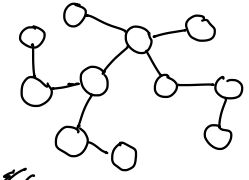
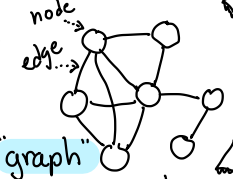
Tree (or "Free Tree")
- undirected
- connected
- acyclic graph



Undirected



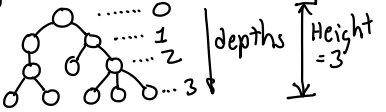
Directed

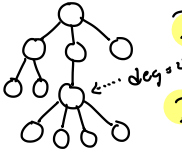
Graph: G = (V, E)
V = finite set of **vertices** (nodes)
E = set of **edges** (pairs of vertices)

Trees: Basic Concepts and Definitions

Depth: path length from root

Height: (of tree) max depth

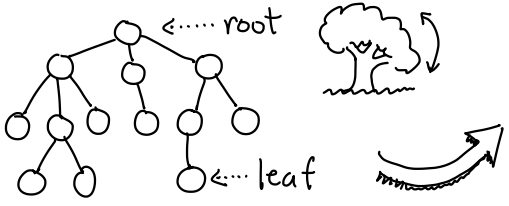




Degree (of node): number of children

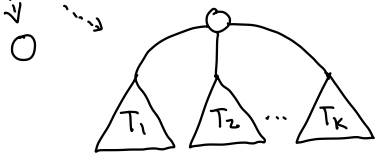
Degree (of tree): max. degree of any node

Rooted tree: A free tree with **root** node

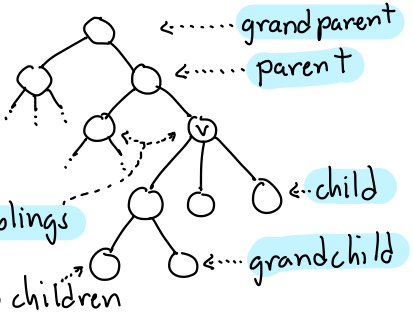


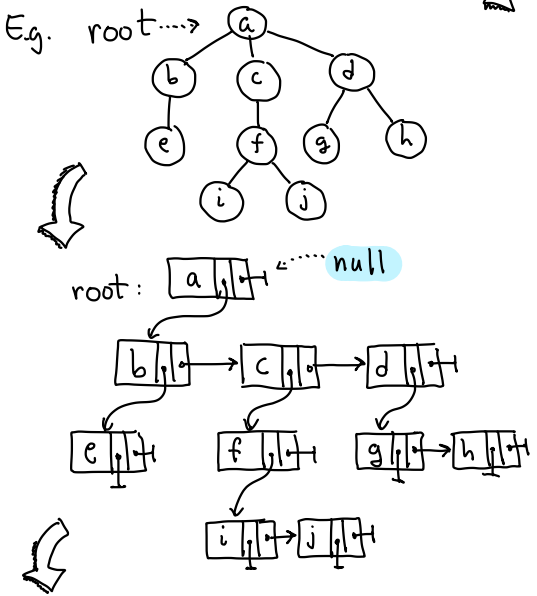
Formal definition:

Rooted tree: is either
- single node (root)
- set of one or more rooted trees ("subtrees") joined to a common root



"Family" Relations



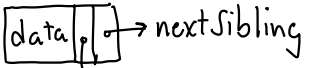


called the **Binary representation**

Binary tree: A rooted tree of degree 2, where each node has two children (possibly null) **left** + **right**

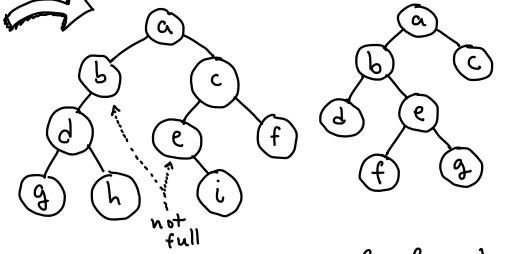
Representing rooted trees:
Each node stores a (linked) list of its children

Node structure:



Trees Representation + Binary Trees (1)

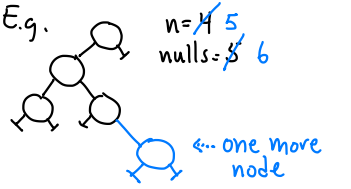
(Not full) Full:



Full: Every non-leaf node has 2 children

Wasted space?

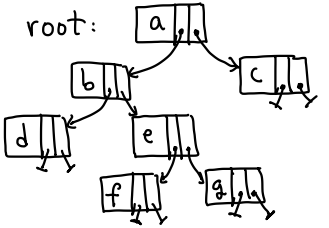
Theorem: A binary tree with n nodes has n+1 null links



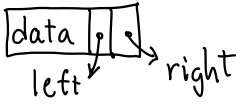
In Java:

```
class BTNode<E> {
    E data;
    BTNode<E> left;
    BTNode<E> right;
    ....
}
```

generic data



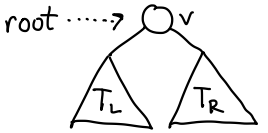
Node structure:



```
traverse(BTNode v) {
  if (v == null) return;
  visit/process v   <- Preorder
  traverse(v.left)
  visit/process v   <- Inorder
  traverse(v.right)
  visit/process v   <- Postorder
}
```

Traversals: How to (systematically) visit the nodes of a rooted tree?

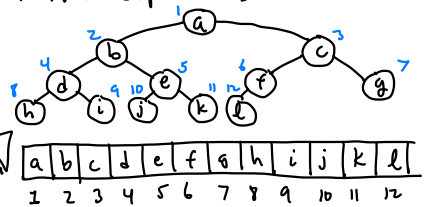
Binary Tree Traversals (can be generalized)



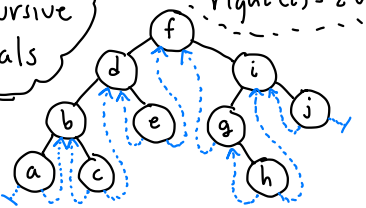
- process/visit v
- traverse T_L
- traverse T_R

} recursive

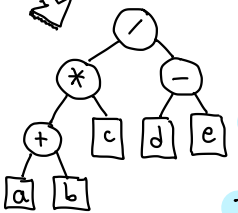
Complete Binary Tree: All levels full (except last)



parent(i) = $\lfloor i/2 \rfloor$
left(i) = 2·i
right(i) = 2·i+1



Challenge: Nonrecursive traversals



Preorder: / * + a b c - d e

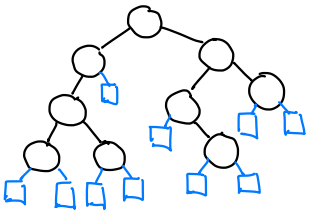
Postorder: a b + c * d e - /

Inorder: a + b * c / d - e

Binary Trees: Traversals, Extension, and More

Those wasteful null links....

Extended binary tree: Replace each null link with a special leaf node: external node



Thm: An extended binary tree with n internal nodes (black) has n+1 external nodes (blue)

Observation: Every extended binary tree is full

Another way to save space...

Threaded binary tree: Store (useful) links in the null links. (Use a mark bit to distinguish link types.)

E.g. **Inorder Threads:**
Null left → inorder predecessor
Null right → " successor