Trees
Definition (Tree)

A tree $T$ is a set of nodes storing elements in a parent-child relationship s.t.,

- $T$ has a special node $r$, called the root of $T$.
- Each node $v$ of $T$ different from $r$ has a parent node $u$. 

Trees: formal definition
Definition (Tree)
A tree $T$ is a set of nodes storing elements in a parent-child relationship s.t.,

- $T$ has a special node $r$, called the root of $T$.
- Each node $v$ of $T$ different from $r$ has a parent node $u$.
- $u$ the parent of $v \Rightarrow v$ the child of $u$. 

Trees: formal definition
Definition (Tree)

A tree $T$ is a set of nodes storing elements in a parent-child relationship s.t.,

- $T$ has a special node $r$, called the root of $T$.
- Each node $v$ of $T$ different from $r$ has a parent node $u$.
- $u$ the parent of $v$ $\Rightarrow$ $v$ the child of $u$.
- Two children of the same parent are siblings. Ordered if there is an order among siblings.
Definition (Tree)

A tree $T$ is a set of nodes storing elements in a parent-child relationship s.t.,

- $T$ has a special node $r$, called the root of $T$.
- Each node $v$ of $T$ different from $r$ has a parent node $u$.
- $u$ the parent of $v$ $\Rightarrow$ $v$ the child of $u$.
- Two children of the same parent are siblings. Ordered if there is an order among siblings.
- A node is external is no child, also known as leaves. Otherwise, it is internal.
Trees: formal definition

Definition (Tree)

A tree $T$ is a set of nodes storing elements in a parent-child relationship s.t.,

- $T$ has a special node $r$, called the root of $T$.
- Each node $v$ of $T$ different from $r$ has a parent node $u$.
- $u$ the parent of $v$ $\Rightarrow$ $v$ the child of $u$.
- Two children of the same parent are siblings. Ordered if there is an order among siblings.
- A node is external is no child, also known as leaves. Otherwise, it is internal.
- An ancestor of a node is either the node itself or an ancestor of the parent of the node. Conversely, a descendant.
Trees

```
  33
 /   \
15    47
|      |
10    38
|    |  |
5   12 18
          |
           36
           |
           39
           |
           49
```
The \textbf{depth} of $v$ is the number of ancestors of $v$, excluding $v$ itself. The root has depth 0. Or, equivalently,

- If $v$ is the root, then depth of $v$ is 0.
- Otherwise, the depth of $v = \text{depth of } v\text{'s parent} + 1$. 

Trees

The **depth** of \( v \) is the number of ancestors of \( v \), excluding \( v \) itself. The root has depth 0. Or, equivalently,

- If \( v \) is the root, then depth of \( v \) is 0.
- Otherwise, \( \text{depth of } v = \text{depth of } v\text{'s parent } + 1 \).

The **height** of a tree \( T \) is the maximum of the depth of external nodes of \( T \). Or, equivalently, define the height of a node \( v \) as

- 0 if \( v \) is an external node.
- \( 1 + \max\{ \text{height of a child of } v \} \) otherwise.
The **depth** of $v$ is the number of ancestors of $v$, excluding $v$ itself. The root has depth 0. Or, equivalently,

- If $v$ is the root, then depth of $v$ is 0.
- Otherwise, the depth of $v = \text{depth of } v$’s parent +1.

The **height** of a tree $T$ is the maximum of the depth of external nodes of $T$. Or, equivalently, define the height of a node $v$ as

- 0 if $v$ is an external node.
- $1 + \max\{ \text{height of a child of } v \}$ otherwise.

The **height** of a tree is the height of the root of $T$. 
ADT: Trees

Accessor Methods

- `root()`: return the root of the tree.
- `parent(v)`: return the parent of `v`; error if `v` is the root.
- `child(v)`: return an iterator of the children of `v`.

Query & Generic Methods

- `isExternal()`, `isInternal()`, `isRoot()`;
- `size()`;
- `elements()`;
ADT: Trees

Accessor Methods

▶ root(): return the root of the tree.
▶ parent(v): return the parent of v; error if v is the root.
▶ child(v): return an iterator of the children of v.

Query & Generic Methods

▶ isExternal(), isInternal(), isRoot();
▶ size();
▶ elements();
Return the depth of $v$ in $T$

**Algorithm** depth($T$, $v$)

if $T$.isRoot($v$) then
    return 0;
else
    return 1 + depth($T$, $T$.parent($v$));
end if
Return the depth of $v$ in $T$

**Algorithm** `depth(T, v)`
- **if** `T.isRoot(v)` **then**
  - **return** 0;
- **else**
  - **return** 1 + `depth(T, T.parent(v));`
- **end if**

**Complexity**

$O(n)$: $n$ is $\#$ nodes in $T$. What is the worst case?
Height

Return the height of \( v \) in \( T \)

**Algorithm** \( \text{height}(T, v) \)

if \( T.\text{isExternal}(v) \) then

    return 0;

else

    \( h \leftarrow 0 \)
    for each \( w \in T.\text{children}(v) \) do
        \( h \leftarrow \max(h, \text{height}(T, w)) \)
    end for
    return 1 + \( h \);

end if

Complexity

The height of \( T \) is then \( \text{height}(T, T.\text{root}) \). The complexity is \( O(n) \)!
Return the height of \( v \) in \( T \)

**Algorithm** `height(T, v)`

- if \( T.\text{isExternal}(v) \) then
  - return 0;
- else
  - \( h \leftarrow 0 \)
  - for each \( w \in T.\text{children}(v) \) do
    - \( h \leftarrow \max(h, \text{height}(T, w)) \)
  - end for
  - return \( 1 + h \);
- end if

**Complexity**

The height of \( T \) is then \( \text{height}(T, T.\text{root}()) \). The complexity is \( O(n) \)!
Property about Trees

**Theorem**

Let $T$ be a tree with $n$ nodes, $c_v$ the number of children of node $v$.

$$\sum_{v \in T} c_v = n - 1.$$
Property about Trees

Theorem
Let \( T \) be a tree with \( n \) nodes, \( c_v \) the number of children of node \( v \).

\[
\sum_{v \in T} c_v = n - 1.
\]

Proof.
Counting from another perspective: each node (except the root) is counted only once from its unique parent. \( \square \)
A **traversal** of a tree \( T \) is a systematical way of "visiting" all nodes in \( T \).
Traversals of Trees

A **traversal** of a tree $T$ is a systematical way of "visiting" all nodes in $T$.

- **Pre-order**: Root first and then visit each sub-tree in order.
- **Post-order**: Visit each sub-tree first and then the root.
Traversals of Trees

**Algorithm** preorder($T, v$)

"visit" the node $v$

for each child $w$ of $v$ do

preorder($T, w$)

end for

**Algorithm** postorder($T, v$)

for each child $w$ of $v$ do

post-order($T, w$)

end for

"visit" the node $v$
Traversal of Trees

**Algorithm** preorder($T, v$)

"visit" the node $v$

for each child $w$ of $v$ do

    preorder($T, w$)

end for

**Algorithm** postorder($T, v$)

for each child $w$ of $v$ do

    post-order($T, w$)

end for

"visit" the node $v$

**Complexity**

$O(n)$: similar counting as the analysis in height().
Trees: Pre-order

33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
Trees: Pre-order

Pre-order: 33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
Trees: Post-order

Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33
Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33