

CMPUT 275 Wi18 - INTRO TO TANGIBLE COMPUT II

Combined LBL Wi18

Exercise 5: AVL Trees

The purpose of this exercise is to have you exploit the fact that AVL trees are balanced to quickly answer certain queries that dictionaries based on hash tables cannot.

You will add a new method to the class `AVLDict` in the file `avl_dict.py` ([link](#)) we developed in class.

```
def ith_key(self, index: int):
    """
    Returns the key that would be at the given index in the sorted list of all keys of the tree.
    Raises an IndexError if the index is out of range (i.e. < 0 or >= len(self)).

    Running time: O(log n) where the tree has n items.
    """
```

That is, you should not actually get the list of all items and return the one at the given index. That approach would run in $O(n)$ time (**aside**: this is a helpful approach for testing your solution). Rather, you will need to slightly modify the internal implementation of the AVL tree to support this function efficiently.

An example of how a working solution could be used.

```
>>> from avl_dict import AVLDict
>>> d = AVLDict()
>>> d["arrival"] = "hello"
>>> d["departure"] = "goodbye"
>>> d.ith_key(0)
"arrival"
>>> d.ith_key(1)
"departure"
>>> d["cancel"] = "cancelled"
>>> d.ith_key(1)
"cancel"
>>> d.ith_key(2)
"departure"
>>> del d["arrival"]
>>> d.ith_key(0)
"cancel"
```

Requirements

- For each `AVLNode`, also maintain an integer `num`. This should be such that `node.num` is the number of nodes in the subtree rooted at `node`. For example, if `node` is a leaf node, then `node.num` should be 1; if `node` is one of the "empty" nodes with `key == None` then `node.num` should be 0.

- Implement the new `ith_key()` function so it runs in $O(\log n)$ time.
- Remove the `import bstviz` statement at the top before you submit.



Hints

- You can maintain this number in a very similar way to how the "height" of an AVLNode was maintained through insertion, deletion, and rotations.
Don't forget to set the "num" value of the node properly when inserting a new key during update!
- Now, assuming each node is correctly storing its corresponding value "num", how can you search from the root toward the correct node by comparing "num" with "index"? Draw some examples to help you see how you could do it.


Your solution will be eligible for partial credit if it correctly maintains the "num" value of an AVLNode, even if it does not correctly implement the `ith_key` function. Your final solution must run in $O(\log n)$ time to be eligible for full marks.

Submit only your modified version of `avl_dict.py`. Do not zip it. As usual, adhere to the code submission guidelines and practice clean style with appropriate comments.

Submission status

| | |
|---------------------|--|
| Attempt number | This is attempt 1 (1 attempts allowed). |
| Submission status | Submitted for grading |
| Grading status | Graded |
| Due date | Monday, 12 March 2018, 11:55 PM |
| Time remaining | Assignment was submitted 1 day 2 hours early |
| Last modified | Sunday, 11 March 2018, 9:09 PM |
| File submissions | <div> <code>avl_dict.py</code> </div> <div>Export to portfolio</div> |
| Submission comments | ► Comments (0) |

Feedback

| | |
|-----------|---|
| Grade | 55.00 / 100.00 |
| Graded on | Saturday, 7 April 2018, 6:32 PM |
| Graded by |  Somjit Nath |

2018-04-14, 1:01 p.m.

