Binary Search Trees (BSTs)

2/24/16
Announcements

• HW due Wednesday: batch simulator

• Quiz on Friday: Radix sort and/or hashing

• Reading:
  – Friday: Chapter 23 (don’t worry about the code and some is review from today)
  – Monday (maybe delayed): Chapter 26 (heaps)
Recall: Set ADT

- Represents unordered set of values
  \{"hello", "there", "=)", "CS 142"\}

- Supports:
  - boolean add(value) //returns whether it was new
  - boolean contains(value)
  - Iterator iterator()

Maybe also: size, remove, clear
Also recall...

• Implementation 1:
  Array w/ vals in arbitrary order, but no duplicates
  – O(n) add, O(n) contains

• Implementation 2:
  Array with values in sorted order
  – Allows binary search
How long does it take to add a new key into the sorted array?

A. $O(1)$
B. $O(1)$ amortized
C. $O(\log n)$
D. $O(n)$
E. None of the above
How long does it take to add a new key into the sorted array?

A. O(1)
B. O(1) amortized
C. O(log n)
D. O(n)
E. None of the above
Desired properties

• Ability to quickly “jump” later in the sequence of keys (like binary search)

• Ability to easily add keys into the appropriate position (like linked lists)
Trees
Trees

root: top of tree

children of w

children of x
Trees

root: top of tree

parent of x and y

children of w

children of x

z is grandchild of w

y is uncle (or aunt) of z

Other familial relationships:
x and y are siblings
w is grandparent of z
Trees

root: top of tree

parent of x and y

children of w

children of x

leaves: nodes w/o children

Other familial relationships:
- x and y are siblings
- w is grandparent of z
- z is grandchild of w
- y is uncle (or aunt) of z
Binary Search Trees (BSTs)

• Tree with key stored at each node such that
  – Every node has 2 children (left and right)
    • Children can be null (nodes have 0, 1, 2 actual children)
  – All keys in a node’s right subtree are greater than its key and all in the left subtree are less
Which of the following is a legal BST?

A. I only
B. I and II only
C. II and III only
D. All of them
E. None of the choices above is correct
Which of the following is a legal BST?

A. I only
B. I and II only
C. II and III only
D. All of them
E. None of the choices above is correct
Implementing a BST

```java
public class BST<E extends Comparable<E>> { 
    Node root;

    private class Node { 
        Node left;
        Node right;
        E key;
    }
}
```
How can a BST implement contains?
How can a BST implement contains?

```java
boolean contains(E key) {
    Node curr = root;
    while(curr != null) {
        int cmp = key.compareTo(curr.key);
        if(cmp < 0) curr = curr.left;
        else if(cmp > 0) curr = curr.right;
        else return true;
    }
    return false;
}
```
Recursive implementation

```java
boolean contains(E key) {
    return contains_helper(root, key);
}

boolean contains_helper(Node curr, E key) {
    if (curr == null) return false;
    int cmp = key.compareTo(curr.key);
    if (cmp < 0) return contains_helper(curr.left, key);
    else if (cmp > 0) return contains_helper(curr.right, key);
    else return true;
}
```
Running time of tree operations

• The *depth* of a node is the number of edges between it and the tree’s root
• The *height* of a tree is the maximum number of edges from the root to a leaf
$O(\log n) \leq \text{height of n-node BST} \leq n-1$
A tree is balanced if its height is $O(\log n)$
How can a BST implement add?

• Follow contains until curr is about to become a null reference (until “about to fall off”)

• Replace that null with the new node
How can a BST implement add?

- Follow contains until curr is about to become a null reference (until “about to fall off”)
- Replace that null with the new node

What is the running time of a BST add?

A. $O(1)$
B. $O(\log n)$
C. $O(\text{tree height})$
D. $O(n)$
E. None of the above
How can a BST implement add?

• Follow contains until curr is about to become a null reference (until “about to fall off”)

• Replace that null with the new node

What is the running time of a BST add?

A. $O(1)$
B. $O(\log n)$
C. $O(\text{tree height})$
D. $O(n)$
E. None of the above
How can a BST implement remove?

• Begin by finding node with the key to remove

• Case 1: This node has 0 or 1 non-null children
How can a BST implement remove?

• Begin by finding node with the key to remove

• Case 1: This node has 0 or 1 non-null children
  – Remove node and replace it with child or null
How can a BST implement remove?

• Begin by finding node with the key to remove

• Case 1: This node has 0 or 1 non-null children
  – Remove node and replace it with child or null

How long does it take to move a subtree whose parent is removed?
A. O(1)
B. O(log n)
C. O(n)
D. Not enough information
E. None of the above
How can a BST implement remove?

• Begin by finding node with the key to remove

• Case 1: This node has 0 or 1 non-null children
  – Remove node and replace it with child or null

How long does it take to move a subtree whose parent is removed?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. Not enough information
E. None of the above
How can a BST implement remove?

- Begin by finding node with the key to remove

- Case 1: This node has 0 or 1 non-null children
  – Remove node and replace it with child or null

- Case 2: This node has 2 non-null children
How can a BST implement remove?

• Begin by finding node with the key to remove

• Case 1: This node has 0 or 1 non-null children
  – Remove node and replace it with child or null

• Case 2: This node has 2 non-null children
  – Replace the key with its *in-order successor* (next key in sorted order)
  – Remove the successor
How can we advance a Node reference to the next key?
How can we advance a Node reference to the next key?

• Case 1: Node has right subtree
  – Go one step right and then left as far as possible

• Case 2: Right subtree is null
  – Go up tree until reaching a node whose left child is the ancestor of your starting place
    (requires parent references in the Node)
  – If you try to go up from the root, there is no next key
What is the running time of remove in a balanced BST?

A. $O(1)$
B. $O(\log n)$
C. $O(\log^2 n) = O((\log n)^2)$
D. $O(n)$
E. None of the above
What is the running time of remove in a balanced BST?

A. $O(1)$
B. $O(\log n)$
C. $O(\log^2 n) = O((\log n)^2)$
D. $O(n)$
E. None of the above