Week 8: Binary Trees

Arnamoy Bhattacharyya
Announcements

1. Extra extra hours has been allocated for A2. Today, Monday and Tuesday (TAs, Professors)
2. Implement minimax in 2 ways (not 4 ways, not 10 ways)
3. Your Minimax should work for any game:
   a. Use the notion that each game is a subclass from `Game` and state is subclass of `GameState`
   b. Use hooks (get_possible_moves, is_over, who won? etc.)
   c. **DO NOT** use game specific assumptions
   d. Should work on any game
Agenda

1. Binary trees
2. BinaryTree Class (NOT a subclass of Tree)
3. Work on an exercise (handout)
4. How to implement contains as a module level function vs class level method
Recap

Tree level_order_visit

1. Using queue (no recursion)
2. Using recursion (no additional data structure)
Recursive level_order_visit

Idea:

1. Visit the root
2. Visit nodes at each level until there are no children
3. While visiting a level, use recursion to traverse from root until that level
Take home message

1. Some data structures can be used to solve recursive problems (stack or queue)
2. When you do not use any, there is an implicit data structure used: call stack
Binary tree

A tree with arity (maximum branching factor) 2
Binary tree

How many non-binary trees?
Binary Tree Design

1. Think of a *special* General Tree
2. Not a good idea to make a subclass of Tree
   a. Keep checking if client code violates arity
3. If subclass, make children immutable
   a. Then make mutable in some subclasses (general trees)
   b. Complicated!!!!!
4. We will redesign a BinaryTree Class
class BinaryTree:
    
    A Binary Tree, i.e. arity 2.
    
    def __init__(self, value, left=None, right=None):
        
        Create BinaryTree self with value and children left and right.
        @param BinaryTree self: this binary tree
        @param object value: value of this node
        @param BinaryTree|None left: left child
        @param BinaryTree|None right: right child
        @rtype: None
        
        self.value, self.left, self.right = value, left, right
Binary Tree Class

```python
class BinaryTree:
    def __init__(self, value, left=None, right=None):
        A Binary Tree, i.e. arity 2.

        Create BinaryTree self with value and children left and right.
        @param BinaryTree self: this binary tree
        @param object value: value of this node
        @param BinaryTree|None left: left child
        @param BinaryTree|None right: right child
        @rtype: None

        self.value, self.left, self.right = value, left, right
```

General Tree:

```python
self._value, self._children = value, children[:] if children is not None else []
```
Creating a Binary Tree

Same bottom up design:

```python
>>> childTree1 = BinaryTree(2, BinaryTree(3))
>>> childTree2 = BinaryTree(4)
>>> b = BinaryTree(1, childTree1, childTree2)
```
Special methods

__eq__
__repr__
__str__
Exercise 1

Handout

Design a function contains(tree, value)
Exercise 2

Designing this function as a class level method -- what modifications necessary?