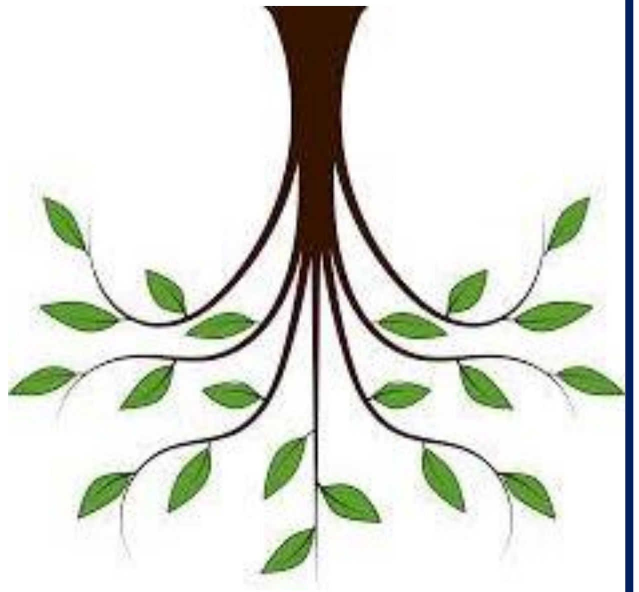
Part 2
About **Tree**



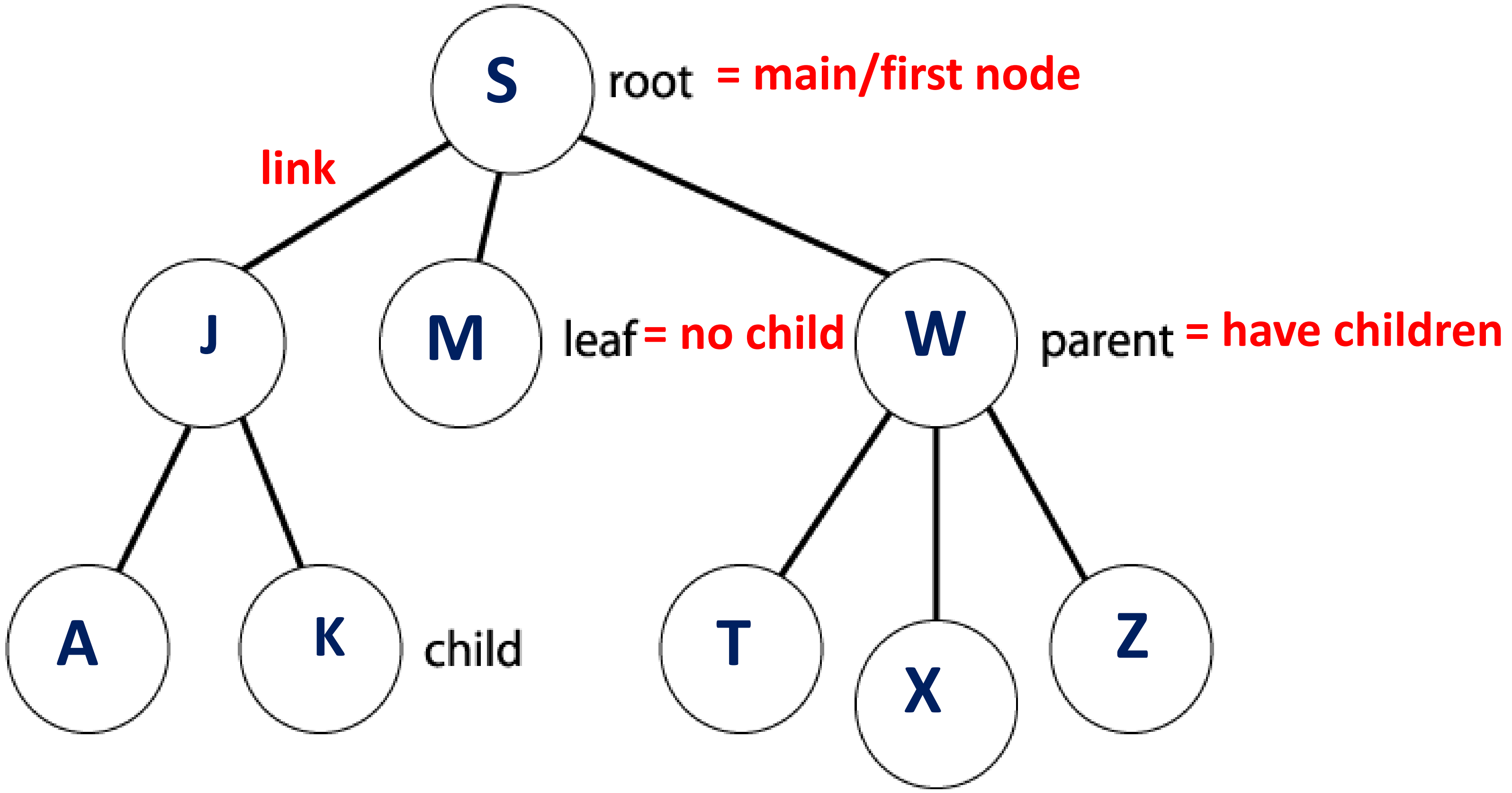
Tree

Binary Tree

Binary Search Tree

Tree Data Structure

Info about Tree: https://www.youtube.com/watch?v=qH6yxkw0u78

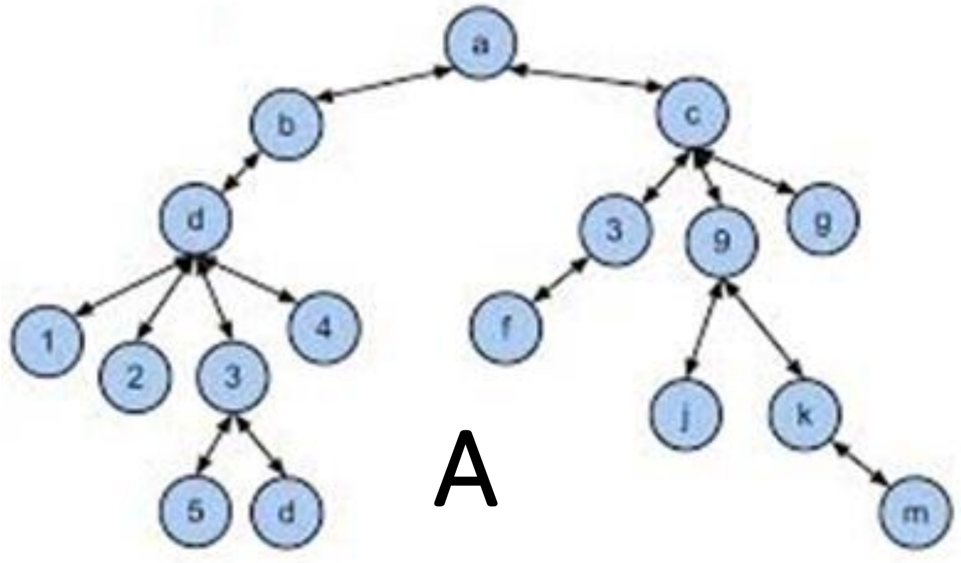


- A : leaf, child of J
- M ??
- J ??
- S ??
- X ??

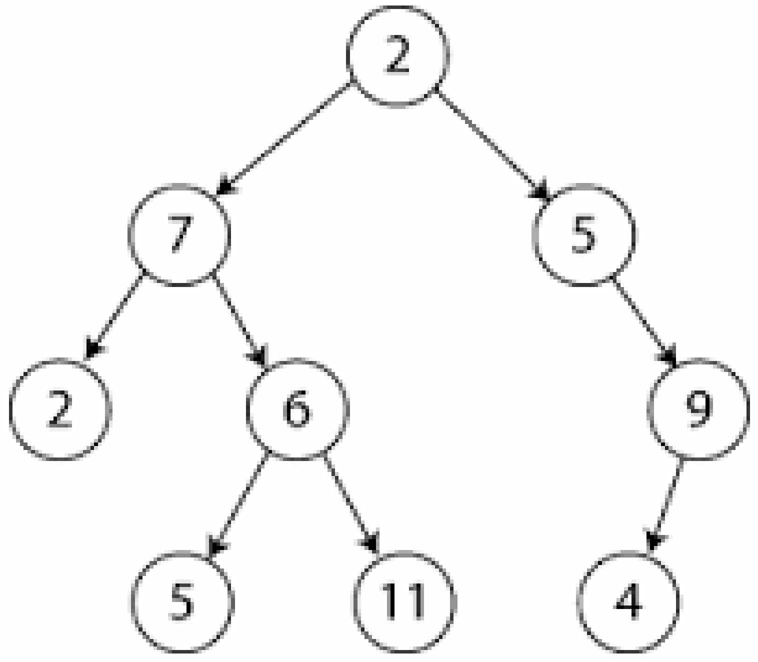
Definition:

Tree (T)	Binary Tree (BT)	Binary Search Tree (BST)
Non Linear Data Structure. Consist of nodes. Main/first node == Root. Each node connected to other node by a link Node can have children, children == Nodes		
Nodes in tree can have 0 or many children.	A tree whose elements have at most 2 children: left and right child.	Binary Search Tree is a node-based binary tree data structure which has the following properties: • The left subtree of a node contains only nodes with keys lesser than the node's key. • The right subtree of a node contains only nodes with keys greater than the node's key. • The left and right subtree each must also be a binary search tree.

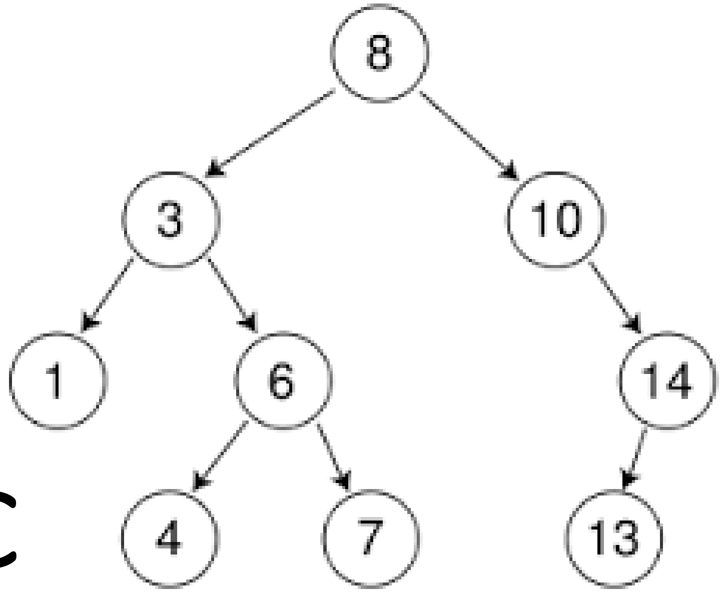
Tree
Binary Tree
Binary Search Tree



A

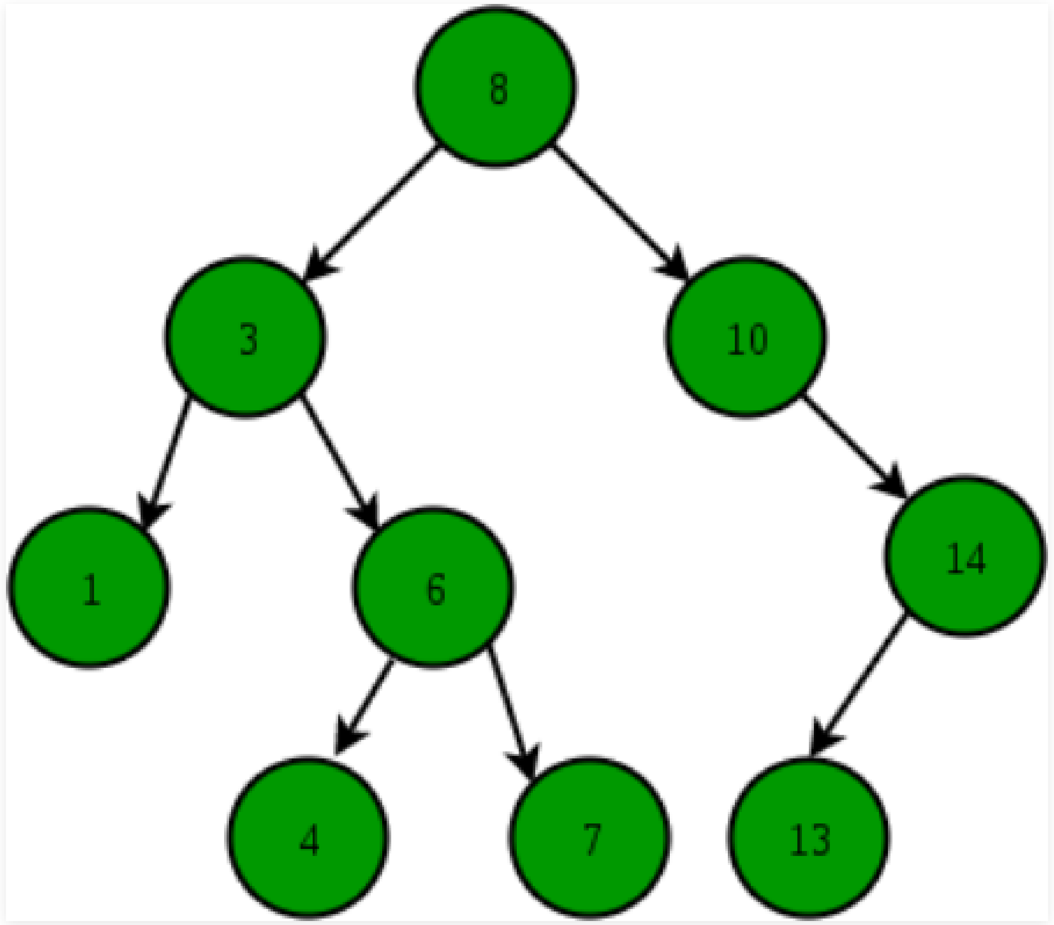


B



C

Tree, Binary Tree OR Binary Search Tree??

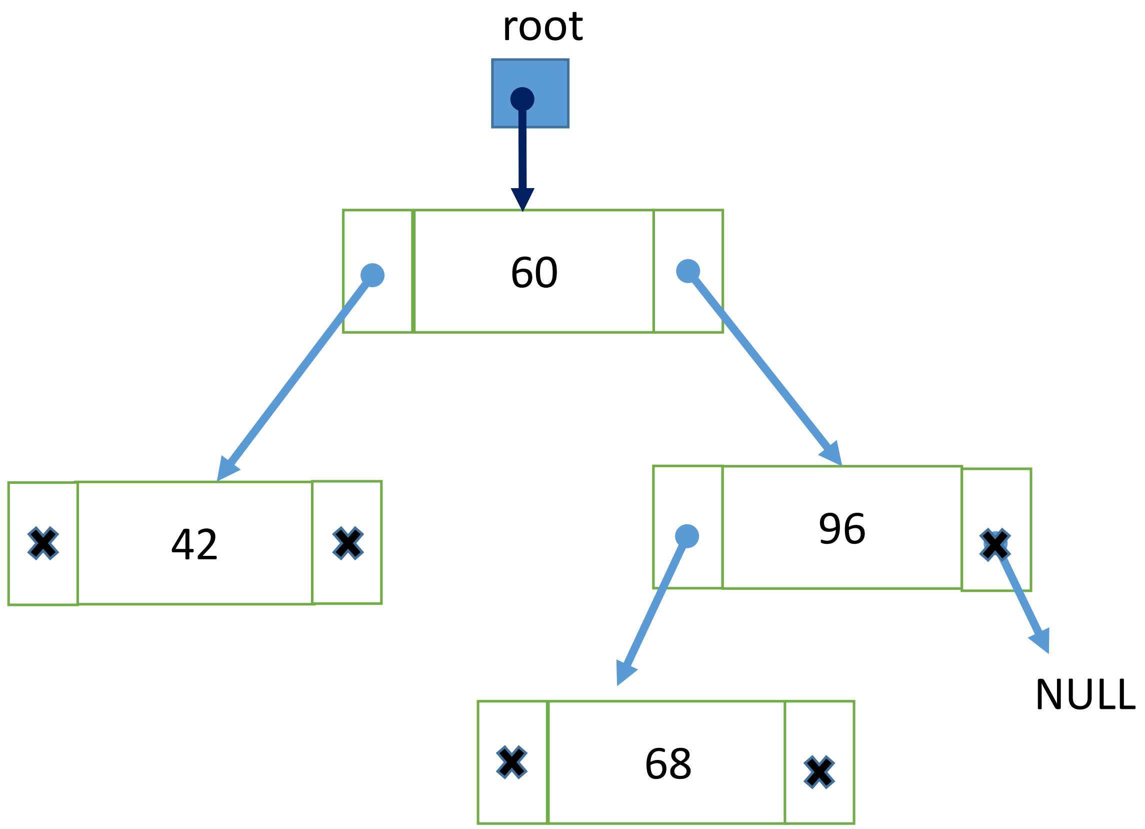


Part 3
About **Binary Search Tree**

Binary Search Tree (BST)

- **Binary Search Tree:**
  - Data/key of left child < its parent, AND
  - Data/key of right child > than its parent.

- **Define BST**:
  1. Pointer
  2. class

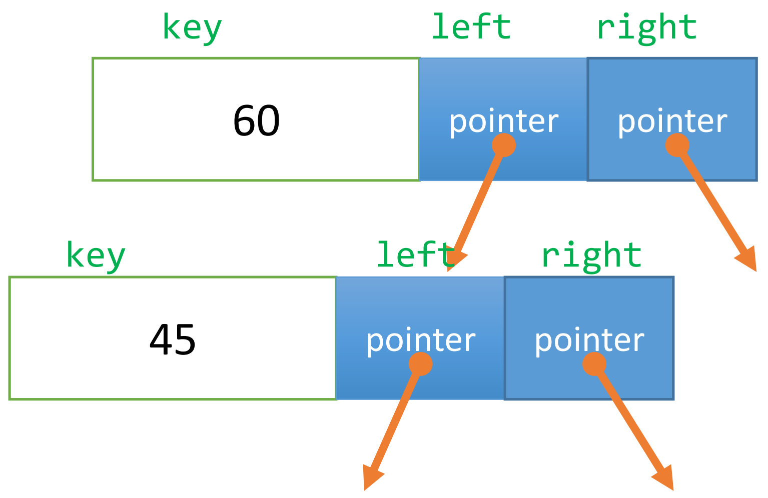


Class Node

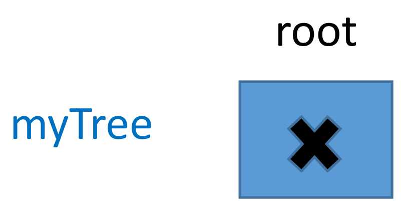
A BST is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL.

Every tree node contains following parts: **(1) Key**, **(2) Pointer to left child** and **(3) Pointer to right child**

```java
public class Node{
      int key;
      Node left, right;

      public Node(int item) {  // constructor
         key = item;
         left = right = null;
      }
}
```



```
class BinarySearchTree
{
      class Node {
          int key;
          Node left, right;
          public Node(int item) {
               key = item;
               left = right = null;
          }
      }
      Node root; // Root of BST
      BinarySearchTree() { // Constructor
           root = null;
      }
}

public static void main(String[] args) {

      BinarySearchTree myTree = new BinarySearchTree();

}
```



Common method for BST

1. **Insert Node in BST**
2. Delete a node
3. Search element in a tree
4. Traverse the tree, hence display the elements

Insert Node in BST

```
// This method mainly calls insertRec()
void insert(int key) {
    root = insertRec(root, key);
}
```

```
// A recursive function to insert a new key in BST
Node insertRec(Node root, int key) {

    // If the tree is empty, return a new node
    if (root == null) {
         root = new Node(key);
         return root;
     }

    // Otherwise, recur down the tree
    if (key < root.key)
         root.left = insertRec(root.left, key);
    else if (key > root.key)
         root.right = insertRec(root.right, key);

     // return the (unchanged) node pointer
                  return root;
 }
```

Recursive method

```
public static void main(String[] args) {

     BinarySearchTree tree = new BinarySearchTree();

     tree.insert(50);
     tree.insert(30);
     tree.insert(80);
     tree.insert(40);

}
```

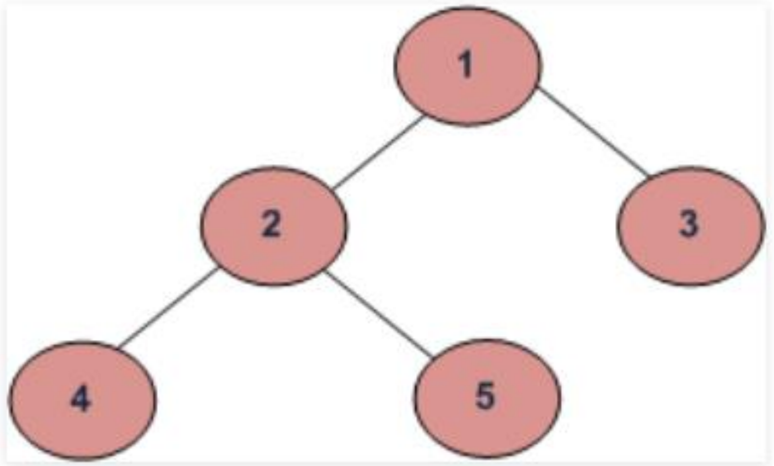
```
        50
   /          \
  30          80
     \
     40
```

Common method for BST

1. Insert Node in BST
2. **Traverse the tree, hence display the elements**
3. Search element in a tree
4. Delete a node

Tree Traversals

- Linear data structures (Array, Linked List, Queues, Stacks, etc) have only one logical way to traverse them.
- 3 ways of tree traversal:
  (a) Inorder (L**N**R) : 4 2 5 1 3
  (b) Preorder (**N**LR) : 1 2 4 5 3
  (c) Postorder (LR**N**) : 4 5 2 3 1



Example Tree

(1) InOrder Tranversal

```
void printInorder(Node node)
{
    if (node == null)
       return;

L   printInorder(node.left);
N   System.out.print(node.key + " ");
R   printInorder(node.right);
}
```

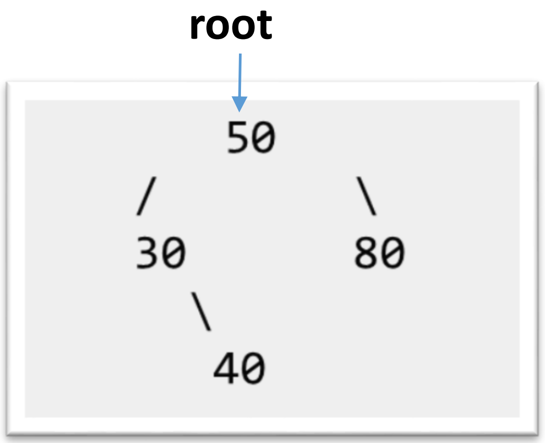
```
Algorithm Inorder(LNR)
    1.Traverse the left subtree
    2.Visit the root.
    3.Traverse the right subtree
```

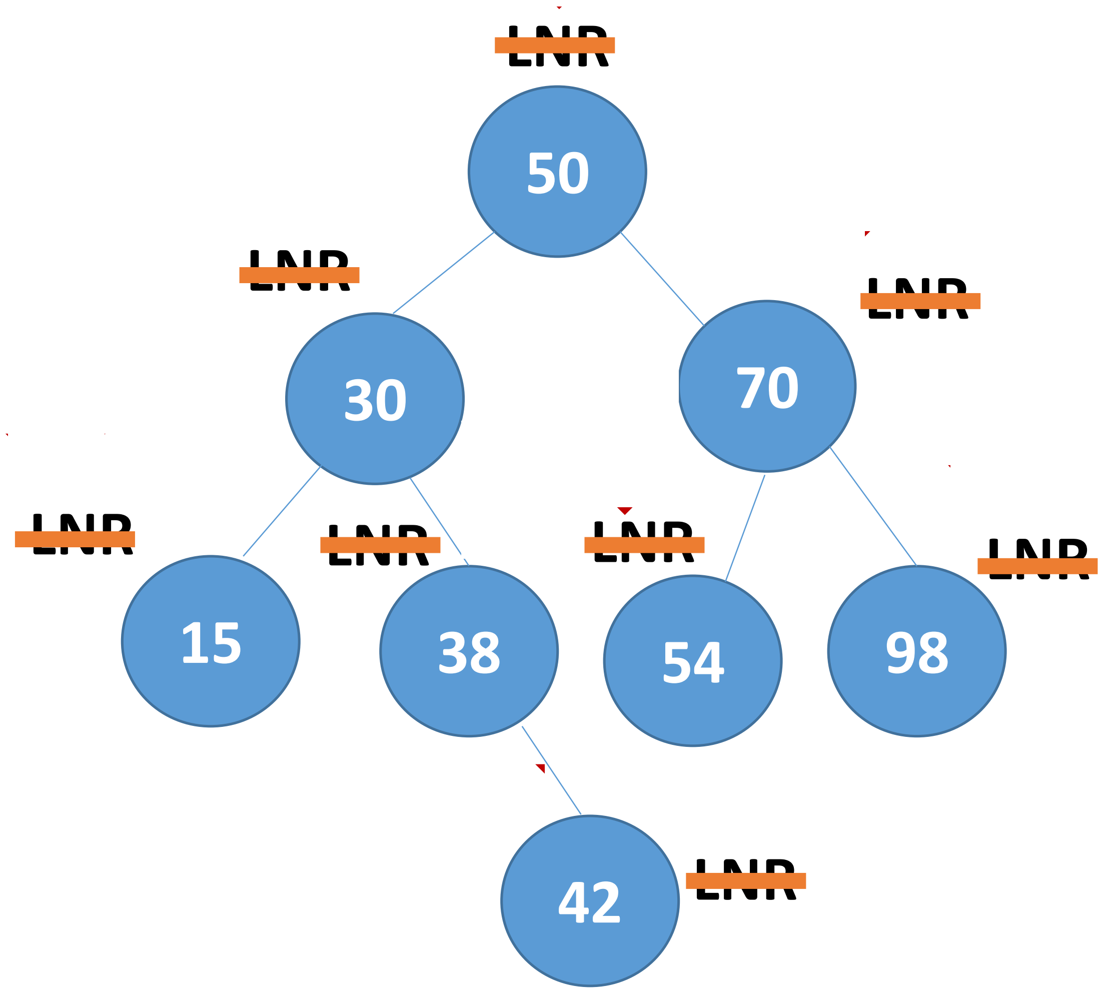
Recursive method

```
void inorder() // calls PrintInorder()
{
        PrintInorder(root);
}
```

```
void printInorder(Node node)
{
    if (node == null)
         return;

    printInorder(node.left);
    System.out.print(node.key + " ");
    printInorder(node.right);
}
```

```
public static void main(String[] args) {

     BinarySearchTree tree = new BinarySearchTree();

     tree.insert(50);
     tree.insert(30);
     tree.insert(80);
     tree.insert(40);

     System.out.println("Inorder traversal of the
                         given tree");
     tree.inorder();

}
```





Output: 15 30 38 42 50 54 70 98

Preorder Tranversal

```
void printPreorder(Node node)
{
    if (node == null)
       return;

 N  System.out.print(node.key + " ");
 L  printPreorder(node.left);
 R  printPreorder(node.right);
}
```

```
Algorithm Preorder (NLR)
    1. Visit the root.
    2. Traverse the left subtree
    3. Traverse the right subtree
```

Recursive method

Postorder Tranversal

```
void printPostorder(Node node)
{
    if (node == null)
       return;
  L printPostorder(node.left);
  R printPostorder(node.right);
  N System.out.print(node.key + " ");
}
```

```
Algorithm Postorder (LRN)
   1. Traverse the left subtree
   2. Traverse the right subtree
   3. Visit the root
```

Recursive method

Demo..
Insert and tree traversal

(Tree1)

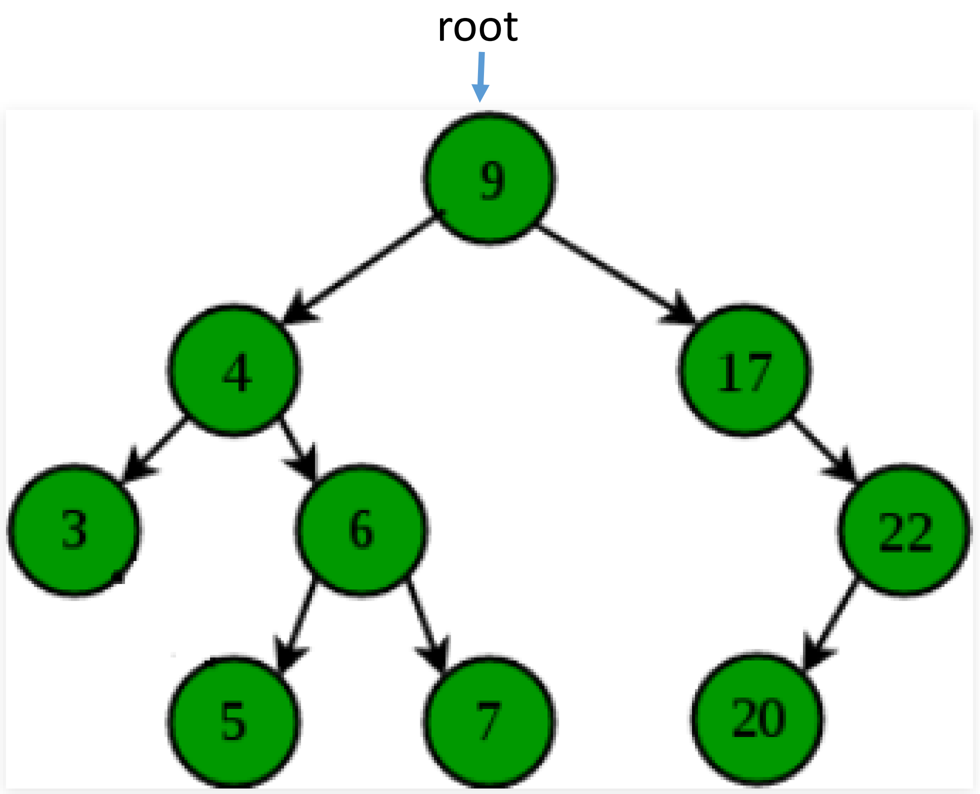
Common method for BST

1. Insert Node in BST
2. Traverse the tree, hence display the elements.
3. **Search element in a tree**
4. Delete a node

Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

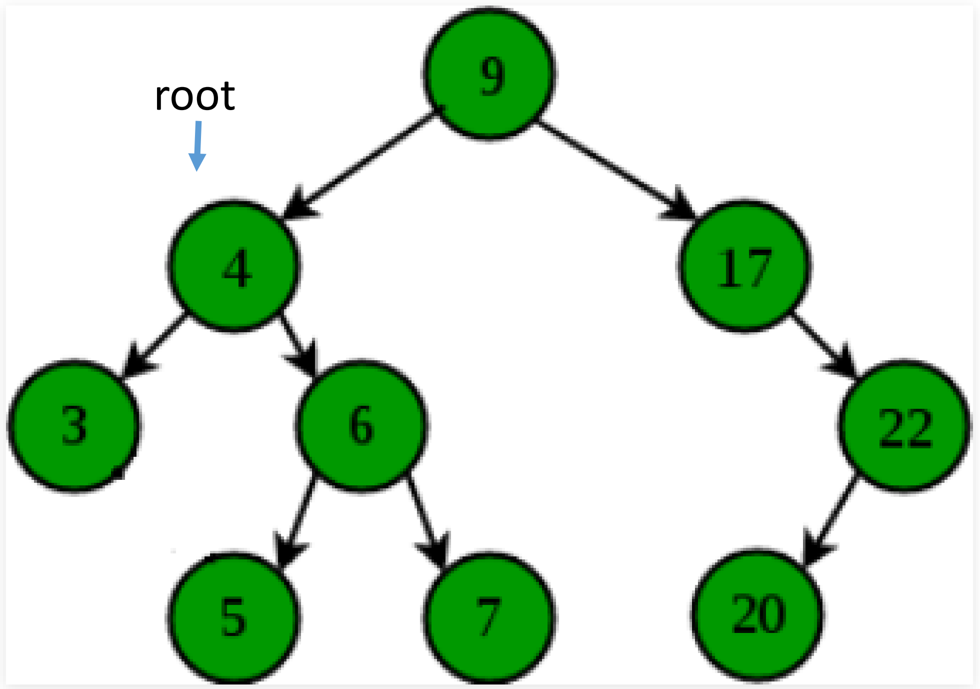
Search: 6



Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

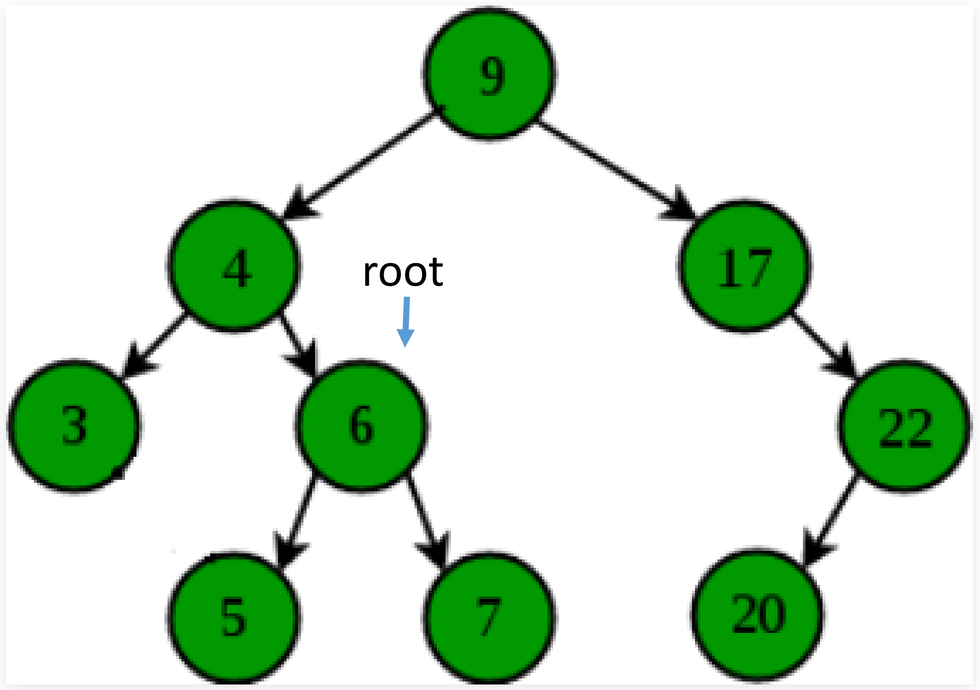
Search: 6



Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

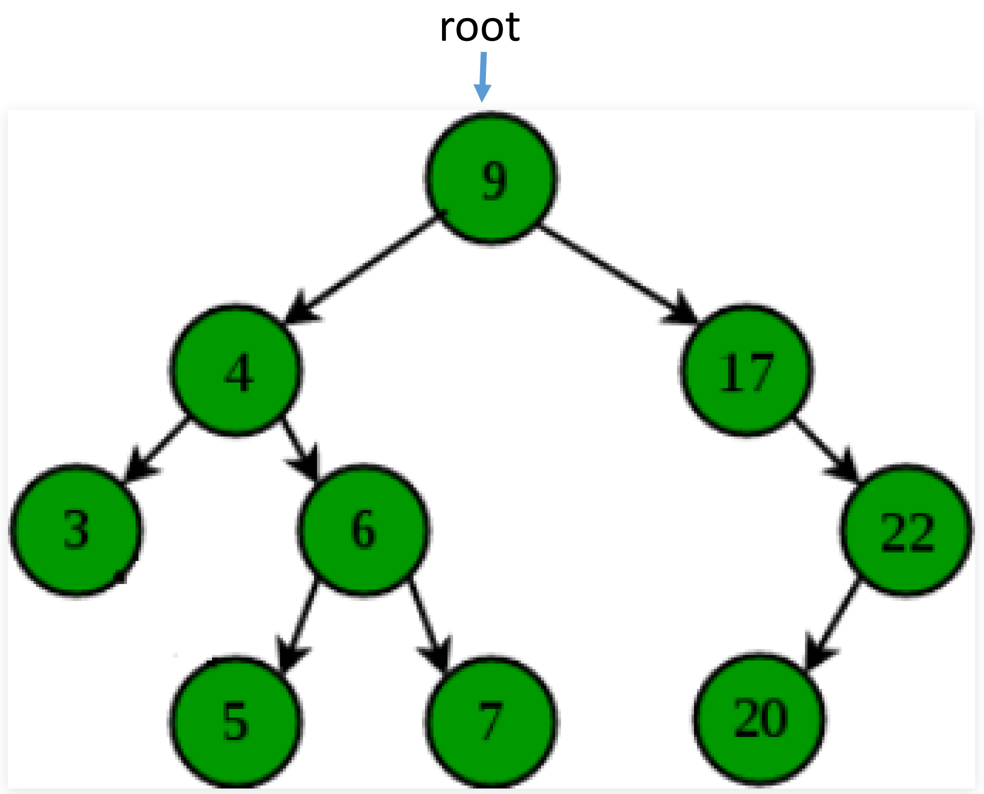
Example: Search 6 - return true (found)



Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

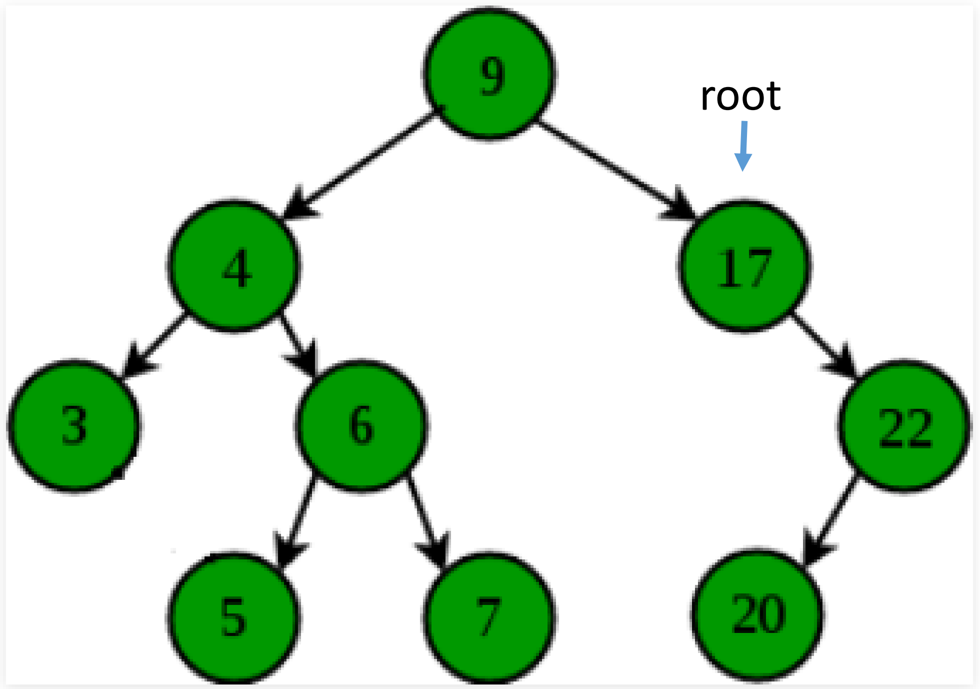
Search: 21



Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

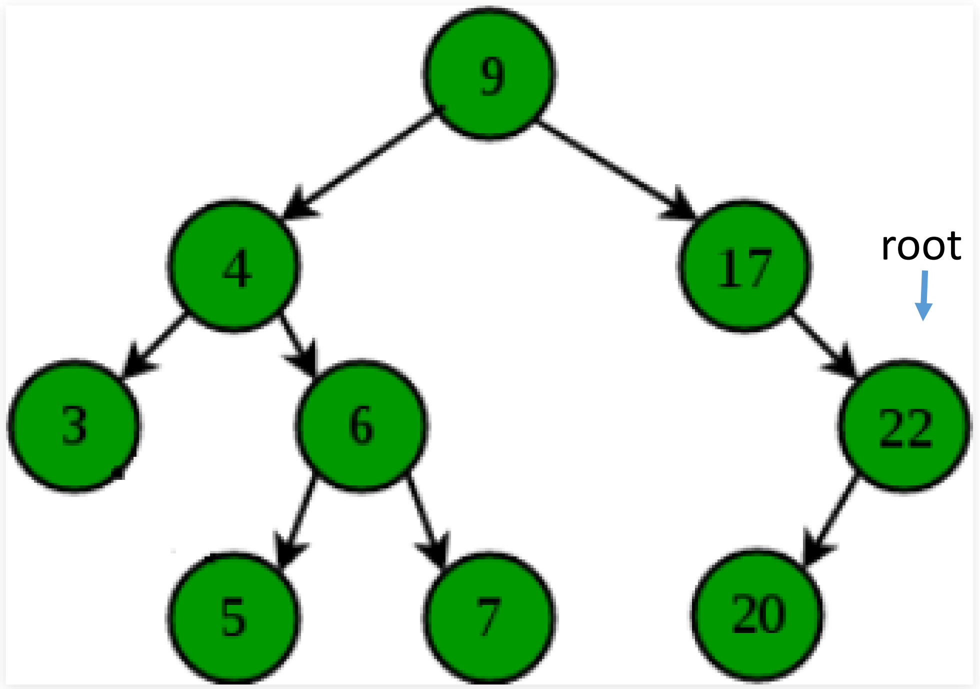
Search: 21



Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

Search 21

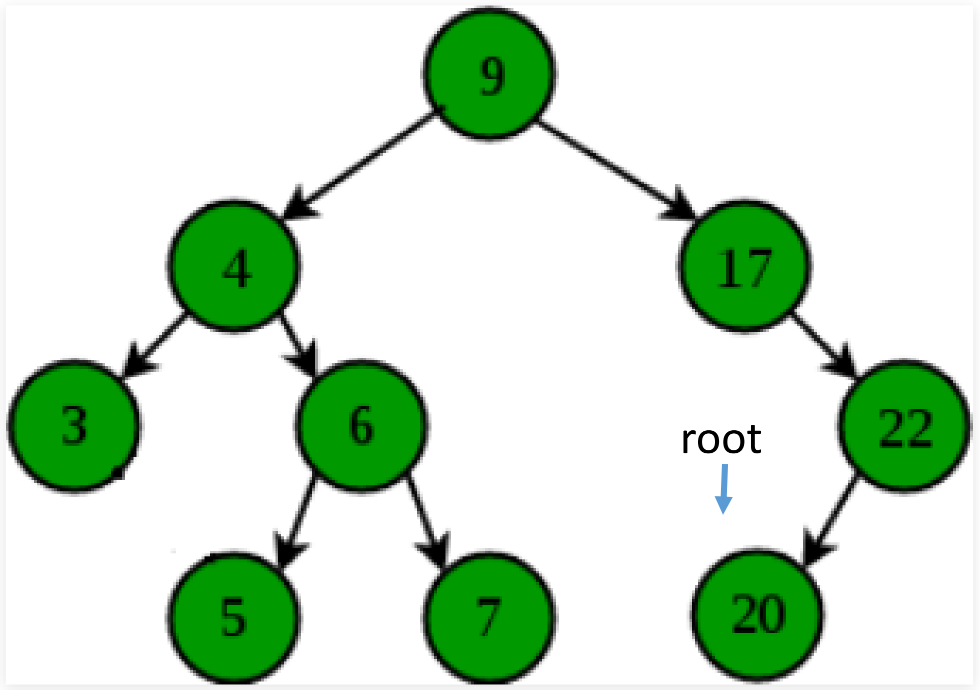


root

Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

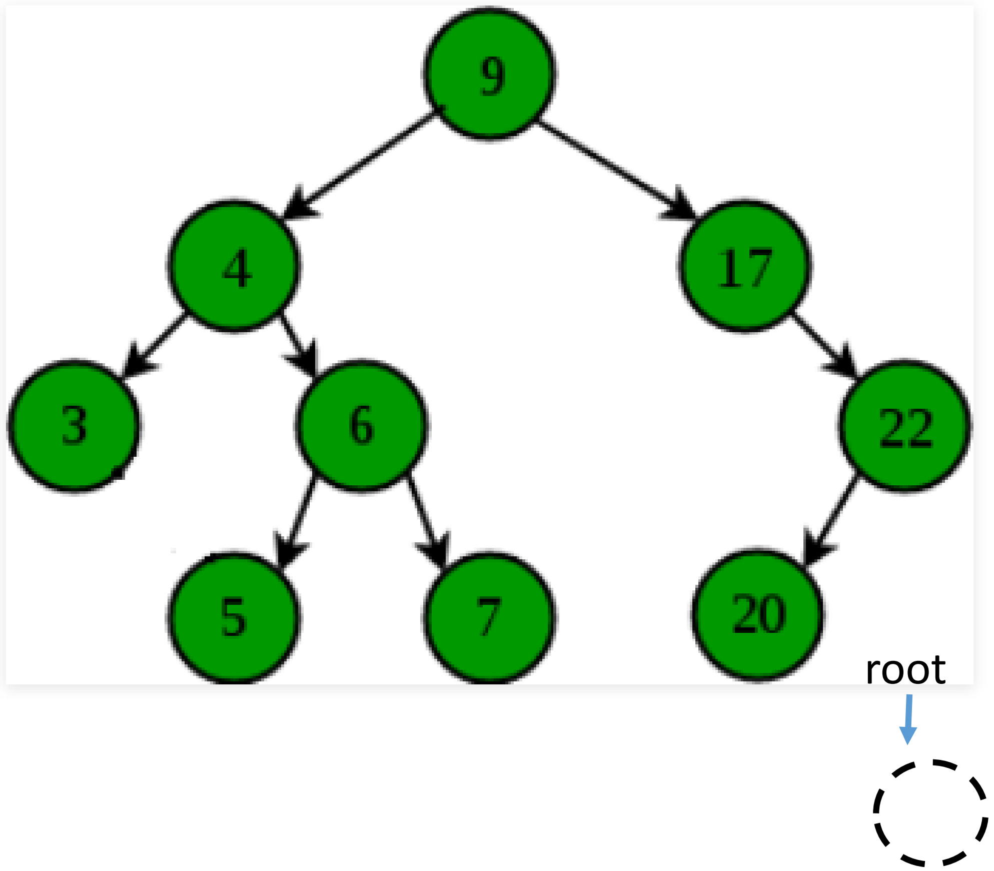
Search 21



Search element in Binary Search Tree

Input: the value/key to search
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

Search 21 – false (not found)



Algorithm & Method for searching a node:

(https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/)

Input: the value/key to search

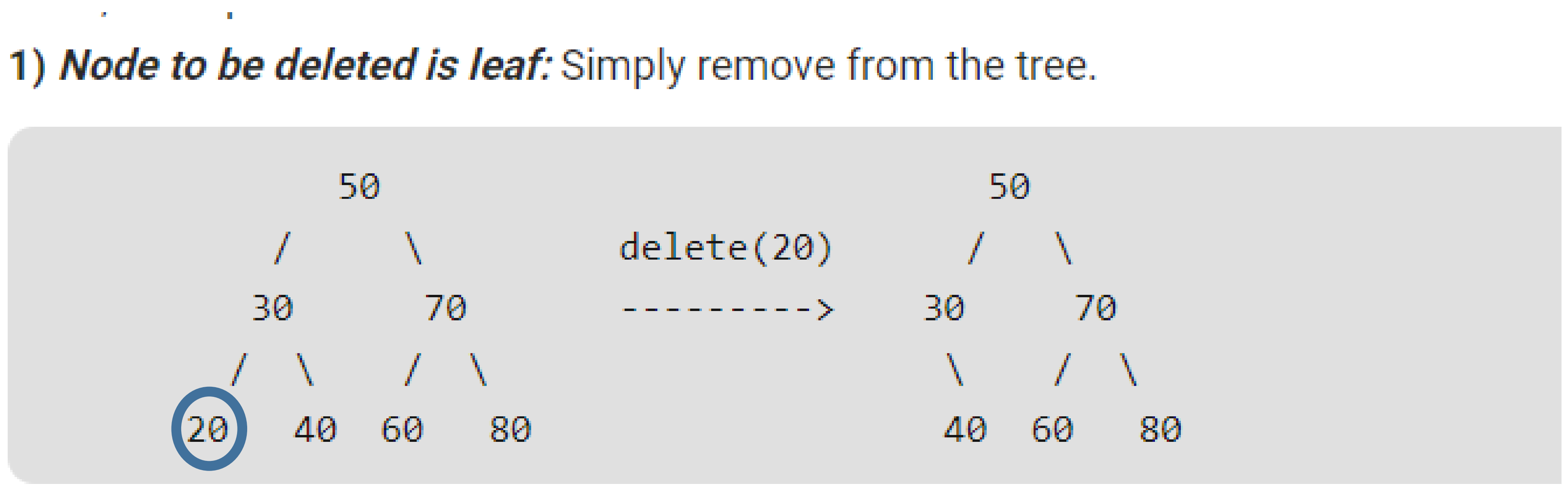
1. Start from root.
2. Compare the inserting element(key) with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.

```
// This method mainly calls searchRec()
public boolean searchRec (int key) {
        Node root1 = search(root, key);

        if (root1.key == key)
                return true;
        else  return false;
}

// A utility function to search a given key in BST
public Node search(Node root, int key)
{
        // Base Cases: root is null or key is present at root
        if (root==null || root.key==key)
                return root;
        // val is greater than root's key
        if (root.key > key)
                return search(root.left, key);
        // val is less than root's key
        return search(root.right, key);
}
```

Common method for BST

1. Insert Node in BST
2. Traverse the tree, hence display the elements.
3. Search element in a tree
4. **Delete a node**

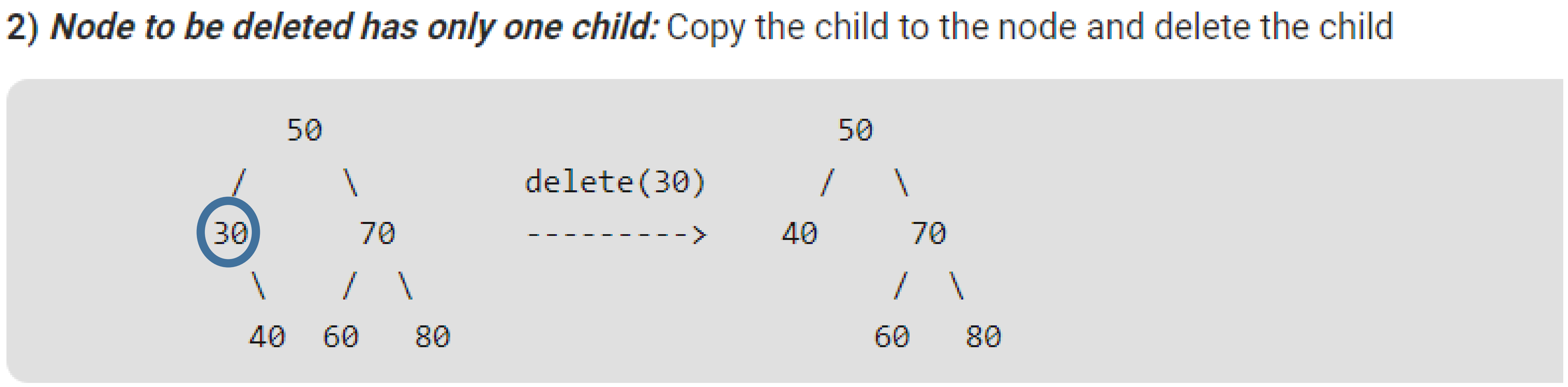
Delete a Node - leaf



1) ***Node to be deleted is leaf:*** Simply remove from the tree.

```
              50                               50
           /     \         delete(20)        /   \
          30      70       --------->      30     70 
         /  \    /  \                        \    /  \ 
       20   40  60   80                       40  60   80
```

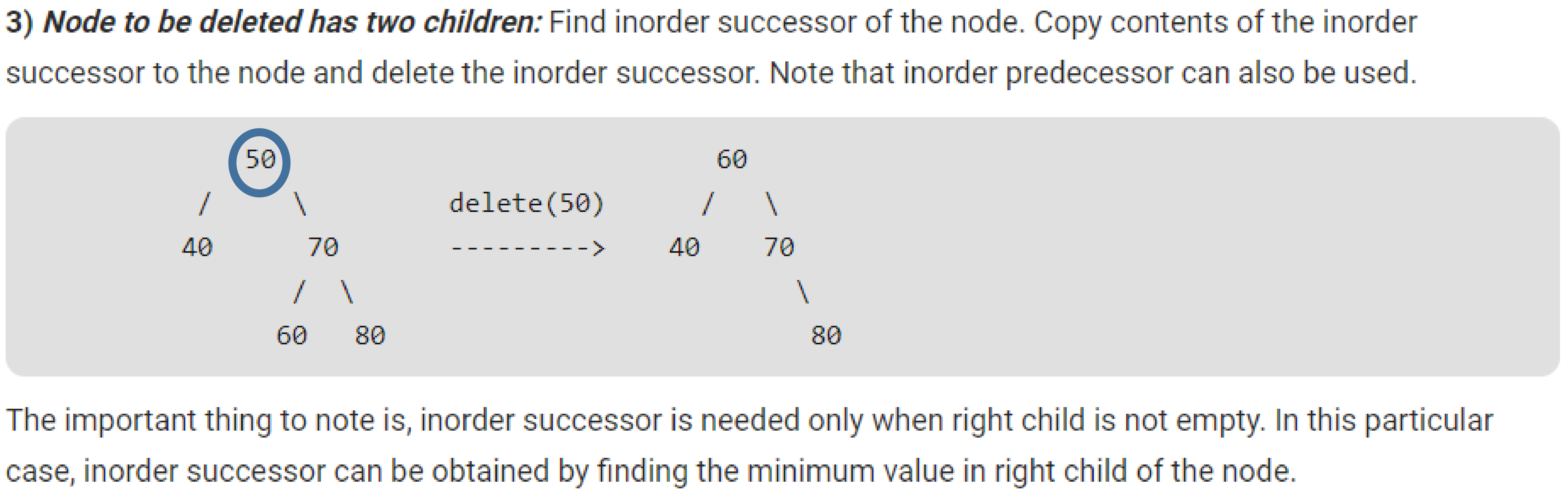
Delete a Node – one child



2) ***Node to be deleted has only one child:*** Copy the child to the node and delete the child

```
    50                            50
 /     \         delete(30)      /   \
30      70       --------->    40     70
  \    /  \                          /  \
  40  60   80                       60   80
```

Delete a Node



3) *Node to be deleted has two children:* Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

```
              50                              60
           /     \         delete(50)       /    \
          40      70       --------->     40     70
                 /  \                              \
                60   80                             80
```

The important thing to note is, inorder successor is needed only when right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in right child of the node.

Delete Node

https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/

```
void deleteKey(int key)
{
        root = deleteRec(root, key);
}
```

```
Node deleteRec(Node root, int key)
 {
          if (root == null) return root; /* Base Case: If the tree is empty */

          /* Otherwise, recur down the tree */
          if (key < root.key)
             root.left = deleteRec(root.left, key);
          else if (key > root.key)
               root.right = deleteRec(root.right, key);

          // if key is same as root's key, then This is the node to be deleted
          else {
              // node with only one child or no child
              if (root.left == null)
                      return root.right;
              else if (root.right == null)
                         return root.left;

              // node with two children: Get the inorder successor
              // (smallest in the right subtree)
                      root.key = minValue(root.right);

              // Delete the inorder successor
                      root.right = deleteRec(root.right, root.key);
          }
       return root;
 }
```

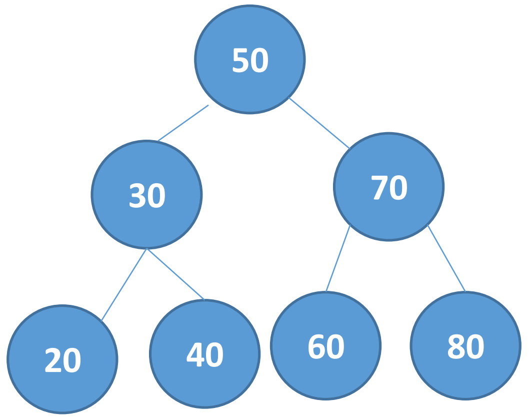
Method: Delete node

```
Node deleteRec(Node root, int key)
 {
       if (root == null) return root; /* Base Case: If the tree is empty */

       /* Otherwise, recur down the tree */
       if (key < root.key)
          root.left = deleteRec(root.left, key);
       else if (key > root.key)
            root.right = deleteRec(root.right, key);

       // if key is same as root's key, then This is the node to be deleted
       else {
           // node with only one child or no child
           if (root.left == null)
                return root.right;
           else if (root.right == null)
                   return root.left;

           // node with two children: Get the inorder successor (smallest in the right subtree)
                root.key = minValue(root.right);

           // Delete the inorder successor
                root.right = deleteRec(root.right, root.key);
       }
      return root;
 }
```

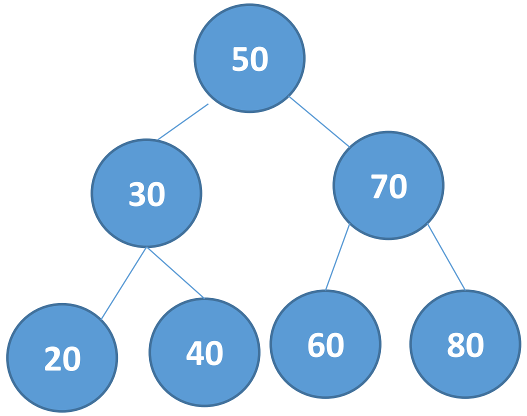
```
//Get the inorder successor (smallest in the right subtree)
int minValue(Node root) {
   int minv = root.key;

   while (root.left != null) {
        minv = root.left.key;
        root = root.left;
   }
   return minv;
}
```

Demo..

```
public static void main(String[] args)
         {
                  BinarySearchTree tree = new BinarySearchTree();

                  tree.insert(50);  tree.insert(30); tree.insert(20);
                  tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                  System.out.println("Inorder traversal 1: ");
                  tree.inorder();

                  System.out.println("\nDelete 20");
                  tree.deleteKey(20);
                  System.out.println("Inorder traversal 2: ");
                  tree.inorder();

                  System.out.println("\nDelete 30");
                  tree.deleteKey(30);
                  System.out.println("Inorder traversal 3: ");
                  tree.inorder();

                  System.out.println("\nDelete 50");
                  tree.deleteKey(50);
                  System.out.println("Inorder traversal 4: ");
                  tree.inorder();
         }
}
```

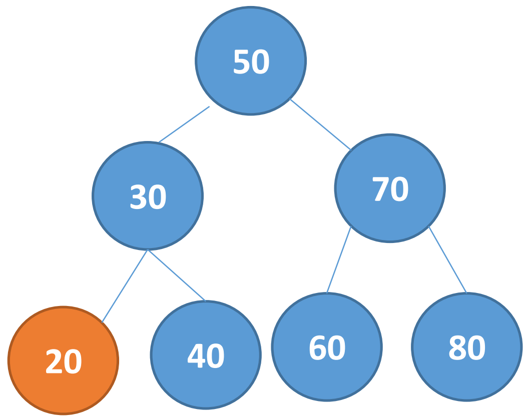


```
public static void main(String[] args)
          {
                    BinarySearchTree tree = new BinarySearchTree();

                    tree.insert(50);  tree.insert(30); tree.insert(20);
                    tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                    System.out.println("Inorder traversal 1: ");
                    tree.inorder();

                    System.out.println("\nDelete 20");
                    tree.deleteKey(20);
                    System.out.println("Inorder traversal 2: ");
                    tree.inorder();

                    System.out.println("\nDelete 30");
                    tree.deleteKey(30);
                    System.out.println("Inorder traversal 3: ");
                    tree.inorder();

                    System.out.println("\nDelete 50");
                    tree.deleteKey(50);
                    System.out.println("Inorder traversal 4: ");
                    tree.inorder();
          }
}
```

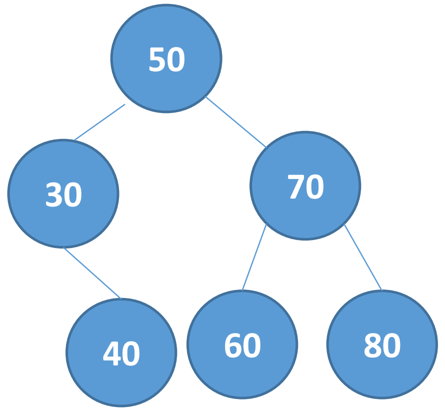


Output Inorder traversal 1: 20 30 40 50 60 70 80

```
public static void main(String[] args)
         {
                  BinarySearchTree tree = new BinarySearchTree();

                  tree.insert(50);  tree.insert(30); tree.insert(20);
                  tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                  System.out.println("Inorder traversal 1: ");
                  tree.inorder();

                  System.out.println("\nDelete 20");
                  tree.deleteKey(20);
                  System.out.println("Inorder traversal 2: ");
                  tree.inorder();

                  System.out.println("\nDelete 30");
                  tree.deleteKey(30);
                  System.out.println("Inorder traversal 3: ");
                  tree.inorder();

                  System.out.println("\nDelete 50");
                  tree.deleteKey(50);
                  System.out.println("Inorder traversal 4: ");
                  tree.inorder();
         }
}
```



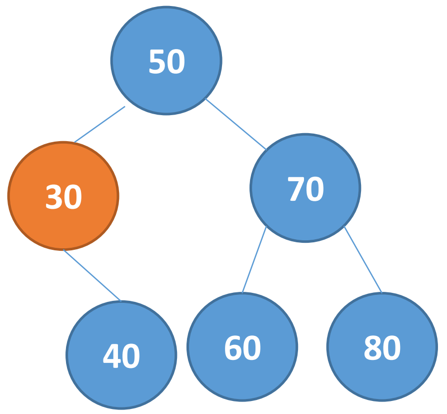
```
public static void main(String[] args)
        {
                BinarySearchTree tree = new BinarySearchTree();

                tree.insert(50);  tree.insert(30); tree.insert(20);
                tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                System.out.println("Inorder traversal 1: ");
                tree.inorder();

                System.out.println("\nDelete 20");
                tree.deleteKey(20);
                System.out.println("Inorder traversal 2: ");
                tree.inorder();

                System.out.println("\nDelete 30");
                tree.deleteKey(30);
                System.out.println("Inorder traversal 3: ");
                tree.inorder();

                System.out.println("\nDelete 50");
                tree.deleteKey(50);
                System.out.println("Inorder traversal 4: ");
                tree.inorder();
        }
}
```



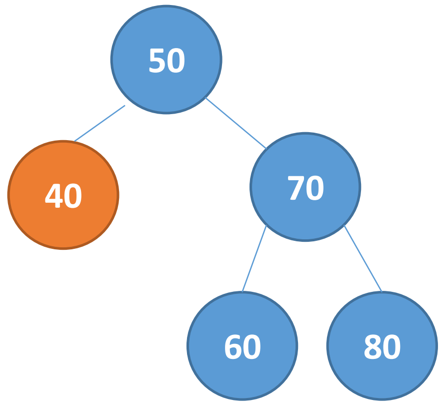
Output

```
Inorder traversal 1: 20 30 40 50 60 70 80
Inorder traversal 2: 30 40 50 60 70 80
```

```
public static void main(String[] args)
         {
                 BinarySearchTree tree = new BinarySearchTree();

                 tree.insert(50);  tree.insert(30); tree.insert(20);
                 tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                 System.out.println("Inorder traversal 1: ");
                 tree.inorder();

                 System.out.println("\nDelete 20");
                 tree.deleteKey(20);
                 System.out.println("Inorder traversal 2: ");
                 tree.inorder();

                 System.out.println("\nDelete 30");
                 tree.deleteKey(30);
                 System.out.println("Inorder traversal 3: ");
                 tree.inorder();

                 System.out.println("\nDelete 50");
                 tree.deleteKey(50);
                 System.out.println("Inorder traversal 4: ");
                 tree.inorder();
         }
}
```



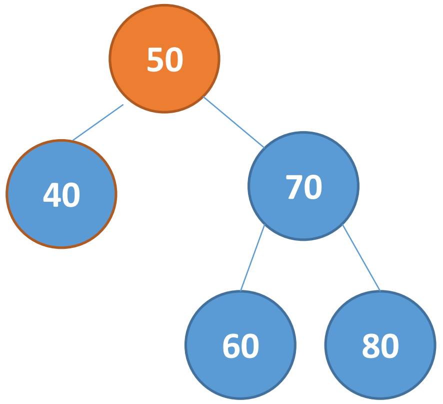
```java
public static void main(String[] args)
         {
                  BinarySearchTree tree = new BinarySearchTree();

                  tree.insert(50);  tree.insert(30); tree.insert(20);
                  tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                  System.out.println("Inorder traversal 1: ");
                  tree.inorder();

                  System.out.println("\nDelete 20");
                  tree.deleteKey(20);
                  System.out.println("Inorder traversal 2: ");
                  tree.inorder();

                  System.out.println("\nDelete 30");
                  tree.deleteKey(30);
                  System.out.println("Inorder traversal 3: ");
                  tree.inorder();

                  System.out.println("\nDelete 50");
                  tree.deleteKey(50);
                  System.out.println("Inorder traversal 4: ");
                  tree.inorder();
         }
}
```



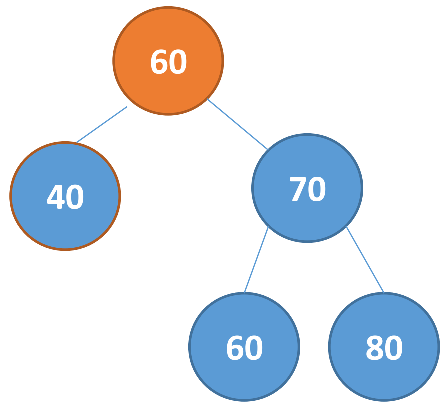
Output

```
Inorder traversal 1: 20 30 40 50 60 70 80
Inorder traversal 2: 30 40 50 60 70 80
Inorder traversal 3: 40 50 60 70 80
```

```
public static void main(String[] args)
        {
                BinarySearchTree tree = new BinarySearchTree();

                tree.insert(50);  tree.insert(30); tree.insert(20);
                tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                System.out.println("Inorder traversal 1: ");
                tree.inorder();

                System.out.println("\nDelete 20");
                tree.deleteKey(20);
                System.out.println("Inorder traversal 2: ");
                tree.inorder();

                System.out.println("\nDelete 30");
                tree.deleteKey(30);
                System.out.println("Inorder traversal 3: ");
                tree.inorder();

                System.out.println("\nDelete 50");
                tree.deleteKey(50);
                System.out.println("Inorder traversal 4: ");
                tree.inorder();
        }
}
```



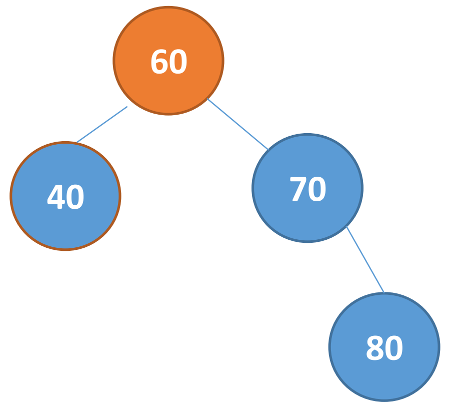
```
public static void main(String[] args)
          {
                    BinarySearchTree tree = new BinarySearchTree();

                    tree.insert(50);  tree.insert(30); tree.insert(20);
                    tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                    System.out.println("Inorder traversal 1: ");
                    tree.inorder();

                    System.out.println("\nDelete 20");
                    tree.deleteKey(20);
                    System.out.println("Inorder traversal 2: ");
                    tree.inorder();

                    System.out.println("\nDelete 30");
                    tree.deleteKey(30);
                    System.out.println("Inorder traversal 3: ");
                    tree.inorder();

                    System.out.println("\nDelete 50");
                    tree.deleteKey(50);
                    System.out.println("Inorder traversal 4: ");
                    tree.inorder();
          }
}
```



```
public static void main(String[] args)
         {
                  BinarySearchTree tree = new BinarySe

                  tree.insert(50);  tree.insert(30); tree.insert(20);
                  tree.insert(40);  tree.insert(70); tree.insert(60); tree.insert(80);

                  System.out.println("Inorder traversal 1: ");
                  tree.inorder();

                  System.out.println("\nDelete 20");
                  tree.deleteKey(20);
                  System.out.println("Inorder traversal 2: ");
                  tree.inorder();

                  System.out.println("\nDelete 30");
                  tree.deleteKey(30);
                  System.out.println("Inorder traversal 3: ");
                  tree.inorder();

                  System.out.println("\nDelete 50");
                  tree.deleteKey(50);
                  System.out.println("Inorder traversal 4: ");
                  tree.inorder();
         }
}
```

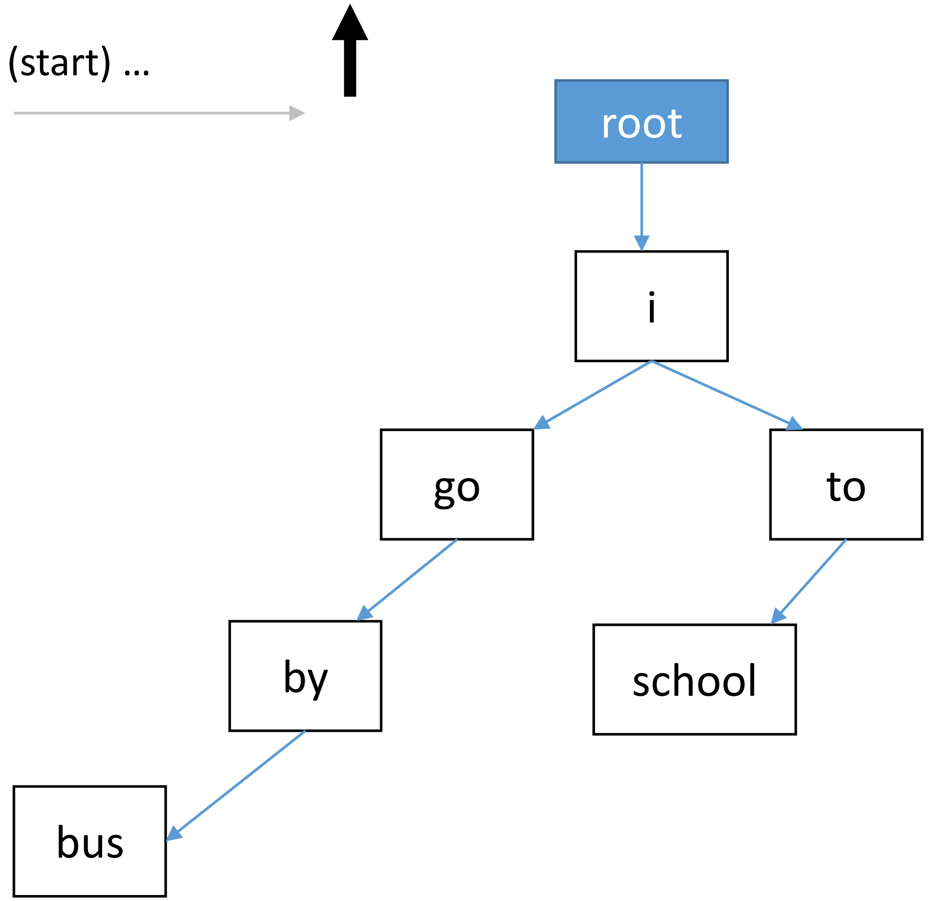
Output

```
Inorder traversal 1: 20 30 40 50 60 70 80
Inorder traversal 2: 30 40 50 60 70 80
Inorder traversal 3: 40 50 60 70 80
Inorder traversal 4: 40 60 70 80
```

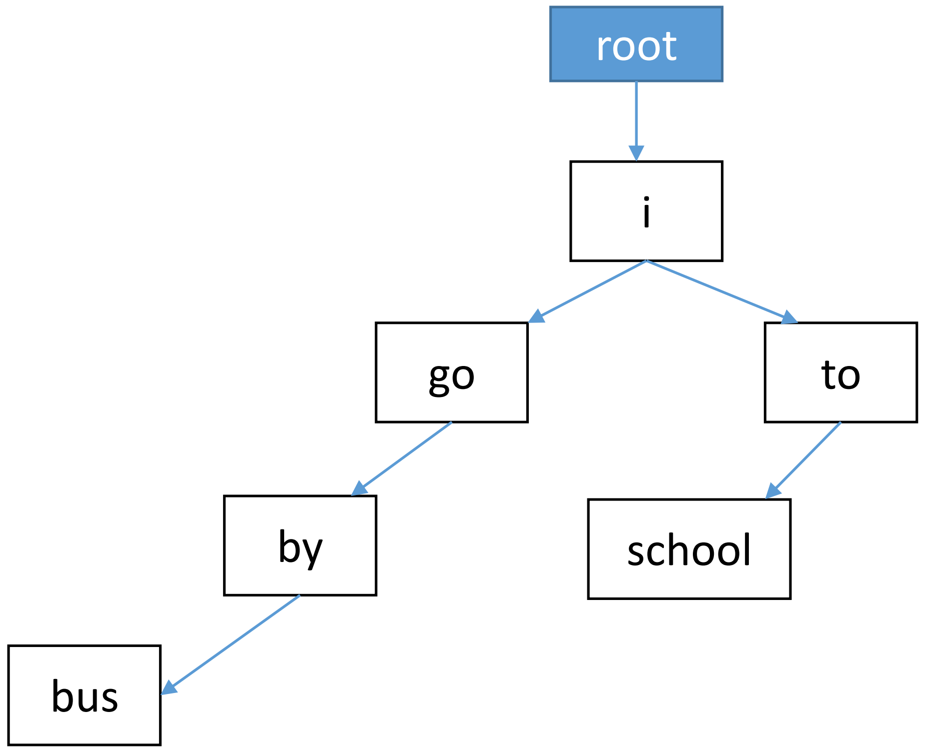


Problem :Text Concordance Revisited

Input Passage: “i go to school by bus. the bus is big. the school is also big. i like big school and big bus.”



Input Passage: "i go to school by bus. the bus is big. the school is also big. i like big school and big bus."


root
i
go
to
by
school
bus