Tree-Based Data Structures

Martin Blom[*] and Jonatan Langlet[†]

Karlstad University, Sweden

November 7, 2019

1 Introduction

A tree is a collection of items that are hierarchically ordered in an ancestor-descendent relationship. A root node designates the start of the tree, and it may have zero ore more children where each such child node is recursively defined as a tree down towards the leaves. A node is said to be a parent to its immediate children, and children that share the same parent are referred to as siblings. The height of an empty tree is zero, and similar definitions include depth and level. Get yourself familiar with additional terminology, such as breadth-first search, inorder traversal, and perfect trees.

[*] `martin.blom@kau.se`
[†] `jonatan.langlet@gmail.com`

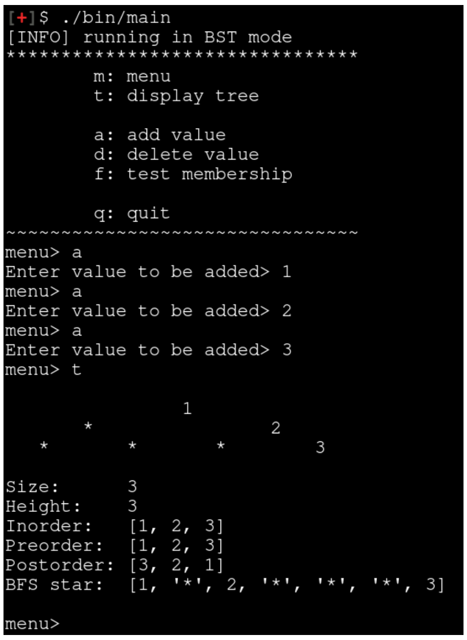
1.1 Learning Goals and Preperation

Throughout the lab you will learn the *principles* of two lexicographically ordered tree data structures. In part one, you will start from some Python code and implement the remainder of a Binary Search Tree (BST). In part two, you will extend it to an Adelson-Velsky-Landis (AVL) tree. Given that the lab uses Python, you have the opportunity to rapidly learn a new programming language if not familiar already. We encourage you to spend a little bit of time to get started with Python *before the lab*. For example, take a look at the overview by Derek Banas [1] and study the start code briefly.

2 Task

2.1 Getting Started

The start-code is available on Canvas. Alternatively, you can download it from the terminal: `git clone https://git.cs.kau.se/rasmoste/dvgb03.git`. Next, run `./bin/main --help` to show available options and refer to the `README.md` file if necessary. A running example is shown in Figure 1. The mode can be changed from BST to AVL as follows: `./bin/main -m avl`.



```
[+]$ ./bin/main
[INFO] running in BST mode
********************************
        m: menu
        t: display tree

        a: add value
        d: delete value
        f: test membership

        q: quit
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
menu> a
Enter value to be added> 1
menu> a
Enter value to be added> 2
menu> a
Enter value to be added> 3
menu> t

                1
       *                2
   *       *       *        3

Size:      3
Height:    3
Inorder:   [1, 2, 3]
Preorder:  [1, 2, 3]
Postorder: [3, 2, 1]
BFS star:  [1, '*', 2, '*', '*', '*', 3]

menu>
```

Figure 1: Running example with default settings.

2.2 BST

We already implemented the start of a BST in `src/bst.py`. New values can be added in BST-order, and the stored values can be listed from a preorder traversal. Your task is to implement the remaining methods: `is_member`, `size`, `height`, `inorder`, `postorder`, `bfs_order_star`, and `delete`. Use the basic tree operations that were inherited from `src/bt.py`, such as `is_empty` and `cons`. Add submethods where appropriate to keep *all* methods short. As a rule of thumb, a reasonably large (sub)method is at most a few lines.

2.2.1 Strict implementation criteria

- The height of an empty tree is zero, and the height of a single-node tree is one. Be aware that the height can be defined differently.
- The delete function must not make the tree more unbalanced than prior to the delete. If the node to be deleted has two children, replace it with the smallest (largest) node in the right (left) subtree. If it is possible to replace with a node from either subtree, always pick left.
- The public interface of each TODO method must not be modified. In other words, keep the method name and number of arguments intact.

To ensure that you understand these criteria better, inspect the output that was produced by our reference implementation (see `bst.txt`).

2.3 UI

The terminal-based UI is defined in `src/ui.py`. Your task is to implement the remaining method `show_2d`, which prints the structure and the content of the tree based on the output of `bfs_order_star`. An example of a "pretty 2D print" is shown in Figure 1. Your solution need not be identical in terms of spaces, but it must scale proportionally as the number of levels are increased. Unsure if your solution is sufficient? Show it to a lab supervisor.

2.4 AVL

The start of an AVL tree is defined in `src/avl.py`. It inherits all BST methods in `src/bst.py`, which means that you need not (re)implement methods such as `height` and `size`. Your task is to implement and apply the following methods: `balance`, `slr`, `srr`, `dlr`, and `drr`. One way to do this is by overriding the `add` and `delete` methods, but maybe there is a better solution?

2.4.1 Strict implementation criteria

- The balance function fixes at most *one violation* of the AVL property. In other words, it runs one or none of `slr`, `srr`, `dlr`, and `drr`.
- Rotate the tree as little as possible when fixing an AVL violation. For example, if a single rotation is sufficient you should not double rotate.
- The public interface of each TODO method must not be modified. You may overide any existing method without changing its arguments.

To ensure that you understand these criteria better, inspect the output that was produced by our reference implementation (see `avl.txt`).

2.5 Points Awarded

You will be awarded up to six points based on correct implementation and good code quality. Back-end methods in `src/bst.py` and `src/avl.py` are corrected automatically by inspecting their output. This is the reason why you cannot change any public interface. In contrast, front-end methods and code quality are corrected manually by a lab supervisor. Notably we only evaluate a subset of methods for good code quality. These methods are selected randomly and announced after the submission deadline.

Table 1: Score overview; points are awarded iff all criteria are met.

Task	**Points**
`is_member`, `height`, `size`	0.5
`inorder`, `postorder`, `bfs_order_star`	0.5
`show_2d`	1
`delete`	2
`balance`	1
code quality	1

3 Submission

You may work alone or in pairs of two. Use the provided start-code and follow all criteria as described in this specification. Submit the following on Canvas no later than **December 6, 2019**:

- Names of all group members in the comment field.
- `ui.py`, `bst.py`, and `avl.py` without any zipping.

Acknowledgements

This lab was initially created by Donald F. Ross. The current start-code and specification was repackaged by Rasmus Dahlberg and Martin Blom.

References

[1] D. Banas. Python programming, November 2014. accessed 2019-09-27.