Intro to Data Structures

Lecture #21 – BSTs (finale)
November 11, 2014

Mark Stehlik

Outline for Today

- HW6 due 11/18
- HW5 grading coming along…
- Some unfinished business – BST wrapup
  - an alternative add method
  - remove(value)
  - an alternate traversal
  - some vocabulary

Implementing a generic BST class

- What methods do we implement on a data structure?
  - ✓ constructor
  - ✓ isEmpty
  - ➢ add (today, an alternative implementation)
  - ✓ traversal [inOrder, the rest are variants]
  - ✓ size/count [O(n), for practice]
  - ✓ contains/find [O(?? log n), but O(n) if tree is not balanced]
  - ✓ toString [a rotated tree]
  - ➢ remove [discuss algorithm]

Add (an alternative implementation)

- Add looks a little different from, say, inOrder() or size() – where's the (simple) base case?
- You don't need to (and probably shouldn't) look ahead in a recursive method
- **But**, you need a way to communicate a change in the parameter across a method call
- **And**, the only way to do that is???

Add (an alternative implementation)

- How would I write a method to double an int?
- What if I wanted to only double odd int's?
- And how would I call that method to effect the change in a particular int variable?
- The programming idiom is

  x = changed(x);

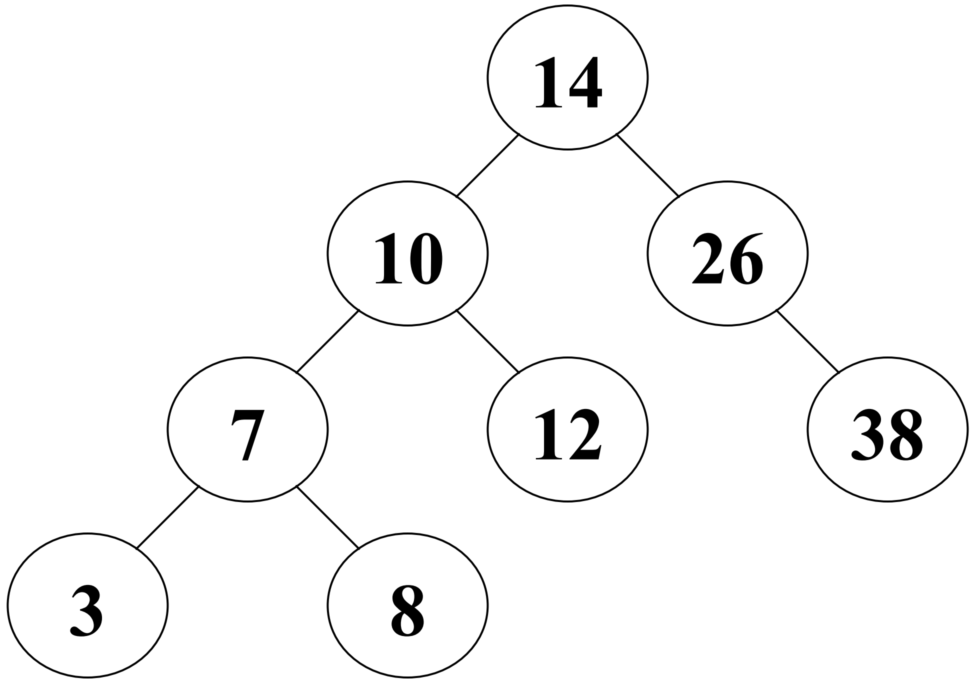
- Let's write an alternative add without lookahead using this idiom.

Tree traversal (revisited)

- All the traversals we've seen so far (pre-, in-, post-order) are *depth-first*, that is, they start at the root and go as far down a single branch as possible before *backtracking* up that branch and continuing forward as specified.
- Thinking about the array-based implementation of a tree (which is a good quiz question) made me think about a traversal we haven't discussed yet:
  - *breadth-first*

Tree traversal (revisited)

- Given the following BST



Tree traversal (revisited)

- And its array representation

	1	2	3	4	5	6	7	8	9	
	14	**10**	**26**	**7**	**12**		**38**	**3**	**8**	

- What would a natural traversal of this representation be?

Tree traversal (breadth-first)

- In a *breadth-first* search, you start at the root and look at all the nodes at the next level, and from each of those, the next level…
- How to do this?
- Queues!

```
enqueue the root
while (queue not empty)
        dequeue node  // print (visit)
        enqueue children
```

Remove

- Three cases to consider for the node to remove. It can have
  - no children (a leaf – just null that node)
  - one child (loop over it to the child)
  - two children (here's where it gets complicated):
    - could just delete value and reinsert left/right child, but…
    - instead, replace value in node with smallest value in right subtree or largest value in the left subtree, and then delete the node that contained that value (either 1-child or leaf)

Some "final" vocabulary

- *Perfect* binary tree
  - all internal nodes have 2 children
  - all leaf nodes are at same depth
  - a perfect binary tree of height k has $\mathbf{2^{k+1} - 1}$ nodes
  - alternatively, you can optimally place n nodes into a binary tree of minimum height ?? **log n**
- *Complete* binary tree
  - every level, except possibly the last is completely filled
  - last (deepest) level has all leaves as far left as possible