

Prof. Dr. Goran Glavaš,

M.Sc. Fabian David Schmidt

M.Sc. Benedikt Ebing

Lecture Chair XII for Natural Language Processing, Universität Würzburg

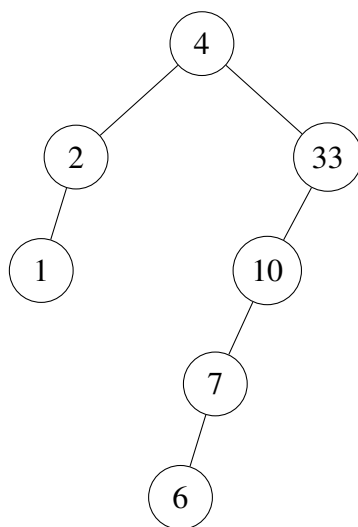
1. Exercise for “Algorithmen, KI & Data Science 1”

1.12.2023

You may hand in scans of **readable**, hand-written solutions for the below exercises to possibly simplify the submission of your solutions. The submission nevertheless **must** be concatenated to a single PDF file.

1 Binary Search Trees

1. Draw the binary search tree resulting from inserting the array $[4, 33, 10, 2, 1, 7, 6]$ in the given order starting from an empty tree.



2. You have a drink with a curious stranger at Nachtwächter, who claims the following: if you insert the same elements in different order into an initially empty binary

search tree, the resulting binary search trees differ. Illustrate in at most two sentences based on an example as to why (or why) not this claim is true.

■ This claim is true and easy to see if you insert $[1, 2, 3]$ in varying orders.

3. Assume that we build a binary search tree that comprises the numbers 1 to 1000. We are now searching for the number 412. Which of the following sequences **cannot be** the order of nodes examined? If so, why not?

- (i) 998, 14, 36, 512, 247, 309, 499, 412
- (ii) 565, 501, 181, 673, 500, 480, 427, 412
- (iii) 2, 837, 547, 137, 230, 453, 402, 412
- (iv) 765, 699, 643, 555, 270, 315, 411, 412
- (v) 666, 245, 598, 301, 572, 365, 500, 41

-
- (i) works
 - (ii) We traverse left sub-tree of 565 and run into 673 which can't be as left sub-tree **only** can comprise nodes with values smaller than the root.
 - (iii) works
 - (iv) works
 - (v) works

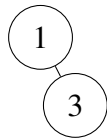
2 Balanced Trees (AVL Trees)

1. Enter the keys $[1, 3, 5, 6, 6, 4, 2]$ in order into an initially empty AVL-tree. Write down the tree after each step of insertion and rotation.

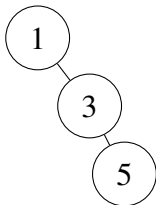
(1) Insert 1



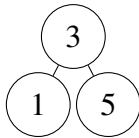
(2) Insert 3



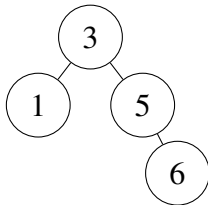
(2) Insert 5



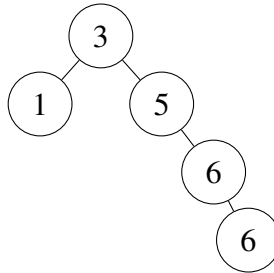
(3) Rotate 1 to left.



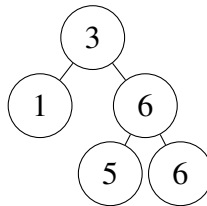
(4) Insert 6.



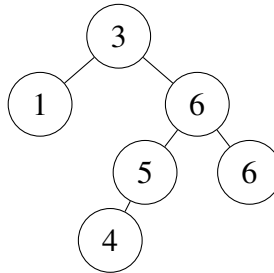
(5) Insert 6



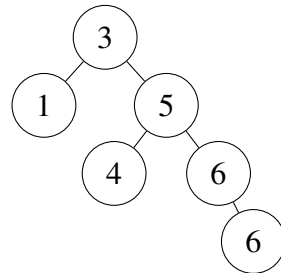
(6) Rotate 5 to left



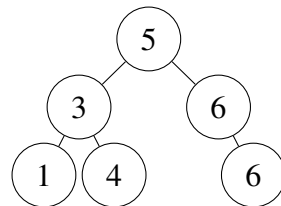
(7) Insert 4



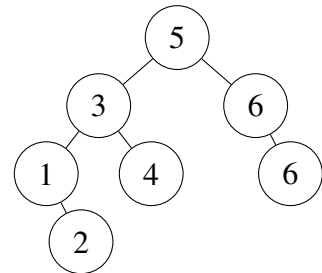
(8) Rotate 6 to right



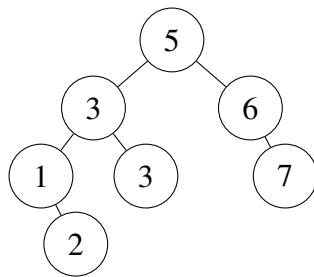
(9) Rotate 3 to left



(10) Insert 2



2. Delete the node returned by the search for key 3 from the below AVL-tree. Write down the tree after each step of deletion and rotation.



3 Programming Exercise for Binary Search & AVL trees

In this exercise, you will be given a `Node` implementation that stores its value (`self.value`), a pointer to the left child (`self.left`), a pointer to the right child (`self.right`), and possibly a pointer to the parent node (`self.parent`). You are supposed to implement the functions `get_min`, `get_max`, `search`, `is_balanced`, and `is_binary_search_tree`.

The implementation of AVL-tree insertion and deletion is optional. As an incentive, should you decide to implement these functions and do so correctly, your submission will automatically be awarded 2 points for this exercise. Why? To appropriately implement `insert` and `delete`, you will require implementing previous functions and fundamentally have acquired the key learnings of this exercise as the other tasks are practically a subset of your implementation.