CSC 212
Data Structures and Abstractions
Spring 2016

Lecture 11: Binary Search Trees II
Previously …

Binary Trees

Binary Search Trees
  definition
  search
Today ...

Binary Search Trees

insertion
deletion
traversals
Quiz

Full? Complete? Perfect?
Quiz

Full? Complete? Perfect?
Quiz

Full? Complete? Perfect?
Quiz

Full? Complete? Perfect?
Basic Operations
Insert

Do a Search operation

if found, no need to insert (may increase counter)
else, insert node where Search stopped
Insert key = 65
Insert key = 65
Insert key = 65
Insert key = 65
Insert key = 27
Insert key = 27
Insert key = 27
Insert key = 27
Insert key = 27
Insert key = 27
void BSTree::insert(BSTNode **p, int d) {
    if (! *p) {
        // insert node here
        (*p) = new BSTNode(d);
    } else {
        // call recursively
        if ((*p)->data < d) insert(&((*p)->right), d);
        else if ((*p)->data > d) insert(&((*p)->left), d);
    }
}

void BSTree::insert(int d) {
    insert(&root, d);
}
Remove

Case 1: node is a leaf
trivial, delete node and set parent’s pointer to NULL

Case 2: node has 1 child
trivial, set parent’s pointer to the only child and delete node

Case 3: node has 2 children
find successor and copy successor’s data to node
delete successor
Case 1, Remove key = 27
Case 1, Remove key = 27
Case 1, Remove key = 27
Case 1, Remove key = 40
Case 1, Remove key = 40
Case 1, Remove key = 40
Case 2, Remove key = 80
Case 2, Remove key = 80
Case 2, Remove key = 80
Case 2, Remove key = 80
Case 2, Remove key = 20
Case 2, Remove key = 20
Case 2, Remove key = 20
Case 2, Remove key = 20
Case 3, Remove key = 30
Case 3, Remove key = 30
Case 3, Remove key = 30

The diagram shows a binary search tree with the following structure:

- Root: 50
- Left of 50: 35 (Successor)
  - Left of 35: 25
    - Left of 25: 21
    - Right of 25: 27
  - Right of 35: 40
- Right of 50: 70
  - Left of 70: 75
  - Right of 70: 80
  - Right of 70: 60
Case 3, Remove key = 30
Case 3, Remove key = 30
Remove key = 50 ??
Traversals
Preorder Traversal

```
algorithm preorder(p) {
    if (p) {
        visit(p)
        preorder(p->left)
        preorder(p->right)
    }
}
```
Preorder Traversal

algorithm preorder(p) {
    if (p) {
        visit(p)
        preorder(p->left)
        preorder(p->right)
    }
}
Inorder Traversal

```
algorithm inorder(p) {
    if (p) {
        inorder(p->left)
        visit(p)
        inorder(p->right)
    }
}
```
Inorder Traversal

```
algorithm inorder(p) {
    if (p) {
        inorder(p->left)
        visit(p)
        inorder(p->right)
    }
}
```
Postorder Traversal

```
algorithm postorder(p) {
    if (p) {
        postorder(p->left)
        postorder(p->right)
        visit(p)
    }
}
```
Postorder Traversal

Algorithm postorder(p) {
    if (p) {
        postorder(p->left)
        postorder(p->right)
        visit(p)
    }
}
How to destroy a binary tree?