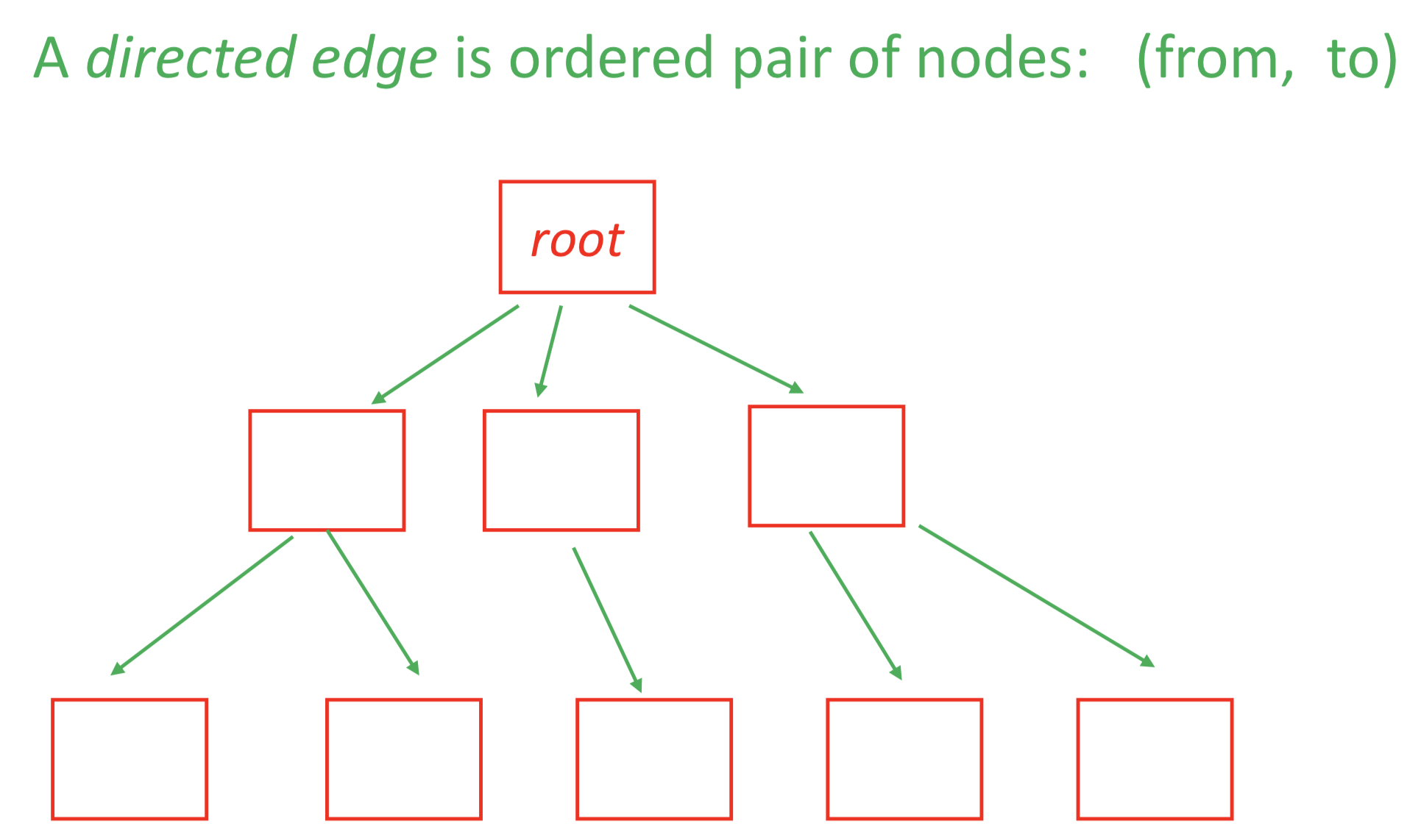
COMP250-Tree



siblings

child

parent

For some trees,

• edges are from parent to child (for today)

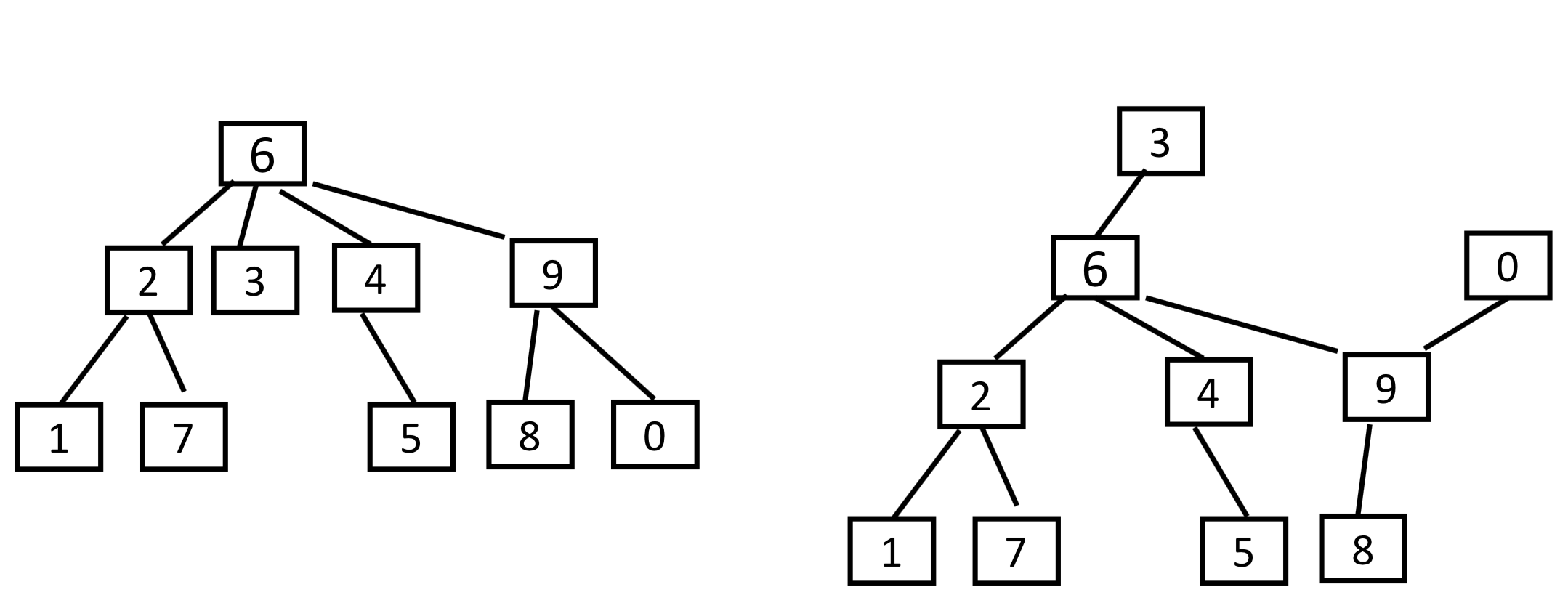
• edges are from child to parent

• edges are both from parent to child and child to parent.

• edge direction is ignored

You will see non-rooted trees mostly commonly when edges are not directed, and there is no natural way to define the ‘root’.

Note the two non-rooted trees below are the same.



If a (rooted) tree has 𝑛 nodes, then how many edges does it have ?

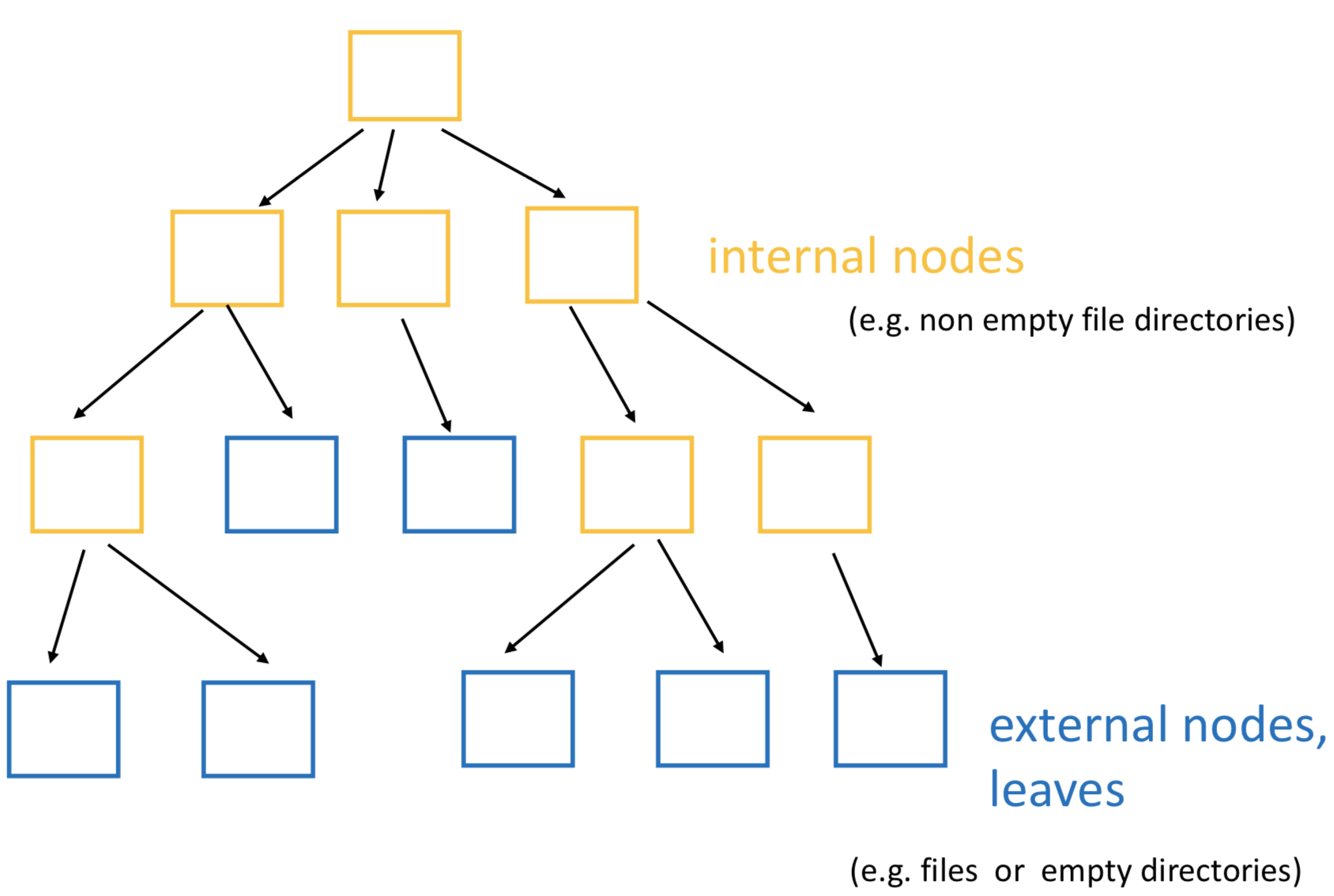
A: 𝑛−1. Since every edge is of the form (parent, child), and each node except the root is a child and each child has exactly one parent.

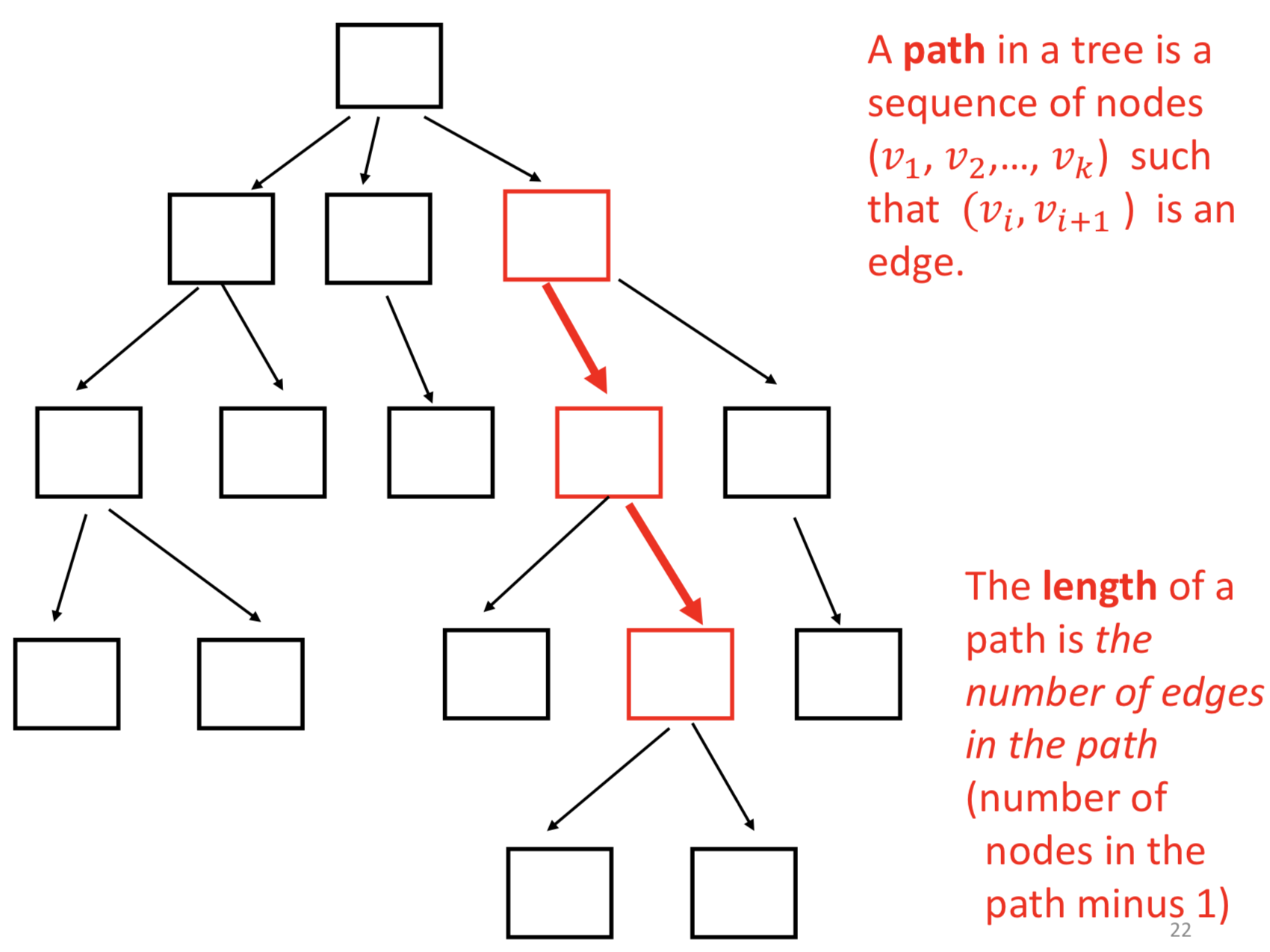
[Definition]

*A tree T is a finite set of 𝑛 ≥ 0 nodes such that:*

*• if 𝑛 > 0 then one of the nodes is the root r*

*• if 𝑛>1 then the 𝑛−1 non-root nodes are partitioned into non- empty subsets T1, T2, ..., Tk, each of which is a tree (called a subtree”), and the roots of the subtrees are the children of root r.*





A path with just one node (𝑣1) has length = 0, since it has no edges.

depth

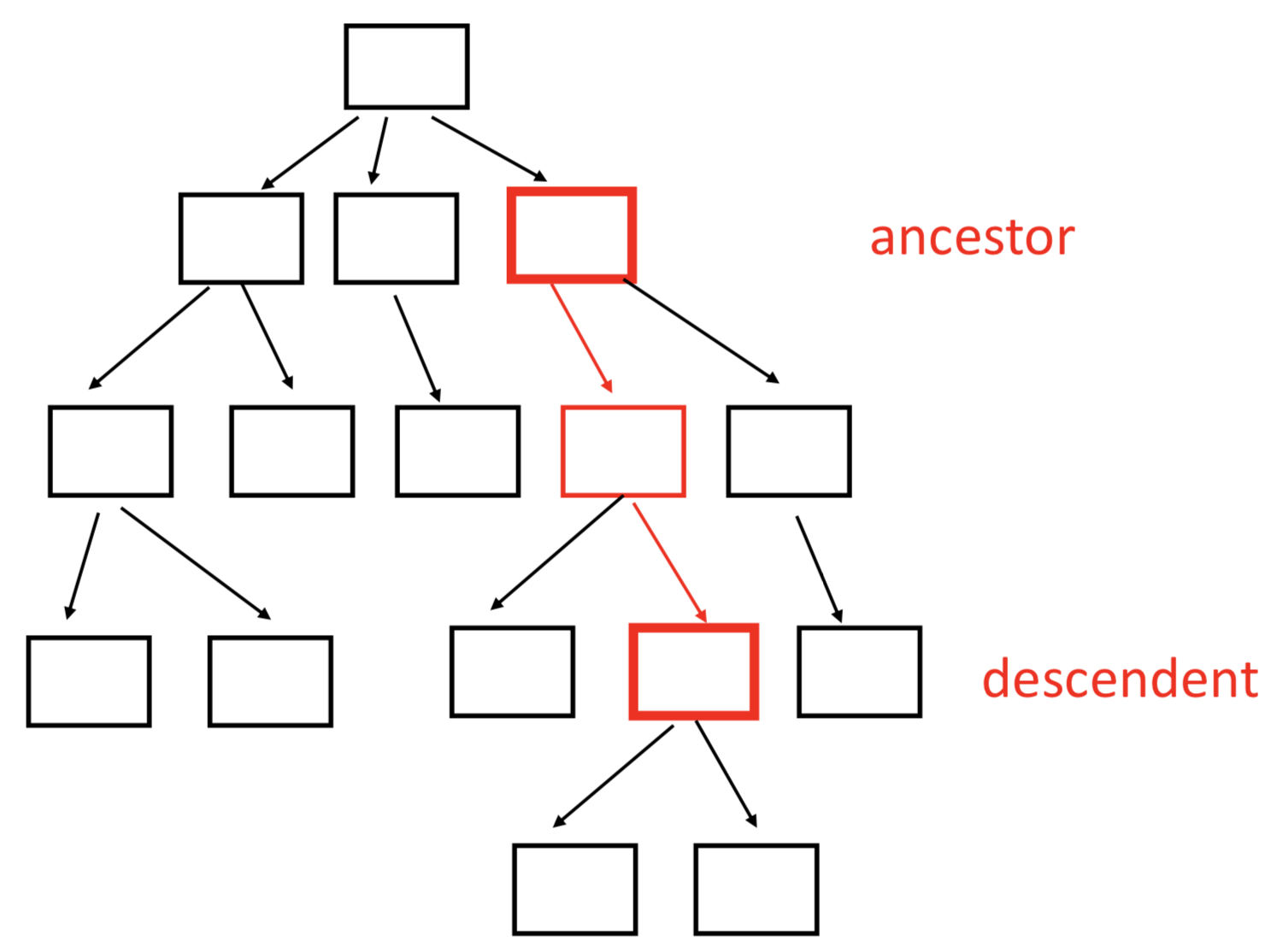
0

1

2

3

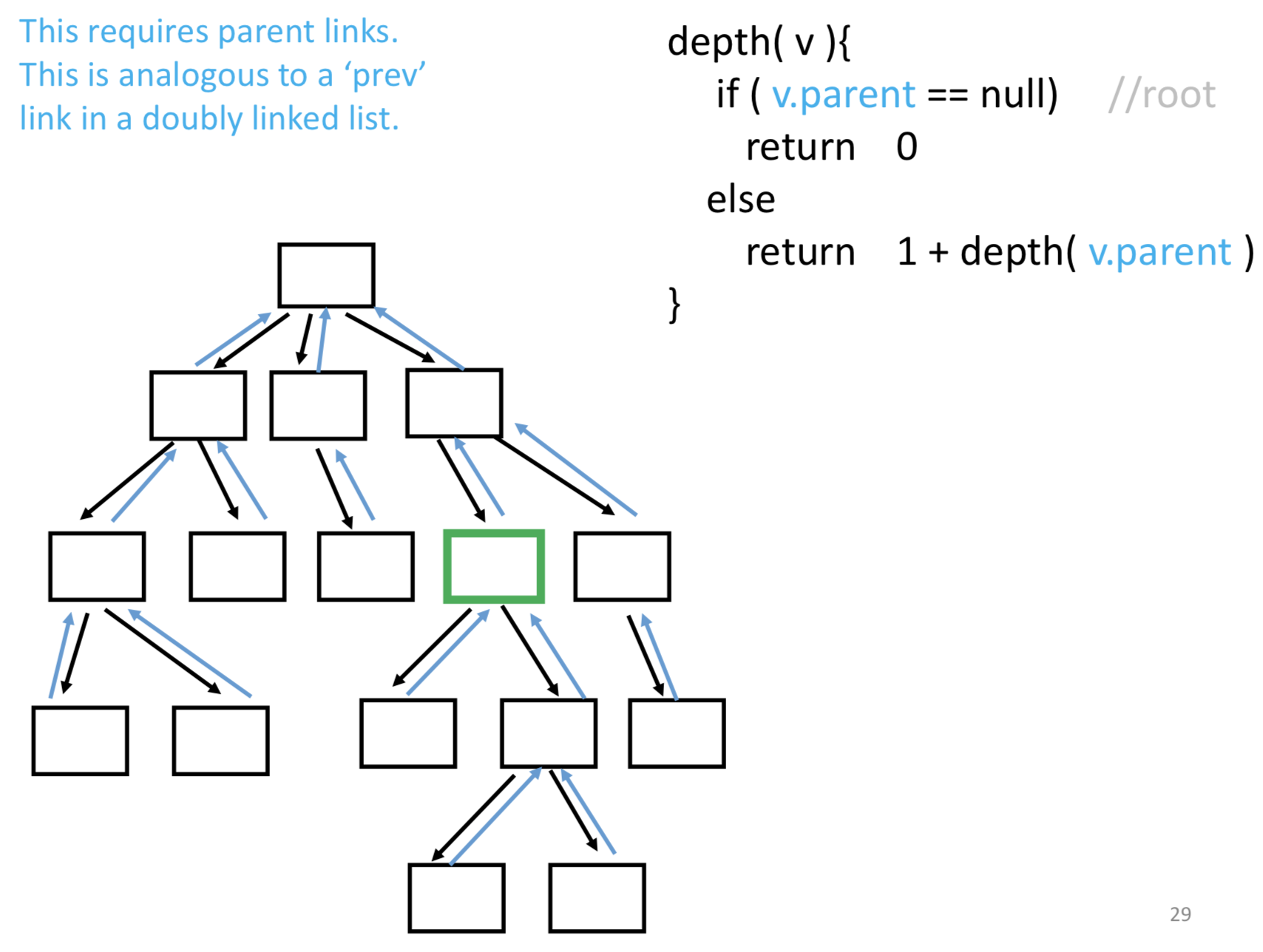
4

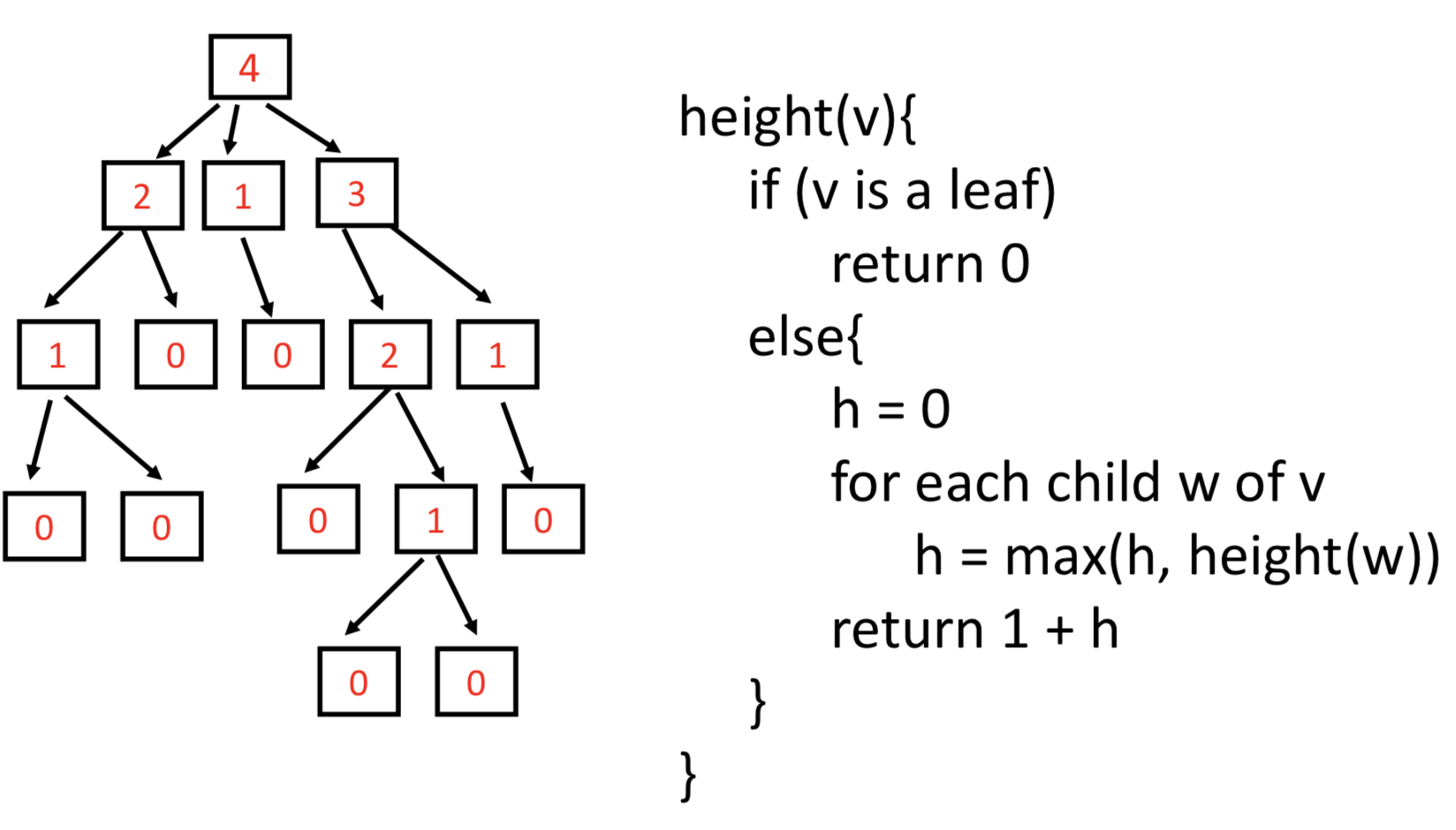


Node w is a descendent of node v.

Node v is an ancestor of node w if there is a path from v to w.

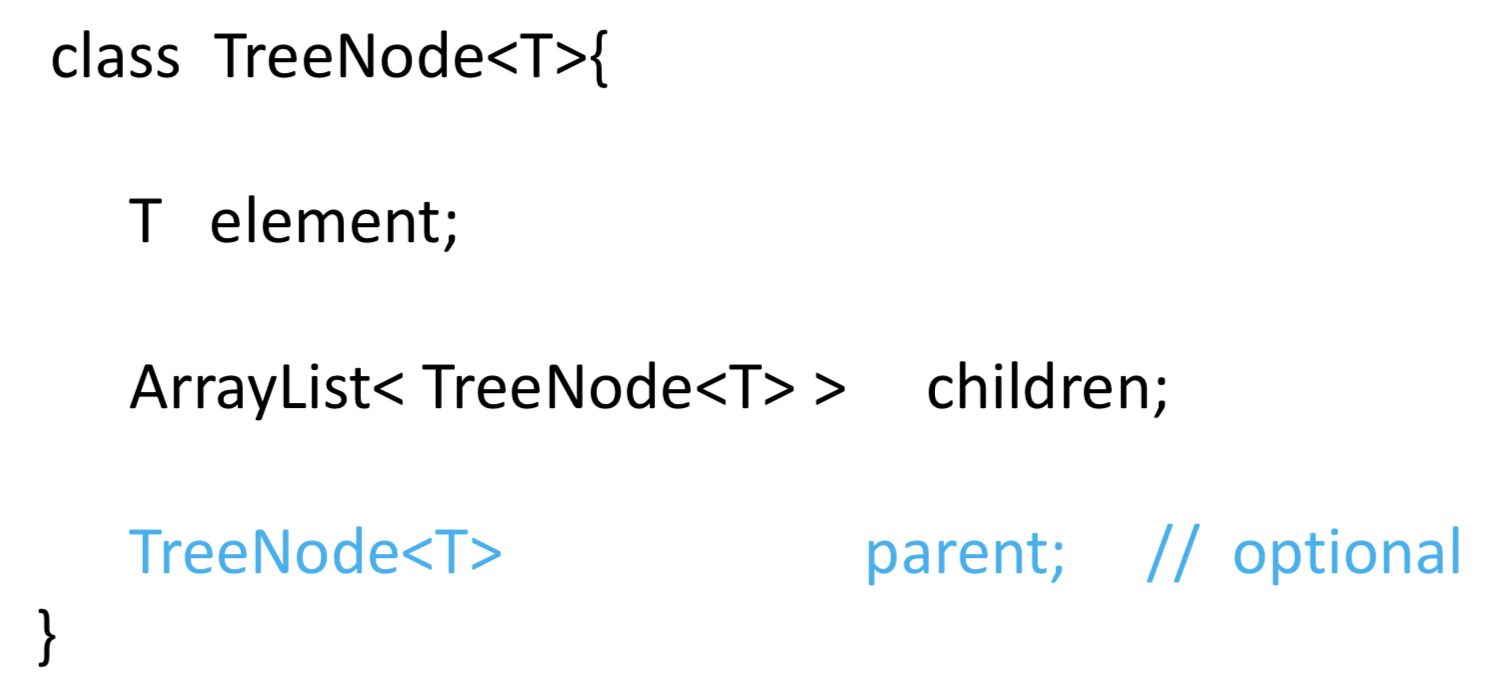
The depth or level of a node is the length of the path from the root to the node.



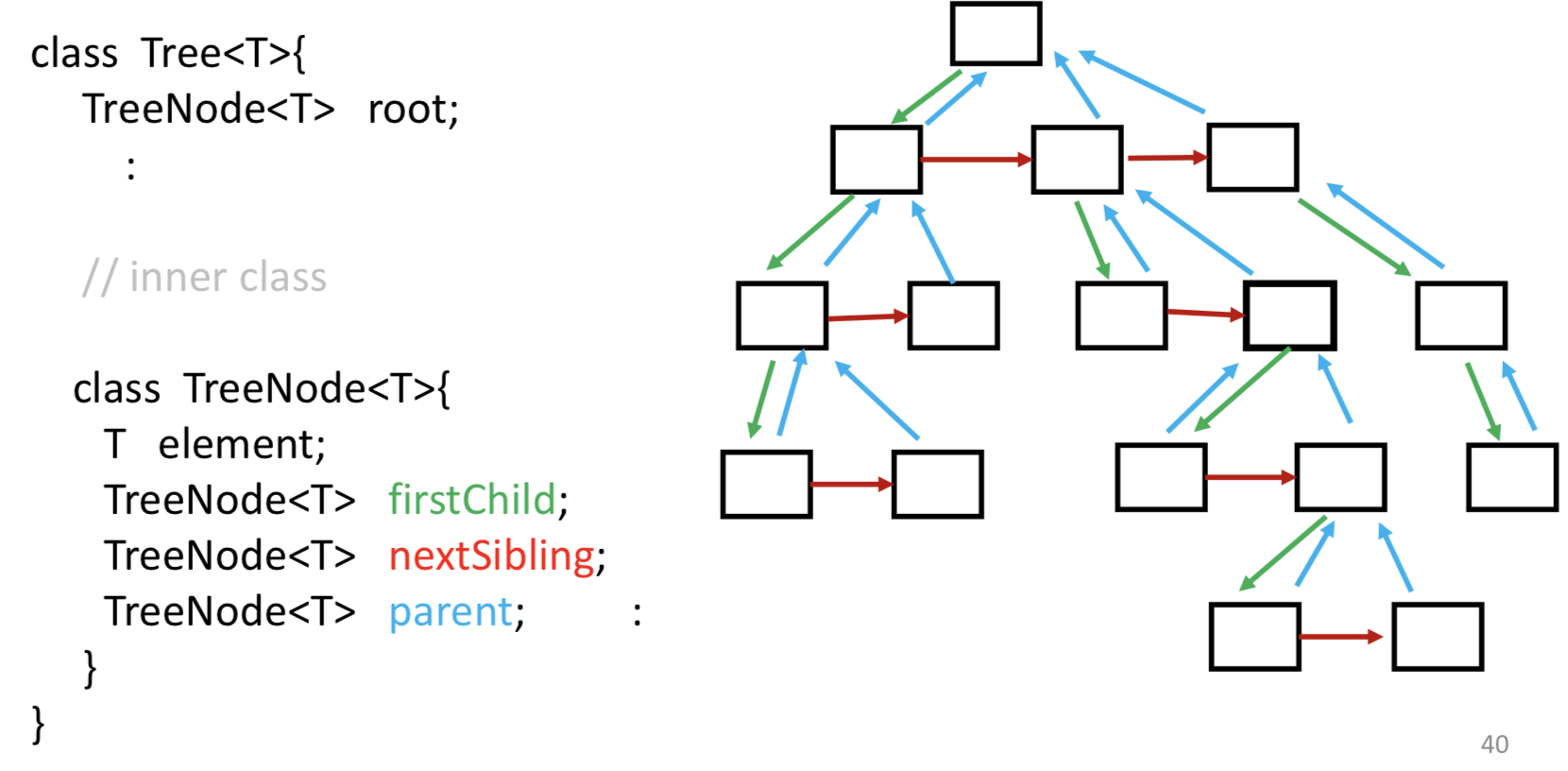


The height of a node is the maximum length of a path from that node to a leaf.

**[implementation 1]**



**[implementation 2]** ‘first child, next sibling’



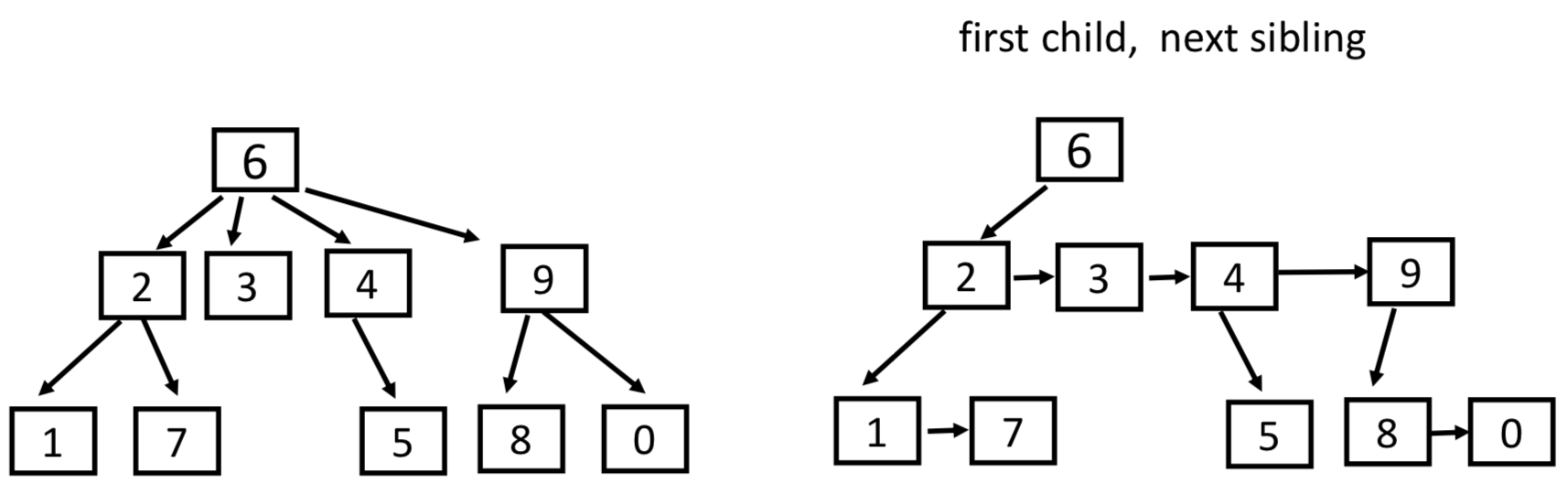
A tree can be represented using lists, as follows:

tree = root | ( root listOfSubTrees )

listOfSubTrees = tree | tree listOfSubTrees

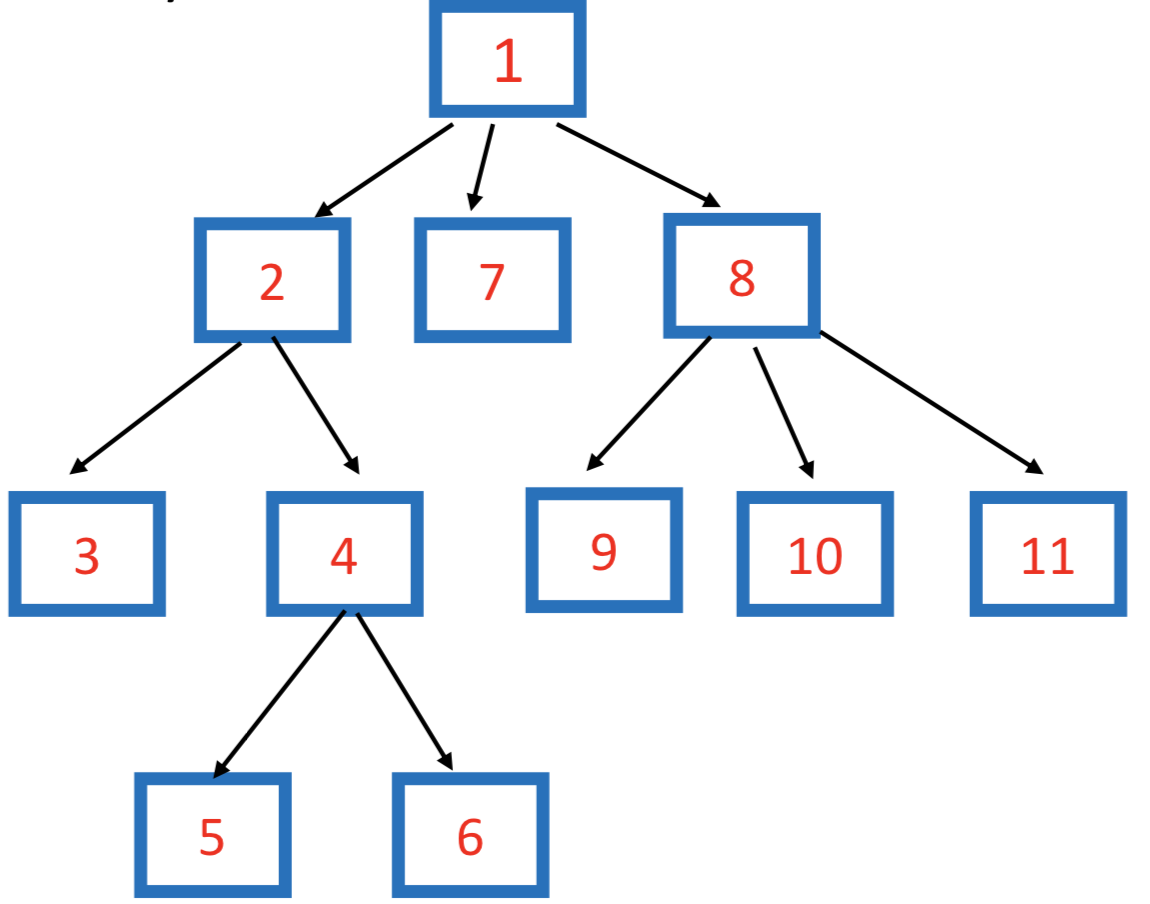
Note that listOfSubTrees cannot be empty i.e. ( ) is not allowed.

EX. ( 6 (2 1 7 ) 3 ( 4 5 ) ( 9 8 0 ) )



Tree Traversal

***Recursion***

* ***“preorder” traversal:***

*visit the root before the children*

depthfirst (root){

if (root is not empty){

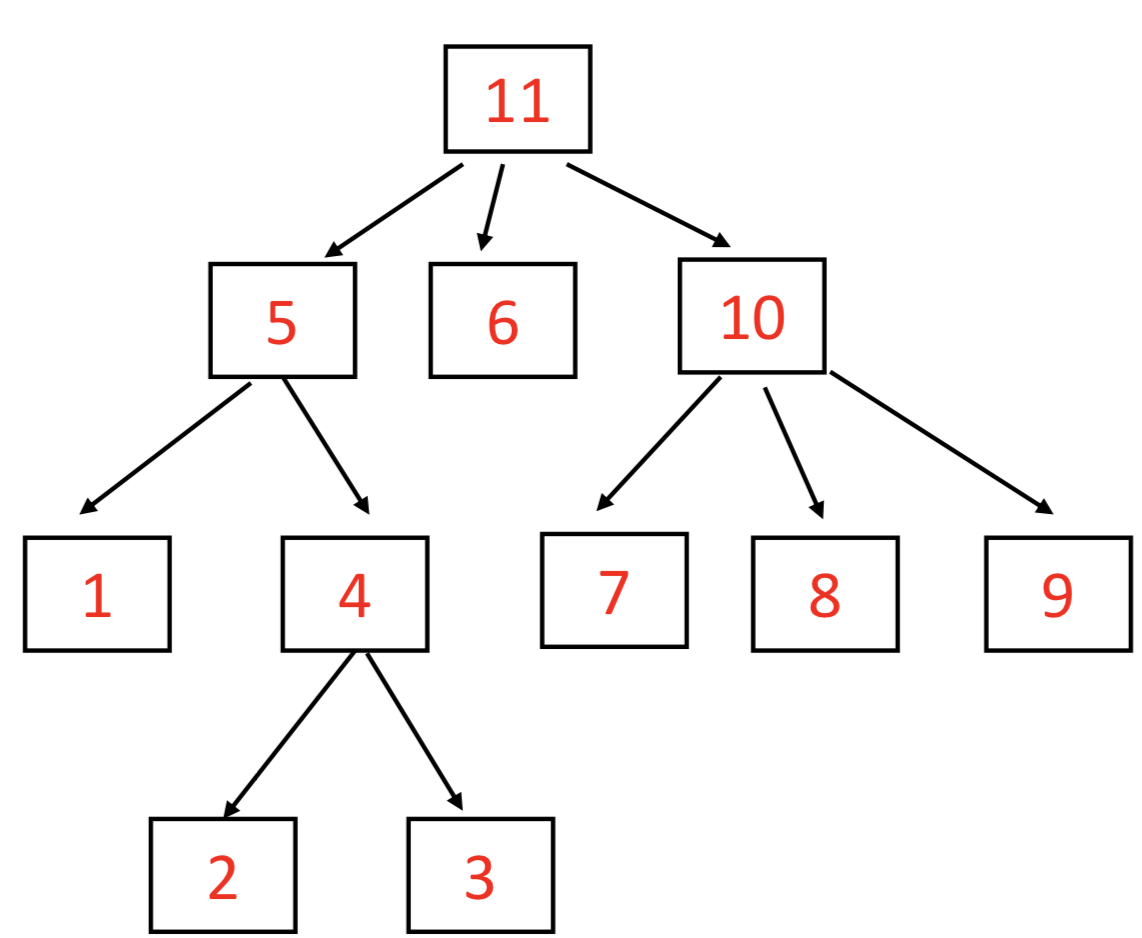
visit root

for each child of root

depthfirst(child)

}

}

* *****“postorder” traversal:***

*visit the root after the children*

depthfirst (root){

if (root is not empty){

for each child of root

depthfirst(child)

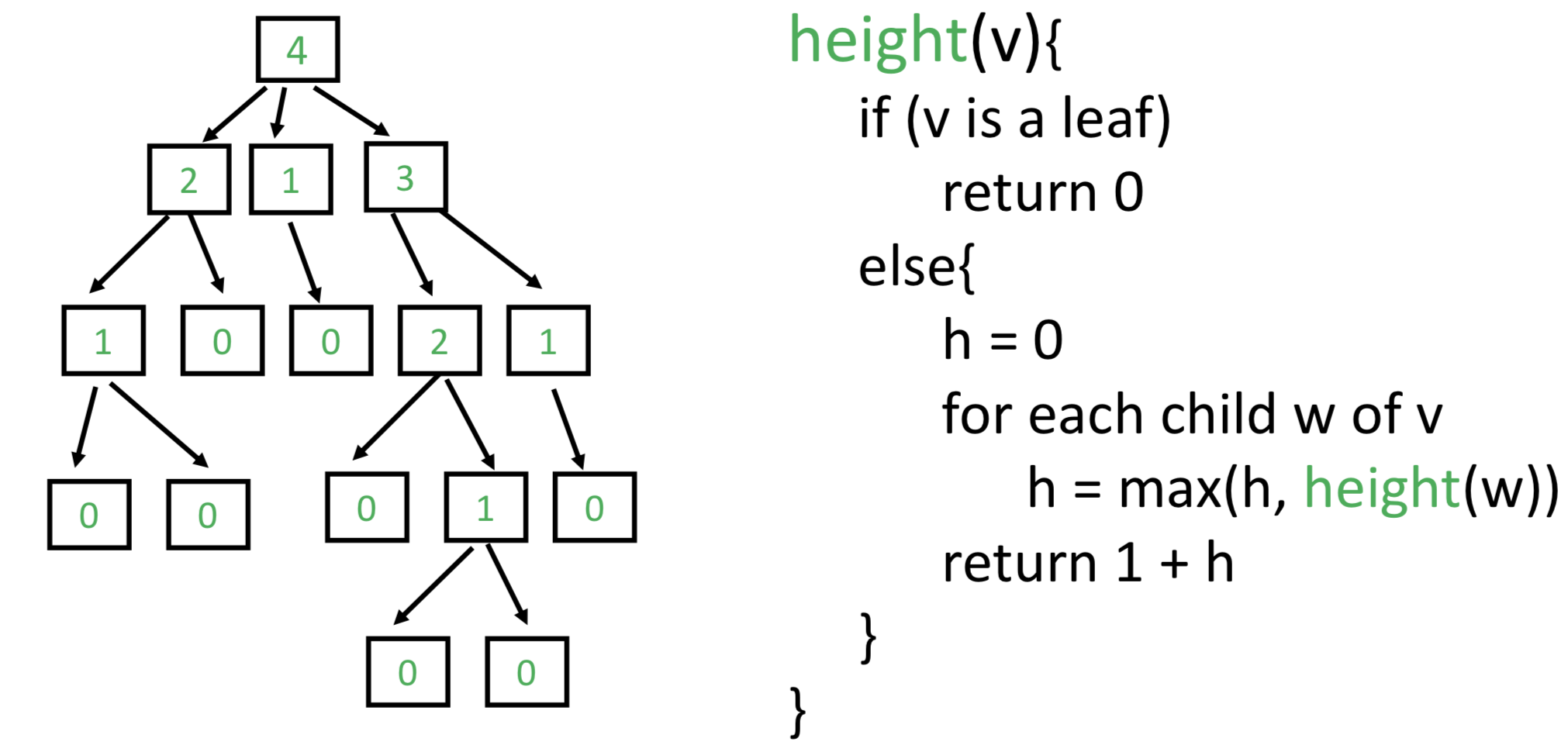
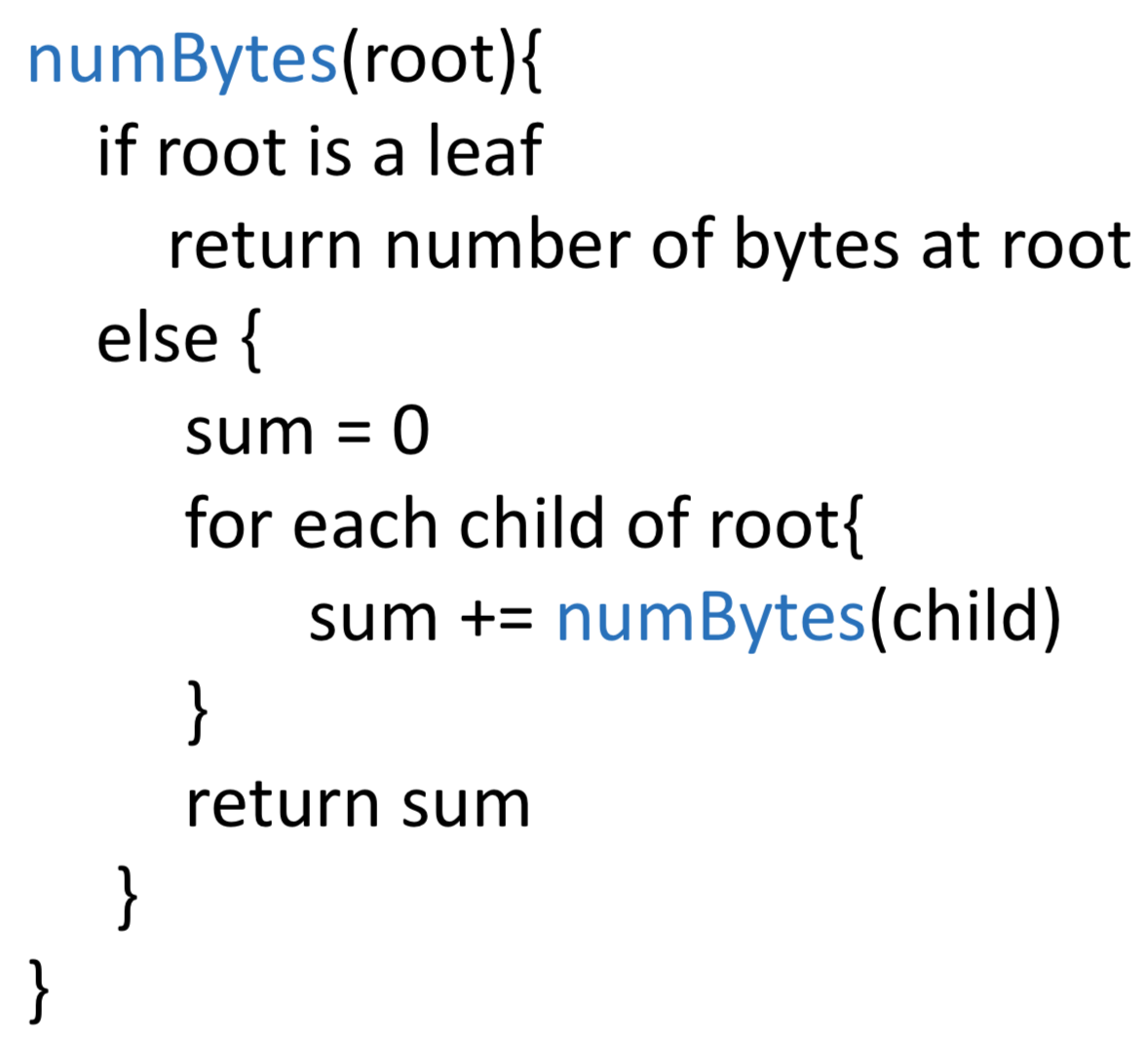
visit root

}

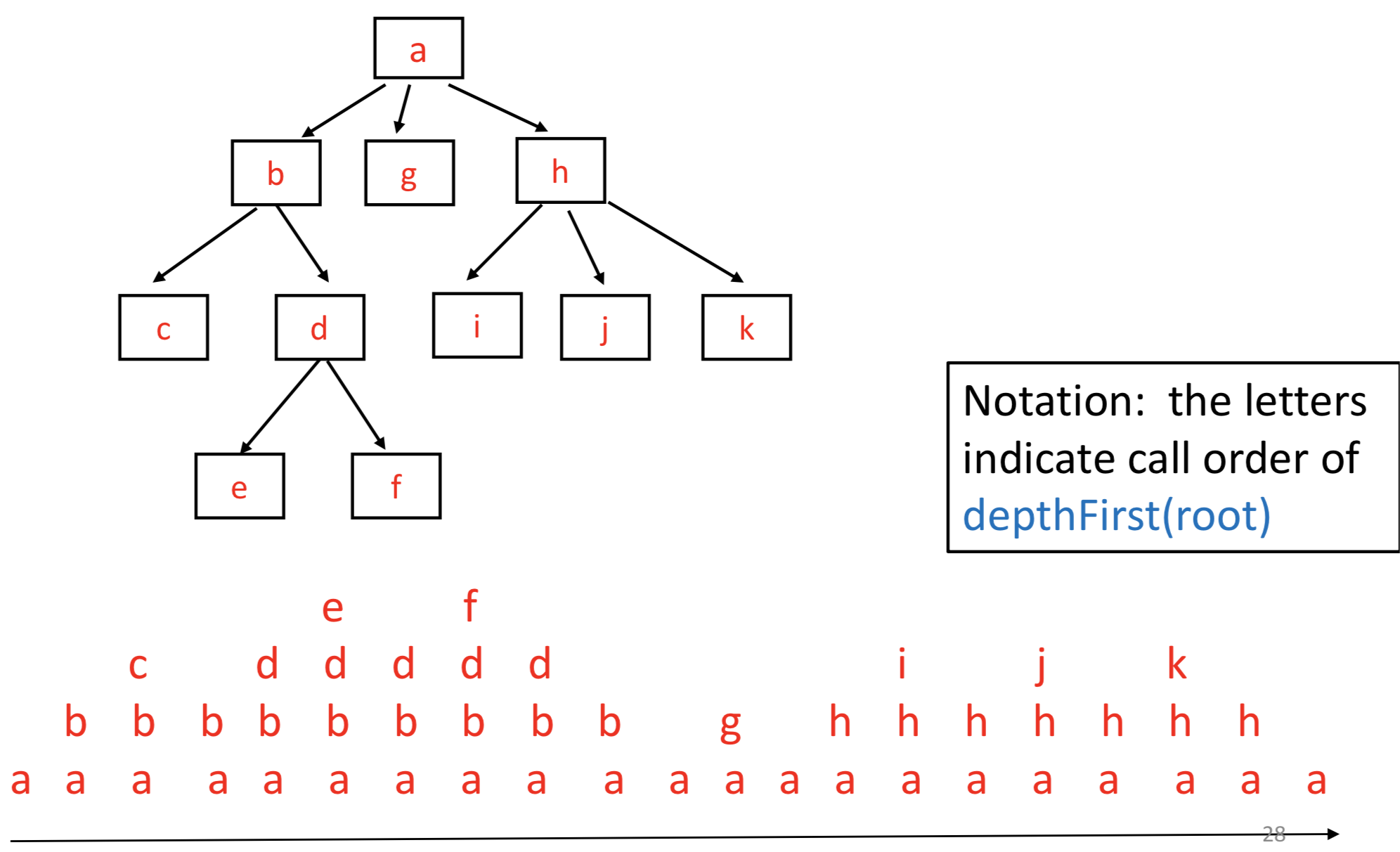
}

<postorder: What is the total number of bytes in all files in a directory?>

<postorder-height(v)>

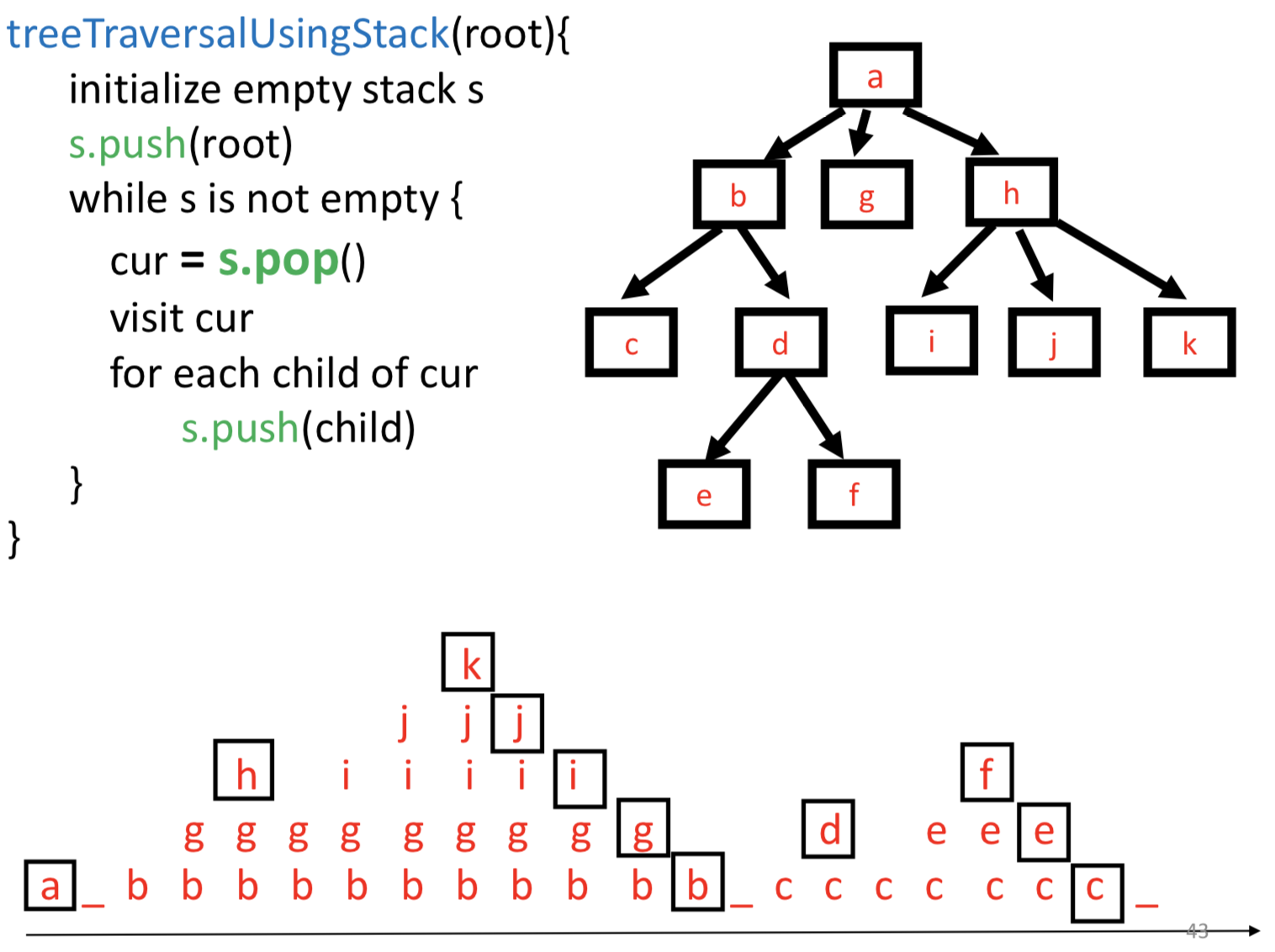
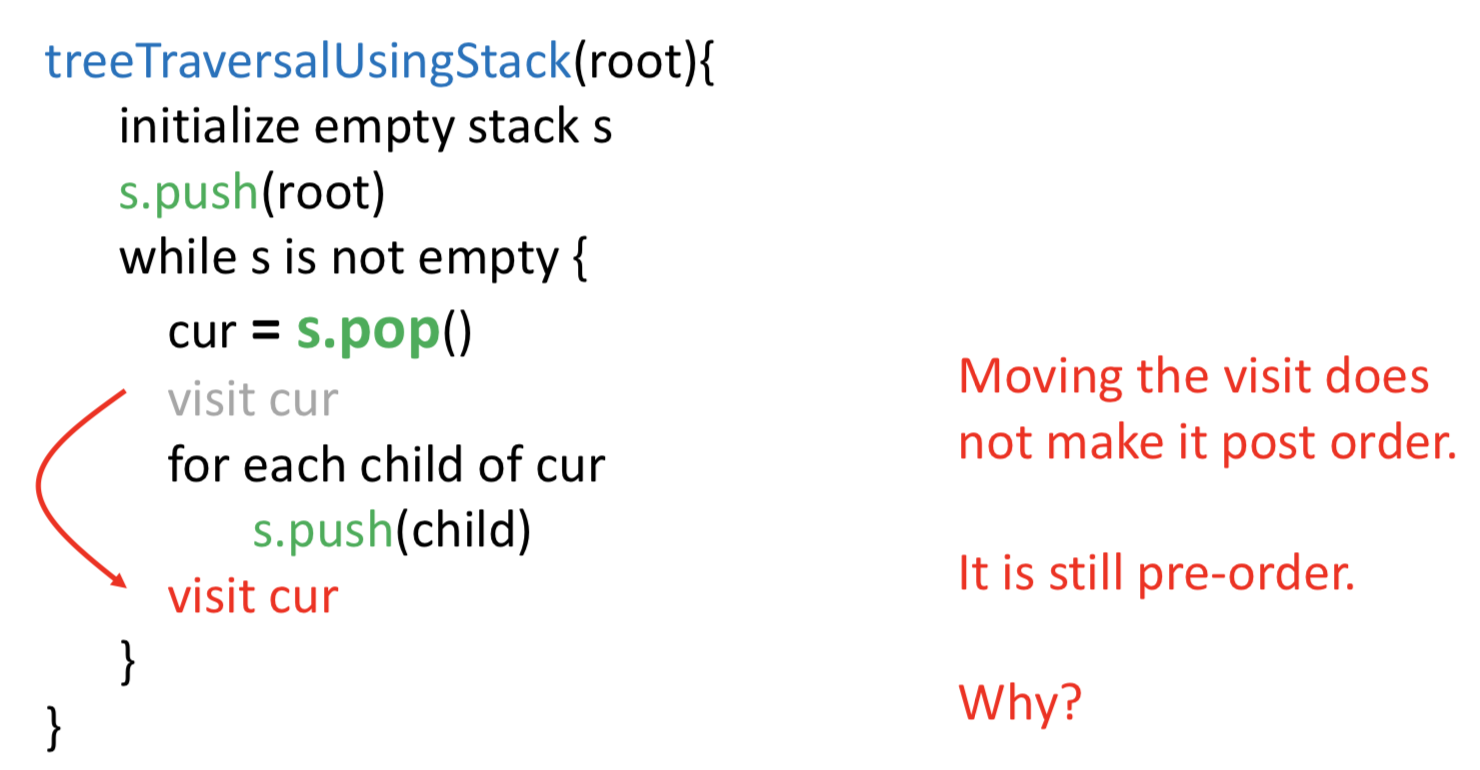
[Call stack for depthfirst(root)]



***Iterative***

* ***stack***

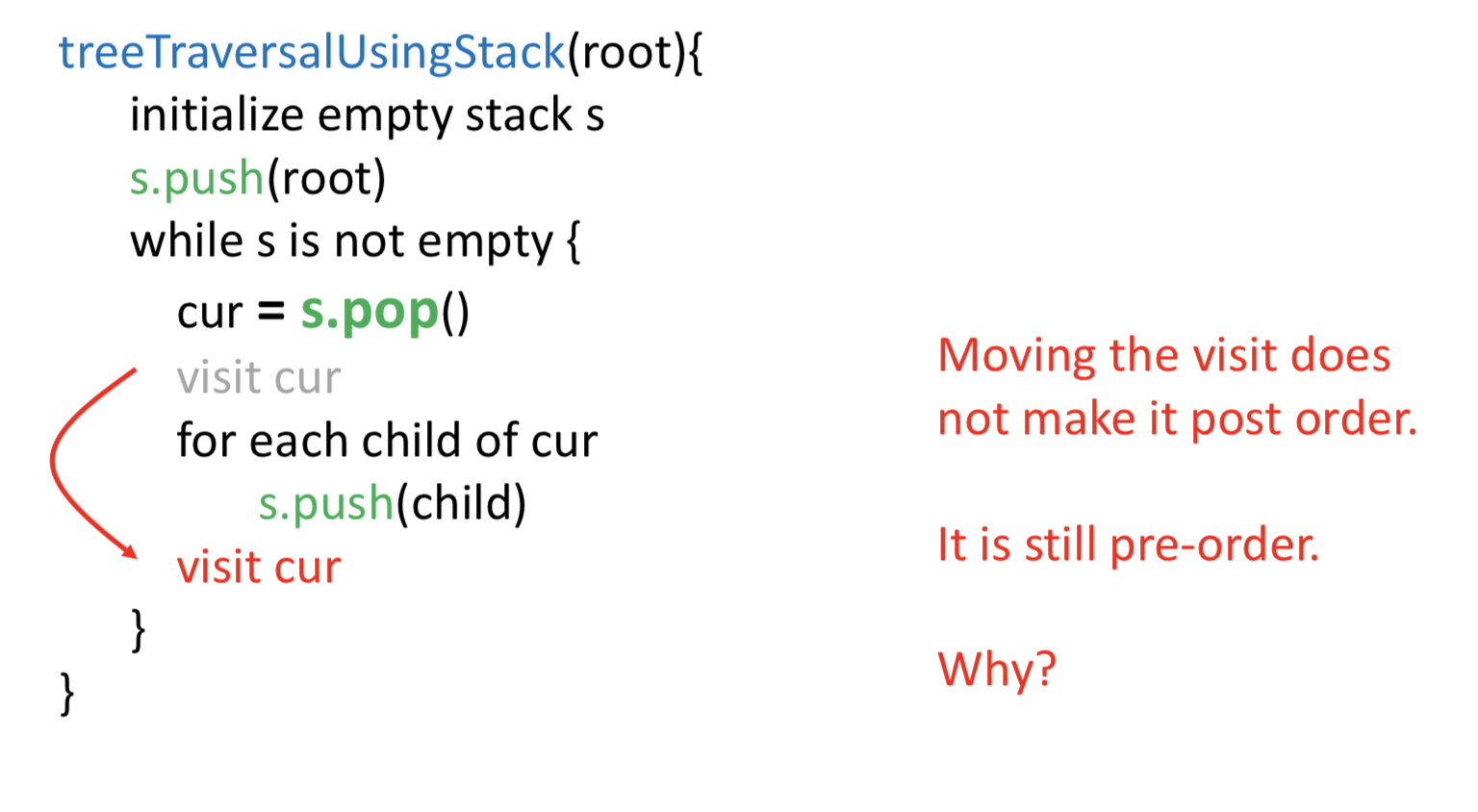
*Stack based method is also depth first, but visits children from right to left*



* ***queue***

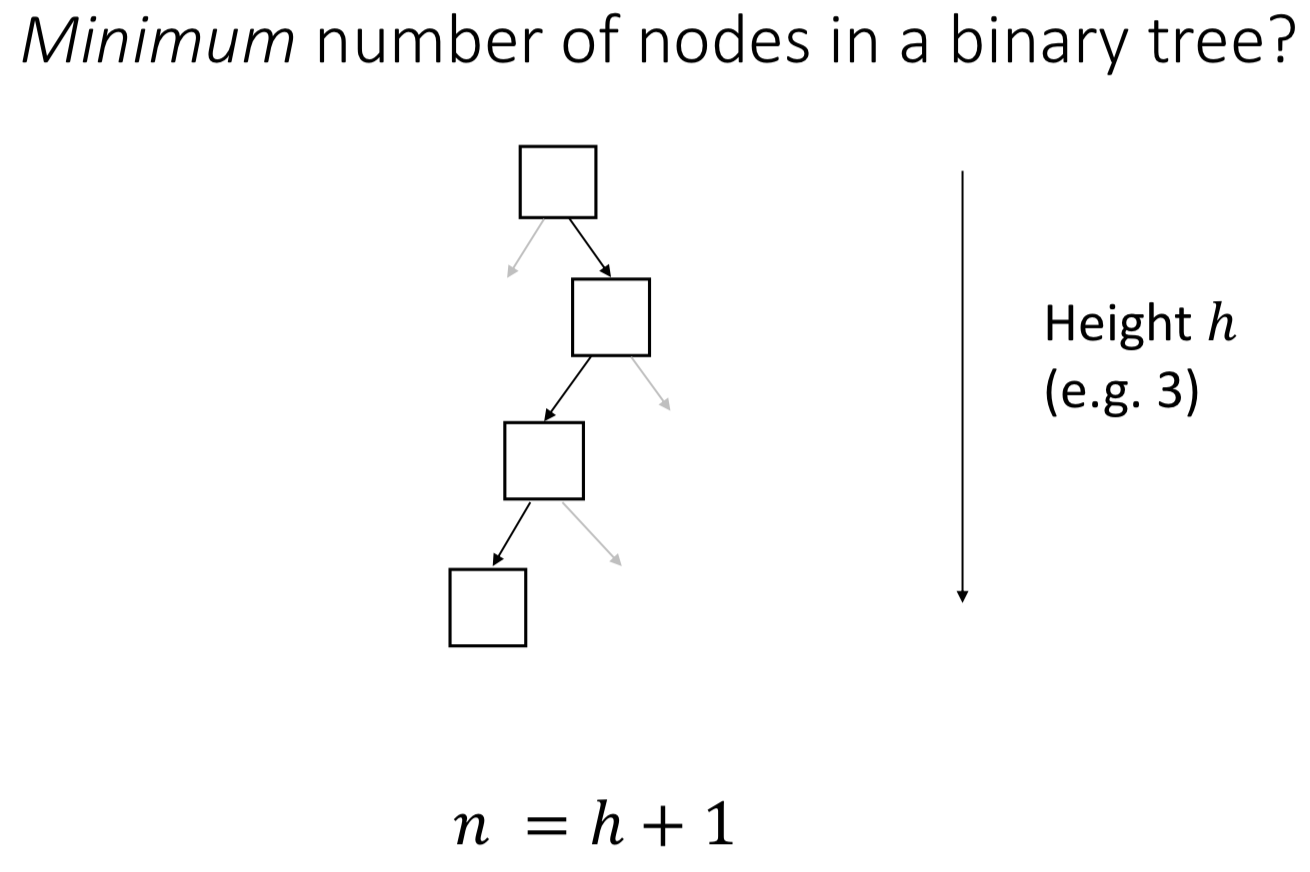
*breadth first traversal: for each level I, visit all nodes at level i*

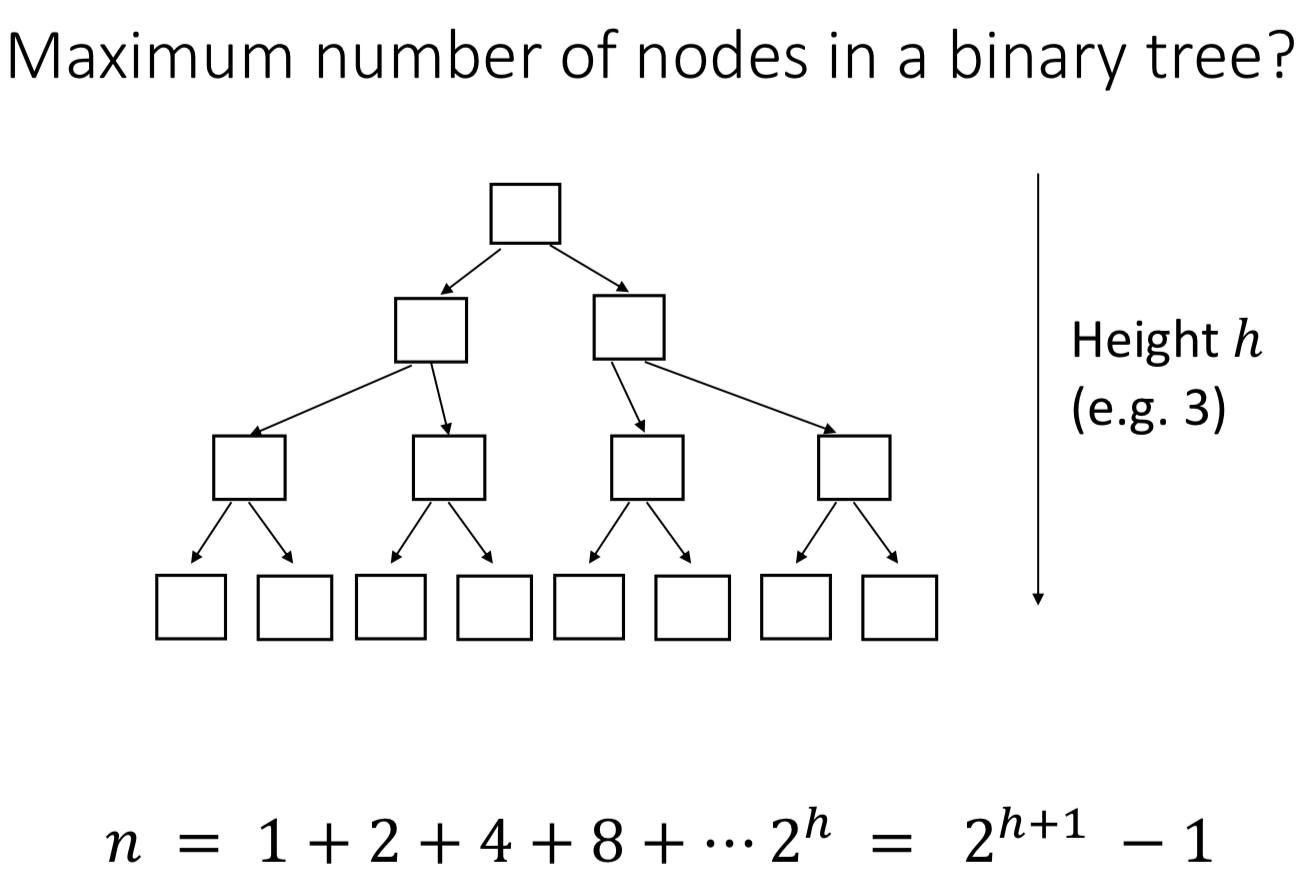


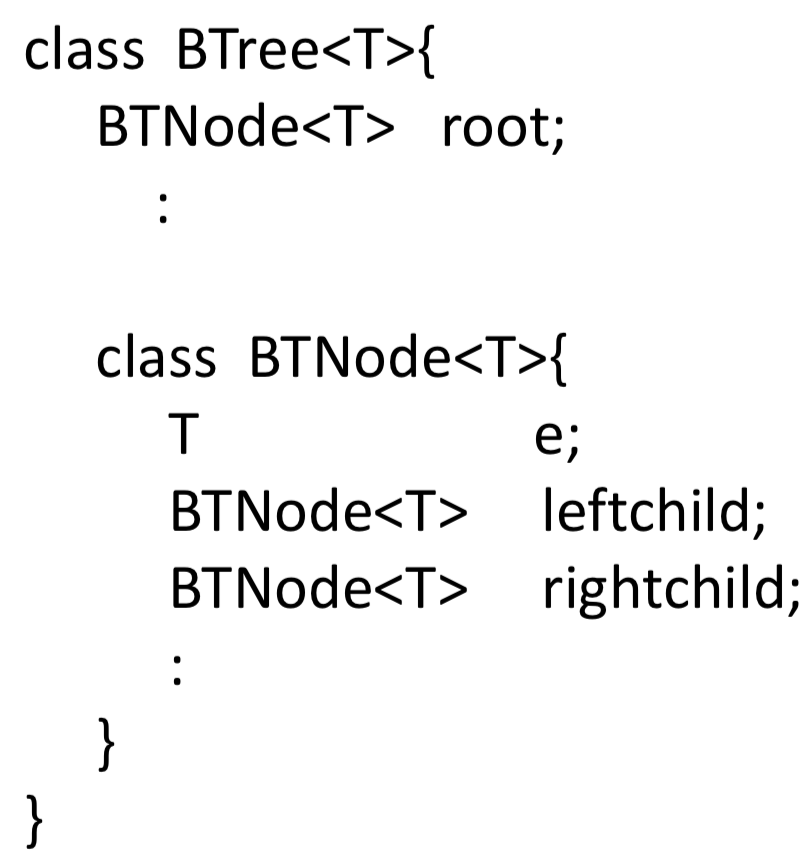


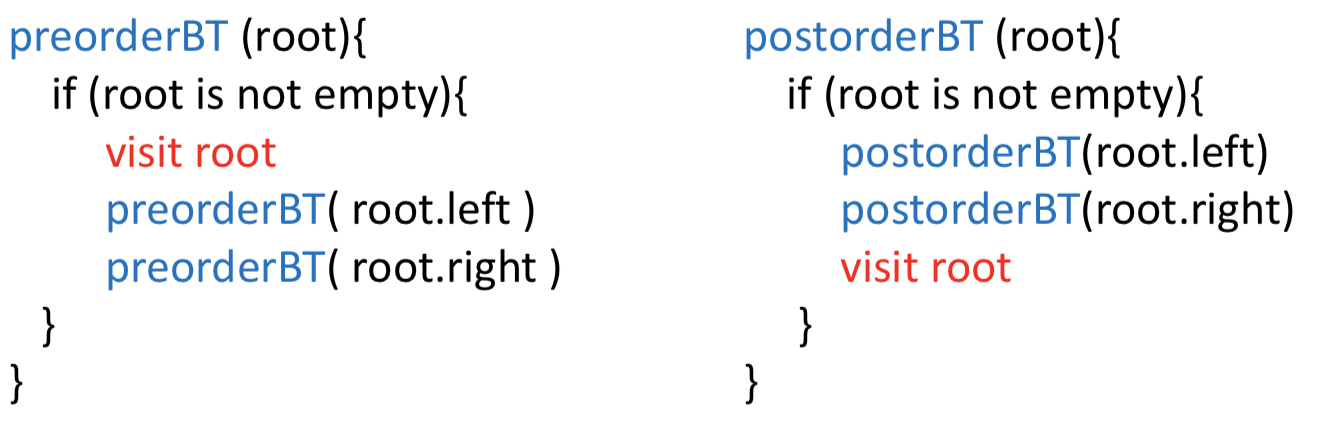
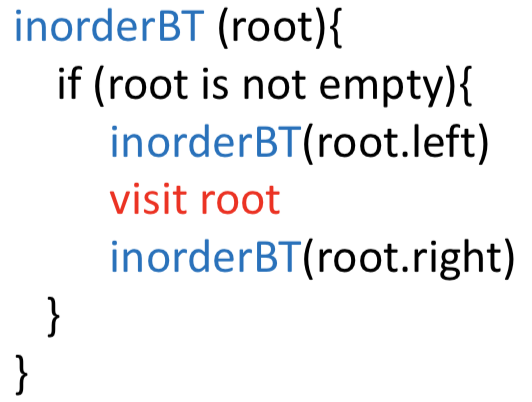
Binary Tree

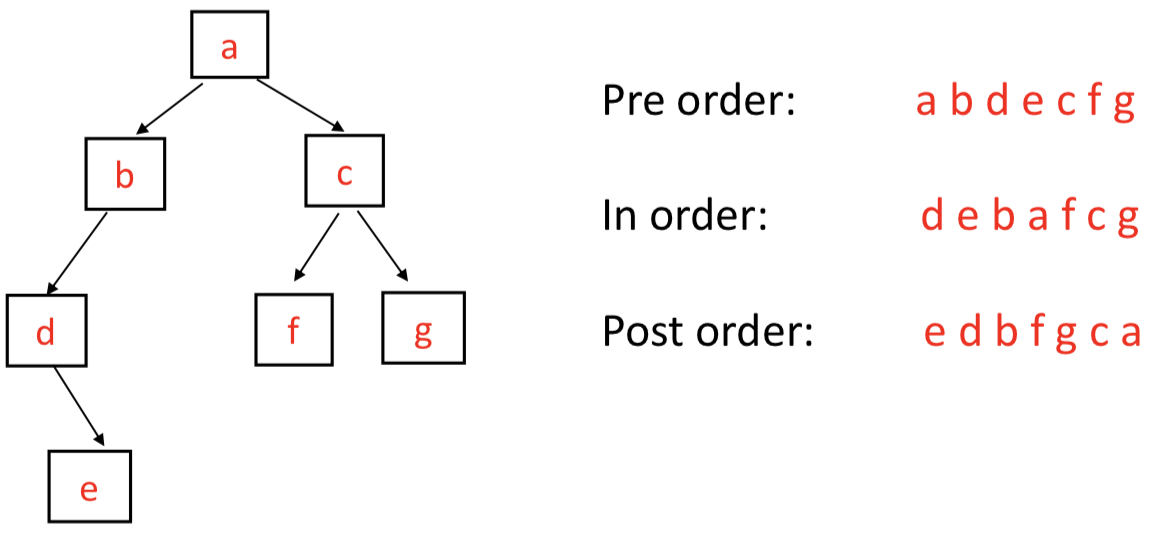
*Each node has at most two children.*



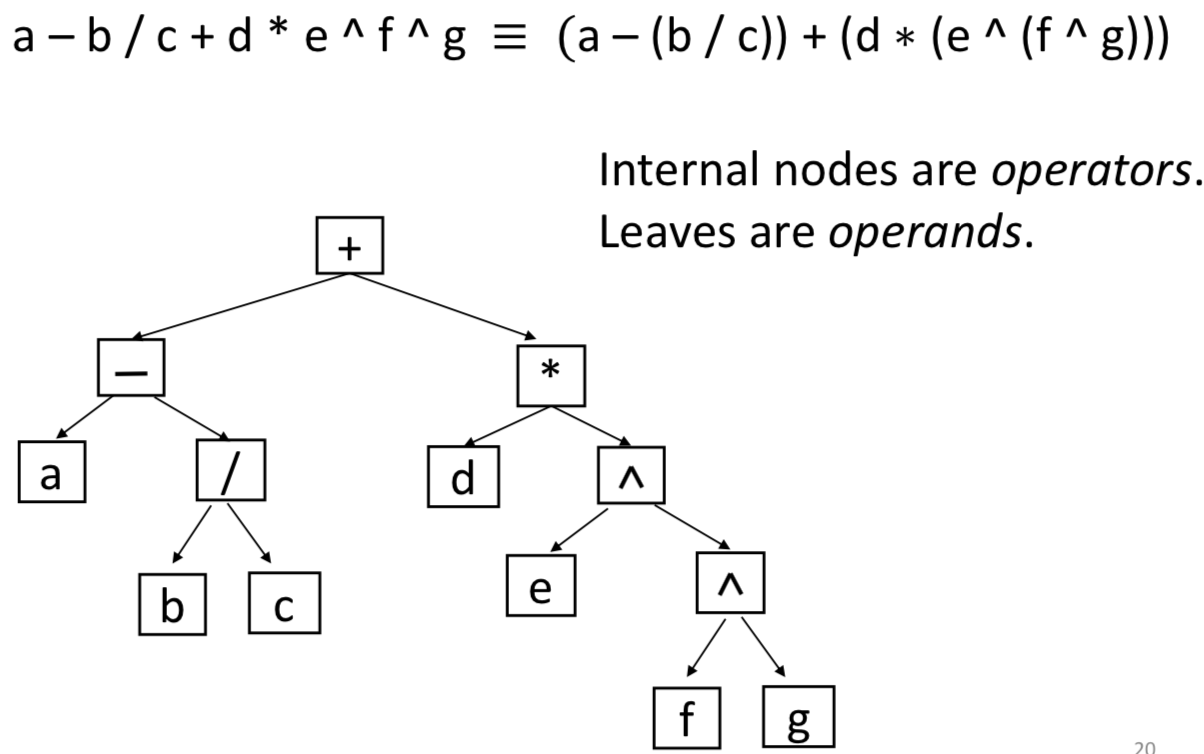








Expression Tree

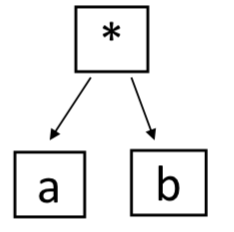


preorder traversal : + – a / b c \* d ^ e ^ f g

inorder traversal gives : a – b / c + d \* e ^ f ^ g

postorder traversal gives : a b c / - d e f g ^ ^ \* +

**[Prefix, infix, postfix expressions]**



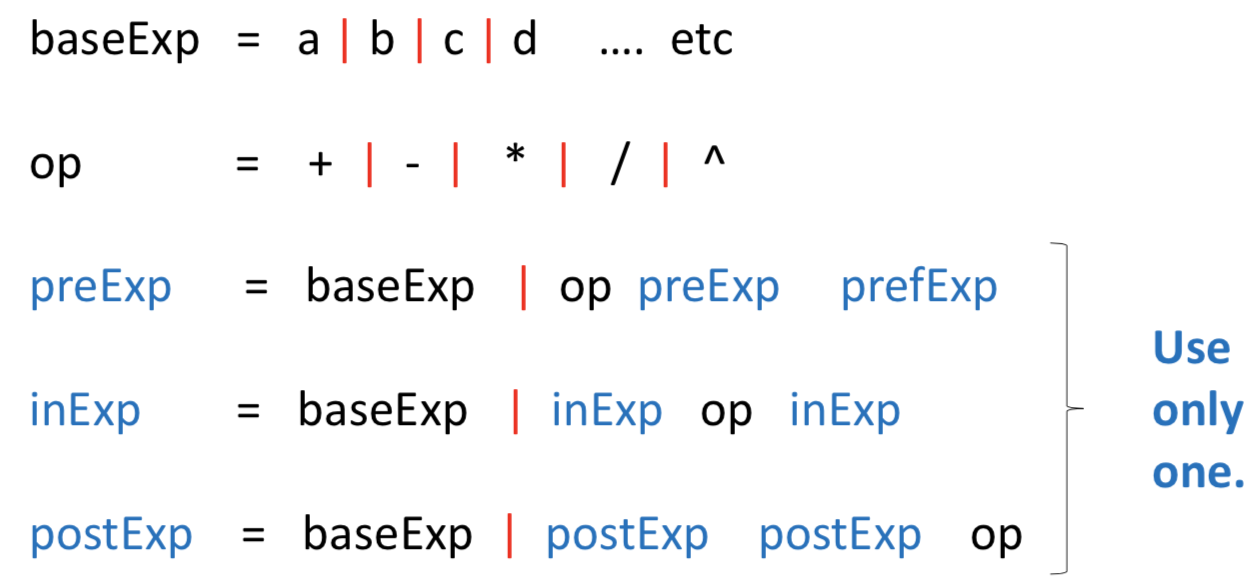
prefix: \* a b

infix: a \* b

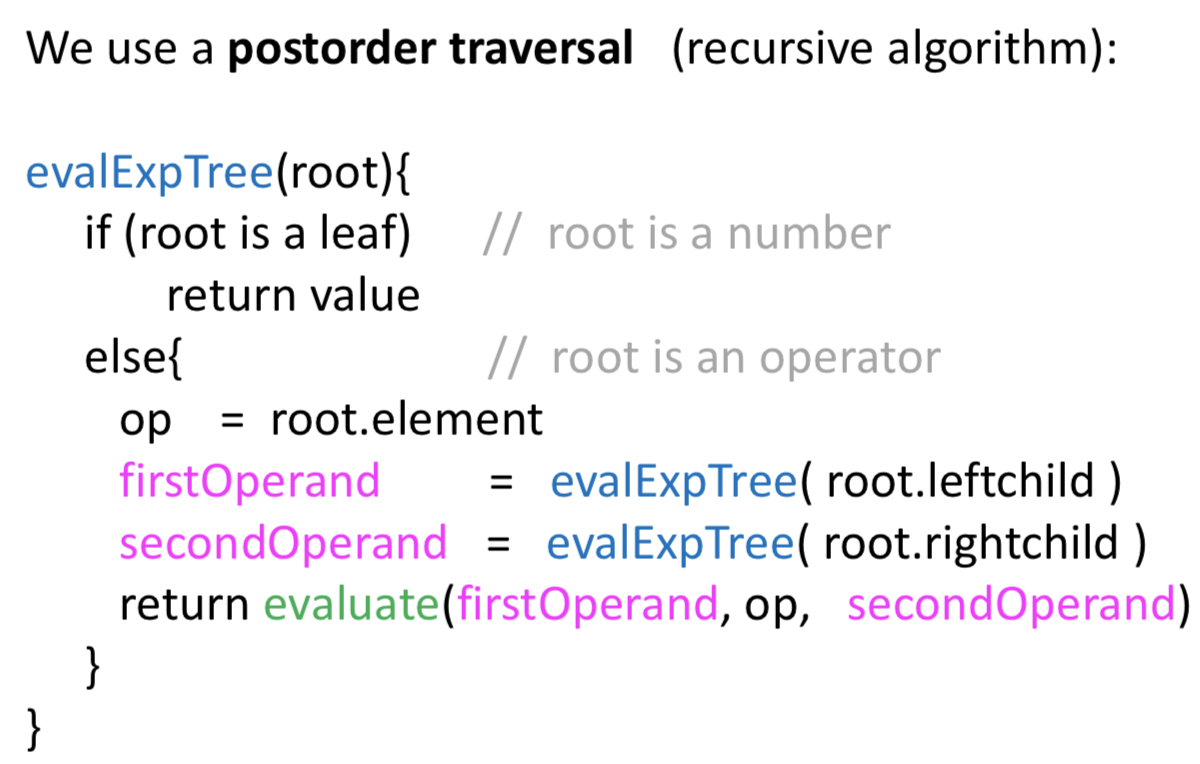
postfix: a b \*

* Prefix expressions called “Polish Notation”
* Postfix expressions are called “Reverse Polish notation” (RPN)

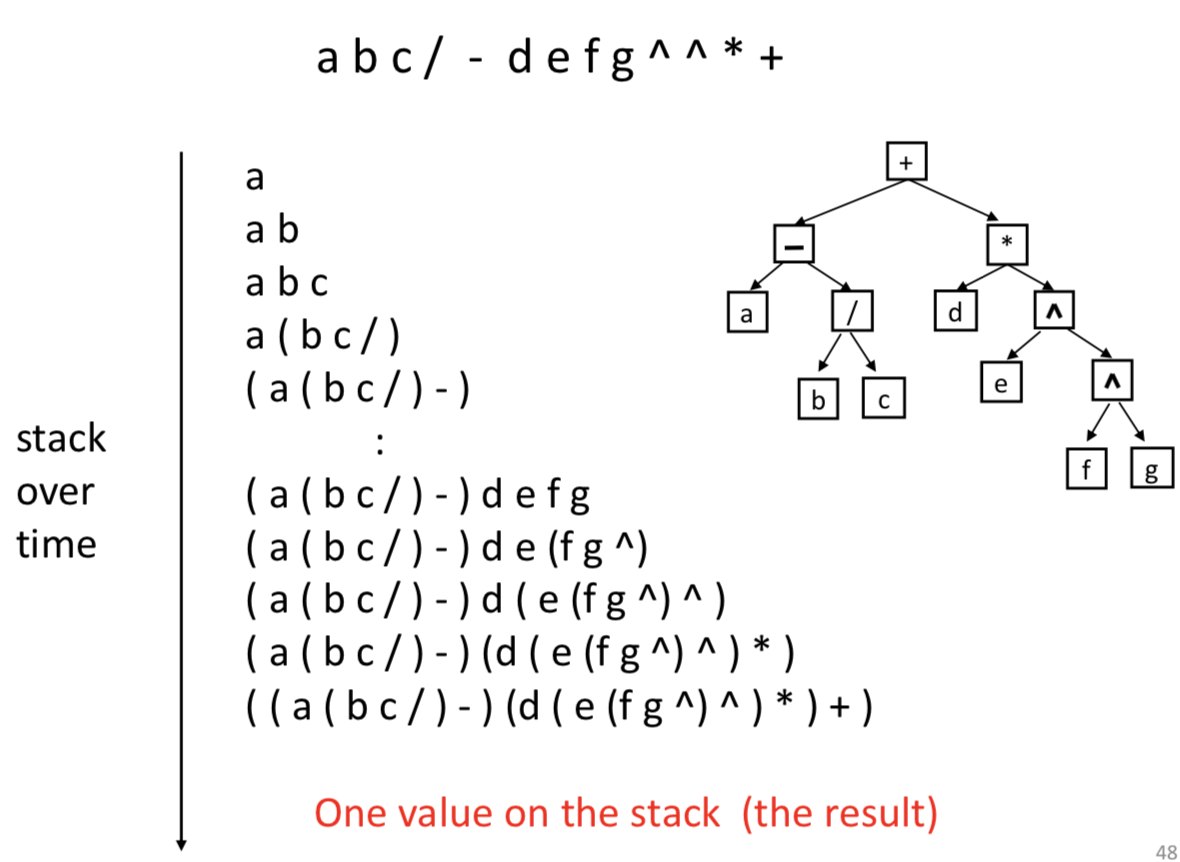
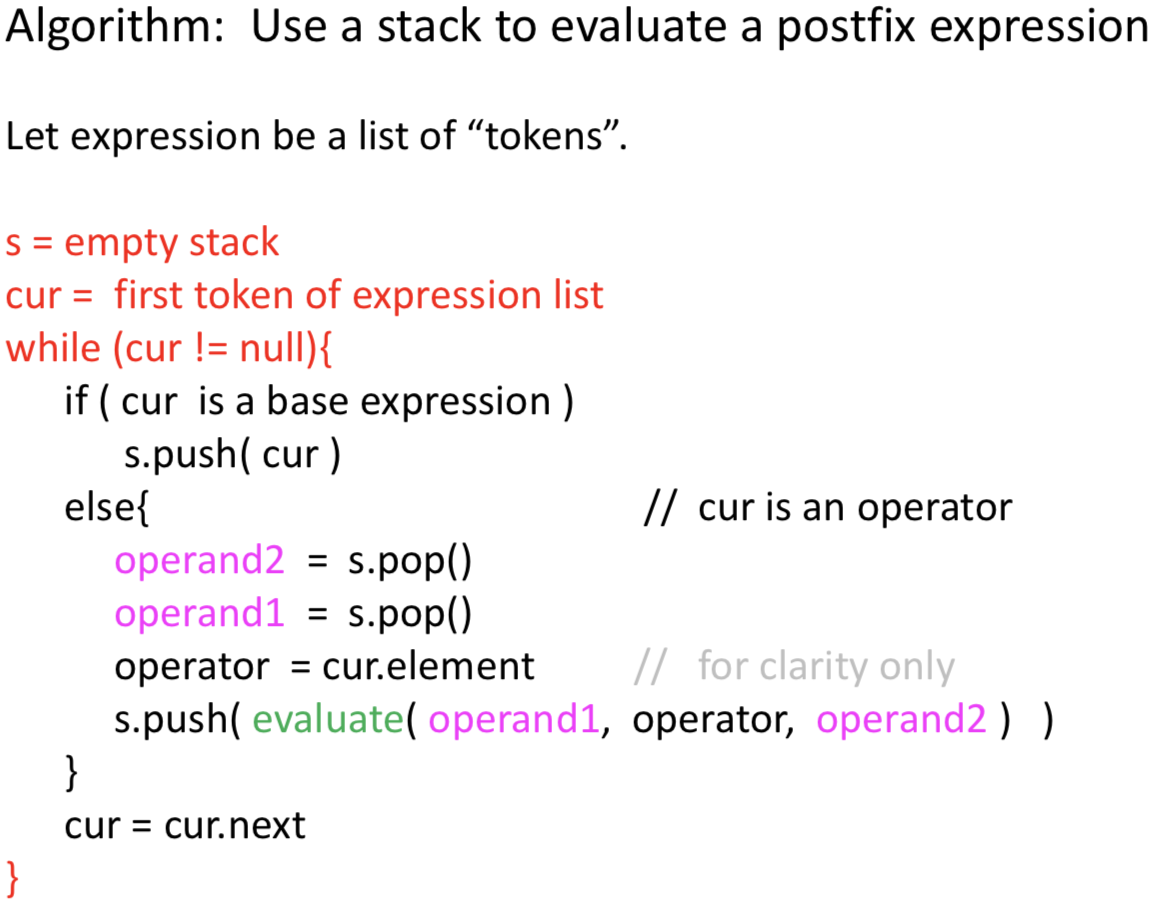
Some calculators (esp. Hewlett Packard) require users to input expressions using RPN.



**[Evaluation - postorder traversal(recursive)]**

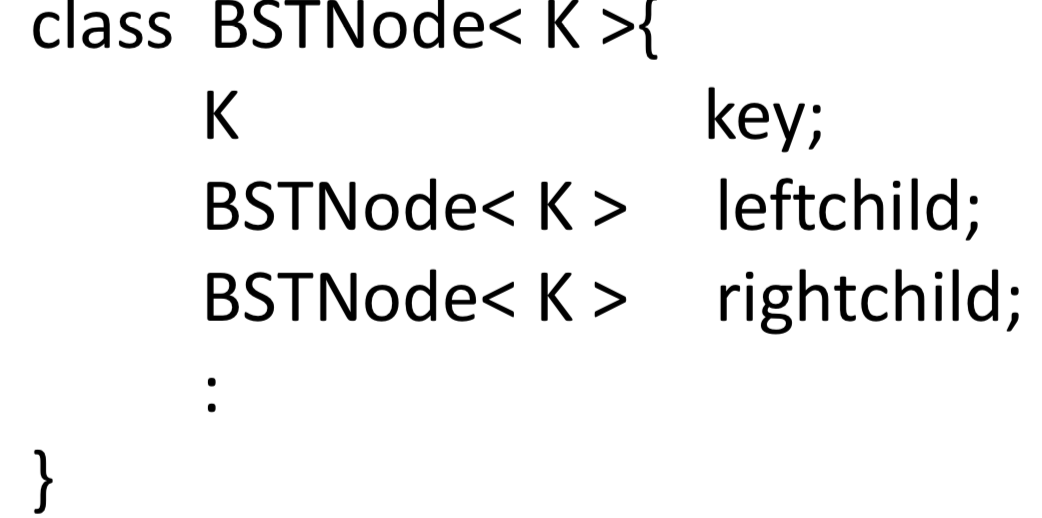


Note: When you write an expression in Java, you use infix. The computer needs to evaluate that expression. How does it do so? The answer is that the compiler converts the infix expression into a postfix (or prefix) expression.

Binary Search Tree

The keys are “comparable” <, =, > e.g. numbers, strings.

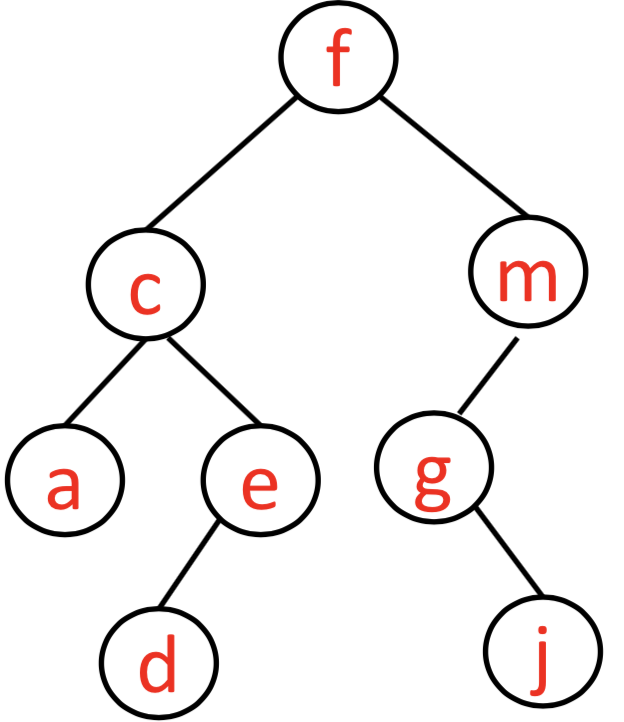


**[Binary Search Tree Definition]**

* binary tree
* keys are comparable, unique (no duplicates)
* for each node, all descendents in left subtree are less than the node, and all descendents in the node’s right subtree are greater than the node

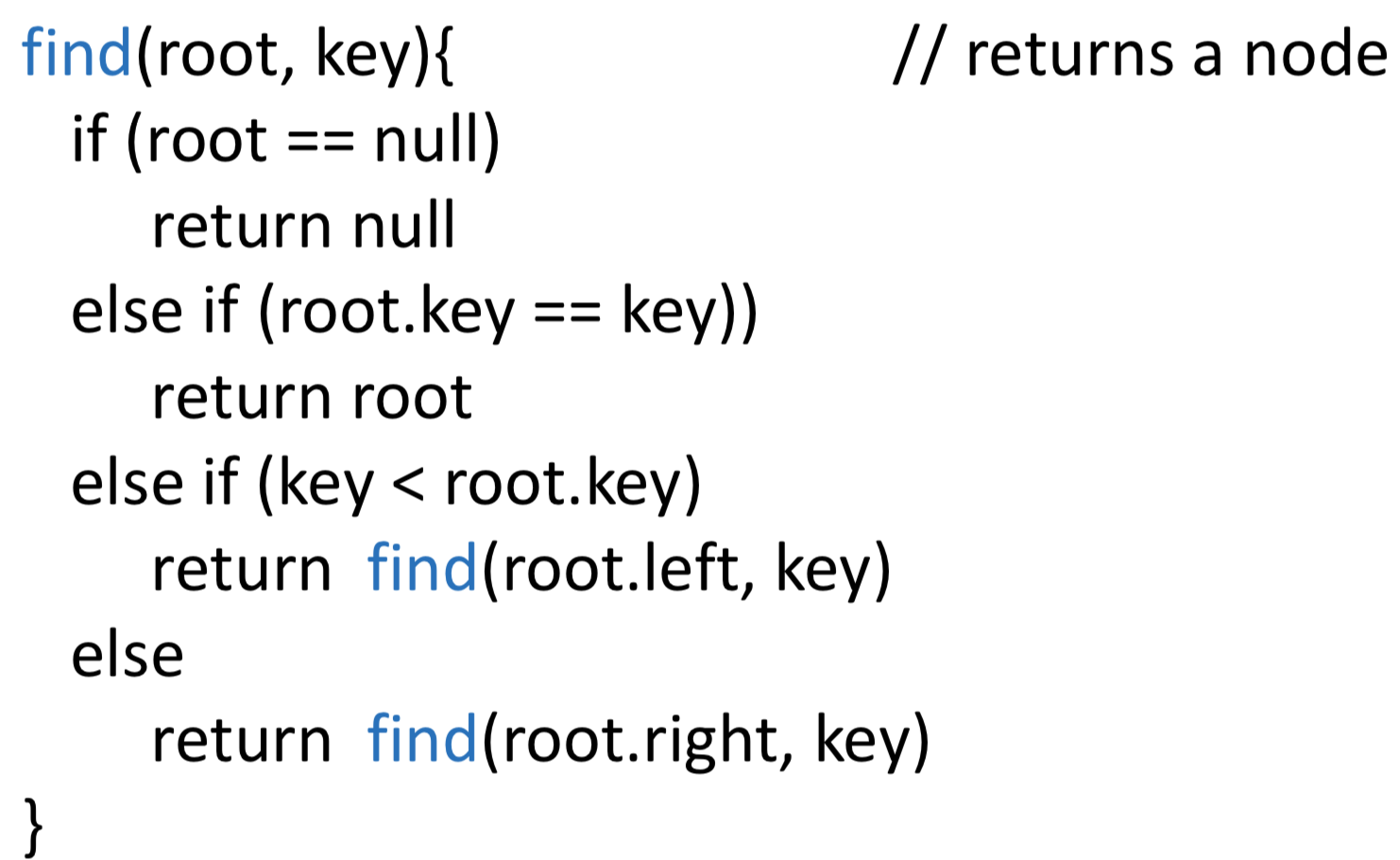
(comparison is based on node key)

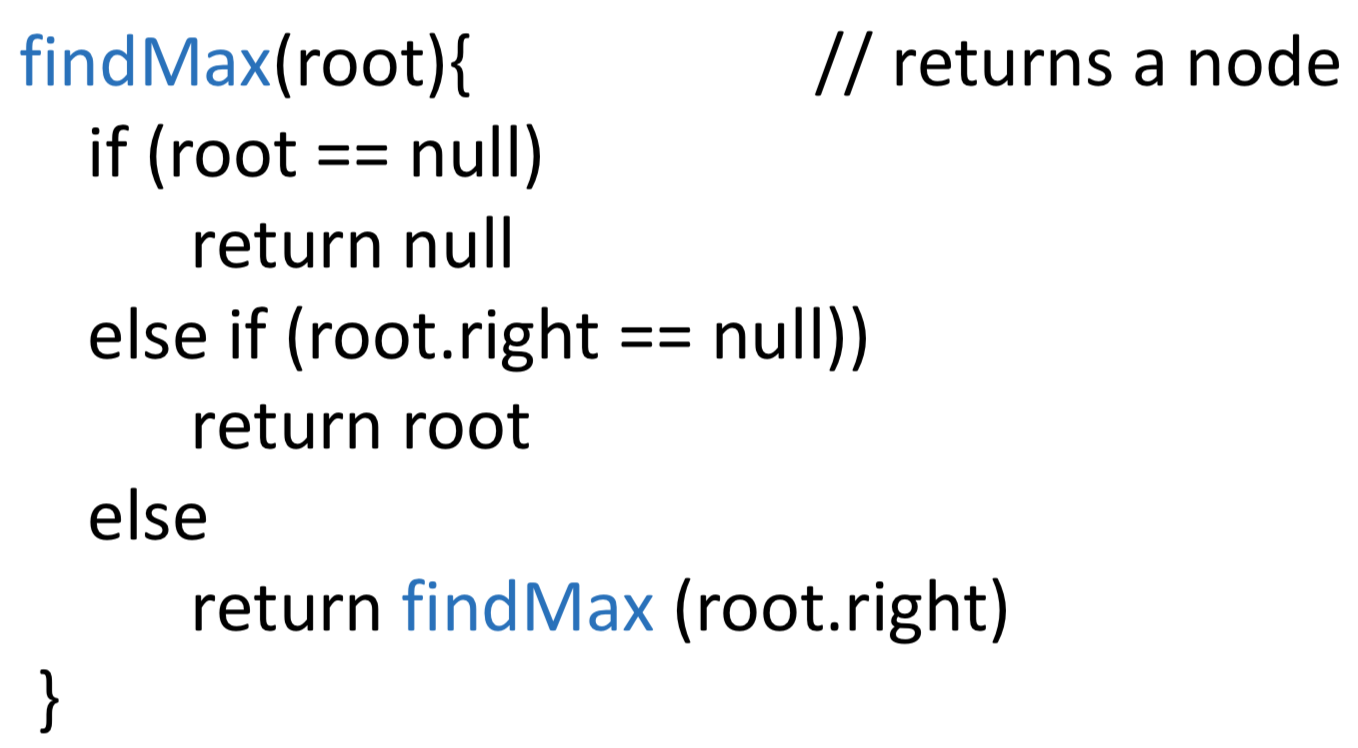
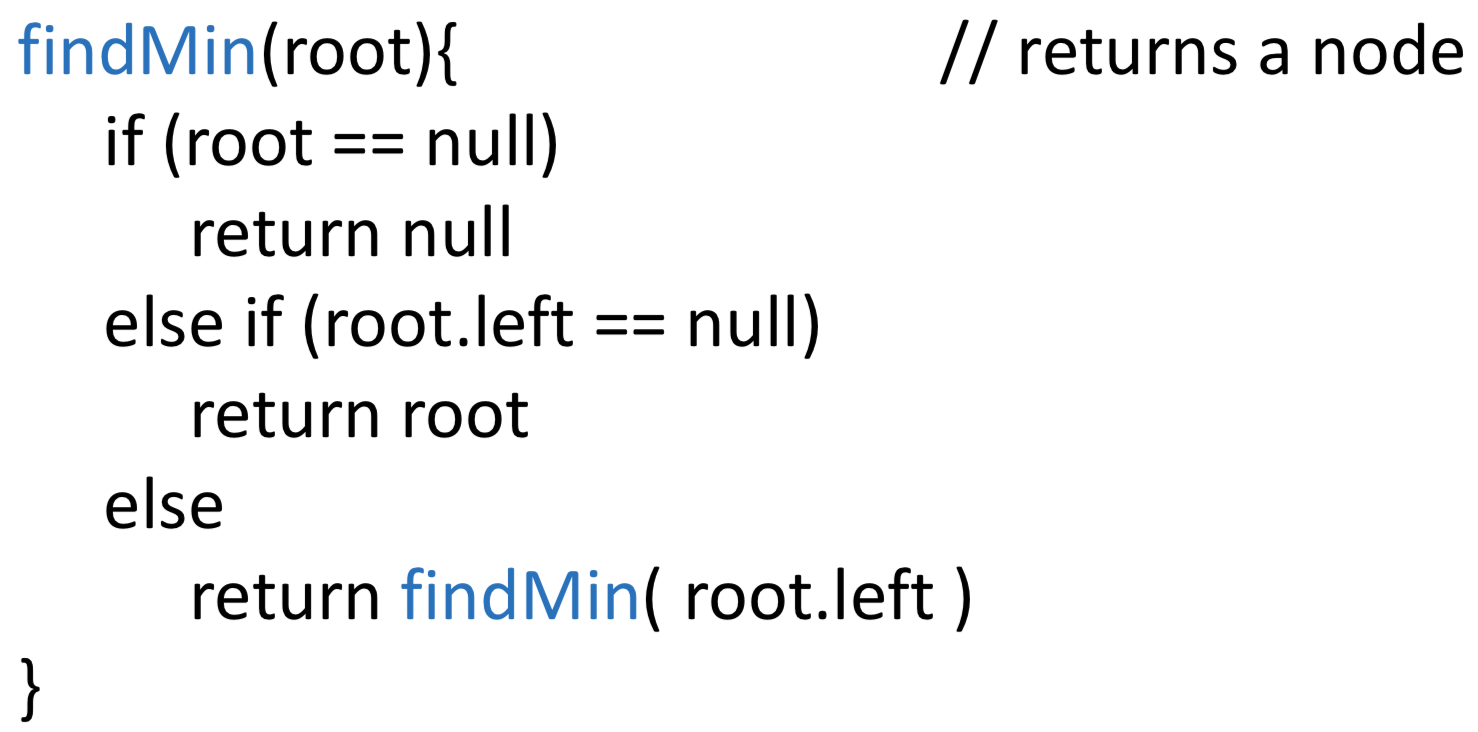
An *in-order traversal* on a BST visits the nodes in the natural order defined by the key.

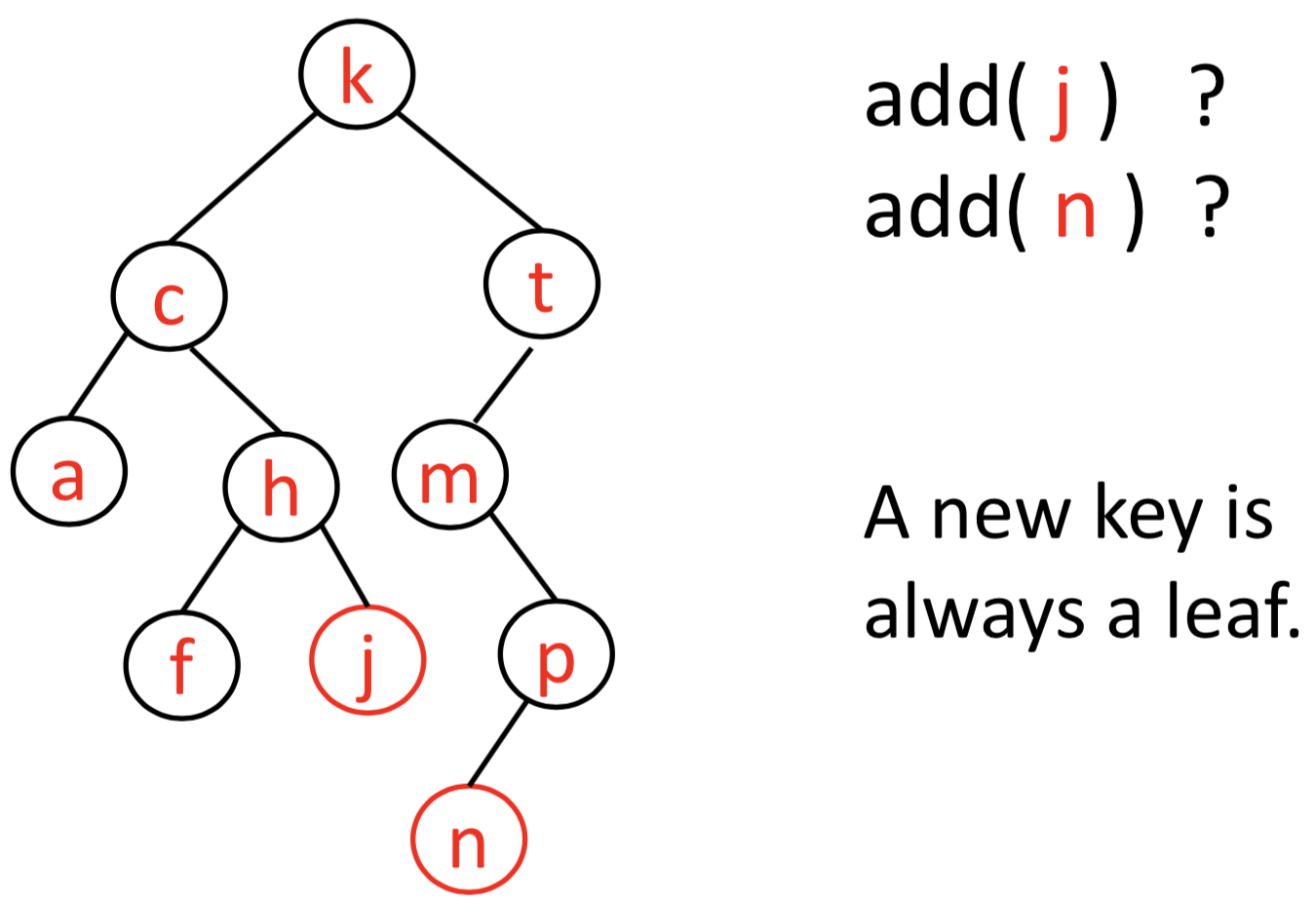
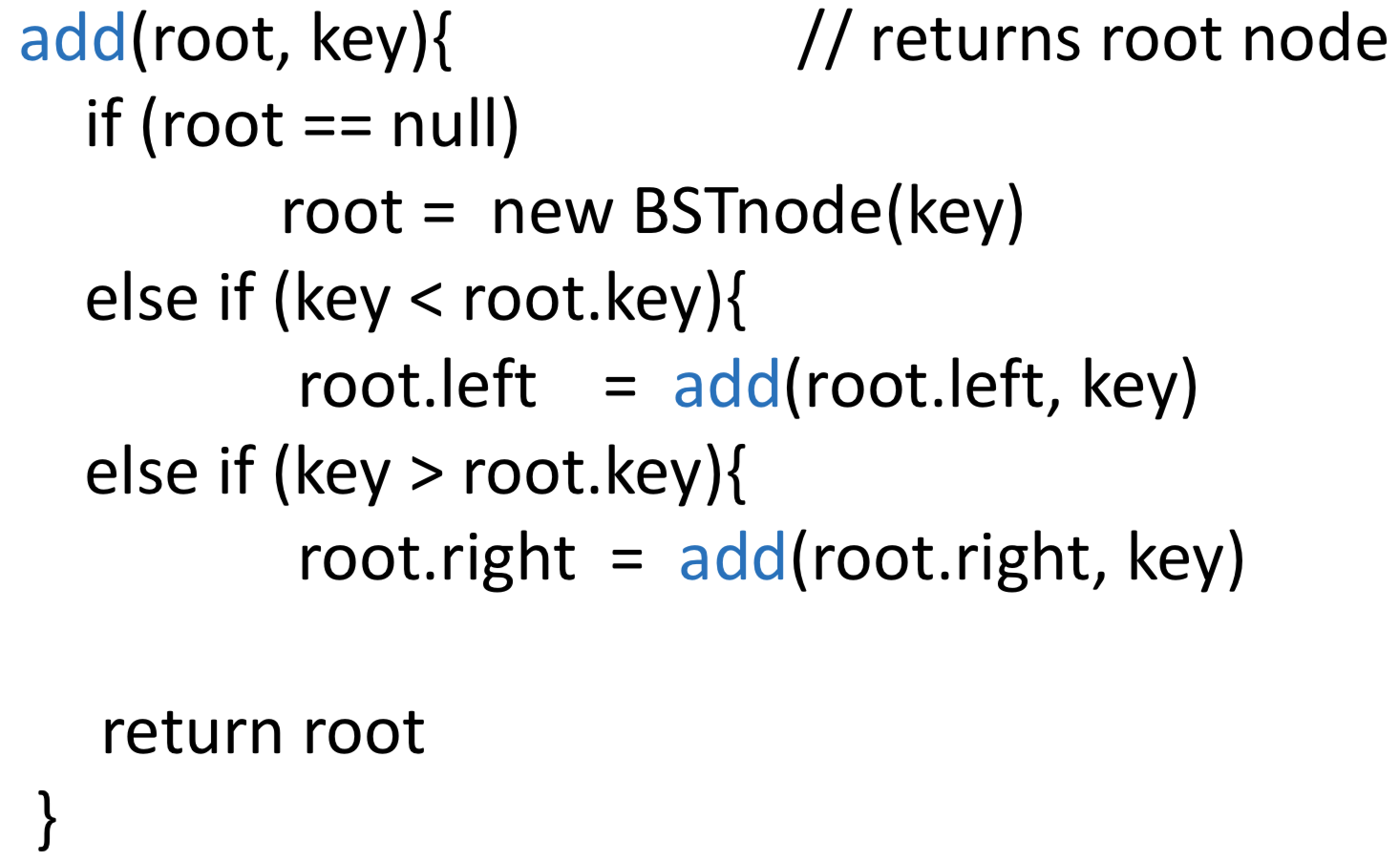


acdefgjm

**[Binary Search Tree ADT]**



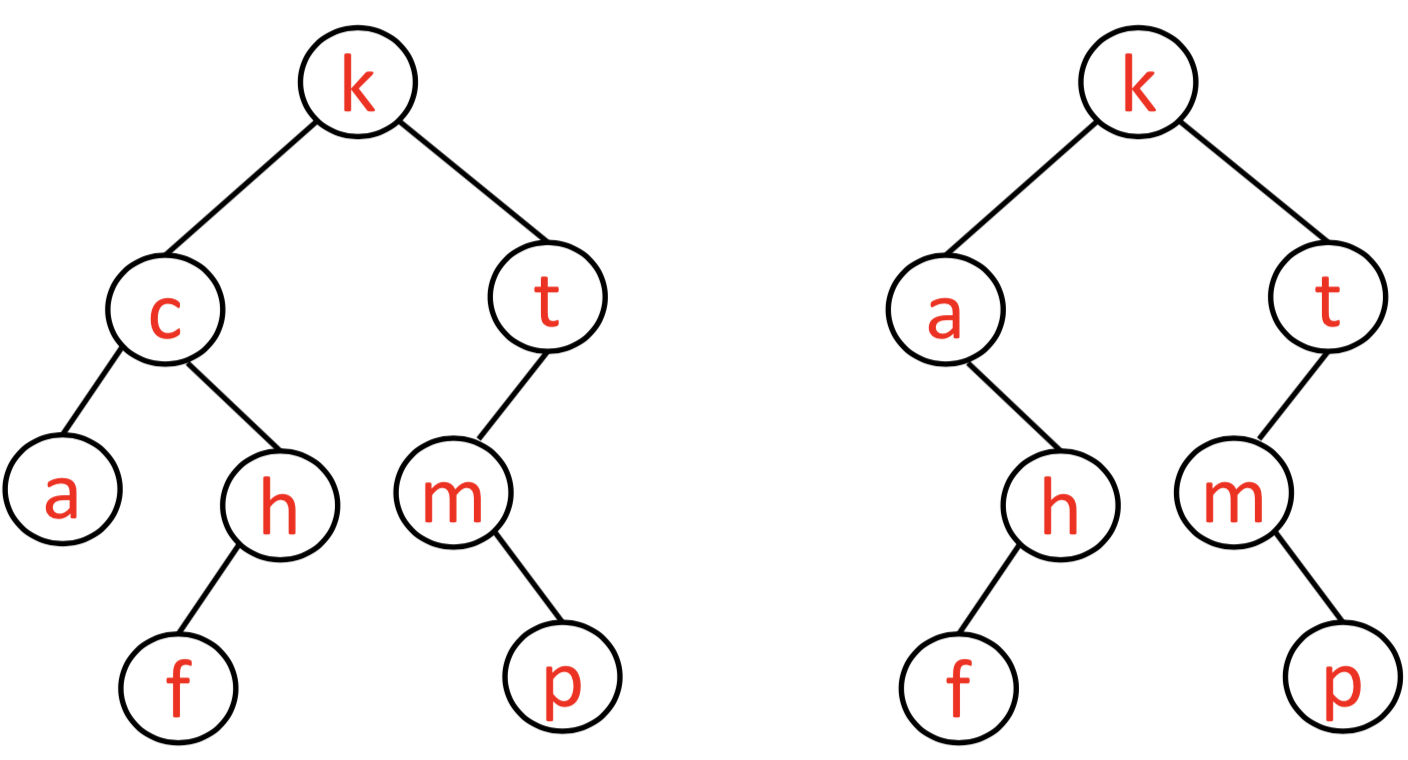
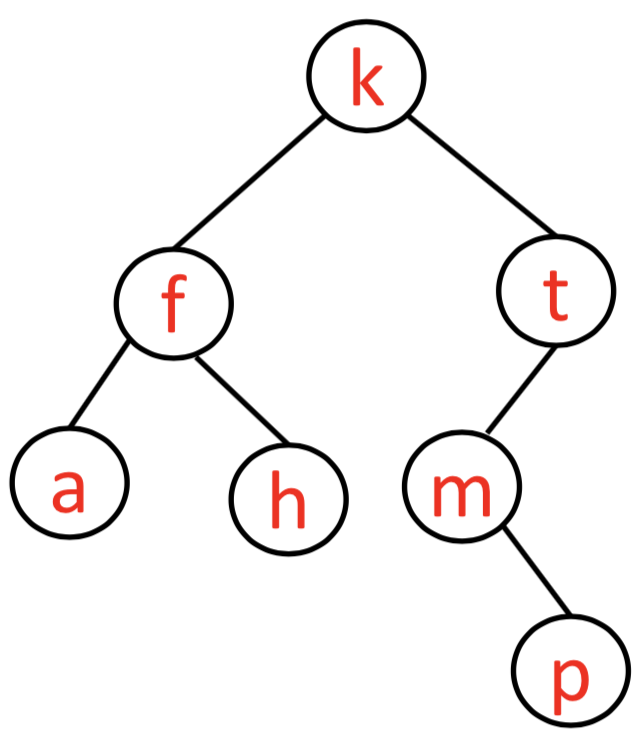




It does nothing if root.key == key.

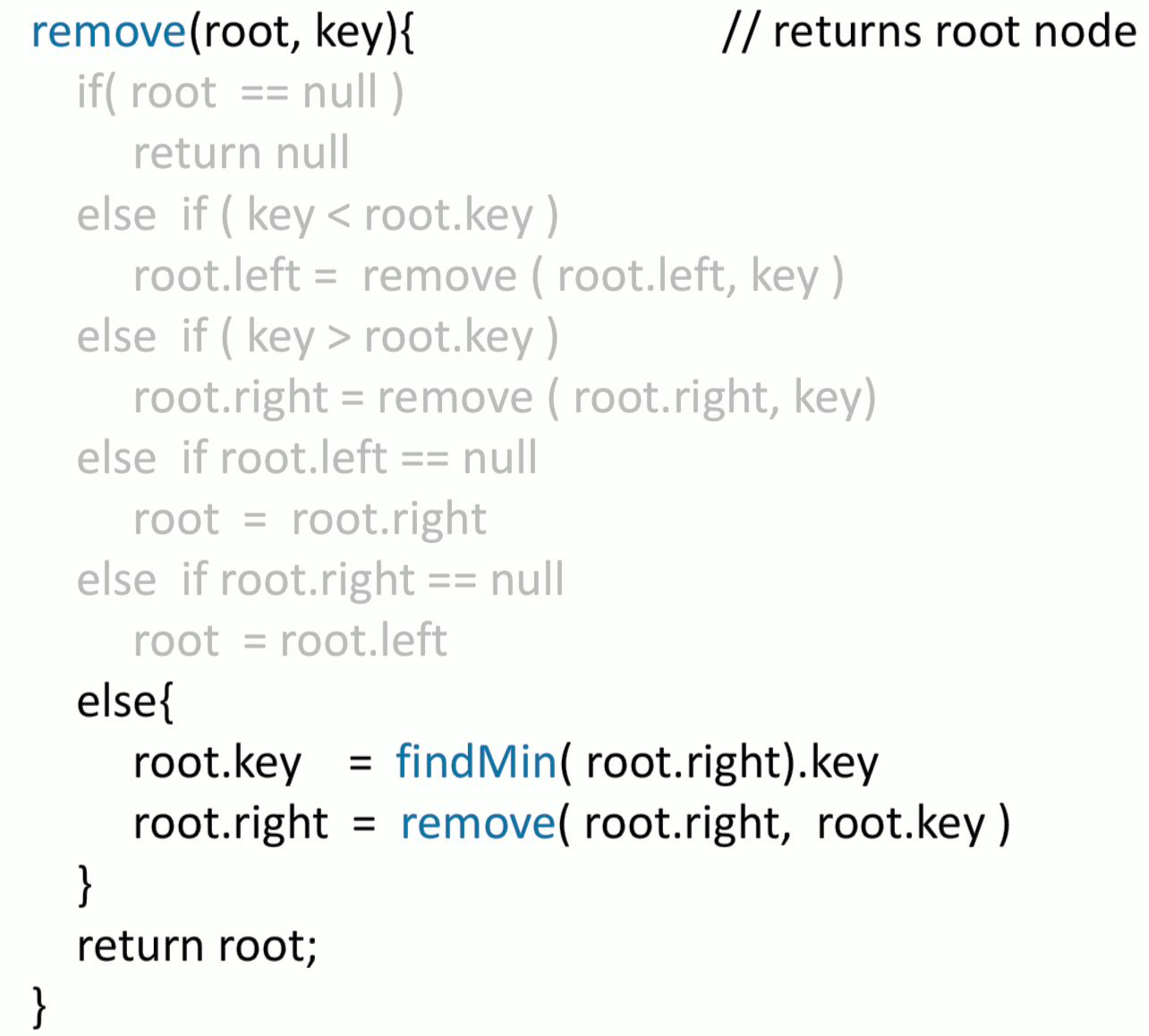
(cannot duplicate)

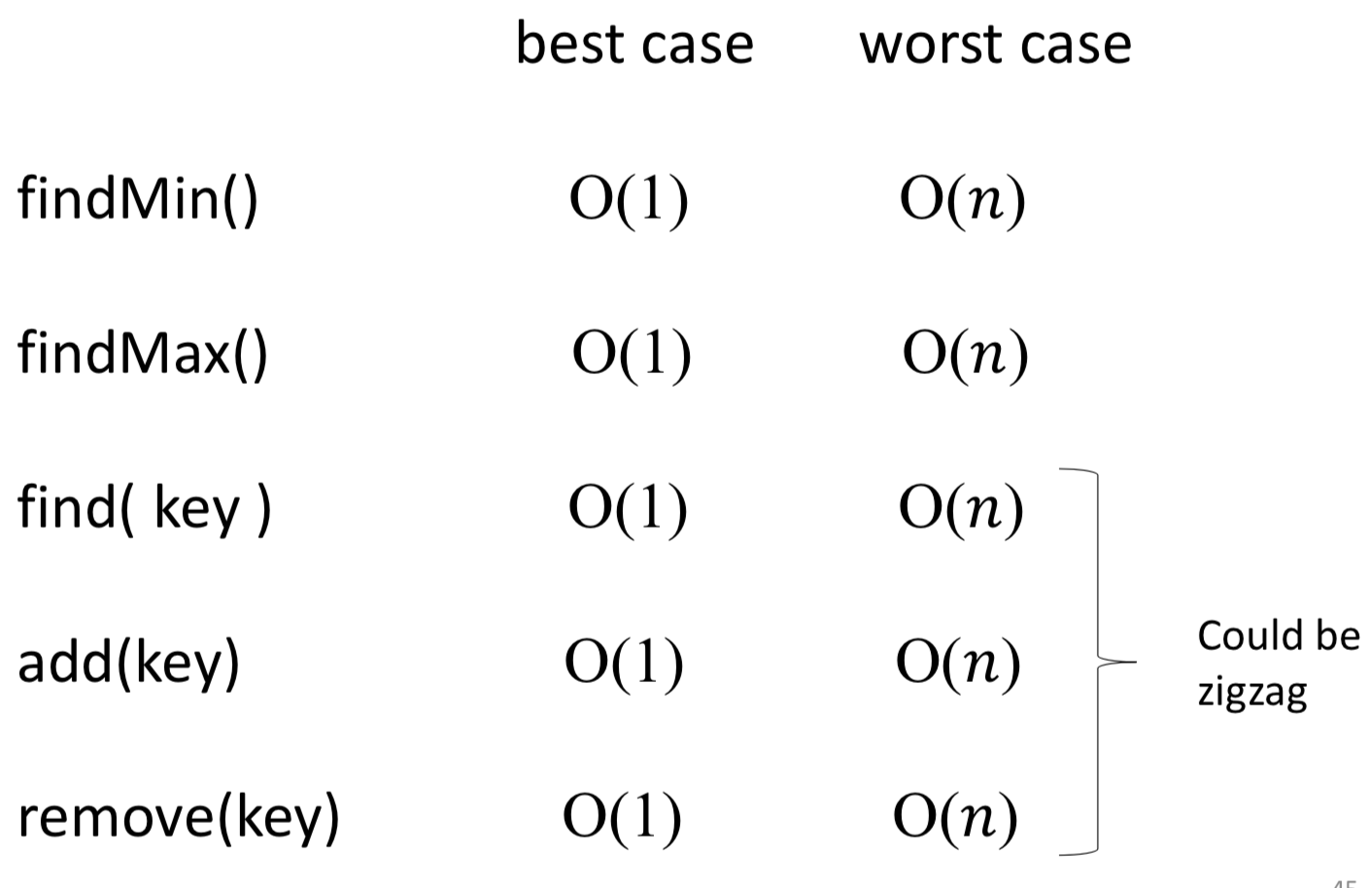
remove(c)

method 1

method 2





When a binary search tree is balanced, then finding a key is very similar to a binary search.

