Trees

CMSC 420: Lecture 5
Hierarchies

Many ways to represent tree-like information:

1. A
   1. B
      a. D
      i. E
   2. C
      a. G


nested, labeled parenthesis

linked hierarchy

outlines, indentations

nested sets
Definition – Rooted Tree

- \( \Lambda \) is a tree
- If \( T_1, T_2, \ldots, T_k \) are trees with roots \( r_1, r_2, \ldots, r_k \) and \( r \) is a node not in any \( T_i \), then the structure that consists of the \( T_i \), node \( r \), and edges \((r, r_i)\) is also a tree.
**Terminology**

- $r$ is the *parent* of its *children* $r_1, r_2, ..., r_k$.

- $r_1, r_2, ..., r_k$ are *siblings*.

- *root* = distinguished node, usually drawn at top. Has no parent.

- If all children of a node are $\Lambda$, the node is a *leaf*. Otherwise, the node is a *internal node*.

- A *path* in the tree is a sequence of nodes $u_1, u_2, ..., u_m$ such that each of the edges $(u, u_{i+1})$ exists.

- A node $u$ is an *ancestor* of $v$ if there is a path from $u$ to $v$.

- A node $u$ is a *descendant* of $v$ if there is a path from $v$ to $u$. 
Height & Depth

- The **height** of node $u$ is the length of the longest path from $u$ to a leaf.

- The **depth** of node $u$ is the length of the path from the root to $u$.

- Height of the tree = maximum depth of its nodes.

- A **level** is the set of all nodes at the same depth.
Subtrees, forests, and graphs

- A *subtree* rooted at \( u \) is the tree formed from \( u \) and all its descendants.

- A *forest* is a (possibly empty) set of trees. The set of subtrees rooted at the children of \( r \) form a forest.

- As we’ve defined them, trees are **not** a special case of graphs:
  - Our trees are *oriented* (there is a root which implicitly defines directions on the edges).
  - A *free tree* is a connected graph with no cycles.
Alternative Definition – Rooted Tree

- A tree is a finite set $T$ such that:
  - one element $r \in T$ is designated the root.
  - the remaining nodes are partitioned into $k \geq 0$ disjoint sets $T_1, T_2, \ldots, T_k$, each of which is a tree.

This definition emphasizes the *partitioning* aspect of trees:

As we move down the we’re dividing the set of elements into more and more parts.

Each part has a distinguished element (that can represent it).
Basic Properties

• Every node except the root has exactly one parent.

• A tree with $n$ nodes has $n-1$ edges (every node except the root has an edge to its parent).

• There is exactly one path from the root to each node. (Suppose there were 2 paths, then some node along the 2 paths would have 2 parents.)
Binary Trees – Definition

- An **ordered** tree is a tree for which the order of the children of each node is considered important.

- A **binary tree** is an ordered tree such that each node has ≤ 2 children.

- Call these two children the **left** and **right** children.
Example Binary Trees

The edge cases:

- Only left child
- Only right child
- Single node
- Empty Binary Tree

Small binary tree:
Extended Binary Trees

Every internal node has exactly 2 children.

Every leaf (external node) has exactly 0 children.

Each external node corresponds to one Λ in the original tree – let's us distinguish different instances of Λ.
# of External Nodes in Extended Binary Trees

**Thm.** An extended binary tree with $n$ internal nodes has $n+1$ external nodes.

**Proof.** By induction on $n$.

$X(n) :=$ number of external nodes in binary tree with $n$ internal nodes.

**Base case:** $X(0) = 1 = n + 1$.

**Induction step:** Suppose theorem is true for all $i < n$. Because $n \geq 1$, we have:

$$X(n) = X(k) + X(n-k-1) = k+1 + n-k-1 + 1 = n + 1 \square$$

Related to Thm 5.2 in your book.
**Thm.** An extended binary tree with $n$ internal nodes has $n+1$ external nodes.

**Proof.** Every node has 2 children pointers, for a total of $2n$ pointers.

Every node except the root has a parent, for a total of $n - 1$ nodes with parents.

These $n - 1$ parented nodes are all children, and each takes up 1 child pointer.

\[(\text{pointers}) - (\text{used child pointers}) = (\text{unused child pointers})\]
\[2n - (n-1) = n + 1\]

Thus, there are $n + 1$ null pointers.

Every null pointer corresponds to one external node by construction. □
**Full and Complete Binary Trees**

- If every node has either 0 or 2 children, a binary tree is called *full*.

- If the lowest \(d-1\) levels of a binary tree of height \(d\) are filled and level \(d\) is partially filled from left to right, the tree is called *complete*.

- If all \(d\) levels of a height-\(d\) binary tree are filled, the tree is called *perfect*.
# Nodes in a Perfect Tree of Height \( h \)

**Thm.** A perfect tree of height \( h \) has \( 2^{h+1} - 1 \) nodes.

**Proof.** By induction on \( h \).

Let \( N(h) \) be number of nodes in a perfect tree of height \( h \).

**Base case:** when \( h = 0 \), tree is a single node. \( N(0) = 1 = 2^{0+1} - 1 \).

**Induction step:** Assume \( N(i) = 2^{i+1} - 1 \) for \( 0 \leq i < h \).

A perfect binary tree of height \( h \) consists of 2 perfect binary trees of height \( h-1 \) plus the root:

\[
N(h) = 2 \times N(h - 1) + 1
= 2 \times (2^{h-1+1} - 1) + 1
= 2 \times 2^h - 2 + 1
= 2^{h+1} - 1 \quad \square
\]

\( 2^h \) are leaves
\( 2^h - 1 \) are internal nodes
Full Binary Tree Theorem

Thm. In a non-empty, full binary tree, the number of internal nodes is always 1 less than the number of leaves.

Proof. By induction on $n$.

L($n$) := number of leaves in a non-empty, full tree of $n$ internal nodes.

Base case: L(0) = 1 = $n + 1$.

Induction step: Assume L($i$) = $i + 1$ for $i < n$.

Given T with $n$ internal nodes, remove two sibling leaves.

T' has $n-1$ internal nodes, and by induction hypothesis, L($n-1$) = $n$ leaves.

Replace removed leaves to return to tree T.

Turns a leaf into an internal node, adds two new leaves.

Thus: L($n$) = $n + 2 - 1 = n + 1$. 

Thm 5.1 in your book.
Array Implementation for Complete Binary Trees

Mapping of nodes to integers

left(i): $2i$ if $2i \leq n$ otherwise 0
right(i): $(2i + 1)$ if $2i + 1 \leq n$ otherwise 0
parent(i): $\lfloor i/2 \rfloor$ if $i \geq 2$ otherwise 0
Binary Tree ADT

A tree can be represented as a linked collection of its nodes:

```cpp
template <class ValType>
class BinaryTree {
  public:

    virtual ValType & value() = 0;
    virtual void set_value(const ValType &) = 0;
    virtual BinaryTree * left() const = 0;
    virtual void set_left(BinNode *) = 0;
    virtual BinaryTree * right() const = 0;
    virtual void set_right(BinNode *) = 0;
    virtual bool is_leaf() = 0;
};
```

virtual ⇒ this function can be overridden by subclassing.
“= 0” ⇒ a pure function with no implementation. Must subclass to get implementation.
Linked Binary Tree Implementation

template <class ValType>
class BinNode : public BinaryTree<ValType>
{
  public:
    BinNode(ValType * v);
    ~BinNode();
    ValType & value();
    void set_value(const ValType&);
    BinNode * left() const;
    void set_left(BinNode *);
    BinNode * right() const;
    void set_right(BinNode *);
    bool is_leaf();

  private:
    ValType * _data;
    BinNode<ValType> * _left_child;
    BinNode<ValType> * _right_child;
};
Binary Tree Representation
List Representation of General Trees
Representing General Trees with Binary Trees

General K-ary Tree

Representation as Binary Tree

Each node represented by:

<table>
<thead>
<tr>
<th>_data</th>
<th>_first_child</th>
<th>_right_sibling</th>
</tr>
</thead>
</table>

How would you implement an ordered general tree using a binary tree?