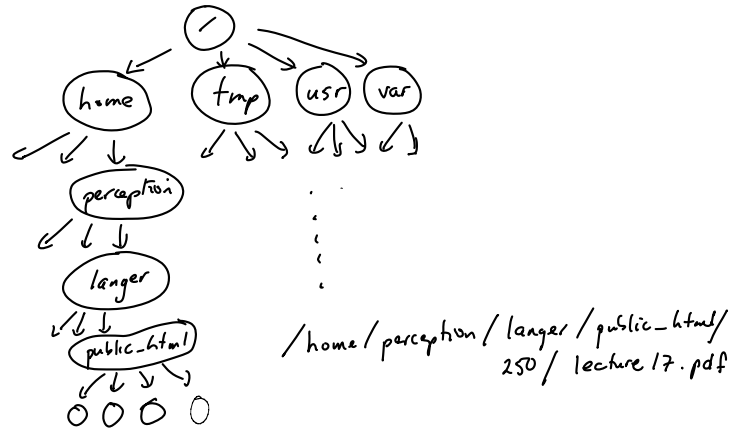


lecture 17

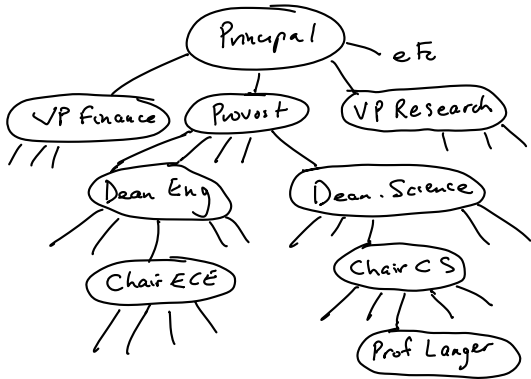
trees

(Rooted) Trees

Example: linux file system

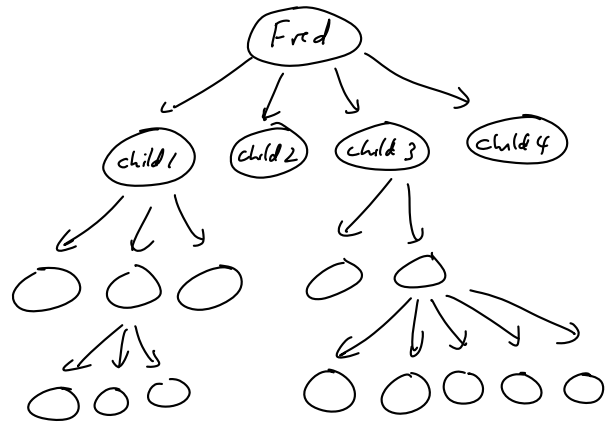


Example: McGill

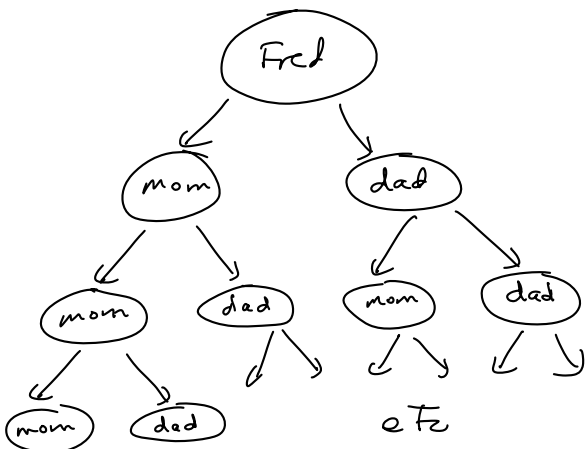


www.mcgill.ca/orgchart

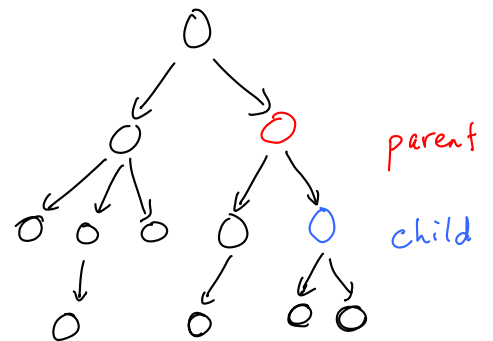
Family Tree



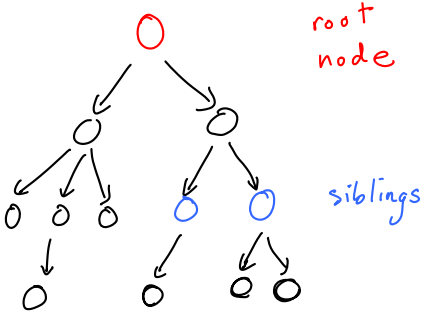
Family Tree (alternative)



Tree Terminology



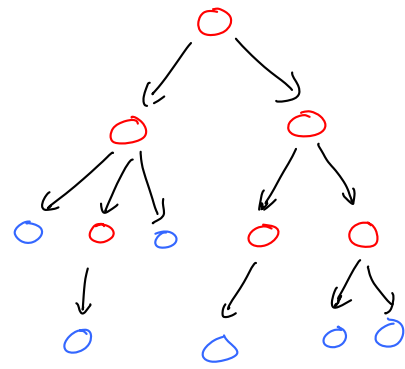
"edge": (parent, child)



root node

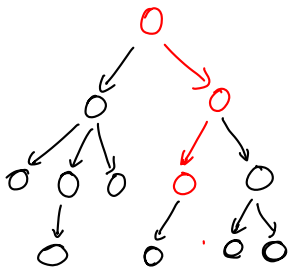
siblings

Every node except for one (the root) has exactly one parent.
Two nodes are siblings if they have the same parent.



internal nodes
e.g. file directories

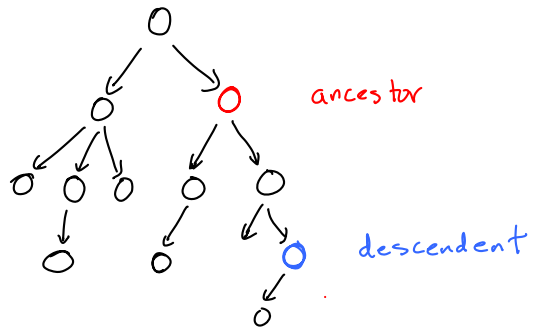
leaf nodes
(external nodes)
e.g. files



path : sequence of nodes $(v_1, v_2, v_3, \dots, v_k)$ such that v_i is parent of v_{i+1}

length of path : number of edges (v_i, v_{i+1}) , i.e. $k-1$

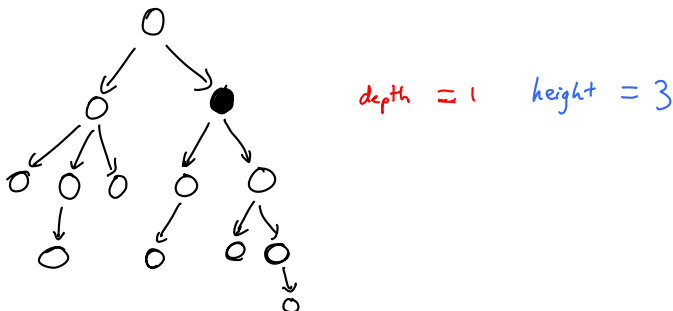
Note: a one node path (v_i) has length 0.



ancestor

descendant

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

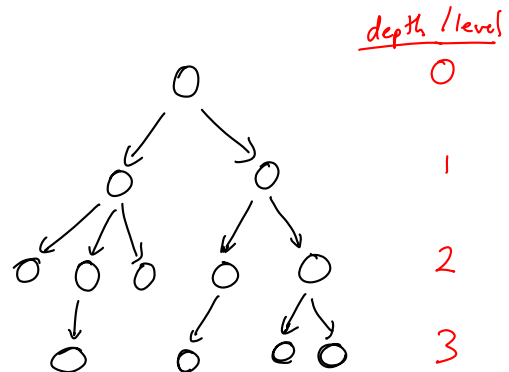


depth = 1 height = 3

depth (level) of a node : length of path from root to the node

height of a node : max length of path from node to a leaf

depth (level) of a node : length of path from root to the node



depth / level

0

1

2

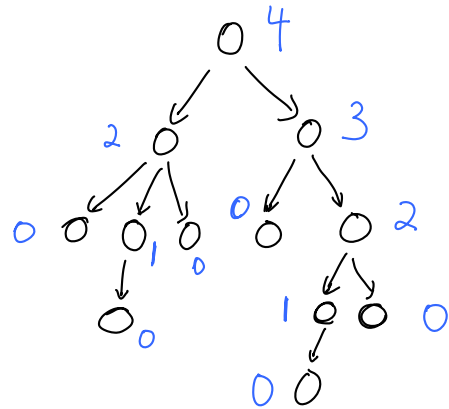
3

```

depth(v) {
  if (v is root)
    return 0
  else
    return 1 + depth(v.parent)
}

```

height of a node :
max length of path from node to a leaf



```

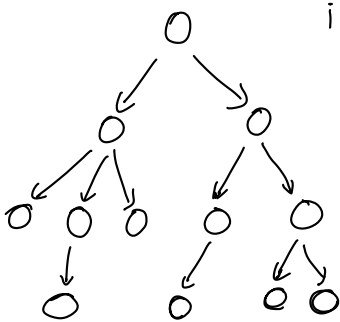
height(v) {
  if v is a leaf
    return 0
  else
    return 1 +
      maximum height
      of v's children
}

```

(Semi) Formal Definition of a (Rooted) Tree

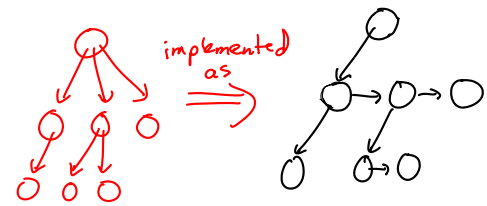
- A tree T is a finite set possibly empty of nodes such that :
- If the set is non-empty, then one of the nodes is the root r .
 - The remaining nodes are partitioned into subsets, each of which is a tree (called a subtree).
 - The roots of these subtrees are the children of node r .

A subtree of a tree T is the tree defined by a node of T and all its descendants.



(There are other definitions of subtree that you will see later.)

Java Implementation



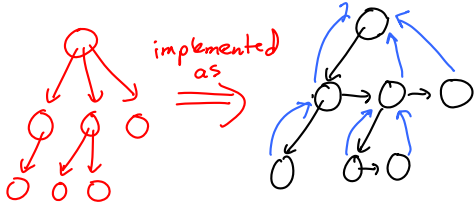
```

class TreeNode<E> {
  E element
  TreeNode<E> firstChild
  TreeNode<E> nextSibling
  ...
}

class Tree<E> {
  TreeNode<E> root
  ...
}

```

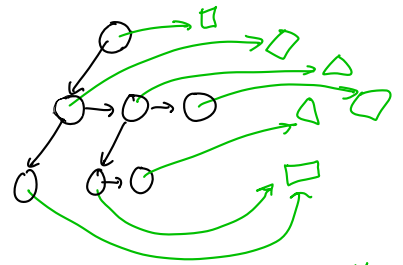
(Java) Implementation



```

class TreeNode <E> {
    E element
    TreeNode <E> parent // optional
    TreeNode <E> firstChild
    TreeNode <E> nextSibling
    ...
}
    
```

Don't forget the object referenced by each node.



```

class TreeNode <E> {
    E element
    TreeNode <E> firstChild
    TreeNode <E> nextSibling
    ...
}
    
```

Nothing in the definition stops two nodes from referencing the same object.

The above implementation is very common. It uses a singly linked list for the children. In Java, though, you might prefer to use a LinkedList or ArrayList for the children to take advantage of these predefined classes. (See lecture notes.)