CSC148 winter 2018

binary trees
week 8

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

March 2, 2018

Outline

general trees continued...

binary trees

traversals

binary *search* trees

queues, stacks, recursion

You may have noticed in the last slide there were no recursive calls, and a **queue** was used to process a recursive structure in level order.

Careful use of a **stack** allows you to process a tree in preorder.

tree inheritance issues

one approach to BinaryTree would be to make it a subclass of Tree, but there are some design considerations

- any client code that uses Tree would be required not to violate the branching factor (2) of BinaryTree
- one way to achieve this would be to make Tree immutable: make sure there is no way to change children or value, and then have subclasses that might be mutable

we will take a different approach: a completely separate BinaryTree class

BTNode

Change our generic **Tree** design so that we have two named children, **left** and **right**, and can represent an empty tree with **None**

```
class BTNode:
    """
    A Binary Tree, i.e. arity 2.
    """

    def __init__(self, value: object,
                 left: Union["BTNode", None]=None,
                 right: Union["BTNode, None]=None) -> None:
        """
        Create BTNode self with value and children left and right.
        """
        self.value, self.left, self.right = value, left, right
```

special methods...

We'll want the standard special methods:

- `__eq__`
- `__str__`
- `__repr__`

contains

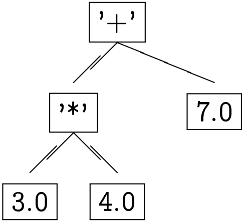
you've implemented contains on linked lists, nested Python lists, general Trees before; implement this function, then modify it to become a method

```
def contains(node: BTNode, value: object) -> bool:
    """
    Return whether tree rooted at node contains value.

    >>> contains(None, 5)
    False
    >>> contains(BTNode(5, BTNode(7), BTNode(9)), 7)
    True
    """
```

arithmetic expression trees

Binary arithmetic expressions can be represented as binary trees:



evaluating a binary expression tree

- there are no empty expressions
- if it's a leaf, just return the value
- otherwise...
  - evaluate the left tree
  - evaluate the right tree
  - combine left and right with the binary operator

Python built-in eval might be handy.

inorder

A **recursive** definition:

- visit the left subtree **inorder**
- visit this node itself
- visit the right subtree **inorder**

The code is almost identical to the definition.

preorder

- visit this node itself
- visit the left subtree in **preorder**
- visit the right subtree in **preorder**

postorder

- visit the left subtree in **postorder**
- visit the rightsubtree in **postorder**
- visit this node itself

level order

- visit root
- visit root's children
- visit root's grandchildren
- visit root's greatgrandchildren
- ...

definition

Add ordering conditions to a binary tree:

- data are comparable
- data in left subtree are less than node.data
- data in right subtree are more than node.data

why binary search trees?

Searchs that are directed along a single path are efficient:

- a BST with 1 one has height 1
- a BST with 3 nodes may have height 2
- a BST with 7 nodes may have height 3
- a BST with 15 nodes may have height 4
- a BST with n nodes may have height $\lceil \lg n \rceil$.

bst_contains

If node is the root of a "balanced" BST, then we can check whether an element is present in about lg n node accesses.

```
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
```

mutation: insert

```
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
```