

Binary Search Tree Functions

Homework #8

By

John Robertson

CS 303 Algorithms and Data Structures

October 18, 2019

1. Problem Specification

This assignment requires implementing a binary search tree and some functions associated with it, as well as testing it on a real-world application.

2. Program Design

This program has two files: a file with a node and binary search tree class (including), and a driver file.

The following steps were required to develop this program:

- Implement the Node and Tree classes
- Implement `tree_insert()`, `inorder_tree_walk()`, and `tree_search()`
- Debug the above until they worked, writing a mini-driver within the file
- Write the driver
- Debug the driver

I made it an option to build a tree on the large sorted file.

3. Testing Plan

I first had to test `tree_insert()`, then `inorder_tree_walk()`, then `tree_search()`. `inorder_tree_walk()` required the most debugging. I used small inputs for testing, knowing that I'd have bigger inputs to test on from `input.dat` and `UPC.csv`. Next, all I had to do was test my driver code itself in `driver.py` for correctness using `input.dat`.

4. Test Cases

You should run `bst.py` and then `driver.py`. `bst.py` tests `inorder_tree()` on no elements, a random list, a sorted list, and a reverse sorted list: all the while, `tree_insert()` is by necessity tested. `tree_search()` is tested on a random item that is guaranteed to be inside it and an item that is guaranteed to not be inside it. `driver.py` tests on `tree_insert()` and `inorder_tree_walk()` on `input.dat` before testing the time that it takes to use `tree_insert()` on a shuffled `UPC.csv`. It then tests three manual searches on keys from `input.dat` on the tree with the items from `UPC.csv` before testing every key from `input.dat`. There is also an optional test case for inserting all the *sorted* data from `UPC.csv` in a tree, but this is obviously not recommended.

5. Analysis and Conclusions

Searched Key	Expected Outcome (s)	Time Took (s)	Done as expected?
79	0.001	0.0001	No (Faster)
93	0.001	0.00007	No (Faster)
123	0.001	0.0001	No (Faster)
161	0.001	0.0001	No (Faster)
2140000070	0.01	0.0001	No (Faster)

2140118461	0.01	0.0001	No (Faster)
2144209103	0.01	0.0001	No (Faster)
2144622711	0.01	0.00009	No (Faster)
2147483647	0.01	0.00008	No (Faster)
2158242769	0.01	0.0001	No (Faster)
2158561631	0.01	0.00008	No (Faster)
2158769549	0.01	0.00008	No (Faster)
2160500567	0.01	0.0002	No (Faster)
2172307284	0.01	0.00007	No (Faster)
2177000074	0.01	0.0002	No (Faster)
2184000098	0.01	0.00008	No (Faster)
2187682888	0.01	0.0001	No (Faster)

```

nfs.py  driver.py  input.d  ...  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
30  shuffle(inputs)
31  start = perf_counter()
32  for i in inputs:
33      j = (int(i[0]), i[1] + i[2])
34      tree.tree_insert(*j)
35  print("\nThat took {} seconds.".format
(perf_counter() - start))
36
37  #Can it search though?
38  print("\tThree manual searches to test if
this works:")
39  target = tree.tree_search(79)
40  print(target.val)
41  target = tree.tree_search(93)
42  print(target.val)
43  target = tree.tree_search(2160500567)
44  print(target.val)
45  print("\tNow to do this iteratively:")
46  for i in keys:
47      start = perf_counter()
48      target = tree.tree_search(int(i))
49      print(target.key, end=" ")
50      print(target.val, end=" ")
51      print("\t{} seconds.".format
(perf_counter() - start))
52
53  # On the sorted data
54  bad_idea = input("Press '1' if you want to
build a BST on the sorted inputs (please
don't do this).")
55  if bad_idea == "1":
56      del tree
57      tree = Tree()
58      inputs.sort()
79, INDIANA LOTTO
0.00010249999999996723 seconds.
93, tree 700w
6.85999999999964117e-05 seconds.
123, Wrsi Riversound cafe cd
0.00011220000000022878 seconds.
161, Dillons/Kroger Employee Coupon ($1.25 credit)
9.640000000032956e-05 seconds.
2140000070, Rhinestone Watch
8.0399999999986940e-05 seconds.
2140118461, ""v"": Breakout/The Deception VHS Tape"
0.00016250000000006537 seconds.
2144209103, VHSIntorena - Tiger Shark
0.00012579999999992876 seconds.
2144622711, Taxi : The Collector's Edition VHS
8.5100000000061524e-05 seconds.
2147483647, Toshiba 2805 DVD player
7.5999999999996498e-05 seconds.
2158242769, 288/1.127GREEN SUGAR COOKIE54276
0.000144800000000016702 seconds.
2158561631, HOT COCOA W/BKWK
7.76000000000011e-05 seconds.
2158769549, njhjhngjfhjbgkj
7.160000000000605e-05 seconds.
2160500567, 2.25 oz (64g)Dollar Bar Rich Raspberry
0.0001571000000000211 seconds.
2172307284, Mixed seasonal flower bouquet
7.4600000000047984e-05 seconds.
2177000074, 4 way 13 AMP Extension Lead (Wilkinson UK)
0.000175300000000043344 seconds.
2184000098, 21 ozChristopher's Assorted Fruit Jellies
8.3599999999996140e-05 seconds.
2187682888, fairway
0.000107700000000043612 seconds

```

```

bst.py  ...  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  Code
class Node:
    def __init__(self, key=None, val=None):
        self.key = key
        self.val = val
        self.left = None
        self.right = None
        self.p = None

class Tree:
    def __init__(self):
        self.root = None

    def tree_insert(self, z_key, z_val):
        z = Node(z_key, z_val)
        y = None
        x = self.root
        while x != None:
            y = x
            if z.key < x.key:
                x = x.left
            else:
                x = x.right
        z.p = y
        if y == None:
            self.root = z
        elif z.key < y.key:
            y.left = z
        else:
            y.right = z

    def inorder_tree_walk(self):
        l = []
        def itw(x, y):
Test on no elements
Nothing was printed
List form: [484, 255, 356, 212, 60, 127, 153, 276, 111, 58, 250, 368]
Tree form:
Key: 58, Value: 58
Key: 60, Value: 60
Key: 111, Value: 111
Key: 127, Value: 127
Key: 153, Value: 153
Key: 212, Value: 212
Key: 250, Value: 250
Key: 255, Value: 255
Key: 276, Value: 276
Key: 356, Value: 356
Key: 368, Value: 368
Key: 484, Value: 484
Sorted list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
As a tree:
Key: 0, Value: 0
Key: 1, Value: 1
Key: 2, Value: 2
Key: 3, Value: 3
Key: 4, Value: 4
Key: 5, Value: 5
Key: 6, Value: 6
Key: 7, Value: 7
Key: 8, Value: 8
Key: 9, Value: 9
Key: 10, Value: 10
Key: 11, Value: 11
Reverse-sorted list
[11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
As a tree:

```

The code turned out faster than expected. Obviously, it is slow on the sorted list (I was unwilling to see how long this would take, given past experience with code that takes a long time to run), but the code looks up based on keys very quickly, because the search time is at worst only the log of the height of the tree. It is clear that binary search trees are one fast option for storing large amounts of data.

6. References

Dr. Unan's slides