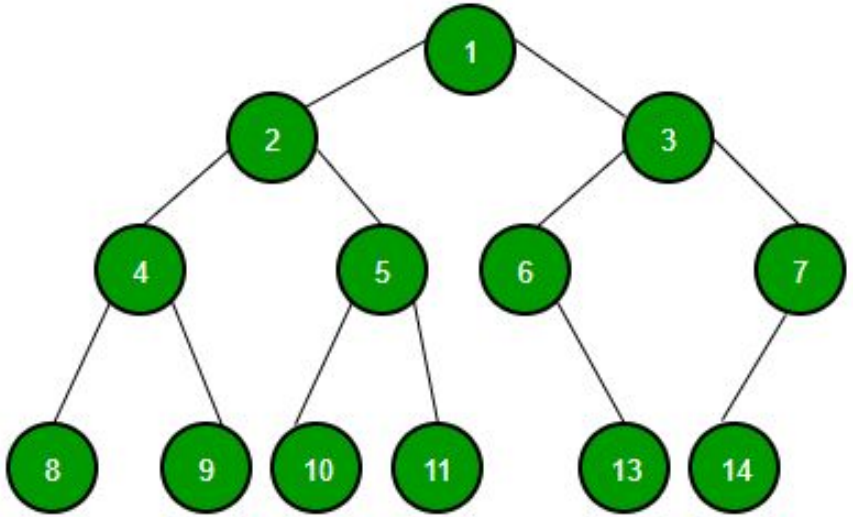
CMPUT 175 Lab 10- Binary Tree

UNIVERSITY OF ALBERTA

Binary Tree

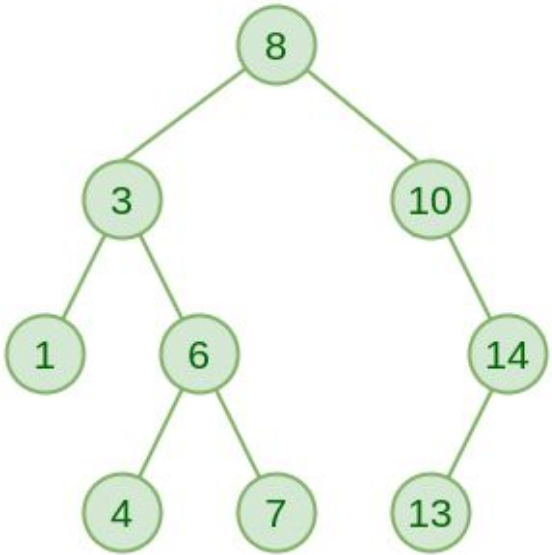
A binary tree is a hierarchical data structure where each node can have **at most two** children: a left child and a right child.



Binary Search Tree (BST)

A specialized form of a binary tree where each node follows a specific ordering rule:

- All nodes in the left subtree contain values smaller than the root node's value.
- All nodes in the right subtree contain values greater than the root node's value.



Binary Tree class implementation

```
class BinaryTree:
    def __init__(self, rootElement):
        self.key = rootElement
        self.left = None
        self.right = None

    '''getters'''
    def getLeft(self):
        return self.left

    def getRight(self):
        return self.right

    def getKey(self):
        return self.key

    '''setters'''
    def setKey(self,key):
        self.key=key

    def setLeft(self,left):
        self.left=left

    def setRight(self,right):
        self.right=right
```

```
def insertLeft(self,newNode):
    if self.left == None:
        self.left = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.left = self.left
        self.left = t

def insertRight(self,newNode):
    if self.right == None:
        self.right = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.right = self.right
        self.right = t
```

Insert without any conditions like BST

Binary Search Tree (BST) Implementation

- Like Linked List Structure Needs for **2** Classes
  - TreeNode Class: Represents **individual nodes** holding (key, value) pairs.
  - BinarySearchTree Class: Manages the operations and functionalities of the BST.

- **TreeNode Class**
  - Attributes:
    - “key”: Stores the key value.
    - “value”: Stores the corresponding value.
    - “left” and “right”: References to left and right child nodes.
    - “parent”: Optionally, a reference to the parent node.

- **BinarySearchTree Class**
  - Attributes:
    - “size”
    - “root”
  - Some of the methods:
    - put(key, value): Insert a new (key, value) pair into the BST.
    - get(key): Retrieve the value associated with the given key.
    - delete(key): Remove the (key, value) pair from the BST.
    - getSize(): Get the count of key-value pairs in the BST.

Example: Inserting a Key-Value Pair Using _add() Method

```python
def put(self, key, value):
    if not self.__root:  # Root node is empty
        self.__root = TreeNode(key, value)  # Create a new tree node as the root
        self.__size += 1  # Increment the cached size
    else:
        self._add(key, value, self.__root)  # Call the helper method for insertion
```

- Functionality:
  - Determines the placement of the new node based on key comparisons.
  - Handles scenarios for left and right node insertions.

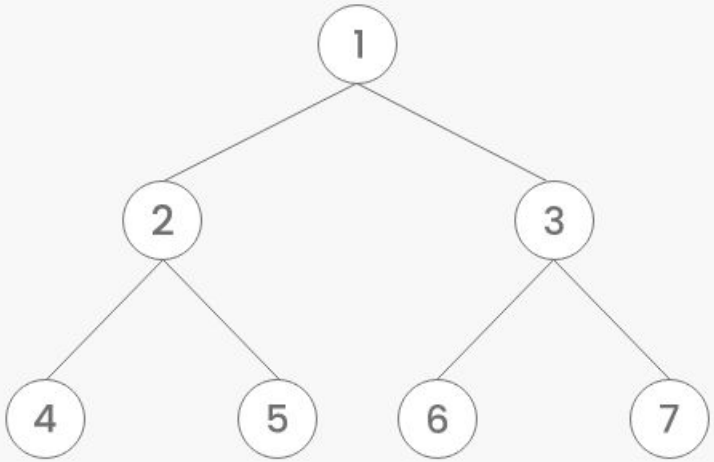
```python
def _add(self, key, val, currentNode):
    # Adds a (key, val) pair at a given node in the BST

    if key < currentNode.getKey():
        if not currentNode.getLeft():  # Left node doesn't exist, create a node
            currentNode.setLeft(TreeNode(key, val, parent=currentNode))
            self.__size += 1  # Increment the size
        else:
            self._add(key, val, currentNode.getLeft())  # Add to the left
    elif key == currentNode.getKey():  # Key already exists
        currentNode.setValue(val)  # Update value
    else:
        if not currentNode.getRight():  # Right node doesn't exist, create a node
            currentNode.setRight(TreeNode(key, val, parent=currentNode))
            self.__size += 1  # Increment the size
        else:
            self._add(key, val, currentNode.getRight())  # Add to the right
```

Binary Tree Traversal

Traversal: The process of visiting and processing each node in a tree data structure.

1. **Preorder Traversal**
   - Sequence: Root, Left, Right.
   - Purpose: Suitable for creating a copy of a tree, prefix notation in expressions.
2. **Inorder Traversal**
   - Sequence: Left, Root, Right.
   - Purpose: For BST (Binary Search Tree), outputs nodes in sorted order, expression evaluation in infix notation.
3. **Postorder Traversal**
   - Sequence: Left, Right, Root.
   - Purpose: Deleting the tree, postfix notation in expressions, obtaining the postfix form of an expression.
4. **Levelorder Traversal**
   - Sequence: Level by level.
   - Purpose: Used in printing nodes at different levels, constructing a tree, level-by-level search.

Binary Tree Traversal



Inorder Traversal

4	2	5	1	6	3	7

Preorder Traversal

1	2	4	5	3	6	7

Postorder Traversal

4	5	2	6	7	3	1

- What is the levelorder traversal?

Reconstructing a Binary Tree Using Inorder and Preorder Traversals

- Method: buildTree(inOrder, preOrder)
  - This is a recursive function.

- Hints for Implementation:
  - **Root Node**: First element in preorder traversal is the root.
  - **Inorder Split**: Root value divides inorder traversal into left and right subtrees.
  - **Subtree Building**: Find start and end indices for each subtree. Recursively build left and right subtrees.