Announcements

Submit P1 Conflict quiz on CMS by end of day Wednesday. We won’t be sending confirmations; no news is good news. Extra time people will eventually get an email from Lacy. Please be patient.
Today’s Topics in JavaHyperText

- Search for “trees”
- Read PDFs for points 0 through 5: intro to trees, examples of trees, binary trees, binary search trees, balanced trees
Data Structures

- **Data structure**
  - Organization or format for storing or managing data
  - Concrete realization of an abstract data type

- **Operations**
  - Always a tradeoff: some operations more efficient, some less, for any data structure
  - Choose efficient data structure for operations of concern
## Example Data Structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>add(val v)</th>
<th>get(int i)</th>
<th>contains(val v)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array</strong></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><img src="chart" alt="Array Diagram" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Linked List</strong></td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><img src="chart" alt="Linked List Diagram" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**add(v):** append v

**get(i):** return element at position i

**contains(v):** return true if contains v

![Array Diagram](chart)

![Linked List Diagram](chart)
Tree

Singly linked list:

Node
object

int value

pointer

Today: trees!
In CS, we draw trees “upside down”
Tree Overview

**Tree:** data structure with nodes, similar to linked list

- Each node may have zero or more successors (children)
- Each node has exactly one predecessor (parent) except the root, which has none
- All nodes are reachable from root

A tree or not a tree?

A tree

Not a tree

A tree

Not a tree

A tree
Tree Terminology (1)

the **root** of the tree (no parents)

child of M

the leaves of the tree (no children)
Tree Terminology (2)

ancestors of B

descendants of W
Tree Terminology (3)

subtree of $M$
A node’s **depth** is the length of the path to the root.

A tree’s (or subtree’s) **height** is the length of the longest path from the root to a leaf.
Tree Terminology (5)

Multiple trees: a forest
General vs. Binary Trees

**General tree**: every node can have an arbitrary number of children

**Binary tree**: at most two children, called *left* and *right*

...often “tree” means binary tree
Binary trees were in A1!

You have seen a binary tree in A1.
A PhD object has one or two advisors.
(Note: the advisors are the “children”.)

![Binary tree diagram]

- David Gries
- Friedrich Bauer
  - Fritz Bopp
    - Fritz Sauter
    - Erwin Fues
  - Georg Aumann
    - Heinrich Tietze
    - Constantin Carathodory
Special kinds of binary trees

Max # of nodes at depth $d$: $2^d$

If height of tree is $h$:
  - min # of nodes: $h + 1$
  - max # of nodes: (Perfect tree) $2^0 + \ldots + 2^h = 2^{h+1} - 1$

Complete binary tree

Every level, except last, is completely filled, nodes on bottom level as far left as possible. No holes.
Trees are recursive

a binary tree
Trees are recursive
Trees are recursive
Trees are recursive

Binary Tree

Left subtree, which is also a binary tree

Right subtree (also a binary tree)
A binary tree is either null

or an object consisting of a value, a left binary tree, and a right binary tree.
A Recipe for Recursive Functions

Base case:
If the input is “easy,” just solve the problem directly.

Recursive case:
Get a smaller part of the input (or several parts).
Call the function on the smaller value(s).
Use the recursive result to build a solution for the full input.
A Recipe for Recursive Functions on Binary Trees

Base case: an empty tree (null), or possibly a leaf
If the input is “easy,” just solve the problem directly.

Recursive case:
Get a smaller part of the input (or several parts).
Call the function on the smaller value(s) in each subtree.
Use the recursive result to build a solution for the full input.
### Comparing Searches

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>( \text{add(val v)} )</th>
<th>( \text{get(int i)} )</th>
<th>( \text{contains(val v)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>( O(n) )</td>
<td>( O(1) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td><img src="example.png" alt="Array Diagram" /></td>
<td>2 1 3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linked List</td>
<td>( O(1) )</td>
<td>( O(n) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td><img src="example.png" alt="Linked List Diagram" /></td>
<td>2 1 3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Tree</td>
<td></td>
<td></td>
<td>( O(n) )</td>
</tr>
<tr>
<td><img src="example.png" alt="Binary Tree Diagram" /></td>
<td>2 1 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node could be *anywhere* in tree

Binary search on arrays: \( O(\log n) \)
Requires invariant: array sorted

...analogue for trees?
A binary search tree is a binary tree with a **class invariant**:

- All nodes in the **left** subtree have values that are **less** than the value in that node, and
- All values in the **right** subtree are **greater**.

(assume no duplicates)
Binary Search Tree (BST)

Contains:

- Binary tree: two recursive calls: $O(n)$
- BST: one recursive call: $O(\text{height})$
BST Insert

To insert a value:

- Search for value
- If not found, put in tree where search ends

**Example:** Insert month names in chronological order as Strings, (Jan, Feb...). BST orders Strings alphabetically (Feb comes before Jan, etc.)
BST Insert

insert: January
BST Insert

insert: February

January
BST Insert

insert: March

January

February
BST Insert

insert: April…

January

February  March
BST Insert

January

February

March

April

June

May

August

July

September

December

October

November
# Comparing Data Structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>add(val x)</th>
<th>get(int i)</th>
<th>contains(val x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><img src="image" alt="Array Diagram" /></td>
<td><img src="image" alt="Array Diagram" /></td>
<td><img src="image" alt="Array Diagram" /></td>
<td><img src="image" alt="Array Diagram" /></td>
</tr>
<tr>
<td>Linked List</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><img src="image" alt="Linked List Diagram" /></td>
<td><img src="image" alt="Linked List Diagram" /></td>
<td><img src="image" alt="Linked List Diagram" /></td>
<td><img src="image" alt="Linked List Diagram" /></td>
</tr>
<tr>
<td>Binary Tree</td>
<td>$O(n)$</td>
<td></td>
<td>$O(n)$</td>
</tr>
<tr>
<td><img src="image" alt="Binary Tree Diagram" /></td>
<td><img src="image" alt="Binary Tree Diagram" /></td>
<td><img src="image" alt="Binary Tree Diagram" /></td>
<td><img src="image" alt="Binary Tree Diagram" /></td>
</tr>
<tr>
<td>BST</td>
<td>$O(\text{height})$</td>
<td>$O(\text{height})$</td>
<td>$O(\text{height})$</td>
</tr>
<tr>
<td><img src="image" alt="BST Diagram" /></td>
<td><img src="image" alt="BST Diagram" /></td>
<td><img src="image" alt="BST Diagram" /></td>
<td><img src="image" alt="BST Diagram" /></td>
</tr>
</tbody>
</table>

How big could height be?
Worst case height

Insert in alphabetical order…

April
Worst case height

Insert in alphabetical order…

April

August
Worst case height

Insert in alphabetical order...

Tree degenerates to list!
Need Balance

- **Takeaway:** BST search is $O(n)$ time
  - Recall, big O notation is for **worst** case running time
  - Worst case for BST is data inserted in sorted order

- **Balanced binary tree:** subtrees of any node are about the same height
  - In balanced BST, search is $O(\log n)$
  - Deletion: tricky! Have to maintain balance
  - [Optional] See JavaHyperText “Extensions to BSTs”
  - Also see CS 3110