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>Array</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Linked List</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

- add(v): append v
- get(i): return element at position i
- contains(v): return true if contains v

---

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

---

Trees

In CS, we draw trees “upside down”
Tree Terminology (1)
- the root of the tree (no parents)
- child of M
- 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

Tree Terminology (4)
- 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”.)

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

Left subtree, which is also a binary tree

Right subtree (also a binary tree)

Binary Tree

Value

Left subtree

Right subtree

Value

Left subtree

Right subtree
Trees are recursive

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) of each subtree.
Use the recursive result to build a solution for the full input.

A Binary Search Tree (BST)

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]

Contains:
- Binary tree: two recursive calls: O(n)
- BST: one recursive call: O(depth)
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

BST Insert

insert: March

BST Insert

insert: April…
### Printing contents of BST

```java
/** Print BST t in alpha order */
private static void print(BST<T> t) {
    if (t == null) return;
    print(t.left);
    System.out.print(t.value);
    print(t.right);
}
```

Because of ordering rules for BST, easy to print alphabetically
- Recursively print left subtree
- Print the root
- Recursively print right subtree

### Tree traversals

"Walking" over the whole tree is a tree traversal

Previous example: in-order traversal
- Process left subtree
- Process root
- Process right subtree

(Many more on this in a couple lectures)

Other standard kinds of traversals
- preorder traversal
  - Process root
  - Process left subtree
  - Process right subtree
- postorder traversal
  - Process left subtree
  - Process right subtree
  - Process root
- level-order traversal
  - Not recursive: uses a queue
    (we'll cover this later)

### Comparing Data Structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>add (val x)</th>
<th>get (key)</th>
<th>contain (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>Linked List</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Binary Tree</td>
<td>O(depth)</td>
<td></td>
<td>O(depth)</td>
</tr>
<tr>
<td>BST</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How big could depth be?

### Worst case depth

Insert in alphabetical order...

- April
- August

Tree degenerates to list!
Need Balance

- BST search takes $O(h)$ time, where $h$ is tree height
- If data inserted in sorted order, search is $O(n)$
- **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
  - See CS 3110