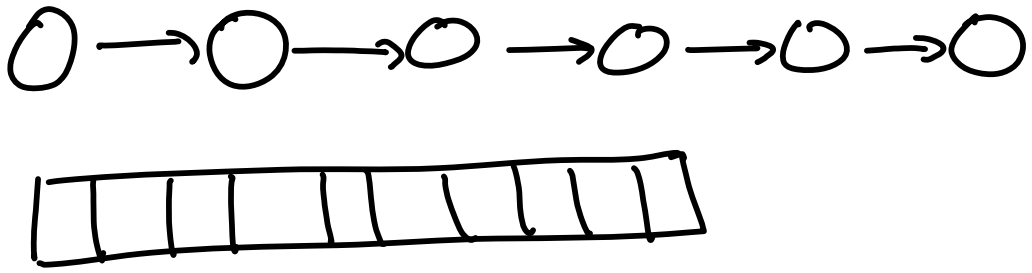
COMP 250

Lecture 17

(rooted) trees

Oct. 19, 2016

Linear Data Structures

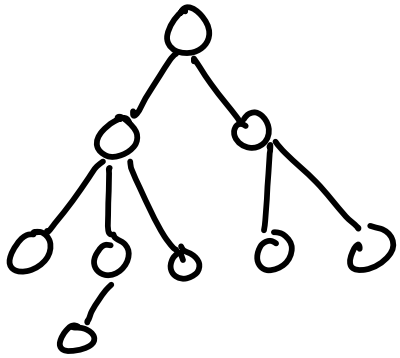


linked list

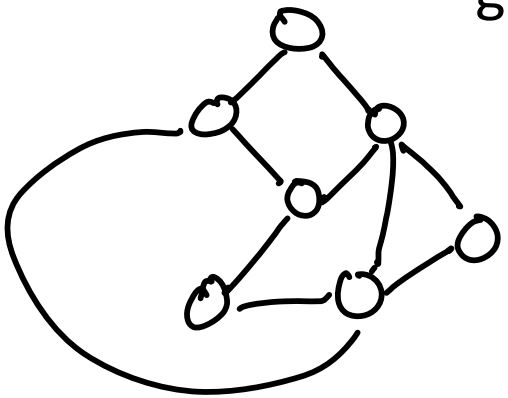
array

Non-Linear Data Structures

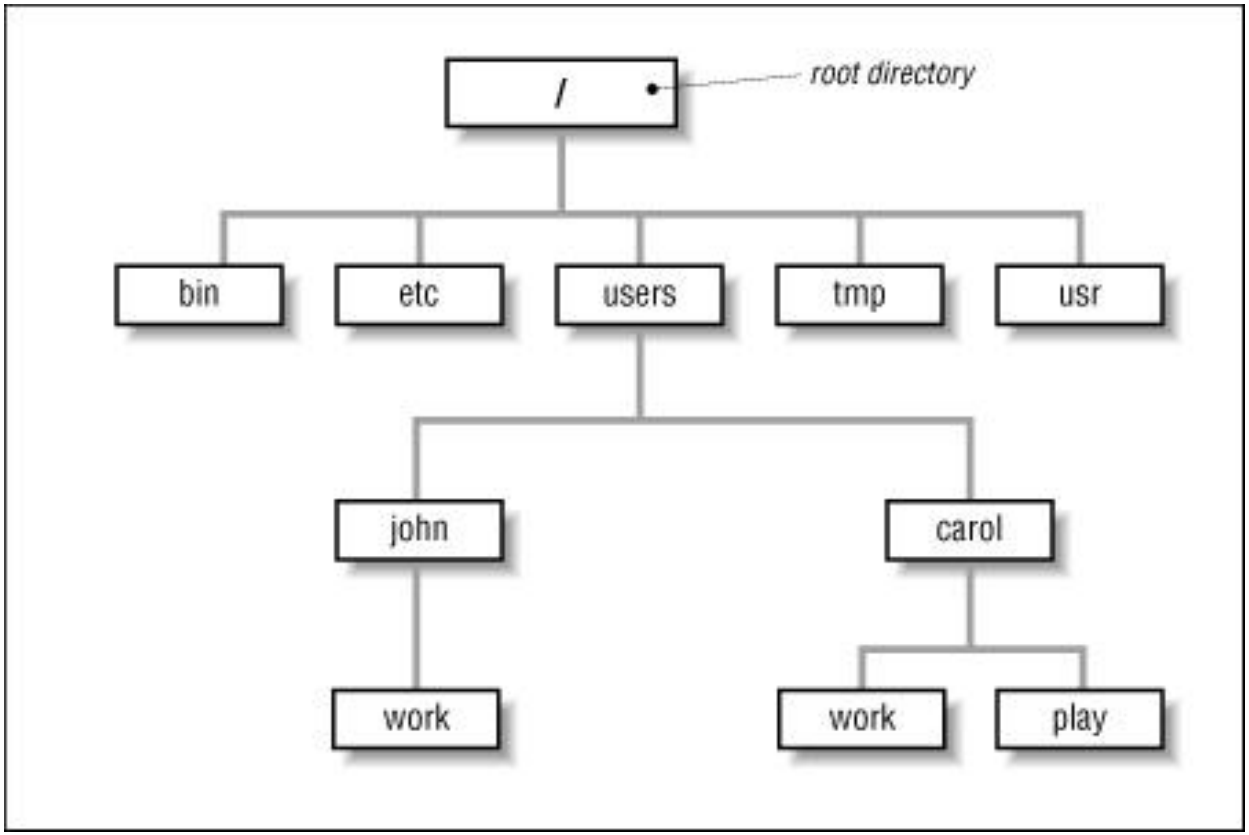
tree



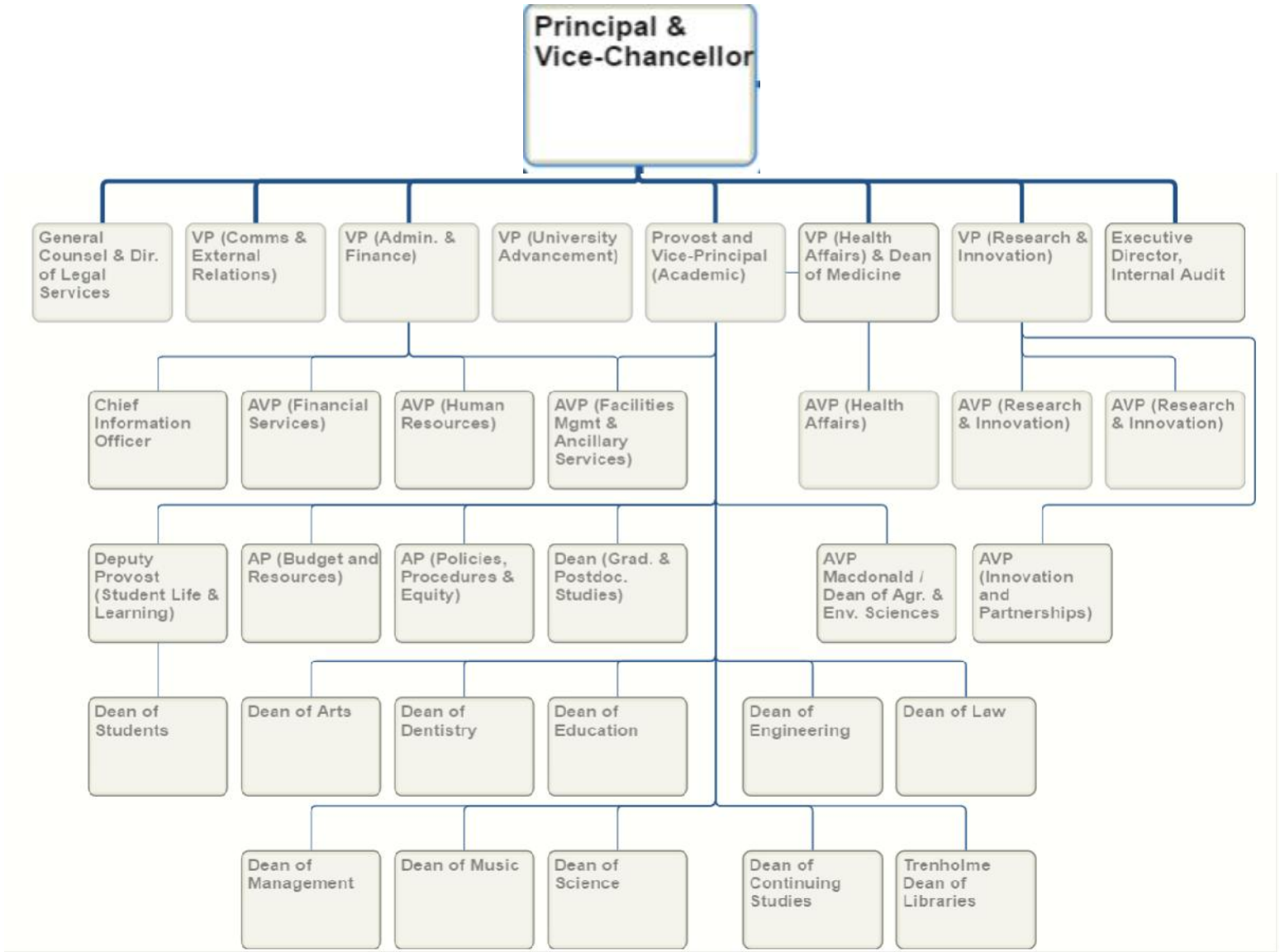
graph



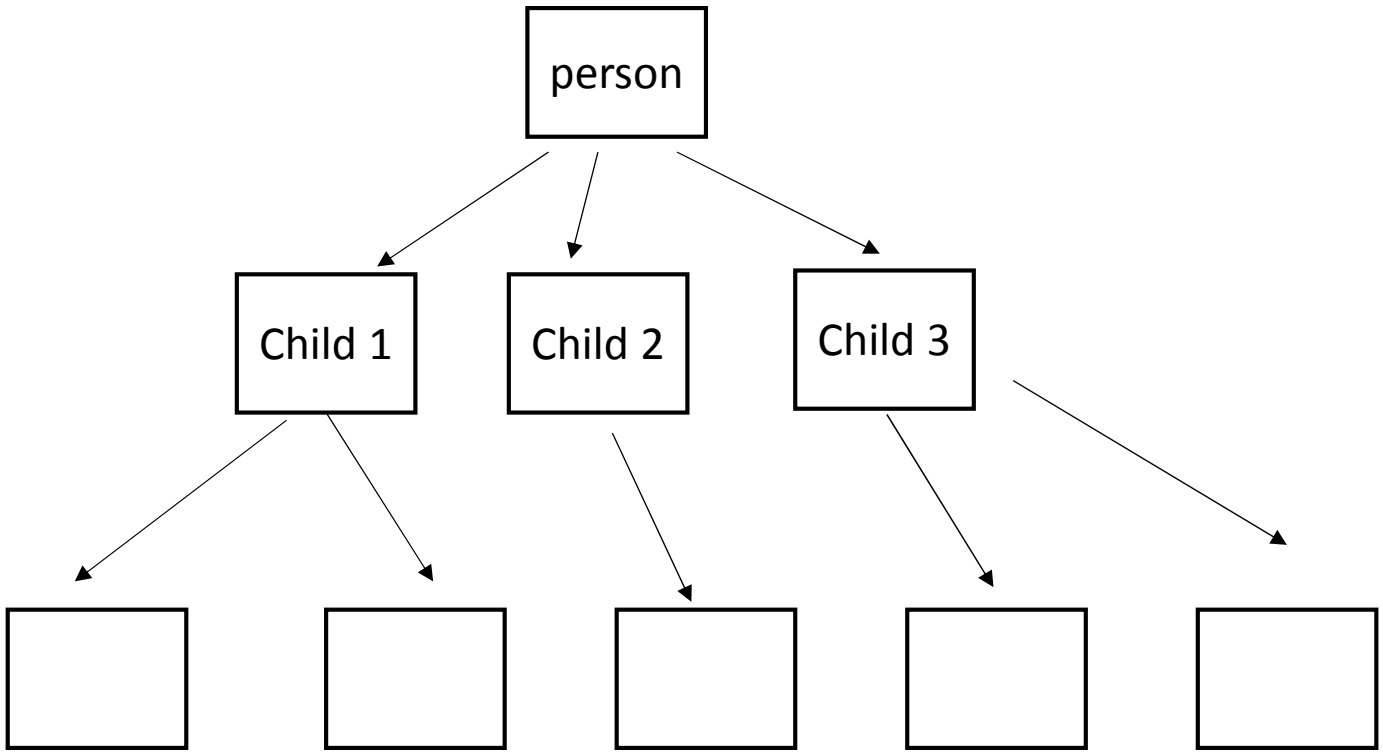
Tree e.g. UNIX file system



e.g. Organization Hierarchy (McGill)

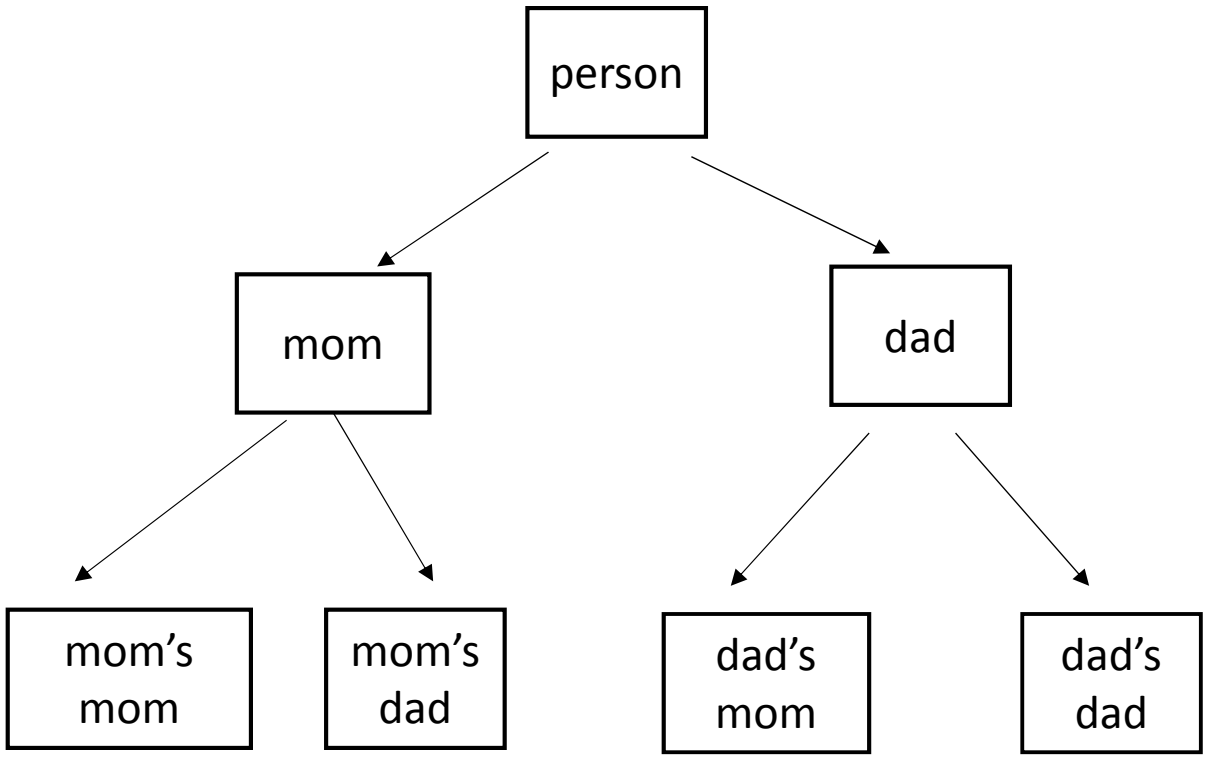


Family Tree (descendents)



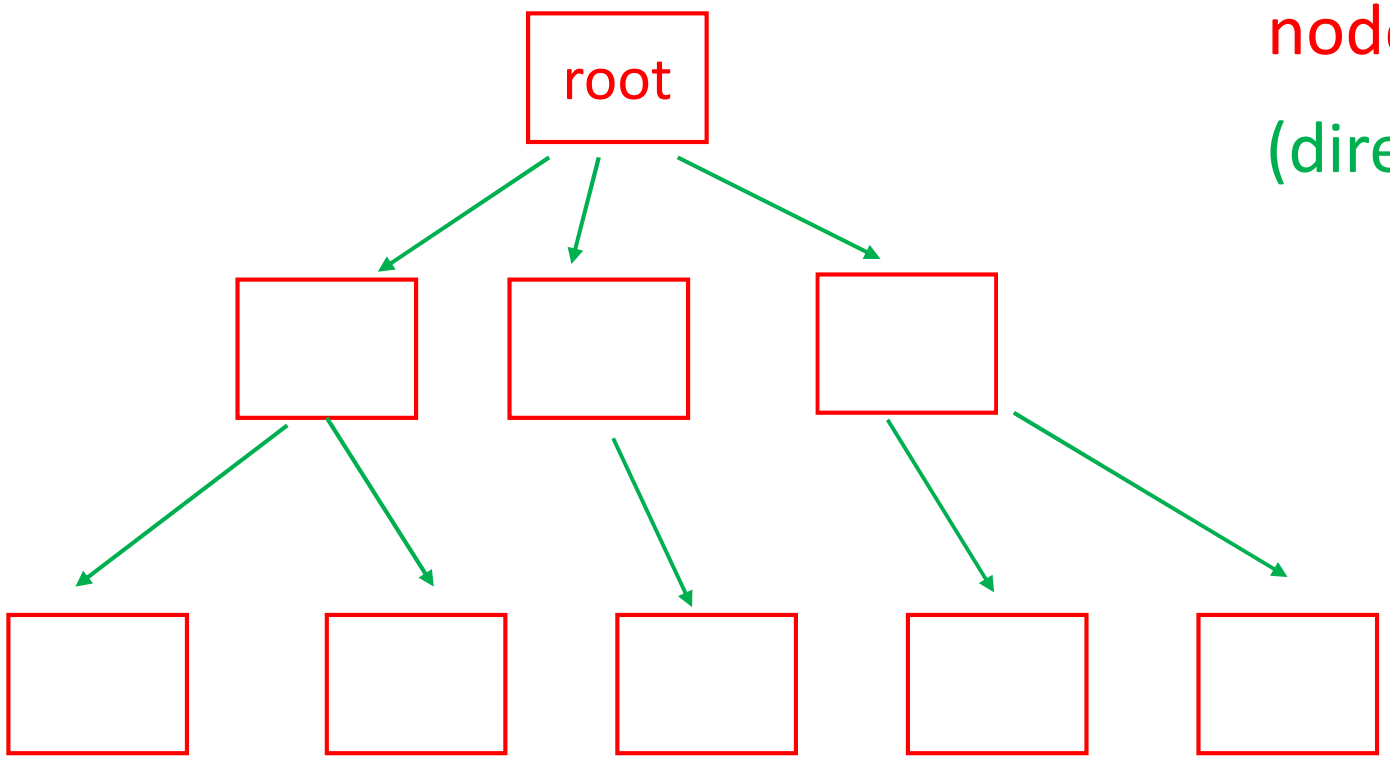
Here I ignore spouses (partner).

Family Tree (ancestors)



This is an example of a binary tree (next week)

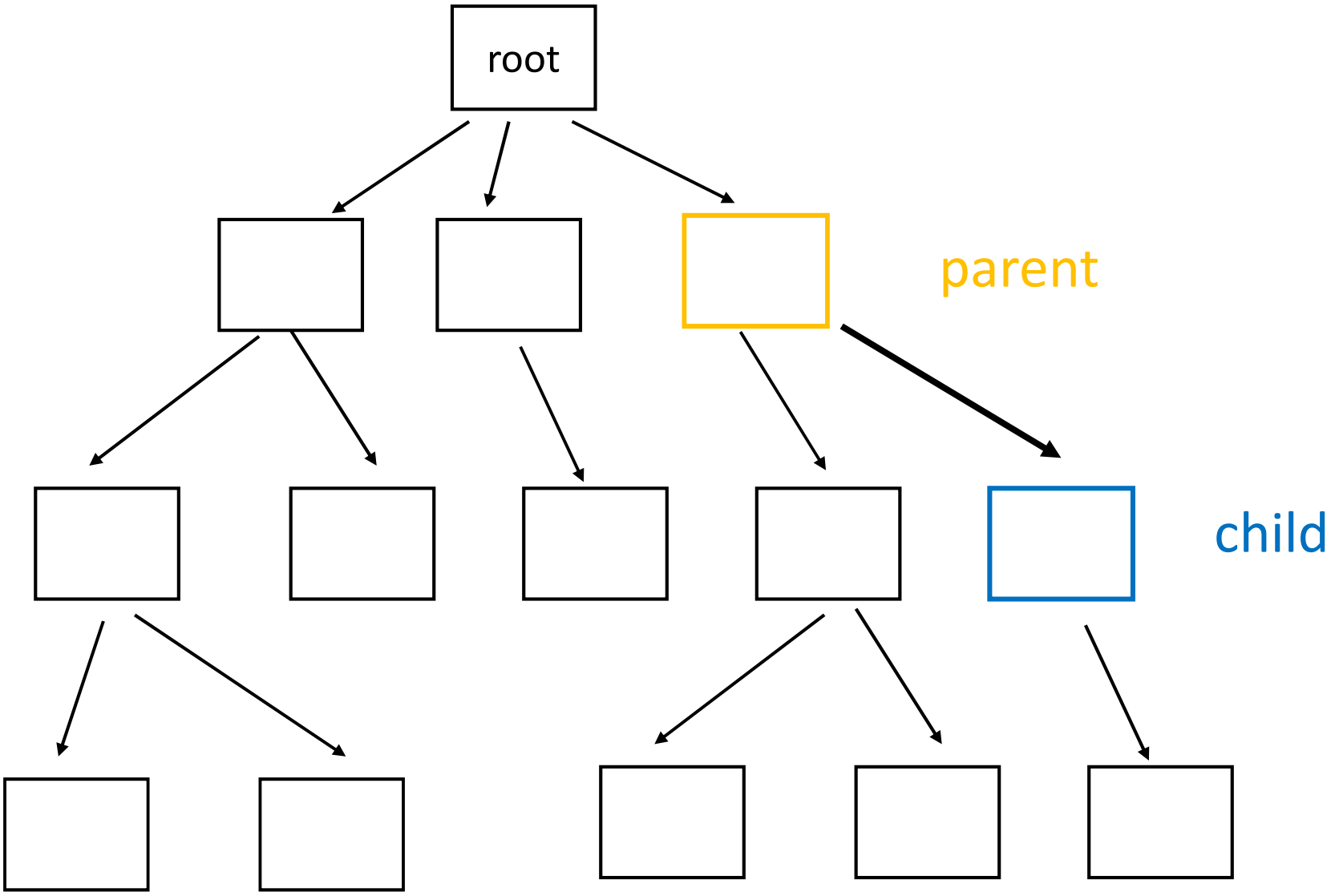
Tree Terminology



node, vertex

(directed) edge

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



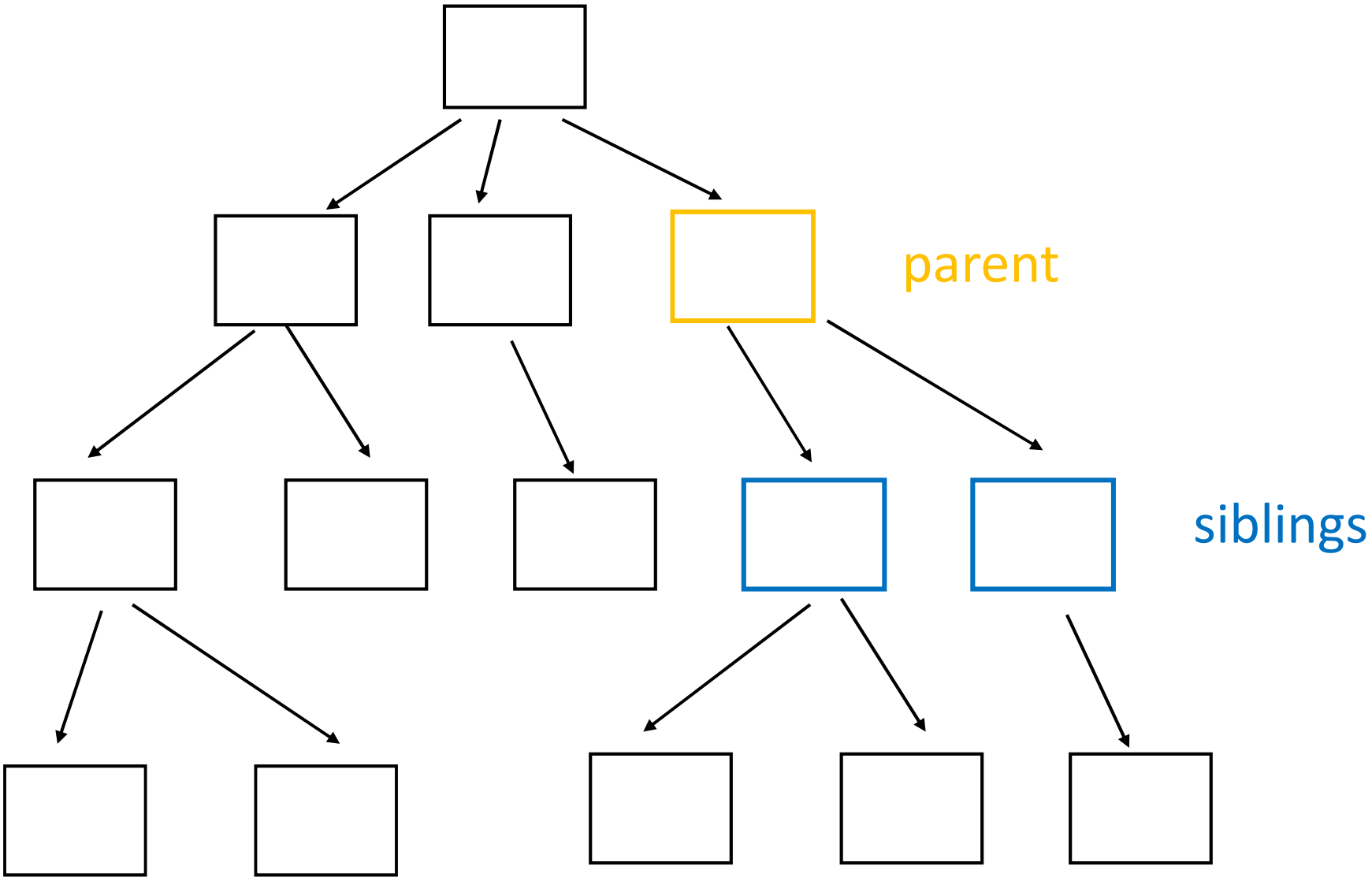
Every node except the root is a child, and has exactly one parent.

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 ?

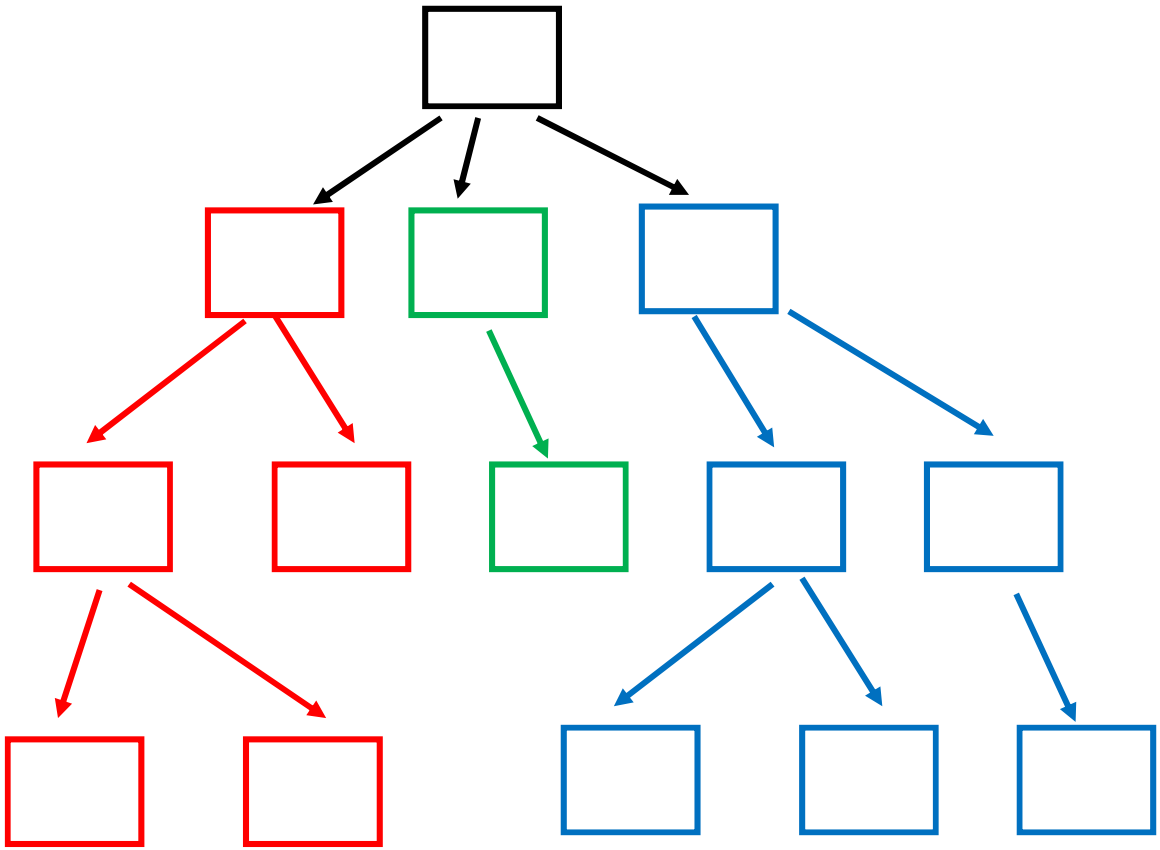
A: N-1

Since every 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.

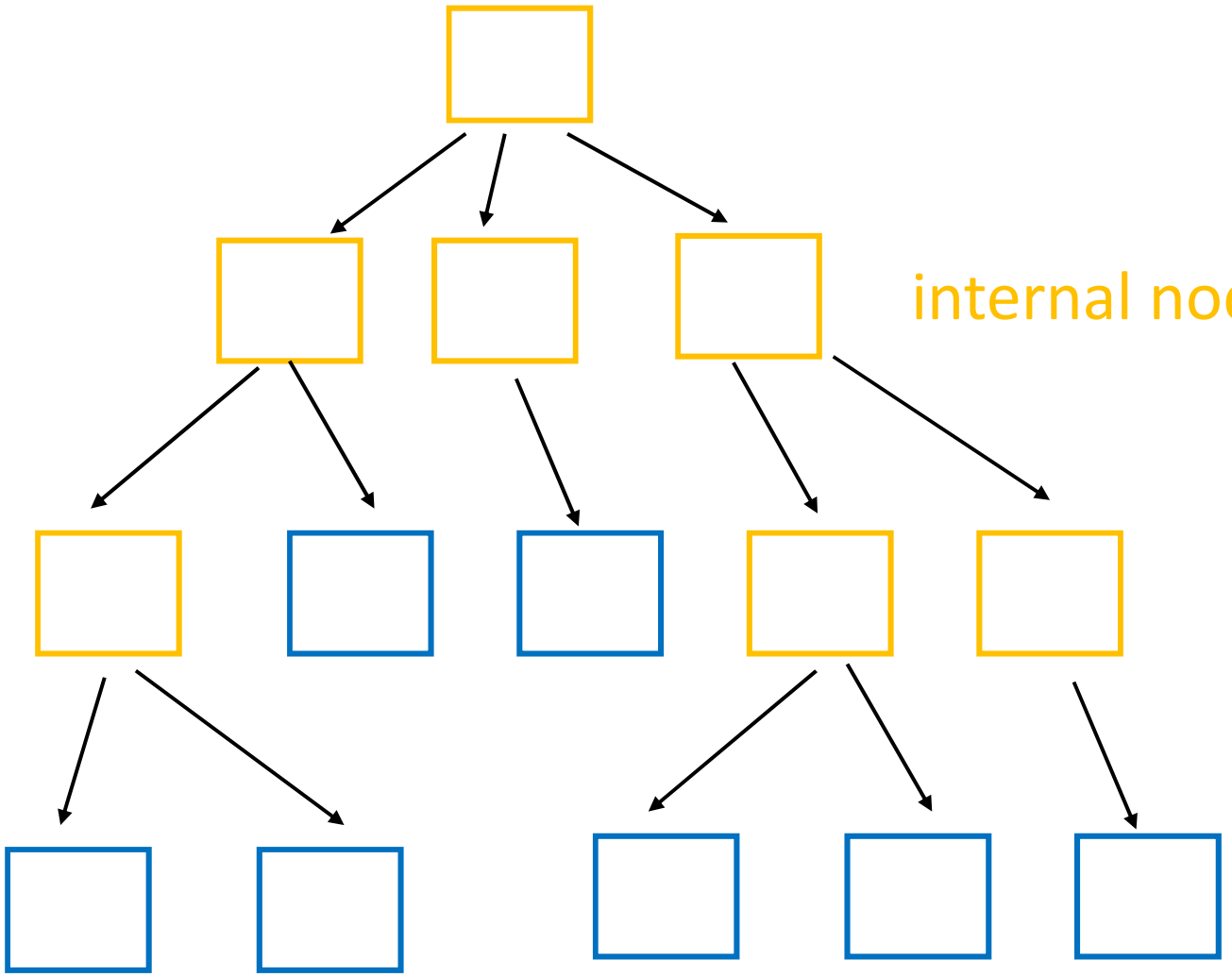
Recursive definition of rooted tree…



Recursive definition of rooted tree

A tree T is a finite (possibly empty) set of nodes such that:

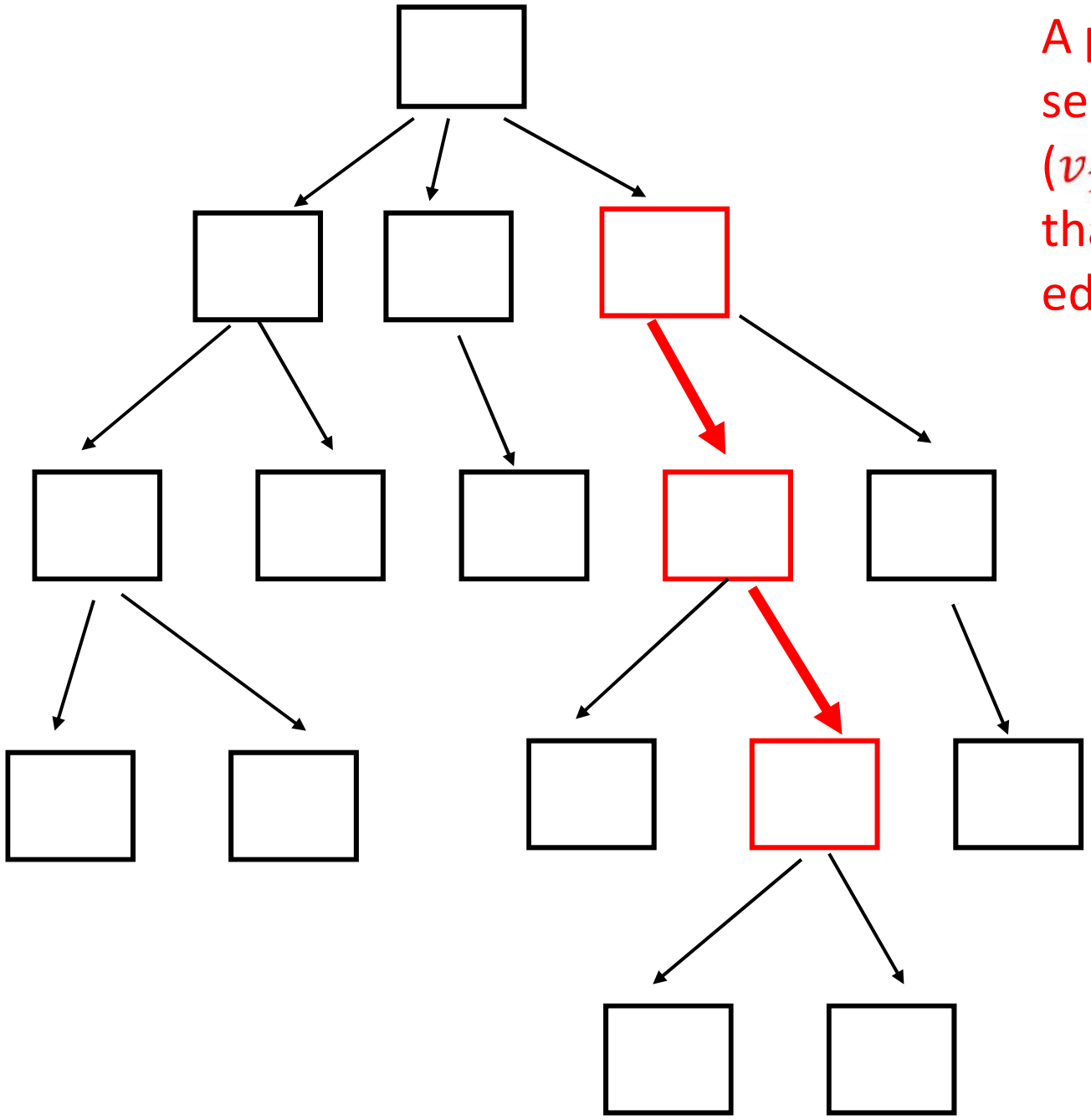
- if the set is non-empty then one of the nodes is the root r
- the non-root nodes are partitioned into subsets
  T1, T2, …, Tk, each of which is a tree (called a "subtree")
- the roots of the subtrees are the children of root r



internal nodes (e.g. file directories)

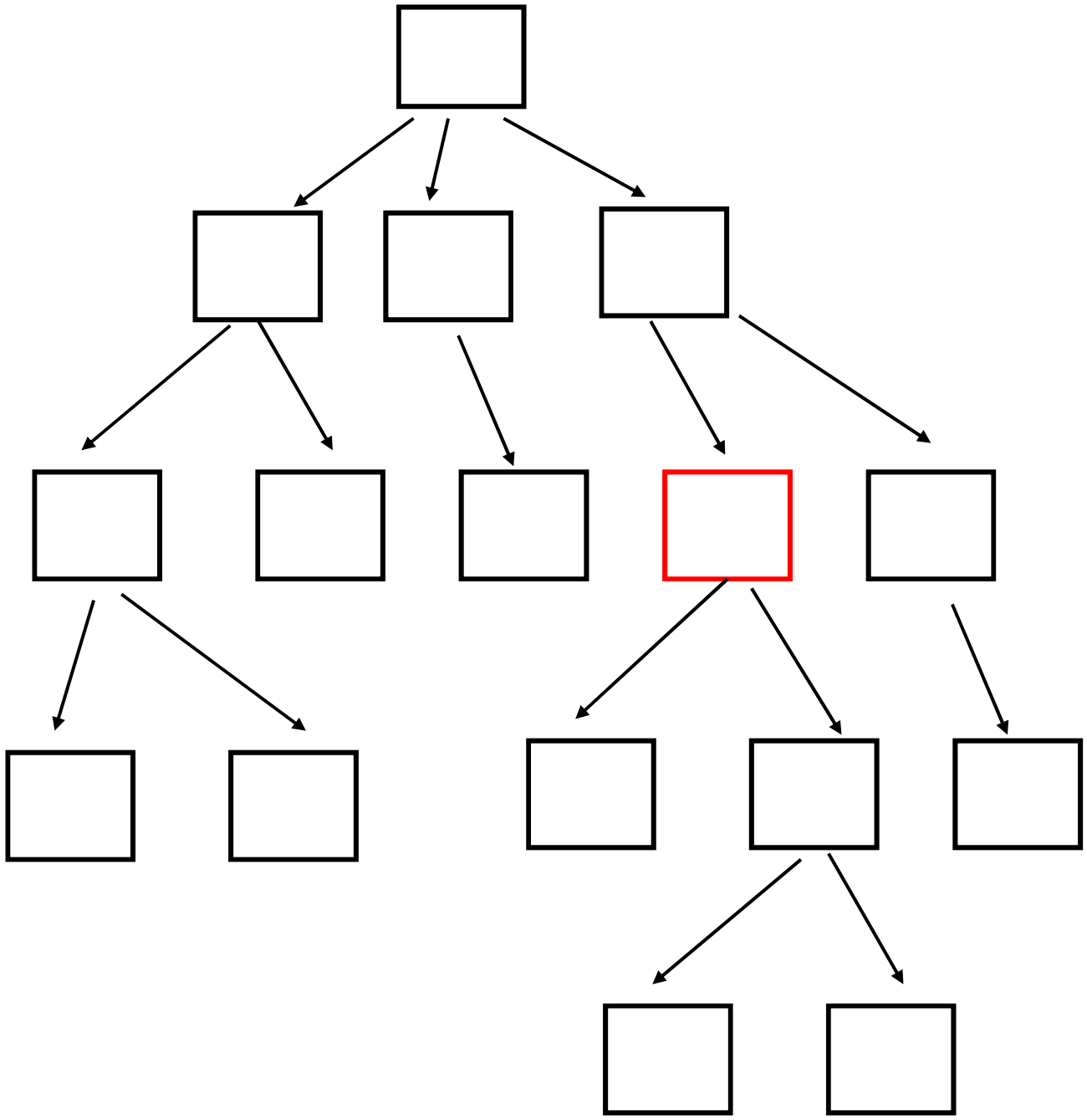
external nodes (leaf)

(e.g. files)

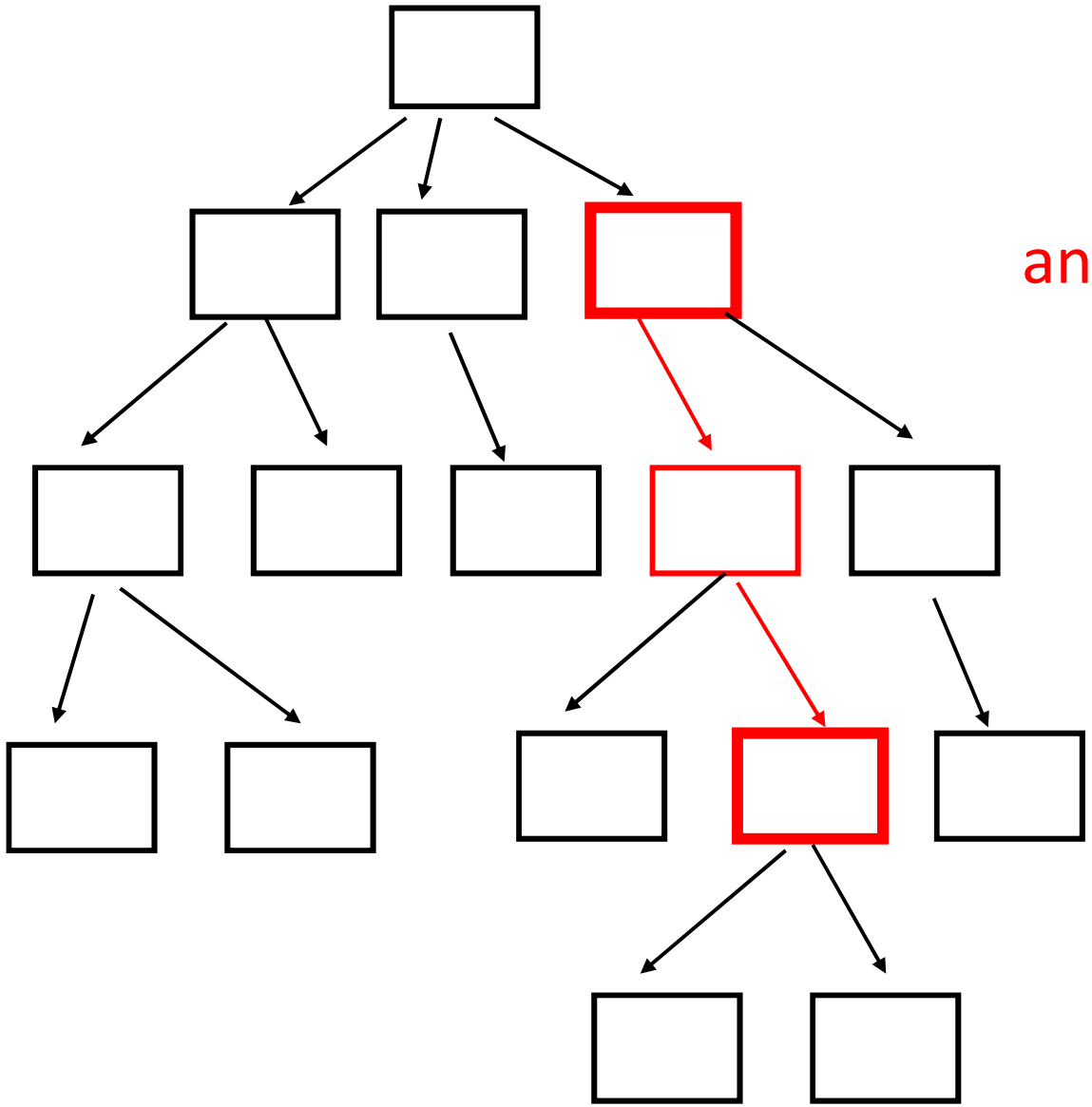


A **path** in a tree is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that (v_i, v_{i+1}) is an edge.

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



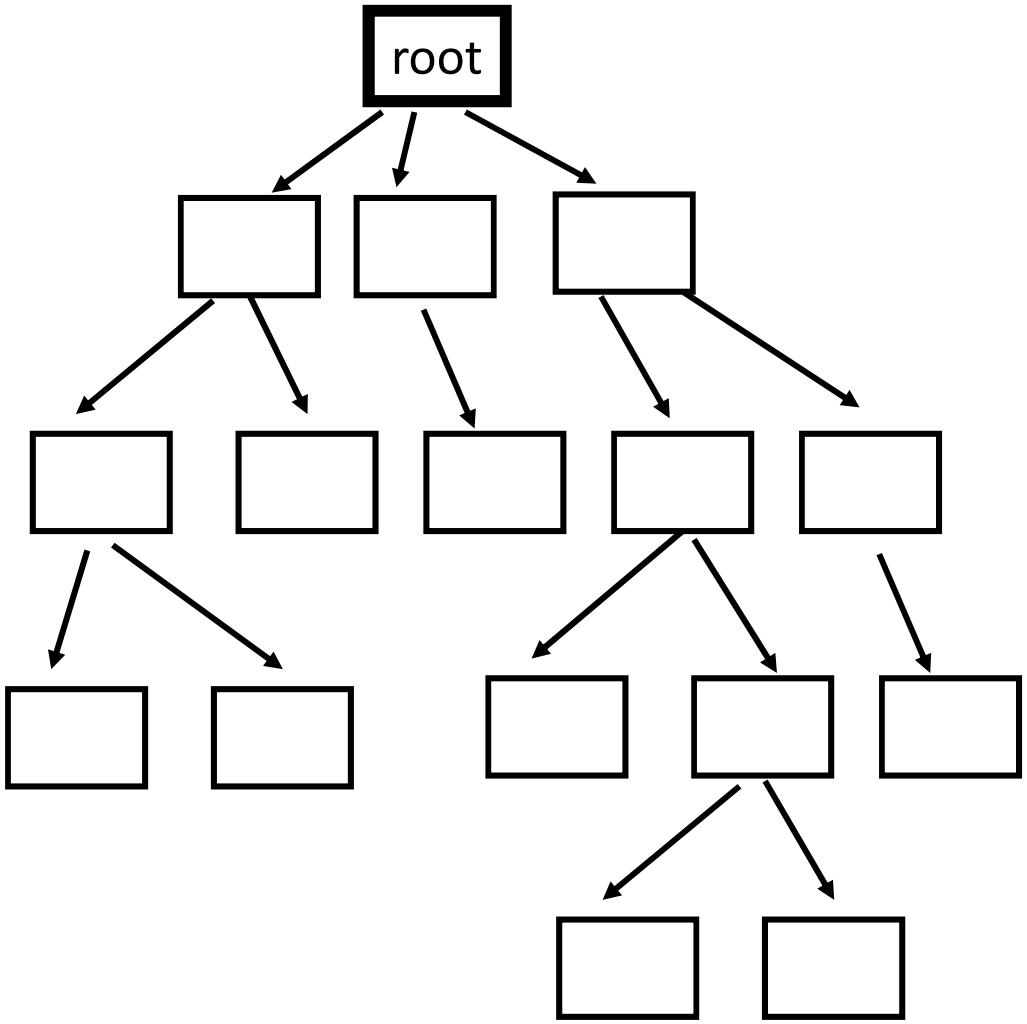
A path with just one node (v_1) has length = 0.



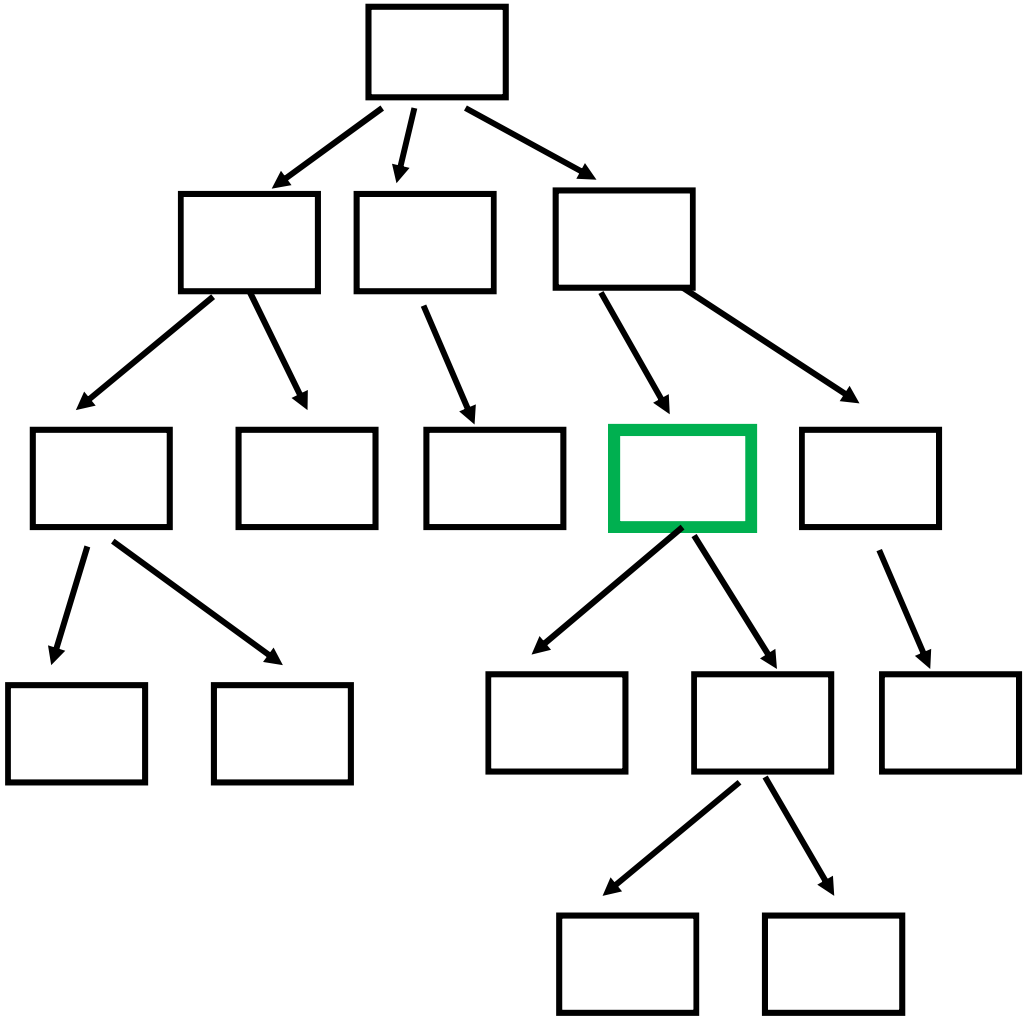
Node v is an ancestor of node w if there is a path from v to w.

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

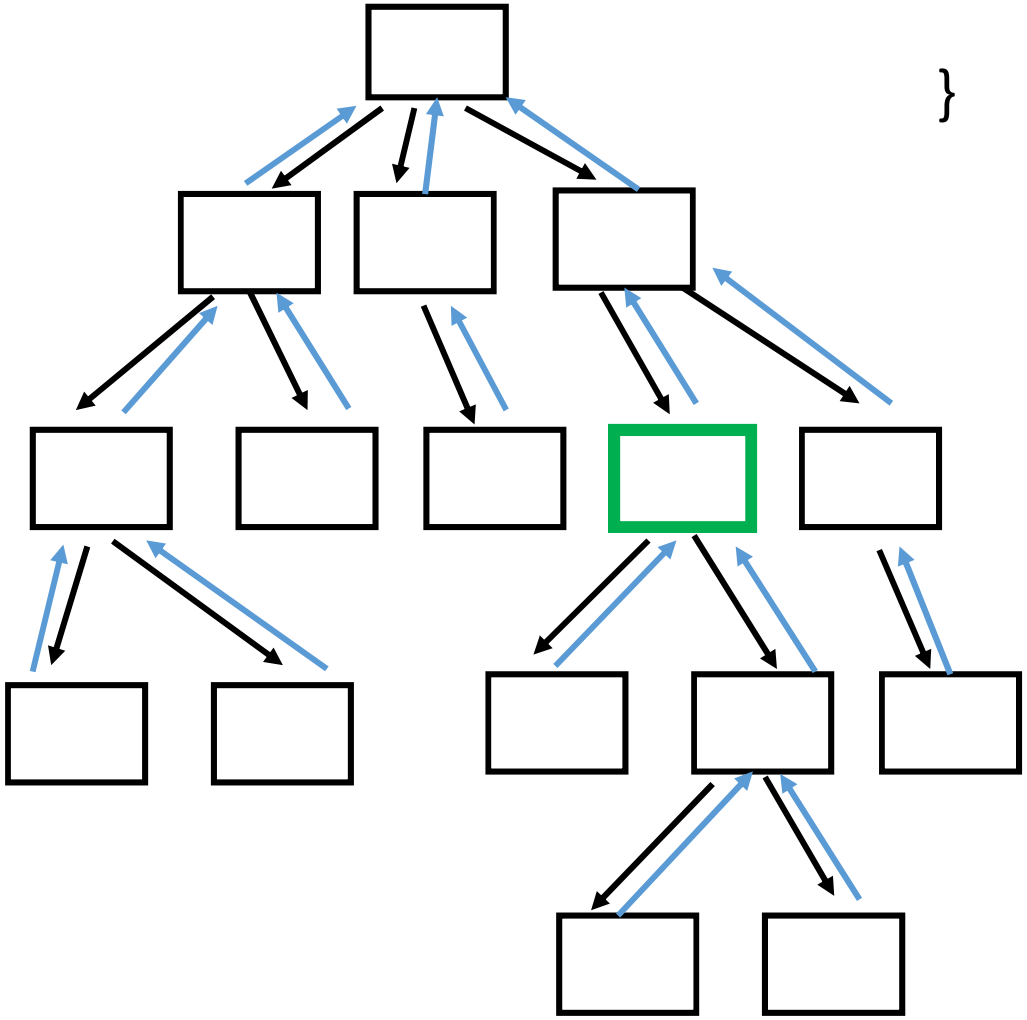
1

2

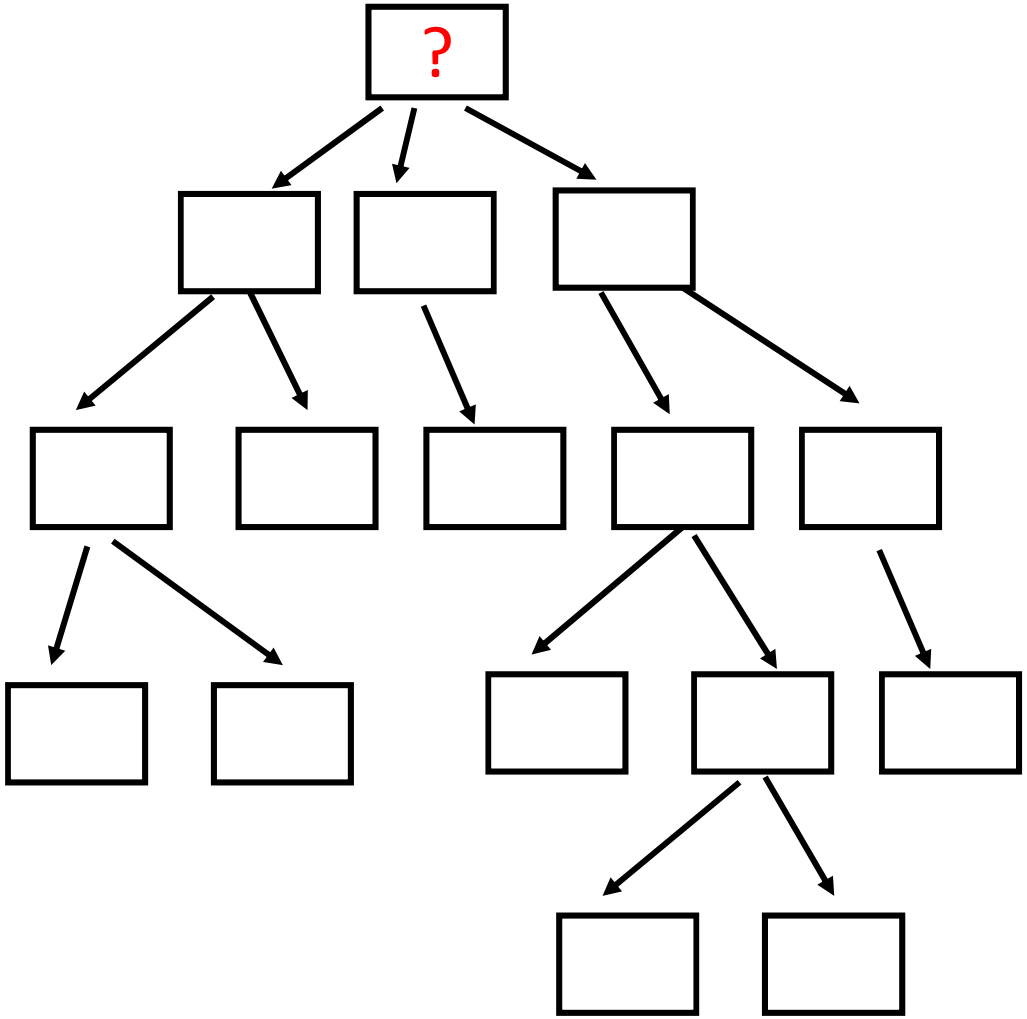
3

4

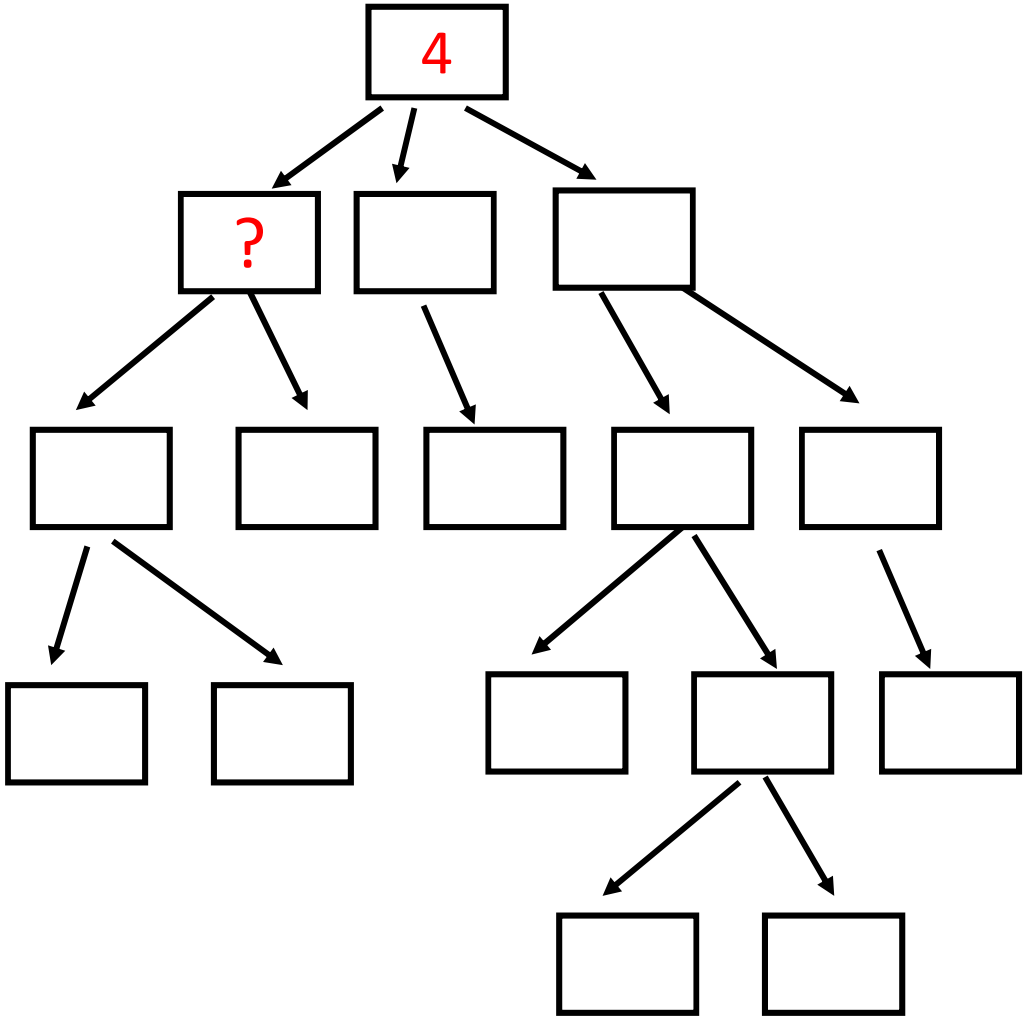
```
depth( v ){
    if ( v.parent == null)     //root
       return    0
    else
       return    1 + depth( v.parent )
}
```



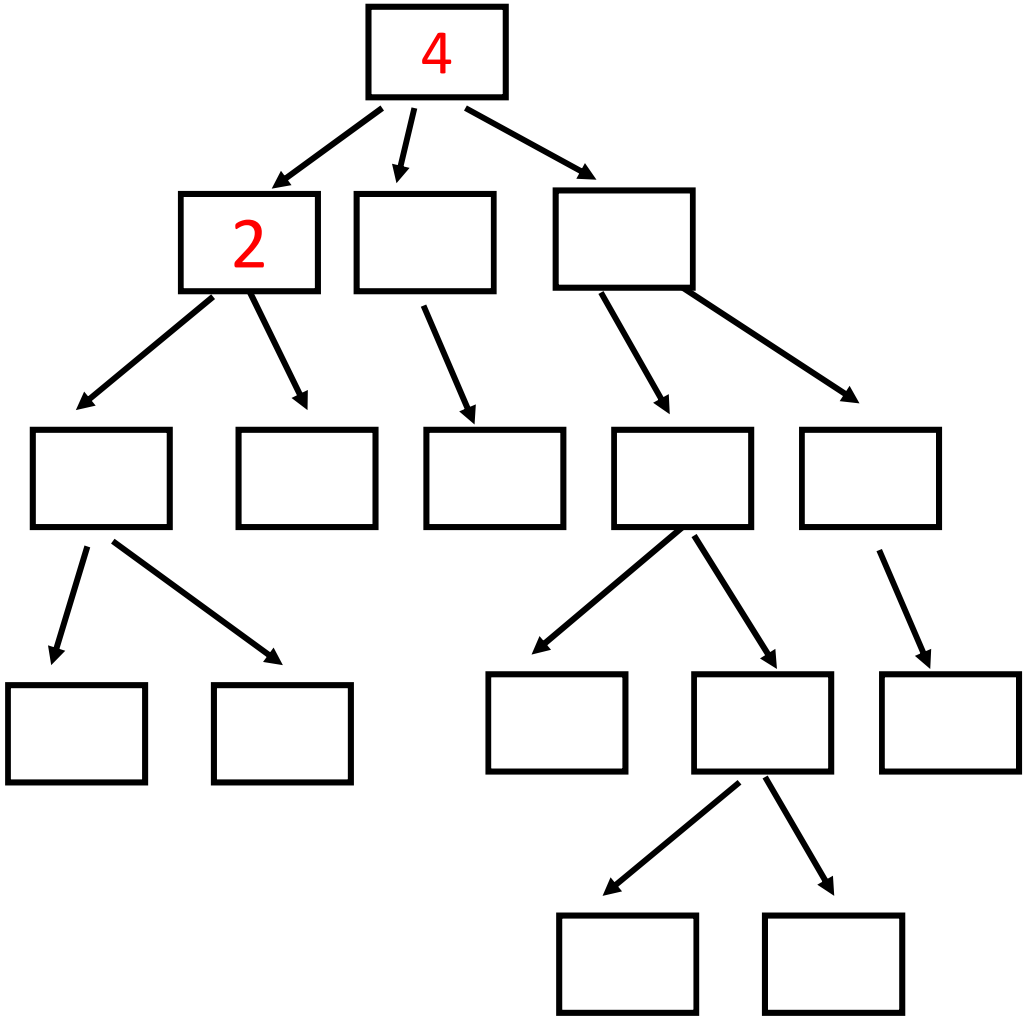
The height of a node is the maximum length of a path from that node to a leaf.



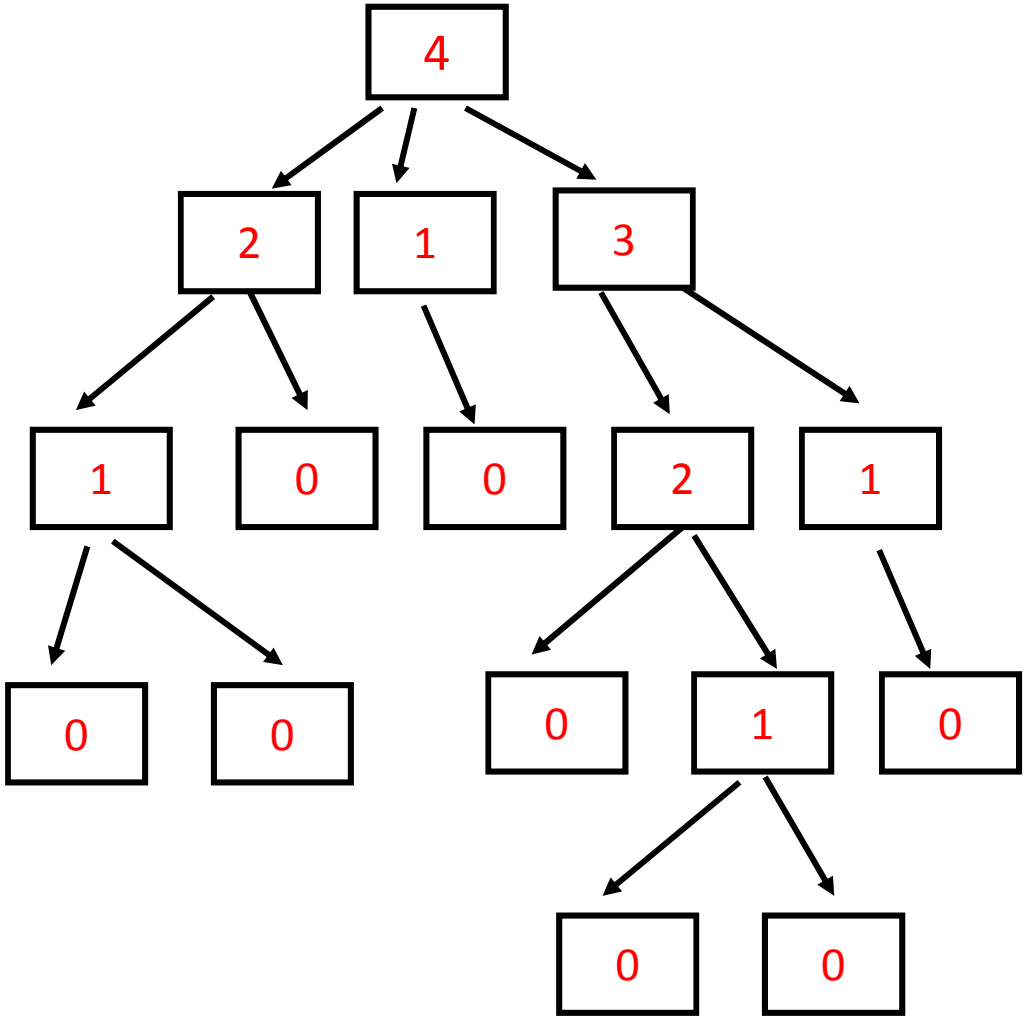
The height of a node is the maximum length of a path from that node to a leaf.

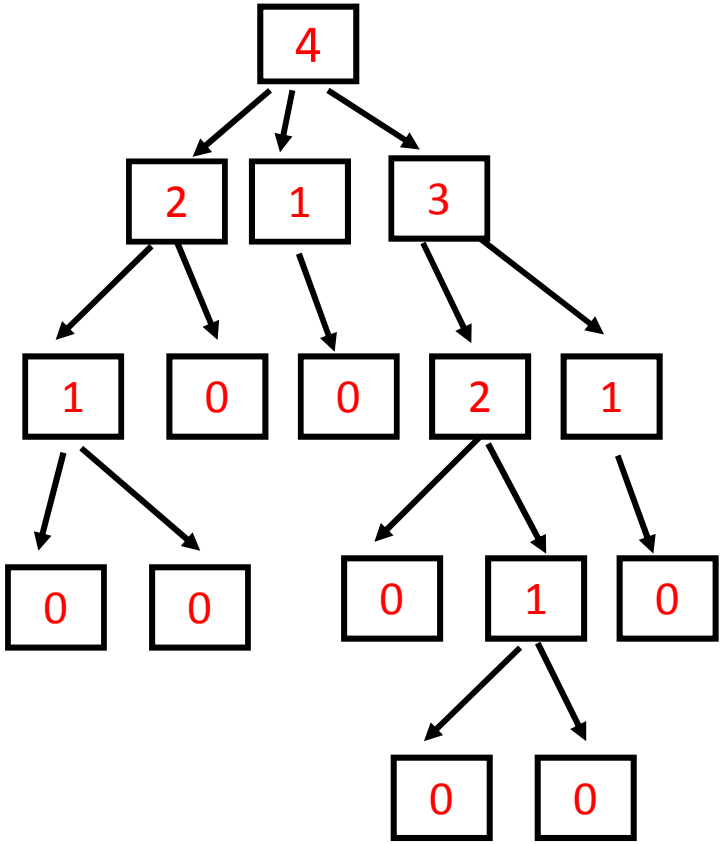


The height of a node is the maximum length of a path from that node to a leaf.



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
    }
}
```

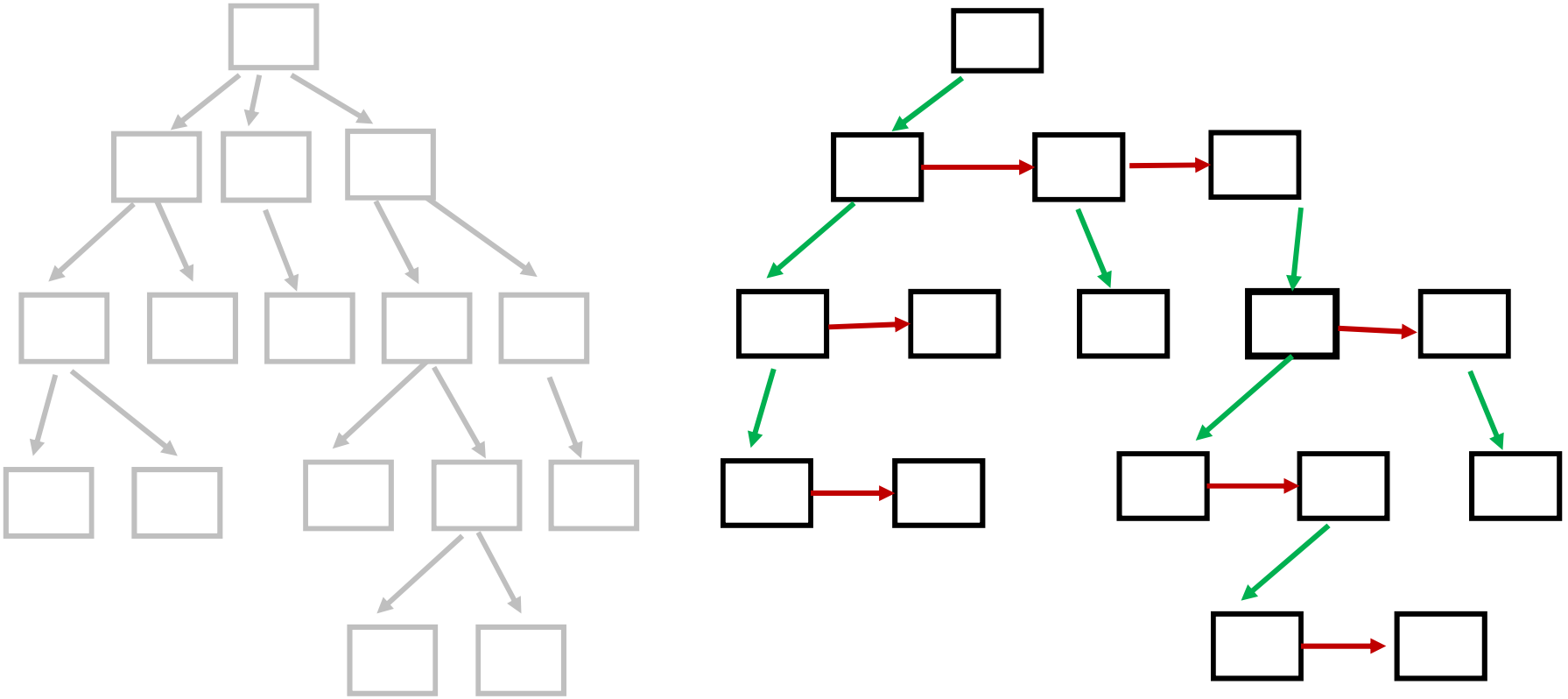
How to implement a tree ?

```
class  TreeNode<T>{

    T   element;




}
```

How to implement a tree ?

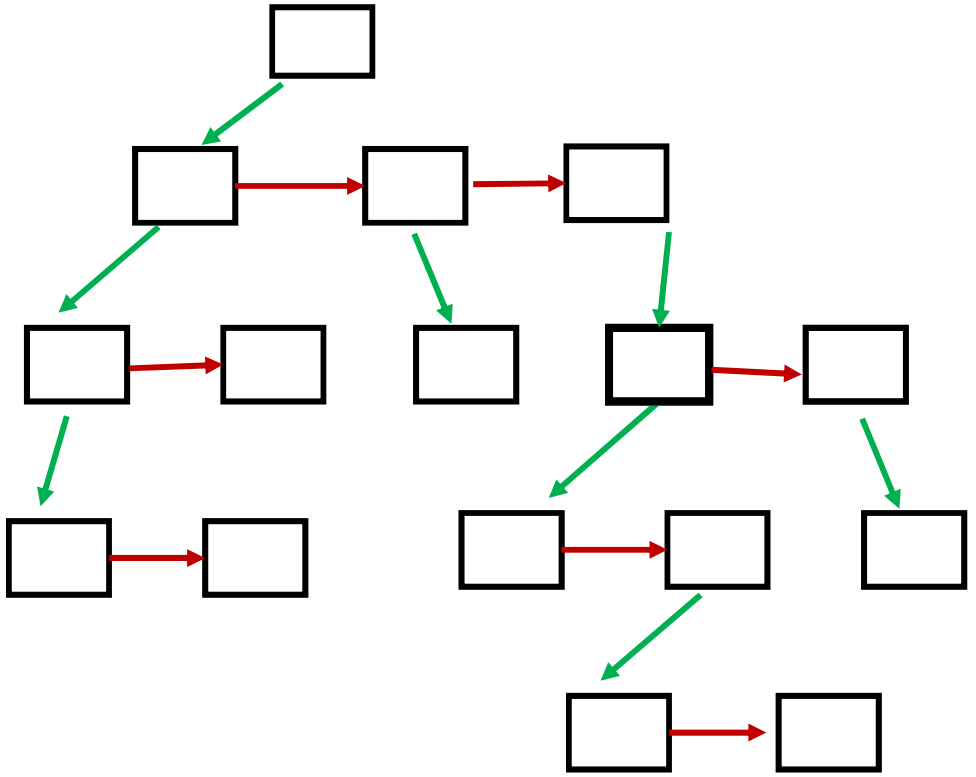
```
class  TreeNode<T>{

    T   element;

    ArrayList< TreeNode<T> >     children;

    TreeNode<T>                  parent;
}
```

Another common implementation: ‘first child, next sibling’



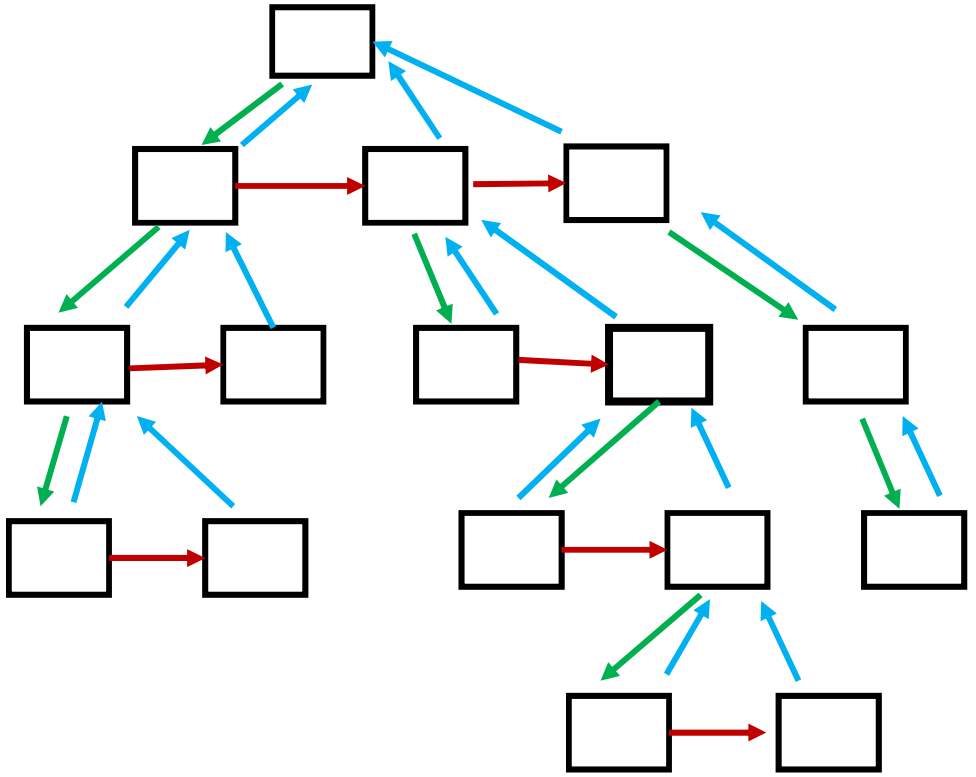
More common implementation: ‘first child, next sibling’

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

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



More common implementation: ‘first child, next sibling’

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

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



A tree of what? Each node also has an element (not shown)

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

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