COMP 250

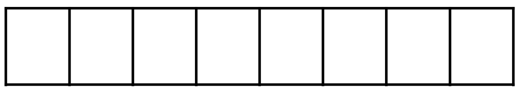
Lecture 23

(rooted) trees

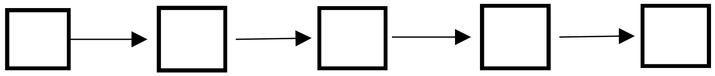
March 7, 2022

Linear Data Structures

array

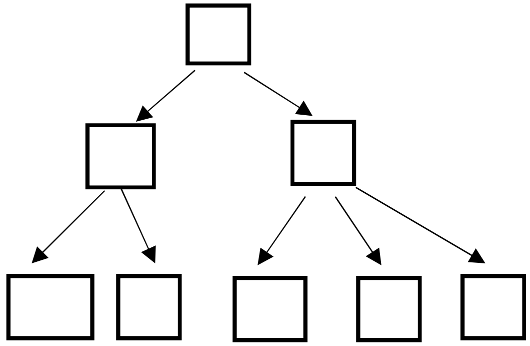


linked list

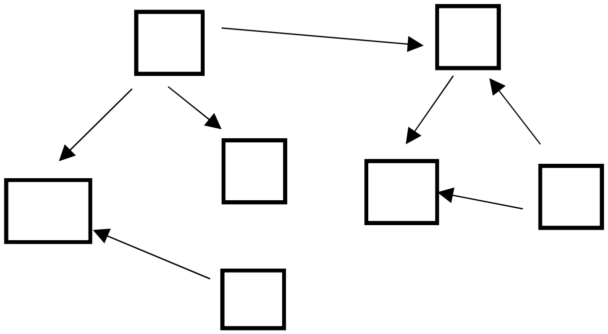


Non-Linear Data Structures

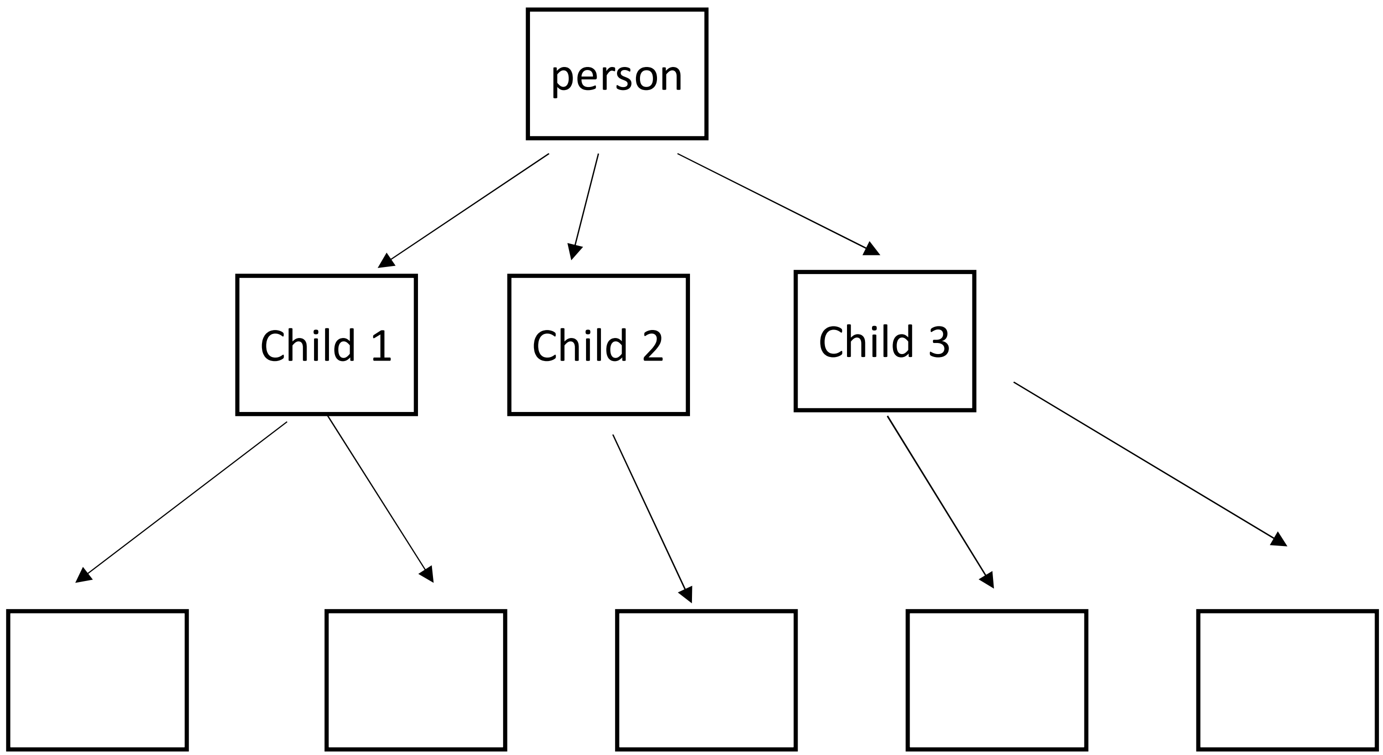
tree



graph

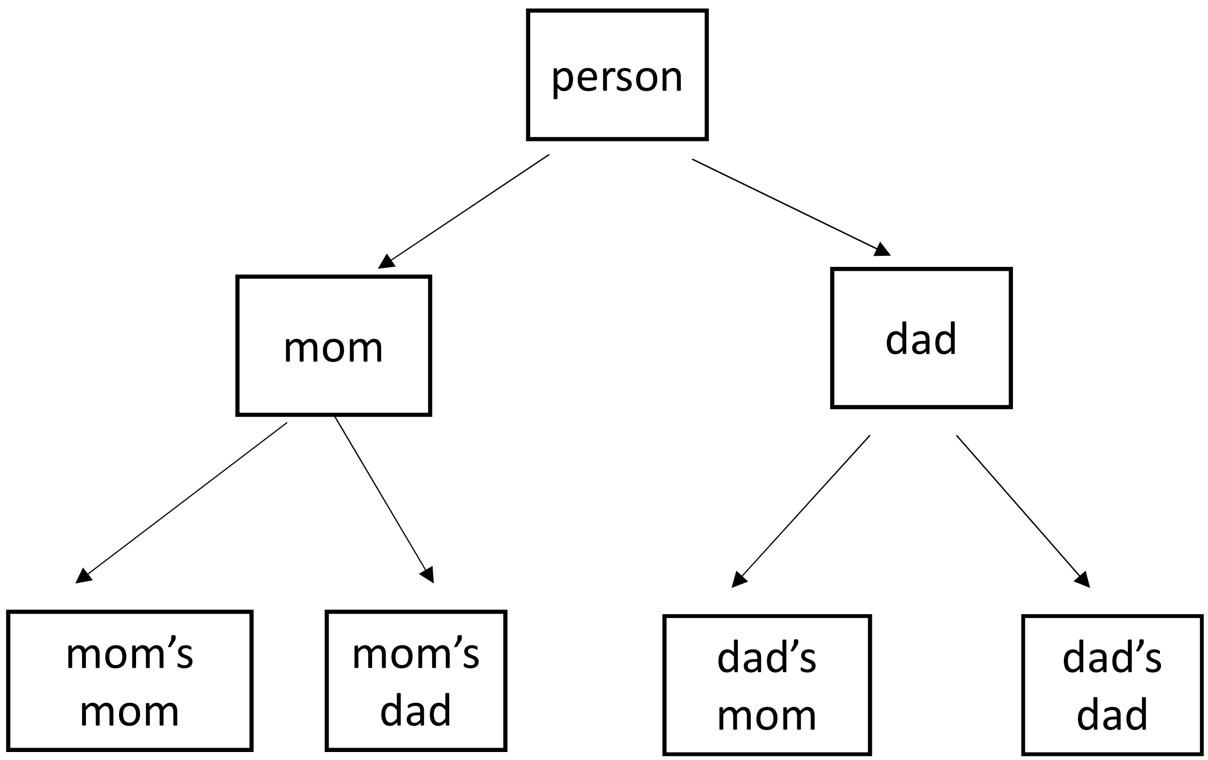


Family Tree (1)



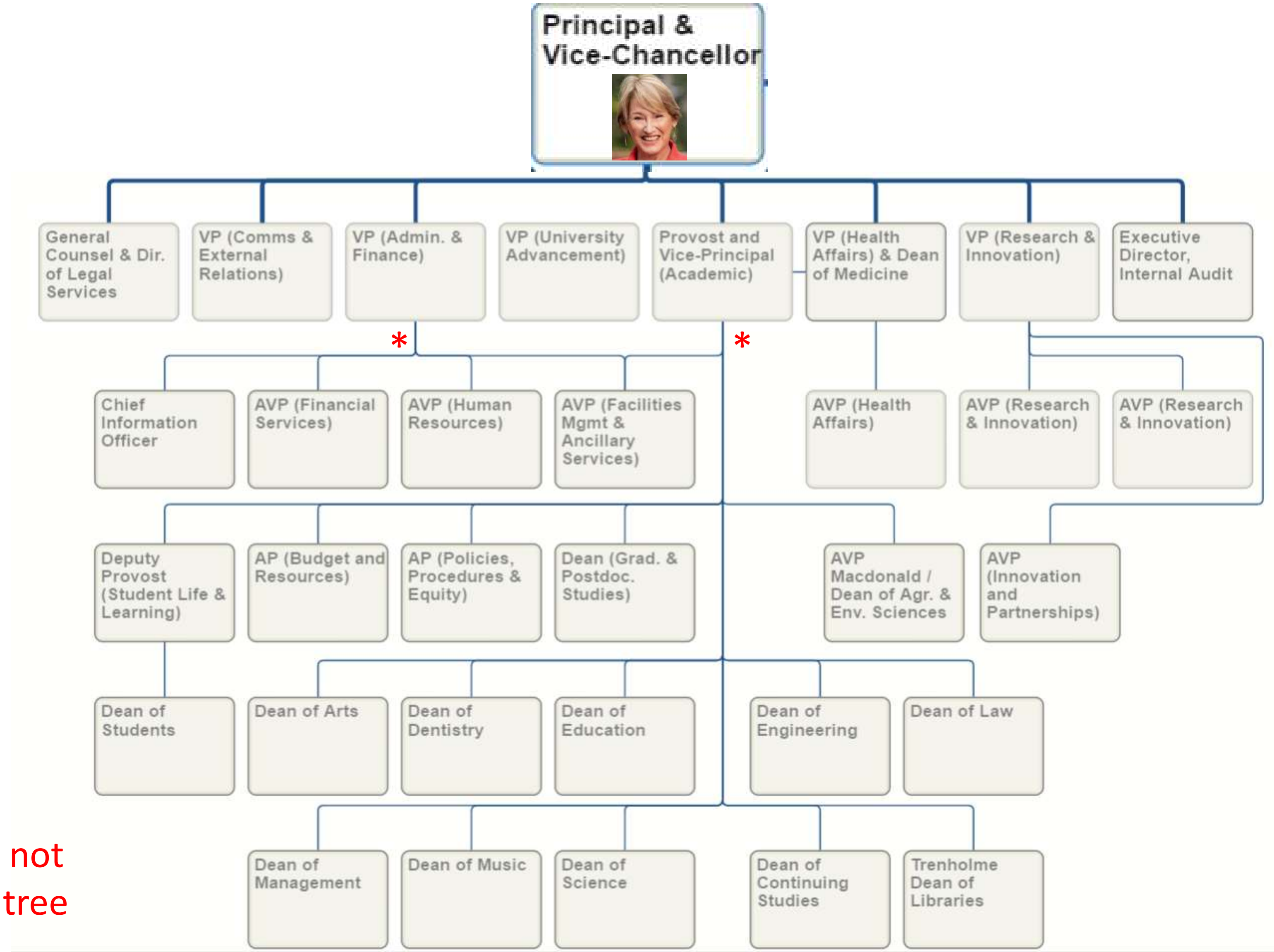
Here I ignore spouses ("partners").

Family Tree (2)



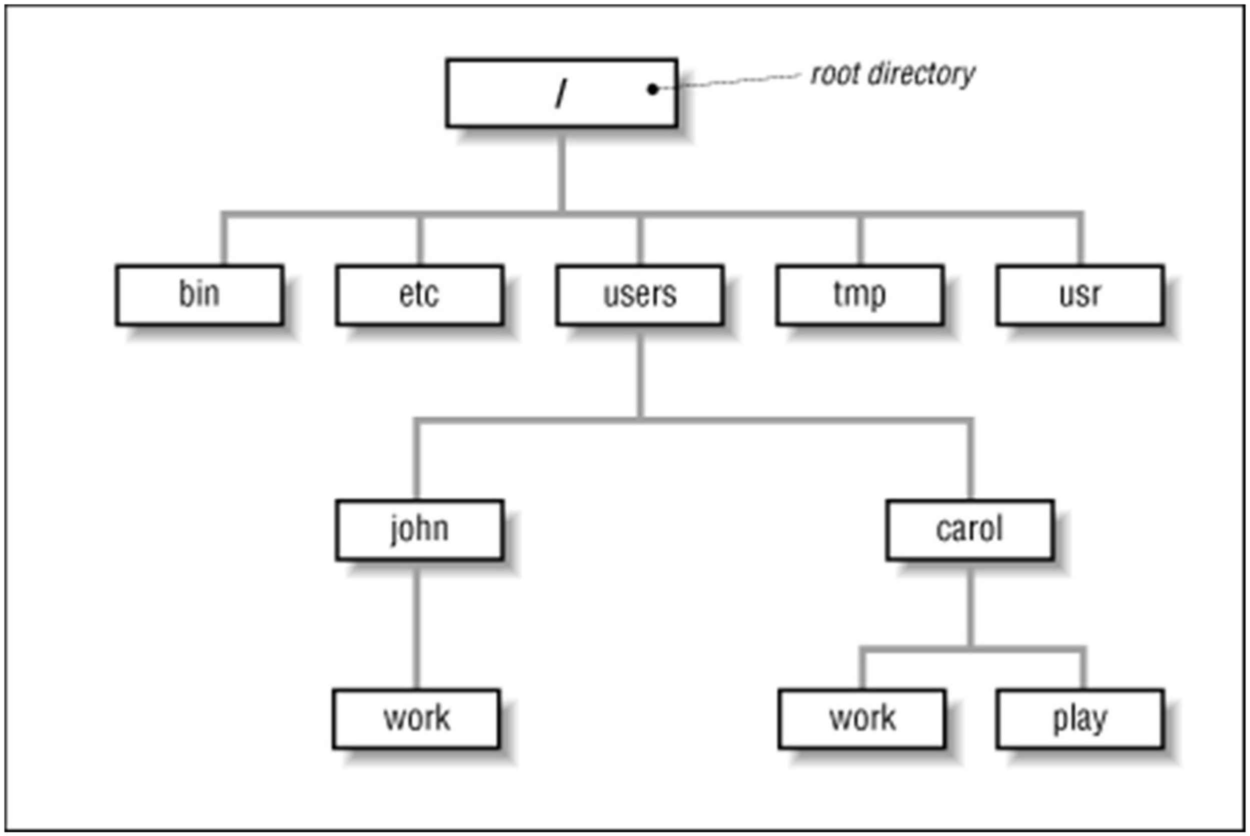
This is an example of a *binary tree*.

Organization Hierarchy (McGill)

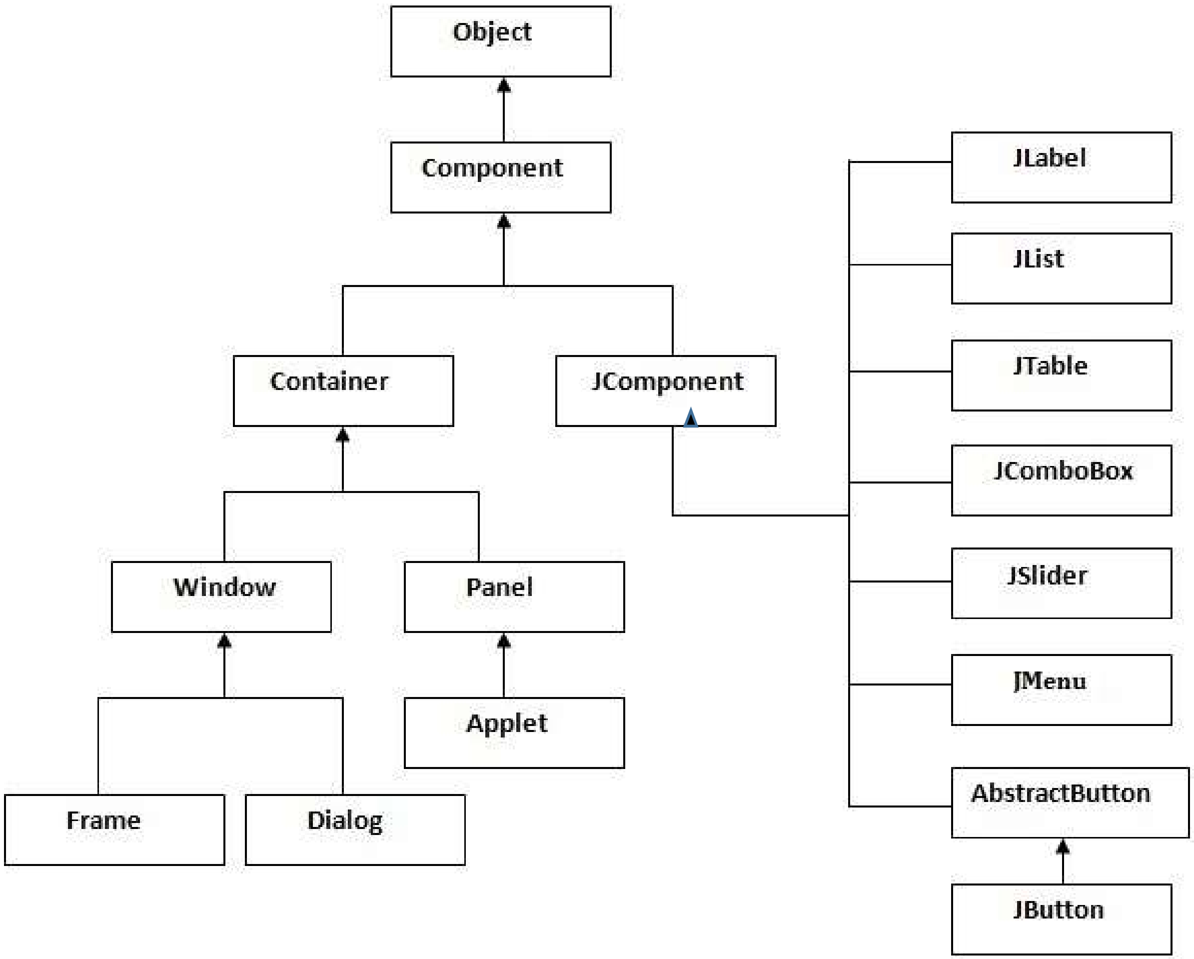


*This is not quite a tree

File system (e.g. UNIX/Linux)

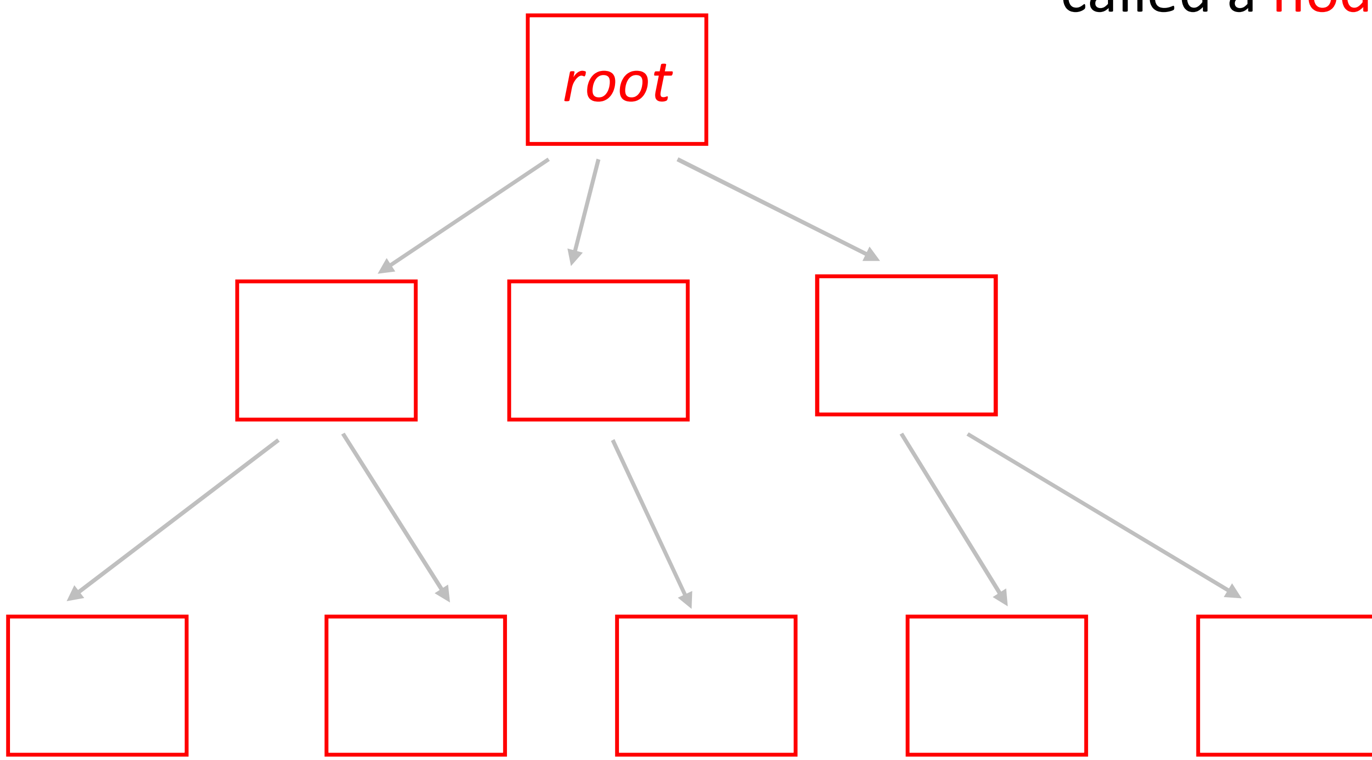


Java Classes e.g. GUI



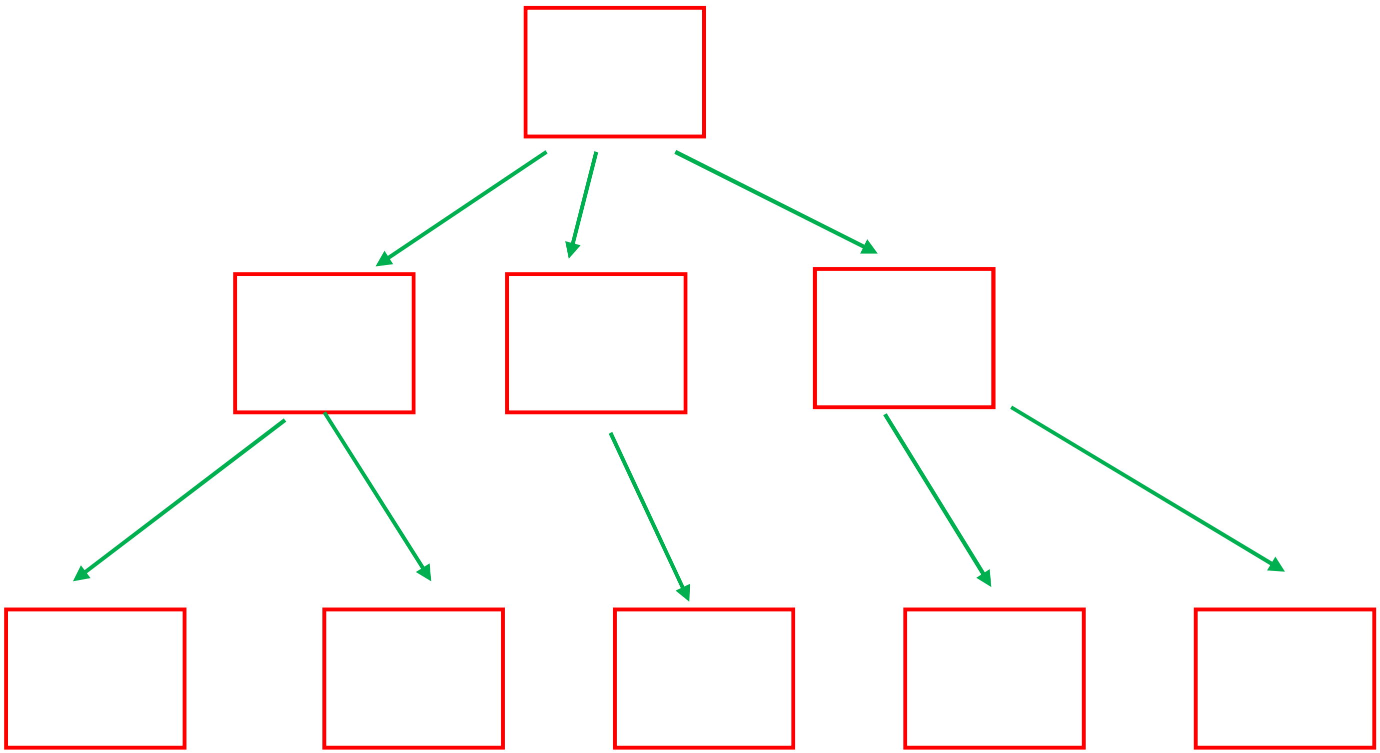
(Rooted) Tree Terminology

a box in the figure is called a node or vertex

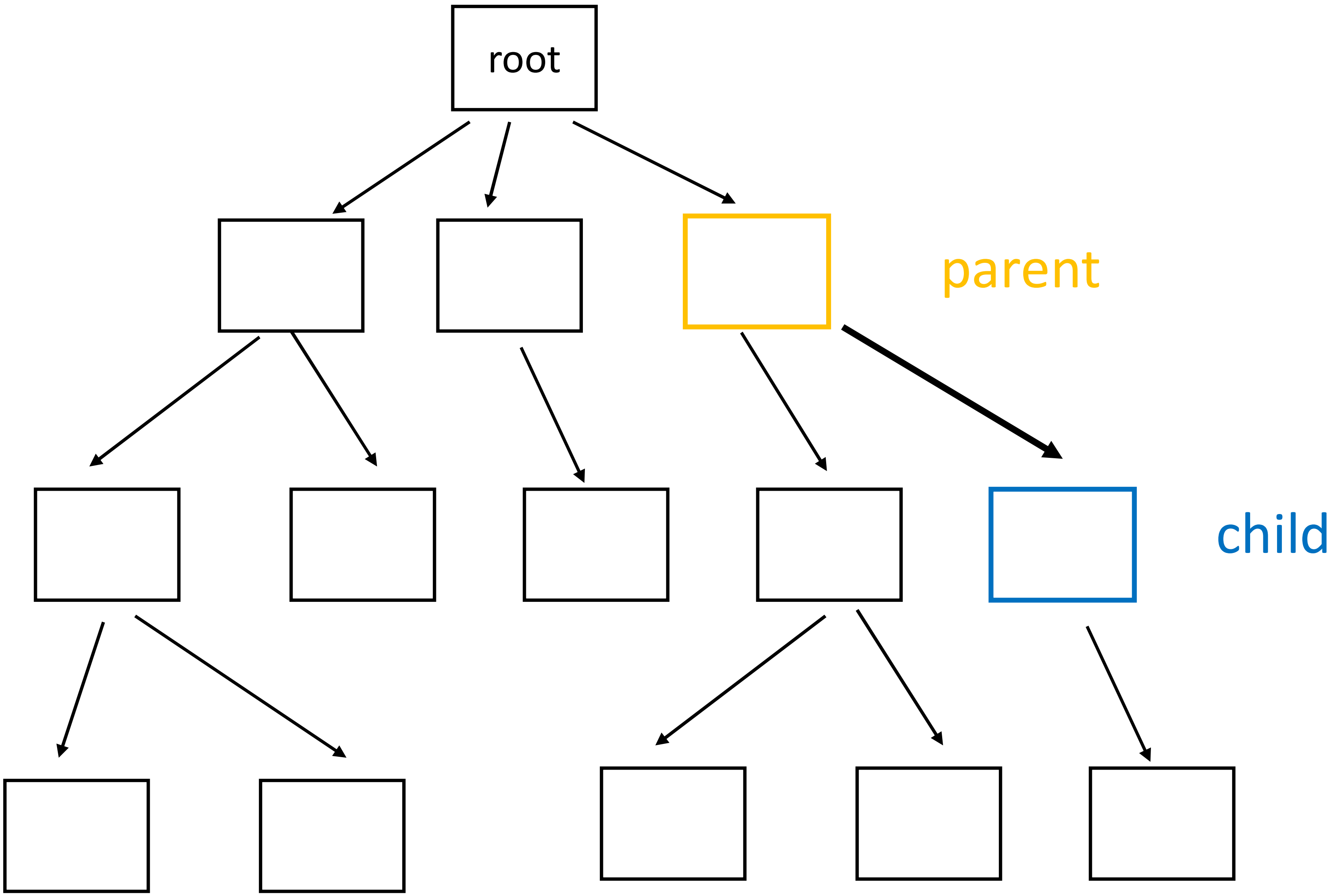


Tree Terminology

A *directed edge* is an ordered pair of nodes: (from, to)



In a rooted tree, every node (except the root) is a child of exactly one parent. (But a parent can have more than one child.)



For some (rooted) trees, edges are...

- from parent to child

Most of definitions today will assume edges are from parent to child.

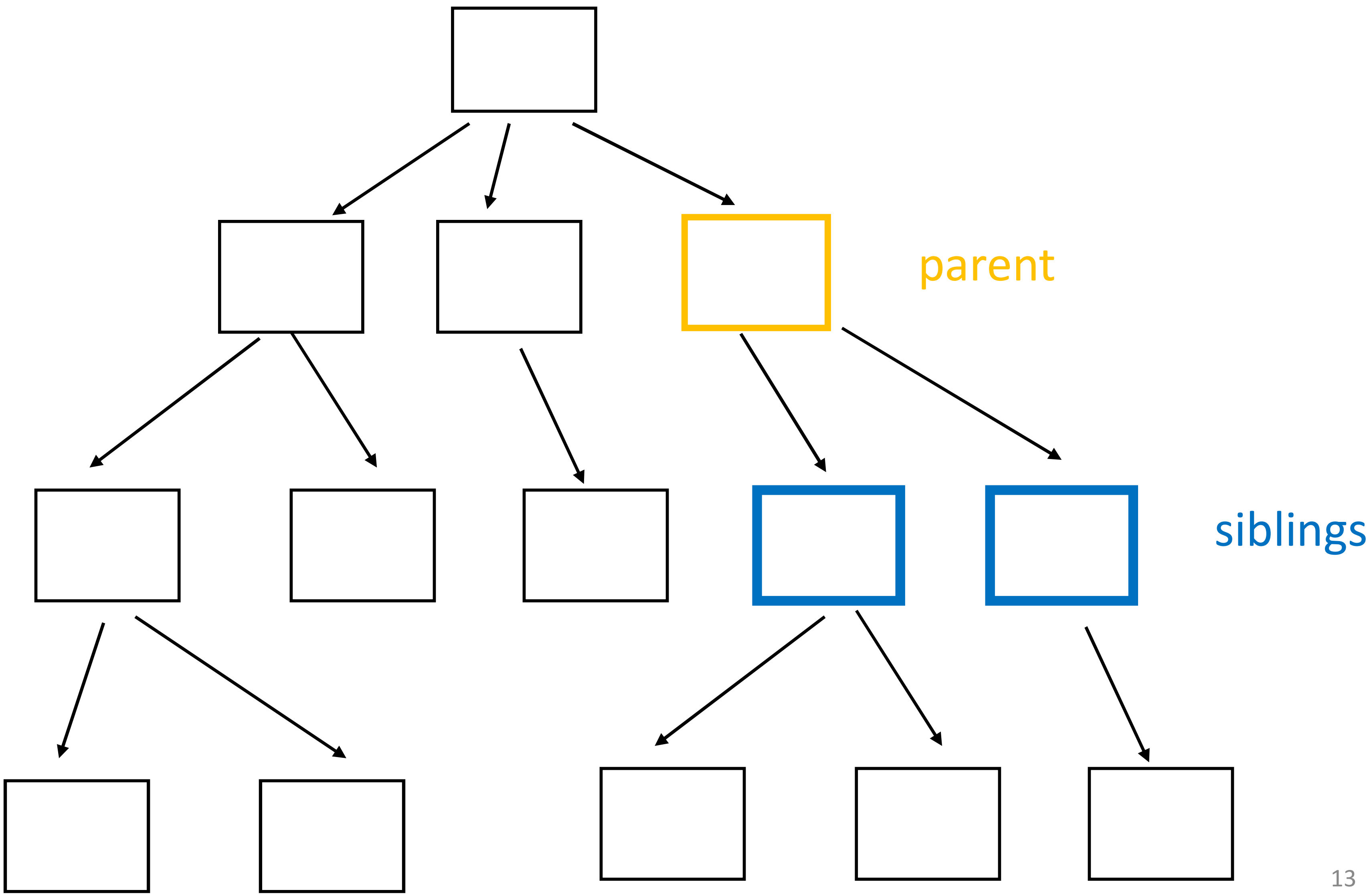
- from child to parent
- both from parent to child and from child to parent
- in neither direction (direction is ignored)

Q: 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 child has exactly one parent, except for the root node which has no parent.

Two nodes are siblings if they have the same parent.



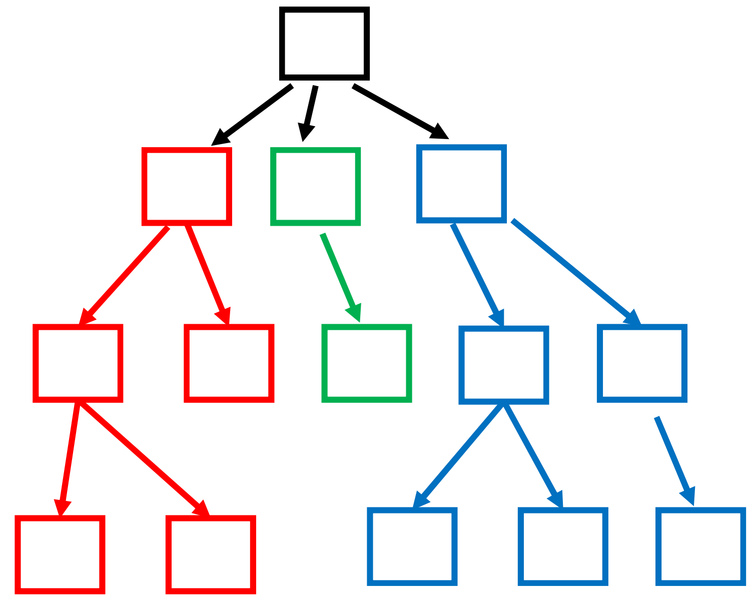
“Recursive” definition of rooted tree

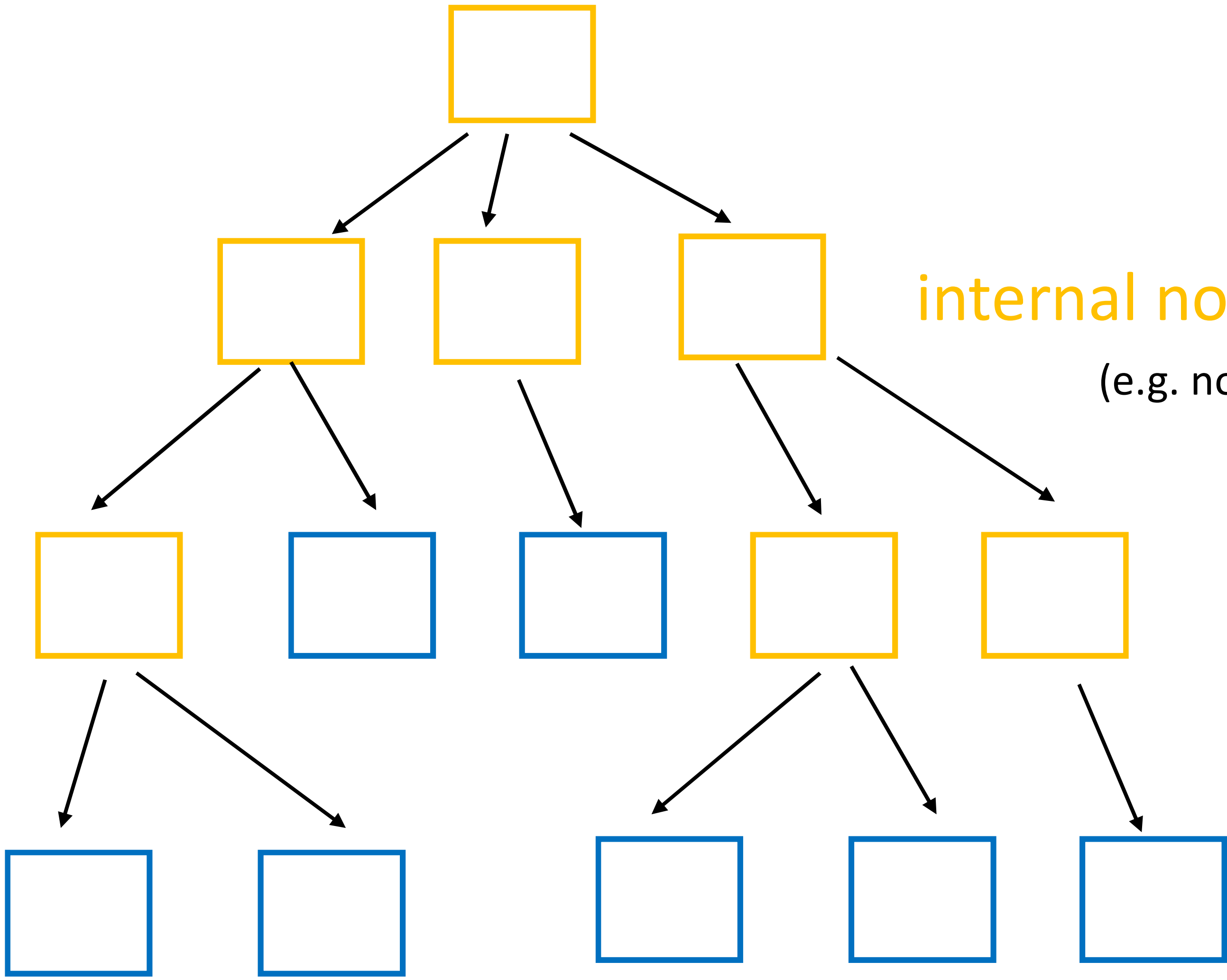
A rooted tree T is a set of $n > 0$ nodes such that:

- one of the nodes is the root r

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

- “base case” is $n = 1$



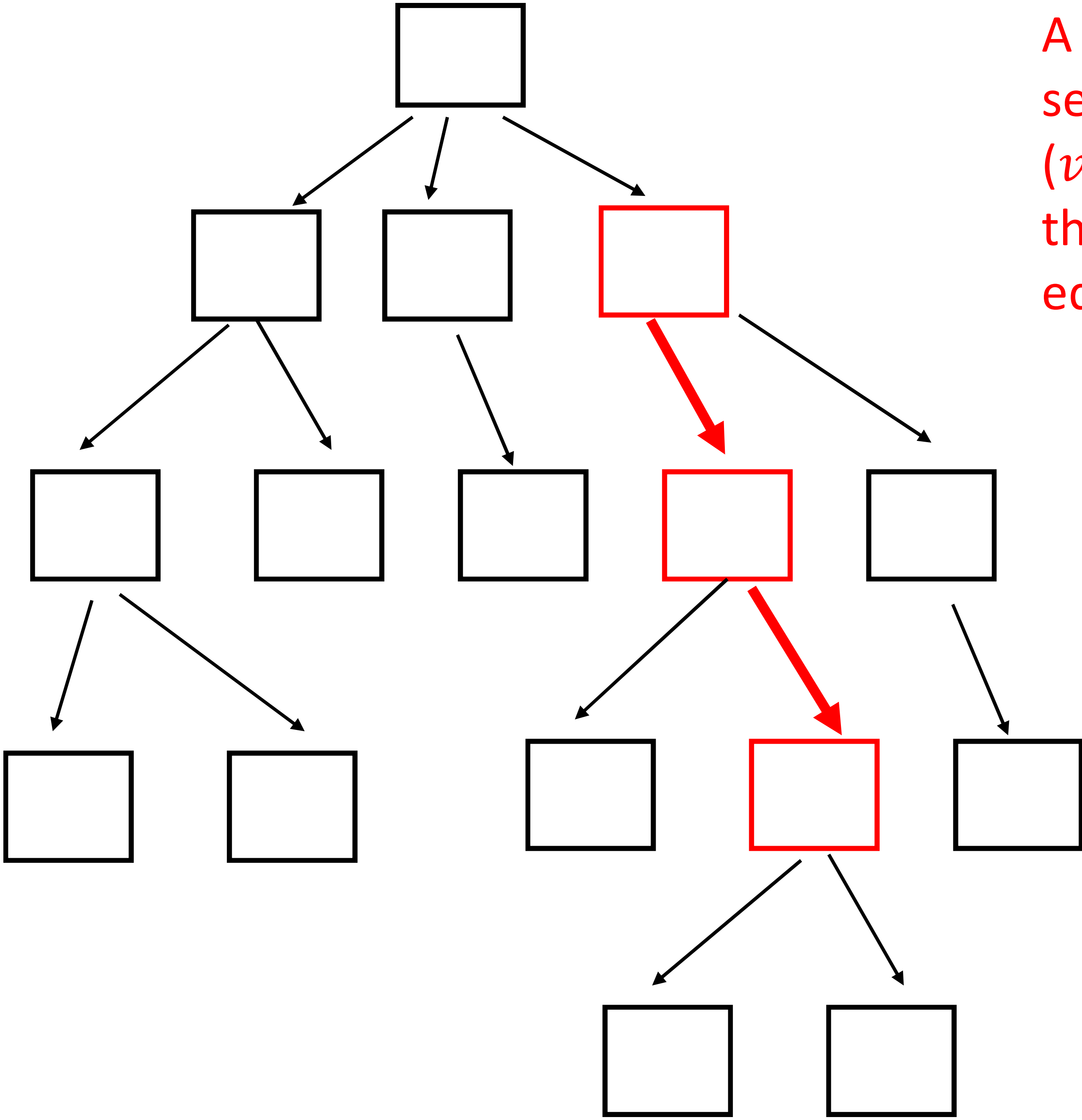


internal nodes

(e.g. non empty file directories)

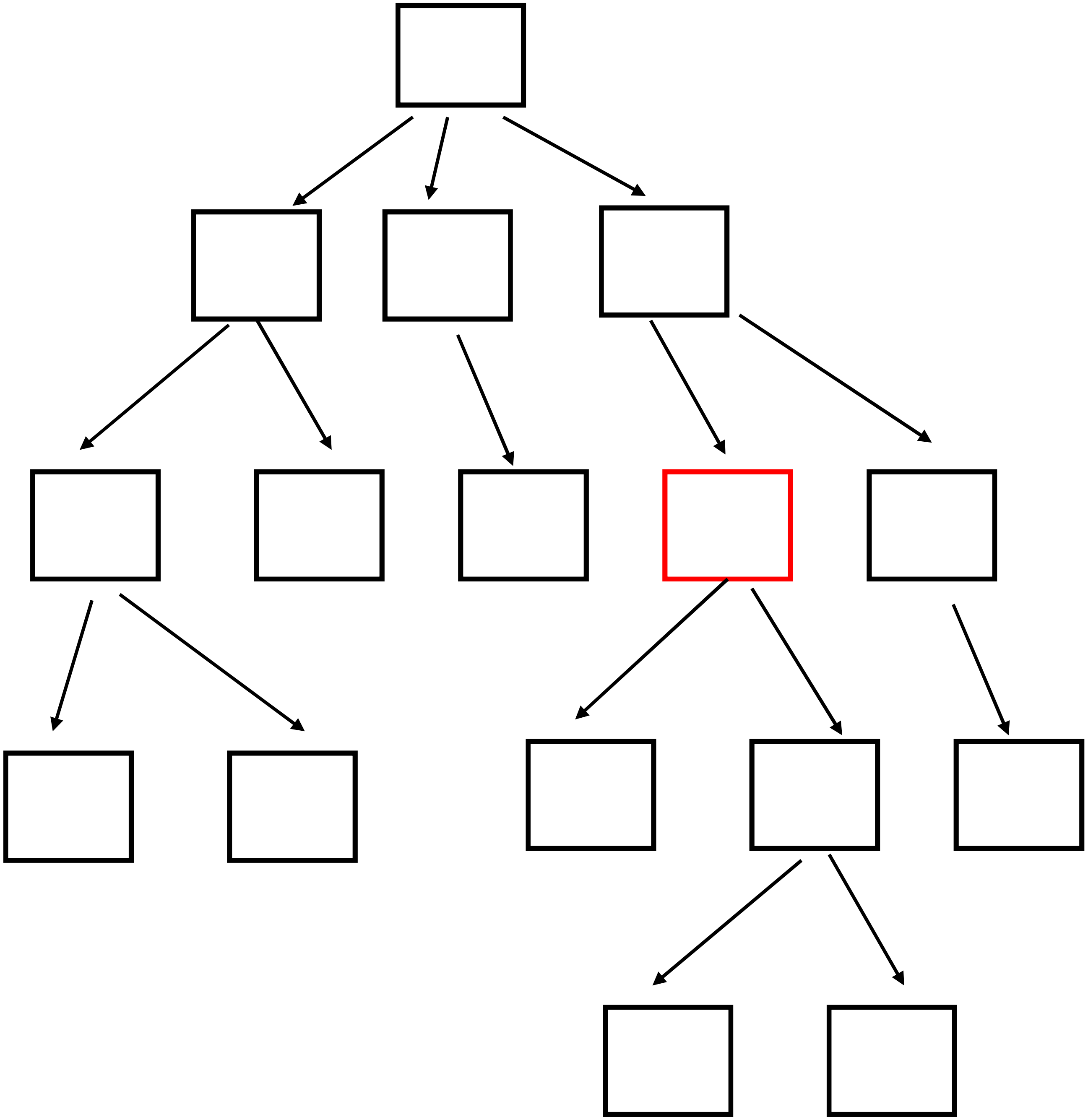
external nodes,
“leaves”

(e.g. files or empty directories)

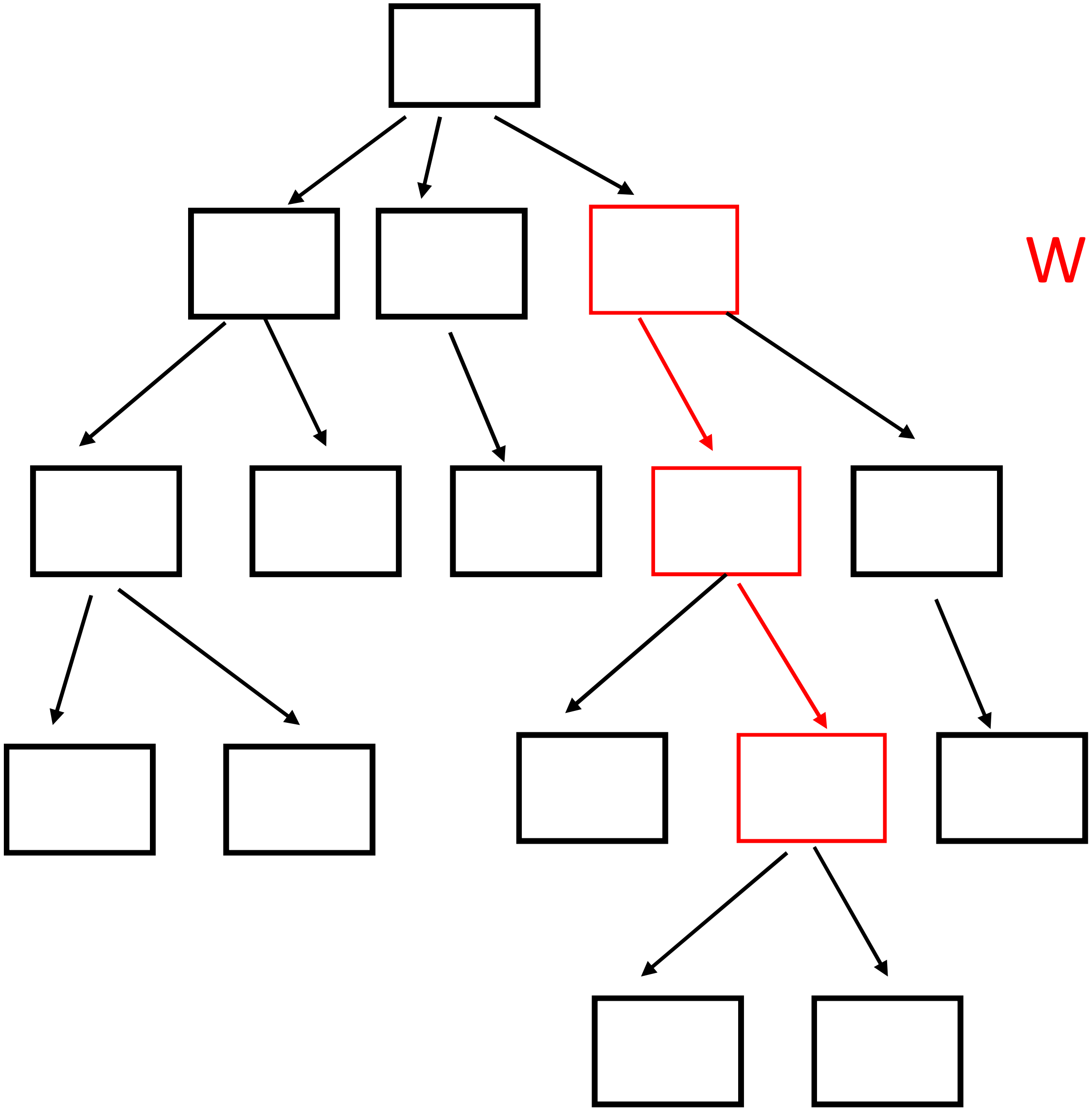


A **path** in a tree is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that (v_i, v_{i+1}) is an edge.

The **length** of a path is *the number of edges in the path* (number of nodes in the path minus 1)



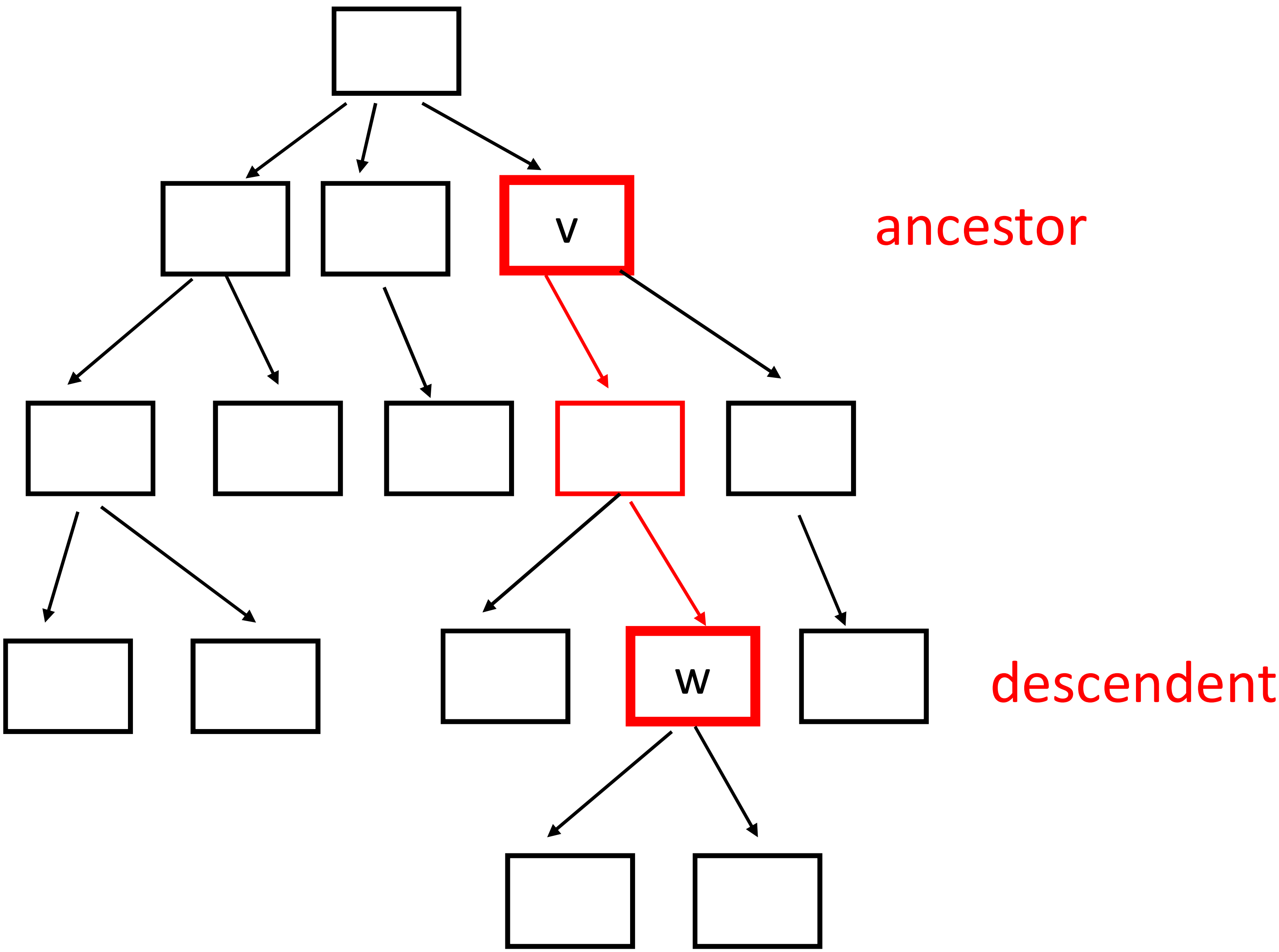
We also can talk about a path with just one node (v_1). Such a path has length = 0, since it has no edges.



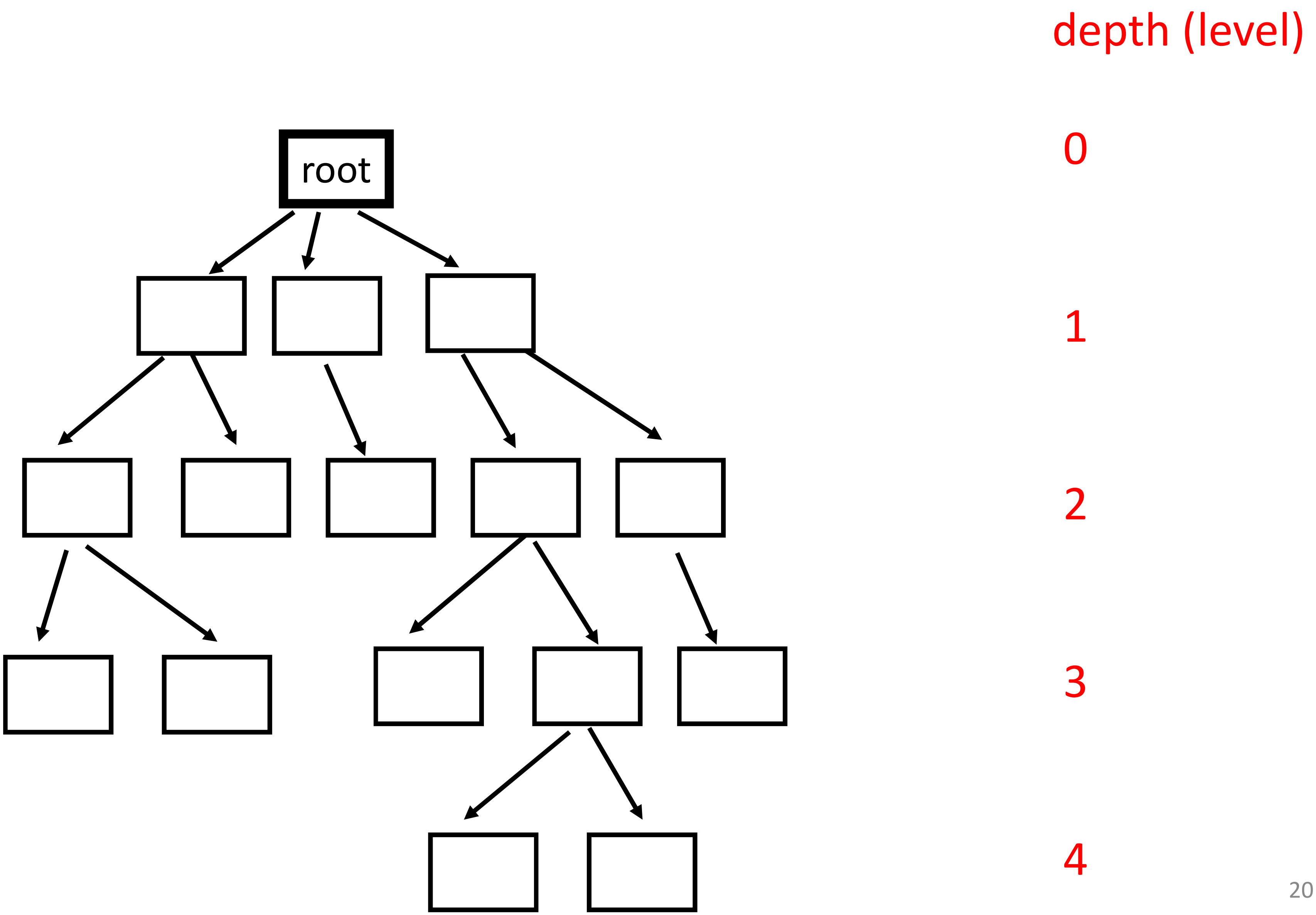
What is the path length?

Answer: 2 (edges)

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

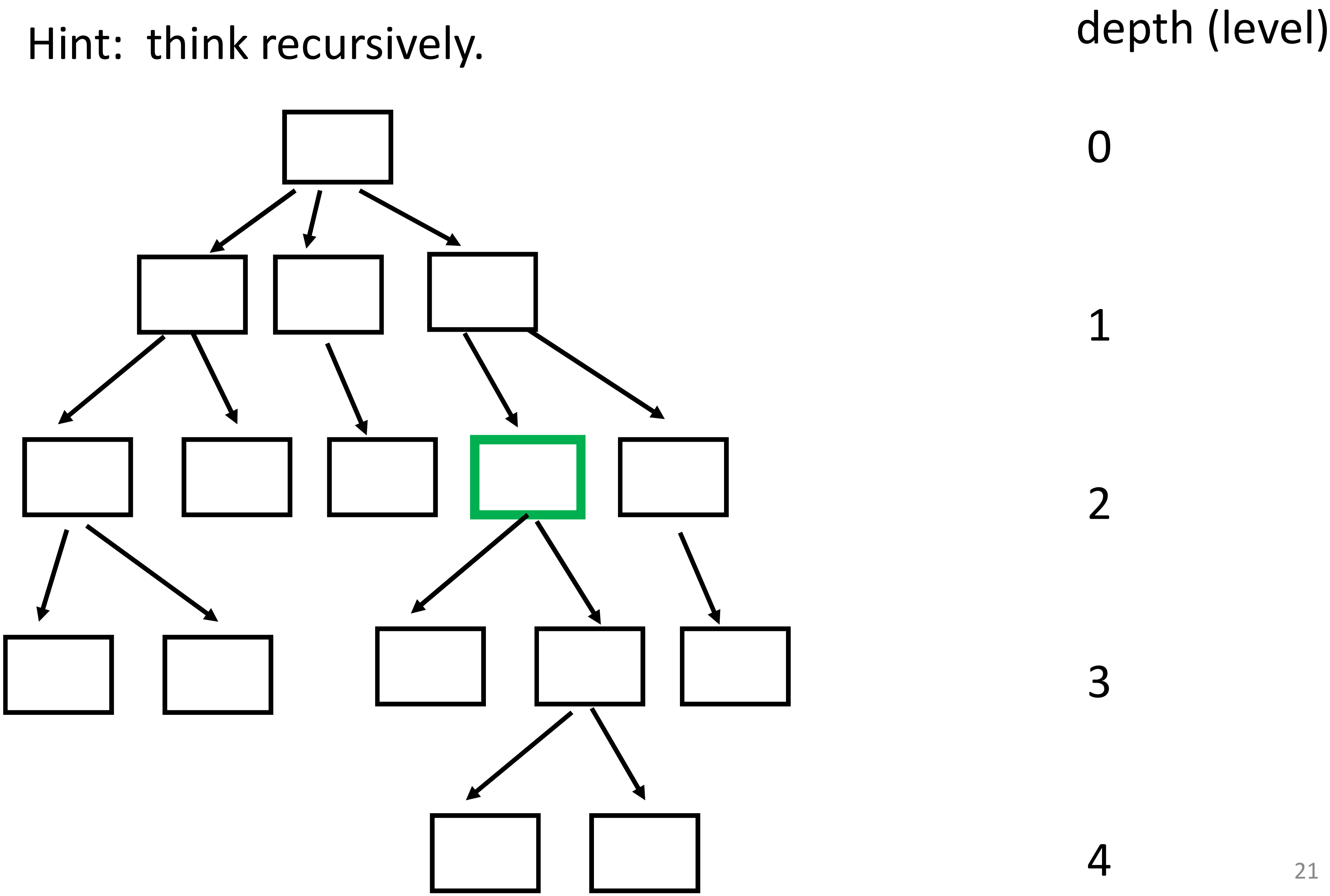


The depth or level of a node is the length of the path *from the root to the node*.



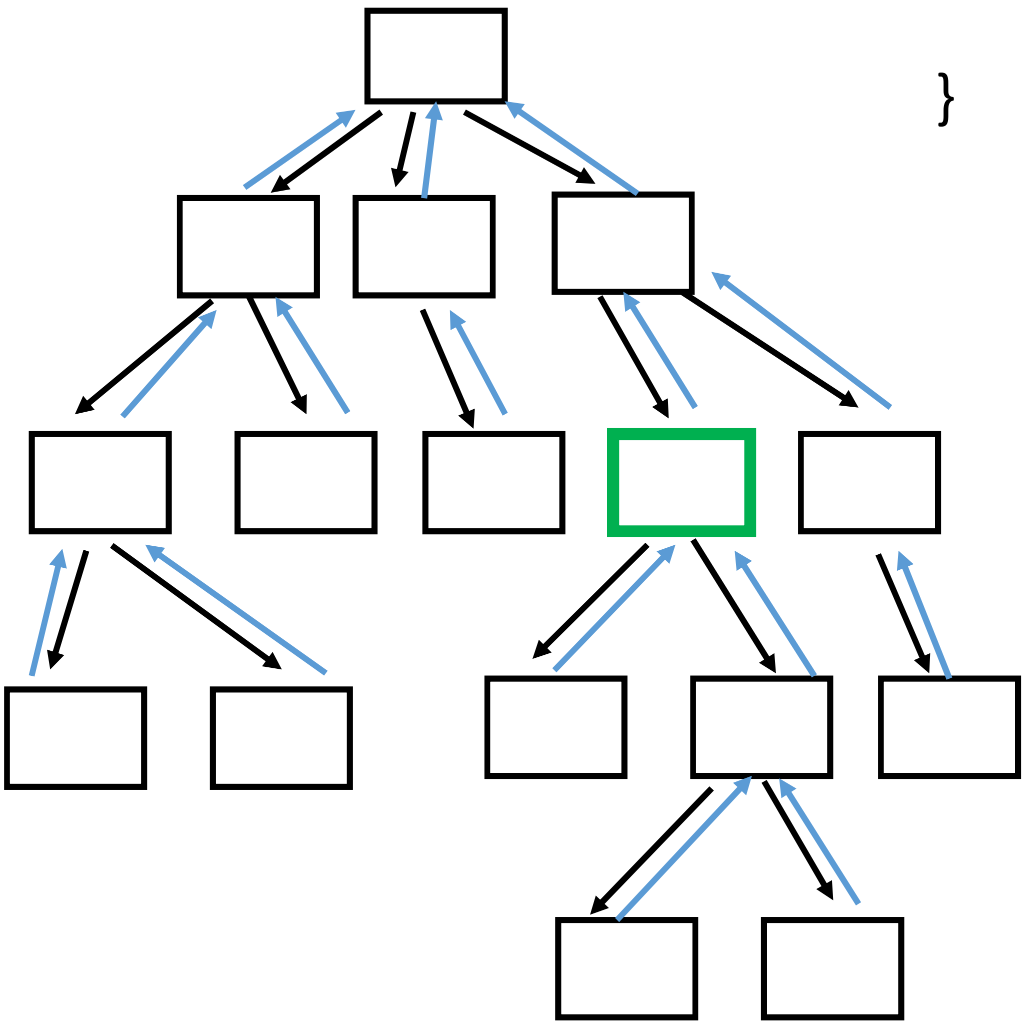
How to compute depth(v) ?

Hint: think recursively.

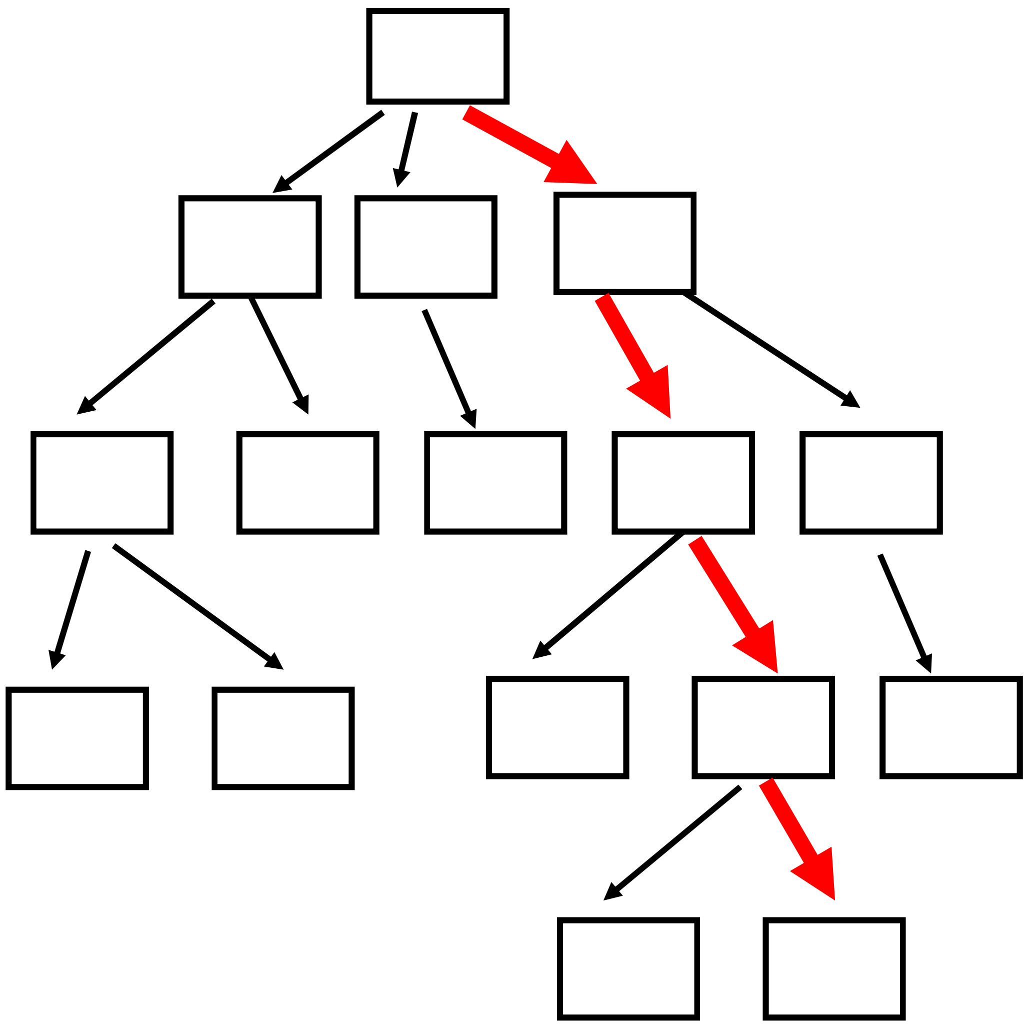


Computing depth requires parent links. This is analogous to a 'prev' link in a doubly linked list.

```
depth( v ){
    if ( v.parent == null)     //root
       return   0
    else
       return   1 + depth( v.parent )
}
```

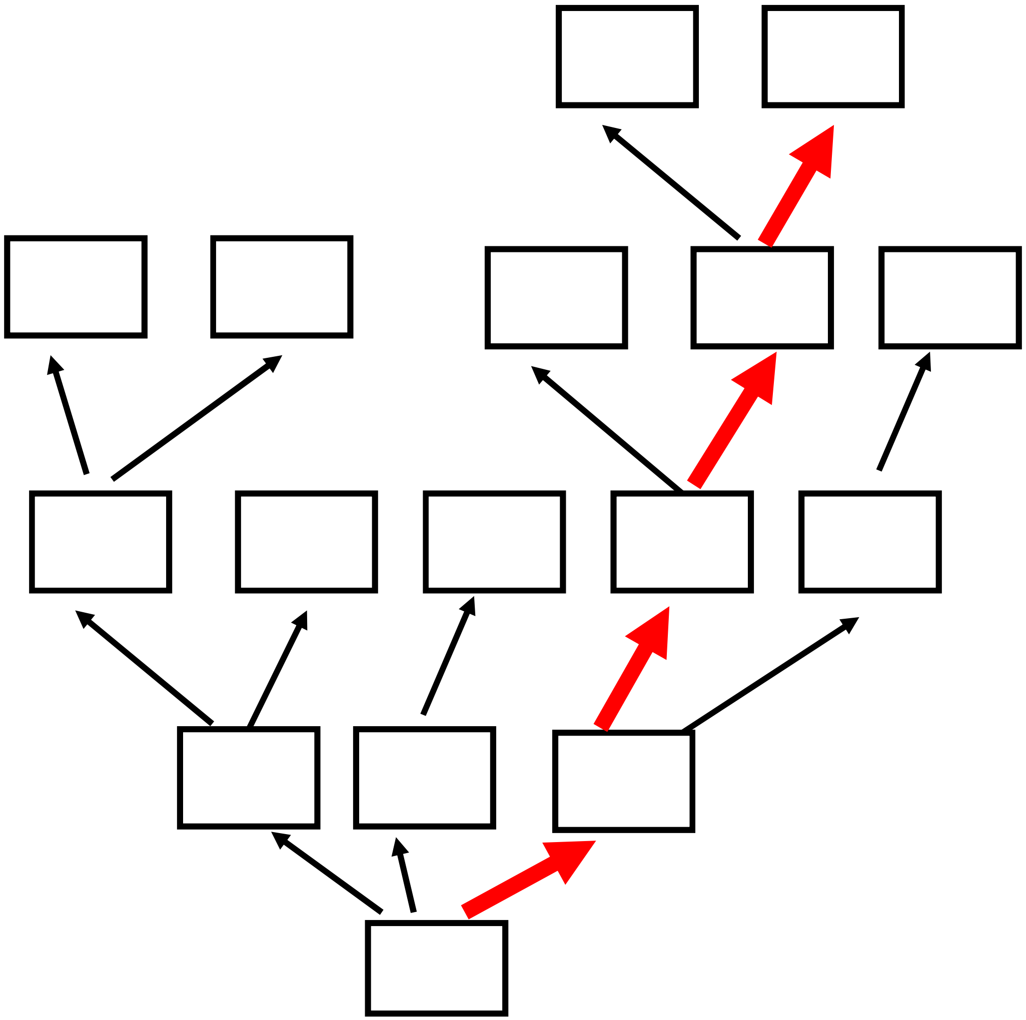


The height of a tree is the *maximum* length of a path from the root node to a leaf.



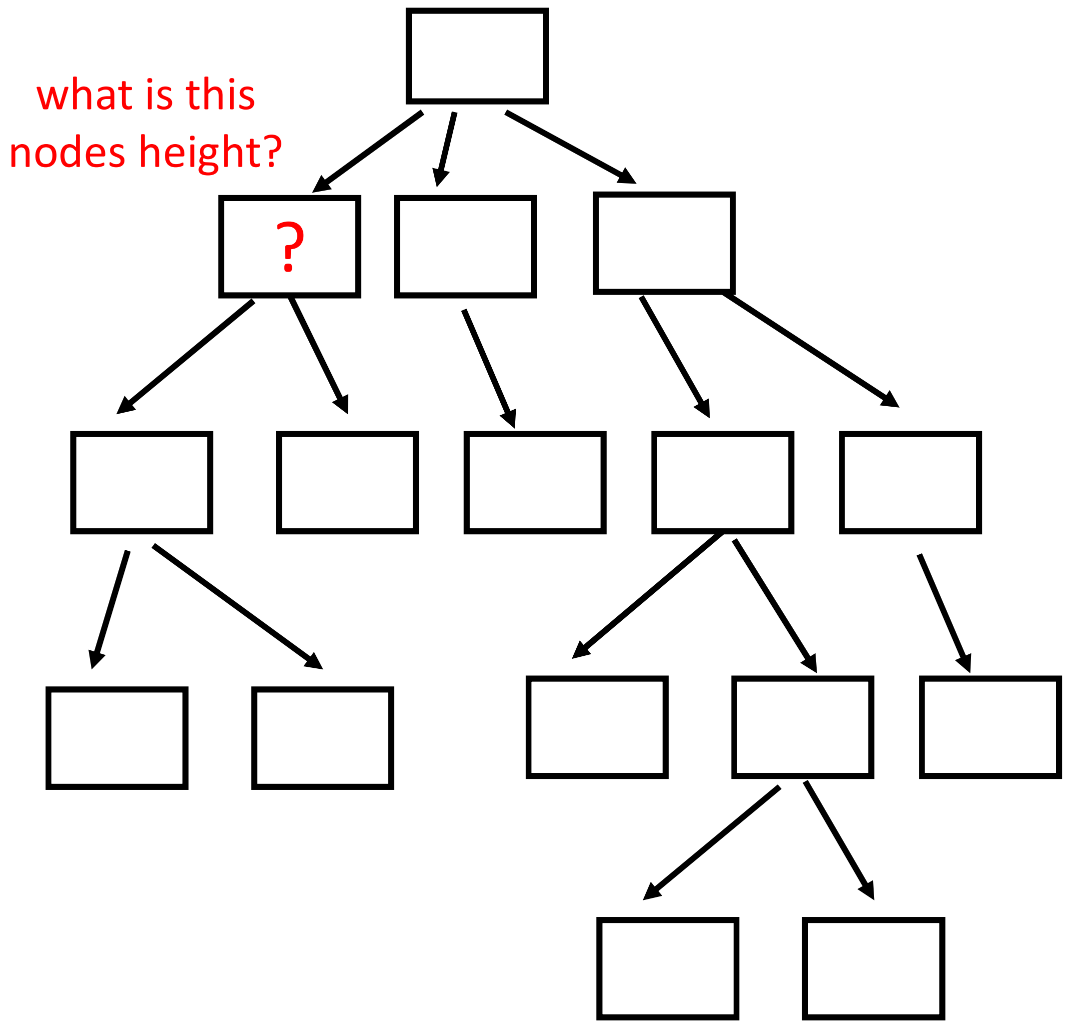
height is 4

The height of a tree is the maximum length of a path from the root node to a leaf. *To visualize height, we flip the tree upside down*.

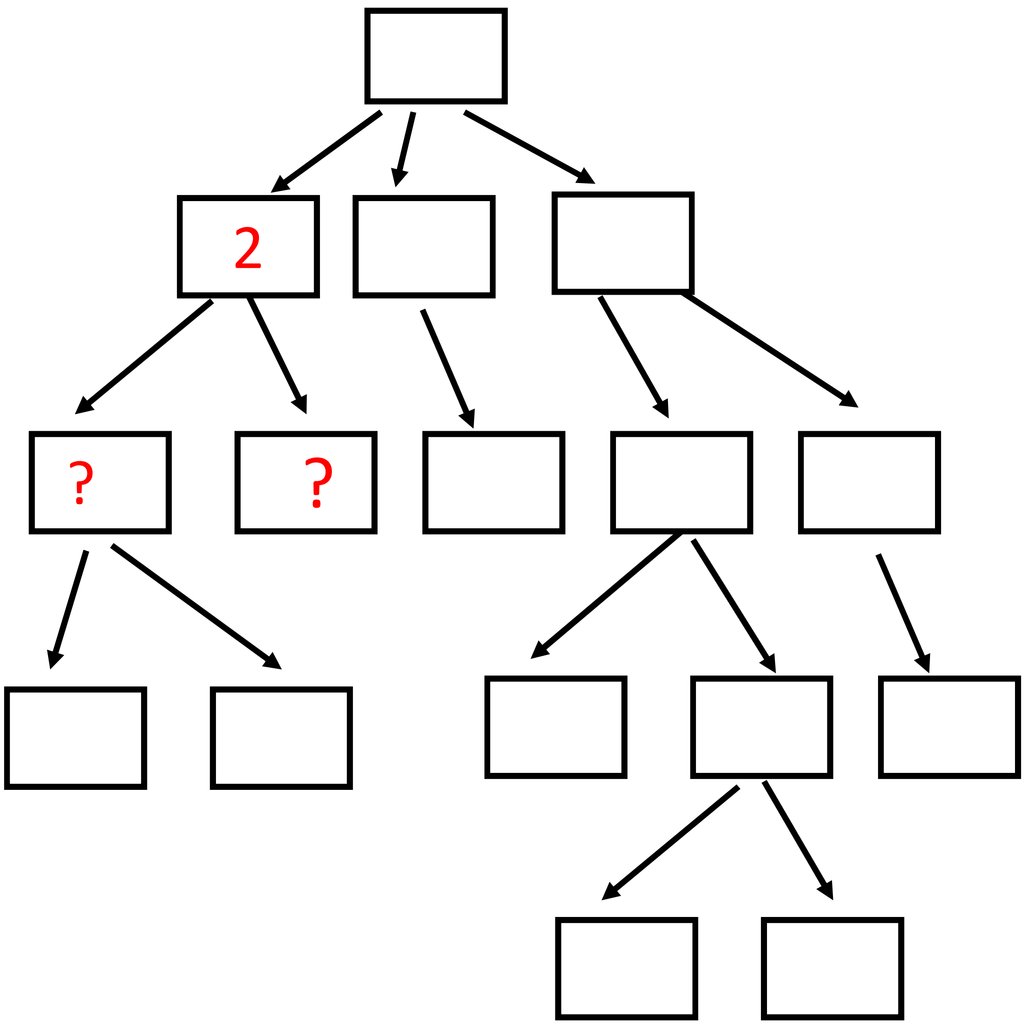


height is 4

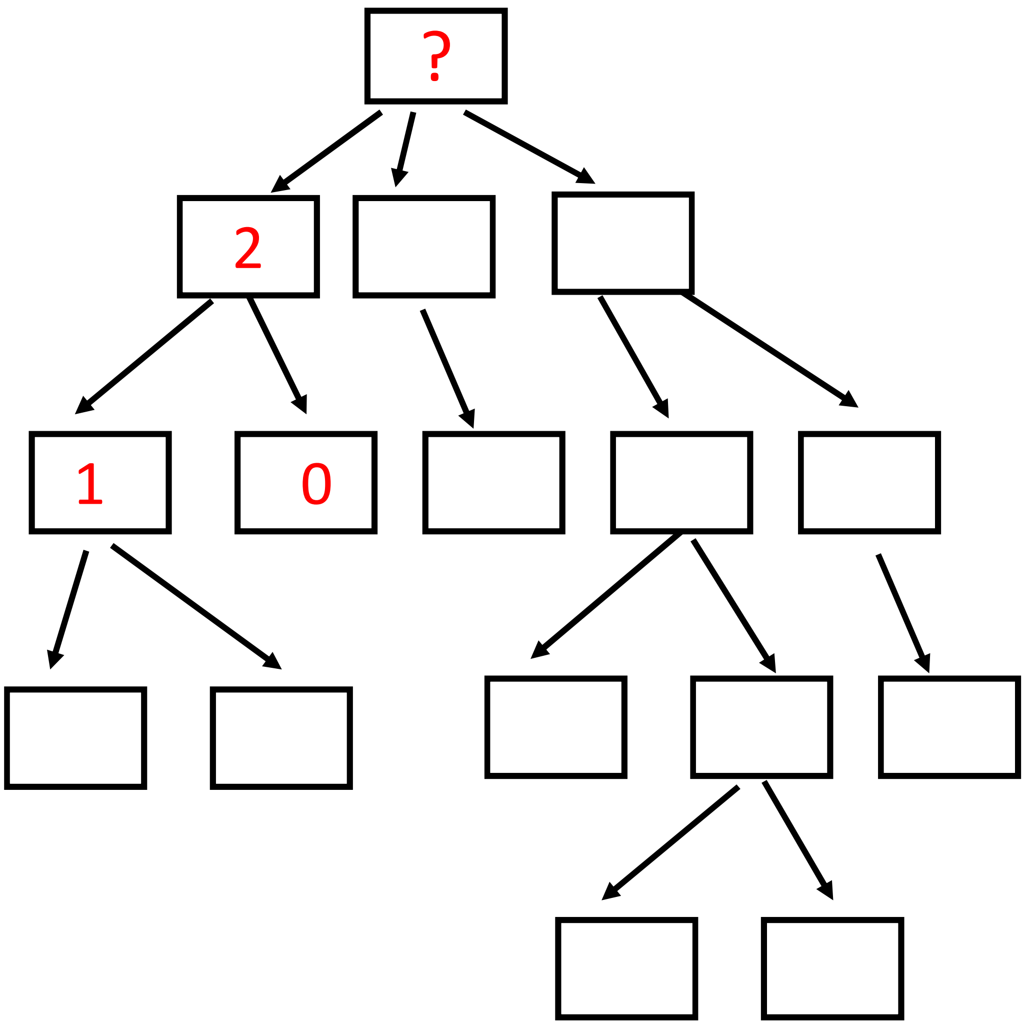
The height of a node is the maximum length of a path from that node to a leaf.   (It is the height of the subtree, rooted at that node.)



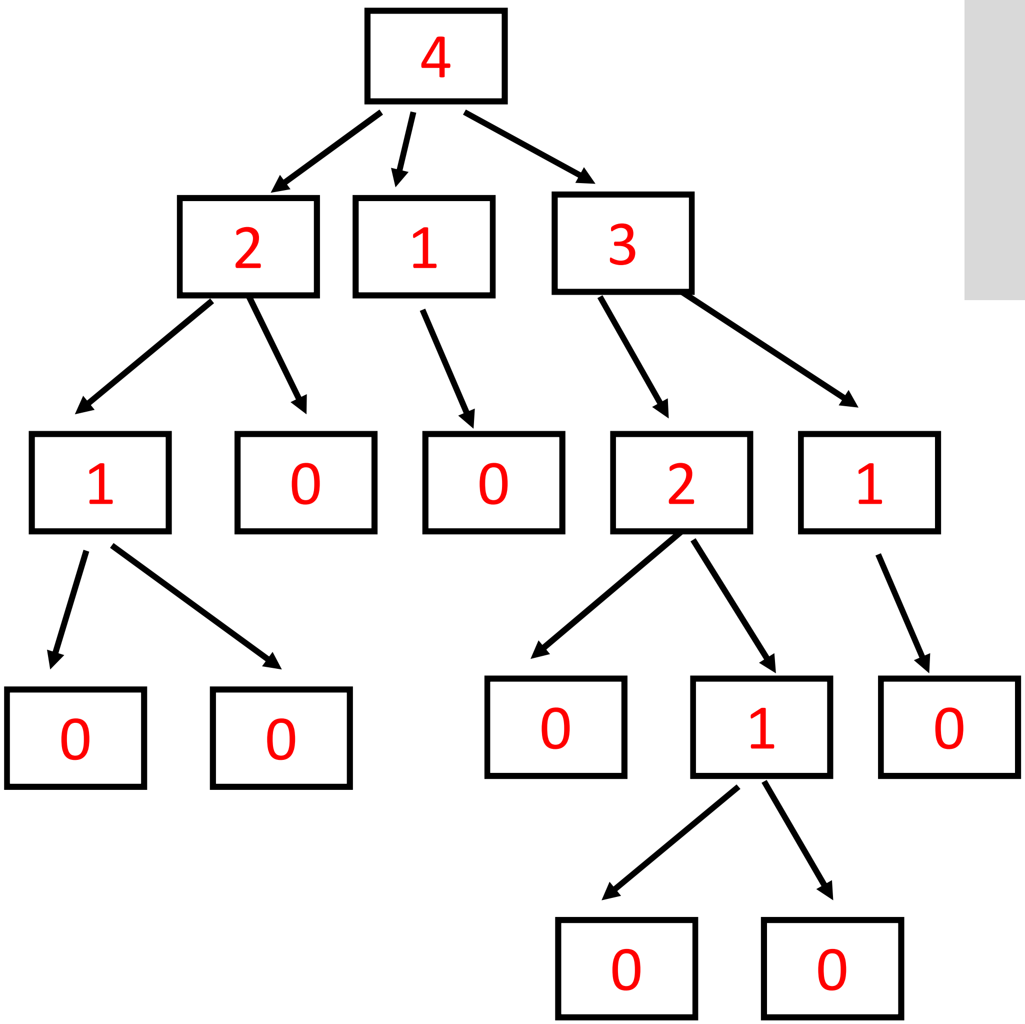
The height of a node is the maximum length of a path from that node to a leaf.   (It is the height of the subtree, rooted at that node.)



The height of a node is the maximum length of a path from that node to a leaf.   (It is the height of the subtree, rooted at that node.)

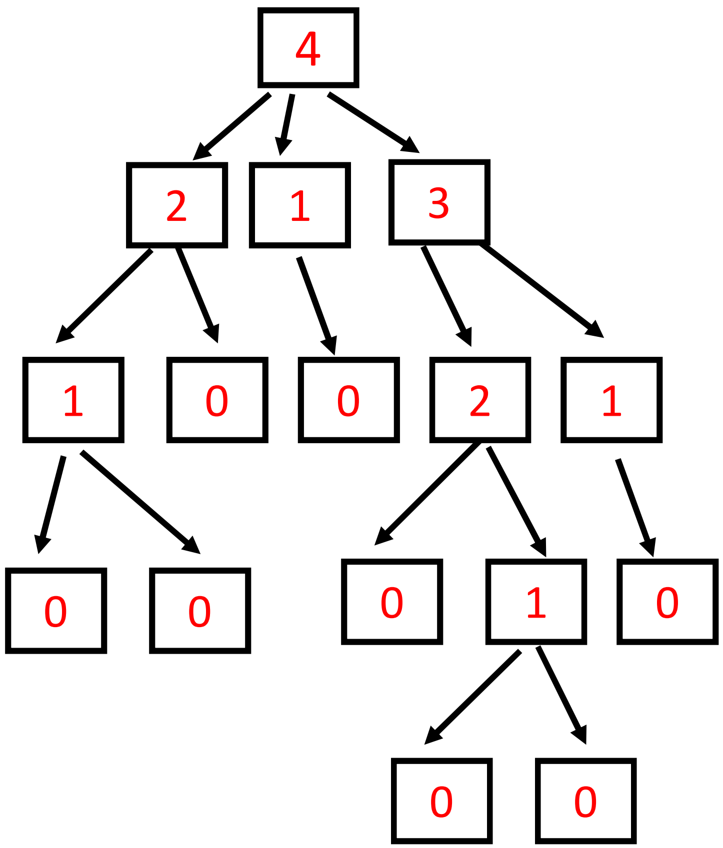


How to compute height(v) ?



Hint: think recursively

How to compute height(v) ?

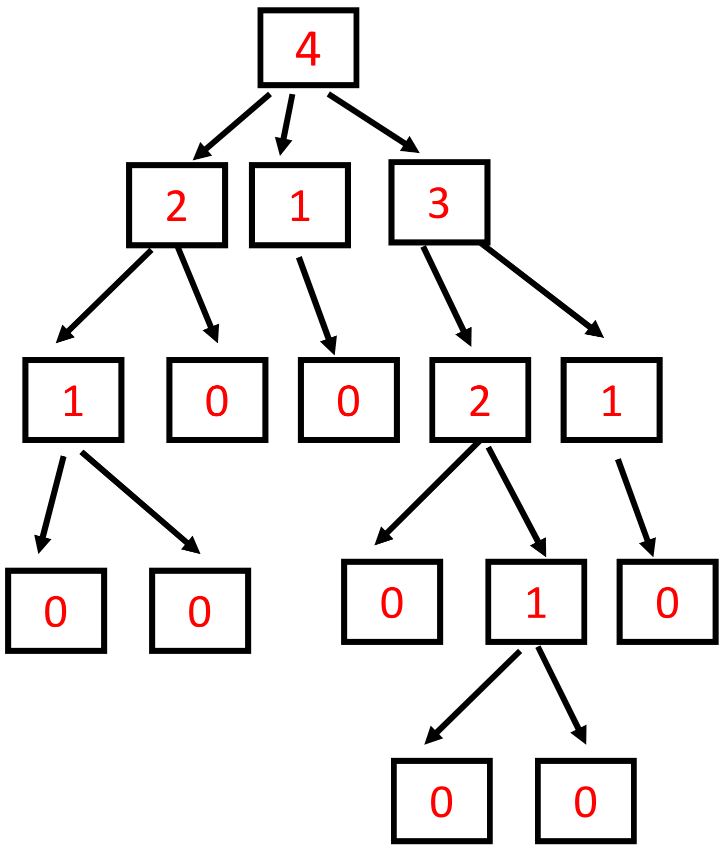



```
height(v){
    if (v is a leaf)
        return 0
    else{
```

Hint: think recursively

```
    }
}
```

How to compute height(v) ?




```
height(v){
    if (v is a leaf)
        return 0
    else{
        h = 0
        for each child w of v
            h = max(h, height(w))
        return 1 + h
    }
}
```

How to implement a tree in Java?

```
class  TreeNode<T>{

   T   element;

}
```

How to implement a tree in Java?

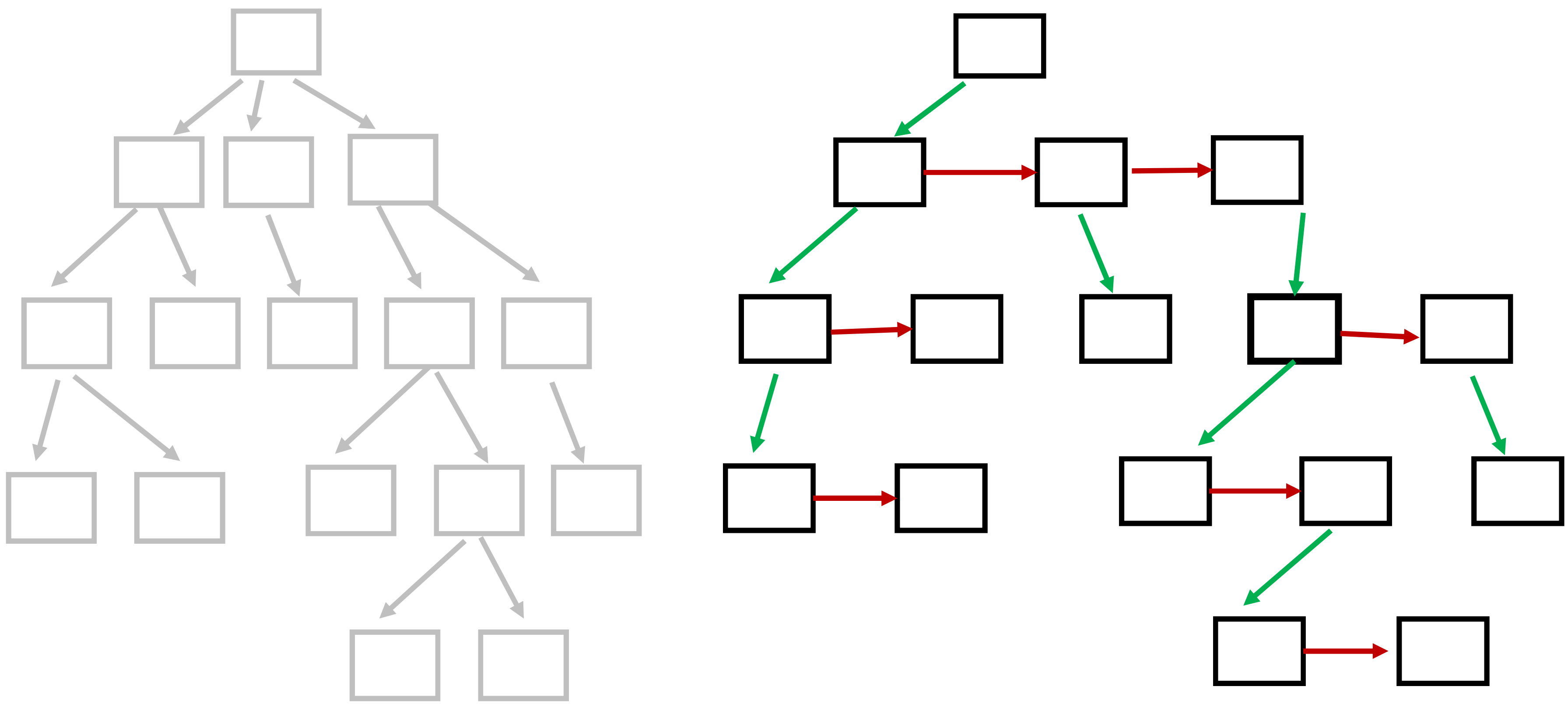
```
class  TreeNode<T>{

    T   element;

    ArrayList< TreeNode<T> >     children;

}
```

How to implement a tree in Java?

```
class  TreeNode<T>{

    T   element;

    ArrayList< TreeNode<T> >     children;

    TreeNode<T>          parent;         //  optional
}
```

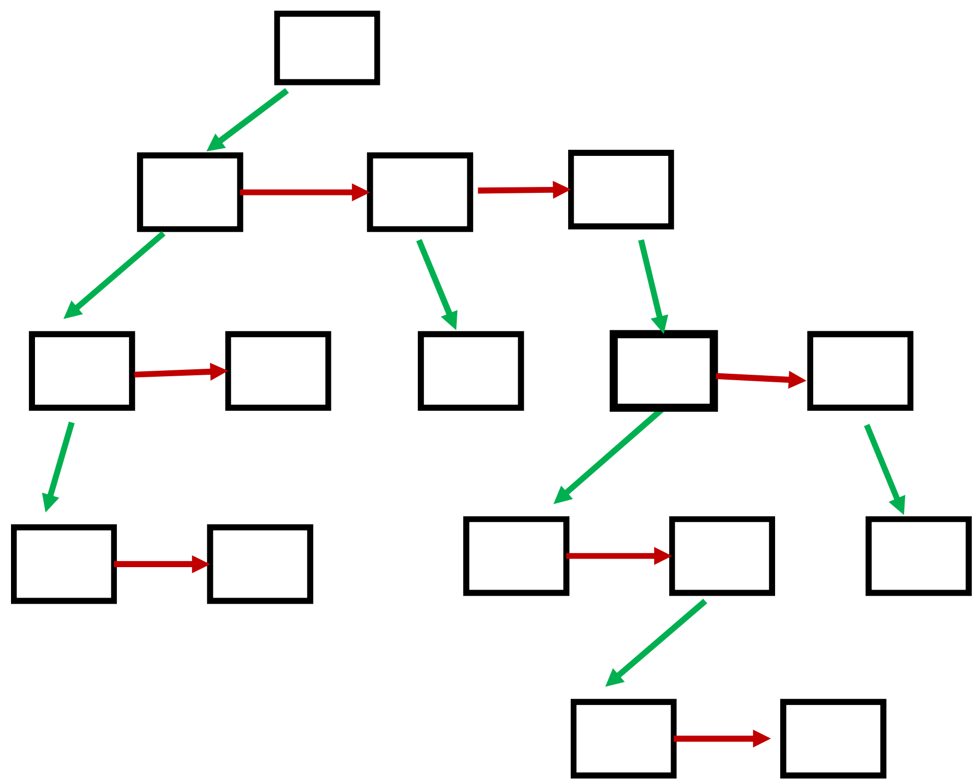
```
class  Tree<T>{          //   could be called  RootedTree
    TreeNode<T>   root;
          :                                //  other methods

   // inner class

   class  TreeNode<T>{
        T   element;
        ArrayList< TreeNode<T> >     children;
        TreeNode<T>        parent;       //  optional
          :
    }
}
```

Another common implementation:
‘first child, next sibling’
(similar to singly linked lists)



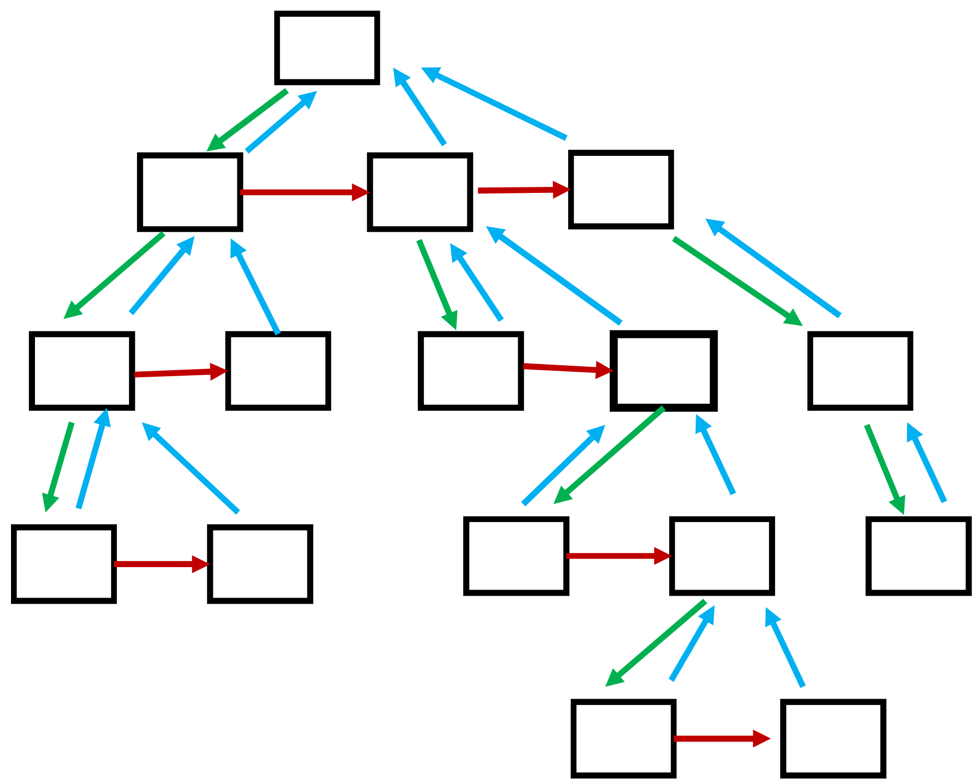
Another common implementation:
'first child, next sibling'
(similar to singly linked lists)

```
class  Tree<T>{
   TreeNode<T>   root;
       :

   // inner class

   class  TreeNode<T>{
     T   element;
     TreeNode<T>   firstChild;
     TreeNode<T>   nextSibling;
         :
   }
}
```

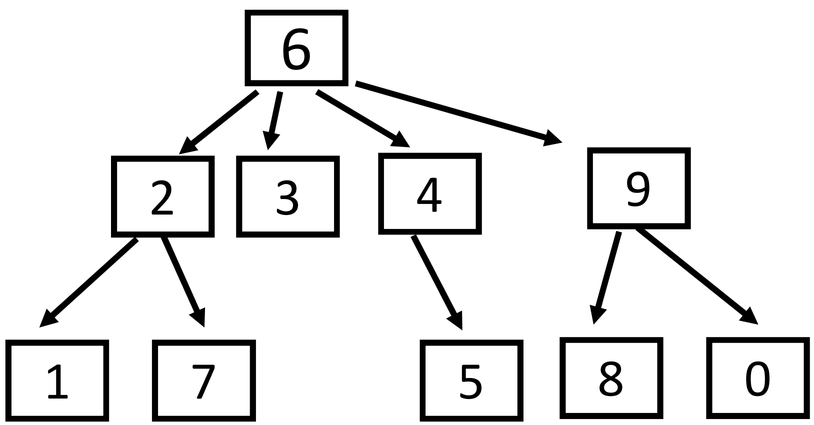


Another common implementation:
'first child, next sibling'

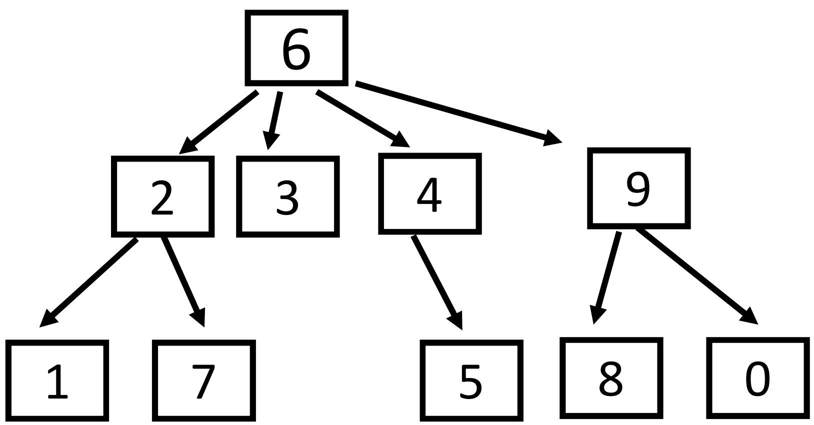
```
class Tree<T>{
    TreeNode<T>   root;
       :

    // inner class

   class TreeNode<T>{
     T   element;
     TreeNode<T>   firstChild;
     TreeNode<T>   nextSibling;
     TreeNode<T>   parent;         :
   }
}
```



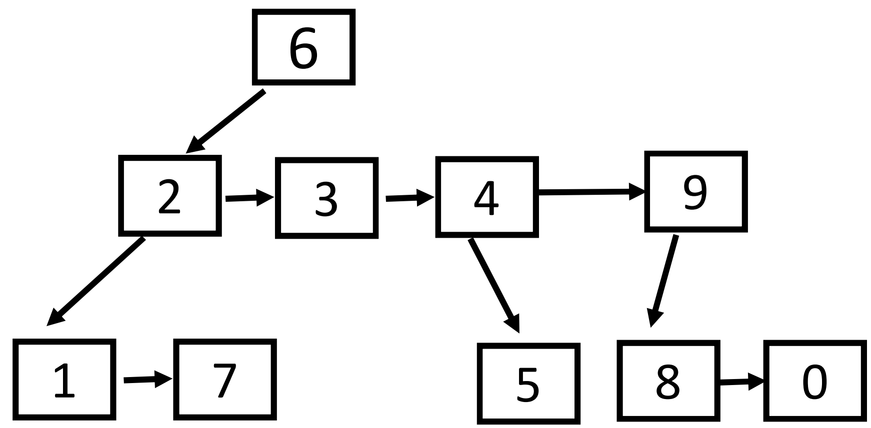
Exercise:
redraw this tree using first child, next sibling



Exercise:
redraw this tree using first child, next sibling



first child, next sibling



Exercise

A tree can be represented using *nested lists*, as follows:

```
tree    =     root   |    ( root listOfSubTrees )

listOfSubTrees   =   tree | tree listOfSubTrees
```

Note that `listOfSubTrees` cannot be empty.

[Updated March 9] Moreover, open brackets ( ) must have at least two elements inside.

Exercise

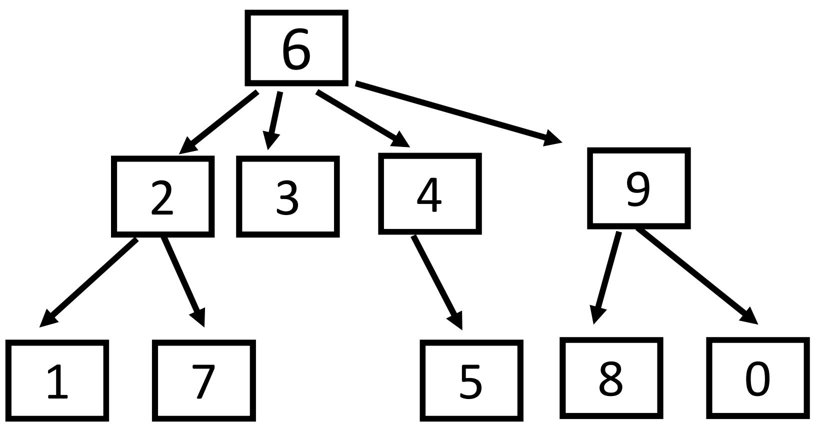
A tree can be represented using *nested lists*, as follows:

```
tree    =     root  |   ( root listOfSubTrees )

listOfSubTrees  =  tree | tree listOfSubTrees
```

Example: Draw the tree that corresponds to the following list, where the elements are single digits.

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

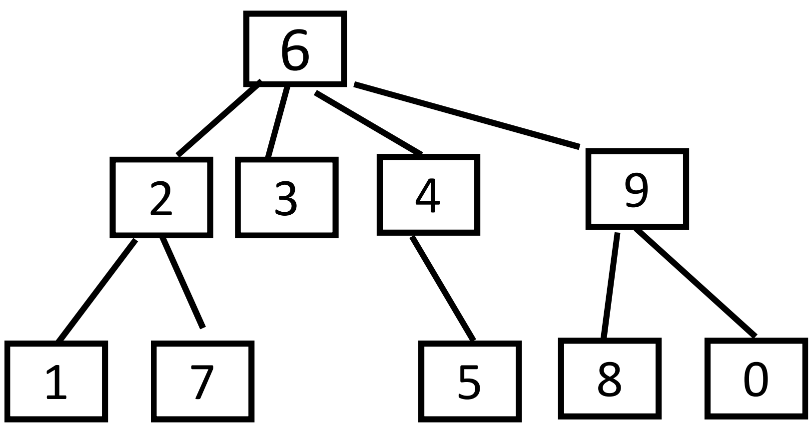
Exercise

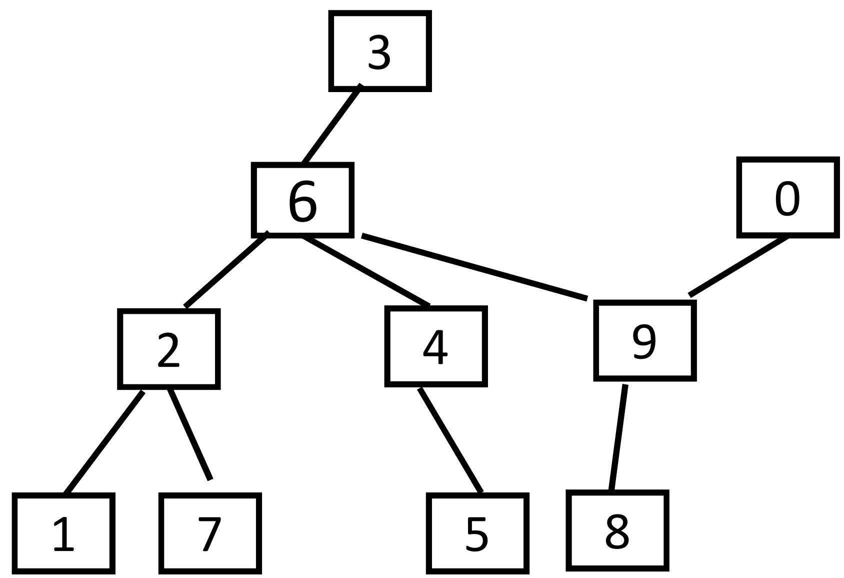


(6 (2 1 7) 3 (4 5) (9 8 0))

ASIDE: Non-rooted trees

You will see non-rooted trees in COMP 251.
For these trees, there is no natural way to define the 'root'.
e.g. The tree on the left is only rooted because I drew it that way.
It is the same (non-rooted) tree as the one on the right.
Edge direction is ignored for this example.





Coming up…

Lectures

Wed. March 9

Tree Traversal

Fri. March 11

Expression Trees

Assessments

Yesterday/Today

Quiz 3

Assignment 3

due Wed. March 16