Mutable Trees: exam-level questions

> If you need help reviewing Mutable Trees, take a look at these resources:
>
> - Albert's and Robert's slides (https://docs.google.com/presentation/d/1_Z2YtfB-tjhD-9FO2KJSOs-wAeIjOfsKUzTz9beCOhw/edit)

We will be using the OOP implementation of `Tree`s from lecture, found here (http://www-inst.eecs.berkeley.edu/~cs61a/sp13/slides/25.py)

Each question has a "Toggle Solution" button -- click it to reveal that question's solution.

Trees

Question 1

Implement a function `equal` which takes two trees and returns `True` if they satisfy all the following conditions:

- The data of both Trees are equal
- The Trees have the same number of children
- All corresponding pairs of sub-Trees are also `equal`

def equal(t1, t2):

```
    """Returns Tree if t1 and t2 are equal trees.

    >>> t1 = Tree(1,
    ...           [Tree(2, [Tree(4)]),
    ...            Tree(3)])
    >>> t2 = Tree(1,
    ...           [Tree(2, [Tree(4)]),
    ...            Tree(3)])
    >>> equal(t1, t2)
    True
    >>> t3 = Tree(1,
    ...           [Tree(2),
    ...            Tree(3, [Tree(4)])])
    >>> equal(t1, t3)
    False
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 2

Implement a function `size` that returns the number of elements in a given Tree.

```
def size(t):
    """Returns the number of elements in a tree.

    >>> t1 = Tree(1,
    ...           [Tree(2, [Tree(4)]),
    ...            Tree(3)])
    >>> size(t1)
    4
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 3

Implement a function `height`, which returns the height of a Tree. The *height* of a tree is defined as the number of branches from the *root* to the bottom-most *leaf* of the Tree.

By definition, a leaf has a height of 0, since there are 0 branches from the root to the root.

```
def height(t):
    """Returns the height of the tree.

    >>> leaf = Tree(1)
    >>> height(leaf)
    0
    >>> t1 = Tree(1,
    ...           [Tree(2, [Tree(4)]),
    ...            Tree(3)])
    >>> height(t1)
    2
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 4

Implement a function `same_shape`, which takes two `Tree`s and returns True if the trees have the same structure, but not necessarily the same entries.

```
def same_shape(t1, t2):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 5

Implement a function `sprout_leaves`, which takes a `Tree` and a list of values. For every leaf of the `Tree`, mutate it so that it has a list of branches where the items are the elements in the list of values.

```
def sprout_leaves(t, vals):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 6

Implement a function `prune_leaves`, which takes a `Tree` and a list of values. For every leaf of the `Tree`, remove it if its entry is in the list of values.

```
def prune_leaves(t, vals):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Binary Search Trees

Question 7

Implement two functions, `max_bst` and `min_bst`, which take a binary search tree and returns the maximum and minimum values, respectively.

```
def max_bst(b):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

```
def min_bst(b):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 8

Implement the function `contains`, which takes a binary search tree and an item, and returns True if the binary search tree contains the item, and False if it doesn't.

```
def contains(b, item):
    """Returns True if B contains ITEM.

    >>> b1 = Tree(2,
    ...           Tree(1),
    ...           Tree(4, Tree(3)))
    >>> contains(b1, 4)
    True
    >>> contains(b1, 8)
    False
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 9

Implement the function `in_order`, which takes a binary search tree, and returns a list containing its items from smallest to largest. In computer science, this is known as an **in-order traversal**.

```
def in_order(b):
    """Returns the items in B, a binary search tree, in sorted
    order.

    >>> b1 = Tree(2,
    ...           Tree(1),
    ...           Tree(4, Tree(3)))
    >>> in_order(b1)
    [1, 2, 3, 4]
    >>> singleton = Tree(4)
    >>> in_order(singleton)
    [4]
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 10

Implement a function `nth_largest`, which takes a **binary search tree** and a number `n` (greater than or equal to 1), and returns the `nth` largest item in the tree. For example, `nth_largest(b, 1)` should return the largest item in `b`. If `n` is greater than the number of items in the tree, return None.

Hint: You can assume there is a `size` function that returns the number of elements in a given tree.

```
def nth_largest(b, n):
    """Returns the Nth largest item in T.

    >>> b1 = Tree(2,
    ...           Tree(1),
    ...           Tree(4, Tree(3)))
    >>> nth_largest(b1, 1)
    4
    >>> nth_largest(b1, 3)
    2
    >>> nth_largest(b1, 4)
    1
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution