Week 12 Applied Sheet

Objectives: The applied sessions, in general, give practice in problem solving, in analysis of algorithms and data structures, and in mathematics and logic useful in the above.

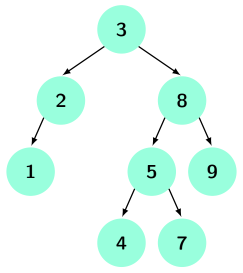
Instructions to the class: You should actively participate in the class.

Instructions to Tutors: The purpose of the applied class is not to solve the practical exercises! The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

Supplementary problems: The supplementary problems provide additional practice for you to complete after your applied class, or as pre-exam revision. Problems that are marked as **(Advanced)** difficulty are beyond the difficulty that you would be expected to complete in the exam, but are nonetheless useful practice problems as they will teach you skills and concepts that you can apply to other problems.

Problems

Problem 1. Insert 6 into the following AVL tree and show the rebalancing procedure step by step.



Problem 2. Recall the algorithm for deleting an element from a binary search tree. If that element has no children, we do nothing. If it has one child, we can simply remove it and move its child upwards. Otherwise, if the node has two children, we replace the node we are deleting with its *successor* (and if the successor has one child, move the successor child upwards to be a child of the previous parent of the successor). This algorithm works because of the following fact: the successor of a node with two children has no left child. Prove this fact.

(a) First prove that the successor of a node x with two children must be contained in the right subtree of x.

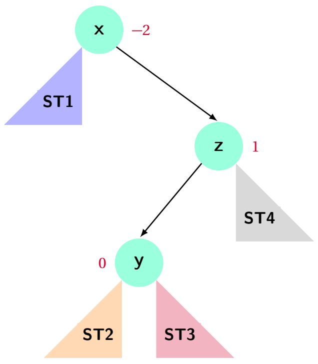
(b) Use this fact to prove that the successor of x cannot have a left child.

Problem 3. We know that when inserting an element into a binary search tree, there is only one valid place to put that item in the tree (and that is an important property for the correctness of binary search trees). Let's prove this fact rigorously. Let T be a binary search tree and let x be an integer not contained in T. Prove that exactly one of the following statements is true:

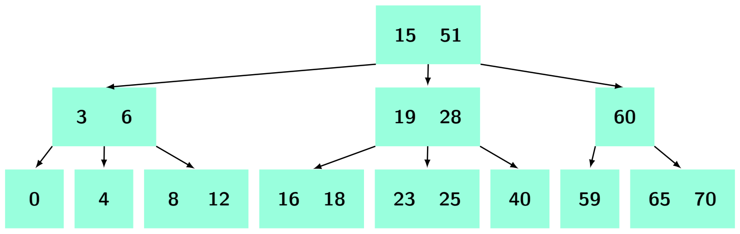
- Statement 1: The successor of x has no left child.
- Statement 2: The predecessor of x has no right child.

That is, prove that it cannot be the case that both of these statements are true simultaneously, nor that both of these statements are false simultaneously. This implies that there is a unique insertion point for x since upon insertion x must either become the left child of its successor or the right child of its predecessor.

Problem 4. Consider the binary search tree shown below, with balance factors as indicated. Assume that the subtrees **ST1**, **ST2**, **ST3** and **ST4** are AVL trees. Show that performing the rotation algorithm for the right-left case results in an AVL tree.



Problem 5. Consider the following 2-3 Search Tree:



(a) Visualise the state of the tree after sequentially performing each of the following actions:

  i. Insert key 5

  ii. Insert key 62

  iii. Delete key 59

  iv. Insert key 10

  v. Delete key 40

  Note: We are only covering deletion if the key is a leaf node.

(b) Convert the resulting 2-3 Search Tree from part (a) into the equivalent Left-Leaning Red-Black Tree.

Problem 6. Sequentially perform the following operations on an empty Left-Leaning Red-Black-Tree. For each operation; (1) draw the resulting Red-Black-Tree, (2) the equivalent 2-3 Search Tree, and (3) state the black height of the Red-Black-Tree

(a) Insert key 40

(b) Insert key 20

(c) Insert key 5

(d) Insert key 1

(e) Insert key 32

(f) Insert key 64

(g) Insert key 13

(h) Insert key 5

(i) Insert key 38

(j) Insert key 36

Problem 7. You are given a set of distinct keys $x_1, x_2, ..., x_n$.

(a) Design an algorithm that creates a binary search tree of minimal height containing those keys in $O(n\log(n))$ time.

(b) Prove that your algorithm produces a BST of height at most $\log(n)$.

(c) Prove that the fastest algorithm for this problem in the comparison-based model has time complexity $\Theta(n\log(n))$.

Supplementary Problems

Problem 8. In the Seminar we claimed that AVL trees are good because the balance property guarantees that the tree always has height $O(\log(n))$. Let's prove this.

(a) Write a recurrence relation for $n(h)$, the **minimum** number of nodes in an AVL tree of height h. [Hint: It should be related to the Fibonacci numbers]

(b) Find an exact solution to this recurrence relationship in terms of the Fibonacci numbers[1]. [Hint: Compare the sequence with the Fibonacci sequence, find a pattern, and then prove that your pattern is right using induction]

(c) Prove using induction that the Fibonacci numbers satisfy $F(n) \geq 1.5^{n-1}$ for all $n \geq 0$

(d) Using (a), (b), and (c), prove that a valid AVL tree with n elements has height at most $O(\log(n))$

[1]For this problem, index the Fibonacci numbers from zero with the base case $F(0) = F(1) = 1$.