

Binary Search Trees

CSE 373
Data Structures & Algorithms
Linda Shapiro
Spring 2013

Today's Outline

- **Announcements**
 - Assignment #2 due Fri, Oct 19 at the BEGINNING of lecture. This is a pen and paper assignment; print nicely or type, please.
- **Today's Topics:**
 - **Binary Search Trees**
 - **Complexity Analysis (always)**

4/12/2013

cse 373 13sp - Binary Search Trees

2

Why Do We Need Trees?

- Lists, Stacks, and Queues are linear relationships
- Information often contains hierarchical relationships
 - File directories or folders
 - Moves in a game
 - Hierarchies in organizations
- Can build a tree to support fast searching

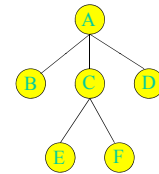
4/12/2013

cse 373 13sp - Binary Search Trees

3

Tree Jargon

- root
- nodes and edges
- leaves
- parent, children, siblings
- ancestors, descendants
- subtrees
- path, path length
- height, depth



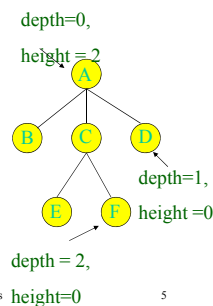
4/12/2013

cse 373 13sp - Binary Search Trees

4

More Tree Jargon

- **Length of a path** = number of edges
- **Depth of a node N** = length of path from root to N
- **Height of node N** = length of longest path from N to a leaf
- **Depth of tree** = depth of deepest node
- **Height of tree** = height of root



4/12/2013

cse 373 13sp - Binary Search Trees

5

Implementation of Trees

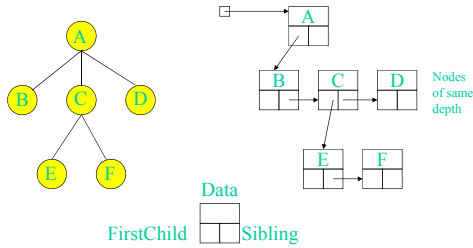
- One possible pointer-based Implementation
 - tree nodes with value and a pointer to each child
 - but how many pointers should we allocate space for?
- A more flexible pointer-based implementation
 - 1st Child / Next Sibling List Representation
 - Each node has 2 pointers: one to its first child and one to next sibling
 - Can handle arbitrary number of children

4/12/2013

cse 373 13sp - Binary Search Trees

6

Arbitrary Branching



4/12/2013

cse 373 13sp - Binary Search Trees

7

Tree Calculations

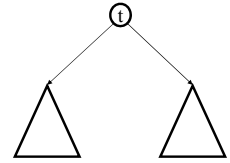
Recall: height is max number of edges from root to a leaf

Find the height of the tree...

Height(null) = -1

Height = max (height(left), height(right)) + 1

runtime for tree of TS nodes: $O(TS)$



4/12/2013

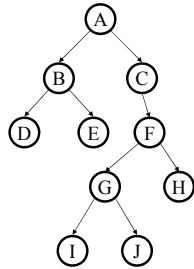
cse 373 13sp - Binary Search Trees

8

Binary Trees

- Binary tree is
 - a root
 - left subtree (maybe empty)
 - right subtree (maybe empty)

• Representation:

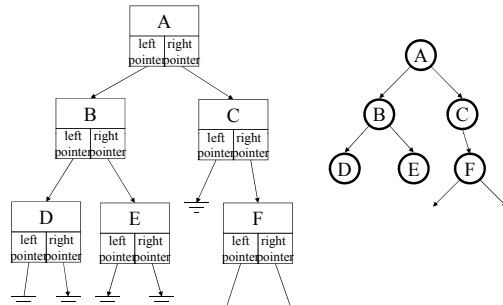


4/12/2013

cse 373 13sp - Binary Search Trees

9

Binary Tree: Representation



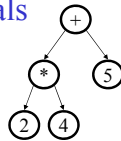
4/12/2013

cse 373 13sp - Binary Search Trees

10

More Recursive Tree Calculations: Binary Tree Traversals

A *traversal* is an order for visiting all the nodes of a binary tree



Three types:

- Pre-order: Root, left subtree, right subtree (an expression tree)
 $+ * 2 4 5$
- In-order: Left subtree, root, right subtree $2 * 4 + 5$
- Post-order: Left subtree, right subtree, root $2 4 * 5 +$

4/12/2013

cse 373 13sp - Binary Search Trees

11

Traversals

```
void traverse(BNode t){
    if (t != NULL)
        traverse (t.left);
        print t.element;
        traverse (t.right);
    }
}
```

Which one is this?

4/12/2013

cse 373 13sp - Binary Search Trees

12

ADTs Seen So Far

- Stack
 - Push
 - Pop
- Queue
 - Enqueue
 - Dequeue

What are these in Java?

- add
- remove

4/12/2013 cse 373 13sp - Binary Search Trees 13

The Dictionary ADT

- Data:
 - a set of (key, value) pairs
- Operations:
 - Insert (key, value)
 - Find (key)
 - Remove (key)

• **jigile**
Jacob Gile
OH: T 12:30-1:20pm
CSE 220

• **khend**
Daphna Khen
OH: Th 11:30-12:20
CSE 220

• **genelkim**
Gene Kim
OH: Th 4:30-5:20
CSE 220

insert(kedzior, ...) →
 ← find(genelkim)
 • genelkim
 Gene Kim ...

• Java has a Map interface that has similar operations with different names: put, containsKey, get,

4/12/2013 cse 373 13sp - Binary Search Trees 14

A Modest Few Uses

- Search (databases) : phone directories or other large data sets (genome maps, web pages)
- Networks : Router tables
- Operating systems : Page tables
- Compilers : Symbol tables
- Image Analysis : Object-feature tables
- Image Retrieval : Large image databases

Probably the most widely used ADT!

4/12/2013 cse 373 13sp - Binary Search Trees 15

Implementations: Complexity

For dictionary with n key/value pairs

insert	find	delete
--------	------	--------

- Unsorted Linked-list
- Unsorted array
- Sorted array

4/12/2013 cse 373 13sp - Binary Search Trees 16

Implementations

For dictionary with n key/value pairs

	insert	find	delete
• Unsorted linked-list	$O(1)$ *	$O(n)$	$O(n)$
• Unsorted array	$O(1)$ *	$O(n)$	$O(n)$
• Sorted linked list	$O(n)$	$O(n)$	$O(n)$
• Sorted array	$O(n)$	$O(\log n)$	$O(n)$

*Note: If we do not allow duplicates values to be inserted, we would need to do $O(n)$ work (a find operation) to check for a key's existence before insertion

4/12/2013 cse 373 13sp - Binary Search Trees 17

Binary Search Tree Data Structure

- Structural property
 - each node has ≤ 2 children
 - result:
 - storage is small
 - operations are simple
 - average depth is small
- Order property
 - all keys in left subtree smaller than root's key
 - all keys in right subtree larger than root's key
 - result: easy to find any given key

So, when I store, I have to test where to put the new node.

4/12/2013 cse 373 13sp - Binary Search Trees 18

Are these BSTs?

4/12/2013 cse 373 13sp - Binary Search Trees 19

Find in BST, Recursive

```
Node Find(Object key,
          Node root) {
    if (root == NULL)
        return NULL;

    if (key < root.key)
        return Find(key,
                   root.left);
    else if (key > root.key)
        return Find(key,
                   root.right);
    else
        return root;
}
```

Runtime: Worst Case: $O(n)$
Actual: $O(\text{depth})$

4/12/2013 cse 373 13sp - Binary Search Trees 20

Find in BST, Iterative

```
Node Find(Object key,
          Node root) {
    while (root != NULL &&
           root.key != key) {
        if (key < root.key)
            root = root.left;
        else
            root = root.right;
    }
    return root;
}
```

Runtime:

4/12/2013 cse 373 13sp - Binary Search Trees 21

Insert Operation

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

4/12/2013 cse 373 13sp - Binary Search Trees 22

Insert 95

4/12/2013 cse 373 13sp - Binary Search Trees 23

Can we convert Find into Insert?

```
Node Find(Object key,
          Node root) {
    while (root != NULL &&
           root.key != key) {
        if (key < root.key)
            root = root.left;
        else
            root = root.right;
    }
    return root;
}
```

4/12/2013 cse 373 13sp - Binary Search Trees 24

Can we convert Find into Insert?

The idea (incomplete)

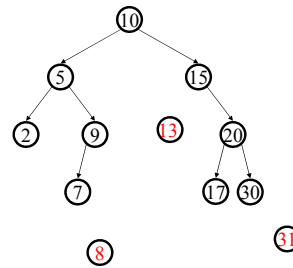
```
while (root != NULL &&
       root.key != key) {
  if (key < root.key)
    if root.left != null
      root = root.left;
    else root.left = new ....
  else
    if root.right != null
      root = root.right;
    else root.right = new ....
}
```

4/12/2013

cse 373 13sp - Binary Search Trees

25

Try some Inserts in a BST



Insert(13)
Insert(8)
Insert(31)

Runtime: $O(\text{depth})$

4/12/2013

cse 373 13sp - Binary Search Trees

26

BuildTree for BST

- Suppose keys 1, 2, 3, 4, 5, 6, 7, 8, 9 are inserted into an initially empty BST.

Runtime depends on the order!

- in given order
- in reverse order
- median first, then left median, right median, etc.

4/12/2013

cse 373 13sp - Binary Search Trees

27

In General

- Binary Search Trees are not balanced.
- The depth can range from depth n for an n node tree in the worst case, in which case the tree is just a linear list
- To the best case which is a perfectly balanced tree, in which case the depth is $\log_2 n$.

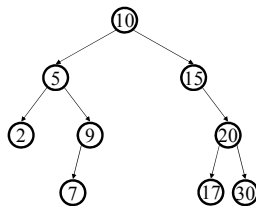
4/12/2013

cse 373 13sp - Binary Search Trees

28

FindMin/FindMax are Easy

- Find minimum
- Find maximum



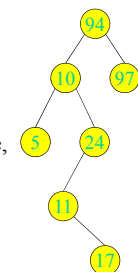
4/12/2013

cse 373 13sp - Binary Search Trees

29

Delete Operation

- Delete is a bit trickier than insert... 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?



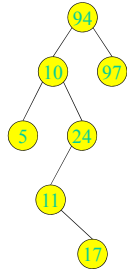
4/12/2013

cse 373 13sp - Binary Search Trees

30

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 successor of the node)



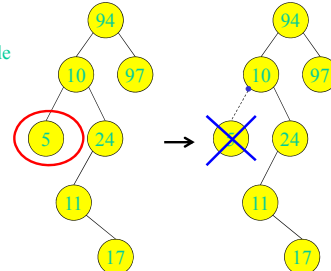
4/12/2013

cse 373 13sp - Binary Search Trees

31

Delete "5" - No children

Find 5 node



Then Free the 5 node and NULL the pointer to it

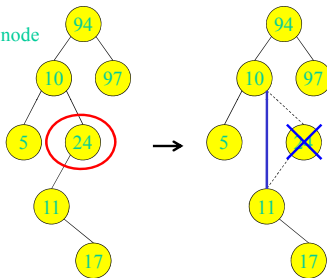
4/12/2013

cse 373 13sp - Binary Search Trees

32

Delete "24" - One child

Find 24 node



Then Free the 24 node and replace the pointer to it with a pointer to its child

4/12/2013

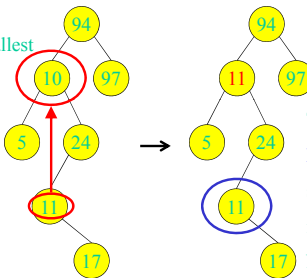
cse 373 13sp - Binary Search Trees

33

Delete "10" - two children

Find 10,

Copy the smallest value in right subtree into the node



Then (recursively) Delete node with smallest value in right subtree
Note: it cannot have two children

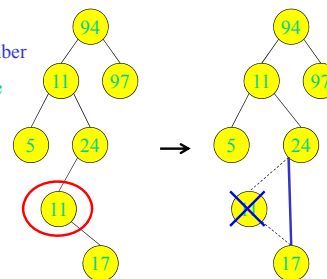
4/12/2013

cse 373 13sp - Binary Search Trees

34

Then Delete "11" - One child

Remember 11 node



Then Free the 11 node and replace the pointer to it with a pointer to its child

4/12/2013

cse 373 13sp - Binary Search Trees

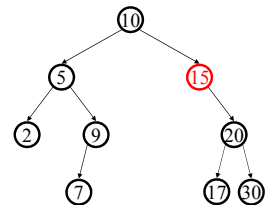
35

Lazy Deletion

Instead of physically deleting nodes, just mark them as deleted

- + simpler
- + physical deletions done in batches
- + some adds just flip deleted flag

- extra memory for deleted flag
- many lazy deletions slow finds
- some operations may have to be modified (e.g., min and max)



4/12/2013

cse 373 13sp - Binary Search Trees

36

Balanced BST

Observation

- BST: **the shallower the better!**
- For a BST with n nodes
 - Average height is $\Theta(\log n)$
 - Worst case height is $\Theta(n)$
- Simple cases such as insert(1, 2, 3, ..., n) lead to the worst case scenario

Solution: Require a **Balance Condition** that

1. ensures depth is $\Theta(\log n)$ – strong enough!
2. is easy to maintain – not too strong!

4/12/2013

cse 373 13sp - Binary Search Trees

37

Potential Balance Conditions

1. Left and right subtrees of the root have equal number of nodes

2. Left and right subtrees of the root have equal *height*

4/12/2013

cse 373 13sp - Binary Search Trees

38

Potential Balance Conditions

3. Left and right subtrees of *every node* have equal number of nodes

4. Left and right subtrees of *every node* have equal *height*

4/12/2013

cse 373 13sp - Binary Search Trees

39