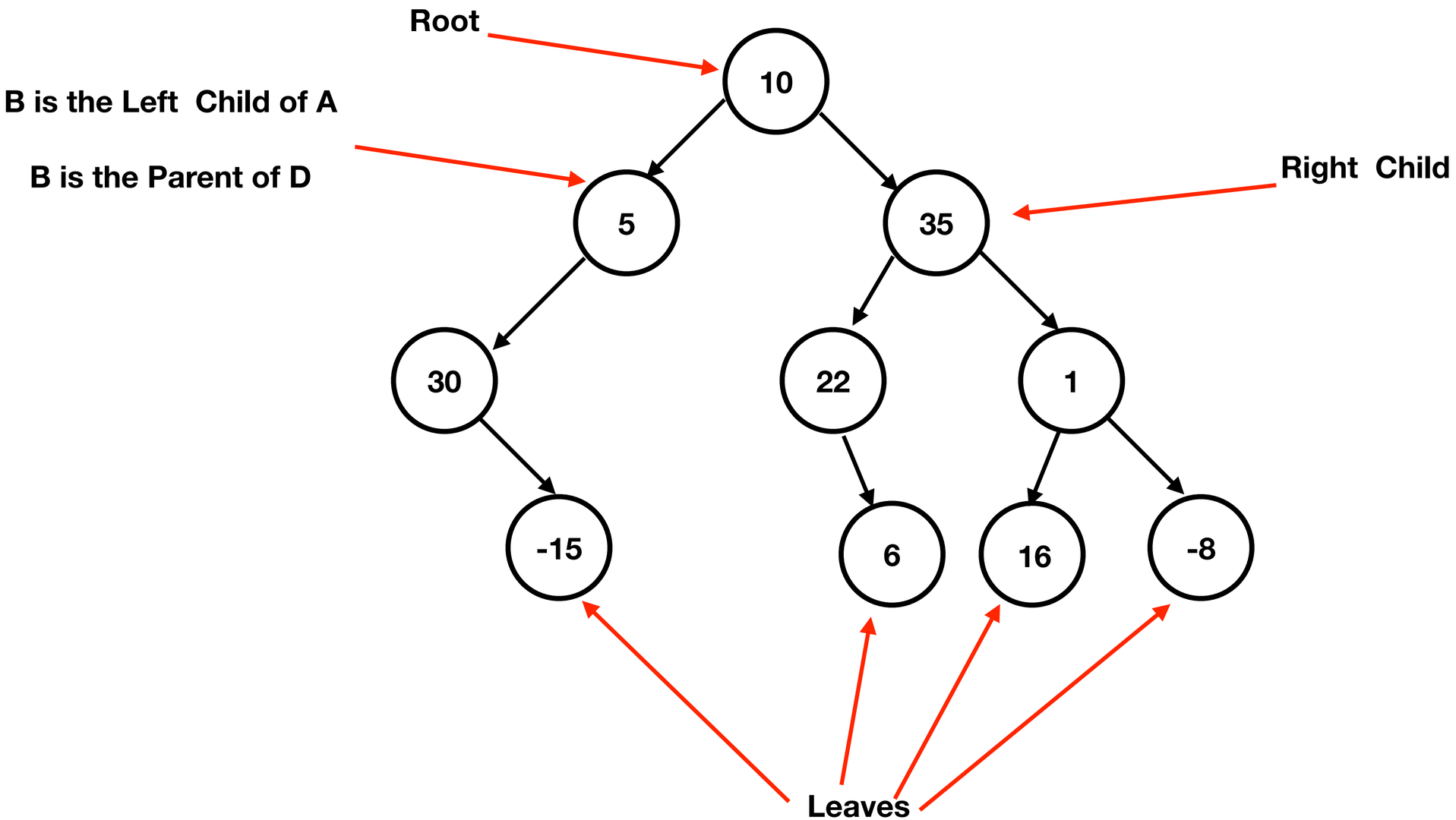
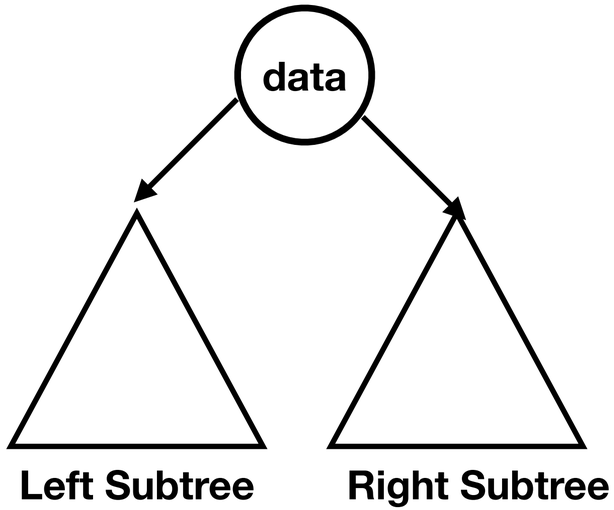
Binary Trees 2

Binary Trees



Binary Tree Algorithms



Binary Tree Algorithms

```
import java.util.*;
import java.io.*;

public class BinaryTree {
  private class Node {
    private Node left;
    private int data;
    private Node right;

    private Node(Node L, int d, Node r) {
      left = L;
      data = d;
      right = r;
    }
  }
  private Node root;
```

Binary Tree Algorithms

```
public void inorder() {
     inorder(root);
     System.out.println();
 }

 private void inorder(Node r) {
     if (r != null) {
         inorder(r.left);
         System.out.print(r.data+" ");
         inorder(r.right);
     }
 }
```

Binary Tree Algorithms

```
public int countPos() {
//returns the number of ints in the tree that are
//greater than 0
	return countPos(root);
}

 private int countPos(Node r) {
	if (r == null) return 0;
	int i = 0;
	if (r.data > 0) i = 1;
	return i + countPos(r.left) + countPos(r.right);
 }
```

Binary Tree Algorithms

```
public int height() {
    //return the height of the tree
    return height(root);
}

private int height(Node r) {
    if (r == null) return -1;
    int leftHeight = height(r.left);
    int rightHeight = height(r.right);
    return leftHeight > rightHeight ? leftHeight+1 : rightHeight+1;
}
```

Binary Tree Algorithms

```
public int size() {
     //return the number of nodes in the tree
     return size(root);
 }

 private int size(Node r) {
     if (r == null) return 0;
     return size(r.left) + size(r.right) + 1;
 }
```

Binary Tree Algorithms

```
public int countRange(int x, int y) {
    //return the number of the ints in the tree that are between x and y inclusive
    return countRange(root, x, y);
}

private int countRange(Node r, int x, int y) {
    if (r == null) return 0;
    int inRange = r.data >= x && r.data <= y ? 1 : 0;
    return inRange + countRange(r.left, x, y) + countRange(r.right, x, y);
}
```

Binary Tree Constructors

```
public BinaryTree() {
    root = null;
}

public BinaryTree(int d) {
    root = new Node(null, d, null);
}

public BinaryTree(String t) {
    Scanner s = new Scanner(t);
    try {
        root = (buildTree(s)).root;
    } catch (Exception e) {
            System.out.println(e.getMessage());
    }
}

public BinaryTree(BinaryTree b1, int d, BinaryTree b2) {
    root = new Node(b1.root, d, b2.root);
}
```

Binary Tree String Representation
Preorder Format

- Empty Tree: !
- Non-Empty Tree: (data left right)
- Examples
  - !
  - (10 ! !)
  - (10 (20 (30 ! !) !) !)
  - (10 ! (5 ! (1 ! !)))
  - (10 (2 ! !) (4 ! !))

Binary Tree buildTree

```
private BinaryTree buildTree(Scanner s) throws Exception {
      int d = 0;;
      BinaryTree b1;
      BinaryTree b2;
      String t  = s.next();
      if (t.equals("!")) return new BinaryTree();
      d = s.nextInt();
      b1 = buildTree(s);
      b2 = buildTree(s);
      s.next(); //closing )
      return new BinaryTree(b1,d,b2);
 }
```

Binary Tree printTree

```
public void printTree() {
    printTree(root);
    System.out.println();
}

private void printTree(Node r) {
    if (r == null) { System.out.print("! "); return;}
    System.out.print("( "+ r.data+" ");
    printTree(r.left);
    printTree(r.right);
    System.out.print(" ) ");

}
```