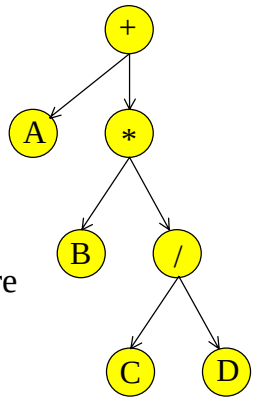
Trees, Trees, and Trees

- Today's agenda:
  - Traversing trees
  - Binary Search Trees
  - ADT Operations: Find, Insert, Remove (Delete), etc…
- Covered in Chapter 4 of the text

Example: Representing Arithmetic Expressions

Example Arithmetic Expression:
A + (B * (C / D))

Tree for the above expression:
Leaves = operands (constants/variables)
Non-leaf nodes = operators



- Used in most compilers
- No parenthesis need – use tree structure
- Can speed up calculations e.g. replace / node with C/D if C and D are known

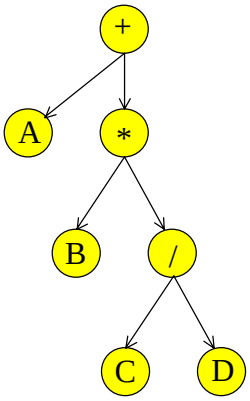
How do you evaluate the expression represented by the tree?

Evaluating Arithmetic Expression Trees

How do you evaluate the expression represented by this tree?

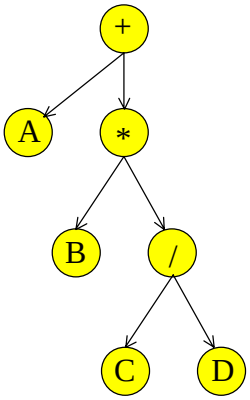
1. Recursively evaluate left and right subtrees
2. Apply operation at the root

Known as "Postorder traversal"
Process children first and then the root (therefore "post" order)



Traversing Trees

- Postorder: Children first, then Root
  A B C D / * +
- Preorder: Root, then Children
  + A * B / C D
- Inorder: Left child, Root, Right child
  A + B * C / D



Example Pseudocode for Recursive Preorder

```
void print_preorder (TreeNode T)
{ TreeNode P;
  if  ( T == NULL ) return;
  else  {  <Print Element stored in T>
           P = T.FirstChild;
           while (P != NULL) {
             print_preorder ( P );
             P = P.NextSibling; }
        }
}
```

What is the running time for a tree with N nodes?

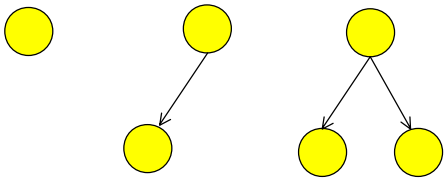
Preorder Traversal with a Stack

```
void Stack_Preorder (TreeNode T, Stack S) {
 if (T == NULL) return; else push(T,S);
 while (!isEmpty(S)) {
  T = pop(S);
  <Print Element stored in T>
  if (T.NextSibling !=  NULL)
     push(T.NextSibling, S);
  if (T.FirstChild !=  NULL)
     push(T.FirstChild, S);
 }
}
```

What is the running time for a tree with N nodes?

Binary Trees

- Every node has at most two children
  - Most popular tree in computer science
- Given N nodes, what is the minimum depth of a binary tree?



Depth 0: N = 1 = 2^0 nodes
Depth 1: N = 2 to 3 nodes = 2^1 to 2^{1+1}-1 nodes
At depth d, N = ?

Deep Facts about Binary Trees

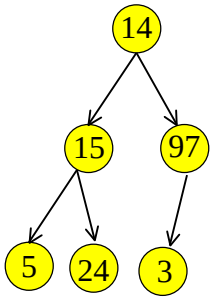
- Given N nodes, what is the minimum depth of a binary tree?
  Depth 0: N = 1 node = 2^0
  Depth 1: N = 2 to 3 nodes = 2^1 to 2^{1+1}-1
  At depth d, N = 2^d to 2^{d+1}-1 nodes (a full tree)
  So, minimum depth d is:
  $\log N \leq d \leq \log(N+1)-1$ or $\Theta(\log N)$
- What is the maximum depth of a binary tree?

More Deep Facts about Binary Trees

- Minimum depth of N-node binary tree is $\Theta(\log N)$
- What is the maximum depth of a binary tree?
  - Degenerate case: Tree is a linked list!
  - Maximum depth = N-1
- Goal: Would like to keep depth at around log N to get better performance than linked list for operations like Find.

Array Implementation of Binary Trees

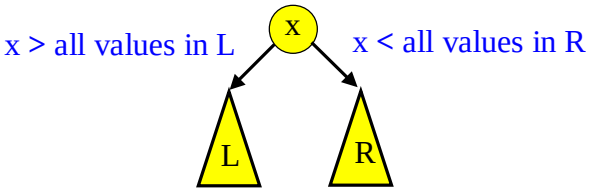
- Used mostly for complete binary trees
  - A complete tree has no gaps when you scan the nodes left-to-right, top-to-bottom
- Idea: Use left-to-right scan to impose a linear order on the tree nodes
- Implementation:
  - **Children of A[i] = A[2i+1], A[2i+2]**
  - Use a default value to indicate empty node
  - Why is this implementation inefficient for non-complete trees?

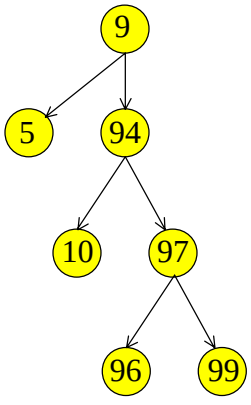


0	1	2	3	4	5	…	MAX
14	15	97	5	24	3	…	

Binary Search Trees

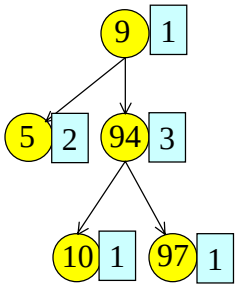
- Binary search trees are binary trees in which **the value in every node is**:
  - > all values in the **node's left subtree**
  - < all values in the **node's right subtree**





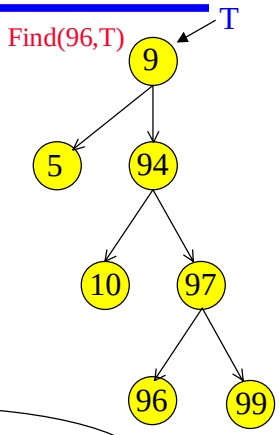
Binary Search Trees

- Handling Duplicates:
  - Increment a counter stored in item's node or
  - Use a linked list or another search tree at item's node
- Application: "Look-up" table
  - E.g.: Academic records systems:
    Given SSN, return student record
    SSN stored in each node as the key value
  - E.g.: Given zip code, return city/state
  - E.g.: Given name, return address/phone no.
    - Can use dictionary order for strings



Operations on Binary Search Trees

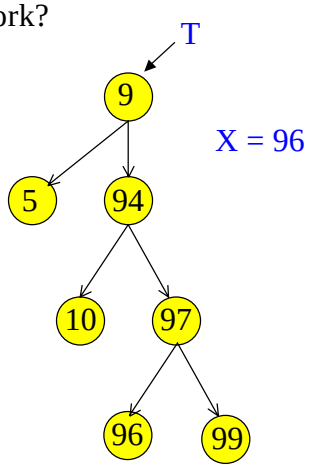
- Main Operations:
  FindMin(BinaryNode T)
  FindMax(BinaryNode T)
  Find(Comparable X, BinaryNode T)
  Insert(Comparable X, BinaryNode T)
  Remove(Comparable X, BinaryNode T)
- How would you implement these?
  - Exercise: How does Find(X,T) work?



Hint: Recursive definition of BSTs allows recursive routines!

Find on BSTs

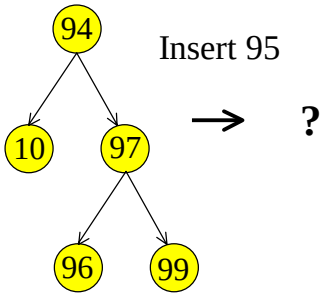
- Exercise: How does Find(X,T) work?
  1. If T null, return null
  2. If X < T.Element
     return Find(X,T.left)
     else if X > T.Element
     return Find(X,T.right)
     else return T //Found!



Insert Operation

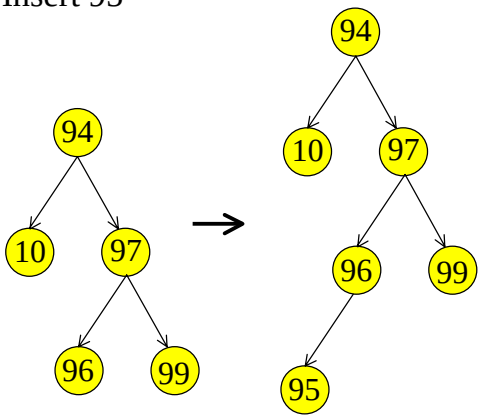
- Similar to Find
- Insert(Comparable X, BinaryNode T)

```
If T null, Insert X at T
else if X < T.Element
     T.left =  Insert(X,T.left)
else if X > T.Element
     T.right = Insert(X,T.right)
else
    Duplicate:
    Update duplicates counter
Return T
```



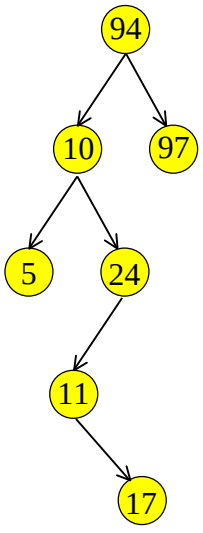
Example of Insert

- Example: Insert 95



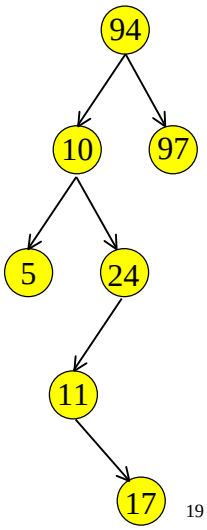
Remove (or Delete) Operation

- Remove is a bit trickier…Why?
- Suppose you want to remove 10
- Strategy:
  1. Find 10
  2. Remove the node containing 10
- Problem: When you remove a node, what do you replace it with?



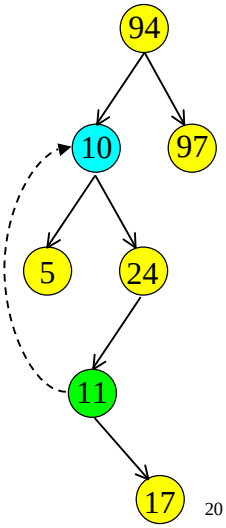
A Problem with Remove

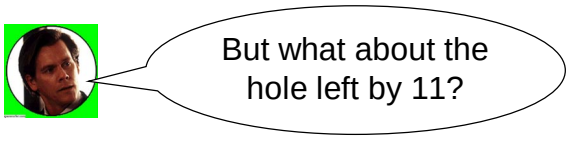
- Problem: When you remove a node, what do you replace it with?
- Three cases:
  1. If it has no children, with NULL
     E.g. Remove 5
  2. If it has 1 child, with that child
     E.g. Remove 24 (replace with 11)
  3. If it has 2 children, with the smallest value node in its right subtree
     E.g. Remove 10 (replace with 11)
- Preserves BST property



More Trouble with Remove

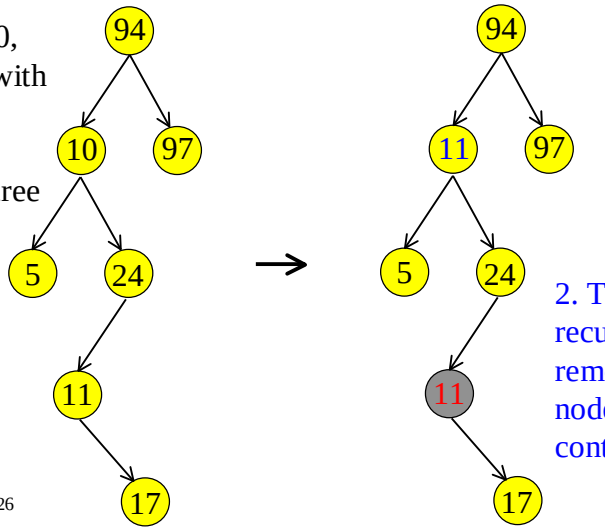
- Problem: When you remove a node, what do you replace it with?
  3. If it has 2 children, with the smallest value node in its right subtree
     E.g. Remove 10 (replace with 11)





Example: Remove “10”

1. Find 10, Replace with smallest value in right subtree

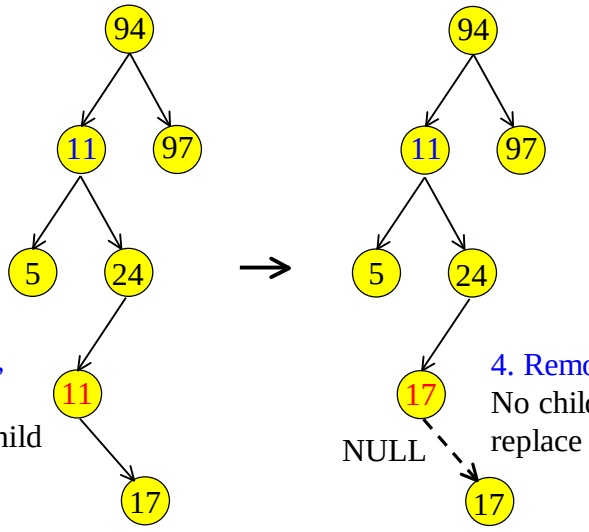


2. Then, recursively remove the node containing 11

Example: Remove “10”

2. Remove “11” in right subtree (recursive remove)

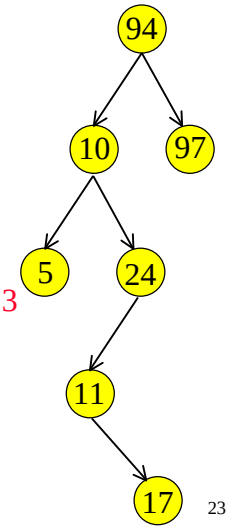
3. Find “11”, 1 child – so replace by child



4. Remove “17” No child, so replace by NULL

Summary of Remove

- Removing a node containing X:
  1. Find the node containing X
  2. Replace it with:
     If it has no children, with NULL
     If it has 1 child, with that child
     If it has 2 children, with the node with the smallest value in its right subtree, (or largest value in left subtree)
  3. Recursively remove node used in 2 and 3
- Worst case: Recursion propagates all the way to a leaf node – time is O(depth of tree)



Next Class:

A Balancing Act with Trees

Species of Trees: AVL, splay, and B

To Do:

Read Chapter 4

Finish Homework #1 (due Wednesday)