CSC148 winter 2018
mutating BSTs
week 9

Danny Heap
heap@cs.toronto.edu / BA4270 (behind elevators)
http://www.teach.cs.toronto.edu/~csc148h/winter/
416-978-5899

March 11, 2018
Outline

binary search tree operations

mutating binary search tree

term test #2
If node is the root of a “balanced” BST, then we can check whether an element is present in about \( \log n \) node accesses.

```python
def bst_contains(node: BTNode, value: object) -> bool:
    """
    Return whether tree rooted at node contains value.
    
    Assume node is the root of a Binary Search Tree
    """
    >>> bst_contains(None, 5)
    False
    >>> bst_contains(BTNode(7, BTNode(5), BTNode(9)), 5)
    True
    """
    # use BST property to avoid unnecessary searching
```
def insert(node: BTNode, data: object) -> BTNode:
    """
    Insert data in BST rooted at node if necessary, and return new root.
    Assume node is the root of a Binary Search Tree.
    """

>>> b = BTNode(8)
>>> b = insert(b, 4)
>>> b = insert(b, 2)
>>> b = insert(b, 6)
>>> b = insert(b, 12)
>>> b = insert(b, 14)
>>> b = insert(b, 10)
>>> print(b)
     14
      12
       10
      8
     6
    4
   2
deletion of value from BST rooted at node?

- what return value?
- what to do if node is None?
- what if value to delete is less than that at node?
- what if it’s more?
- what if the value equals this node’s value and...
  - this node has no left child
  - ... no right child?
  - both children?
algorithm...

# Algorithm for delete:
# 1. If this node is None, return that
# 2. If value is less than node.value, delete it from left child and
#      return this node
# 3. If value is more than node.value, delete it from right child
#      and return this node
# 4. If node with value has fewer than two children,
#      and you know one is None, return the other one
# 5. If node with value has two non-None children,
#      replace value with that of its largest child in the left
#      subtree and delete that child, and return this node
some recursive functions “write themselves” — you write down the base case and general case from a definition, and you have a program:

def fibonacci(n: int) -> int:
    """
    Return the nth fibonacci number, that is n if n < 2, or fibonacci(n-2) + fibonacci(n-1) otherwise.
    """
    pass
break our usual rule about expanding a branching recursive in order to see how much computation is spawned by fibonacci(29)

    if n < 2:
        return n
    else:
        return fibonacci(n-2) + fibonacci(n-1)
def fib_memo(n: int, seen: dict) -> int:
    ""
    Return the nth fibonacci number reasonably quickly.
    ""
    if n not in seen:
        seen[n] = (n if n < 2
                      else fib_memo(n-2, seen) + fib_memo(n-1, seen))
    return seen[n]
test coverage

- recursion on nested Python list
- recursion on class `Tree`
- recursion on class `BinaryTree`
- definitions for trees and binary trees, traversals (inorder, postorder, preorder, levelorder, binary search trees)