CSC148 winter 2018

code tracing, oddities, review week 12

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

April 3, 2018

Outline

summary/review

hash table wrap-up

Our completed hash table has **expected** performance in $\Theta(1)$, for a dictionary with n entries. Now you know why.

bare bones

- 3 hours
- comprehensive
- no aid sheet, but there is an API

topics

object-oriented programming and design: lecture slides and example code, in-class exercises, weeks 1–2, lab #1, lab #2, and assignment #1

abstract data types, stacks, queues: lecture slides and example code, week 3, lab #3

linked lists: lecture slides and example code, in-class exercise, week 4, lab #4

modularity, functional programming idiom lecture slides and example code, week 5

reading, writing recursion on nested lists: lecture slides and example code, in-class exercise, week 6, lab #5

more topics

recursion on general trees: lecture slides and example code week 7, in-class exercises on contains and leaf, lab #6

recursion on binary trees: lecture slides and example code week 8, lab #7

binary search trees, insertion, deletion, mutation: lecture slides and example code week #9, in-class exercise

efficiency, recursion, recursive structures: lecture slides and example code week #10, lab #8

big-Oh, big-Theta, hash table: lecture slides and example code week #11

hash table, tracing and traps: lecture slides and example code week #11

office hours: Mondays 3:00–5:00 p.m., April 9, 16, 23

old exam question...

```
(8 marks out of 54...)
def swap_even(t, depth=0):
    """ Swap left and right children of nodes at even depth.
    ... # stuff omitted for space...
    >>> b1 = BinaryTree(1, BinaryTree(2, BinaryTree(3)))
    >>> b4 = BinaryTree(4, BinaryTree(5), b1)
    >>> print(b4)
        1
            2
                3
    4
        5
    >>> swap_even(b4)
    >>> print(b4)
        5
    4
        1
                3
            2
    """
```

another old question...

```
(8 marks out of 54...)
def pathlength_sets(t: Tree) -> None:
    """
    Replace the value of each node in Tree t by a set containing all
    path lengths from that node to any leaf.  A path's length is the
    number of edges it contains.

    >>> t = Tree(5)
    >>> pathlength_sets(t)
    >>> print(t)
    {0}
    >>> t.children.append(Tree(17))
    >>> t.children.append(Tree(13, [Tree(11)]))
    >>> pathlength_sets(t)
    >>> print(t)
    {1, 2}
       {0}
       {1}
          {0}
    """
```

old exam(ple)

```
(8/73)
def TPBT(root: Union[BTNode, None]) -> bool:
    """ (BTNode or None) -> (int, bool)

    Return a tuple containing: (1) the height of the tallest
    perfect binary tree within the tree rooted at root,
    and (2) whether or not that tallest perfect binary tree
    occurs at the root itself.

    The empty tree is a perfect binary tree of height 0 and
    a leaf is a perfect perfect binary tree of height 1.
    """
```

yet another exam(ple)

(8/73)

```
def unique_paths(t: Tree) -> bool:
    """ Return whether there is a unique path from t to each
    of its descendents.

    Assume that two Trees are the same if they have the same
    memory address, that is id(t1) == id(t2)

    >>> t1 = Tree(1)
    >>> t2 = Tree(2)
    >>> t3 = Tree(3, [t1, t2])
    >>> unique_path(t3)
    True
    >>> t4 = Tree(4, [t3, t1])
    >>> unique_path(t4)
    False
    >>> t3.children.append(t3)
    >>> unique_path(t3)
    False
    """
```