What is the difference between tree depth and height?

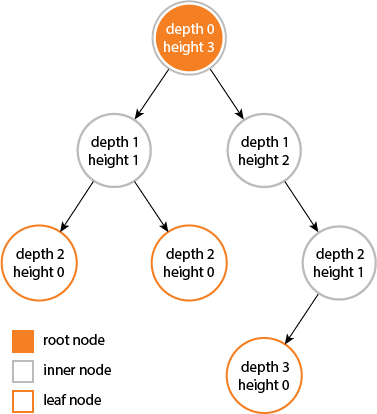
<https://stackoverflow.com/questions/2603692/what-is-the-difference-between-tree-depth-and-height>

I learned that depth and height are properties of a node:

* The **depth** of a node is the number of edges from the node to the tree's root node.  
  A root node will have a depth of 0.
* The **height** of a node is the number of edges on the longest path from the node to a leaf.  
  A leaf node will have a height of 0.

Properties of a tree:

* The **height** of a tree would be the height of its root node,  
  or equivalently, the depth of its deepest node.
* The **diameter** (or **width**) of a tree is the number of nodes on the longest path between any two leaf nodes. The tree below has a diameter of 6 nodes.



**Find depth for each node (stored in map)**

**Use classic Recursive Traversal (so called Top Down DFS [遍历法132: 1.base case -> 3.进行当前层的处理计算 -> 2.递归成为更小的问题] to make sure the depth increasing by the way when traverse the tree from root to leaf)**

**Style 1: void return recursive traversal**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* /

\* 4

\*/

two.left = one;

two.right = three;

three.left = four;

s.findDepth(two);

}

Map<TreeNode, Integer> map;

private void findDepth(TreeNode root) {

map = new HashMap<TreeNode, Integer>();

// root's parent is null, the depth for none existing node is -1

map.put(null, -1);

helper(root, null);

}

private void helper(TreeNode root, TreeNode parent) {

// Base

if(root == null) {

return;

}

// Process on current level

map.put(root, map.get(parent) + 1);

// Recursive into smaller problem

helper(root.left, root);

helper(root.right, root);

}

}

**Style 2: return 'depth' with additional parameter 'depth' pass in recursion function**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* /

\* 4

\*/

two.left = one;

two.right = three;

three.left = four;

s.findDepth(two);

}

Map<TreeNode, Integer> map;

private void findDepth(TreeNode root) {

map = new HashMap<TreeNode, Integer>();

// Additional parameter as depth, for root initial depth is 0

helper(root, 0);

}

private int helper(TreeNode root, int depth) {

// Base

if(root == null) {

return depth; // Not return 0, must return depth

}

// Process on current level

map.put(root, depth);

// Recursive into smaller problem

int leftDepth = helper(root.left, depth + 1);

int rightDepth = helper(root.right, depth + 1);

return Math.max(leftDepth, rightDepth);

}

}

**Find height for each node (stored in map)**

**Use classic Divide and Conquer (so called bottom-up DFS[遍历法123: 1.base case -> 2.递归成为更小的问题 -> 3.进行当前层的处理计算] to make sure the height increasing by the way when traverse the tree from leaf to root)**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* /

\* 4

\*/

two.left = one;

two.right = three;

three.left = four;

int result = s.findHeight(two);

System.out.println(result);

}

Map<TreeNode, Integer> map = new HashMap<TreeNode, Integer>();

private int findHeight(TreeNode root) {

// Base

// For none exist node the height is -1

if(root == null) {

return -1;

}

// Recursive into smaller problem

int left = findHeight(root.left);

int right = findHeight(root.right);

// Process on current level

int height = Math.max(left, right) + 1;

map.put(root, height);

return height;

}

}