CS 2230
CS II: Data structures
Meeting 23: binary search trees
Ryan Wedoff
University of Iowa
Today’s big ideas

• A *binary search tree* is a binary tree where smaller items are on the left and larger items are on the right

• A binary search tree let’s us find/insert elements in worst case $O(\log N)$ time, but only if the tree is *balanced*
Our motivation
seeing the forest for the trees :)

Two meetings ago we saw that trees make sense for naturally representing hierarchical data

We also want to use trees to find data faster

Thought experiment: how might you organize a dictionary in the computer so that you can look words up really quickly?
Our motivation seeing the forest for the trees :)  

Two meetings ago we saw that trees make sense for naturally representing hierarchical data

We also want to use trees to find data faster

**Thought experiment:** how might you organize a dictionary in the computer so that you can look words up really quickly?

We will study “binary search trees” to learn more about this general concept: organizing data cleverly means that we can access it with a lower running time
Binary search tree (BST)
every element in the **left subtree** is **less than** the root
every element in the **right subtree** is **greater than** the root
Peer instruction

Suppose we insert the number 18 into this binary search tree. Where could we put 18 to make sure the tree stays a binary search tree?

- a) right child of 33
- b) right child of 6
- c) left child of 31
- d) left child of 3
- e) right child of 12

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
Peer instruction

Which type of traversal on a BST will produce a sorted list?

a) pre order
b) in order
c) post order

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
Find an element
What is the running time of \texttt{find}(x) if \(x\) is not in the BST?
Answer in terms of \(N\), the number of nodes. Assume all the subtrees are equally sized (as above).

\begin{itemize}
  \item \(a)\) \(O(1)\)
  \item \(b)\) \(O(\log N)\)
  \item \(c)\) \(O(N)\)
  \item \(d)\) \(O(N \log N)\)
  \item \(e)\) \(O(N^2)\)
\end{itemize}
Why is the height $O(\log N)$?

<table>
<thead>
<tr>
<th>N</th>
<th>height</th>
<th>bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

$N = 2^{height} + bottoms - 1$

$(N - bottoms + 1) = 2^{height}$

$log_2(N - bottoms + 1) = height$

or equivalently

$height = \lfloor \log_2 N \rfloor$

and, we know

$\lfloor \log_2 N \rfloor \in O(\log N)$
Insert an element
Binary Search Tree simulation

http://www.cs.usfca.edu/~galles/visualization/BST.html
An exceptional case

insert(30)
insert(24)
insert(18)
insert(10)
insert(7)
insert(5)
insert(2)

If we use our algorithm for insert(), what will this BST look like?
It’s possible to get unlucky

insert(30)
insert(24)
insert(18)
insert(10)
insert(7)
insert(5)
insert(2)

insert and find are worst case \( O(N) \)!
What to look forward to

we’ll fix this problem of unlucky insert order by enhancing our insert algorithm

insert will invariant...
Today’s big ideas

• A *binary search tree* is a binary tree where smaller items are on the left and larger items are on the right

• A binary search tree let’s us find/insert elements in worst case $O(\log N)$ time, but only if the tree is *balanced*