

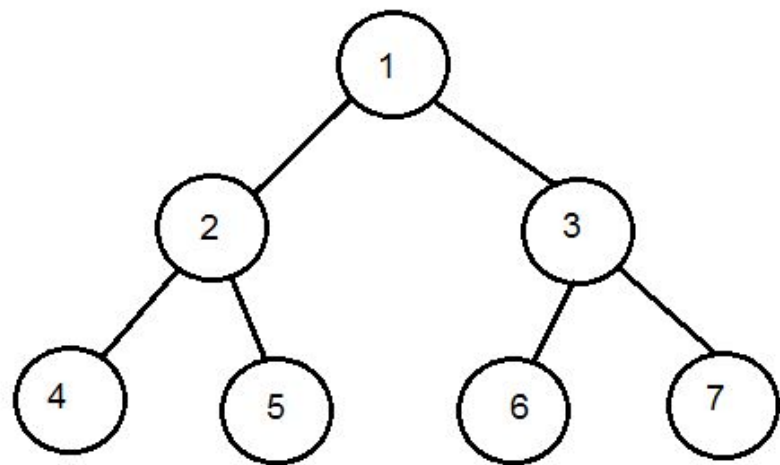
Tree

树

All in one =>

1. Preorder / Inorder / postorder traversal (and construct)
 - recursive(dfs) / iterative (stack)
2. Data Structure
3. DFS, return result of children
4. BFS, level order traversal

Inorder, preorder, postorder



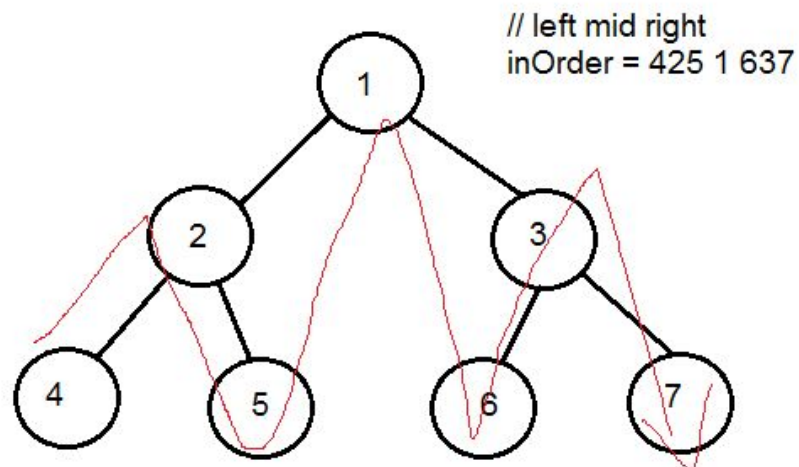
// left mid right
inOrder = 4 2 5 1 6 3 7

// mid left right
preOrder = 1 2 4 5 3 6 7

// left right mid
postOrder = 4 5 2 6 7 3 1

94. Binary Tree Inorder Traversal

<https://leetcode.com/problems/binary-tree-inorder-traversal/>



```
final List<Integer> results = new LinkedList<>();

if (root == null) {
    return results;
}

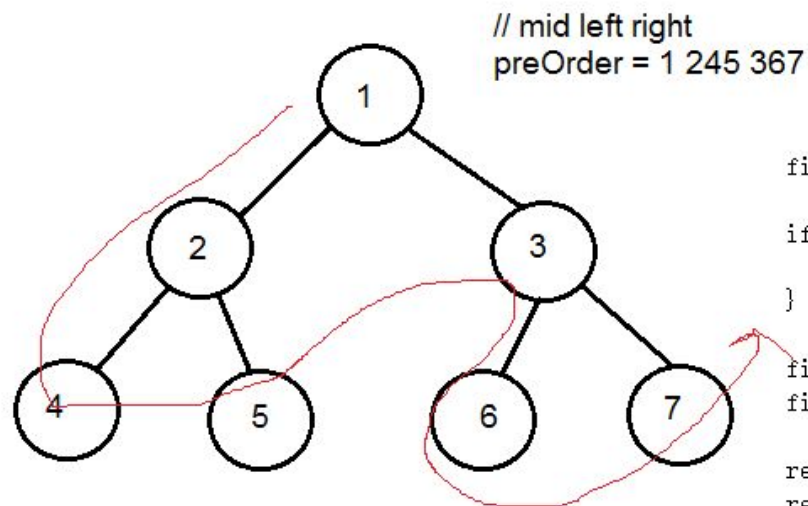
final List<Integer> leftResults = inorderTraversal(root.left);
final List<Integer> rightResults = inorderTraversal(root.right);

results.addAll(leftResults);
results.add(root.val);
results.addAll(rightResults);

return results;
```

144. Binary Tree Preorder Traversal

<https://leetcode.com/problems/binary-tree-preorder-traversal/>



```
final List<Integer> results = new LinkedList<>();

if (root == null) {
    return results;
}

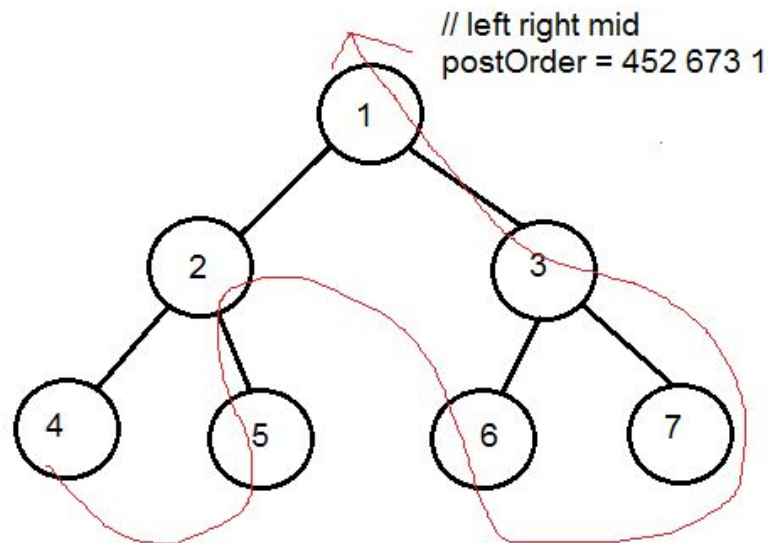
final List<Integer> leftResults = preorderTraversal(root.left);
final List<Integer> rightResults = preorderTraversal(root.right);

results.add(root.val);
results.addAll(leftResults);
results.addAll(rightResults);

return results;
```

145. Binary Tree Postorder Traversal

<https://leetcode.com/problems/binary-tree-postorder-traversal/>



```
final List<Integer> results = new LinkedList<>();
```

```
if (root == null) {  
    return results;  
}
```

```
final List<Integer> leftResults = postorderTraversal(root.left);  
final List<Integer> rightResults = postorderTraversal(root.right);
```

```
results.addAll(leftResults);  
results.addAll(rightResults);  
results.add(root.val);
```

```
return results;
```

105. Construct Binary Tree from Preorder and Inorder Traversal

<https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/>

Inorder ==> left root right

preorder ==> root left right

1. the root is preorder[0]

2. then find root in inorder array

3. now we know how many left nodes, then do it recursively.

```
if (iStart > iEnd || pStart > pEnd) { return null;}
final TreeNode treeNode = new TreeNode(preorder[pStart]);
int index = iStart;
while (iStart < inorder.length && inorder[index] != treeNode.val) {
    ++index;
}
treeNode.left =
    construct(inorder,
        iStart,
        index - 1,
        preorder,
        pStart + 1,
        pStart + index - iStart);
treeNode.right =
    construct(inorder,
        index + 1,
        iEnd,
        preorder,
        pStart + index - iStart + 1,
        pEnd);
return treeNode;
```

106. Construct Binary Tree from Inorder and Postorder Traversal

<https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/>

Inorder ==> left root right

postorder ==> left right root

1. the root is postorder[size-1]

2. then find root in inorder array

3. now we know how many left nodes, then do it recursively.

```
final TreeNode treeNode = new TreeNode(postorder[pEnd]);
int index = iStart;
while (iStart < inorder.length && inorder[index] != treeNode.val) {
    ++index;
}
treeNode.left =
    construct(inorder,
              iStart,
              index - 1,
              postorder,
              pStart,
              pStart + index - 1 - iStart);
treeNode.right =
    construct(inorder,
              index + 1,
              iEnd,
              postorder,
              pStart + index - iStart,
              pEnd - 1);
return treeNode;
```


889. Construct Binary Tree from Preorder and Postorder Traversal

<https://leetcode.com/problems/construct-binary-tree-from-preorder-and-postorder-traversal/>

preorder ==> root left right

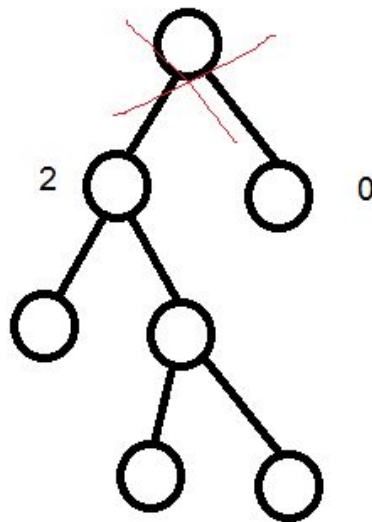
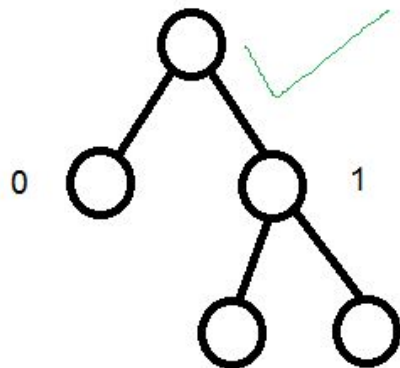
postorder ==> left right root

1. root is preorder[0]
2. find the first left value, preorder[rootIndex + 1]
in postorder,
the first left value in preorder is the last left value in postorder
3. do it recursively.

```
/*  
preorder = 1 245 367  
postorder = 452 673 1  
root = 1  
leftStartValue = 2  
then just find the 2 in postorder  
*/
```

110. Balanced Binary Tree

<https://leetcode.com/problems/balanced-binary-tree/>



balanced ?

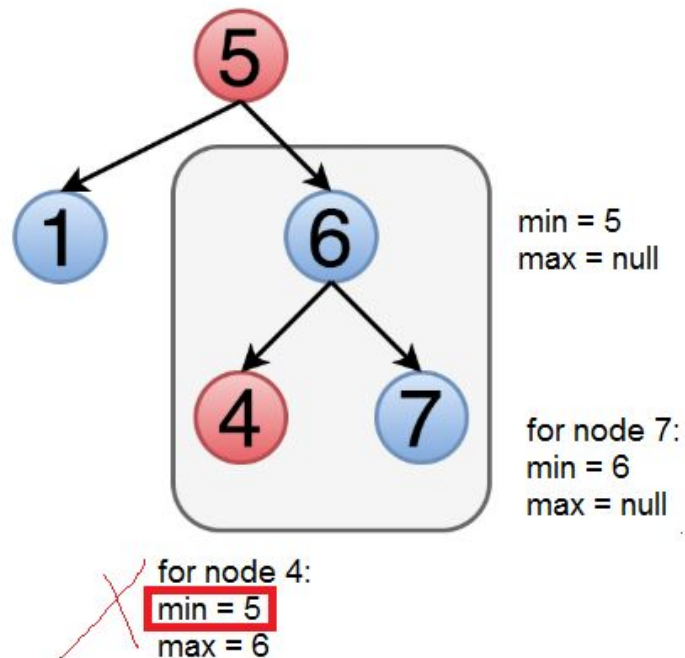
1. left tree is balanced ?

2. right tree is balanced ?

3. $\text{abs}(\text{leftHeight} - \text{rightHeight}) \leq 1$

98. Validate Binary Search Tree

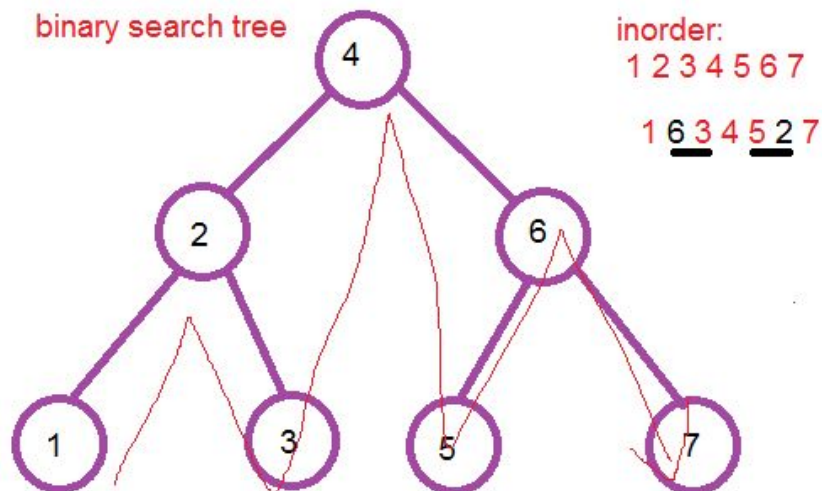
<https://leetcode.com/problems/validate-binary-search-tree/>



```
private boolean helper(TreeNode root, Integer min, Integer max) {  
    if (root == null) {  
        return true;  
    }  
  
    if (min != null && root.val <= min) {  
        return false;  
    }  
    if (max != null && root.val >= max) {  
        return false;  
    }  
  
    return helper(root.left, min, root.val) &&  
        helper(root.right, root.val, max);  
}
```

99. Recover Binary Search Tree

<https://leetcode.com/problems/recover-binary-search-tree/>



current = 4

parent = 2

parent = 3

3. right = 4

current = 2

parent = 1

1. right = 2

current = 1

trait 1

current = 1.right = 2

parent = 1

but 1.right = 2 == current

so 1. right = null

trait 2

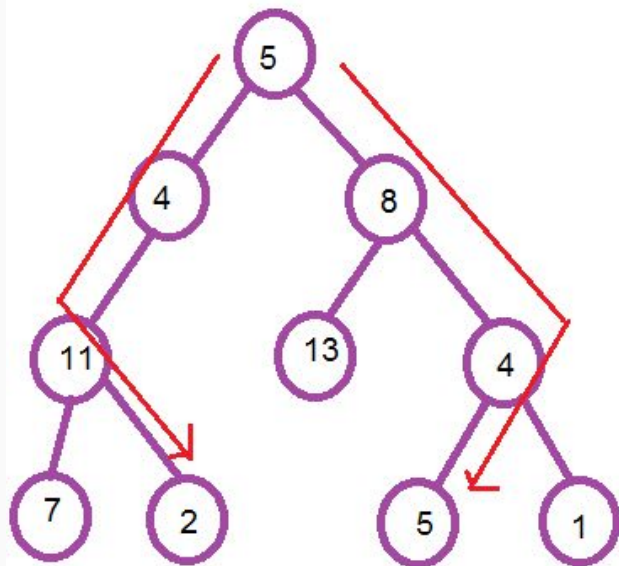
current = 3

trait 3

current = 3.right = 4

113. Path Sum II

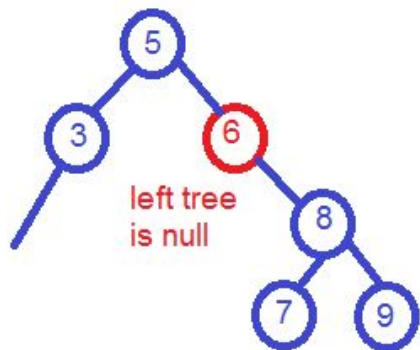
<https://leetcode.com/problems/path-sum-ii/>



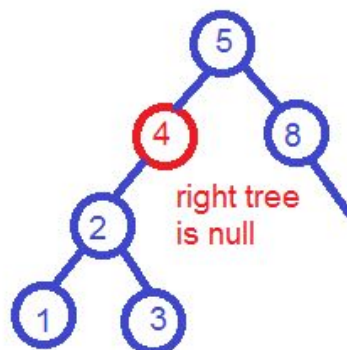
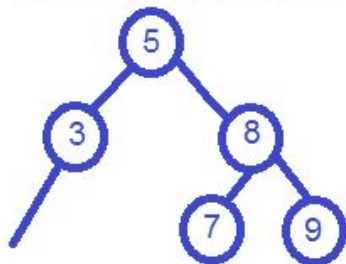
```
private void helperDfs(
    TreeNode root,
    int current,
    int sum,
    List<Integer> subsets,
    List<List<Integer>> results) {
    if (root == null) { return; }
    subsets.add(root.val);
    final int currentSum = current + root.val;
    if (root.left == null && root.right == null) {
        if (currentSum == sum) {
            results.add(new ArrayList<>(subsets));
        }
        subsets.remove(subsets.size() - 1);
        return;
    }
    helperDfs(root.left, currentSum, sum, subsets, results);
    helperDfs(root.right, currentSum, sum, subsets, results);
    subsets.remove(subsets.size() - 1);
}
```

450. Delete Node in a BST

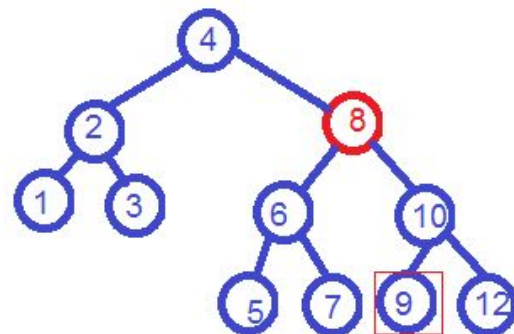
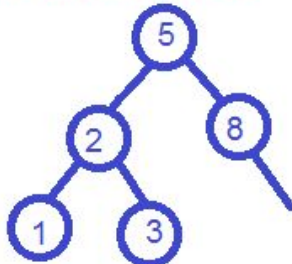
<https://leetcode.com/problems/delete-node-in-a-bst/>



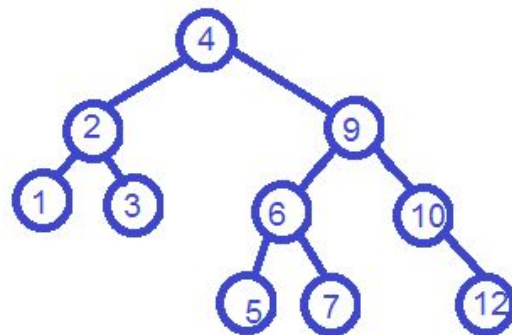
replace it with right tree



replace it with left tree



the minimum value in right tree



116. Populating Next Right Pointers in Each Node

<https://leetcode.com/problems/populating-next-right-pointers-in-each-node/>

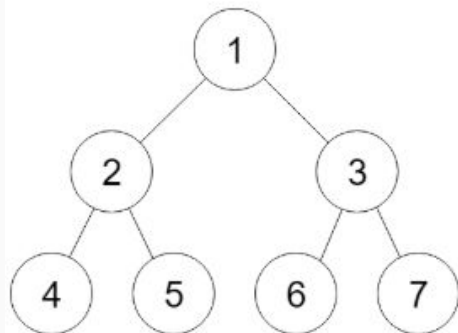


Figure A

```
while (start.left != null) {  
    current = start;  
    while (current != null) {  
        current.left.next = current.right;  
        if (current.next != null) {  
            current.right.next = current.next.left;  
        }  
        current = current.next;  
    }  
    start = start.left;  
}
```

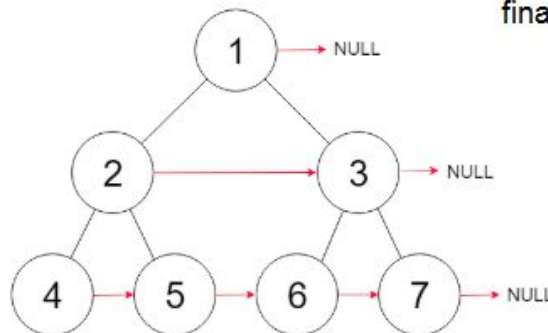
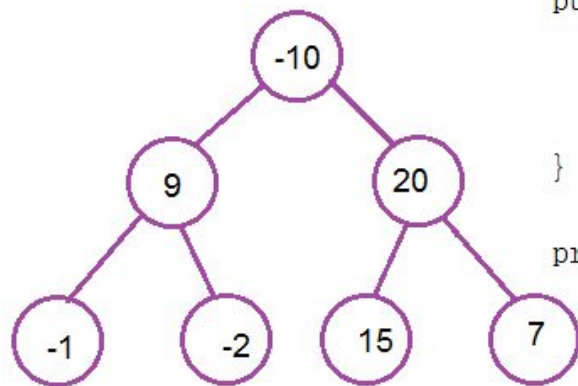


Figure B

```
final int currentSize = queue.size();  
Node previous = null;  
for (int size = 0; size < currentSize; size++) {  
    final Node current = queue.poll();  
    if (previous != null) {  
        previous.next = current;  
    }  
    previous = current;  
    if (current.left != null) {  
        queue.offer(current.left);  
    }  
    if (current.right != null) {  
        queue.offer(current.right);  
    }  
}
```


124. Binary Tree Maximum Path Sum

<https://leetcode.com/problems/binary-tree-maximum-path-sum/>

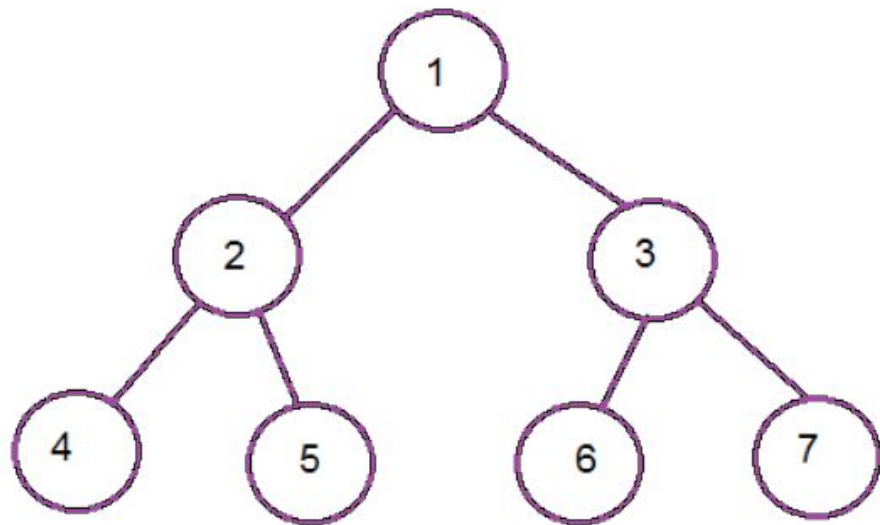


1. if child node < 0, then take 0
2. for each node, the sum will be
 $\max(\text{left}, \text{right}) + \text{current value}$

```
public int maxPathSum(TreeNode root) {  
    result = Integer.MIN_VALUE;  
    dfs(root);  
    return result;  
}  
  
private int dfs(TreeNode root) {  
    if (root == null) {  
        return 0;  
    }  
    final int leftResult = Math.max(0, dfs(root.left));  
    final int rightResult = Math.max(0, dfs(root.right));  
    result = Math.max(result, leftResult + rightResult + root.val);  
    return Math.max(leftResult, rightResult) + root.val;  
}
```


297. Serialize and Deserialize Binary Tree

<https://leetcode.com/problems/serialize-and-deserialize-binary-tree/>



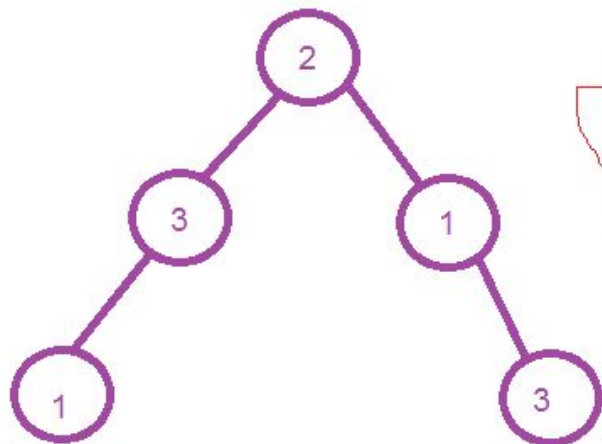
// left mid right
inOrder = 4 2 5 1 6 3 7

// mid left right
preOrder = 1 2 4 5 3 6 7

// left right mid
postOrder = 4 5 2 6 7 3 1

337. House Robber III

<https://leetcode.com/problems/house-robber-iii/>



main idea is to keep an array of length 2,
see comments

```
/*  
results[0] : take current node value  
results[1] : do not take current, but take child nodes' values  
*/  
private fun dfs(root: TreeNode?): IntArray {  
    if (root == null) {  
        return intArrayOf(0, 0)  
    }  
    if (root.left == null && root.right == null) {  
        return intArrayOf(root.`val`, 0)  
    }  
    val leftResults = dfs(root.left)  
    val rightResults = dfs(root.right)  
    val results = IntArray(2)  
    // take current, so we can't take child nodes' values  
    results[0] = root.`val` + leftResults[1] + rightResults[1]  
    // do not take current, take max of left/right  
    results[1] = maxOf(leftResults[0], leftResults[1]) +  
        maxOf(rightResults[0], rightResults[1])  
    return results  
}
```

501. Find Mode in Binary Search Tree

<https://leetcode.com/problems/find-mode-in-binary-search-tree/>

[1 2 2 4 4 6 6 6 6 7]

how to find mode?

1. define a results to store temp candidat.
2. preValue, max, count

recursrive inorder traversal

iterative inorder traversal (stack)

```
// compare with preValue
if (preValue == current.`val`) {
    ++count
} else {
    count = 1
}
```

```
// compare with max
if (count > max) {
    max = countds
    results.clear()
    results.add(current.`val`)
} else if (count == max) {
    results.add(current.`val`)
}
```

```
// reassign preValue
preValue = current.`val`
```

530. Minimum Absolute Difference in BST

<https://leetcode.com/problems/minimum-absolute-difference-in-bst/>

BST : Binary Search Tree

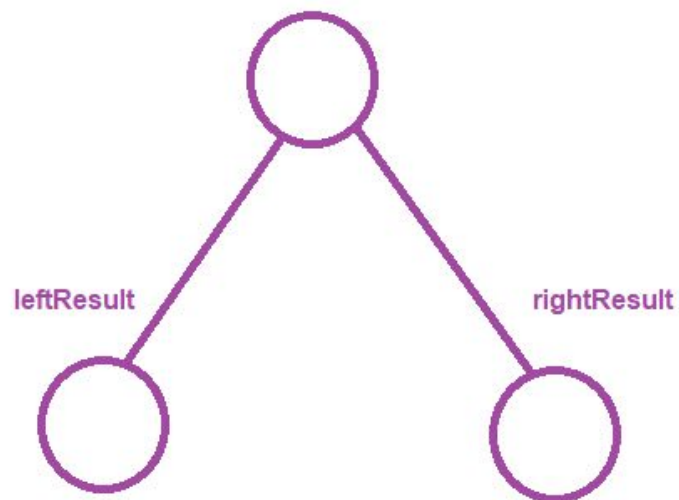
Left < Mid(root) < Right

if we do an inorder traversal, the results list is in ascending order!

1. recursive inorder traversal
2. iterative inorder traversal

Divide and Conquer

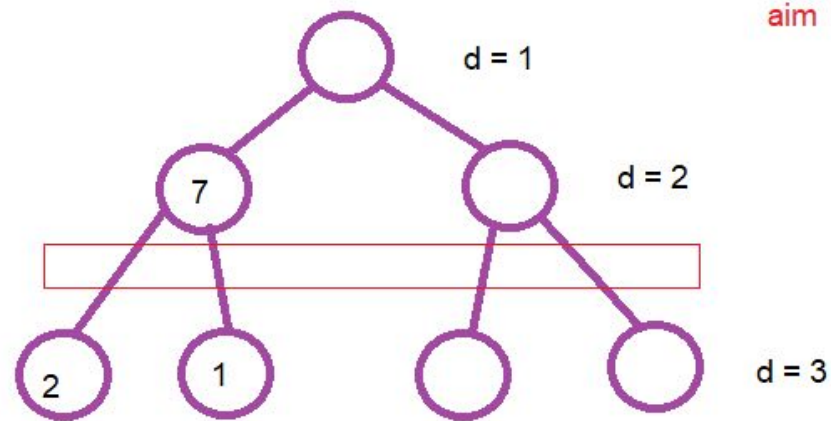
Divide and Conquer



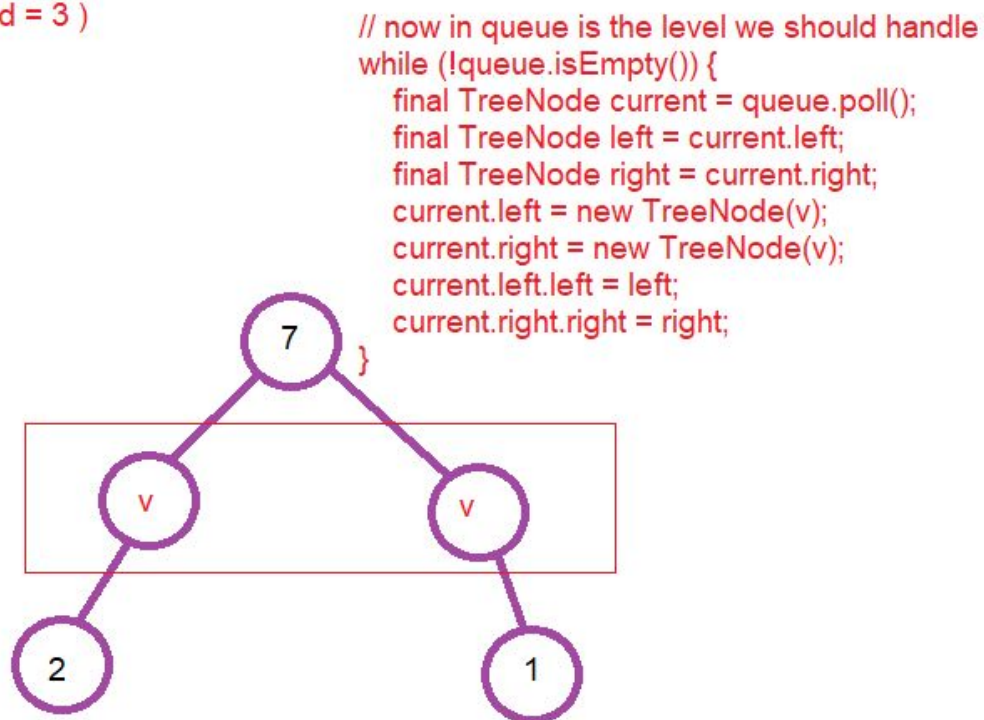
```
private int dfs(TreeNode root) {  
    if (root == null) {  
        return 0;  
    }  
    final int leftVal = dfs(root.left);  
    final int rightVal = dfs(root.right);  
    sum += Math.abs(leftVal - rightVal);  
    return leftVal + rightVal + root.val;  
}
```

623. Add One Row to Tree

<https://leetcode.com/problems/add-one-row-to-tree/>



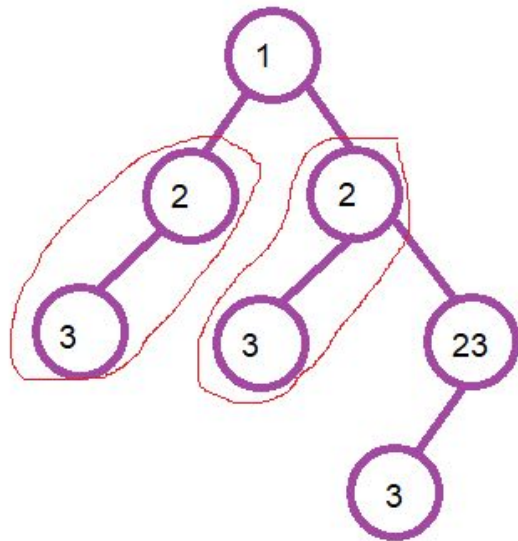
aim ($d = 3$)



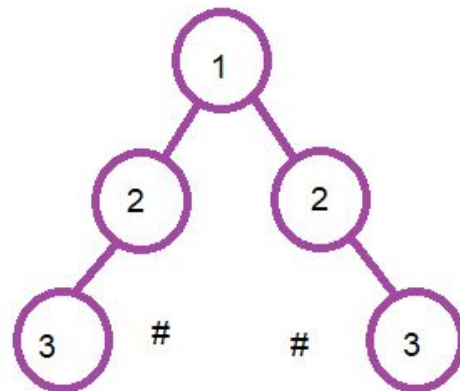
```
// now in queue is the level we should handle
while (!queue.isEmpty()) {
    final TreeNode current = queue.poll();
    final TreeNode left = current.left;
    final TreeNode right = current.right;
    current.left = new TreeNode(v);
    current.right = new TreeNode(v);
    current.left.left = left;
    current.right.right = right;
}
```

652. Find Duplicate Subtrees

<https://leetcode.com/problems/find-duplicate-subtrees/>



1,(2,3),(2,3),23,3



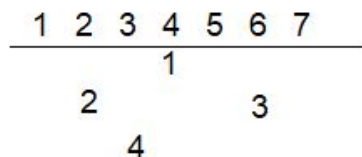
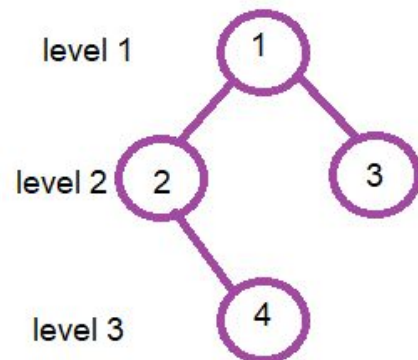
653. Two Sum IV - Input is a BST

<https://leetcode.com/problems/two-sum-iv-input-is-a-bst/>

1. Two Sum : use HashMap to store value,index, $O(N)$ $S(N)$
2. Two Sum sorted array : **binary search**! $\log(n) * \log(n)$
3. Two Sum BST: $O(N)$ $S(N)$

655. Print Binary Tree

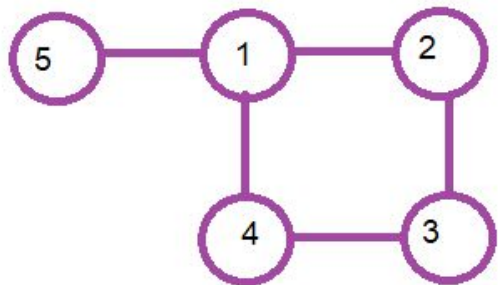
<https://leetcode.com/problems/print-binary-tree/>



1. get the height of tree, 3
2. width is $1 \ll \text{height}$, 8
3. the index of root node is $\text{width} \gg 1$, 4
4. current distance is $\text{width} \gg 2$, is 2
so left node index is $4 - 2 = 2$
so right node index is $4 + 2 = 6$
5. current distance $\gg= 1$

684. Redundant Connection

<https://leetcode.com/problems/redundant-connection/>



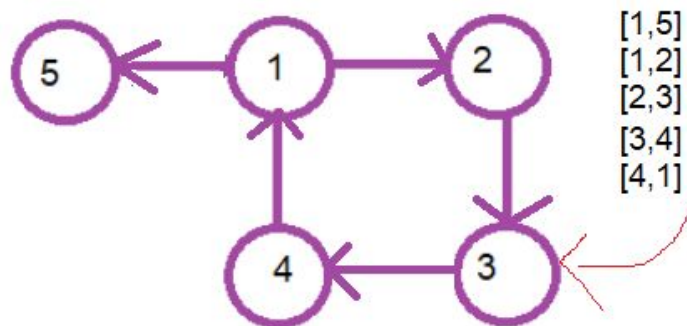
[1,2]
[2,3]
[3,4]
[1,4]
[1,5]

1. for each edge, add it to graph
2. if an edge is already in graph, and can connect, return it
3. use BFS or DFS to check the connection

```
for (edge in edges) {  
    if (graph[edge[0]].isEmpty() && graph[edge[1]].isEmpty() && bfs(graph, edge[0], edge[1])) {  
        return edge  
    }  
    graph[edge[0]].add(edge[1])  
    graph[edge[1]].add(edge[0])  
}
```

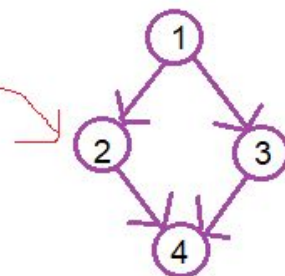
685. Redundant Connection II

<https://leetcode.com/problems/redundant-connection-ii/>



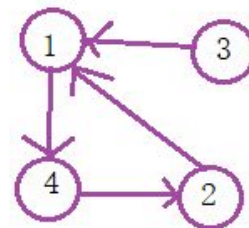
two cases of redundant connections:

1. one node has two parents
2. every node has one parent



orbit function
start from 1,
seen = [1,4,3,2]

return the last node that edge [0] [1] are both in seen,
which is [4,1]



[2,1]
[3,1]
[4,2]
[1,4]

remove [2,1]

DFS on tree

DFS == Recursion

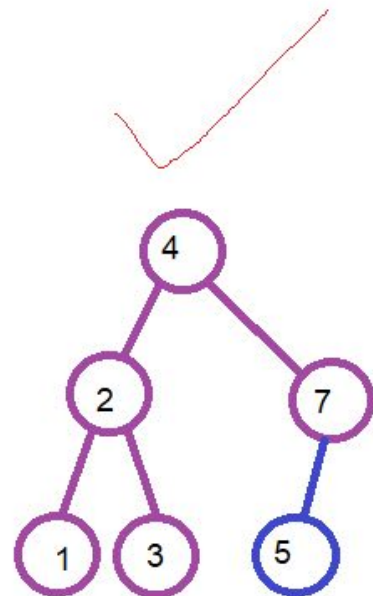
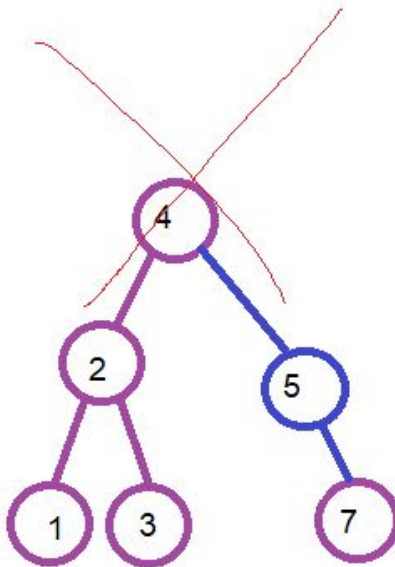
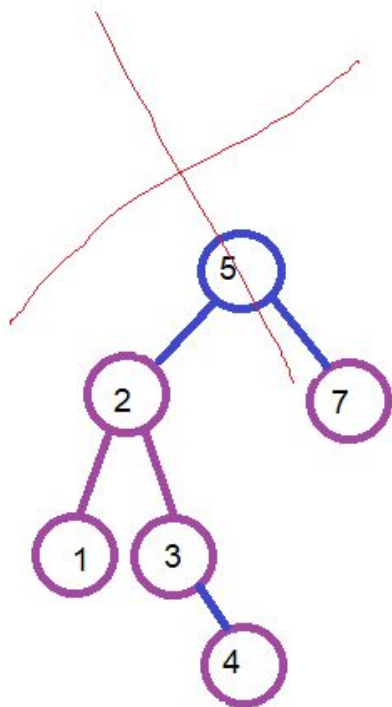
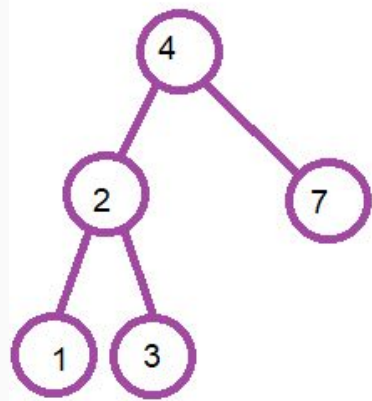
1. definition
2. end condition
3. call dfs

```
leftResult = dfs  
rightResult = dfs
```

```
root.val ?
```

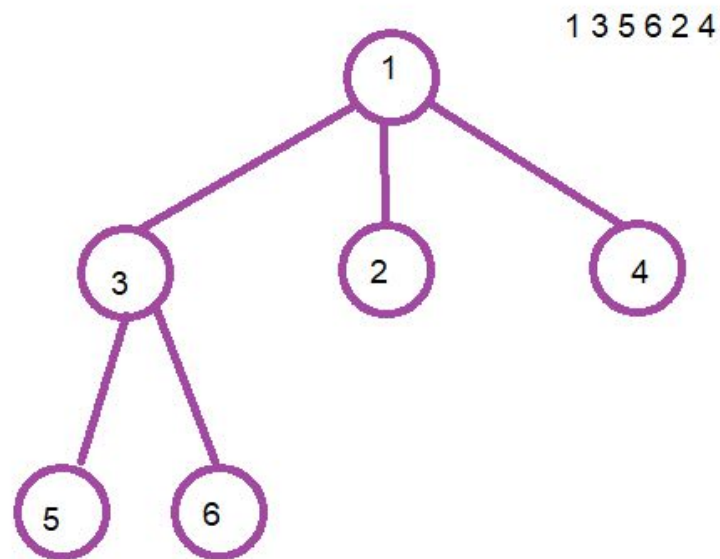
701. Insert into a Binary Search Tree

<https://leetcode.com/problems/insert-into-a-binary-search-tree/>



589. N-ary Tree Preorder Traversal & 590. N-ary Tree Postorder Traversal

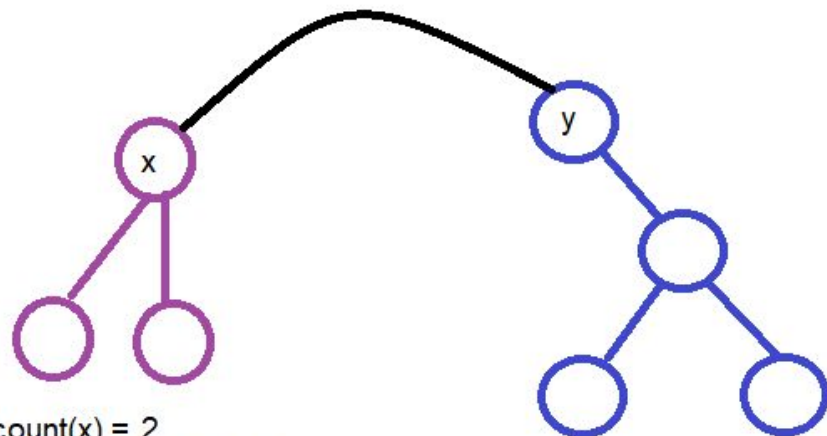
<https://leetcode.com/problems/n-ary-tree-preorder-traversal/>



```
public List<Integer> preorder(Node root) {  
    final List<Integer> results = new LinkedList<>();  
    if (root == null) {  
        return results;  
    }  
    results.add(root.val);  
    for (Node child : root.children) {  
        results.addAll(preorder(child));  
    }  
    return results;  
}
```

834. Sum of Distances in Tree

<https://leetcode.com/problems/sum-of-distances-in-tree/>



$\text{count}(x) = 2$
 $\text{subTreeSum}(x) = 2$

$\text{count}(y) = 4$
 $\text{subTreeSum}(y) = 5$

$\text{answer}[x] = \text{subTreeSum}(x) + \text{subTreeSum}(y) + \text{count}(y) = 2 + 5 + 4 = 11$
 $\text{answer}[y] = \text{subTreeSum}(y) + \text{subTreeSum}(x) + \text{count}(x) = 5 + 2 + 2 = 9$
 $\text{answer}[x] - \text{answer}[y] = \text{count}(y) - \text{count}(x)$

$\text{answer}[x] = \text{answer}[y] + (N - \text{count}(x)) - \text{count}(x)$

1. convert edges to graph
2. use post order traversal to calculate
=> $\text{count}(?)$
=> $\text{subTreeSum}(?)$

$\text{count}(k) += \text{all children}$
 $\text{subTreeSum}(k) = \text{subTreeSum}(\text{children}) + \text{count}(\text{children})$

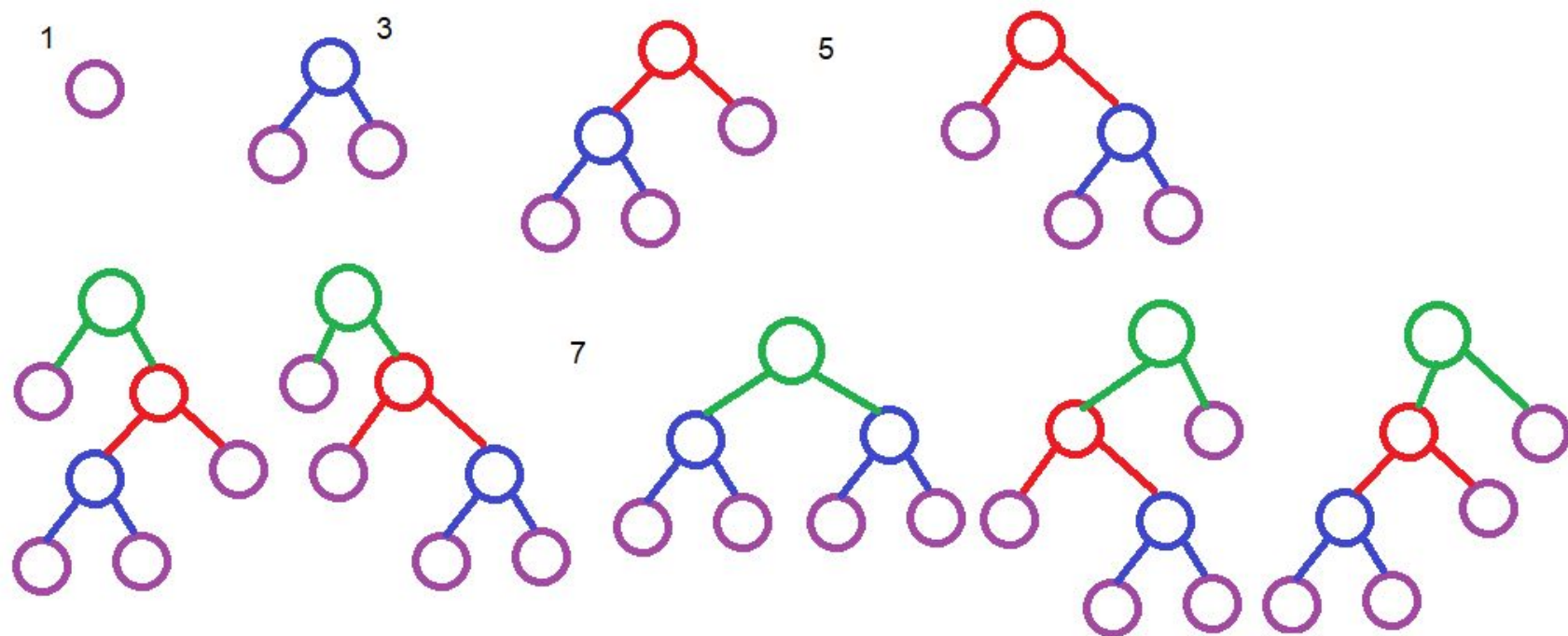
so we have the right answer for **root**

3. for each child, use preorder traversal

$\text{answer}[\text{child}] = \text{answer}[\text{Parent}]$
 $\quad + (N - \text{count}(\text{child}))$
 $\quad - \text{count}(\text{child})$

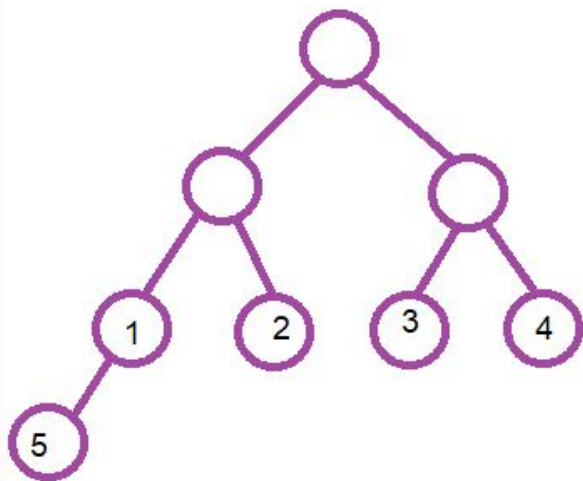
894. All Possible Full Binary Trees

<https://leetcode.com/problems/all-possible-full-binary-trees/>



919. Complete Binary Tree Inserter

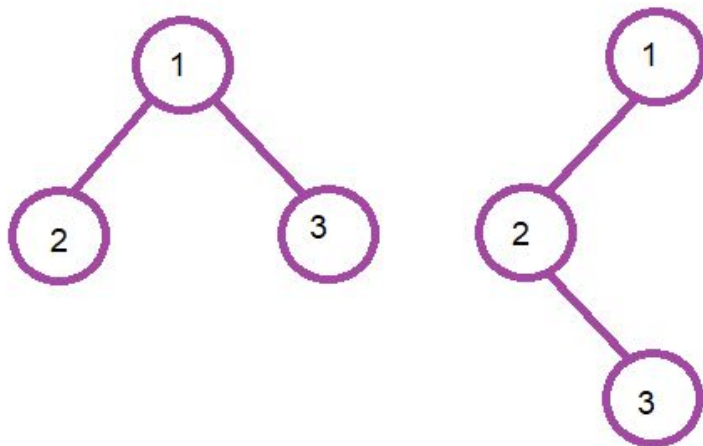
<https://leetcode.com/problems/complete-binary-tree-inserter/>



1. bfs to find uncompleted nodes ($\text{left} == \text{null} \parallel \text{right} == \text{null}$)
2. add them into a queue
3. when insert, peek() a parent in queue, after add new node, if parent node is completed, queue.pop().

951. Flip Equivalent Binary Trees

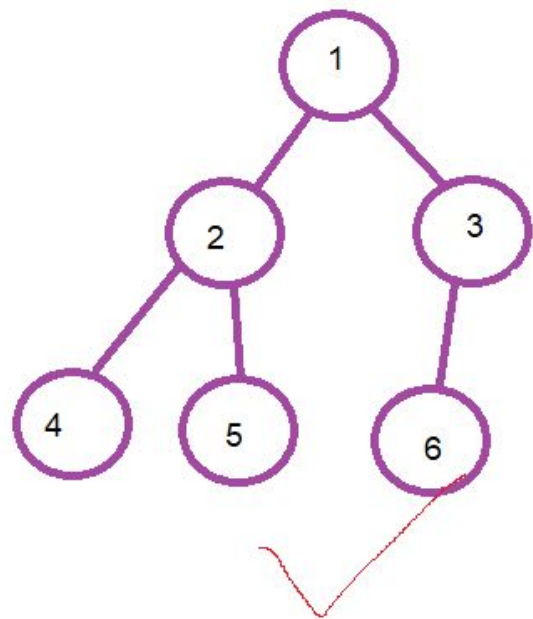
<https://leetcode.com/problems/flip-equivalent-binary-trees/>



```
private void traversal(TreeNode node, List<Integer> results) {
    if (node == null) {
        results.add(null);
        return;
    }
    results.add(node.val);
    final int left = node.left == null ? -1 : node.left.val;
    final int right = node.right == null ? -1 : node.right.val;
    if (left < right) {
        traversal(node.left, results);
        traversal(node.right, results);
    } else {
        traversal(node.right, results);
        traversal(node.left, results);
    }
}
```

958. Check Completeness of a Binary Tree

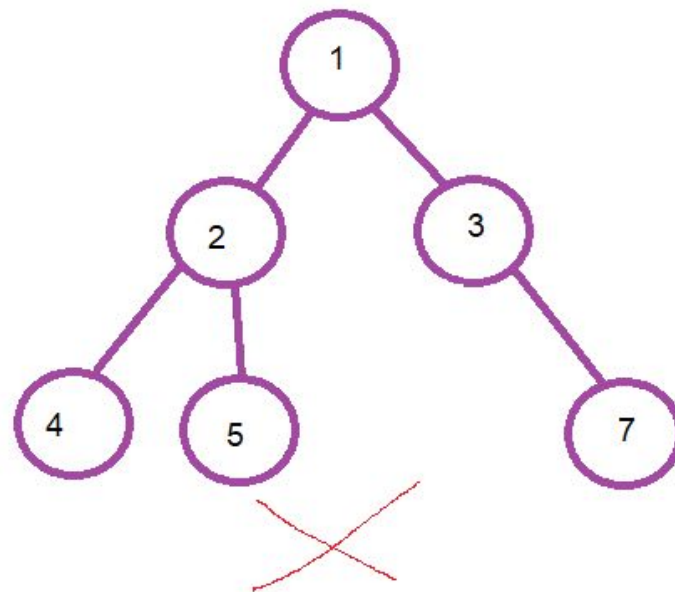
<https://leetcode.com/problems/check-completeness-of-a-binary-tree/>



value = 2

left = value * 2 = 4

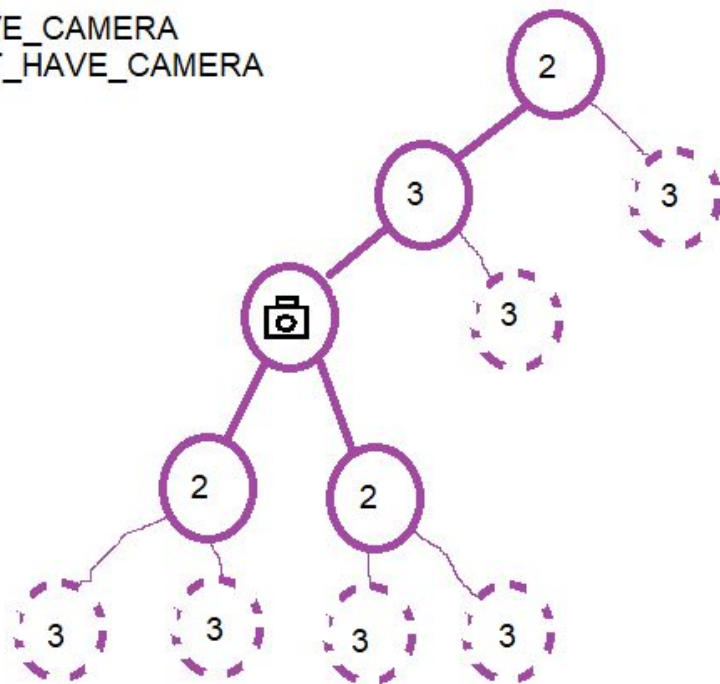
right = value * 2 + 1 = 5



968. Binary Tree Cameras

<https://leetcode.com/problems/binary-tree-cameras/>

1. CAMERA
2. MUST_HAVE_CAMERA
3. MUST_NOT_HAVE_CAMERA



if node == null, 3

if one child is 2, then current is CAMERA

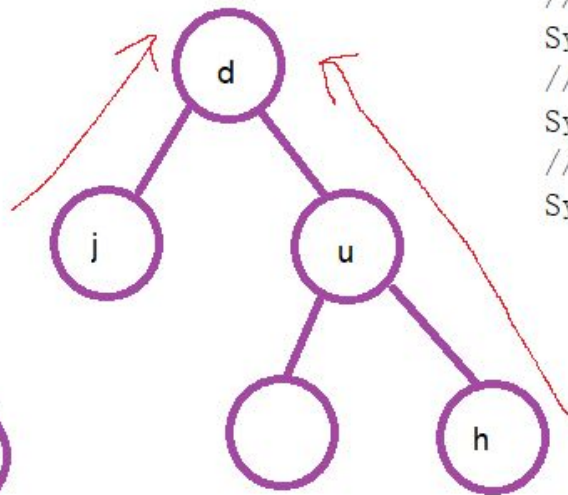
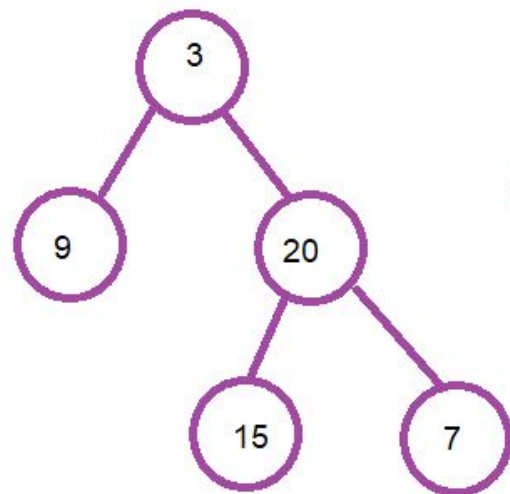
if one child is CAMERA, then current is 3

else (both children are 3), current is 2

notice that if root is type 2, result++

988. Smallest String Starting From Leaf

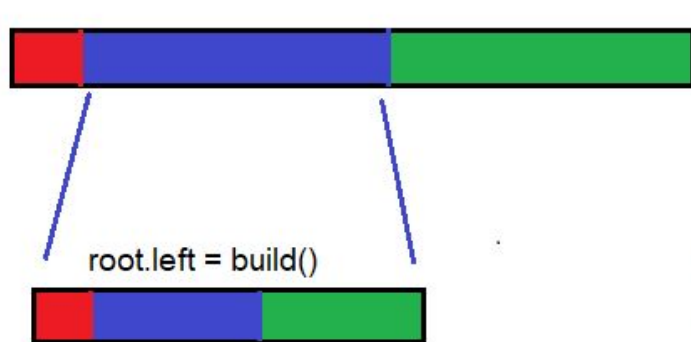
<https://leetcode.com/problems/smallest-string-starting-from-leaf/>



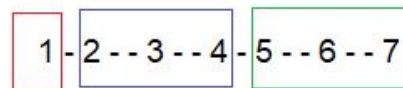
```
// -1
System.out.println(Integer.compare(0, 1));
// -1
System.out.println("ab".compareTo("abc"));
// 2
System.out.println("jd".compareTo("hud"));
```

1028. Recover a Tree From Preorder Traversal

<https://leetcode.com/problems/recover-a-tree-from-preorder-traversal/>



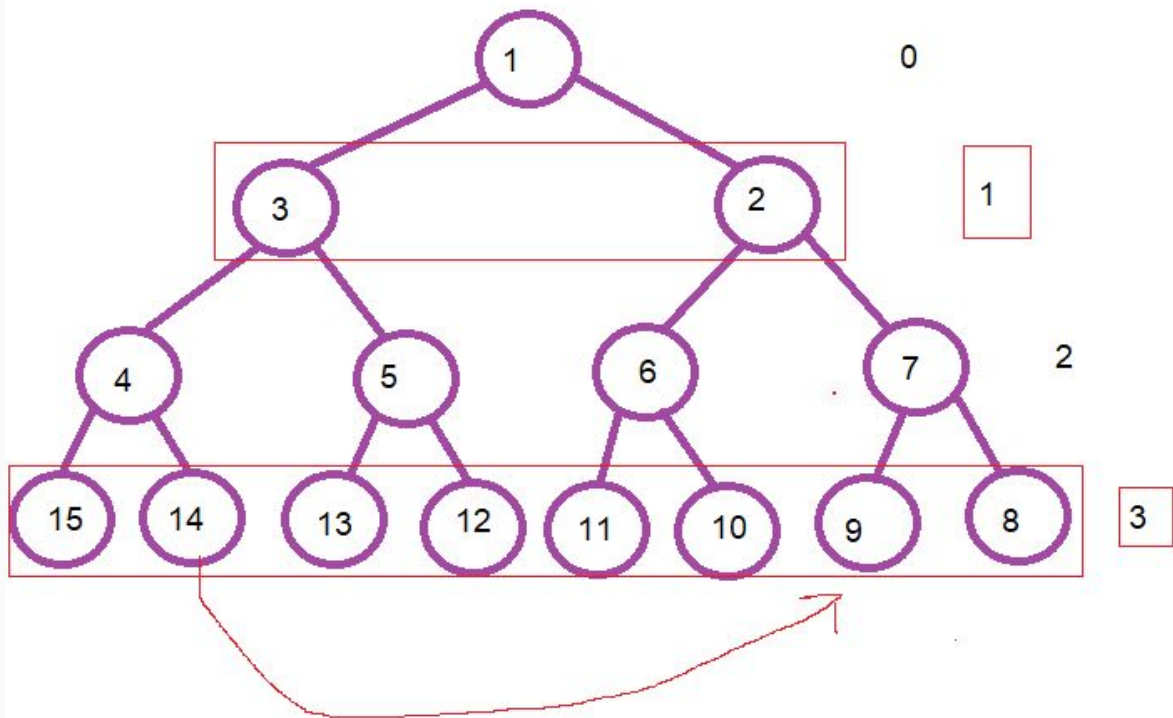
root, left, right



1. build tree from charArray
2. take care of chars to int, for example 405 - -, when parse the first integer, it's 405, NOT 4
3. the difficult is how to search the start point of **right part** and calculate the end of **left part**

1104. Path In Zigzag Labelled Binary Tree

<https://leetcode.com/problems/path-in-zigzag-labelled-binary-tree/>



1. calculate level

2. calculate the "true value"

3. add "zigzag" value into result list

level 3

trueVal = 9, zigzagVal = 14

level 2

trueVal = 4, zigzagVal = 4

level 1

trueVal = 2, zigzagVal = 3

level 0

1

1130. Minimum Cost Tree From Leaf Values

<https://leetcode.com/problems/minimum-cost-tree-from-leaf-values/>

[7 12 8 10]

we cant sort the given array!

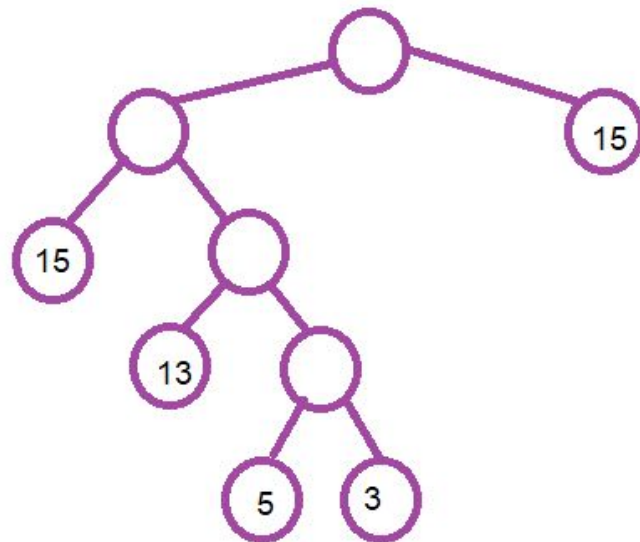
the leaf node should keep the same order.

~~[7 8 10 12]~~ ❌

idea: make sure always use smallest integers to get product.

[15, 13, 5, 3, 15]

$3 * 5$
+
 $5 * 13$
+
 $13 * 15$
+
 $15 * 15$

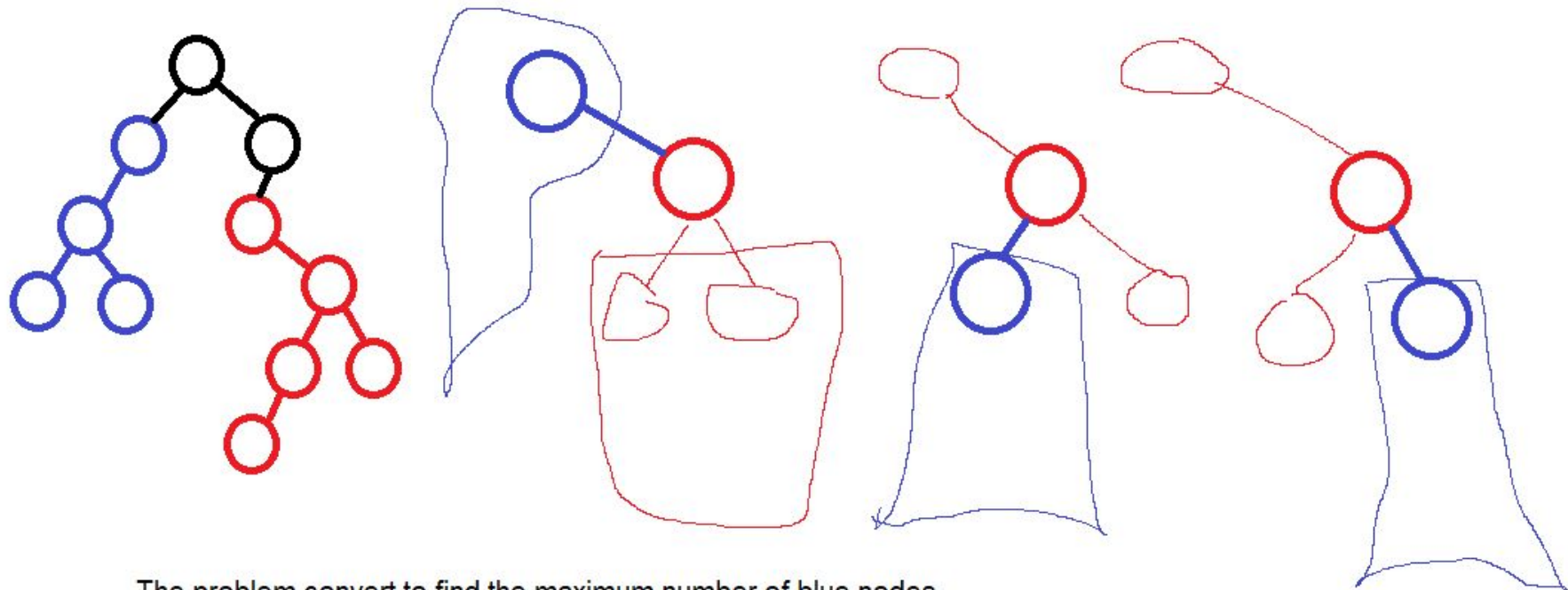


use stack, if `currentVal >= stack.peek()`

we should make sure `stack.peek()` multiply the smallest value
(**current** or **previous value in stack**)

1145. Binary Tree Coloring Game

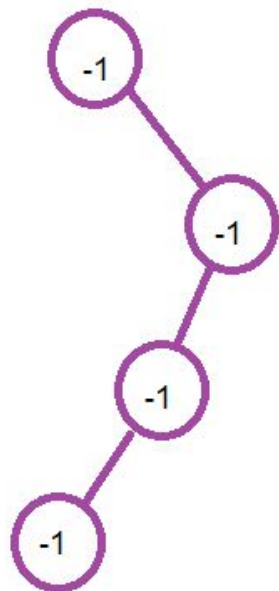
<https://leetcode.com/problems/binary-tree-coloring-game/>



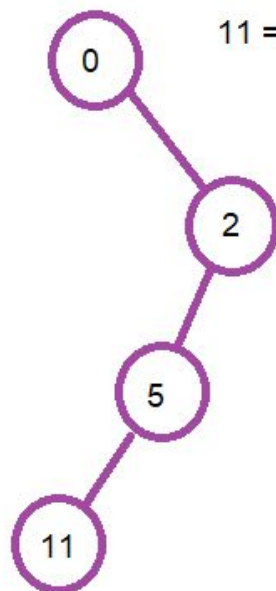
The problem convert to find the maximum number of blue nodes.

1261. Find Elements in a Contaminated Binary Tree

<https://leetcode.com/problems/find-elements-in-a-contaminated-binary-tree/>



$\text{left} = 2 * \text{val} + 1$
 $\text{right} = 2 * \text{val} + 2$



11 => RIGHT, LEFT, LEFT

stack:
LEFT
LEFT
RIGHT

```
final Stack<Integer> stack
```

```
while (target > 0) {
```

```
    if (target % 2 == 0) {  
        stack.push(RIGHT);  
        target = (target - 2) >> 1;
```

```
    } else {
```

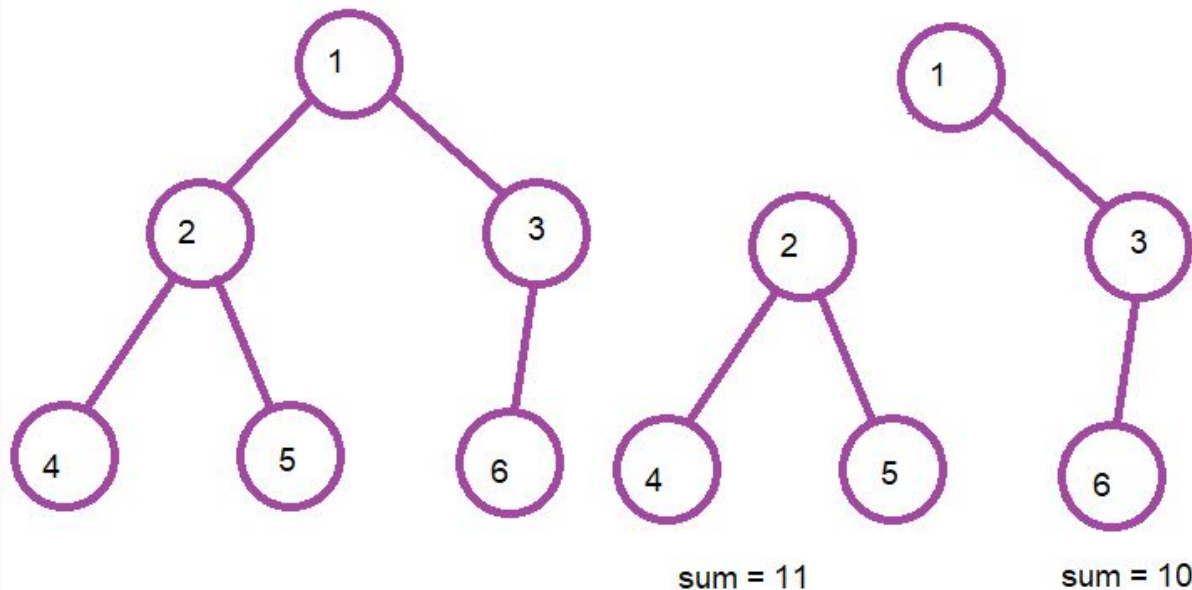
```
        stack.push(LEFT);  
        target = (target - 1) >> 1;
```

```
    }
```

```
}
```

1339. Maximum Product of Splitted Binary Tree

<https://leetcode.com/problems/maximum-product-of-splitted-binary-tree/>



[1 2 3 4 5 6]

preSum

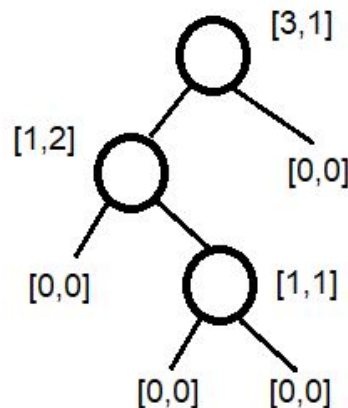
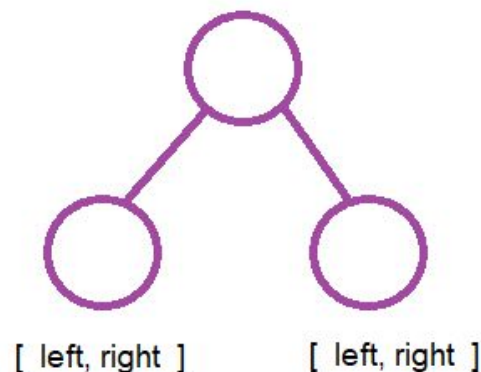
[1 3 6 10 15 21]

Sum - preSum

[0 18 15 11 6 0]

1372. Longest ZigZag Path in a Binary Tree

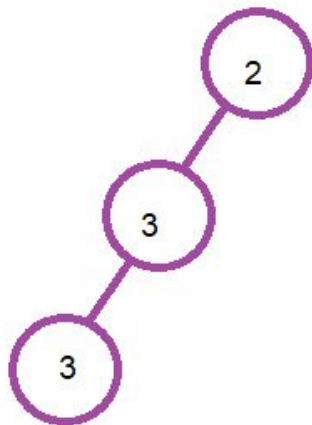
<https://leetcode.com/problems/longest-zigzag-path-in-a-binary-tree/>



```
private int[] dfs(TreeNode root) {  
    if (root == null) {  
        return new int[] {0, 0};  
    }  
    final int[] leftResult = dfs(root.left);  
    final int[] rightResult = dfs(root.right);  
  
    final int leftMax = 1 + leftResult[1];  
    final int rightMax = 1 + rightResult[0];  
  
    max = Math.max(max, leftMax);  
    max = Math.max(max, rightMax);  
  
    return new int[] {leftMax, rightMax};  
}
```

1457. Pseudo-Palindromic Paths in a Binary Tree

<https://leetcode.com/problems/pseudo-palindromic-paths-in-a-binary-tree/>



if there exists a pseudo-palindromic path, there must have at most one single int.

value = 2

mask = mask xor(^) (1<<(val-1))
= 0(00) ^ 2 (10)
= 2(10)

value = 3

mask = 2 (10) xor 4 (100)
= (6)110

value = 3

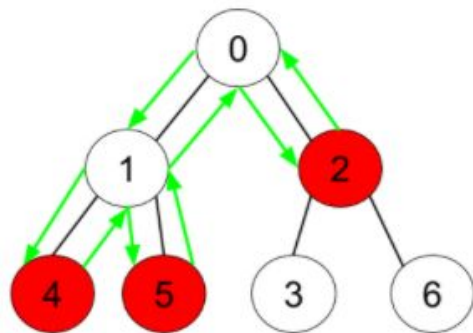
mask = 6 (110) xor 4(100) = 2 (010)

if(mask & (mask-1)){ ++result }

for mask, at most one bit could be 1
so for a given mask, and (mask-1) should have 0.

1443. Minimum Time to Collect All Apples in a Tree

<https://leetcode.com/problems/minimum-time-to-collect-all-apples-in-a-tree/>



1. build graph

2. dfs, result is time to collect

Algorithm:

for each node, count its neighbors's time.

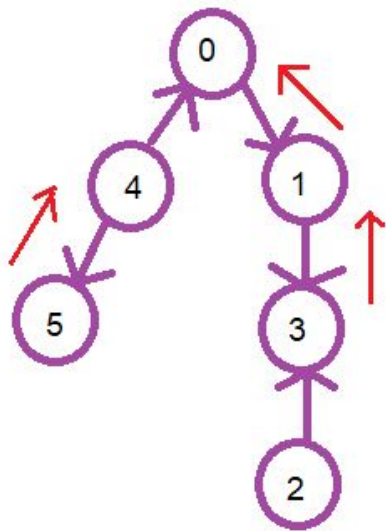
==> if node is "root", return result,

==> if node is "leaf" or result of neighbors is 0,
return 2 if hasApple or 0.

==> Others, return count + 2. (it means node is in path of collecting apples)

1466. Reorder Routes to Make All Paths Lead to the City Zero

<https://leetcode.com/problems/reorder-routes-to-make-all-paths-lead-to-the-city-zero/>



count how many vectors are inversed

two ways:

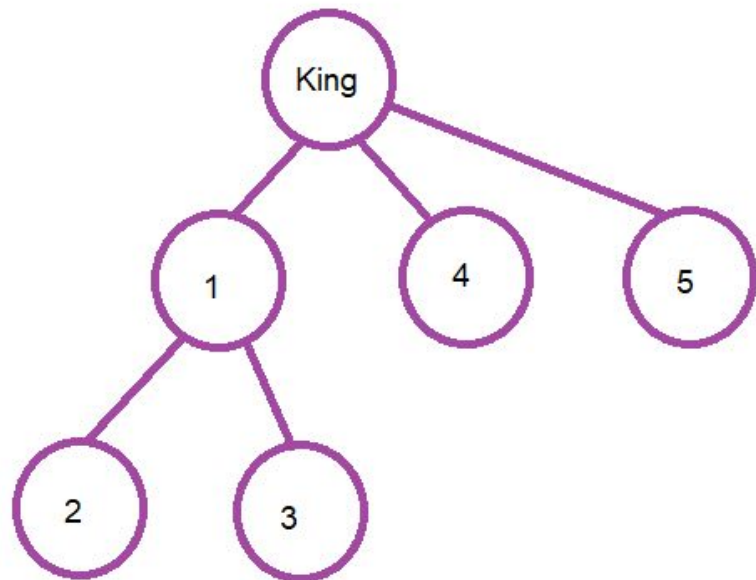
- a ==>
1. build graph, and record the original connections into a map
 2. dfs travel from 0, if exists connect from child to parent, return 0 else 1

(advanced)

- b ==>
1. build graph, original connect marked as [0] to [1],
inversed connection as [1] to -[0]
 2. when travel from parent to child(neighbor), if [1] > 0, return 1 else 0

1600. Throne Inheritance

<https://leetcode.com/problems/throne-inheritance/>



```
private final Map<String, List<String>> graph = new HashMap<>();
```

```
private final HashSet<String> deaths = new HashSet<>();
```

```
public List<String> getInheritanceOrder() {  
    final List<String> results = new LinkedList<>();  
    dfs(king, results);  
    return results;  
}
```

```
private void dfs(String current, List<String> results) {  
    if (!deaths.contains(current)) { here:  
        results.add(current); preorder traversal  
    }  
    if (graph.containsKey(current)) {  
        for (String child : graph.get(current)) {  
            dfs(child, results);  
        }  
    }
```


2385. Amount of Time for Binary Tree to Be Infected