COMP 250

Lecture 19

(rooted) trees

Oct. 23, 2017

Linear Data Structures

linked list

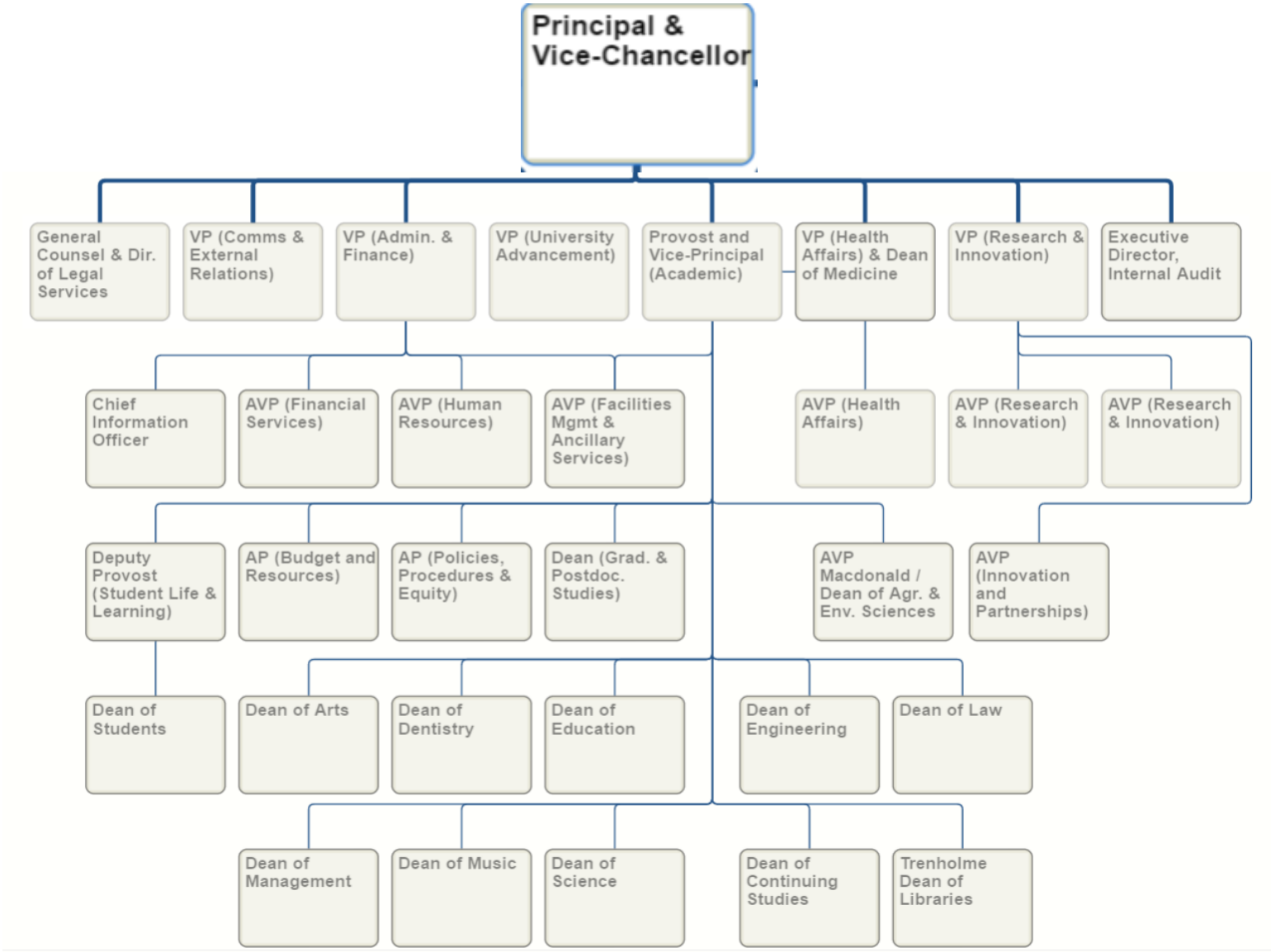
array

Non-Linear Data Structures

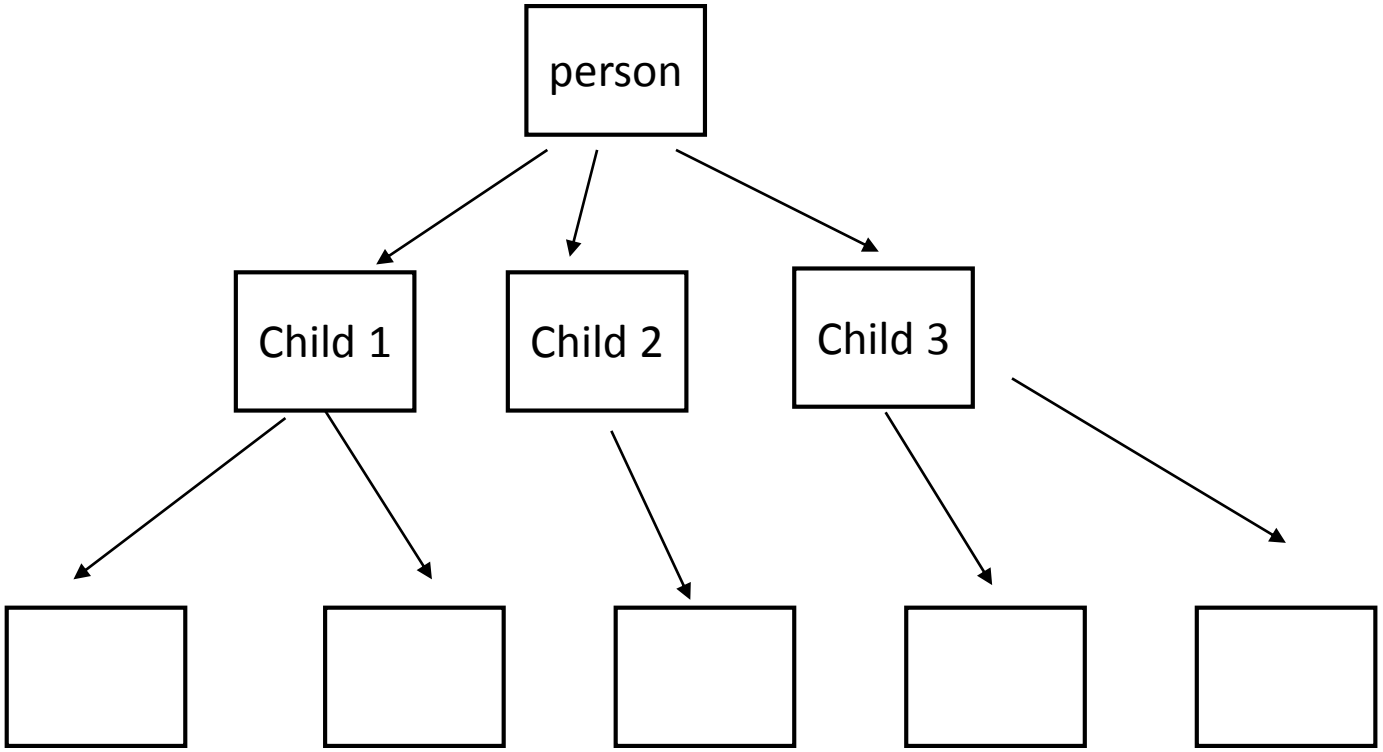
tree

graph

Tree Example: Organization Hierarchy (McGill)

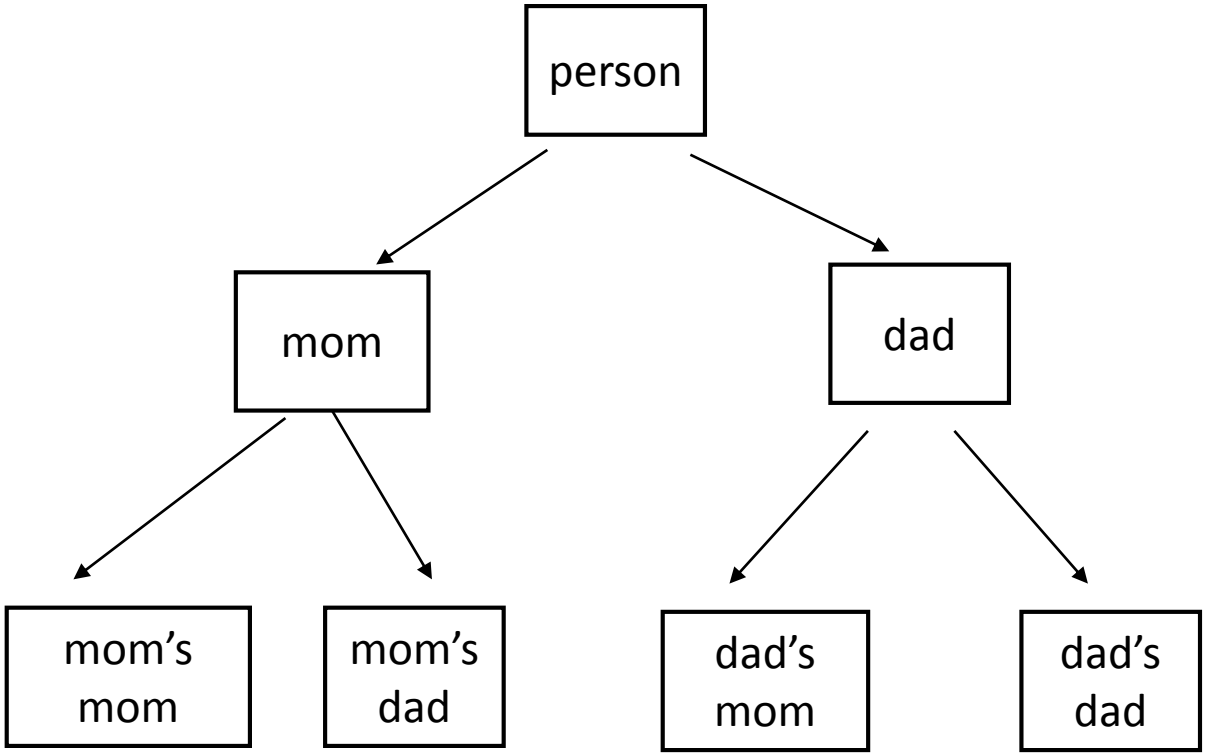


Family Tree (descendents)



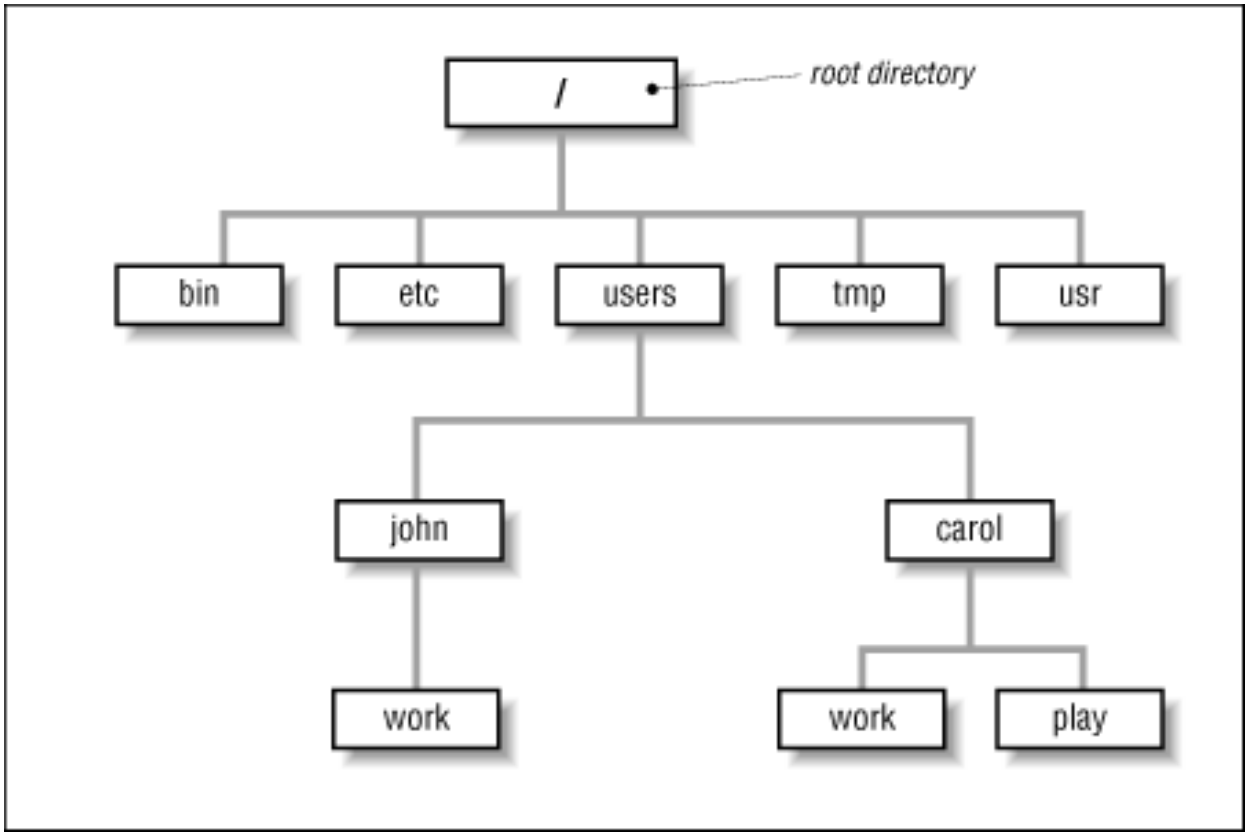
Here I ignore spouses (partner).

Family Tree (ancestors)

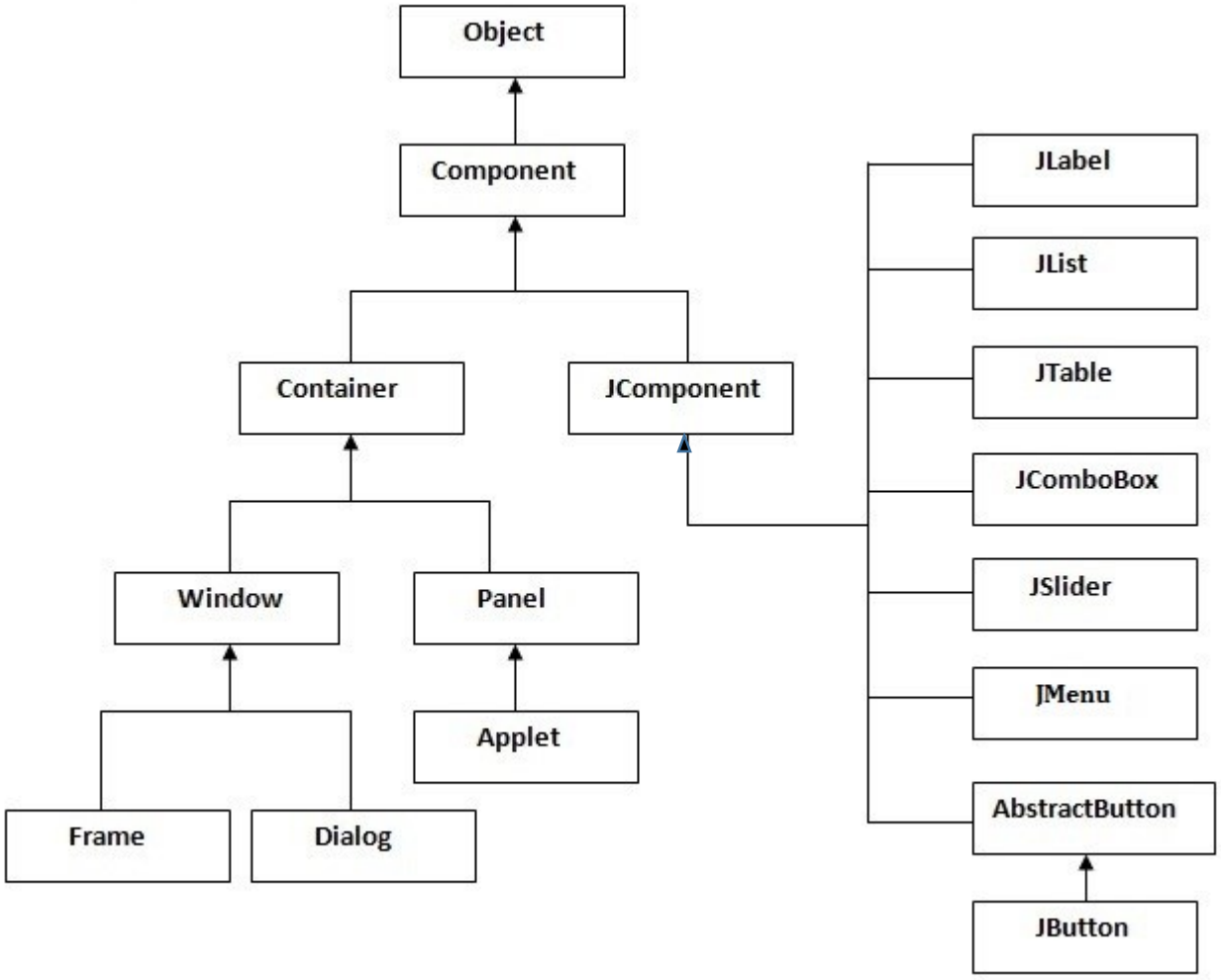


This is an example of a binary tree.

Tree: UNIX file system

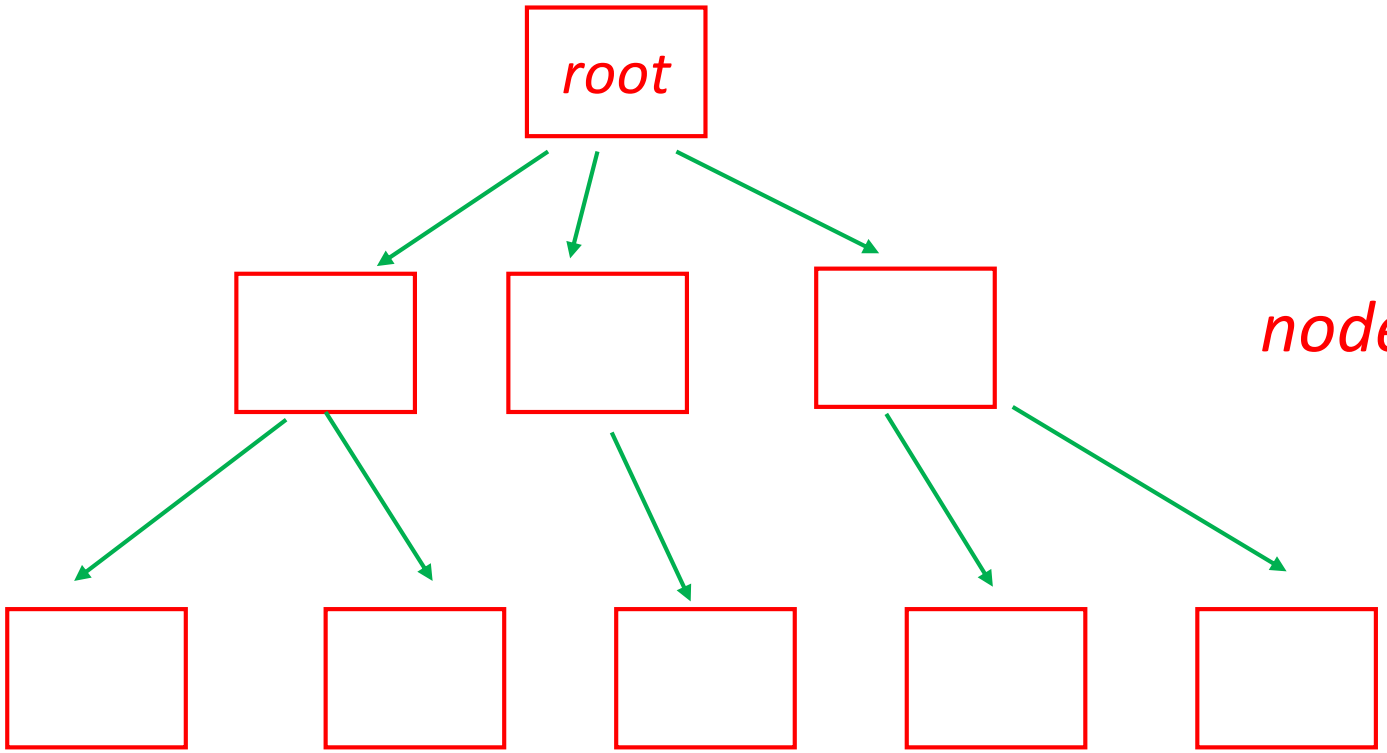


Tree: Java Classes e.g. GUI



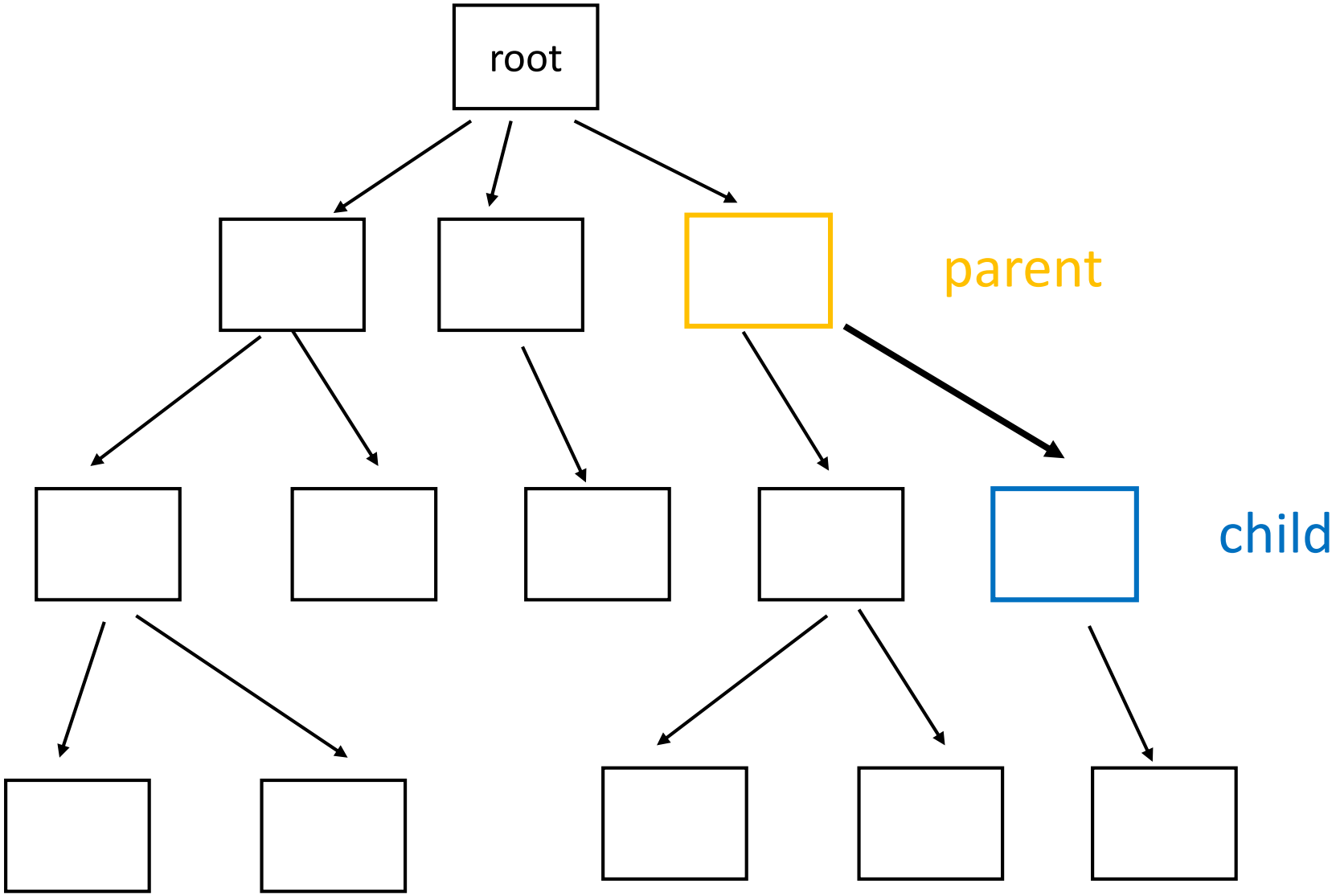
Tree Terminology

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



node, vertex

Every node except the root is a child, and has exactly one parent.



For some trees,

- edges are from parent to child
- edges are from child to parent
- edges are both from parent to child and child to parent.
- edge direction is ignored e.g. common with non-rooted trees (see final slide of today)

For some trees,

- edges are from parent to child

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

- edges are from child to parent
- edges are both from parent to child and child to parent.
- edge direction is ignored e.g. common with non-rooted trees (see final slide of today)

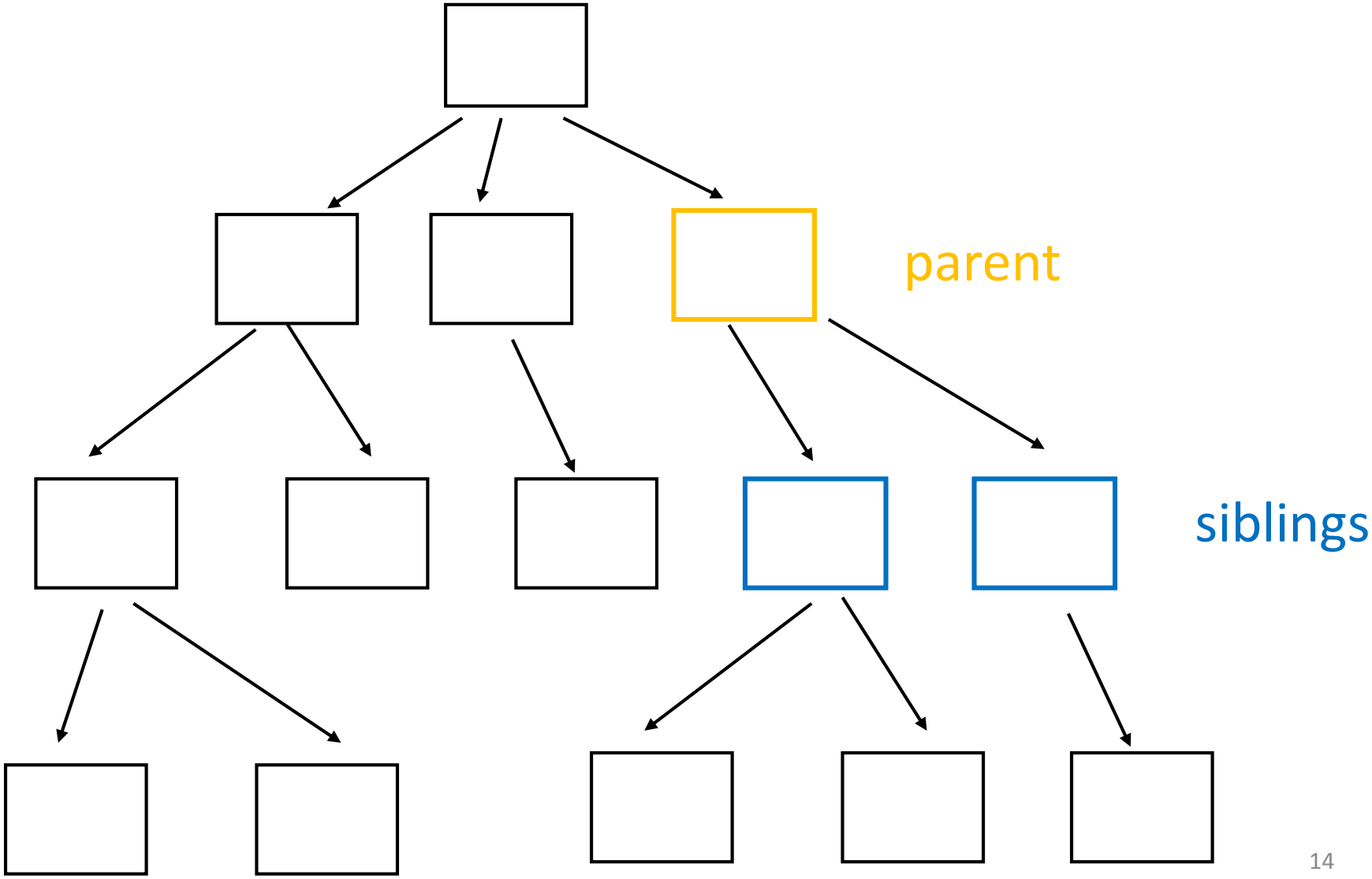
Q: If a tree has N nodes, how many edges does it have ?

Q: If a tree has N nodes, how many edges does it have ?

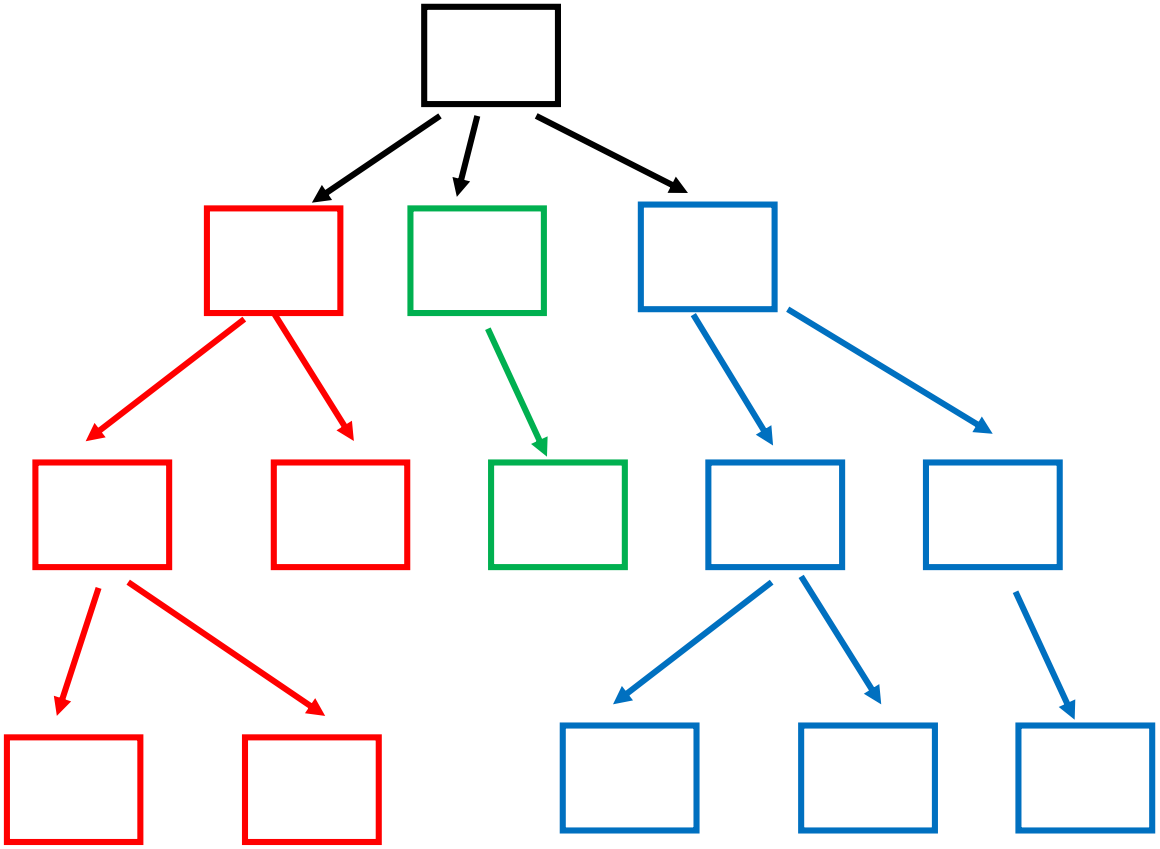
A: N-1

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.

Two nodes are siblings if they have the same parent.



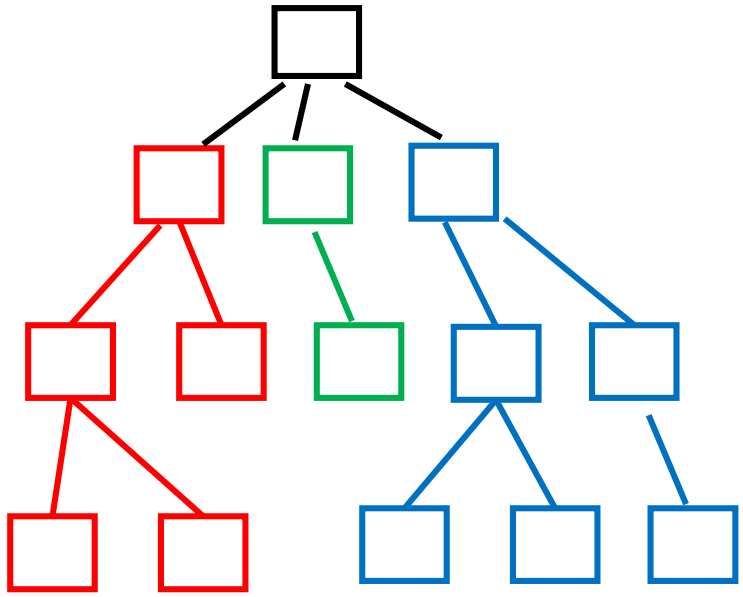
Recursive definition of rooted tree…



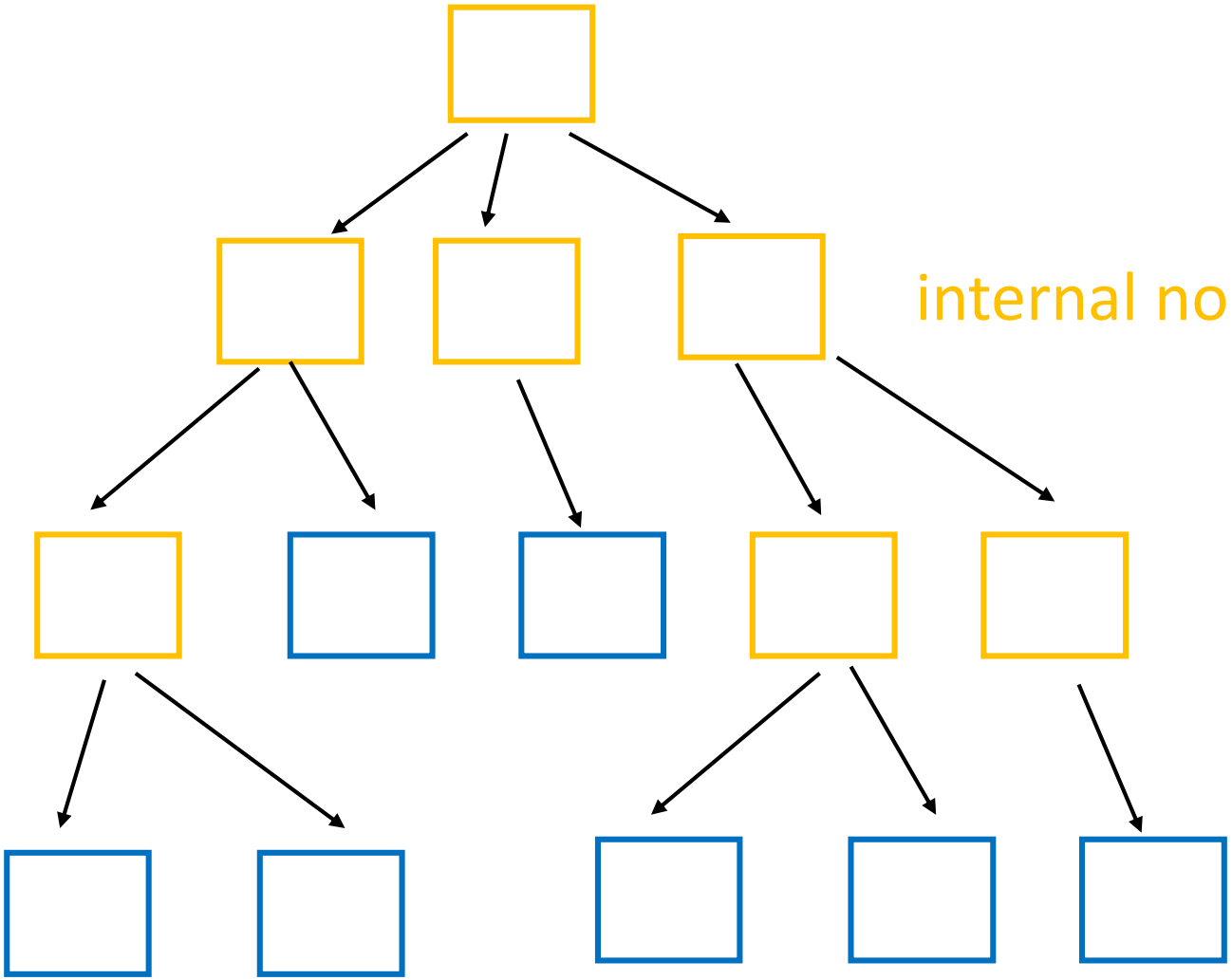
Recursive definition of rooted tree

A tree T is a finite (& possibly empty) set of n 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.



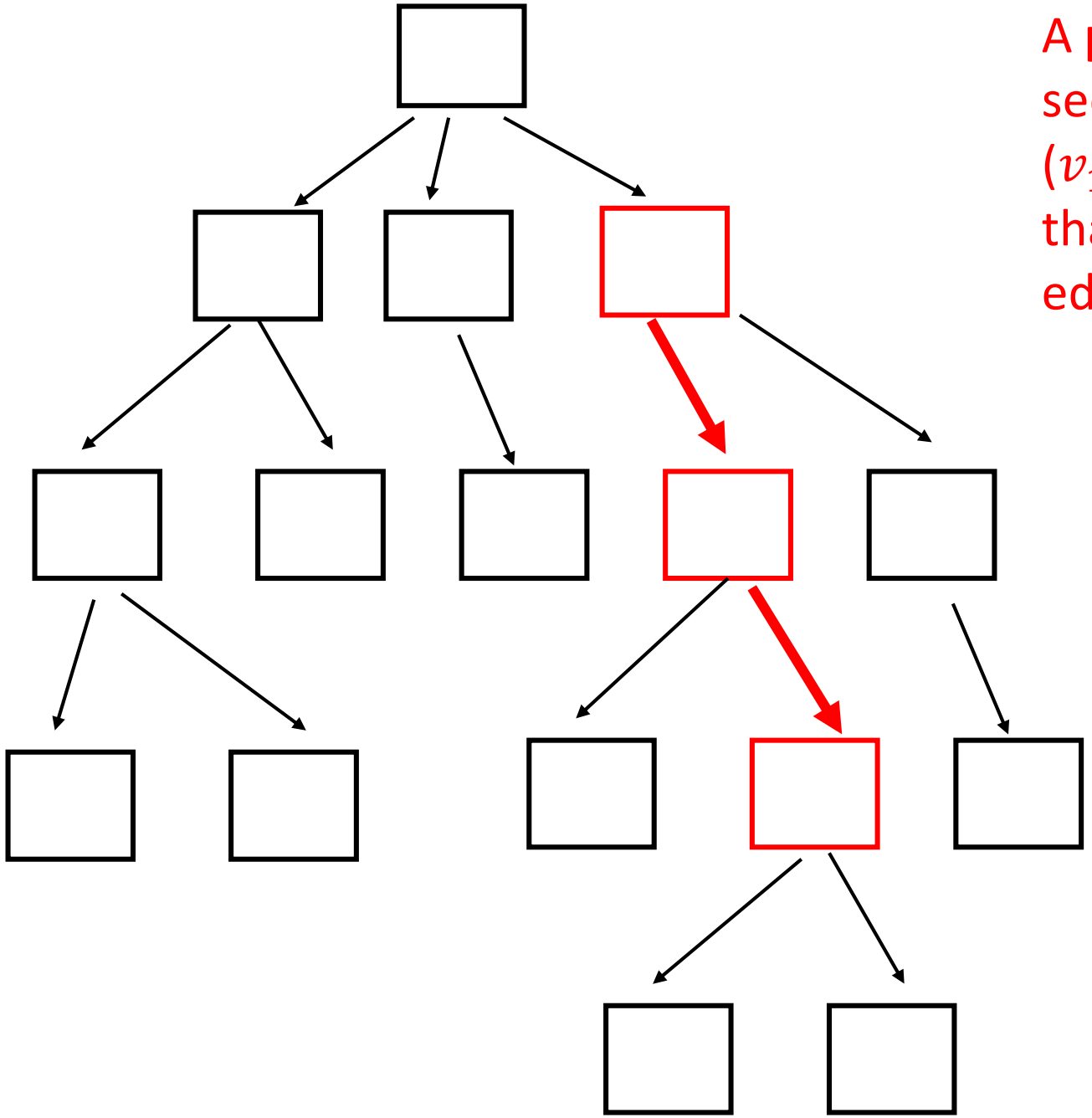
This definition does NOT assume edges are from parent to child.



internal nodes (e.g. file directories)

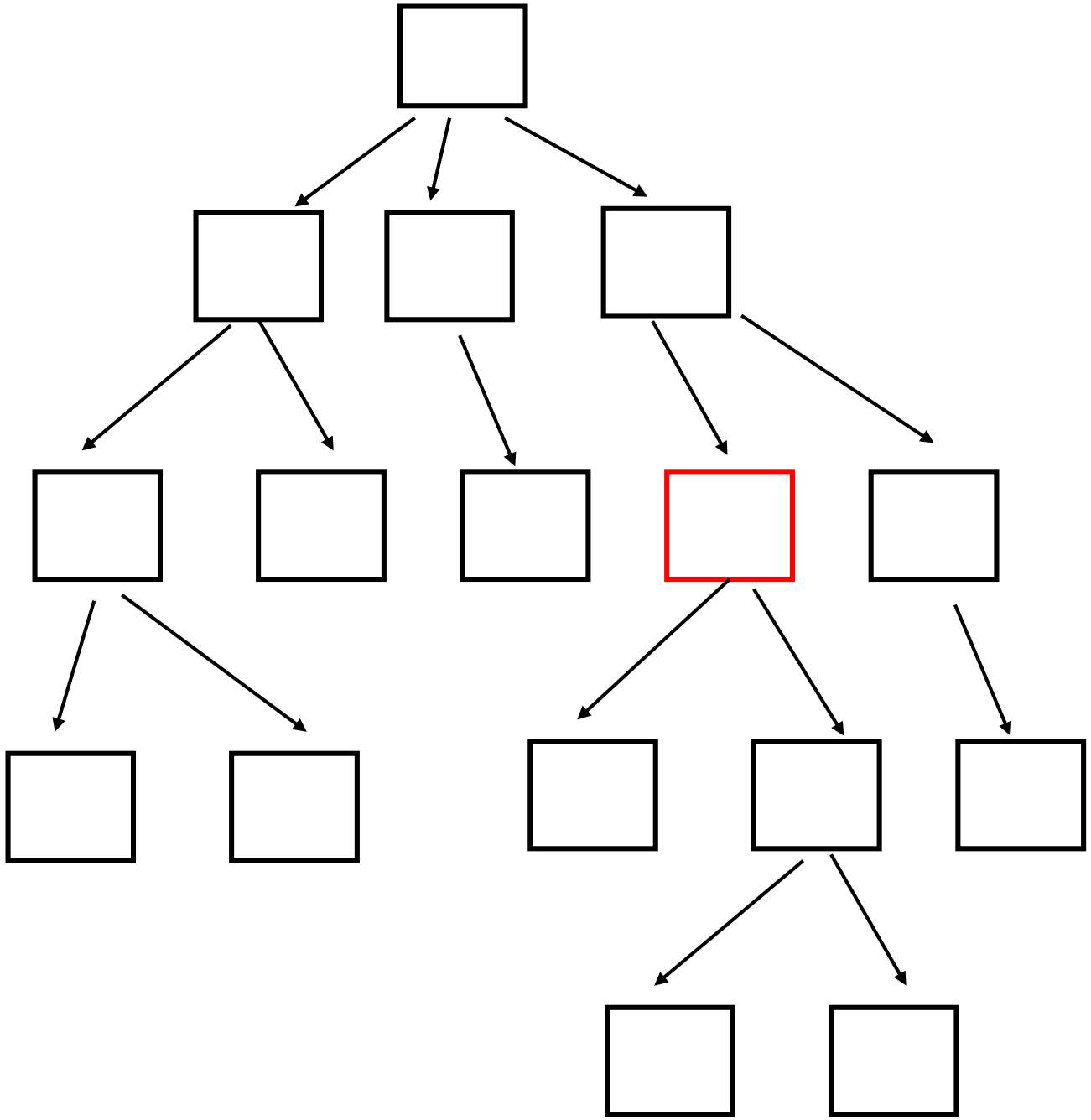
external nodes, leaves

(e.g. files)

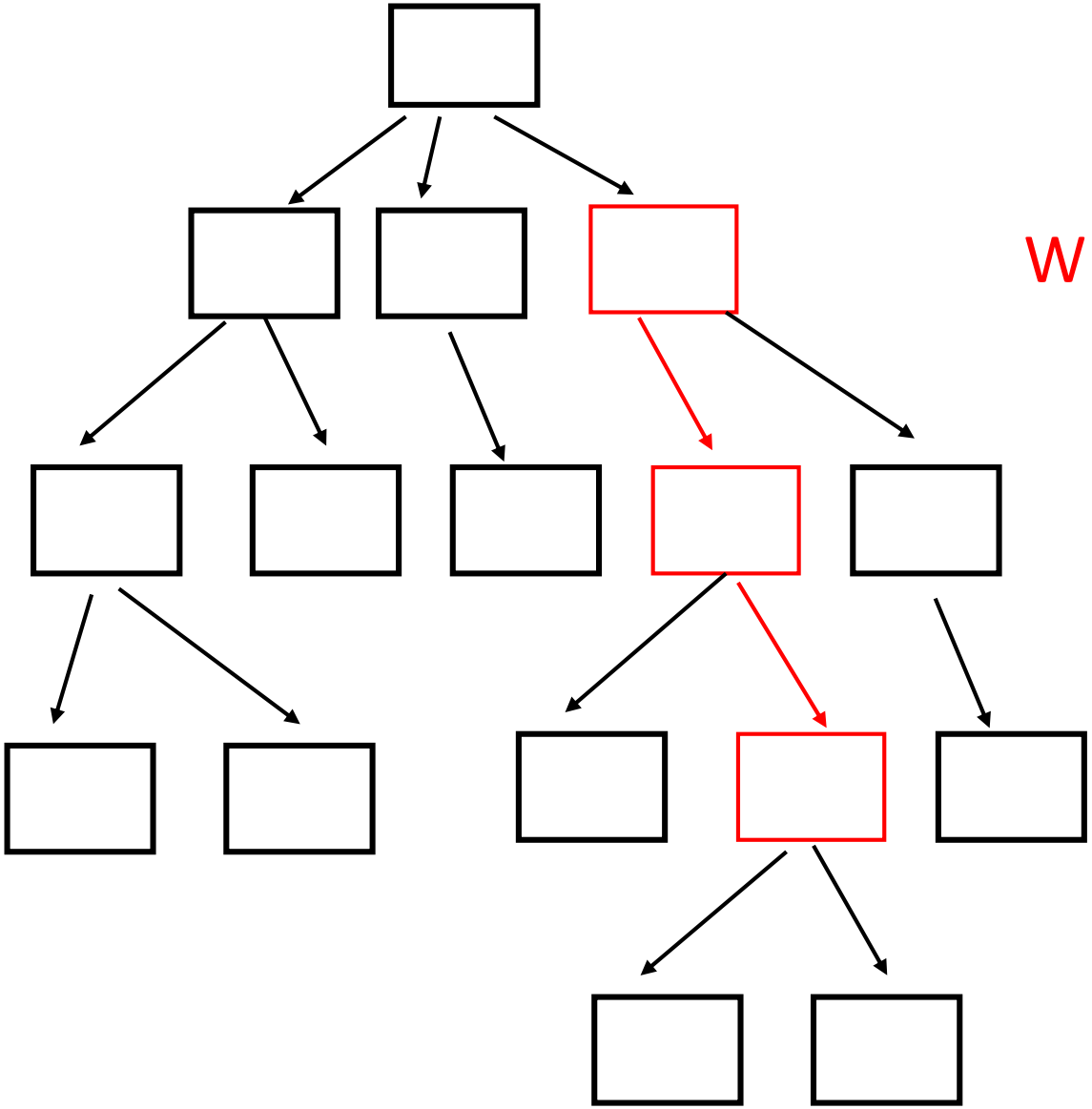


A **path** in a tree is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that (v_i, v_{i+1}) is an edge.

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

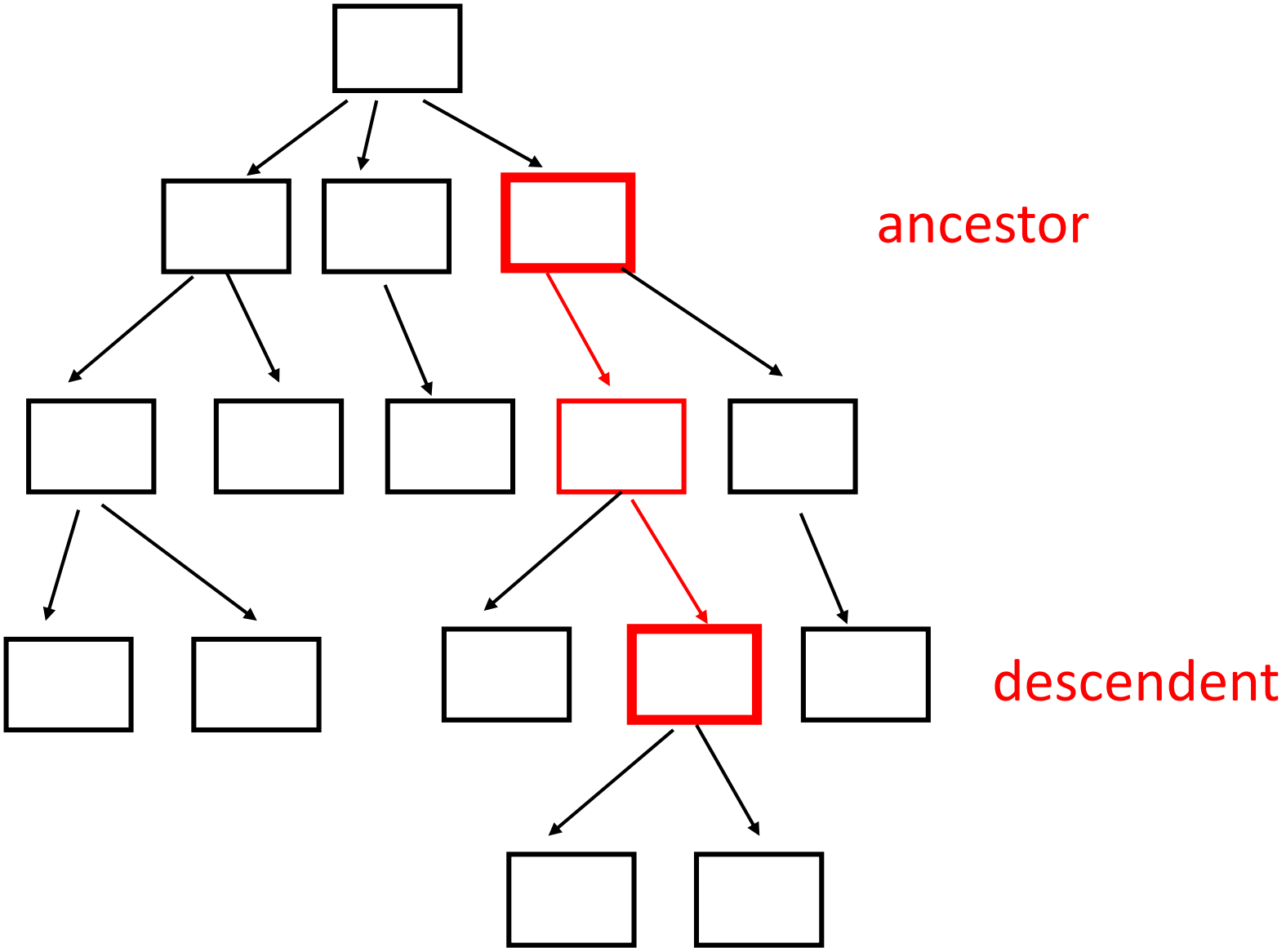


A path with just one node (v_1) has length = 0, since it has no edges.

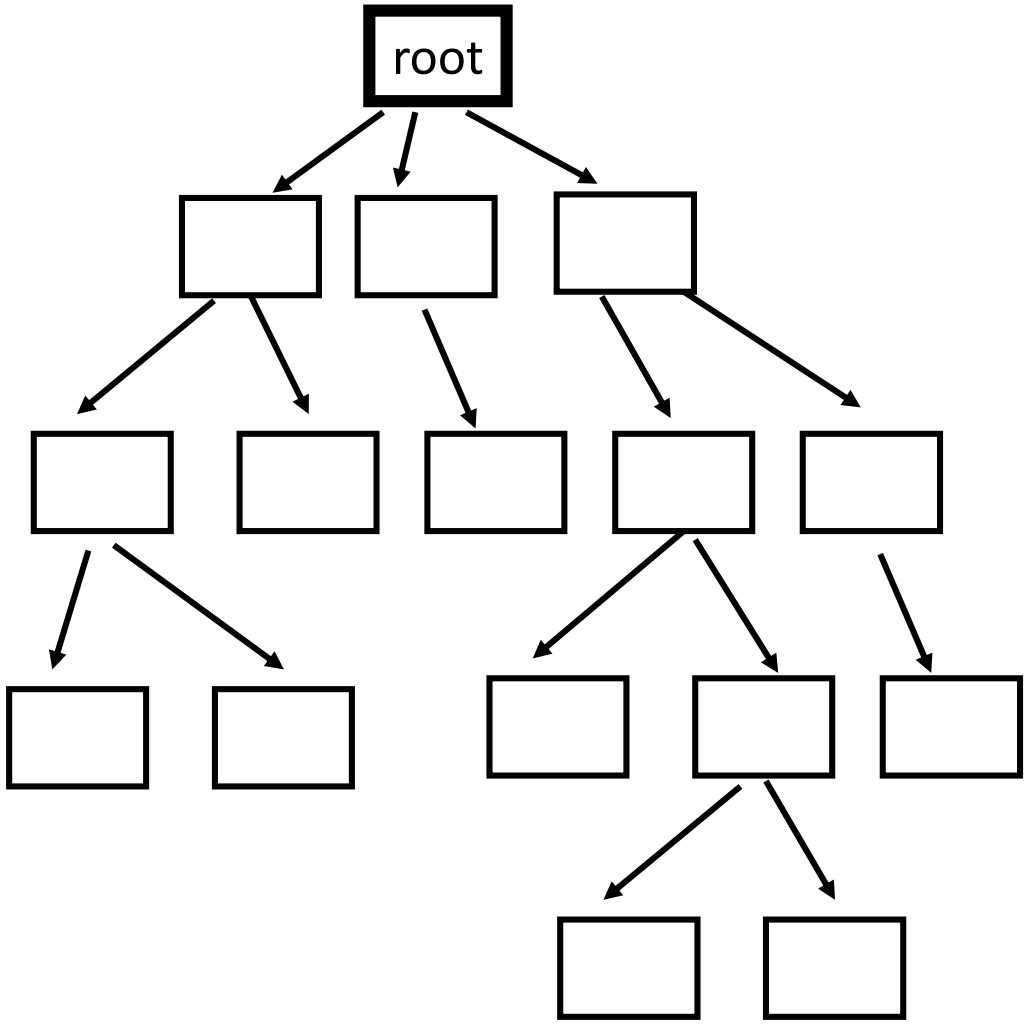


What is the path length?
(2 or 3?)

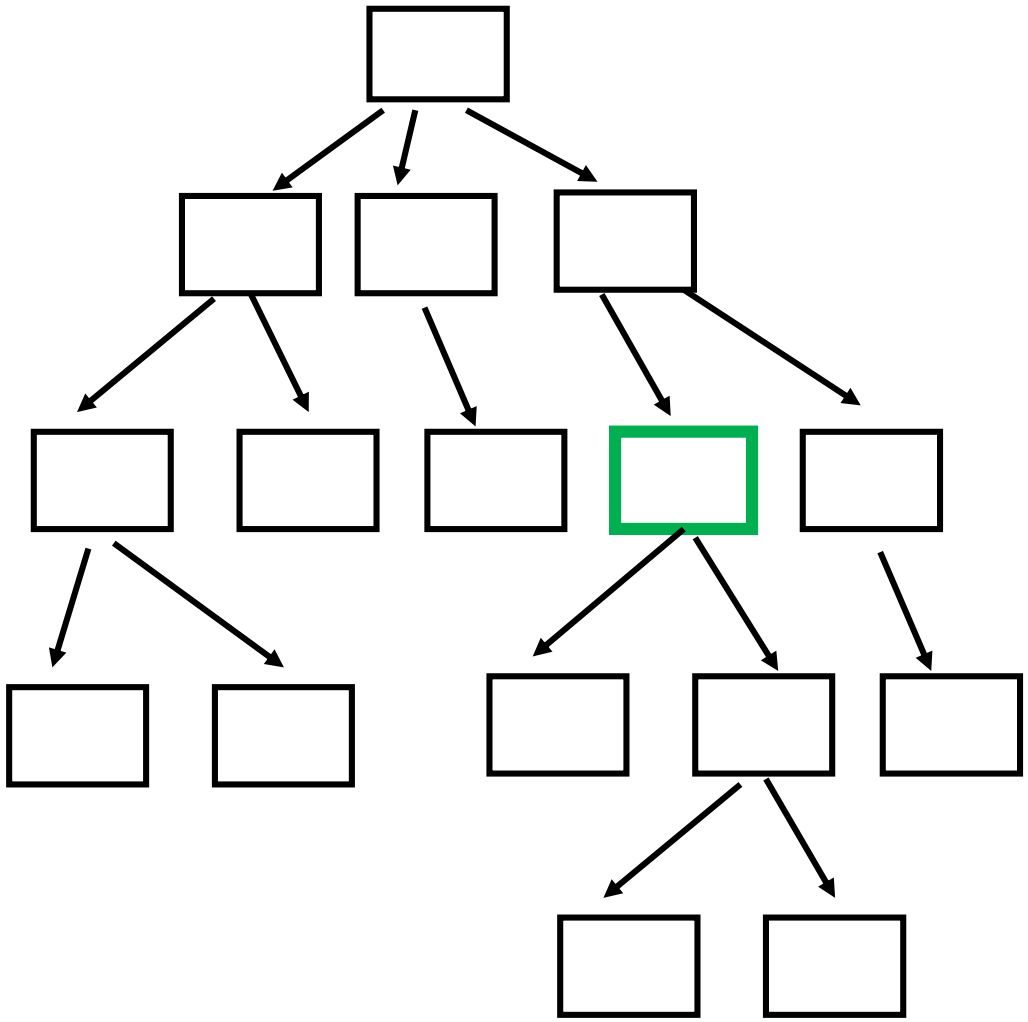
Node v is an ancestor of node w if there is a path from v to w.
Node w is a descendent of node v.



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



How to compute depth(**v**) ?



depth (level)

0

1

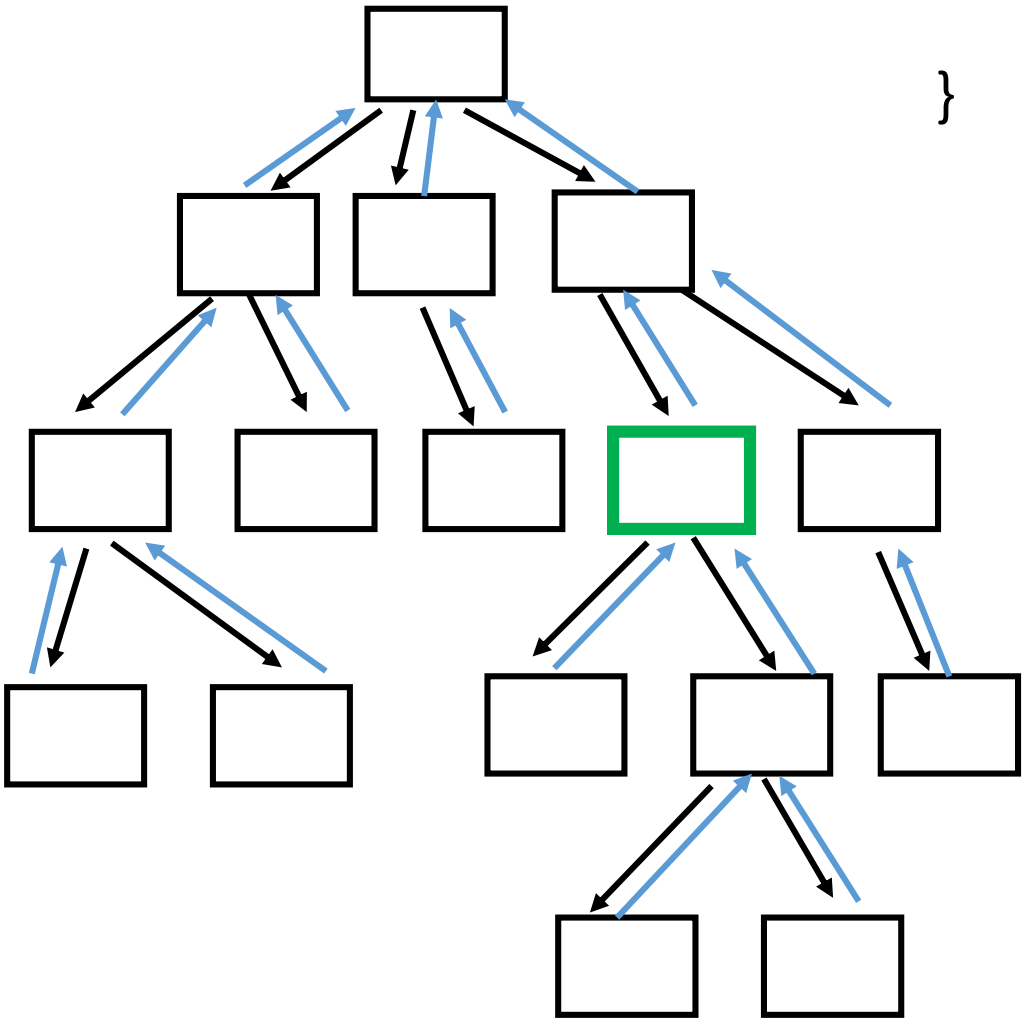
2

3

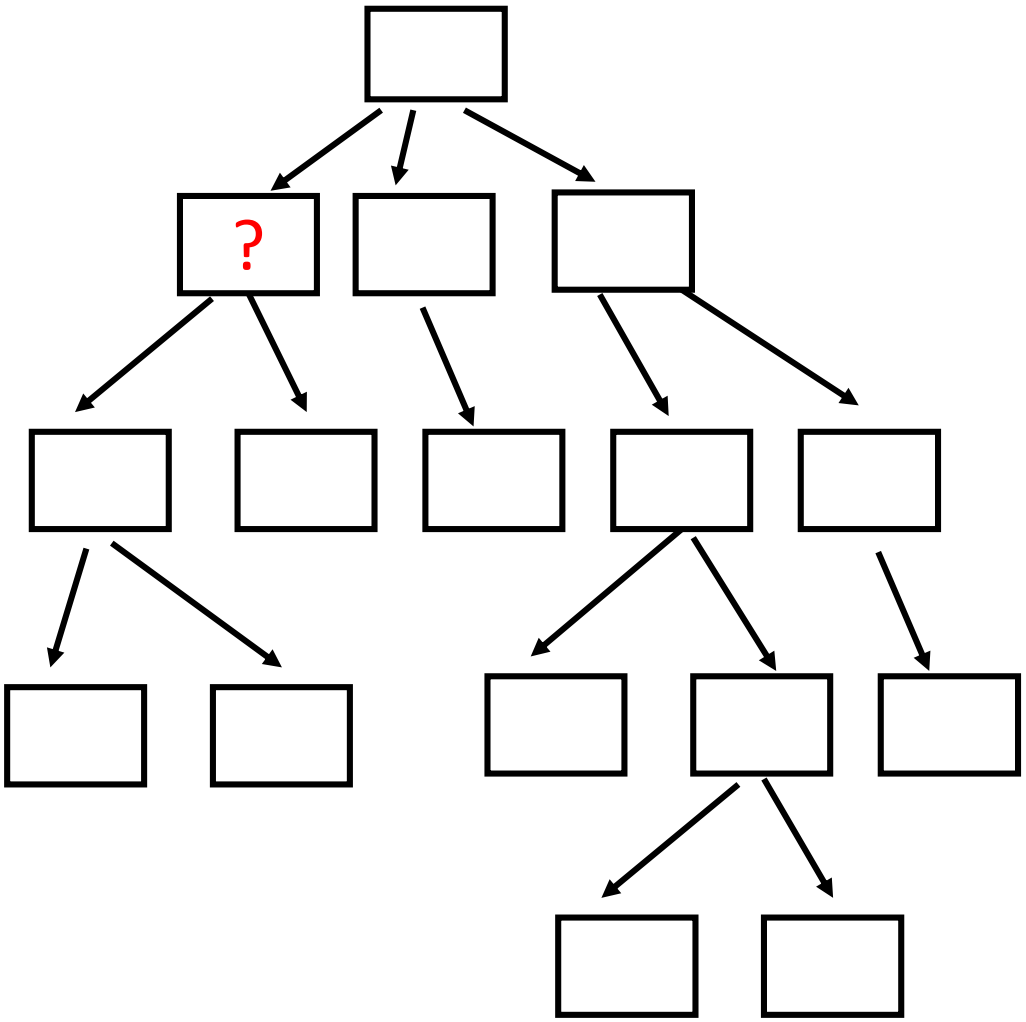
4

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

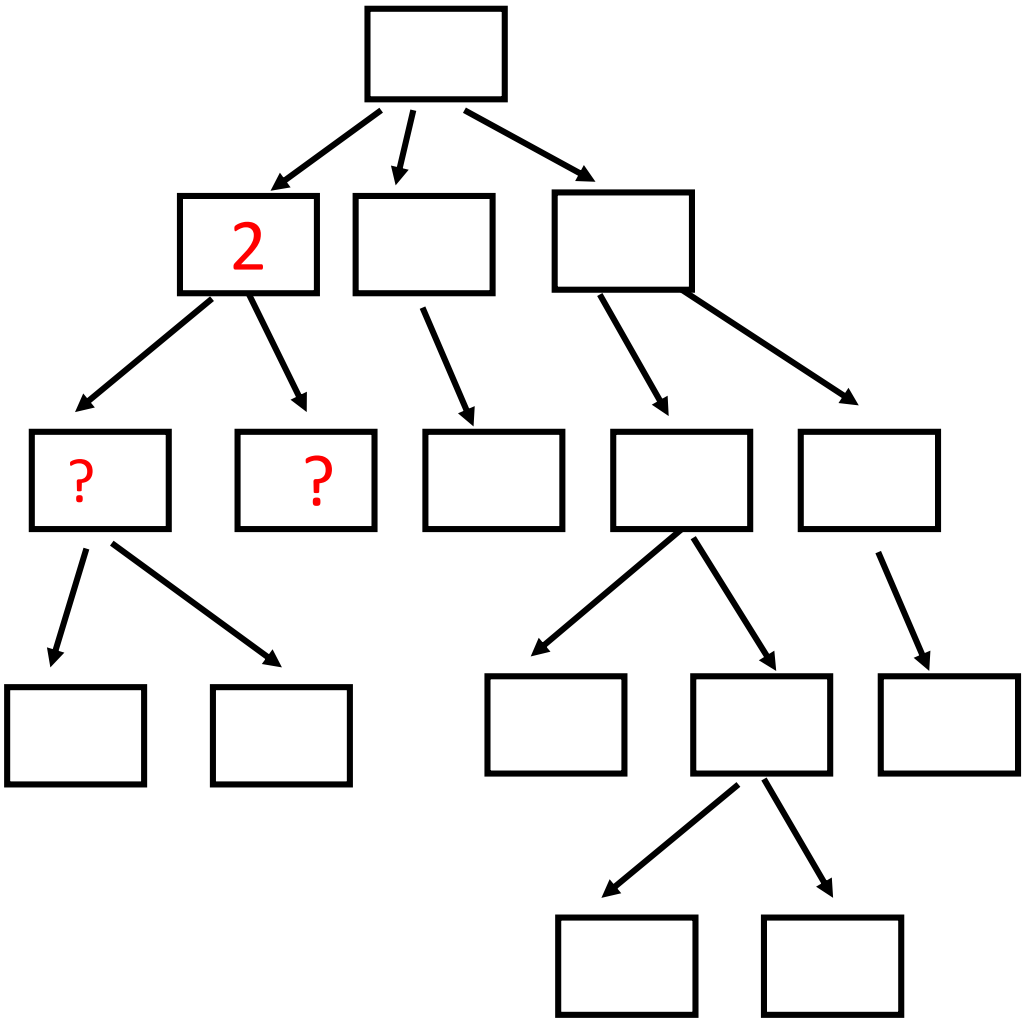
```
depth( v ){
    if ( v.parent == null)     //root
        return   0
    else
        return   1 + depth( v.parent )
}
```



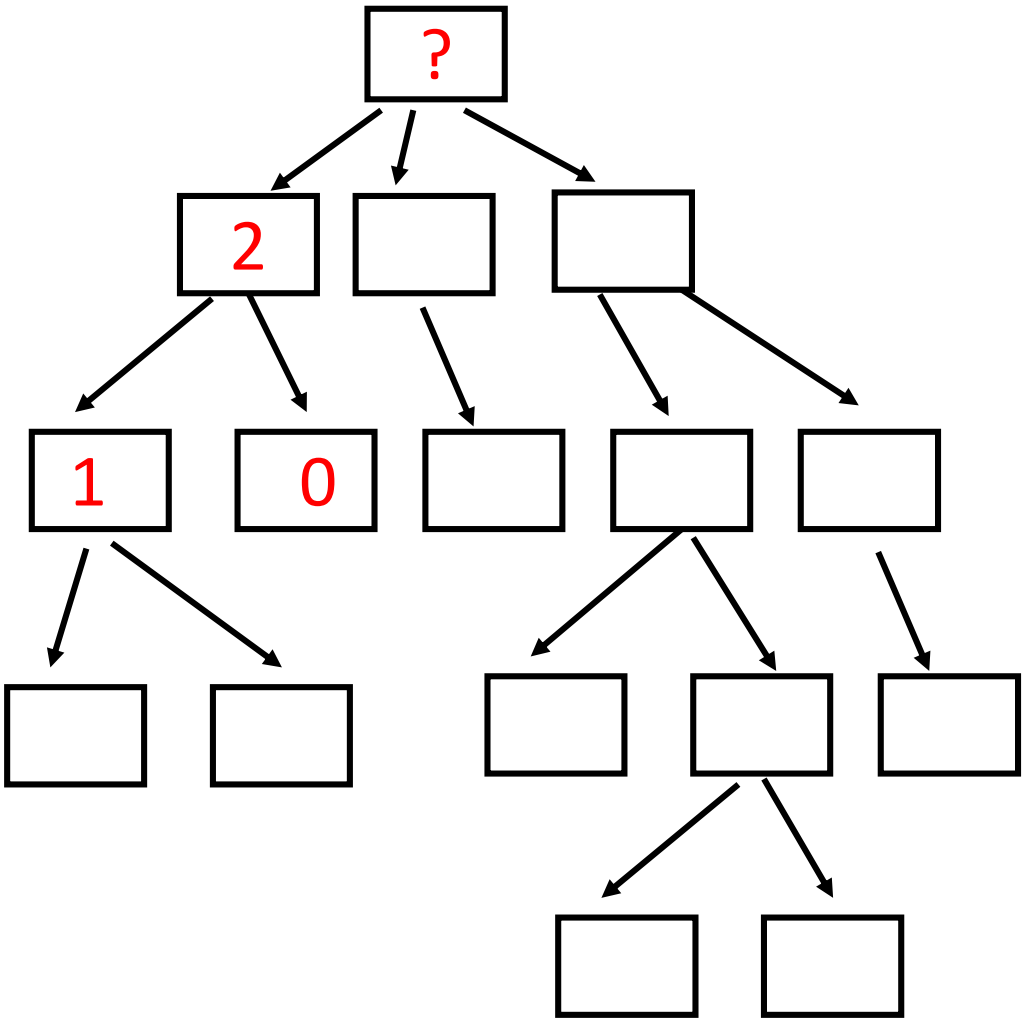
The height of a node is the maximum length of a path from that node to a leaf.



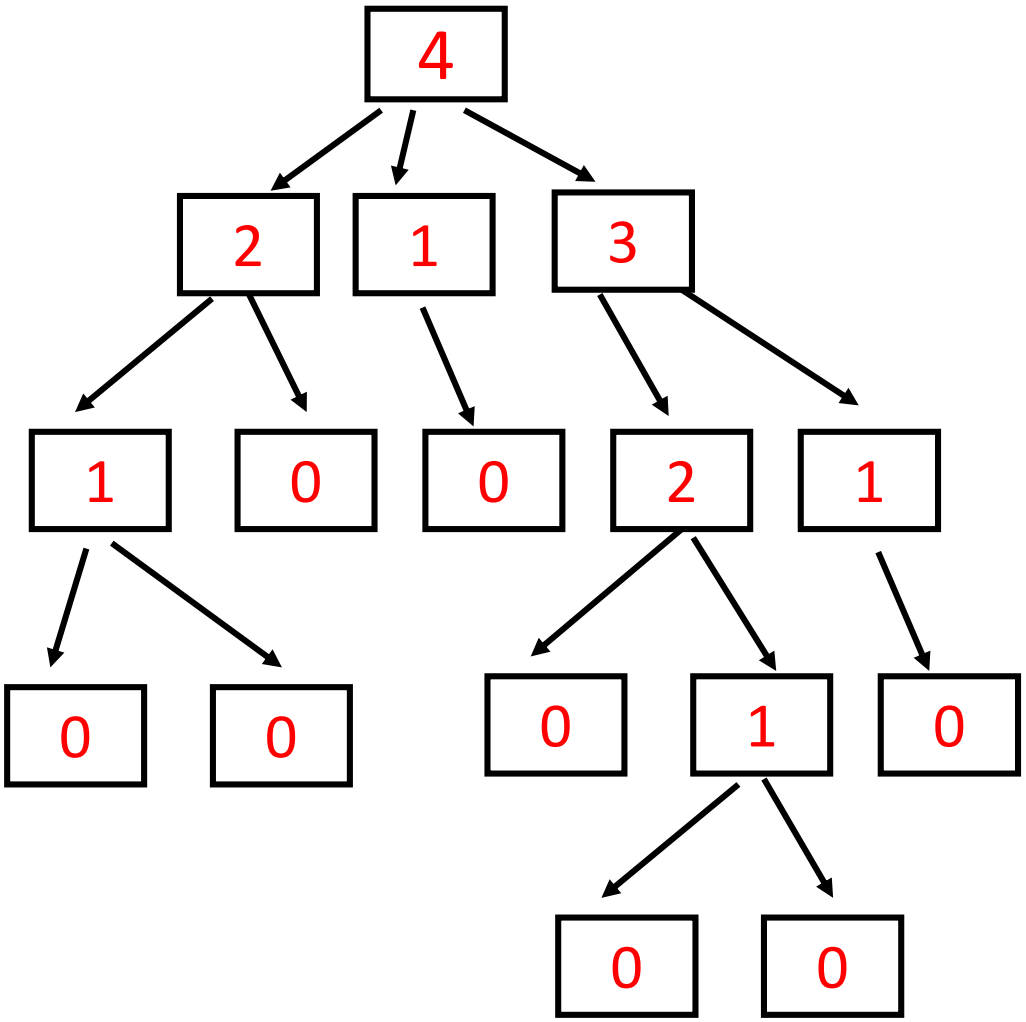
The height of a node is the maximum length of a path from that node to a leaf.

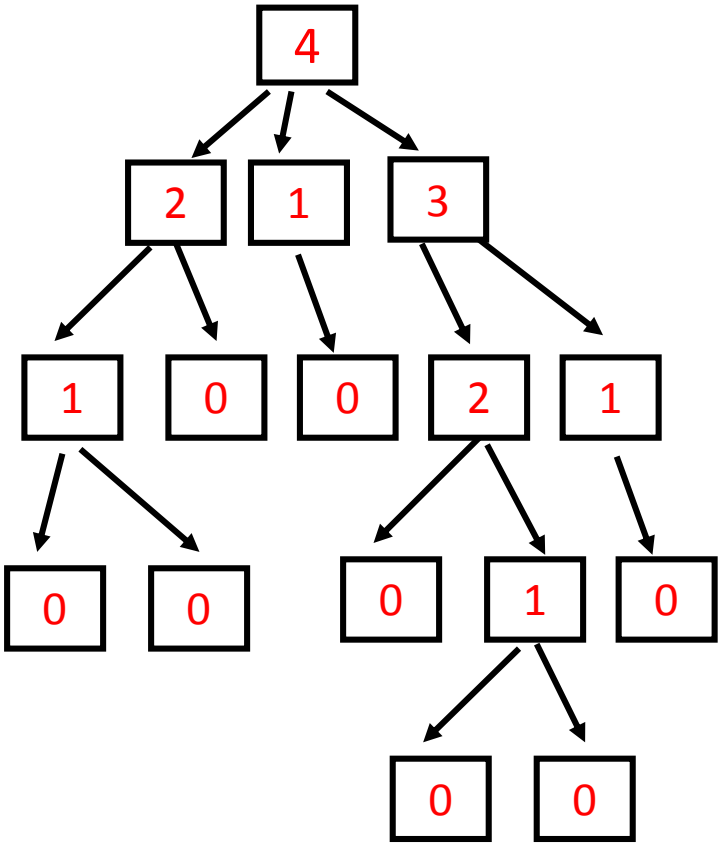


The height of a node is the maximum length of a path from that node to a leaf.



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
    }
}
```

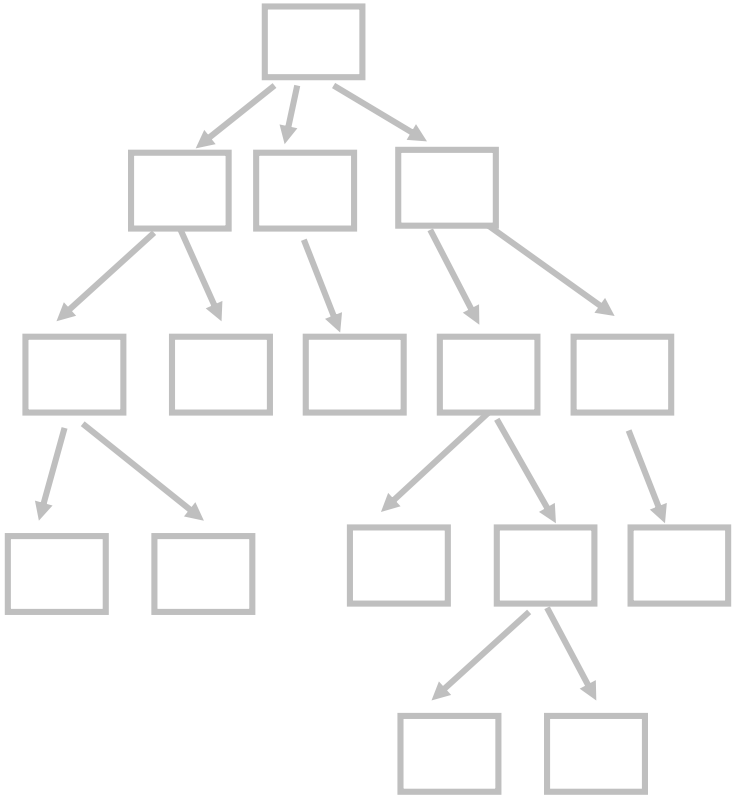
How to implement a tree ?

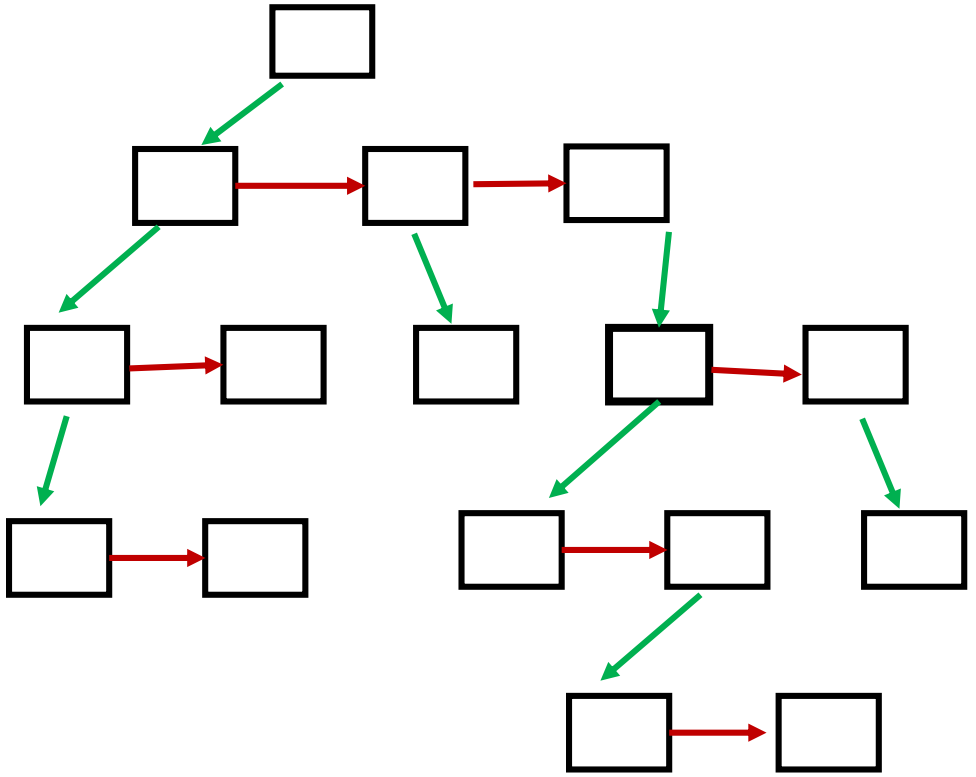
```
class  TreeNode<T>{

    T   element;



}
```

How to implement a tree ?

```
class  TreeNode<T>{

    T   element;

    ArrayList< TreeNode<T> >     children;

    TreeNode<T>                  parent;
}
```

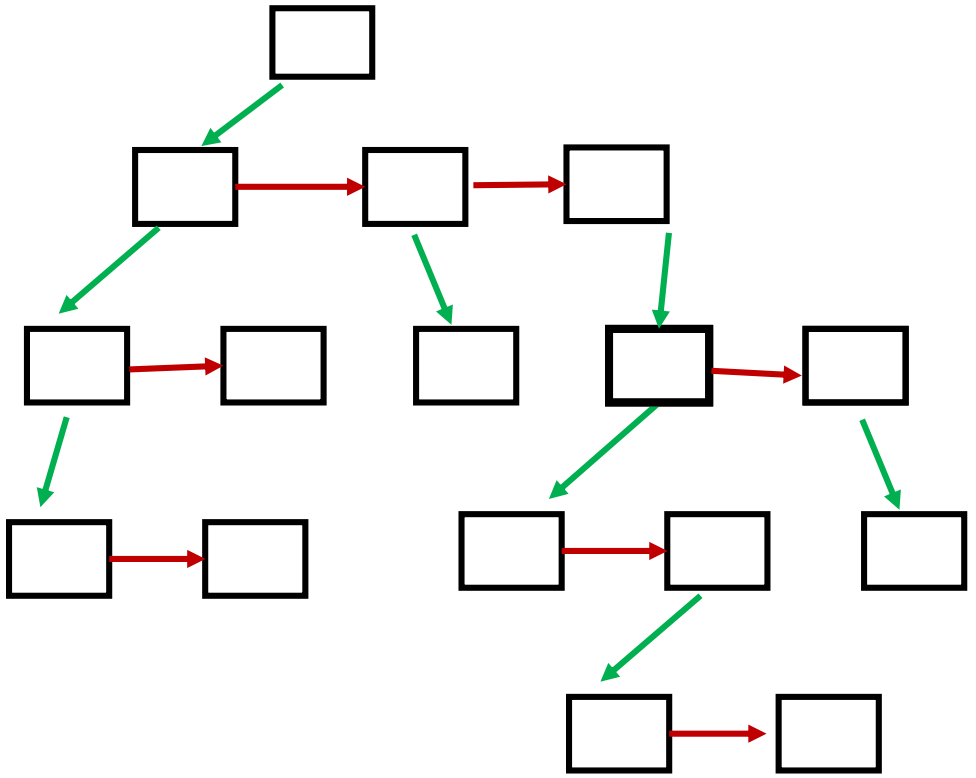
Another common implementation: ‘first child, next sibling’





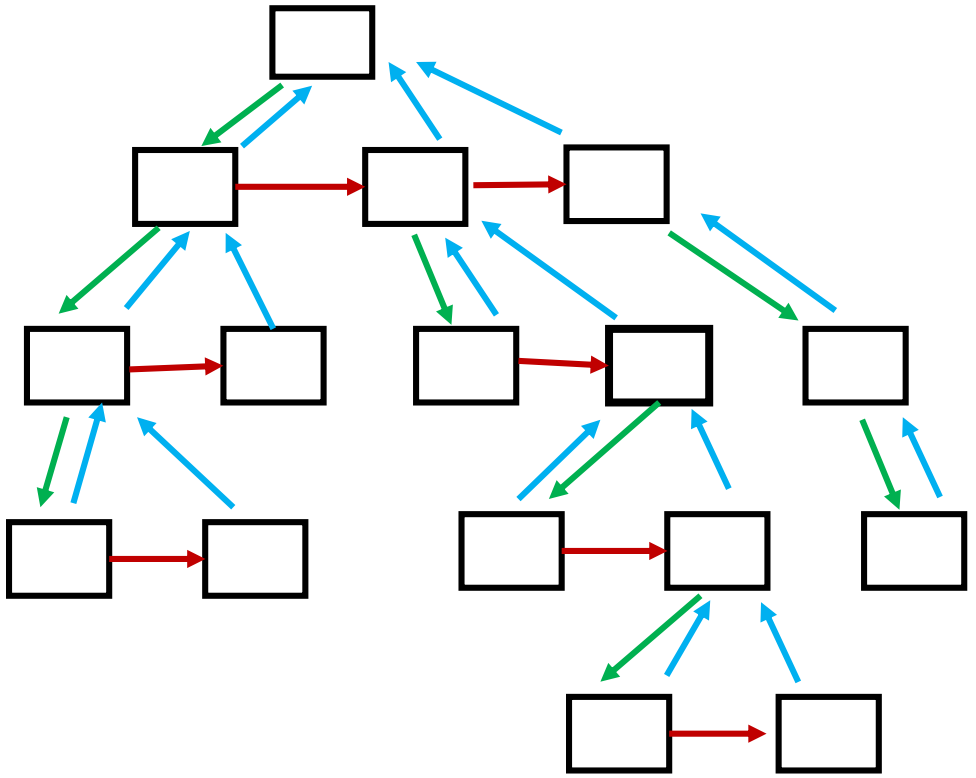
More common implementation: ‘first child, next sibling’ (similar to singly linked lists)

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

    // inner class

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



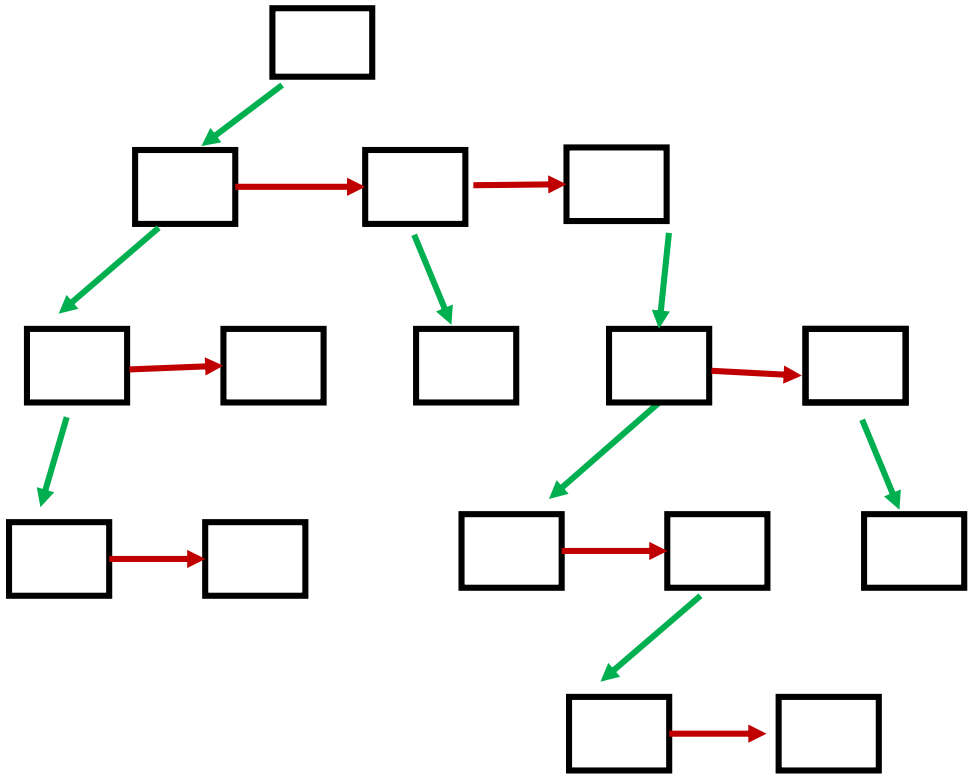
More common implementation: ‘first child, next sibling’ (similar to singly linked lists)

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

    // inner class

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



A tree of what? Each node has an element (not illustrated on right)

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

    // inner class

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



Exercise

A tree can be represented using lists, as follows:

```
tree   =      root   |    ( root listOfSubTrees )

listOfSubTrees  =  tree | tree listOfSubTrees
```

Exercise

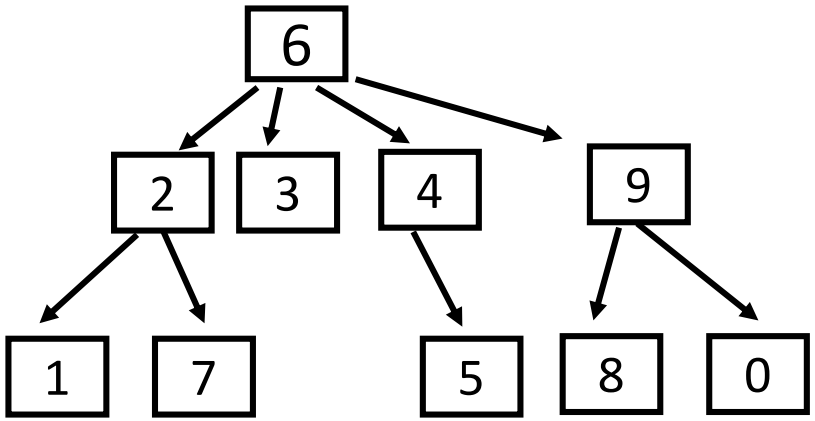
A tree can be represented using lists, as follows:

```
tree   =     root   |    ( root listOfSubTrees )

listOfSubTrees  =  tree | tree listOfSubTrees
```

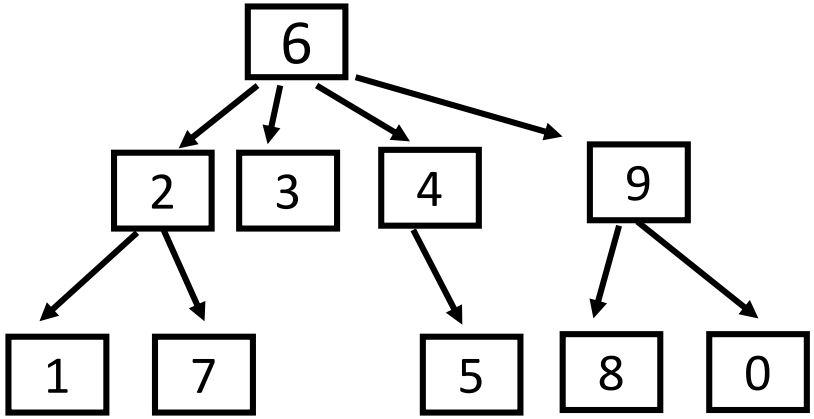
Draw the tree that corresponds to the following list, where the root elements are single digits.

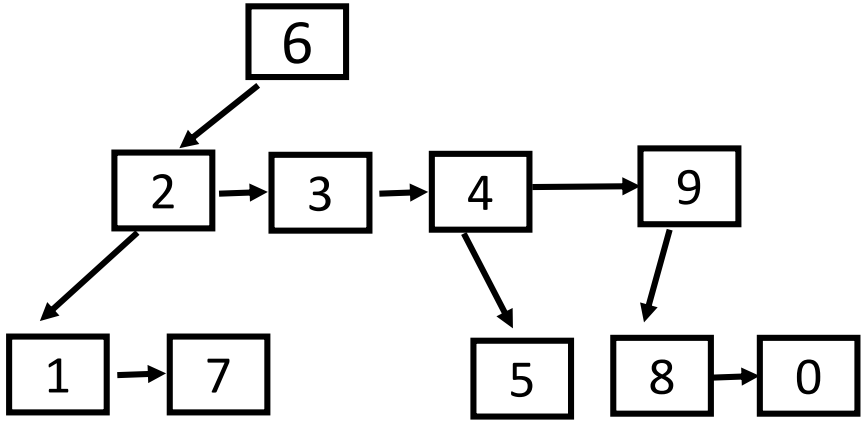
```
( 6 ( 2 1 7 ) 3 ( 4 5 ) ( 9 8 0 ) )
```

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



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





ASIDE: Non-rooted trees

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

You will see examples in COMP 251.

Note the two trees below are the same.