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

**Undirected**
- graph
- \( G = (V, E) \)
- \( V \) = finite set of vertices (nodes)
- \( E \) = set of edges (pairs of vertices)

**Directed**
- digraph
- \( (u, v) \)

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

**Depth**: path length from root
- \( \log_2 \text{ doesn't matter } \Omega(\log n) \)
- \( \log n - \log n \frac{1}{\log n} \)

**Height**: (of tree) max depth
- \( \log n \) (binary)
- \( \log_2 n = \log n \)

**Degree (of node)**: number of children
**Degree (of tree)**: max degree of any node

**Trees: Basic Concepts and Definitions**
- Rooted tree: A free tree with root node
- \( n \) nodes \( n-1 \) edges
- \( (1, 2, 3) \)
- \( \{1, 2, 3\} \)

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

**"Family" Relations**
- grandparent
- parent
- child
- siblings
- cousins
- grandchild

**Tree with \( n \) nodes**
- leaf: no children
Representing rooted trees:
Each node stores a (linked) list of its children

Node structure:

Trees Representation + Binary Trees (1)

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

Full: Every non-leaf node has 2 children

Wasted space?
Theorem: A binary tree with \( n \) nodes has \( n+1 \) null links

E.g.
\( n=15 \Rightarrow n-1 \) non-null links

nulls: \( \geq 6 \)

In Java:
```java
class BTNode<E> {
    E data;
    BTNode<E> left;
    BTNode<E> right;
    ...
}
```

Full:
Every non-leaf node has 2 children

Vanilla
```java
           data
            ↑
            ↑ size
data
left
right
level
```
traverse (BTNode v) {
  if (v == null) return;
  visit/process v ← Preorder
  traverse (v.left) ← Inorder
  traverse (v.right) ← Postorder
  visit/process v
}

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

Binary Tree Traversals (can be generalized)

Complete Binary Tree: All levels full (except last)

Inorder: successor - right
Preorder: parent(i) = ⌊i/2⌋
Postorder: predecessor - left
null left → inorder predecessor

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

Observation: Every extended binary tree is full

Proof: ??

Those wasteful null links....

Binary Trees: Traversals, Extension, and More

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

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

Inorder Threads:
Null left → inorder predecessor
Null right → "successor
Q: Perform a postorder traversal of this tree, given its first-child, next-sibling representation.

Postorder: e b i j f c...

First-child: null

Next-sibling: right
In order traversal: e b i j f c...