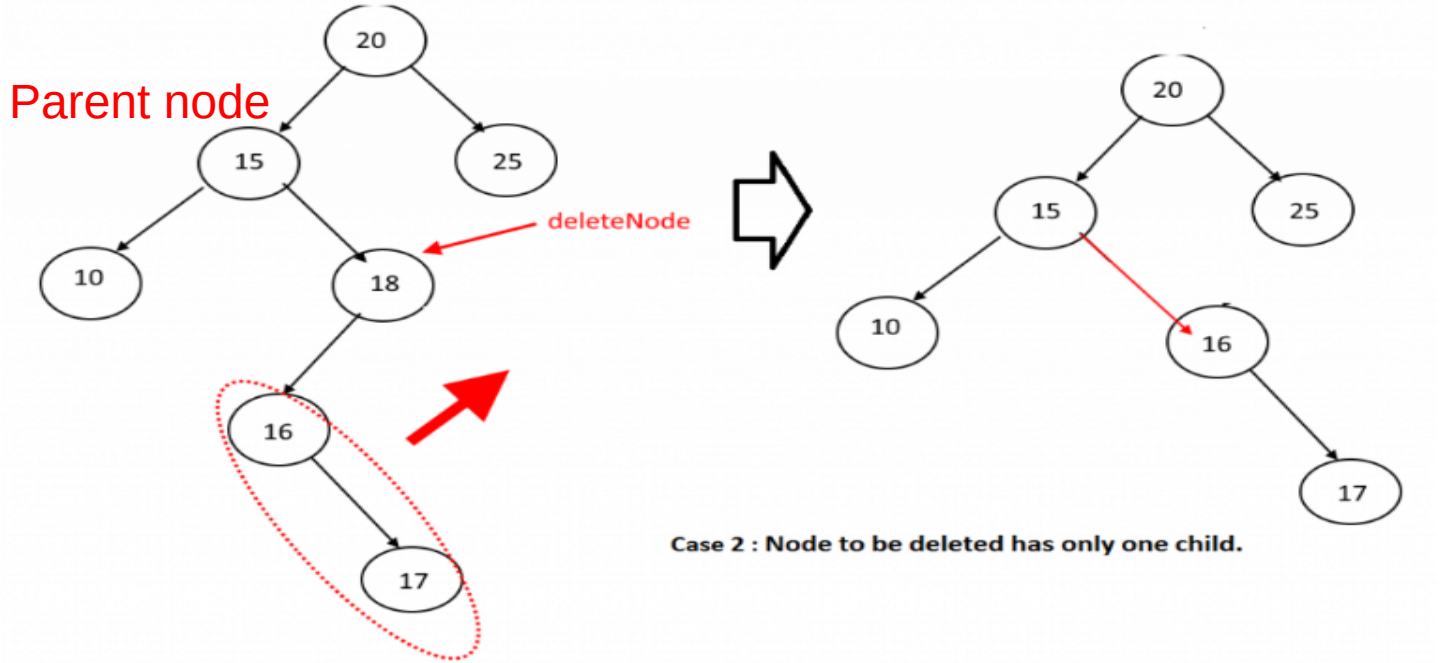
Case 1 : Node to be deleted is a leaf node (No Children).

Binary trees

- Binary search tree
  - Delete one node

    void delete(Node deleteNode)

    - If node to be deleted (deleteNode) has only one subtree
      - Link the parent node to the subtree



Binary trees

- Binary search tree
  - Delete one node

    void delete(Node deleteNode)

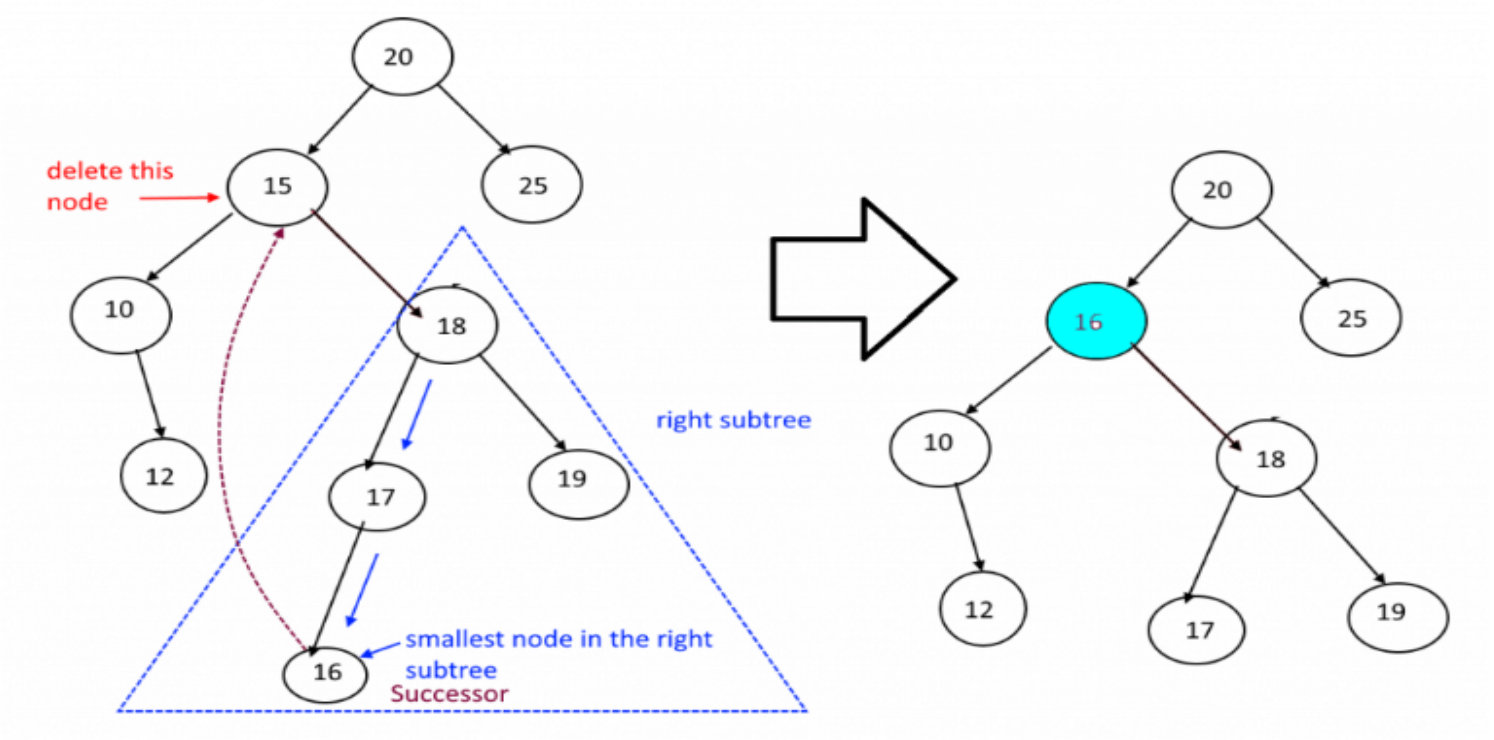
    - If node to be deleted (deleteNode) has two subtrees

      1) Find the successor node

      2)Replace the to be deleted node data with the successor node data

Binary trees

- Successor node: the smaller node in the right sub tree of the node to be deleted.



Resume

- Terminology of tree
- Binary tree
  - Binary search tree
    - Crate Node class
    - Create binary search tree class
      - Insert a node
      - Find a node
      - IsLeaf method
      - Delete a node