Trees

<table>
<thead>
<tr>
<th>Lecture A</th>
<th>Tiefenbruck</th>
<th>Tu, Th 11am-12:20pm</th>
<th>Center 119</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture B</td>
<td>Tiefenbruck</td>
<td>Tu, Th 9:30am-10:50am</td>
<td>Center 119</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Minnes</td>
<td>Tu, Th 3:30pm-4:50pm</td>
<td>WLH 2005</td>
</tr>
</tbody>
</table>

http://cseweb.ucsd.edu/classes/fa15/cse21-abc/

Oct 29, 2015
Today's plan

1. Feedback – thank you!
2. Recap
3. Trees: applications, definitions, algorithms.

*In the textbook: Sections 11.1 and 11.2*
Feedback

89 responses so far … keep 'em coming 😊

70% at the right speed; 26% too quickly; 4% too slowly
60% just right details; 27% too little; 13% too much
74% material at right level; 21% too advanced; 4% too elementary

Keep: clickers; discussion quizzes; review sessions before midterms

Change: homework is hard; where can I find material in the textbook?
Recap

Graphs

Vertices and Edges (directed or undirected) …. lots of vocabulary

Some key facts:
- Graph representations: circles and arcs, adjacency matrix, adjacency list.
- Handshaking Theorem (Theorem 1, page 653).
- A connected graph with at least two vertices has an Eulerian tour if and only if it has either 0 or 2 vertices with odd degree (Theorems 1-2, pages 696-697).
- Every DAG has a topological ordering.
Today: Trees

Program

int id(main) { code
  instruction code
    id(cout) ...
  instruction
    return 0 ;
Today: 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.
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

Root

Leaf

Internal vertices

Leaf
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:

$$h(v) = h(p(v)) + 1 \quad \text{if } v \text{ is not the root, and} \quad h(r) = 0 \quad \text{where } r \text{ is the root}$$
**Height** of vertex $v$: 
$h(v) = h(p(v)) + 1$ if $v$ is not the root, and 
$h(r) = 0$ where $r$ is the root.

What is the height of the red vertex? 
A. 1  
B. 2  
C. 3  
D. 4  
E. None of the above.
**Height** of vertex \( v \): 

\[ h(v) = h(p(v)) + 1 \quad \text{if } v \text{ is not the root, and} \quad h(r) = 0 \quad \text{where } r \text{ is the root} \]

**Height** of tree is maximum height of a vertex in the tree.

*Rosen p. 753*

---

**What is the height of the tree?**

- A. 1
- B. 2
- C. 3
- D. 4
- E. None 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. None 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. None 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.
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

\[
h = \log(n+1) - 1 \quad \text{i.e.} \quad \Theta(\log n)
\]
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?
Today: Trees

1. Definitions of trees ✓
2. Properties of trees ✓
3. Revisiting uses of trees

In data structures:
Binary search trees
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

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

When is \textit{lc} \textbf{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 $lc(v)$, $value(x) \leq value(v)$.
- If $x$ is in the subtree rooted at $rc(v)$, $value(x) \geq value(v)$. 
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 $T = \text{value}(r)$, done 😊.
3. If $T < \text{value}(r)$, search recursively starting at lc(r).
4. If $T > \text{value}(r)$, 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.
Equivalence between rooted and unrooted trees

**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.
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.

Idea: Towards a contradiction, assume there is a cycle and consider the simplest cycle (with no repeated vertices). Start at vertex at highest level in the cycle. Next step must go to a child node, etc. Can never go up to higher level again because vertices in rooted tree only have one incoming edge.
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 \( \geq 2 \). 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 \( u_0, u_1, \ldots, u_n \). By Pigeonhole Principle, must repeat. **Cycle!**
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 an unrooted tree and $v$ has degree 1 in $T$, then $T\setminus\{v\}$ is an unrooted tree.

Proof: To check that $T\setminus\{v\}$ is an 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!**
Reminders

HW 5 due **Friday 11:59pm via Gradescope**.

Please fill out **Feedback Form**. Link is on Piazza.