Binary Tree

—— A binary tree made of binary trees

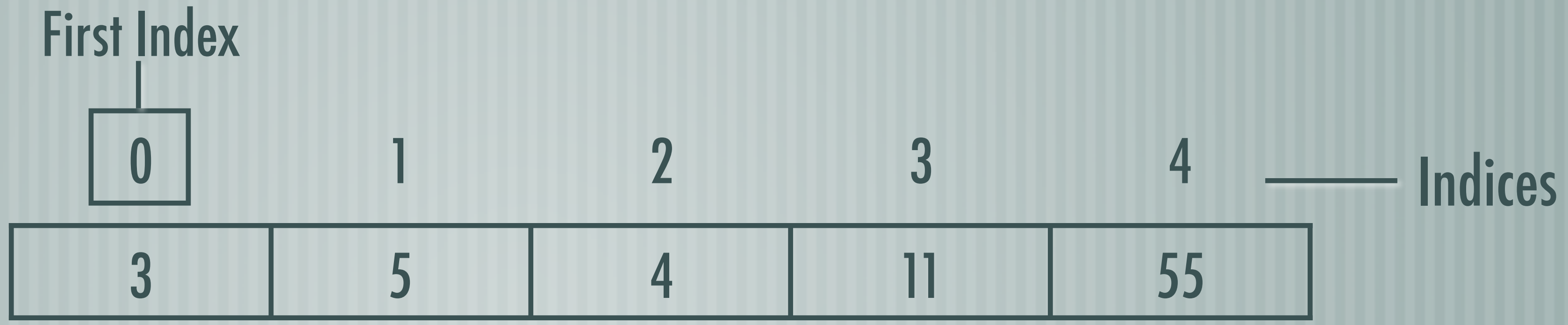
Author: Yun Tian (Tony) Ph.D.
Department of Computer Science
Eastern Washington University

Outline

- Tree Definition and Terminology
- Binary Tree and Binary Search Tree
- Conclusion

Before Introducing Tree

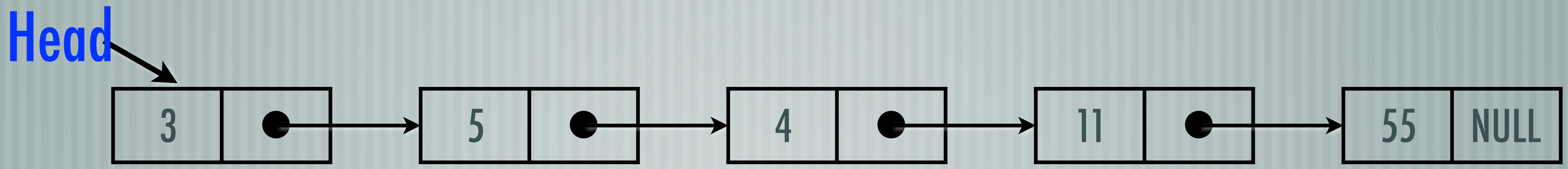
- Arrays: stores a sequence of data elements in a group of contiguous memory locations.
  - A linear structure, i.e. a sequence of data elements.
  - Adjacent elements in array are stored in contiguous memory locations.
  - We can access arbitrary element with its numerical index. For



0	1	2	3	4
3	5	4	11	55

Before Introducing Tree

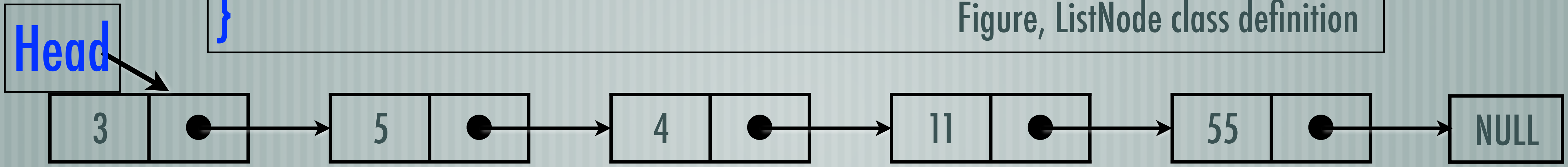
- Linked list: each node consists of a datum and a reference (or a link) to the next node in the sequence.
  - A linear structure, i.e. a sequence of data elements.
  - Adjacent elements are NOT necessarily stored in contiguous memory locations.
  - To access one element, you have to traverse sequentially from head.



Before Introducing Tree

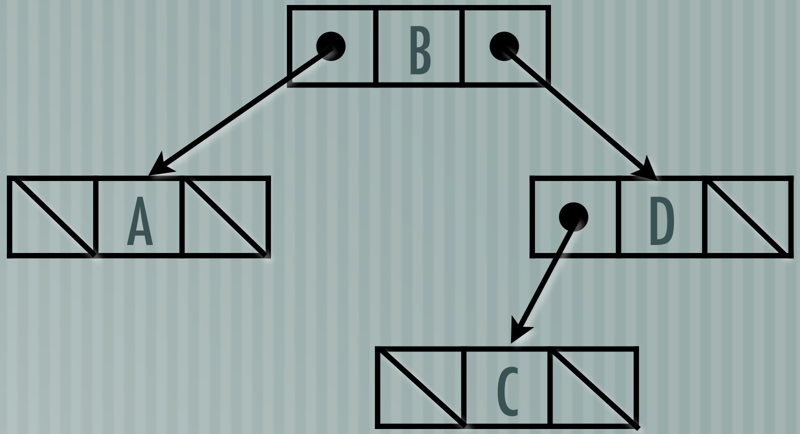
```
public class ListNode
{
    private Object  data;
    private ListNode   next;

    public ListNode(Object  data, Node  next)
    {
        this.data = data;
        this.next = next;
    }
    // other methods
}
```

Figure, ListNode class definition



Tree Data Structure

- A tree is a non-linear, two-dimensional data structure.
- A tree is a linked structure. Each node contains data and two or more links.
- In the figure on the right, each tree node contain two links, though one or both may be NULL. We call this a ***Binary Tree***.
- The back slash \ in box means NULL.



Figure, A graphical representation of a tree

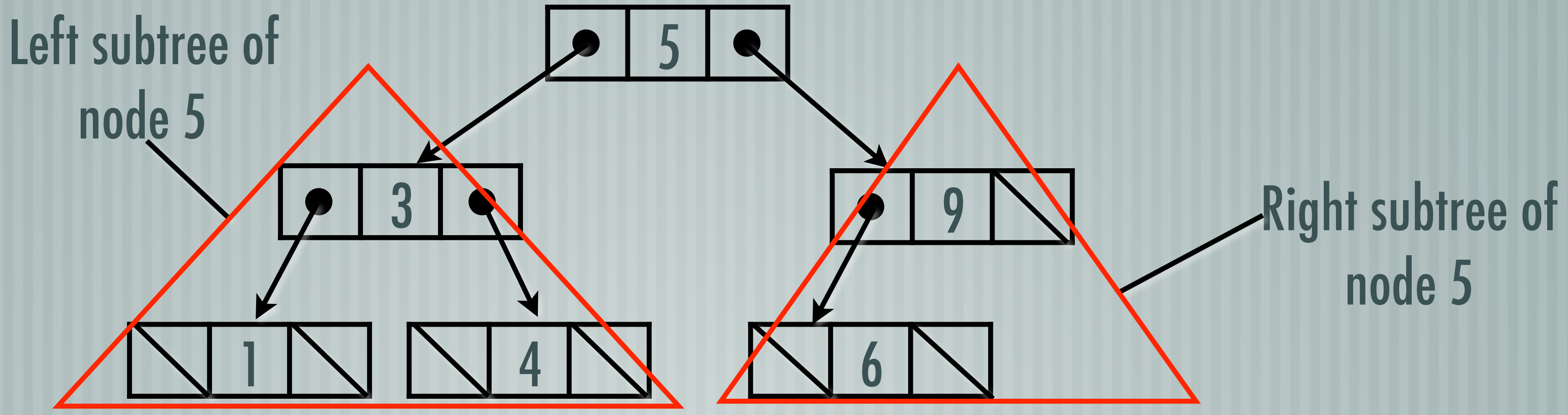
Why Tree Structure?

- Game programming and Artificial Intelligence
- Files system
- Representing arithmetic expressions
- Data indexing in database management system
- Huffman coding in data compression
- Real world problems, like modeling a river system.

Recursive Definition

Binary Tree:

- a binary tree is either empty, [base case]
- or is made of a single node whose left and right links each points to a binary tree. [recursive definition]



Binary Trees

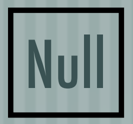


Figure a, empty tree

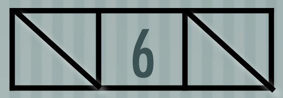


Figure b, tree with one node

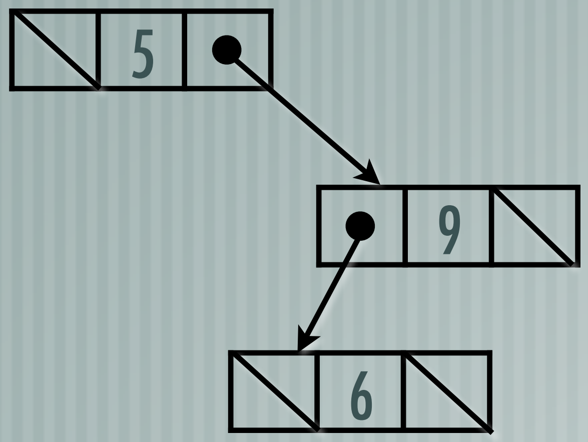


Figure c, binary tree that grows in one direction

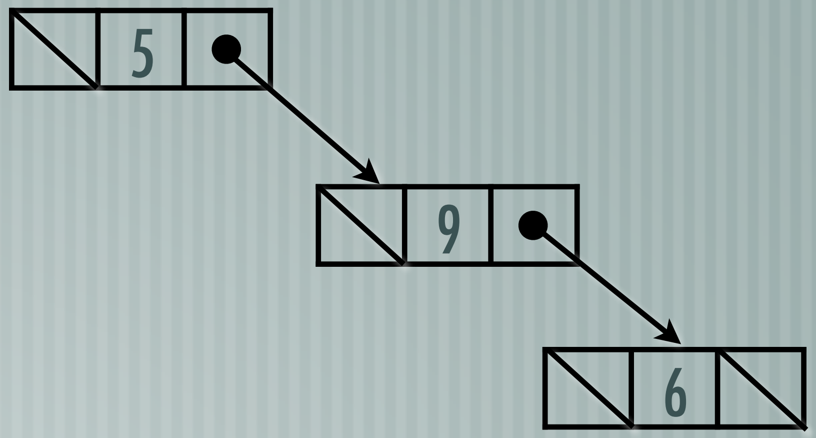
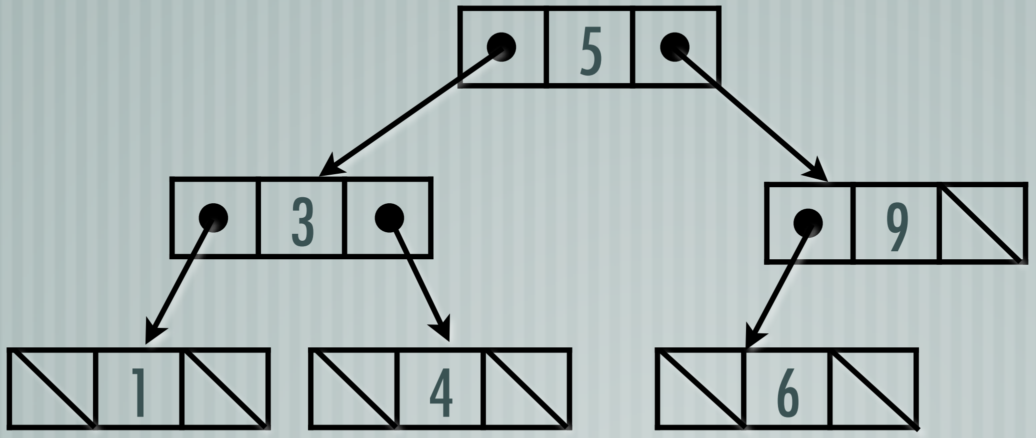


Figure d, binary tree that grows in one direction

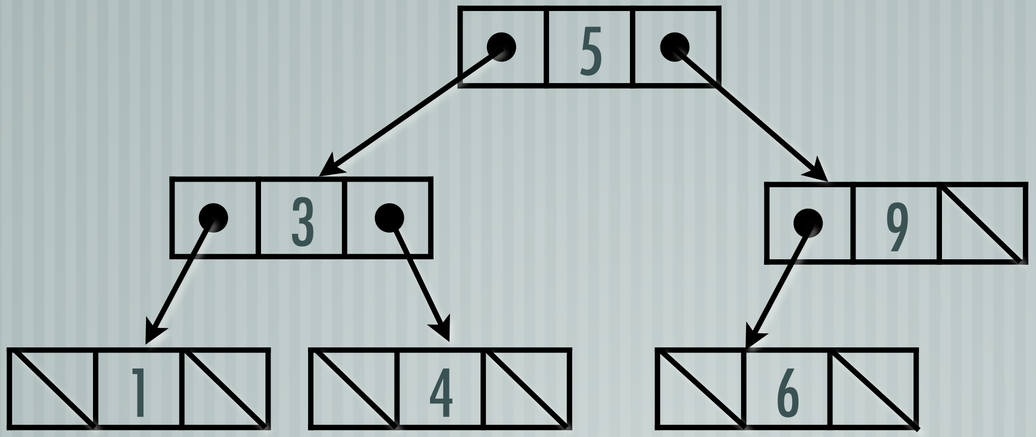
Terminologies in Binary Tree

- Node: an element in a tree stores data and references to other nodes.
- Root node: the topmost node in a tree without parent, node 5 is the root node in the figure below.
- Left/right child: the tree node that left/right link points to. The left child of node 5 is Node 3
- Parent: the node directly above another node. The parent of node 1 is Node 3



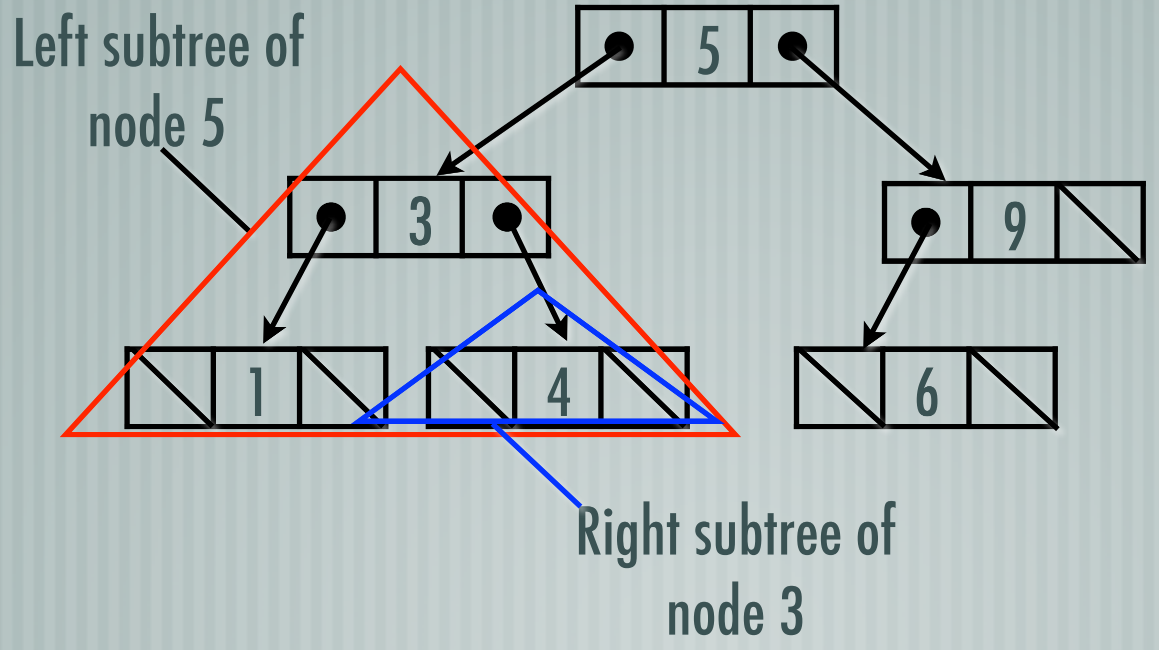
Terminologies in Binary Tree

- Internal node: node with at least one child (5, 3, 9)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors. depth of leaf node 1 is 2.
- Height of a tree: maximum depth of any node, which is 2 in the tree below. (may have different definition in other textbook)
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants



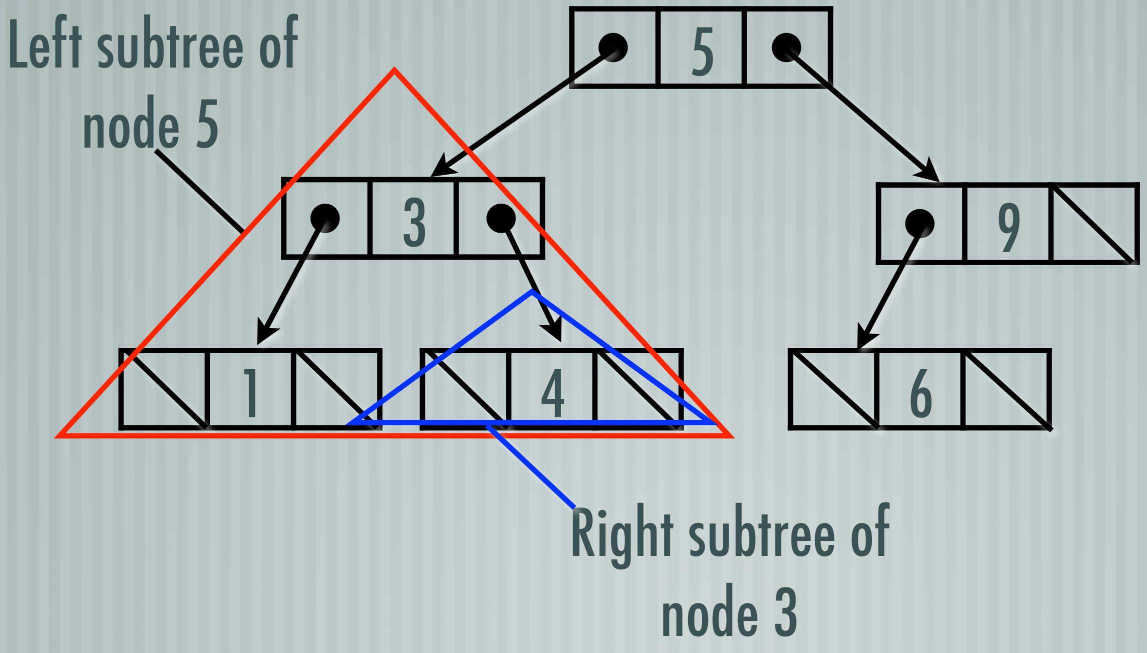
Terminologies in Binary Tree

- Left subtree: a subset of a tree rooted at a node pointed by the left link.
- Right subtree: a subset of a tree rooted at a node pointed by the right link.
- Leaf node: a node with no children.
- Levels: root node 5 is at level 0, and node 6 is at level 2.



Application – Binary Search Tree

- Binary Search Tree (BST) is a special type of Binary Tree:
  - Values in any left subtree are less than the value in its parent node.
  - Values in any right subtree are greater than or equal to the value in its parent node.



BST Implementation

```
public class BinarySearchTree {
    // Root node pointer. Will be null for an empty tree.
    private Node root;

    private class Node {
        Node left;
        Node right;
        int data; //To be general, the type should be Comparable

        public Node(int newData) {
            left = null;
            right = null;
            data = newData;
        }
    }

    /**
         * Creates an empty binary tree -- a null root pointer.
     */
    public BinarySearchTree() {
        root = null;
    }
```

Application — Binary Search Tree

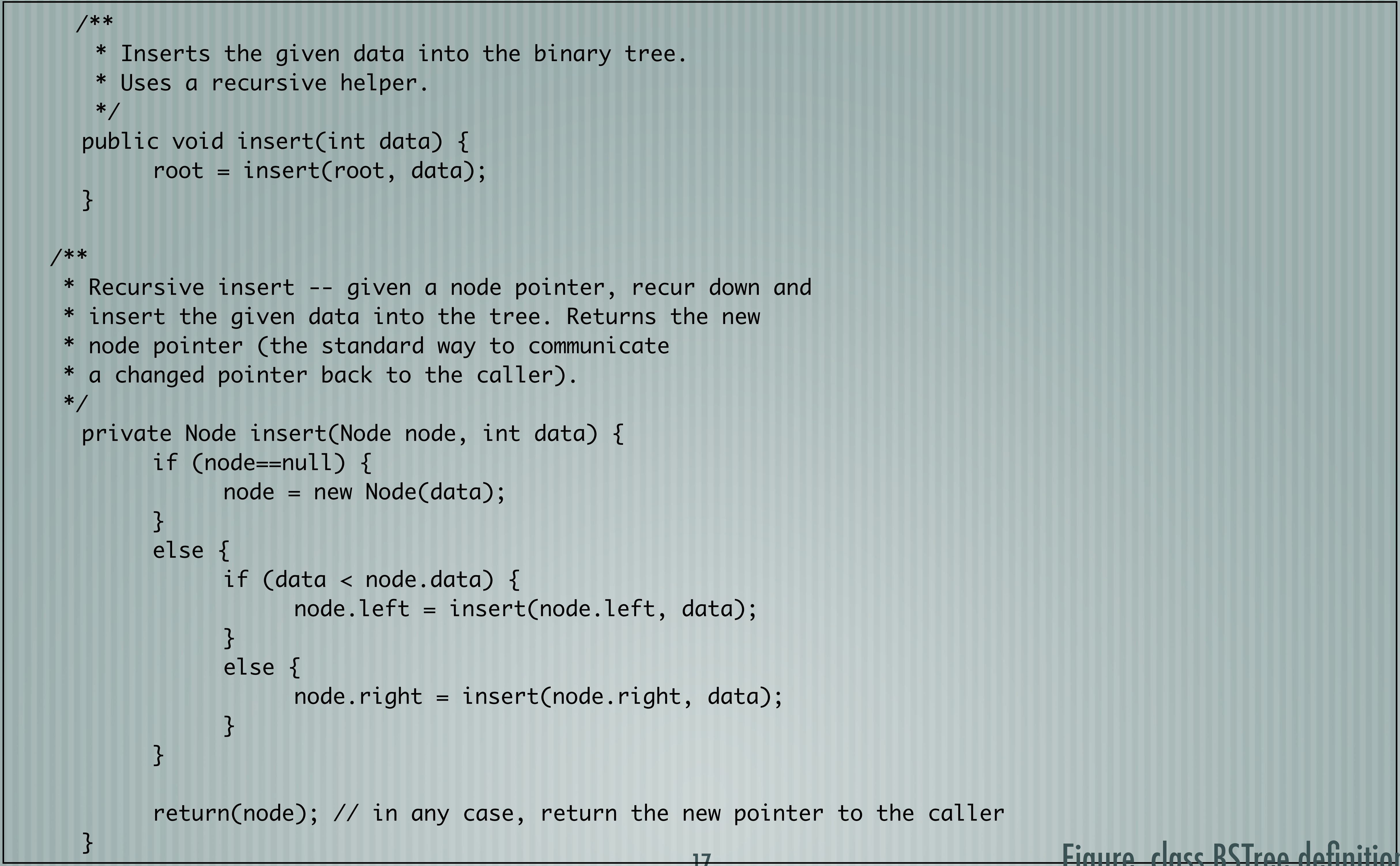
Some Typical Operations on a BST

- Insert a value into the tree.
- Search for a number in the tree, such as minimum, maximum etc.
- Delete a value, if it is found.
- Traversal: visit all values in the tree.
- How many nodes or leaf nodes in the tree?
- Compute the depth of the tree
- Output all paths in the tree

Insert a Value into BST

- Steps to insert data into BST
  - Step 1, if current node is null, make a new node that contains the data to be inserted, and assign the new node as current node.
  - Step 2, if insert value is less than the current node value, the value will be inserted into the left subtree of the current node.
  - Step 3, If insert value is greater than the current node value, the value will be inserted into the right subtree of the current node.

BST Implementation

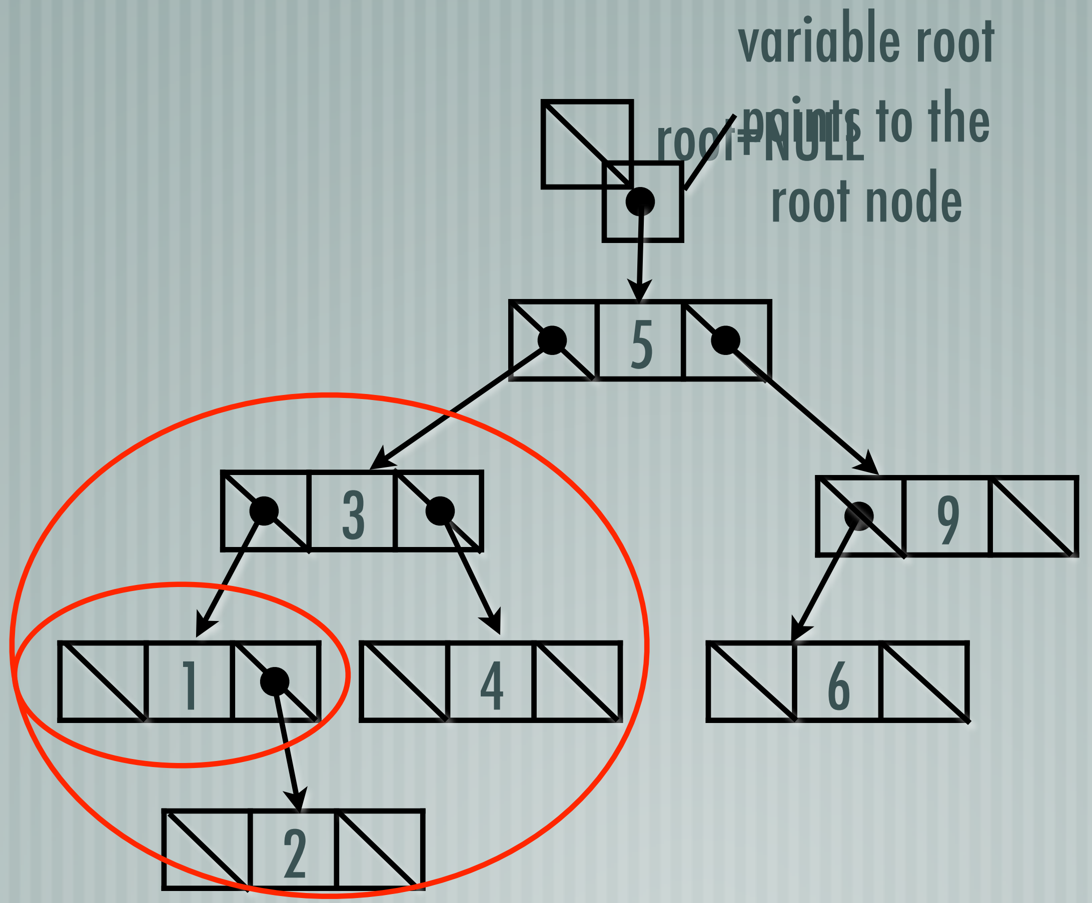


```
  /**
   * Inserts the given data into the binary tree.
   * Uses a recursive helper.
   */
  public void insert(int data) {
        root = insert(root, data);
  }

/**
 * Recursive insert -- given a node pointer, recur down and
 * insert the given data into the tree. Returns the new
 * node pointer (the standard way to communicate
 * a changed pointer back to the caller).
 */
  private Node insert(Node node, int data) {
        if (node==null) {
              node = new Node(data);
        }
        else {
              if (data < node.data) {
                    node.left = insert(node.left, data);
              }
              else {
                    node.right = insert(node.right, data);
              }
        }

        return(node); // in any case, return the new pointer to the caller
  }
```

Figure, class BSTree definition

Insert Data into BST

- Sequentially inserting a sequence of numbers into a BST: 5, 3, 9, 1, 4, 6, 2.

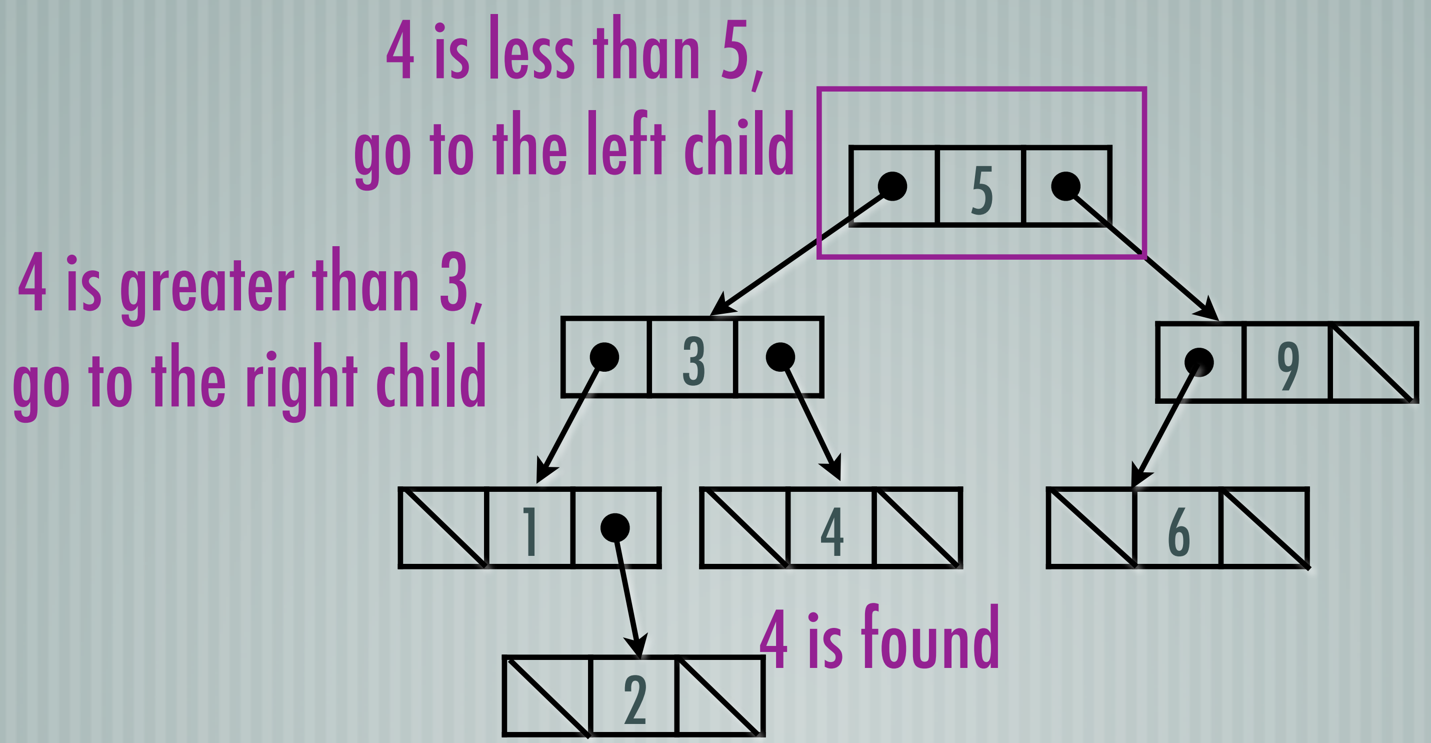


Search a Value in BST

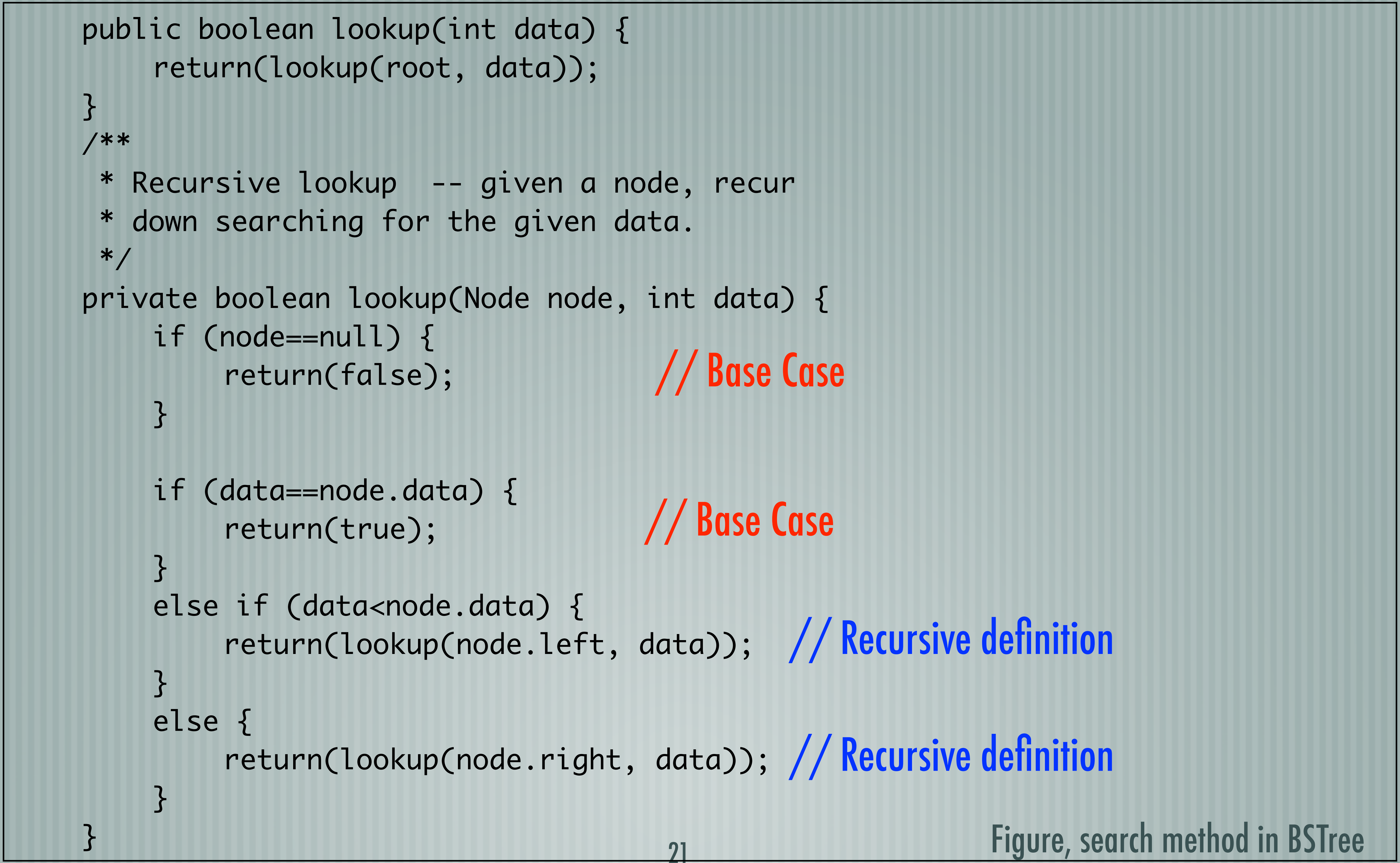
- Three cases when search a value in BST,
  - Case 1, if current node is null, (meaning the current subtree is empty) return false. // Base Case
  - Case 2, check whether value in current node and searched value are equal. If so, return true(means found). Otherwise, // Base Case
  - Case 3, if searched value is less than the node's value // Recursive definition
    - return whether we can find the value in its left subtree.
    - Case 3, if searched value is greater than the node's value. // Recursive definition
    - return whether we can find the value in its right subtree.

Search a Value in BST

- The current node is shown in purple box.
- The current node is changing as we search down the tree.
- The current node is always the root node of the subtree we are currently considering.



BST Search Implementation



```
public boolean lookup(int data) {
    return(lookup(root, data));
}
/**
 * Recursive lookup  -- given a node, recur
 * down searching for the given data.
 */
private boolean lookup(Node node, int data) {
    if (node==null) {
        return(false);              // Base Case
    }

    if (data==node.data) {
        return(true);               // Base Case
    }
    else if (data<node.data) {
        return(lookup(node.left, data));  // Recursive definition
    }
    else {
        return(lookup(node.right, data)); // Recursive definition
    }
}
```

Figure, search method in BSTree

Traverse a general tree

- Any process for visiting the nodes in some order is called a **traversal**.
- Any traversal that lists every node in the tree exactly once is called an **enumeration** of the tree’s nodes.
- NB: an empty node (tree) represented by Java’s null (object) value

Traverse a general tree

- Preorder traversal: Visit each node before visiting its children.
- Postorder traversal: Visit each node after visiting its children.
- Inorder traversal: Visit the left subtree, then the node, then the right subtree.

Traversal Implementation

```
public void printTree() {  //inorder traversal
    printTree(root);
    System.out.println();
}
private void printTree(Node node) {
    if (node == null) return;     // Base Case

    // left, node itself, right
    printTree(node.left);       // Recursive definition
    System.out.print(node.data + "  ");
    printTree(node.right);      // Recursive definition
}
```

Note: These algorithms are also applicable to general binary trees, but not limited to BST.

Traversal Implementation

```
public void printTree() {  //preorder traversal
    printTree(root);
    System.out.println();
}
private void printTree(Node node) {
    if (node == null) return;         // Base Case

     System.out.print(node.data + "  ");
    // node itself, left, right
    printTree(node.left);            // Recursive definition
    printTree(node.right);           // Recursive definition
}
```

Note: These algorithms are also applicable to general binary trees, but not limited to BST.

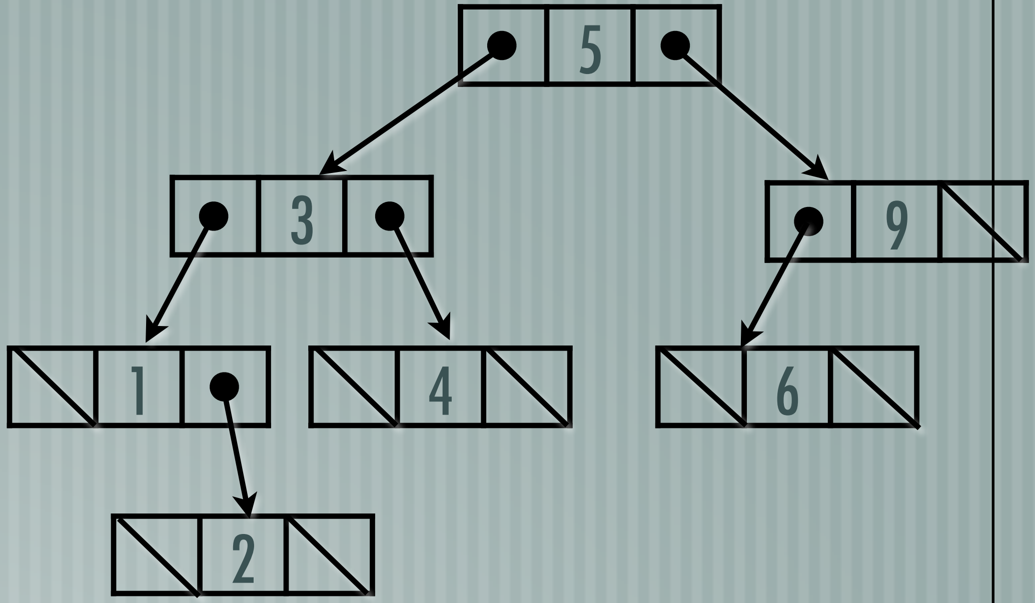
Traversal Implementation

```
public void printTree() {  //postorder traversal
     printTree(root);
     System.out.println();
}
private void printTree(Node node) {
     if (node == null) return;      // Base Case

     // left, right,  then node itself
     printTree(node.left);        // Recursive definition
     printTree(node.right);       // Recursive definition
    System.out.print(node.data + "  ");

}
```

Note: These algorithms are also applicable to general binary trees, but not limited to BST.

Traversal Implementation

```
public void printTree() {  //preorder traversal
    printTree(root);
    System.out.println();
}
private void printTree(Node node) {
    if (node == null) return;

     System.out.print(node.data + "  ");
    // node itself, left, right
    printTree(node.left);
    printTree(node.right);
}
```



// If we run the pre-order traversal code on the tree in the right figure,
what you will see on the standard output?

// If we run the post-order traversal code on the tree in the right figure,
what you will see on the standard output then?

Conclusion

- Recursion is more about problem solving skills.
- Recursion is closely related to algorithm design, data structures.
- Tree structure is recursive in nature, every node in a tree seen as the root node of the subtree rooted at that node.
- We learned Binary Search Tree and its search method.

Any questions?