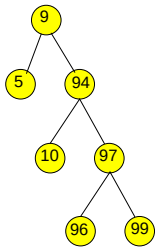
Binary Search Trees

CSE 373
Data Structures
Winter 2007

Binary Search Trees

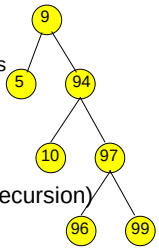
- Binary search trees are binary trees in which
  - all values in the node's left subtree are less than node value
  - all values in the node's right subtree are greater than node value
- Operations:
  - Find, FindMin, FindMax, Insert, Delete

What happens when we traverse the tree in inorder?

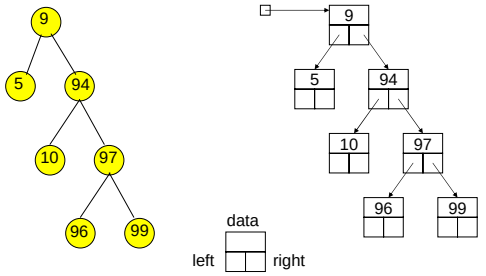


Operations on Binary Search Trees

- How would you implement these?
  - Recursive definition of binary search trees allows recursive routines
- FindMin
- FindMax
- Find
- Insert (but be careful when using recursion)
- Delete (the only tricky one)



Binary SearchTree



Find

```
Find(T : tree pointer, x : element): tree pointer {
case {
  T = null : return null;
  T.data = x : return T;
  T.data > x : return Find(T.left,x);
  T.data < x : return Find(T.right,x)
}
}
```

•Question: How could you do this iteratively?
•Good idea? Bad idea?
•(in terms of what: Implementation ease? Time? Space?)

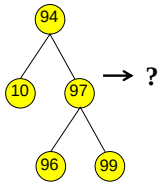
FindMin

- Design recursive FindMin operation that returns the smallest element in a binary search tree.

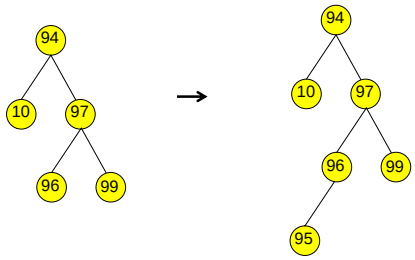
```
FindMin(T : tree pointer) : tree pointer {
// precondition: T is not null //
if T.left = null return T
 else return FindMin(T.left)
 }
```

Insert Operation

- **Insert(T: tree, X: element)**
  - › Do a "Find" operation for X
  - › If X is found → update (no need to insert)
  - › Else, "Find" stops at a NULL pointer
  - › Insert Node with X there
- Example: Insert 95



Insert 95

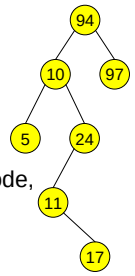


Recursive Insert

```
Insert(T : tree pointer, x : element) : tree pointer {
if T = null then
  T := new tree; T.data := x; return T;//the links to
                                       //children are null
case
  T.data > x : T.left := Insert(T.left, x);
  T.data < x : T.right := Insert(T.right, x);
  T.data = x : break;//Might throw an exception
endcase
}
```

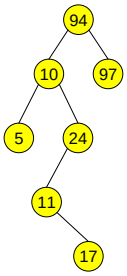

Delete Operation

- Delete is a bit trickier…Why?
- Suppose you want to delete 10
- Strategy:
  - › Find 10
  - › Delete the node containing 10
- Problem: When you delete a node, what do you replace it by?



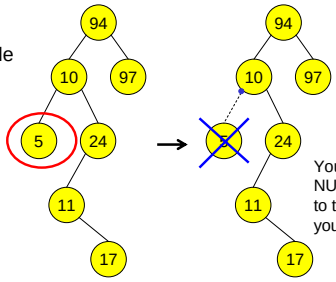
Delete Operation

- Problem: When you delete a node, what do you replace it by?
- Solution:
  - › If it has no children, by NULL
  - › If it has 1 child, by that child
  - › If it has 2 children, by the node with the smallest value in its right subtree (the inorder successor of the node)



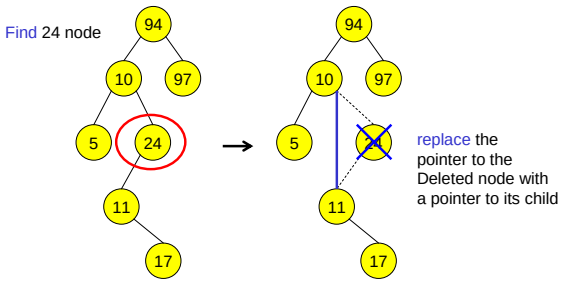
Delete "5" - No children

Find 5 node

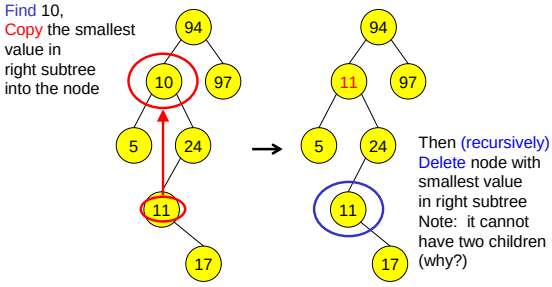


You need to NULL the pointer to the node that you are deleting

Delete “24” - One child



Delete “10” - two children



Then Delete “11” - One child