Trees

Russell Impagliazzo and Miles Jones
Thanks to Janine Tiefenbruck

http://cseweb.ucsd.edu/classes/sp16/cse21-bd/
April 29, 2016
Another Special Type of Graph: Trees
1. Definitions of trees

2. Properties of trees

3. Revisiting uses of trees
A **rooted tree** is a connected directed acyclic graph in which one vertex has been designated the root, which has no incoming edges, and every other vertex has exactly one incoming edge.
(Rooted) Trees: definitions

A **rooted tree** is a connected directed acyclic graph in which one vertex has been designated the root, which has no incoming edges, and every other vertex has exactly one incoming edge.

Special case of DAGs from last class.
Note that each vertex in middle has *exactly one* incoming edge from layer above.
Edges are directed *away from* the root.
Which of the following directed graphs are trees (with root indicated in green)?

A.  

B.  

C.  

D.
(Rooted) Trees: definitions

Rosen p. 747-749
If vertex $v$ is not the root, it has exactly one incoming edge, which is from its parent, $p(v)$.

**Height** of vertex $v$ is given by the recurrence:

$$h(v) = h(p(v)) + 1 \quad \text{if } v \text{ is not the root, and}$$

$$h(r) = 0$$
(Rooted) Trees: definitions

**Height** of vertex v: $h(v) = h(p(v)) + 1$ if v is not the root, and $h(r) = 0$

What is the height of the red vertex?
A. 0  
B. 1  
C. 2  
D. 3  
E. None of the above.
**Height** of vertex v: $h(v) = h(p(v)) + 1$ if v is not the root, and

$h(r) = 0$

What is the height of the tree?

A. 0
B. 1
C. 2
D. 3
E. None of the above.

*Rosen p. 753*
A **binary tree** is a rooted tree where every (internal) vertex has no more than 2 children.

How many leaves does a binary tree of height 3 have?

A. 2  
B. 3  
C. 6  
D. 8  
E. any of the above.
A **binary tree** is a rooted tree where every (internal) vertex has no more than 2 children.

How many leaves does a binary tree of height 3 have?

A. 2  
B. 3  
C. 6  
D. 8  
E. any of the above.

*See Theorem 5 for proof of upper bound*
A full binary tree is a rooted tree where every internal vertex has exactly 2 children.

Which of the following are full binary trees?

A.  

B.  

C.  

D.  

Rosen p. 749
A full binary tree is a rooted tree where every internal vertex has exactly 2 children.

At most how many vertices are there in a full binary tree of height $h$?

A. $\Theta(h)$

B. $\Theta(2^h)$

C. $\Theta(h^2)$

D. $\Theta(\log h)$

Max number of vertices when tree is balanced
A **full** binary tree is a rooted tree where every internal vertex has exactly 2 children.

*Key insight: number of vertices doubles on each level.*

\[
1 + 2 + 4 + 8 + \ldots + 2^h = 2^{h+1} - 1 \quad \text{i.e.} \quad \Theta(2^h)
\]

If \( n \) is number of vertices:

\[
n = 2^{h+1} - 1
\]

so

\[
h = \log(n+1) - 1 \quad \text{i.e.} \quad \Theta(\log n)
\]
Binary tree

Relating height and number of vertices:

$\log(n+1) - 1 \leq h \leq ___$

This is what we just proved.

How do we prove?

What tree with $n$ vertices has the greatest possible height?
Relating height and number of vertices:

\[ \log(n+1) - 1 \leq h \leq n-1 \]

This is what we just proved.

How do we prove?

What tree with \( n \) vertices has the greatest possible height?
Trees

1. Definitions of trees

2. Properties of trees

3. Revisiting uses of trees

In data structures:
Max heap
Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

**Implementation**

Each vertex is an object with the fields

- \( p = \text{parent} \)
- \( lc = \text{left child} \)
- \( rc = \text{right child} \)
- \( \text{value} \)

When is \( p \) null?

A. If we have an error in our implementation.
B. When the value is 0.
C. When the vertex is a leaf node.
D. When the vertex is the root node.
E. None of the above.
Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

Implementation

Each vertex is an object with the fields

\[ p = \text{parent} \]
\[ lc = \text{left child} \]
\[ rc = \text{right child} \]
\[ \text{value} \]

Under which of these conditions is \( lc \) always null?

A. If we have an error in our implementation.
B. When the value is 0.
C. When the vertex is a leaf node.
D. When the vertex is the root node.
E. None of the above.
Binary Search Trees

• Facilitate binary search (must maintain sorted order of data)
• Dynamic

For each vertex \( v \)
• If \( x \) is in subtree rooted at \( l_c(v) \),
  \( \text{value}(x) \leq \text{value}(v) \).
• If \( x \) is in the subtree rooted at \( r_c(v) \),
  \( \text{value}(x) \geq \text{value}(v) \).
Binary Search Trees

- Facilitate binary search (must **maintain sorted order** of data)
- Dynamic

How would you search for "orange?"
Binary Search Trees

• Facilitate binary search (must **maintain sorted order** of data)
• Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done 😊.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).
Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done 😊.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

**How long does this take?**
Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done 😊.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

How long does this take?  Constant time at each level, number of levels is height+1.
Binary Search Trees

- Facilitate binary search (must maintain sorted order of data)
- Dynamic

To search for target T in a binary search tree.

1. Compare T to value(r) where r is the root.
2. If T = value(r), done 😊.
3. If T < value(r), search recursively starting at lc(r).
4. If T > value(r), search recursively starting at rc(r).

How long does this take? Time proportional to height!
An unrooted tree is a connected undirected graph with no cycles.
Theorem: An undirected graph is an unrooted tree if and only if it contains all the edges of some rooted tree.

What does this mean?

(1) If we replace all directed edges in a rooted tree with undirected edges, the result will be an unrooted tree.

(2) There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.
Equivalence between rooted and unrooted trees

**Goal (1):** If we replace all directed edges in a rooted tree with undirected edges, the result will be an **unrooted tree**.

*What do we need to prove?*

A. The resulting undirected graph will be connected.
B. The resulting undirected graph will be undirected.
C. The resulting undirected graph will not have cycles.
D. All of the above.
Equivalence between rooted and unrooted trees

**Goal (1):** If we replace all directed edges in a rooted tree with undirected edges, the result will be an **unrooted tree**.

**SubGoal (1a):** this resulting graph is connected, i.e. between any two vertices $u$ and $v$ there is a path in the graph.

Idea: To find path between purple and orange, follow parents of purple all the way to root, then follow its children down to orange.
Equivalence between rooted and unrooted trees

**Goal (1):** If we replace all directed edges in a rooted tree with undirected edges, the result will be an **unrooted tree**.

**SubGoal (1b):** this resulting graph has no cycles.
Assume by contradiction that there exists a cycle and let $v$ be the vertex with the deepest height. Since $v$ is in a cycle, there are two edges in the cycle incident with $v$.

There is one edge from the parent. All other edges go to deeper vertices contradicting that $v$ is the deepest.
Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

Idea: finding right directions for edges will be similar to finding topological sort last class.
Equivalence between rooted and unrooted trees

**Goal (2):** There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

**SubGoal (2a):** Any unrooted tree with at least two vertices has a vertex of degree exactly 1.

**Proof:** Towards a contradiction, assume that all vertices have degree 0 or >=2. Since a tree is connected, eliminate the case of degree-0 vertices. 
**Goal:** construct a cycle to arrive at a contradiction.

Start at any vertex $u_0$. Pick $u_{i+1}$ so that it is adjacent to $u_i$ but is not $u_{i-1}$. Why?

Get $u_0, u_1, …, u_n$. By Pigeonhole Principle, must repeat. **Cycle!**
Equivalence between rooted and unrooted trees

Goal (2): There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

SubGoal (2b): If $T$ is unrooted tree and $v$ has degree 1 in $T$, then $T\setminus\{v\}$ is unrooted tree.

Proof: To check that $T\setminus\{v\}$ is unrooted tree,

* confirm $T\setminus\{v\}$ is connected and

* $T\setminus\{v\}$ does not have a cycle.
Equivalence between rooted and unrooted trees

**Goal (2):** There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

**SubGoal (2b):** If $T$ is unrooted tree and $v$ has degree 1 in $T$, then $T-\{v\}$ is unrooted tree.

**Proof:** To check that $T-\{v\}$ is unrooted tree,

* confirm $T-\{v\}$ is connected and

* $T-\{v\}$ does not have a cycle.
Equivalence between rooted and unrooted trees

**Goal (2):** There is always some way to put directions on the edges of an unrooted tree to make it a rooted tree.

Using the subgoals to achieve the goal:

Root($T$: unrooted tree with $n$ nodes)
1. If $n=1$, let the only vertex $v$ be the root, set $h(v):=0$, and return.
2. Find a vertex $v$ of degree 1 in $T$, and let $u$ be its only neighbor.
3. Root($T$-$\{v\}$).
4. Set $p(v):=u$ and $h(v):=h(u)+1$. 

Recursion!
Example

```
    A
   / \   
  C   B 
   \ /  
    D   
   / 
  E   F
```

<table>
<thead>
<tr>
<th>vertex</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>