COMP 250

Lecture 22

(rooted) trees

Oct. 31, 2018

Tues.
Nov. 6

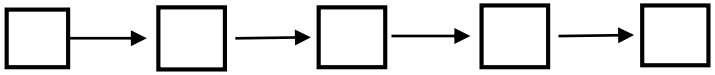
McGill Graduate & Professional Schools Fair (SUS)

Description: The Graduate School Fairs are a great way to connect with a large number of schools right here on campus! Meet representatives from programs in a wide variety of disciplines, who can provide the latest on admission requirements, fellowship opportunities, and other key information. Login to myFuture to view the list of participating schools

Linear Data Structures

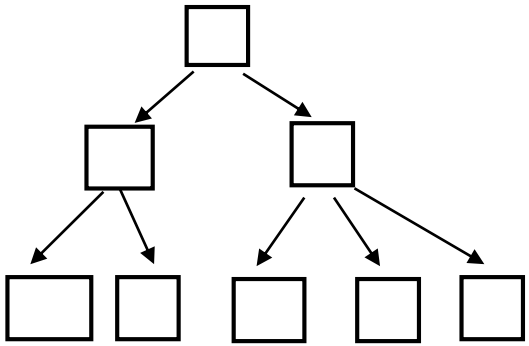
array

linked list

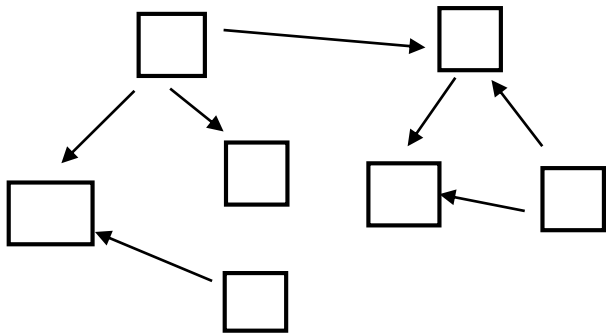


Non-Linear Data Structures

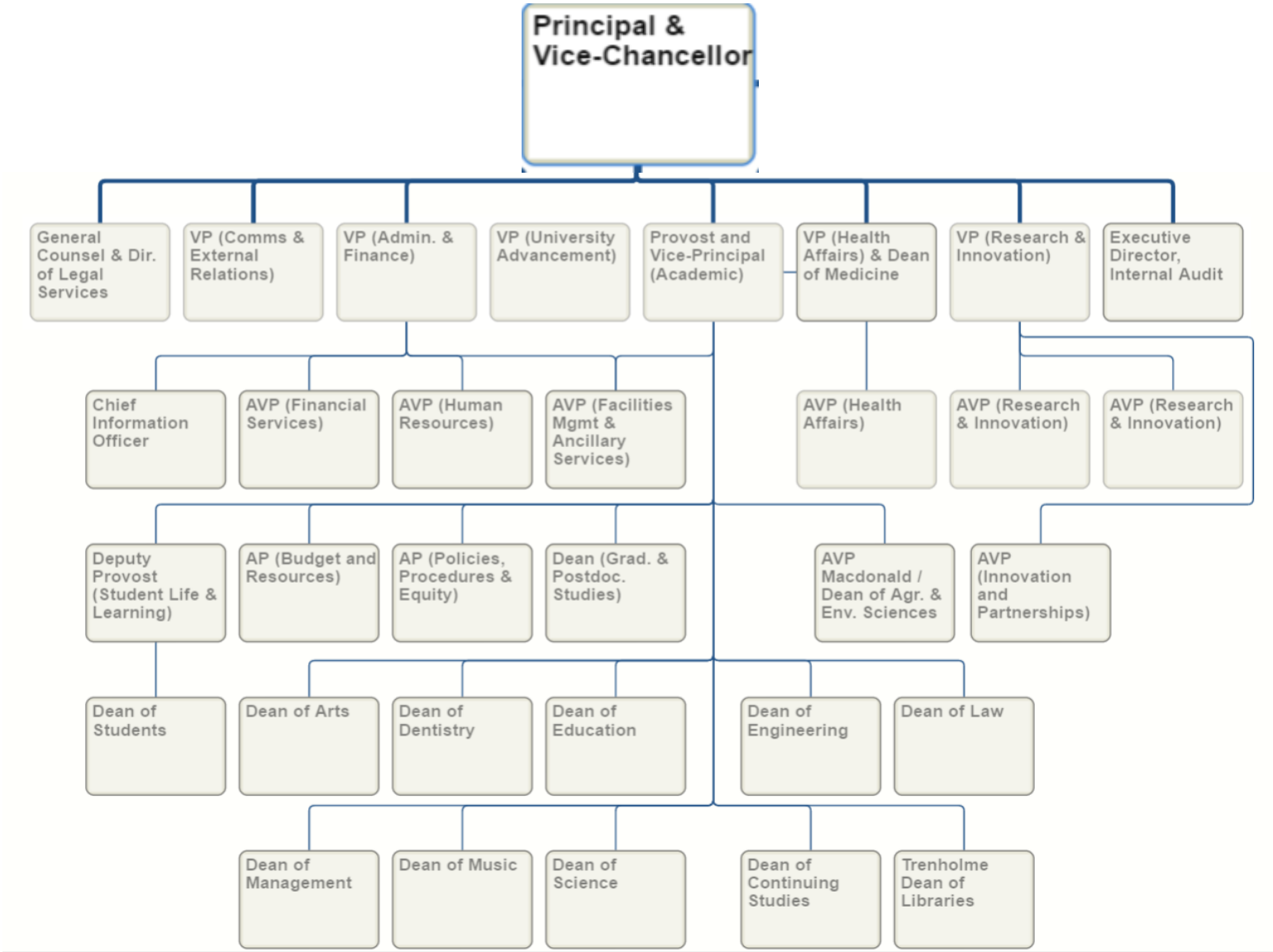
tree



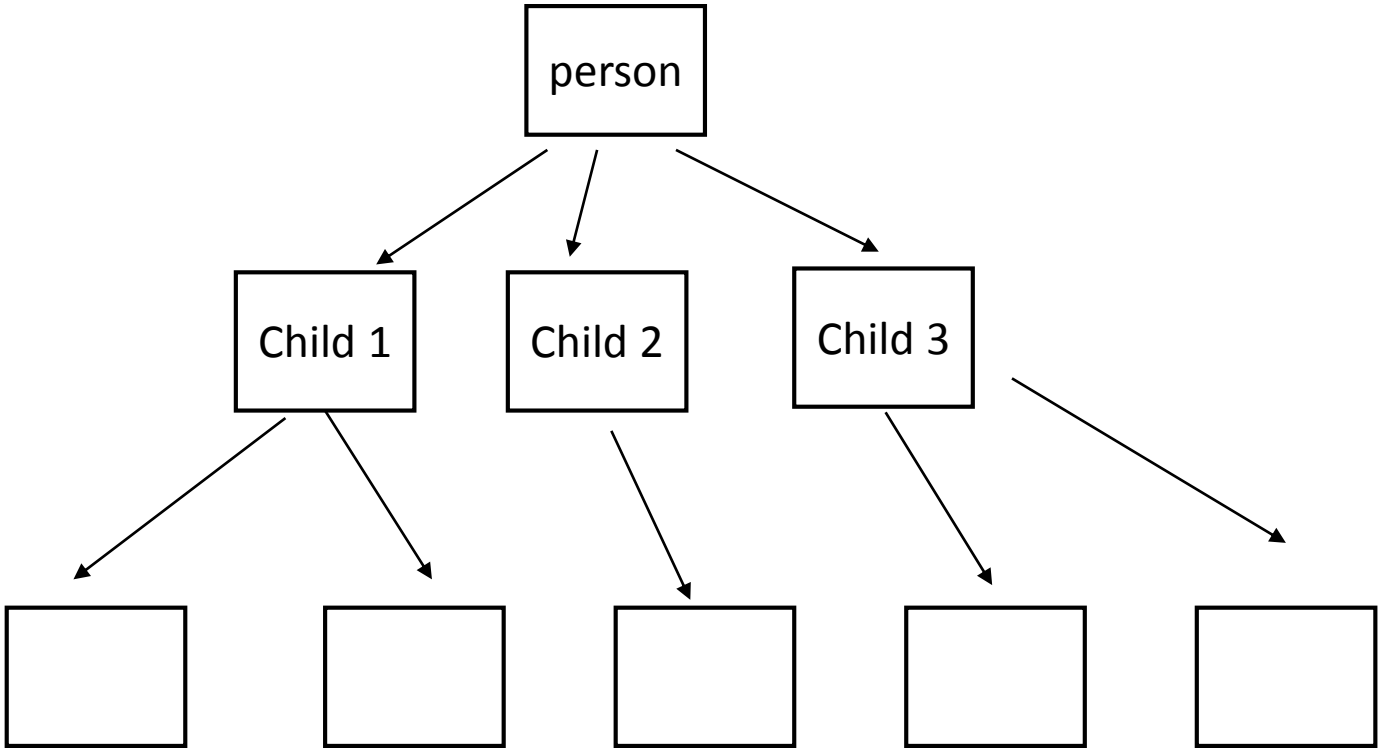
graph



Tree Example: Organization Hierarchy (McGill)

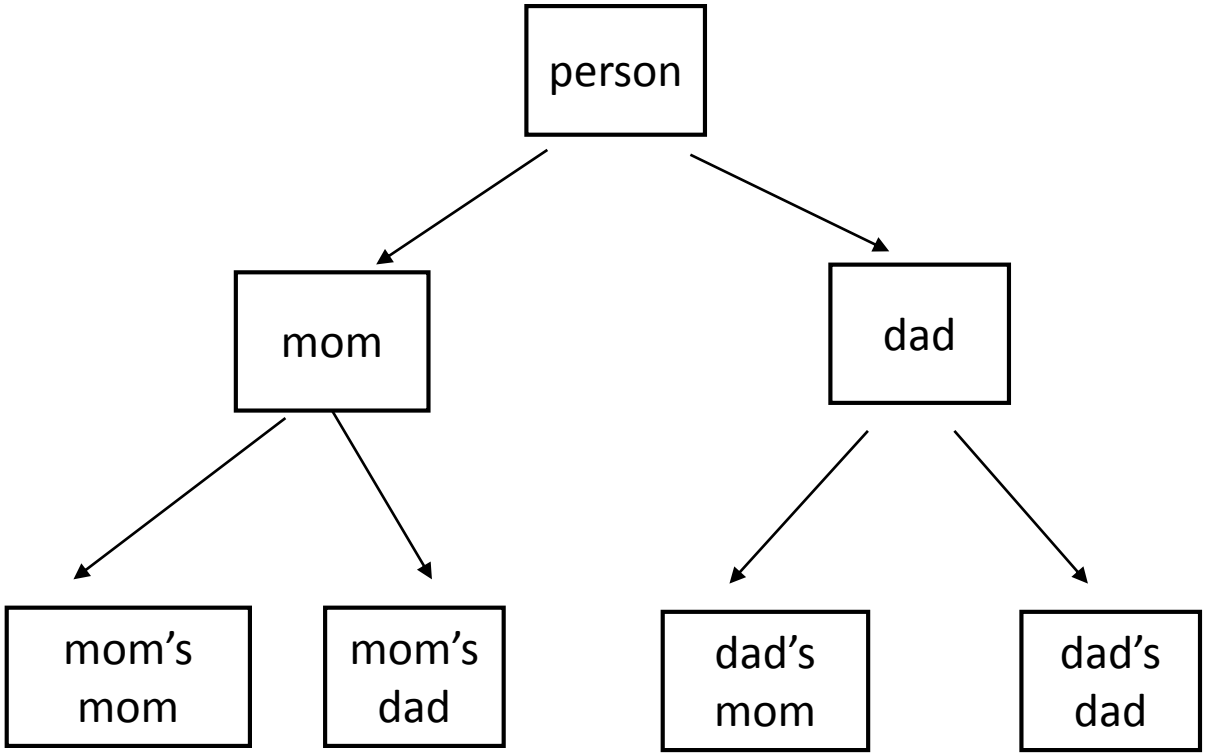


Family Tree (1)



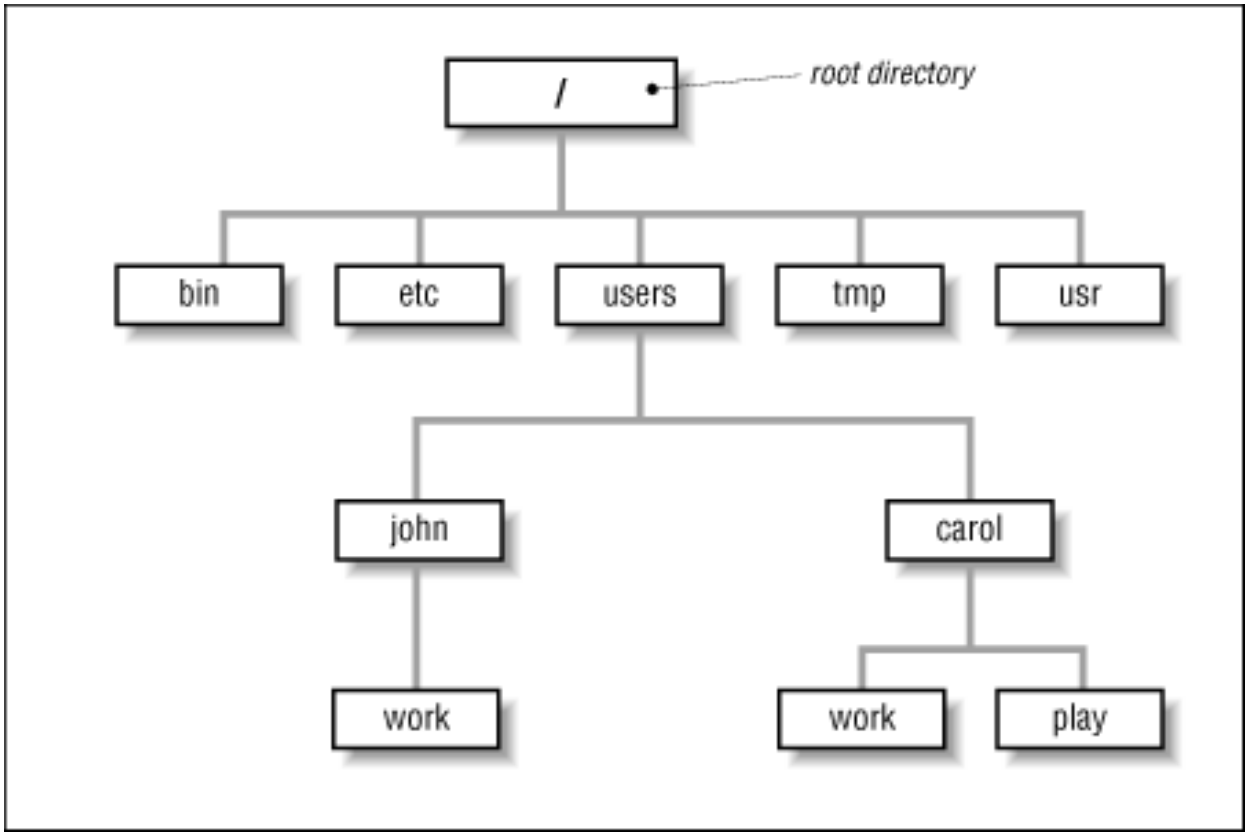
Here I ignore spouses (partner).

Family Tree (2)

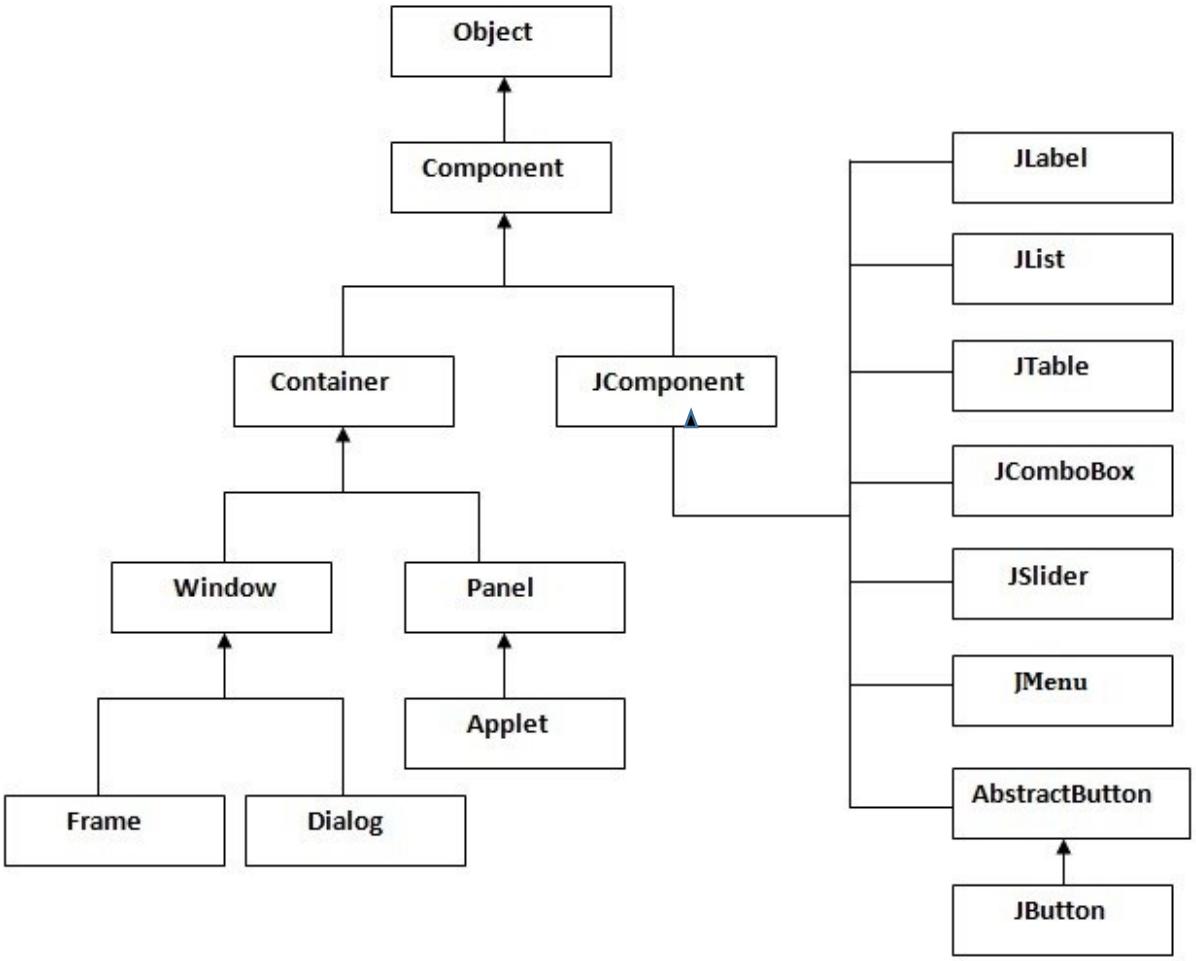


This is an example of a *binary tree*.

(UNIX) file system

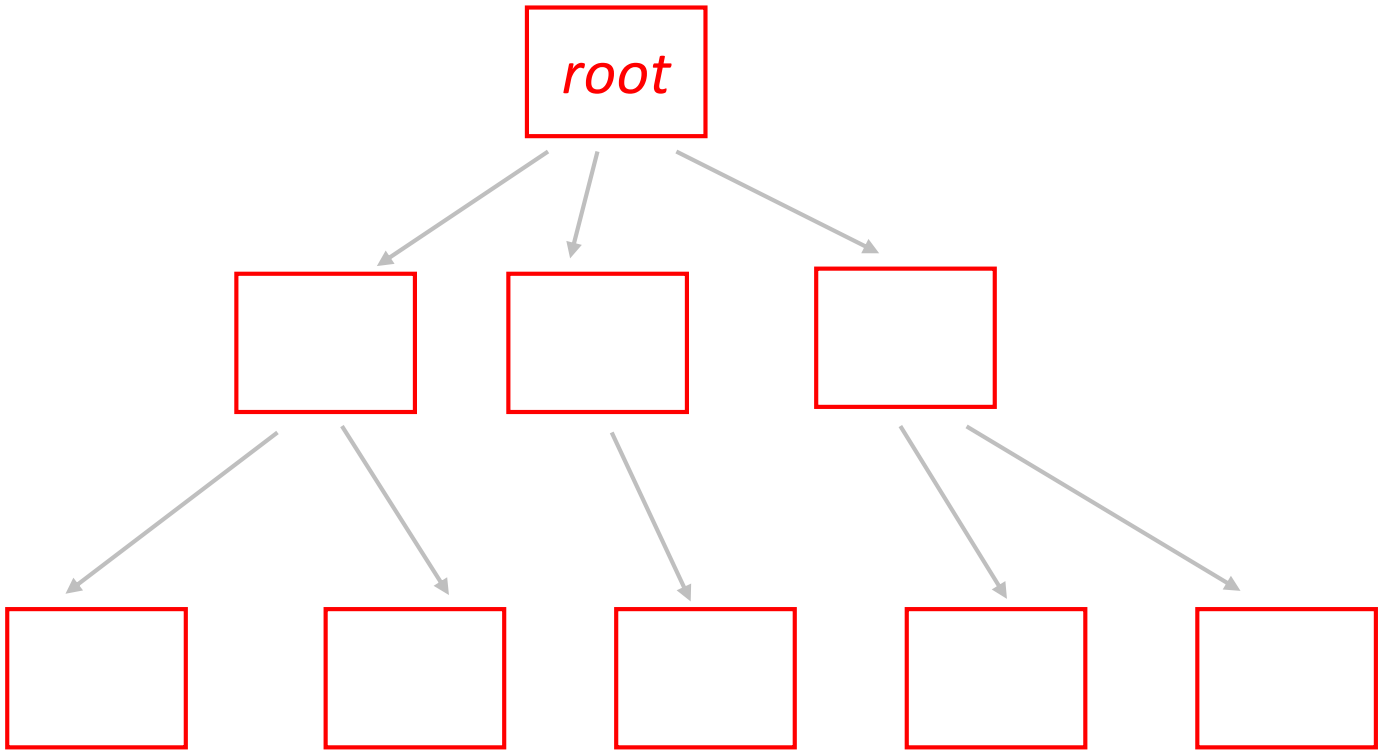


Java Classes e.g. GUI



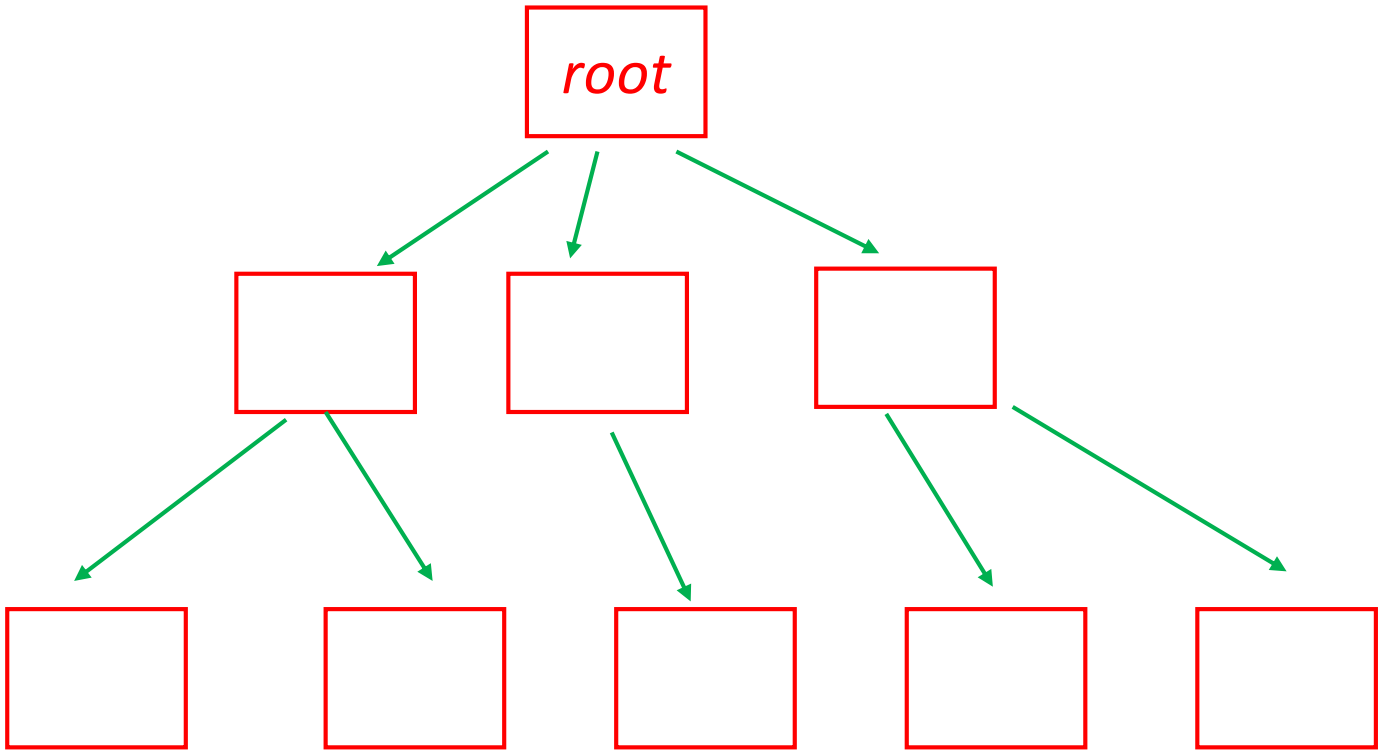
Tree Terminology

node, vertex

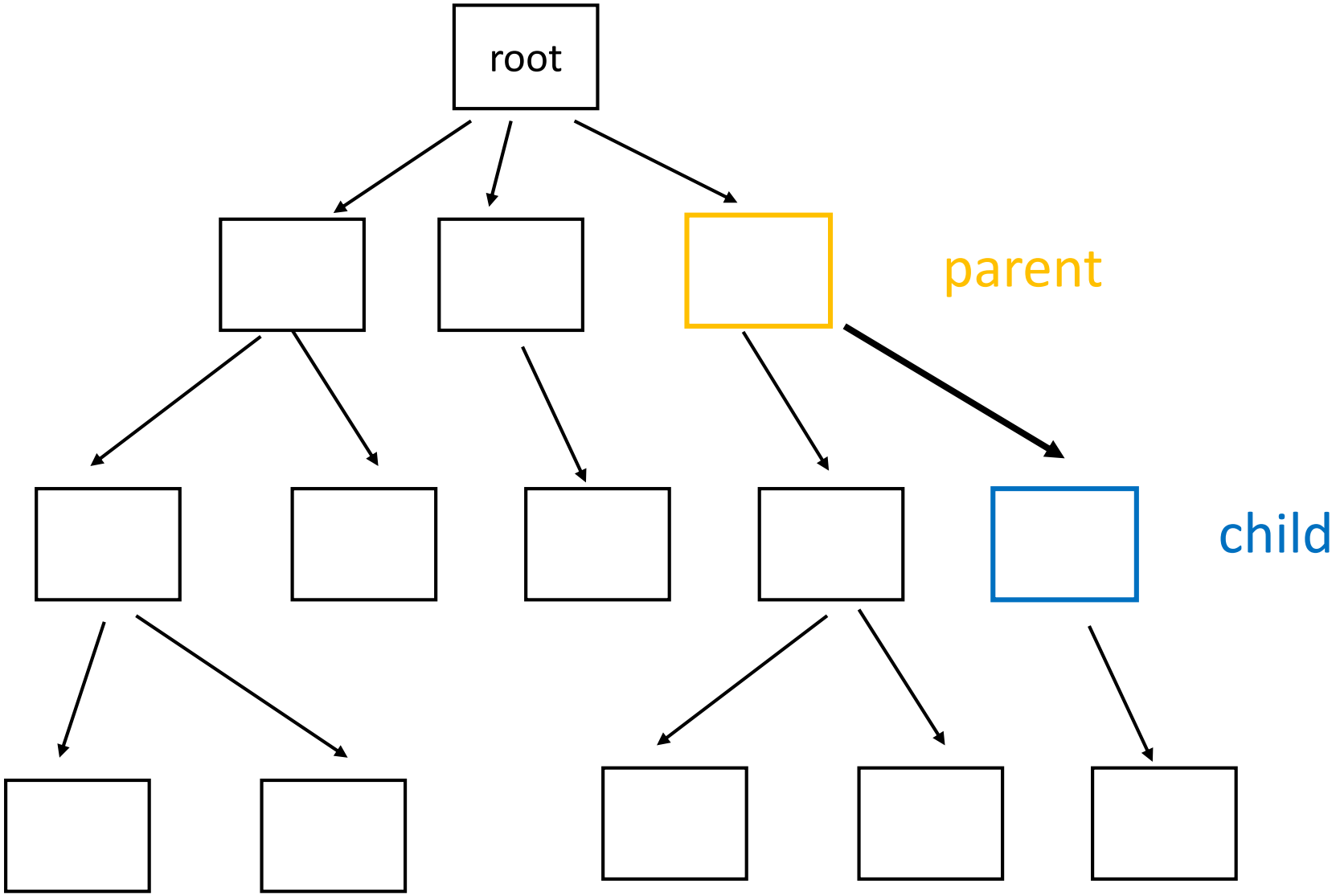


Tree Terminology

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



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. see final slide today

For some trees,

- edges are from parent to child

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

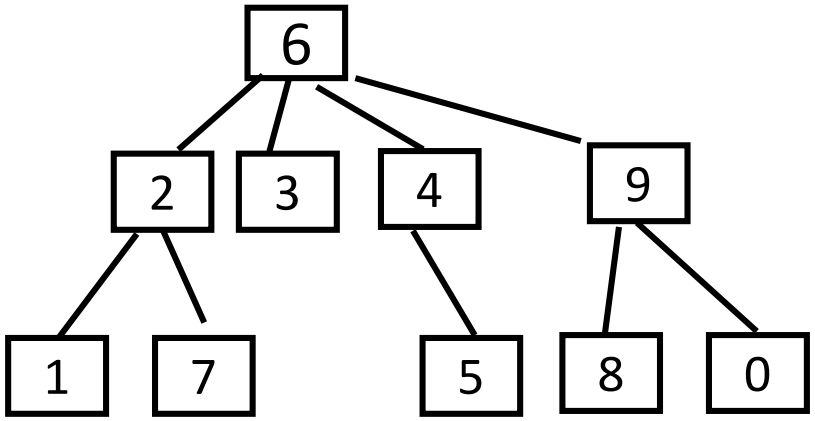
- edges are from child to parent
- edges are both from parent to child and child to parent.
- edge direction is ignored e.g. next slide

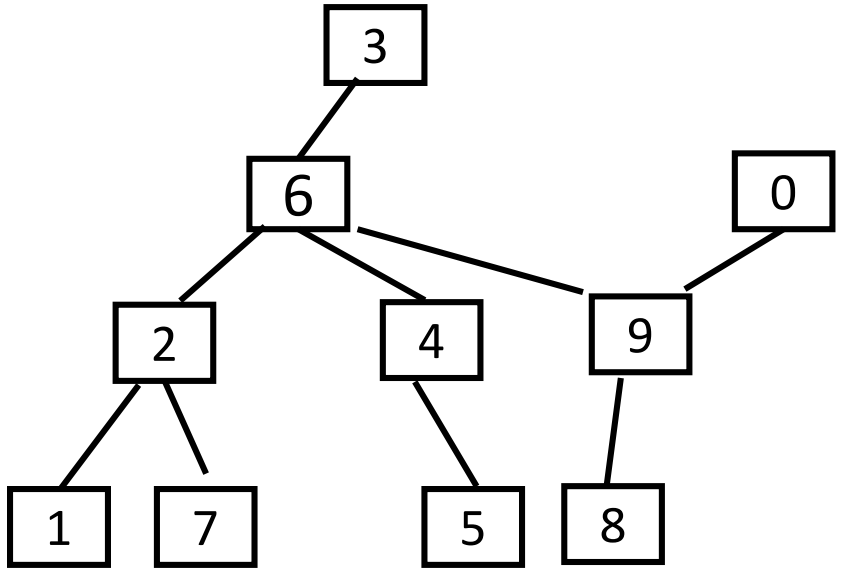
ASIDE: Non-rooted trees

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 non-rooted trees below are the same.





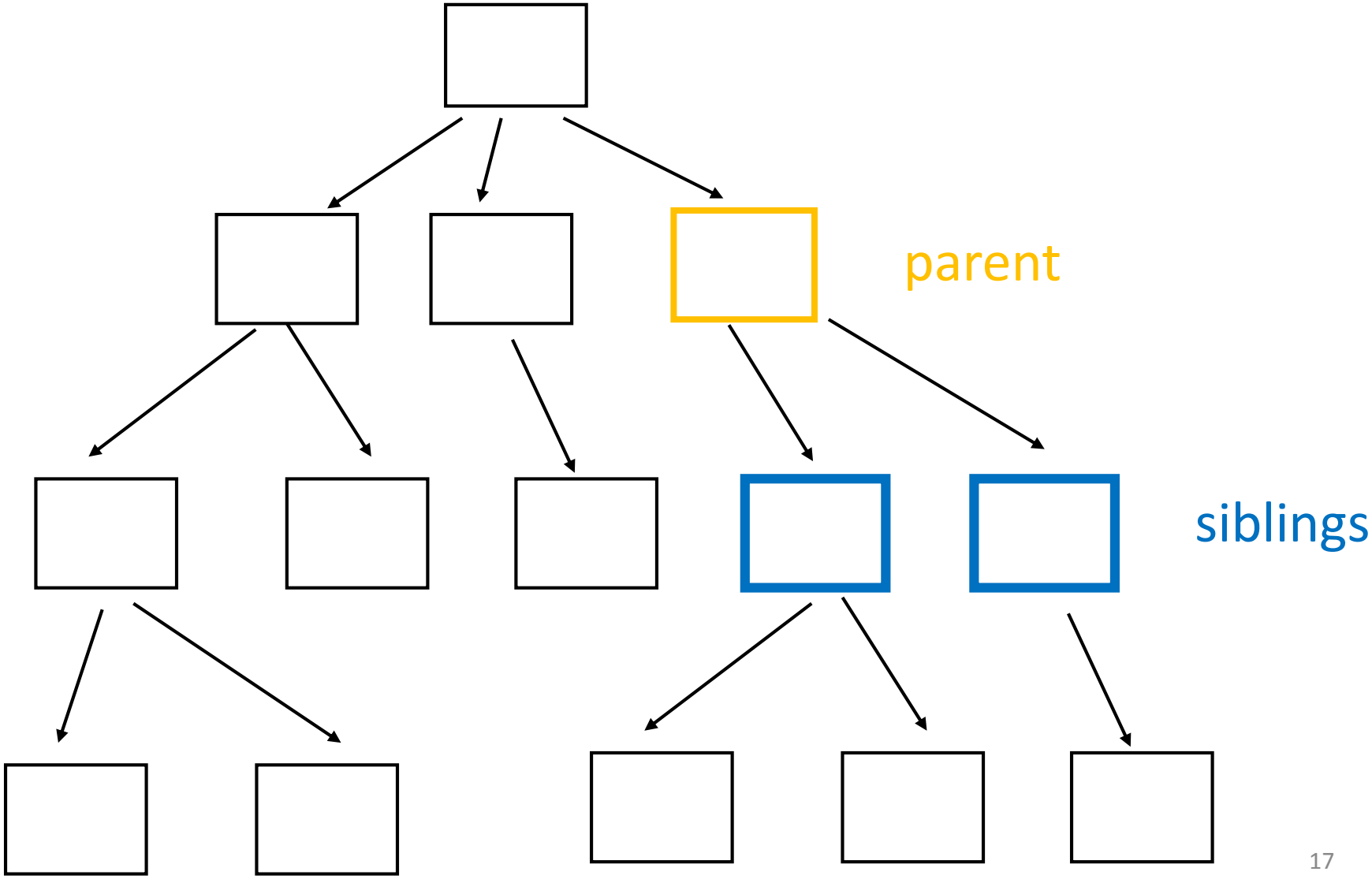
Q: If a (rooted) tree has n nodes, then how many edges does it have ?

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

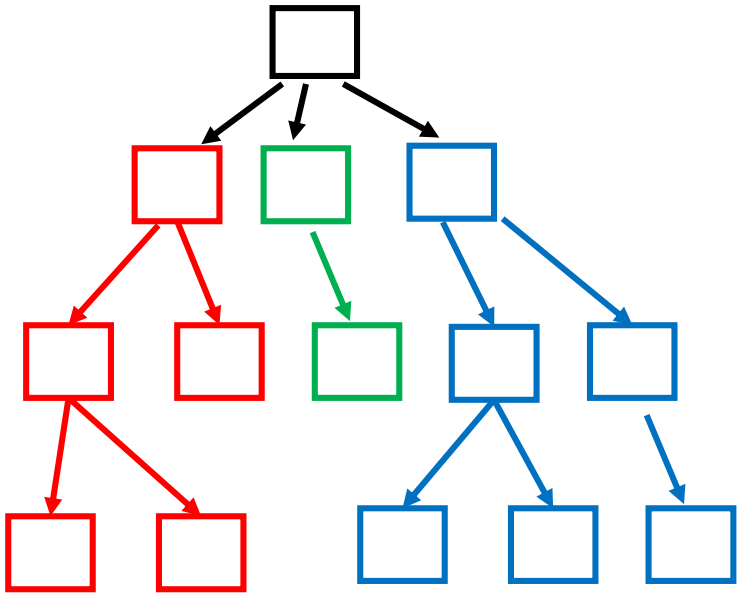
Two nodes are siblings if they have the same parent.



Recursive definition of (rooted) tree

A tree T is a finite set of $n \geq 0$ nodes such that:

?



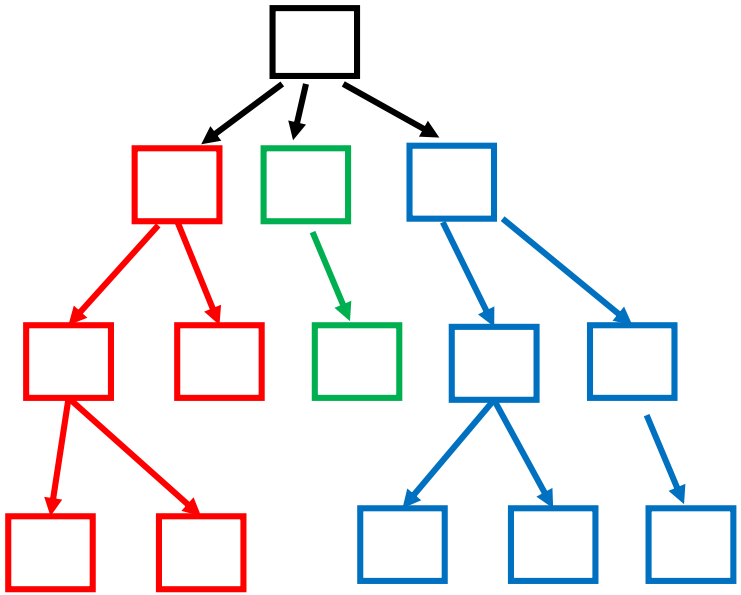
Recursive definition of (rooted) tree

A tree T is a finite set of $n \geq 0$ nodes such that:

- if $n > 0$ then one of the nodes is the root r

  (that is, it has no parent)

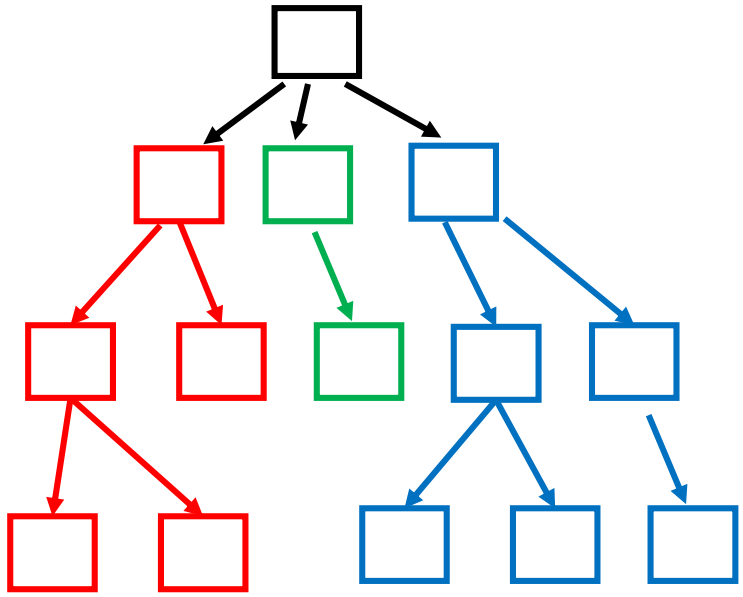
- ?



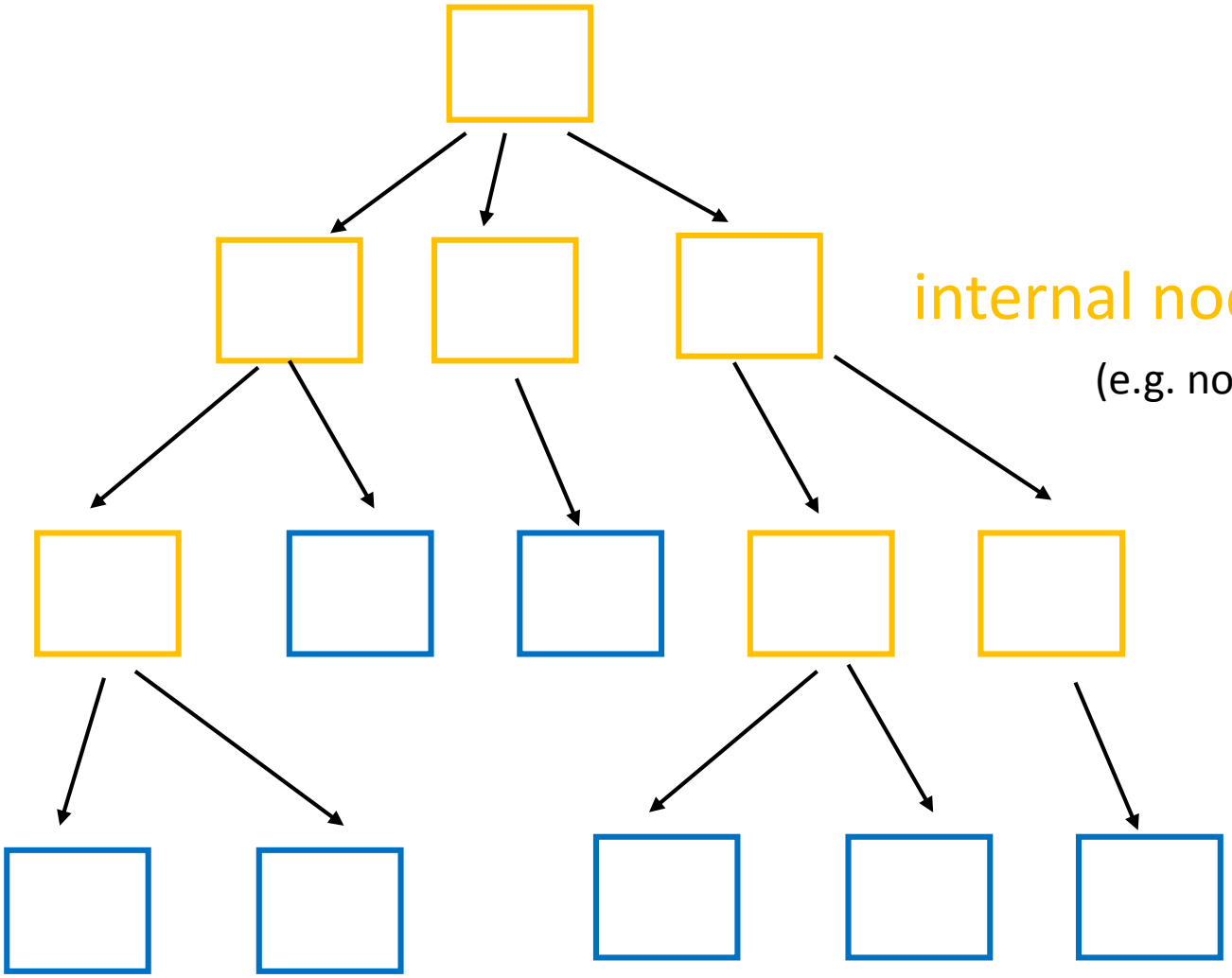
Recursive definition of rooted tree

A tree T is a finite set of $n \geq 0$ 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.



This definition assumes directed edges ("…children…") but we could change the wording so that it does not assume directed edges.

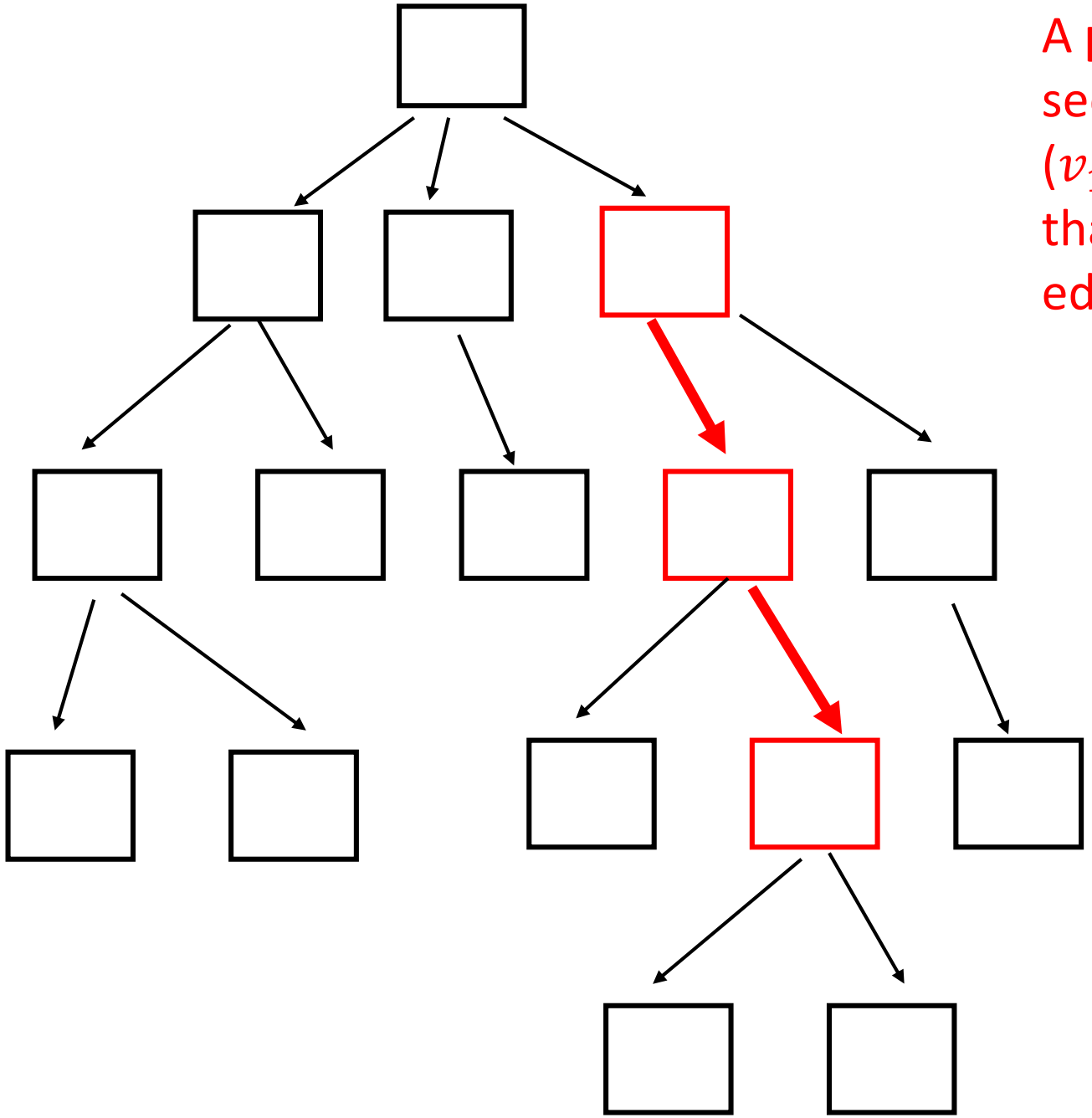


internal nodes

(e.g. non empty file directories)

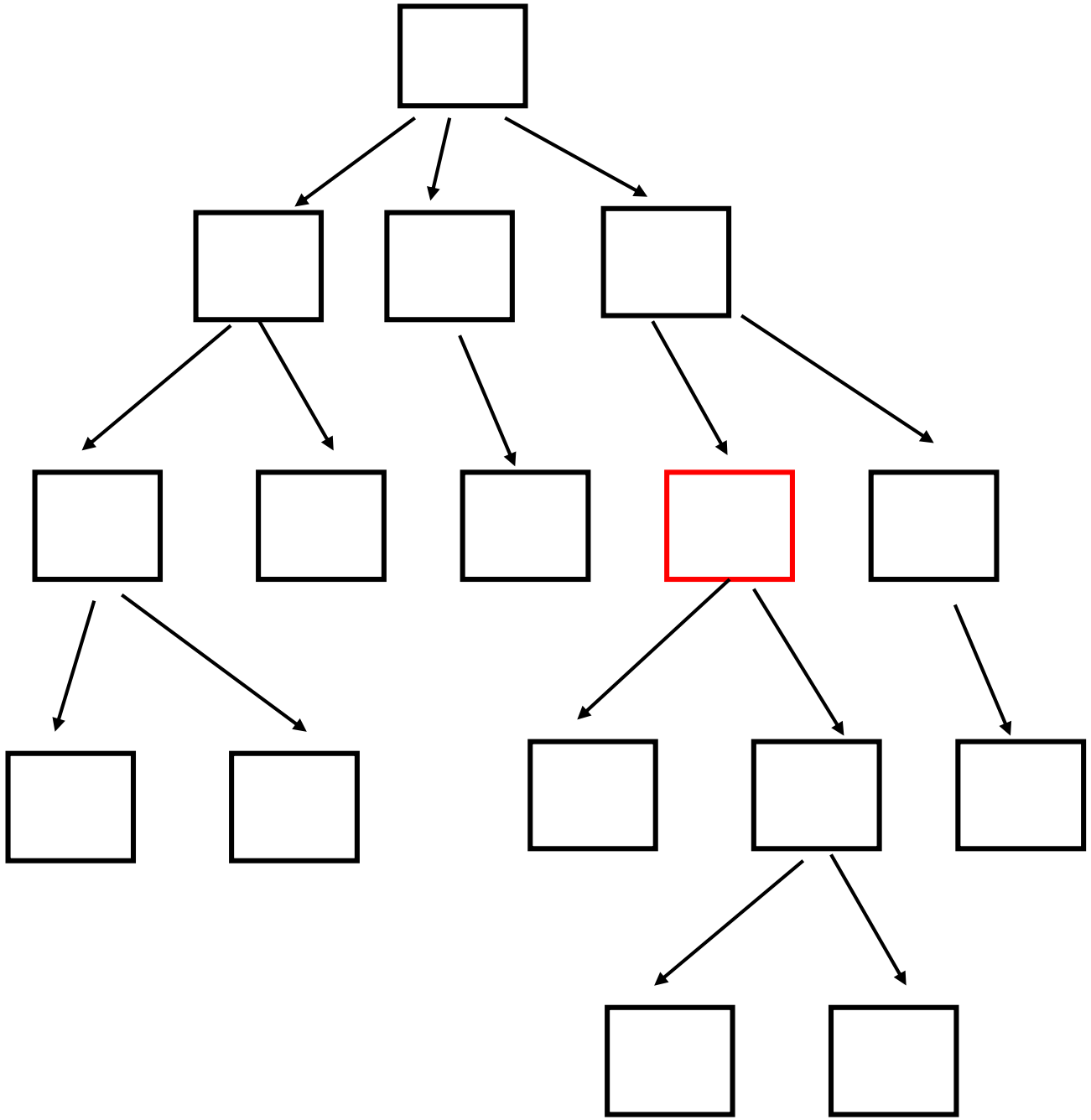
external nodes, leaves

(e.g. files or empty directories)

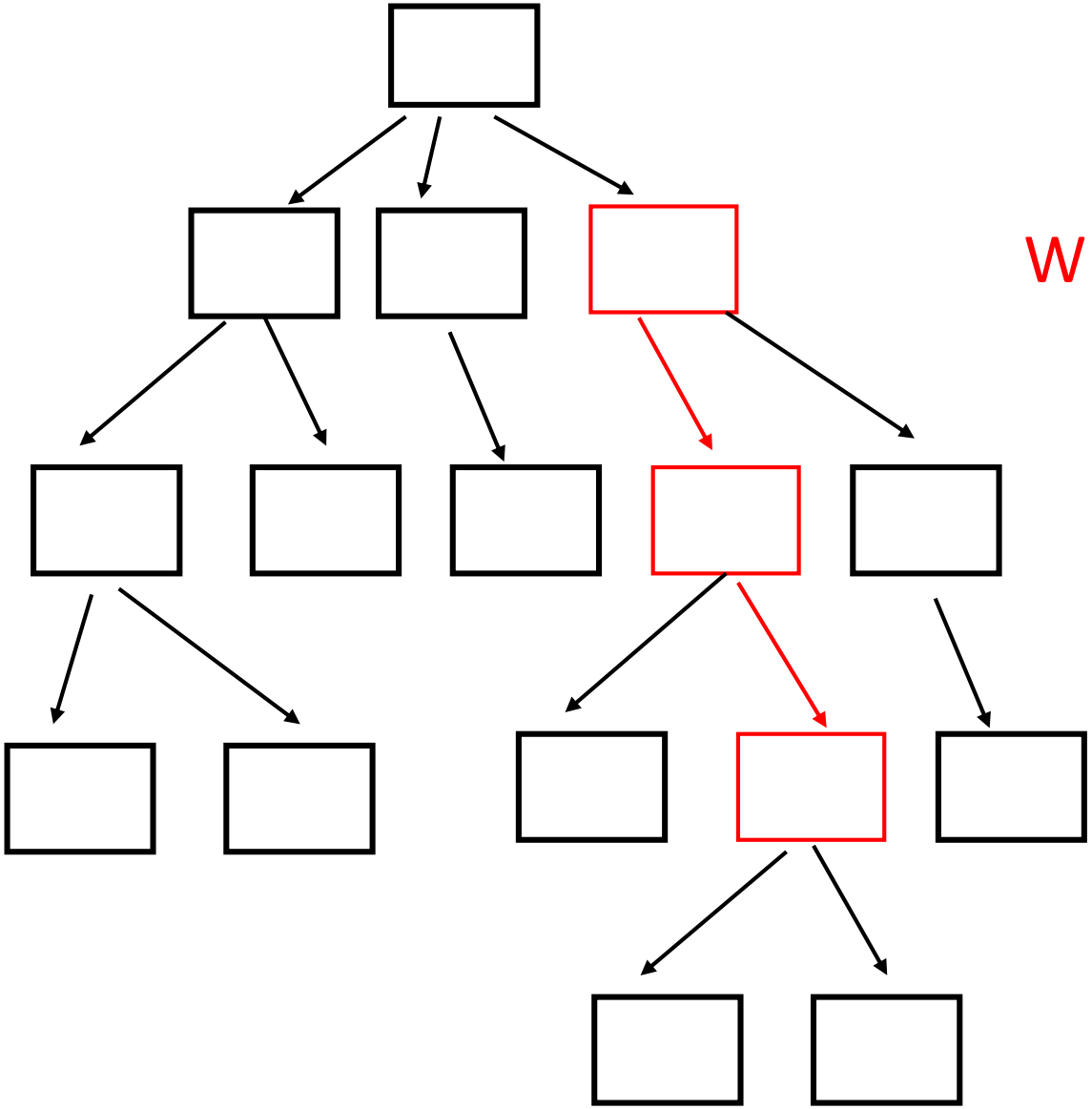


A **path** in a tree is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that (v_i, v_{i+1}) is an edge.

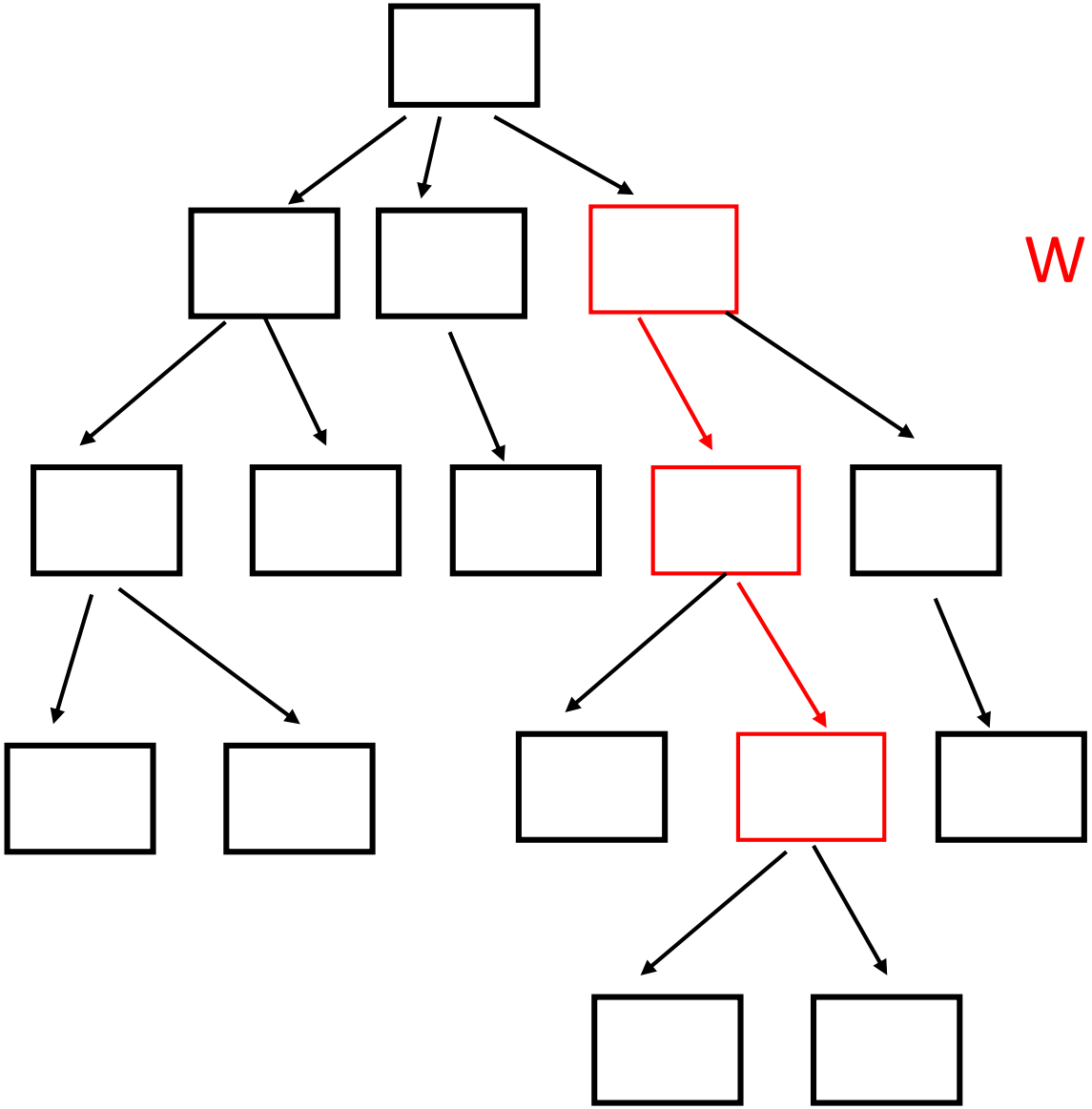
The **length** of a path is *the number of edges in the path* (number of nodes in the path minus 1)



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



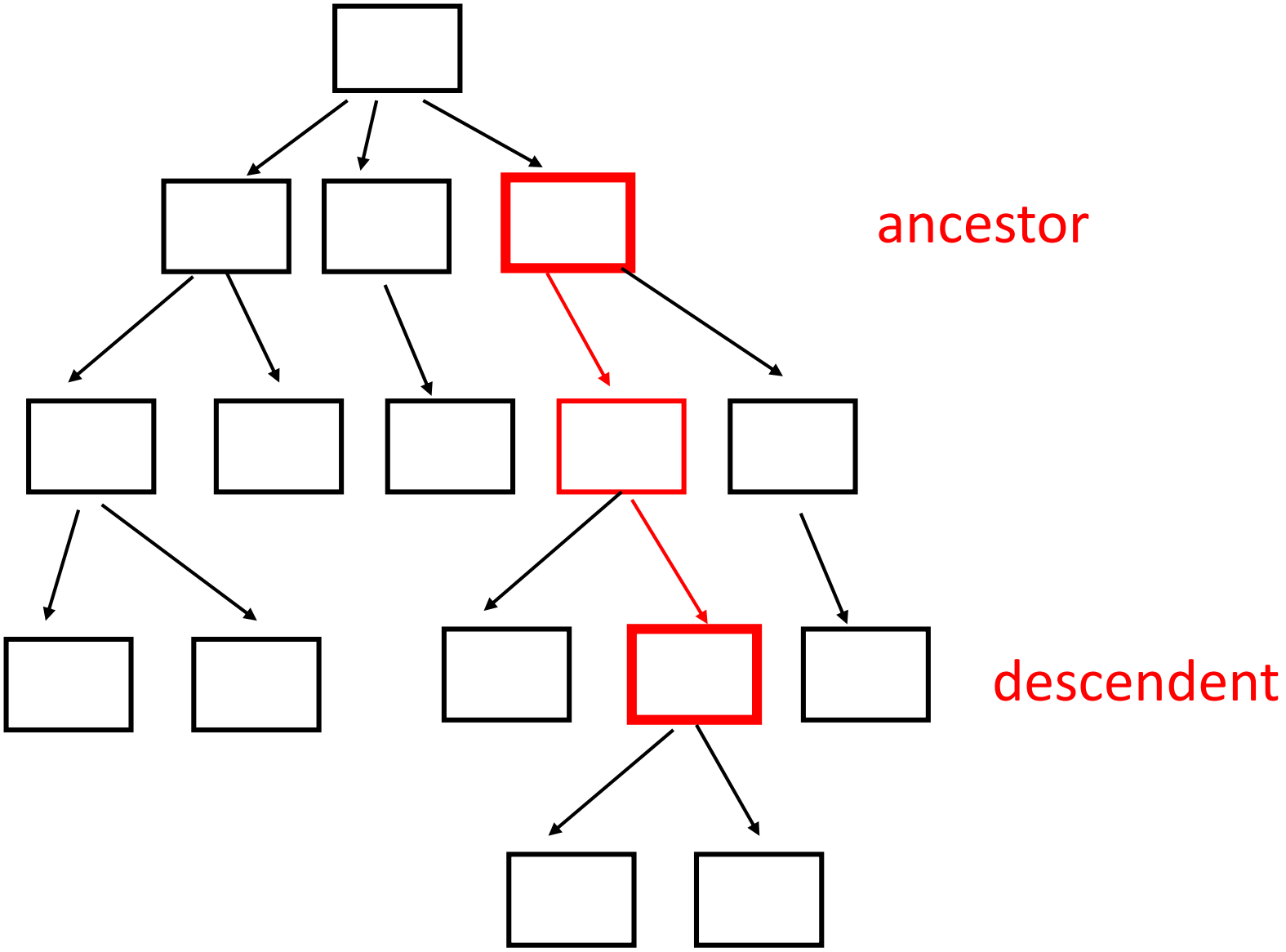
What is the path length?



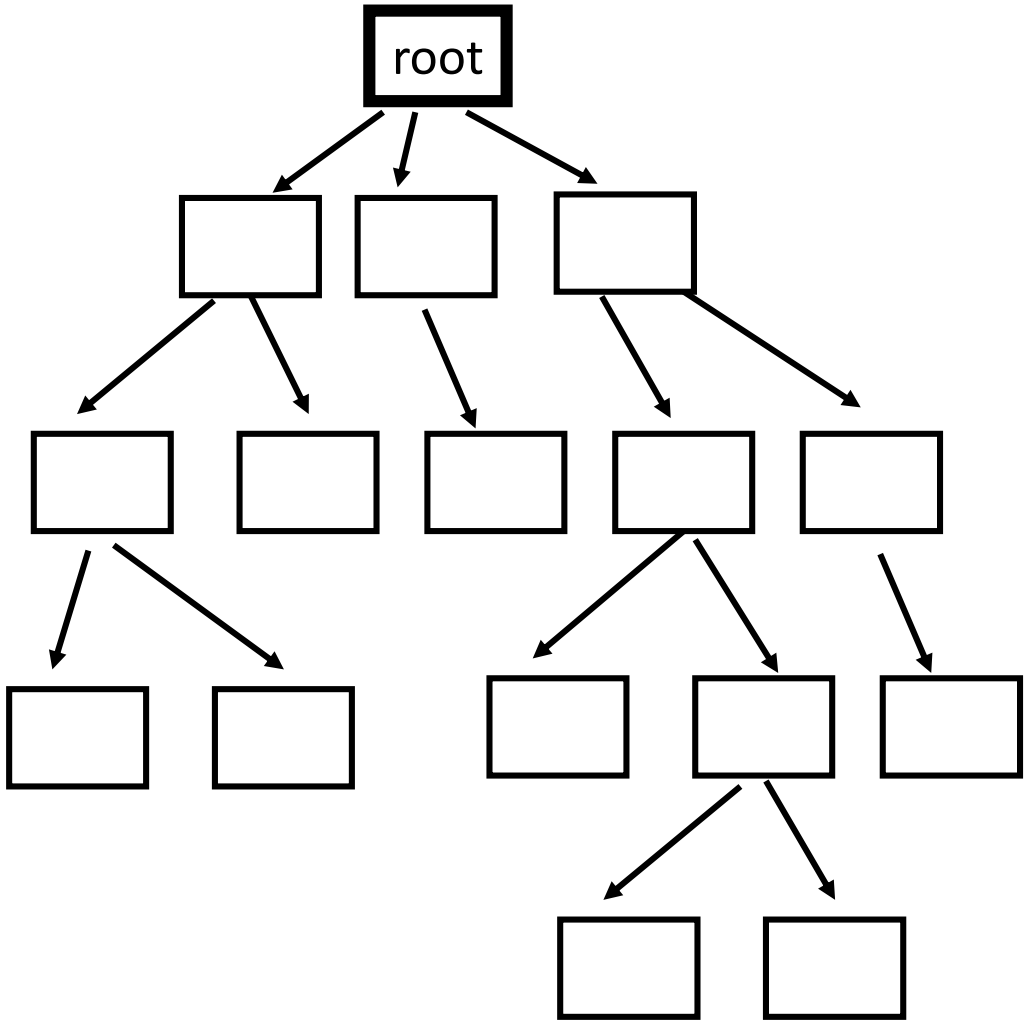
What is the path length?

Answer: 2 (edges)

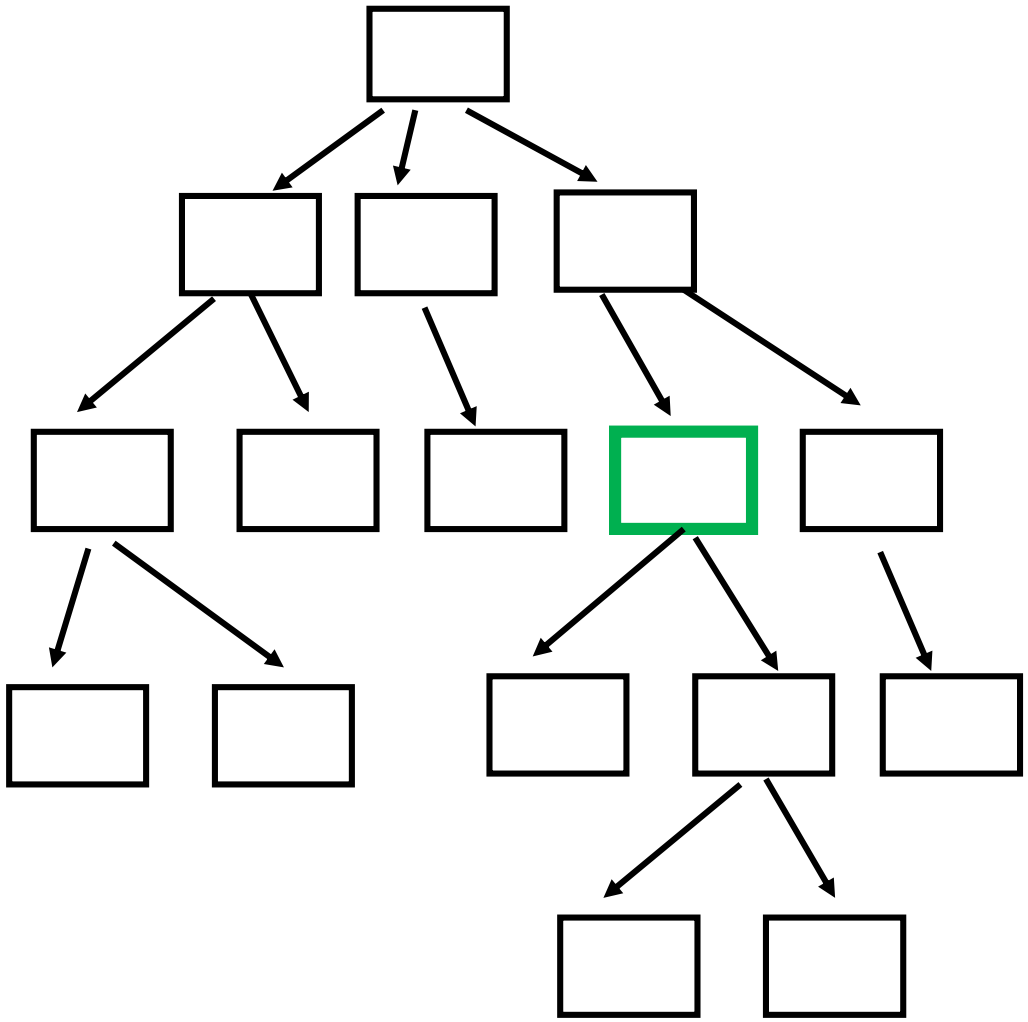
Node v is an ancestor of node w if there is a path from v to w.
Similarly, node w is a descendent of node v.



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



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



depth (level)

0

1

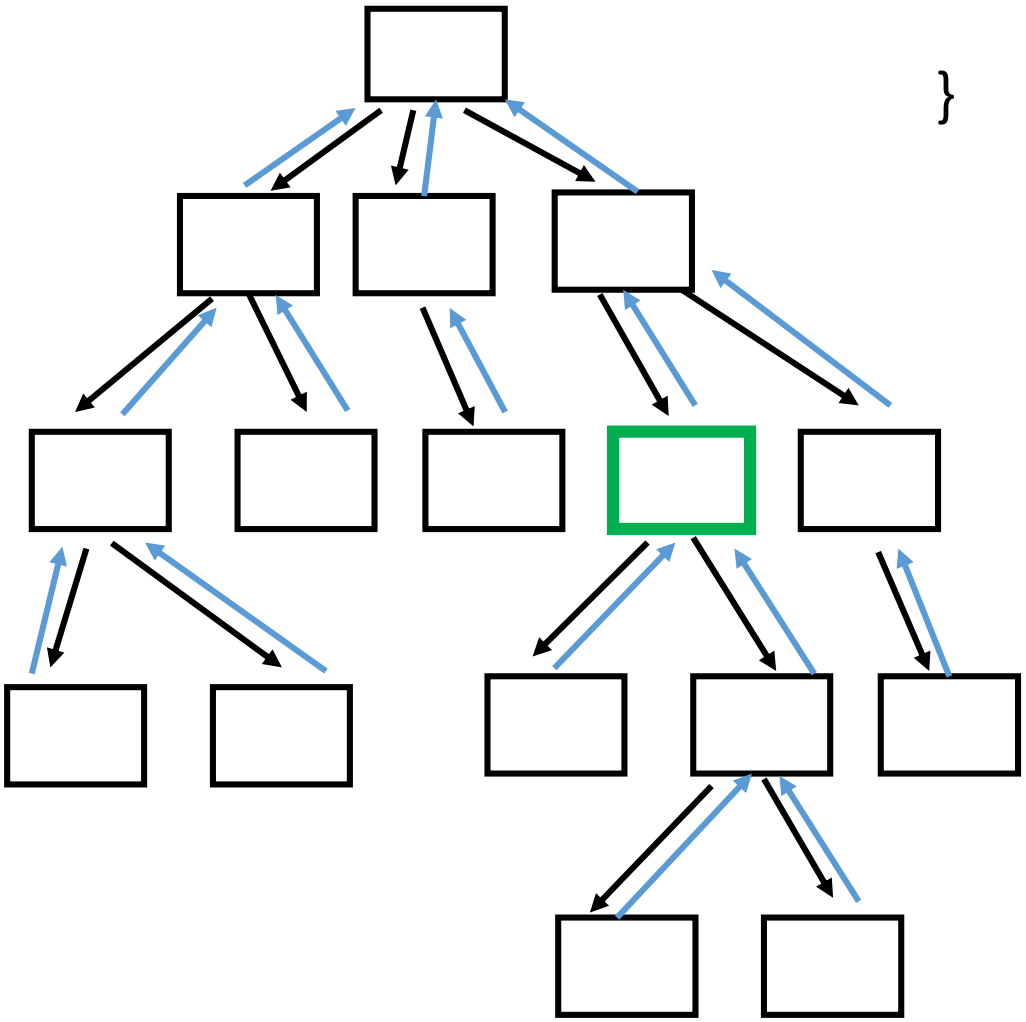
2

3

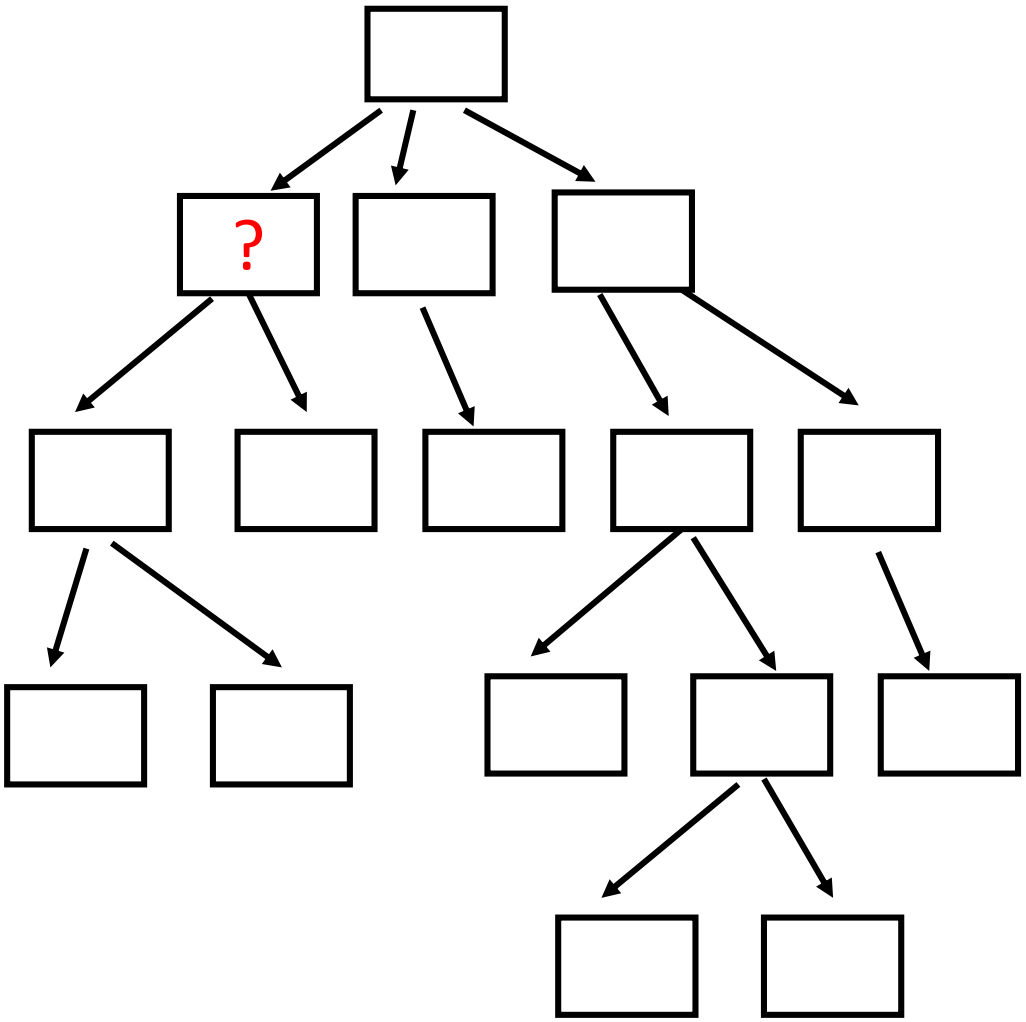
4

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

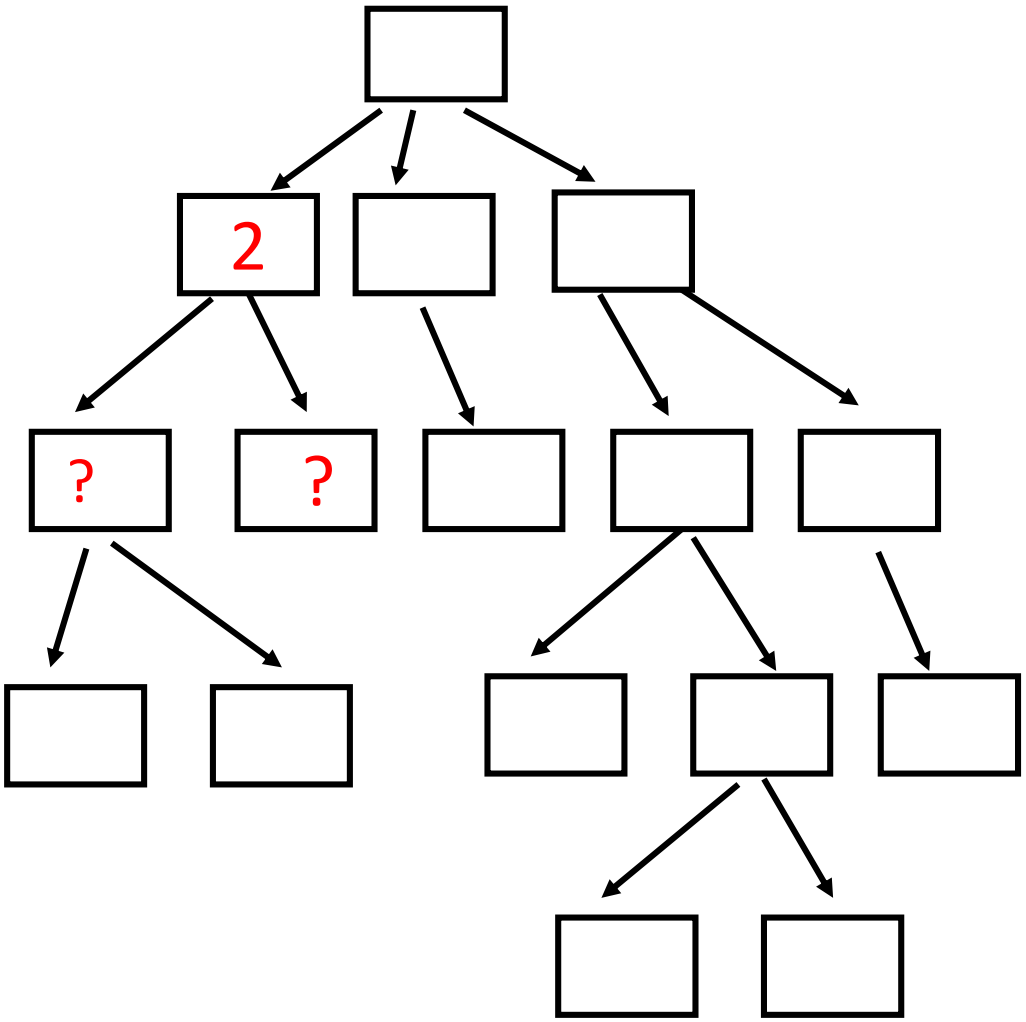
```
depth( v ){
    if ( v.parent == null)     //root
        return   0
    else
        return   1 + depth( v.parent )
}
```



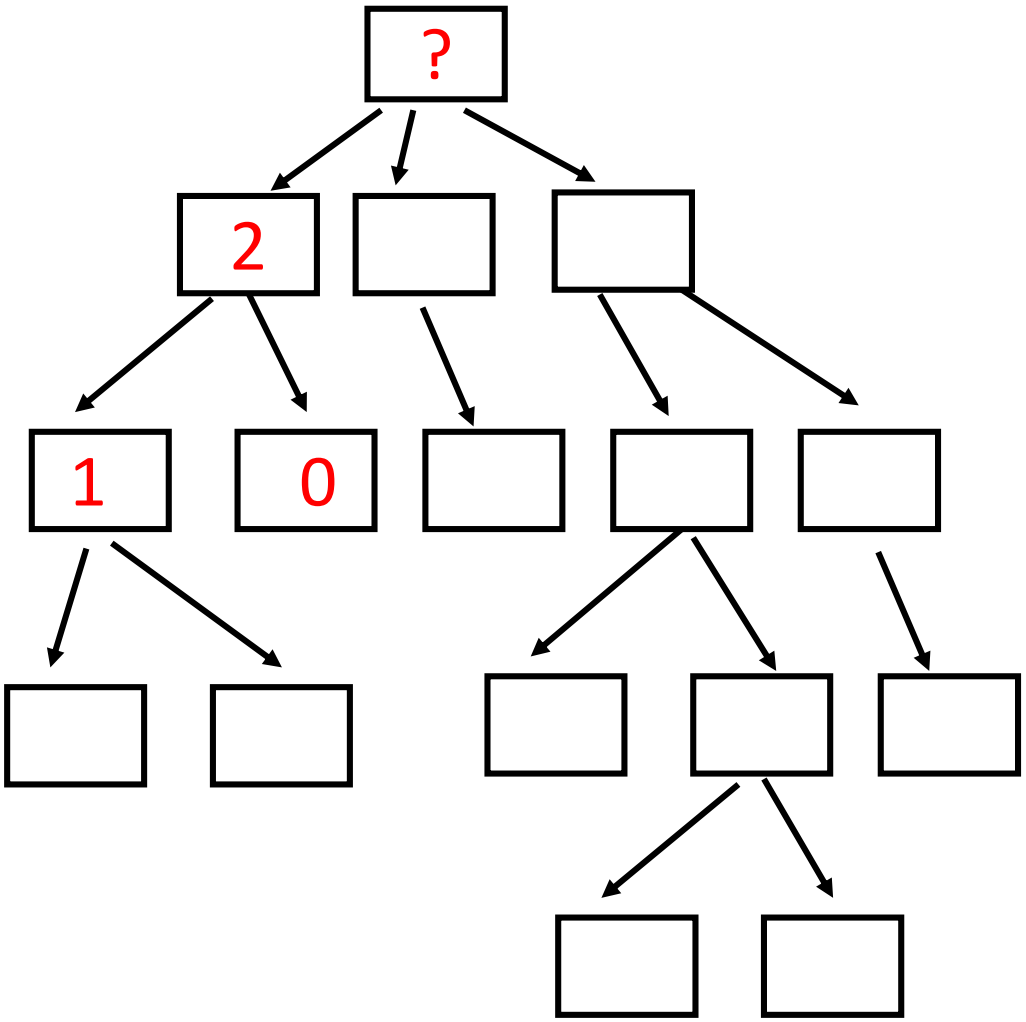
The height of a node is the maximum length of a path from that node to a leaf.



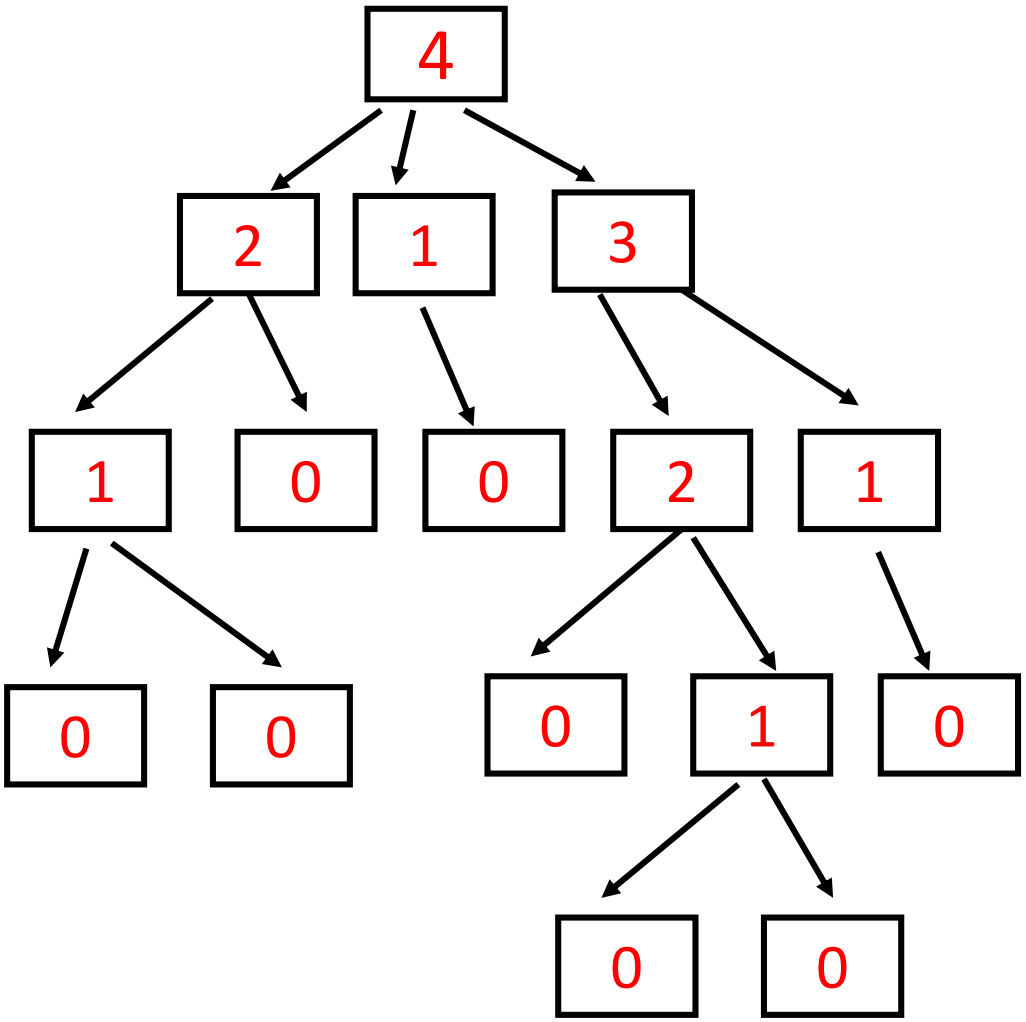
The height of a node is the maximum length of a path from that node to a leaf.

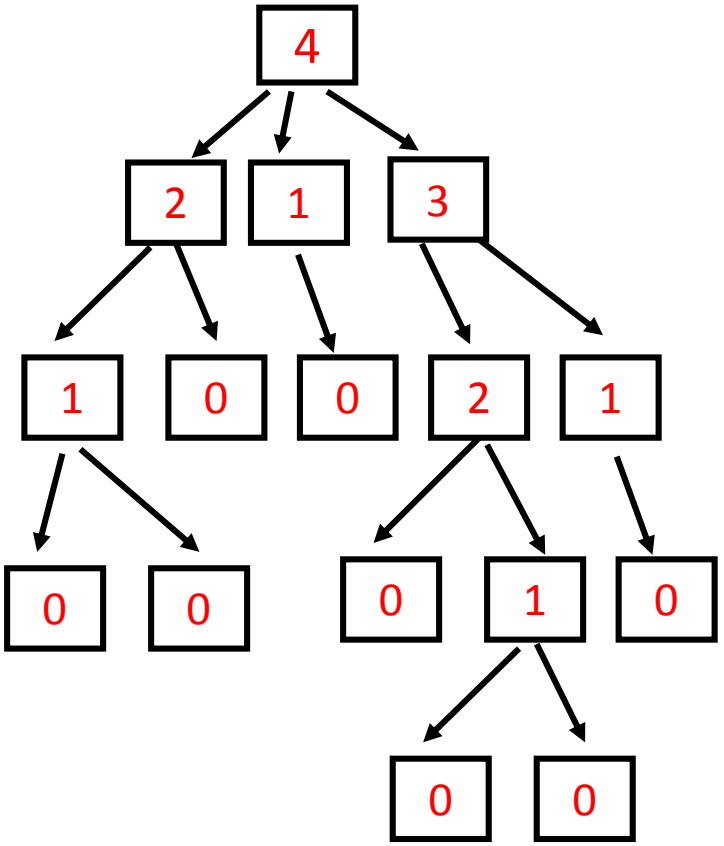


The height of a node is the maximum length of a path from that node to a leaf.



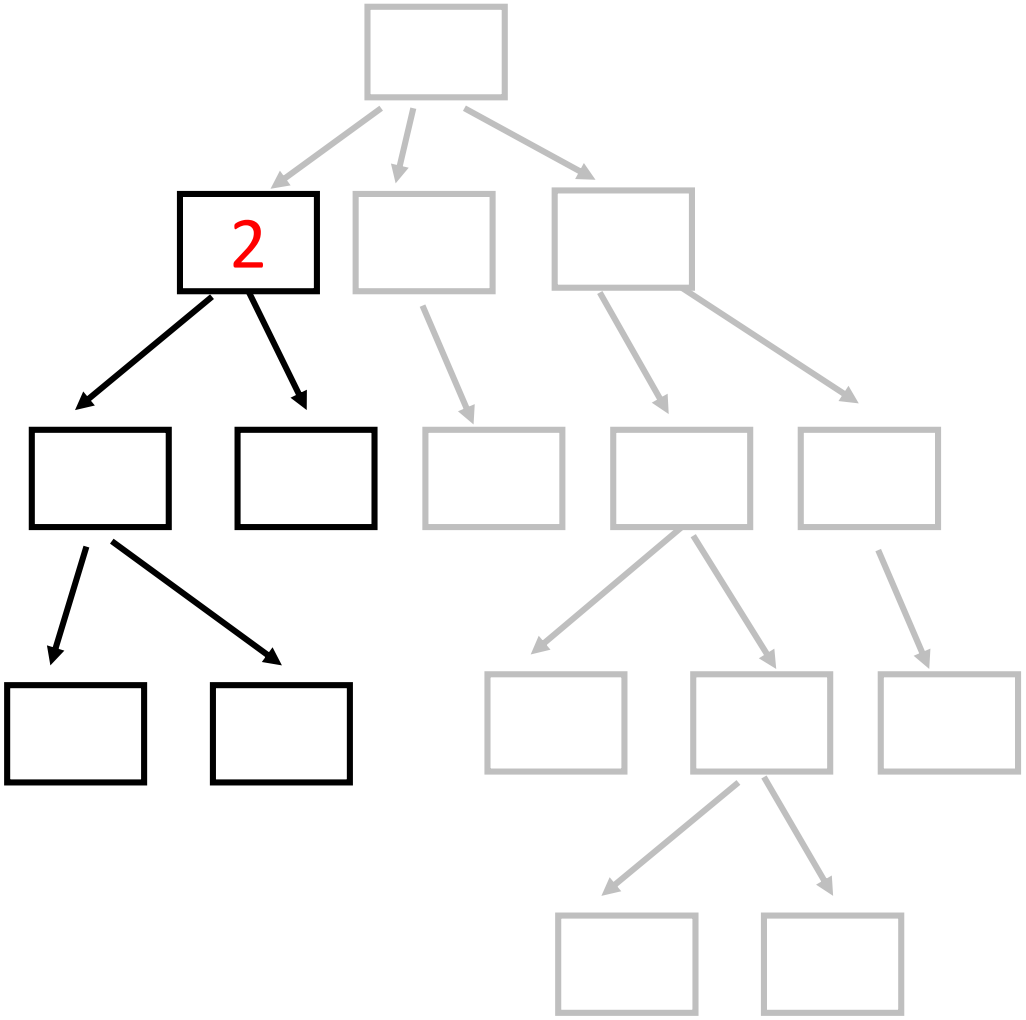
How to compute height(v) ?






```
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
    }
}
```

The height of a node is the maximum length of a path from that node to a leaf. **To understand the definition, think of the height of the subtree defined by that node.**



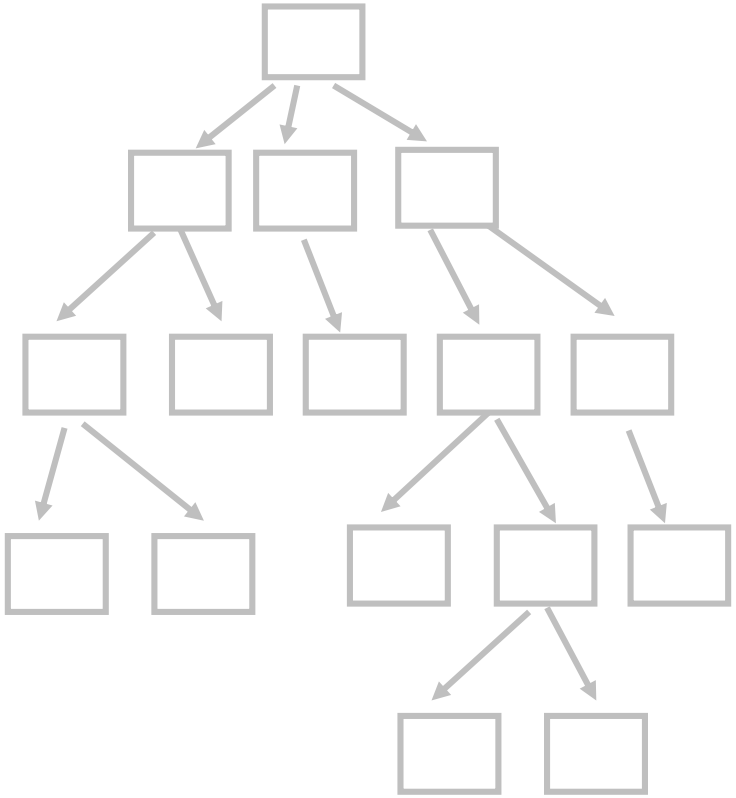
How to implement a tree in Java?

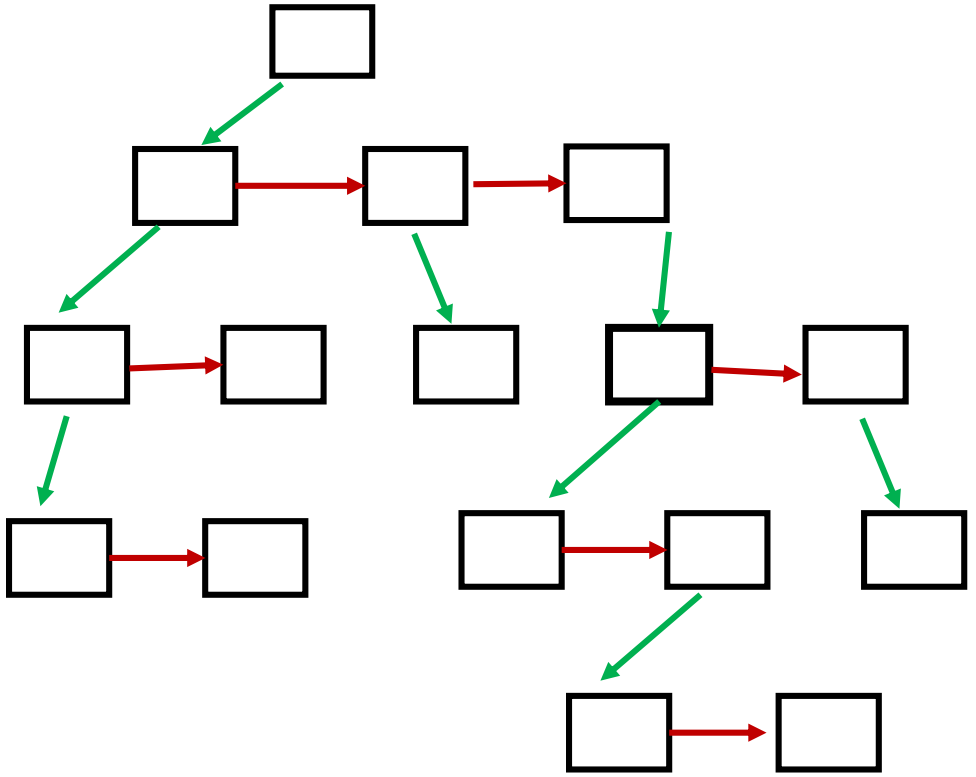
```
class  TreeNode<T>{

    T   element;




}
```

How to implement a tree in Java?

```
class  TreeNode<T>{

    T   element;

    ArrayList< TreeNode<T> >     children;

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

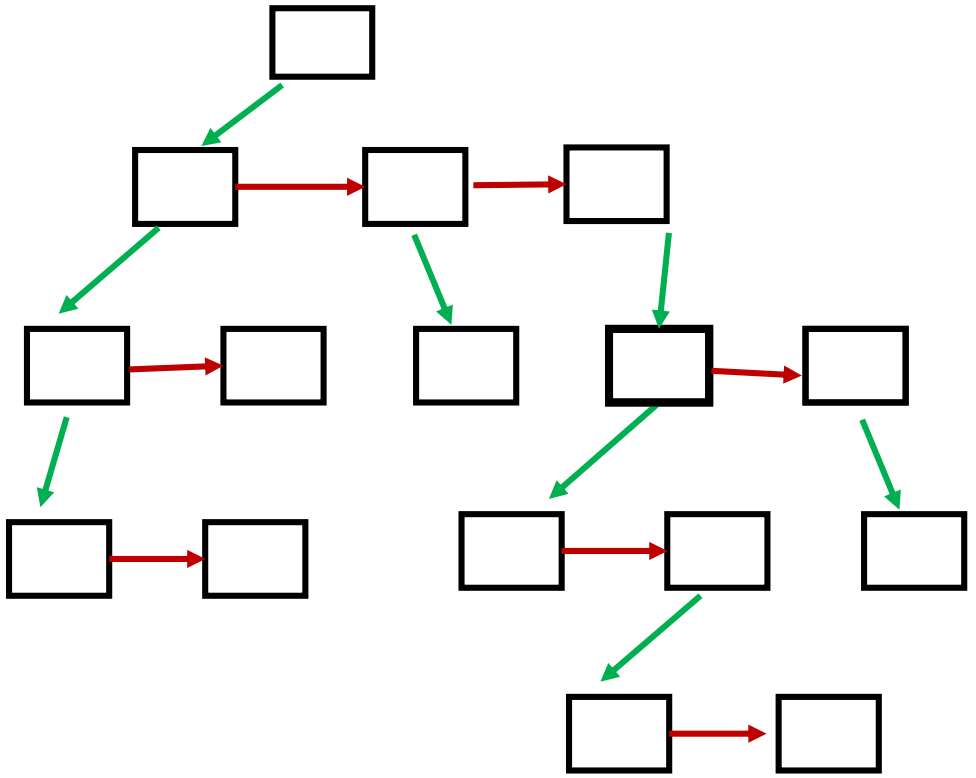
Another common implementation: ‘first child, next sibling’





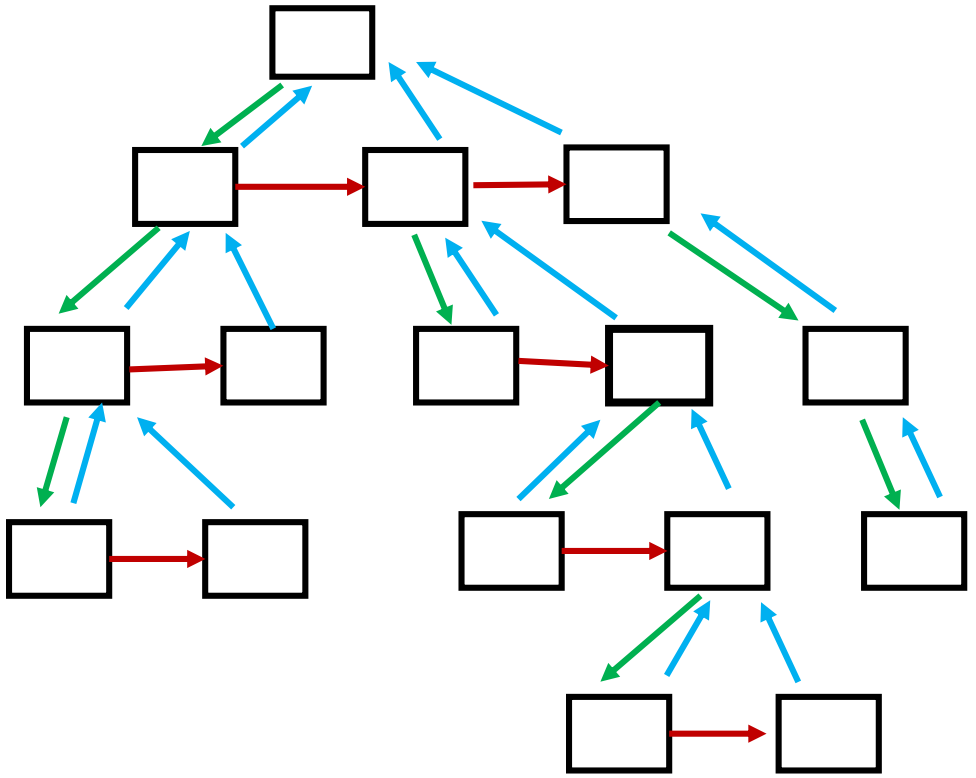
More common implementation: ‘first child, next sibling’ (similar to singly linked lists)

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

    // inner class

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



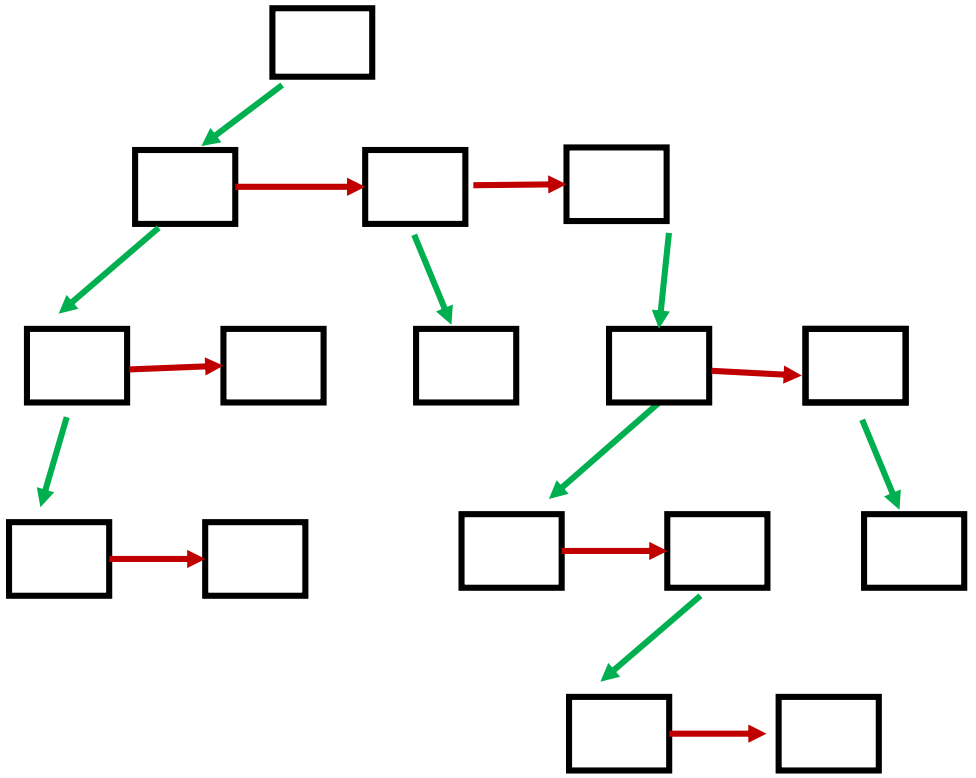
More common implementation: ‘first child, next sibling’ (similar to singly linked lists)

```
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;
        :
    }
}
```



Announcements

- Friday Nov. 2 lecture for Sec. 001 (10:35-11:25) will be in ENGTR 0100

- Quiz 3 on Friday Nov . 2

  (OOD, induction&recursion)

- A3 will be posted by early next week ("decision trees")

Exercise (time permitting)

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.

Exercise

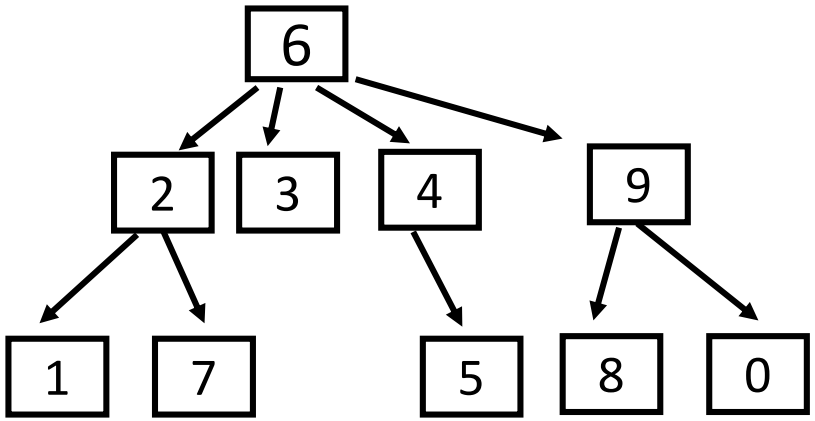
A tree can be represented using lists, as follows:

```
tree   =     root  |   ( root listOfSubTrees )

listOfSubTrees  =  tree | tree listOfSubTrees
```

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

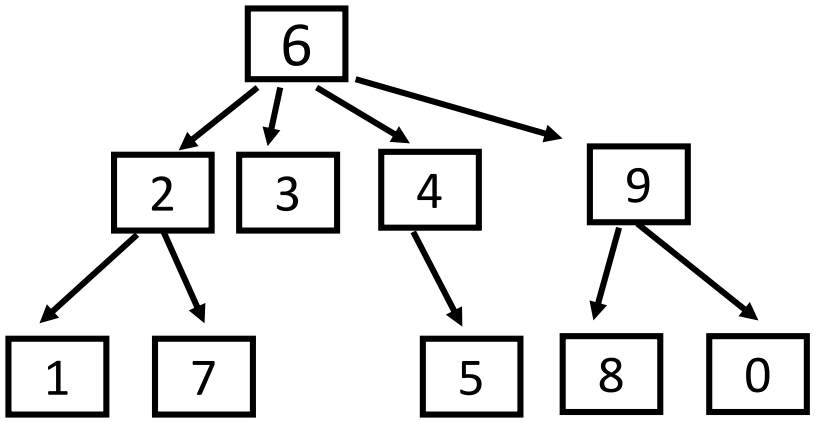
```
( 6 ( 2 1 7 ) 3 ( 4 5 ) ( 9 8 0 ) )
```

Exercise

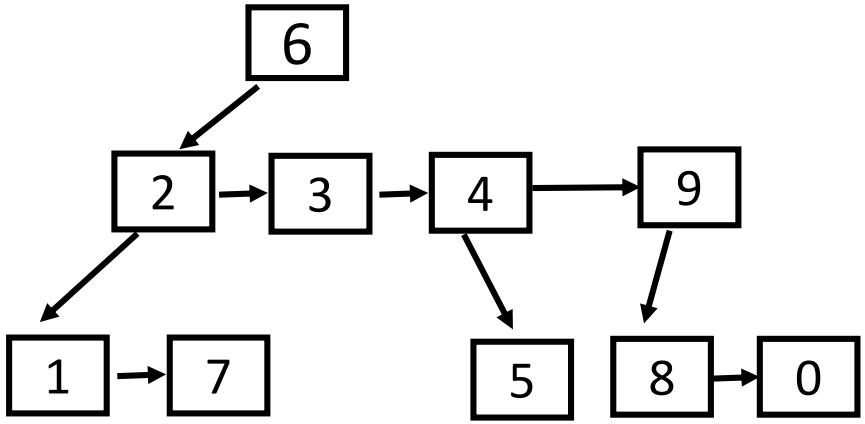


(6 (217) 3 (45) (980))

Exercise



first child, next sibling



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