Lecture 01/27/17

Lecturer: Xiaodi Wu

Reading Assignment: Chapter 10.4, 12.1-3
Trees

```
33
 /   \
15   47
 /     \
10    38
 |     /\n5  12  36
|    |  |    |
18  20 39  49
```

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 $u$.
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 $v \implies v$ the child of $u$. 

- Two children of the same parent are siblings.
- Ordered if there is an order among siblings.
- A node is external if 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 descendent.
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.
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
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 $v = \text{depth of } v\text{'s parent } + 1.$
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 $v = \text{depth of } v\text{'s parent} + 1$.

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.
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, the depth of \( v \) = 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 \( T \).
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.
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** \( \text{depth}(T, v) \)

if \( T.\text{isRoot}(v) \) then
    return 0;
else
    return \( 1 + \text{depth}(T, T.\text{parent}(v)) \);
end if

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

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

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

\[
\begin{align*}
\text{if} & \; \text{T.isRoot}(v) \; \text{then} \\
& \; \text{return} \; 0; \\
\text{else} & \\
& \; \text{return} \; 1 + \text{depth}(T, \text{T.parent}(v)); \\
\text{end if}
\end{align*}
\]

**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)`

```plaintext
if T.isExternal(v) then
  return 0;
else
  $h \leftarrow 0$
  for each $w \in T.children(v)$ do
    $h \leftarrow \max(h, height(T, w))$
  end for
  return 1 + $h$;
end if
```

**Complexity**

The height of $T$ is then `height(T, T.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 $v$. Then

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

Proof.

Counting from another perspective: each node (except the root) is counted only once from its unique parent. \qed
Implementation of Binary Trees

Vector-based Structure

- $p(v)$: the rank of $v$ stored in array $A$ of size $N$.
- If $v$ is the root, then $p(v) = 1$.
- If $v$ is the left child of $u$, then $p(v) = 2p(u)$.
- If $v$ is the right child of $u$, then $p(v) = 2p(u) + 1$. 
Implementation of Binary Trees

Vector-based Structure

- \( p(v) \): the rank of \( v \) stored in array \( A \) of size \( N \).
- If \( v \) is the root, then \( p(v) = 1 \).
- If \( v \) is the left child of \( u \), then \( p(v) = 2p(u) \).
- If \( v \) is the right child of \( u \), then \( p(v) = 2p(u) + 1 \).

- methods: leftChild(), rightChild(), root(), parent(), children(), \( O(1) \) time.
- Space could be as large as \( O(2^{(n+1)/2}) \).
Implementation of Binary Trees

Linked Structure: similar to doubly linked list

- Each node: pointers to parent, leftChild, rightChild, and the element stored.
- methods: leftChild(), rightChild(), root(), parent(), children(), $O(1)$ time.
- Space usage $O(n)$. 