Week 7 - Mutating Trees

Updated Oct 23, 8am

Now that we have some experience working with trees, let's talk about mutating them. There are two fundamental mutating operations that we'd want to perform on trees: insertion and deletion.

Today, we're going to implement the following function:

```
def delete_item(self, item):
    """ (Tree, object) -> NoneType
    Delete *one* occurrence of item from this tree.
    Do nothing if item is not in this tree.
    """
```

As usual, we're going to start with an easier method: one that simply removes the root of a tree.

```
def delete_root(self):
    """ (Tree) -> NoneType
    Delete root of this tree. Do nothing if this tree is empty.
    """
```

There are *many*, *many* ways of doing this. Here's one where we just pick the leftmost subtree (if there is one), and the make all of the other subtrees into subtrees of the leftmost one.

```
def delete_root(self):
    if len(self.children) == 0:
        # Base case when empty or just one node
        self.root = EmptyValue
    else:
        temp = self.subtrees[0]
        self.root = temp.root
        self.subtrees = temp.subtrees + self.subtrees[1:]
```

This maybe isn't very satisfying, because while the result certainly is still a tree, it feels like we've changed around a lot of the structure of the original tree just to delete a single element. You'll explore another way of doing in in your lab this week.

For now, let's see how to use `delete_root` to implement `delete_item`. Note that we've slightly changed what the method does, to make the recursion a bit smoother.

```
def delete_item(self, item):
    """ (Tree, object) -> bool

    Delete *one* occurrence of item from this tree.
    Do nothing if item is not in this tree.
    Return True if item was deleted, and False otherwise.
    """
```

```
    if self.is_empty():
        return False
    elif self.root == item:
        self.delete_root()
        return True
    else:
        for subtree in self.subtrees:
            # Try to delete item from current subtree
            # If it works, return!
            if subtree.delete_item(item):
                return True
        return False
```

Last problem

I realized after the lecture was over that I had forgotten one important detail. If we are successful in deleting the item from a subtree, and that subtree is now empty, then we should remove it from the list `self.subtrees`. Even though this seems like a minor detail, it's actually quite important: if we allow empty trees in `self.subtrees`, then it becomes harder to make sure `delete_root` is correct (look at the code carefully, and think about why).

To fix this problem, we can simply remove the subtree from `self.subtrees` right before returning. Note that in general it is **extremely dangerous** to remove items from a list as you iterate through it, but we avoid this problem because immediately after removing the subtree, we exit the list by returning `True`.

```
def delete_item(self, item):
    """ (Tree, object) -> bool

    Delete *one* occurrence of item from self.
    Do nothing if item is not in self.
    Return True if item was deleted, and False otherwise.
    """
    if self.is_empty():
        return False
    elif self.root == item:
        self.delete_root()
        return True
    else:
        for subtree in self.subtrees:
            # Try to delete item from current subtree
            # If it works, return!
            if subtree.delete_item(item):
                # If the subtree is now empty, remove it!
                if subtree.is_empty():
                    self.subtrees.remove(subtree)
                return True
        return False
```

David Liu (liudavid at cs dot toronto dot edu)
Come find me in BA4260
Site design by Deyu Wang (dyw999 at gmail dot com)