Binary Search Trees (BSTs)

- Binary tree where each node has a value X, a left branch (<X), a right branch (>X) and a parent.
Relation to Tables

- BSTs are a good choice for implementing the Table ADT.
- The tree is constructed using the keys as node values.
- They are good because they're easy to search and do operations on...

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>V1</td>
</tr>
<tr>
<td>73</td>
<td>V2</td>
</tr>
</tbody>
</table>
```
Building a BST: Insertion

1. Follow tree down until you hit a null
2. Insert at null
Blobfish

HSIFBobLB

Insertion order matters
We only add at leaves

O(h)
Search Costs

- We have to be careful to balance the tree:
- We will look at how to auto-balance next lecture
- How do we manually balance?
Right Rotation

Inverse of left

Every rotation → one side lengthened by 1
            → one side shortened by 1
Min and Max

- Minimum of a BST can be found by walking down all of the left branches
- Maximum: all of the right branches
Deletion

Case 1:
- Delete leaf

Case 2:
- Deleted node has 1 child
  - "splice out the node"
Case 3: 2 children

1. Find successor
2. Splice out successor
3. Replace deleted node with successor

No left link.
Examples
Find the Successor

Case 1 has a right tree

⇒ Successor in right subtree

⇒ min. right subtree
Case 2: No right subhree

\[ O(h) \]

\[ \Rightarrow \text{Move up the tree until a node that} \]
\[ \text{i) is an ancestor of node} \]
\[ \text{ii) has left node that is also an ancestor} \]

\[ \text{move up until you go rather} \]
Inorder Traversal

- How do we implement a function to print the keys in order?
- Start with the first node and look for successors recursively?

```java
print_node(node n) {
    if (n.left not null)  print_node( n.left );
    print n;
    if (n.right not null) print_node( n.right );
}

print_node(root)
```
Non-recursive Inorder Traversal

- **Use a stack!**

  1. while n not null
     stack.push(n)
     n=n.left
  2. if stack not empty
     n=stack.pop()
     print n
     n=n.right
  3. if ((n not null) OR
     (stack not empty))
     GOTO 1

Essentially: find min, then find successors.
Threaded Inorder Traversal

- No stack!

1. while n.left not null
   n=n.left
2. print n
3. if (n.right is a thread)
   n=n.right
   else
     n=n.right
     while (n not null) AND (n.left not null)
     n=n.left
4. if (n not null) GOTO 2
BST So Far

- Good complexity for operations
- Easy to represent
- But falls down if it is unbalanced

- Let's see how we can auto-balance...