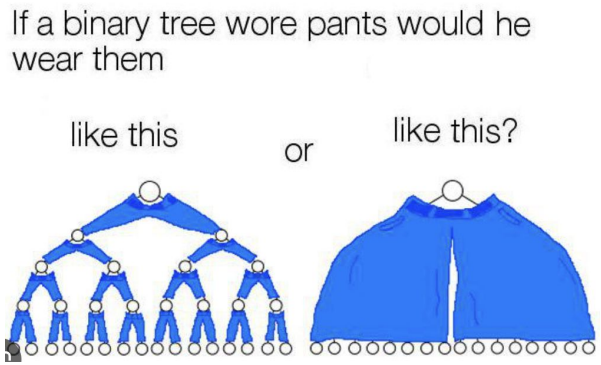
CS 1332R
WEEK 5



BST Remove

Tree Traversals

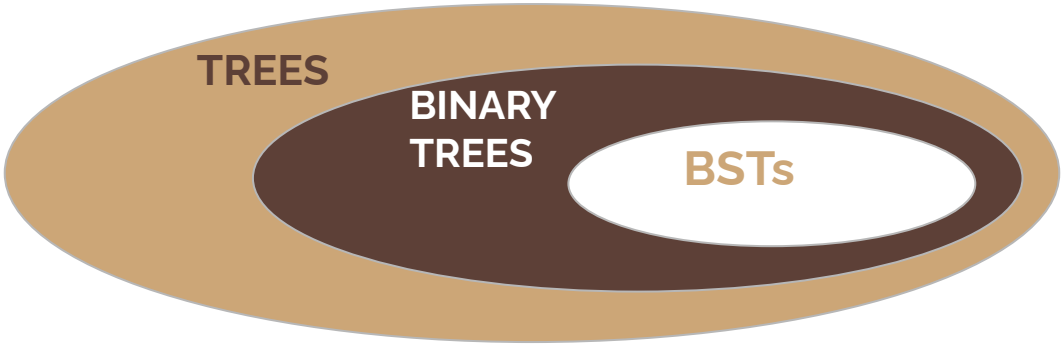
Recursive Tracing

Exam 1 Review

ANNOUNCEMENTS

-

REFRESHER: BSTs



SHAPE Property

A node cannot have more than two children.

ORDER Property

1. The left child's data must be less than the parent's data.
2. The right child's data must be greater than the parent's data.

- ➔ Accessible through a **root** reference
- ➔ **O(logn)** average case search
- ➔ The data type **must implement Comparable<T>**

Binary Search Tree: Remove

- ➔ When removing, we must maintain the order property.
- ➔ We do not allow empty nodes.

PROCEDURE

1. Search for the data you would like to remove.
2. If **currentNode == null**, the data is not in the tree.
3. If **currentNode.data == data**, there are 4 cases:
   1) The node has no children. **return null**
   2) The node only has a left child. **return the left child**
   3) The node only has a right child. **return the right child**
   4) The node has two children. **replace data in the node with the successor/predecessor *and remove the successor/predecessor node***

What do we do in each case?

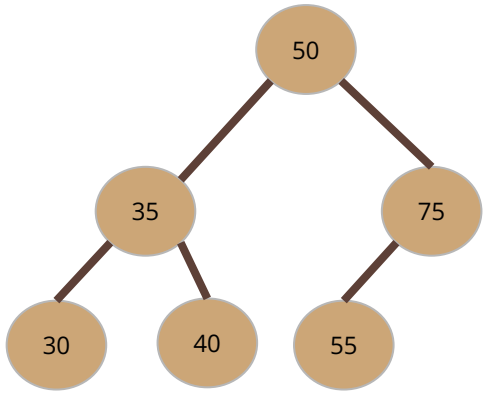
Binary Search Tree: Remove

PREDECESSOR

- Greatest value that is smaller than the node we want to remove
- Go one node to the left, then to the last possible right node

SUCCESSOR

- Smallest value that is greater than the node we want to remove
- Go one node to the right, then to the last possible left node



Predecessor of 50? 40

Successor of 50? 55

Inorder Traversal: 30, 35, **40**, 50, **55**, 75

Binary Search Tree: Remove

Would we use pointer reinforcement?

Yes

Why?

Because we are altering the structure of the tree

Binary Search Tree: Remove

1. Search for the data you would like to remove.
2. If **currentNode == null**, the data is not in the tree.
3. If **currentNode.data == data**, there are 4 cases:
    1) The node has no children → ***return null***
    2) The node only has a left child. → ***return node.left***
    3) The node only has a right child. → ***return node.right***
    4) The node has two children. → ***remove pred/succ node*** ***and replace data with pred/succ data***

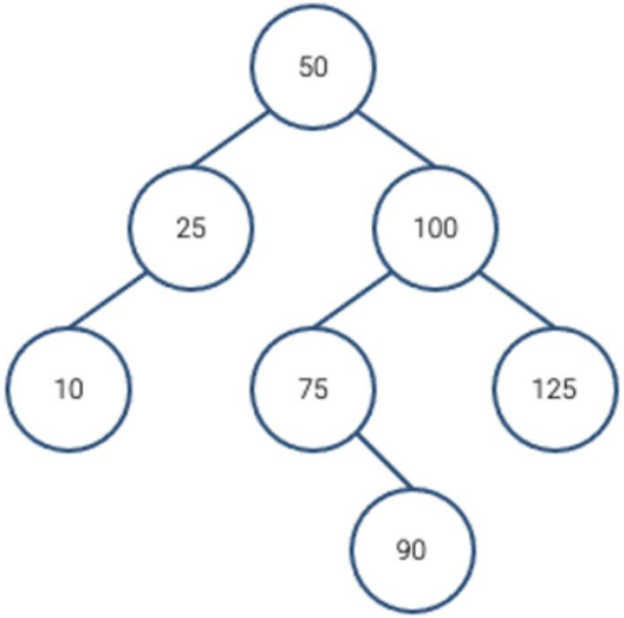
How do we save the data we want to remove when recursing?

"dummy"/container node

How could we remove the predecessor and successor node?

traverse the tree ***starting at our current node*** to find the pred/succ node

Binary Search Tree: Remove



if (100> 50): 50 .right = remove(50 .right , 100)

100

if (100 ==100):
- find predecessor
- remove predecessor

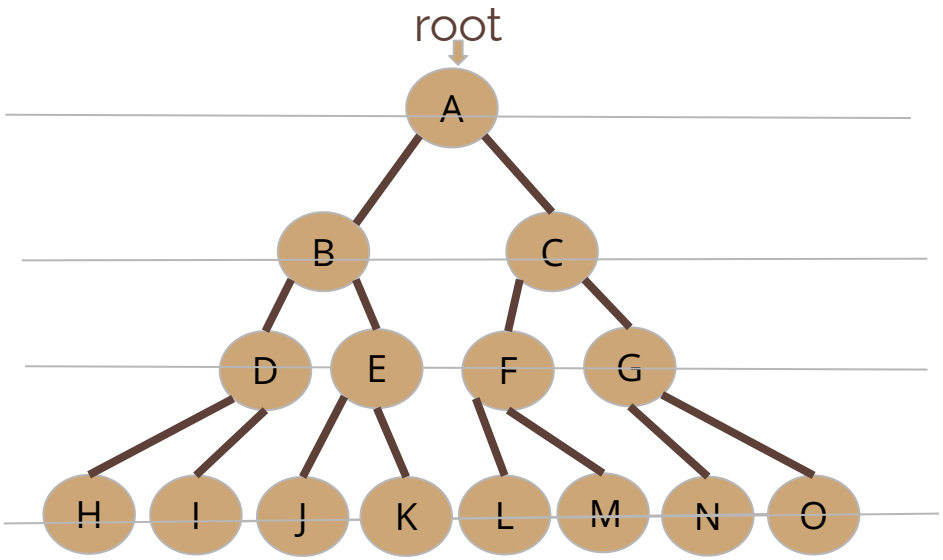
75 .right = removePred(75 .right)

null

- replace data in node 100 with 90, return 100

csvistool.com

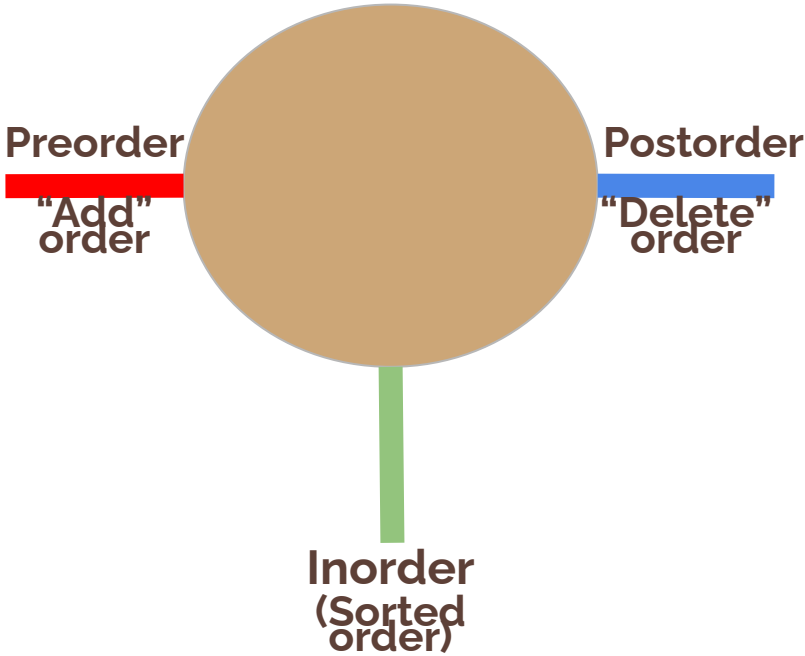
Binary Search Tree: Depth Traversals



Preorder: A B D H I E J K C F L M G N O

Inorder: H D I B J E K A L F M C N G O

Postorder: H I D J K E B L M F N O G C A



Binary Search Tree: Depth Traversals

Would we use recursion?

YES

Would we use pointer reinforcement?

NO

Binary Search Tree: Depth Traversals

3 parts to every traversal:

Write data at the Current node (C)

Traverse Left (L)

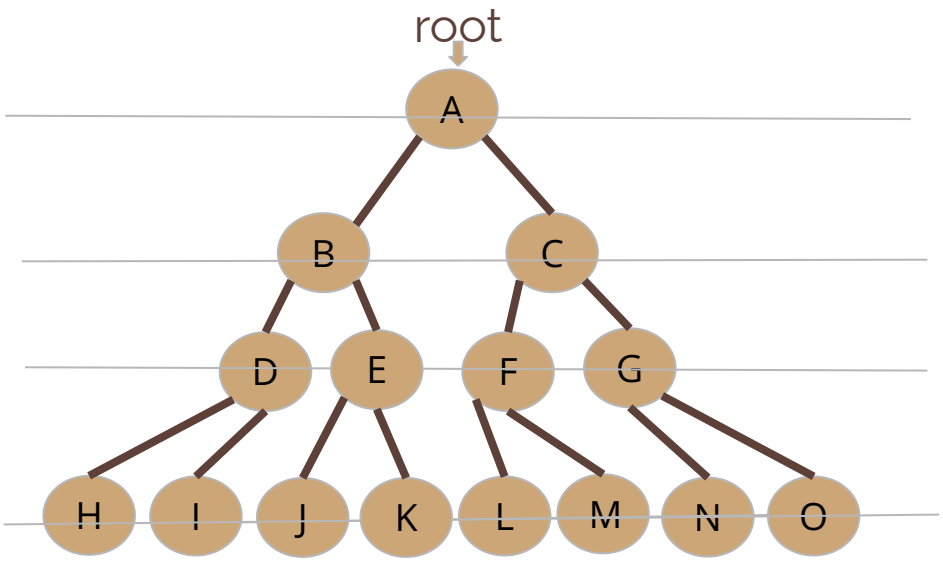
Traverse Right (R)

PREORDER	INORDER	POSTORDER
C	L	L
L	C	R
R	R	C

What is our base case to end recursion?

if the node is null

Binary Search Tree: Breadth Traversals



PROCEDURE

1. Use a queue to represent your frontier of nodes.
2. As you dequeue a node, add its data to your list and enqueue its children.

Iterative or recursive?

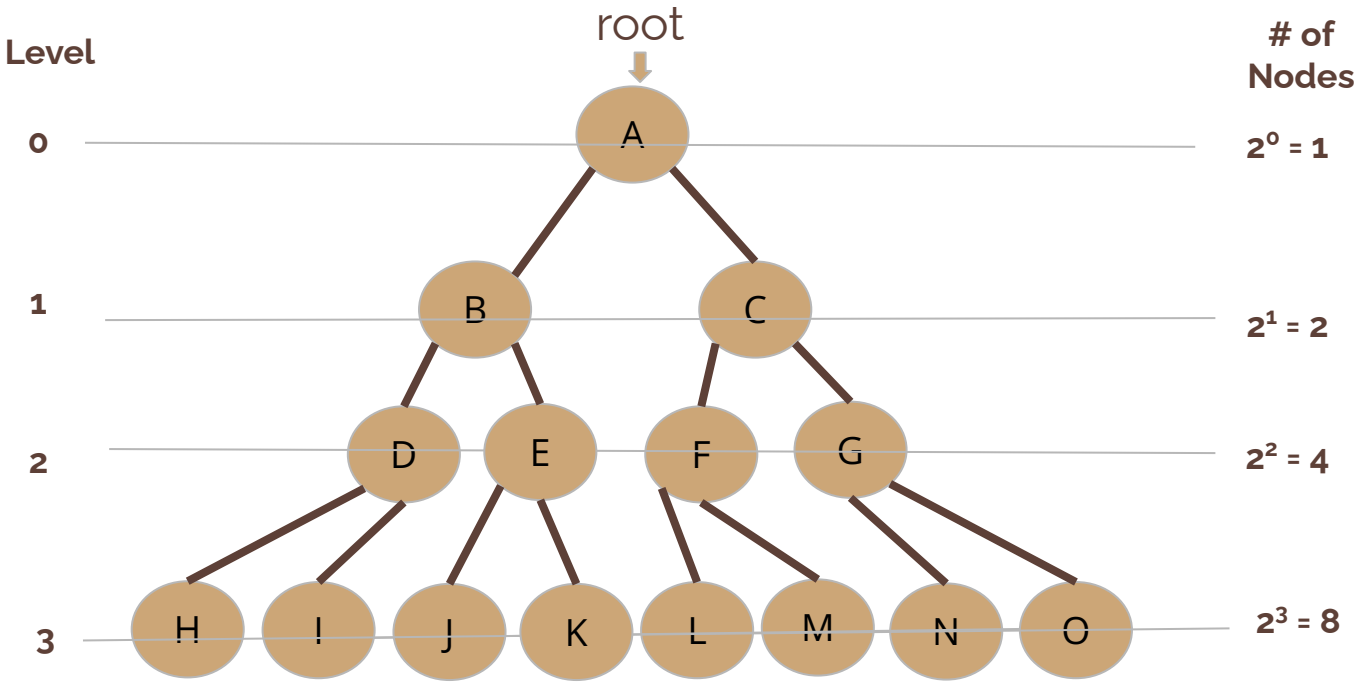
iterative

Level Order: A B C D E F G H I J K L M N O

Data is read left to right, starting at the lowest level to deepest level

Binary Search Tree: Efficiencies

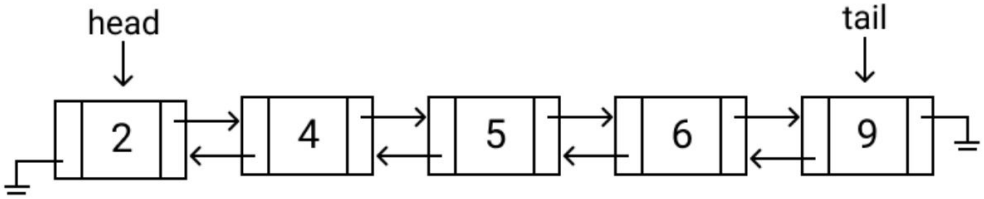
	Adding	Remove	Accessing	Height	Traversals
Average	*O(log n)*	*O(log n)*	*O(log n)*	*O(n)*	*O(n)*
Worst	*O(n)*	*O(n)*	*O(n)*	*O(n)*	*O(n)*



RECURSION EXAMPLE

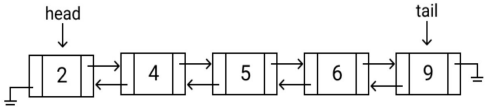
Trace the execution of the mystery method when given the **HEAD of the doubly linked list** diagrammed below. Write the printed output of the method in the indicated output box. Write each print statement's output on a new line.

```
private class Node {
    private int data;
    private Node next;
    private Node prev;
    // constructors & class methods not shown
}

public static void mystery(Node curr) {
    if (curr == null) {
        System.out.println("Base");
    } else {
        if (curr.data % 2 == 0) {
            mystery(curr.next.next);
        } else {
            mystery(curr.prev);
        }
        System.out.println(curr.data);
    }
}
```



RECURSION EXAMPLE: Tracing

```
private class Node {
    private int data;
    private Node next;
    private Node prev;
    // constructors & class methods not shown
}

public static void mystery(Node curr) {
 1  if (curr == null) {
 2      System.out.println("Base");
 3  } else {
 4      if (curr.data % 2 == 0) {
 5          mystery(curr.next.next);
 6      } else {
 7          mystery(curr.prev);
 8      }
 9      System.out.println(curr.data);
    }
}
```



Original Call	Recursive Call	Line # of Recursive Call	"curr"	Returned Value/Print Statement
mystery(null)	N/A	N/A	null	Base
mystery([6])	mystery(null)	Line 5	[6]	6
mystery([4])	mystery([6])	Line 5	[4]	4
mystery([5])	mystery([4])	Line 7	[5]	5
mystery([2])	mystery([5])	Line 5	[2]	2

mystery(head)

LEETCODE PROBLEMS

102. Binary Tree Level Order Traversal

1008. Construct Binary Tree From Preorder Traversal

EXAM 1 REVIEW

Kahoot

(There's also a practice exam in Canvas: Files -> resources -> recitation materials -> recitation practice exams)

Any questions?

Name
Office Hours
Contact

Name
Office Hours
Contact

Let us know if there is anything specific you want out of recitation!