Universität Freiburg
Institut für Informatik
Dr. F. Wei
T. Hornung

Georges-Köhler-Allee, Geb. 051
D-79110 Freiburg i. Br.
Freiburg, 4. Mai 2009

Theory I - Exercise sheet 2

Submission due Tuesday, May 19

Exercise 1: ($0 + 3 + 1$ points) AVL trees

Discuss whether an AVL tree t with F_k leaves (where F_k is a Fibonacci number and $k \geq 7$) has the *internal path length* $l(t) \leq F_k \cdot (k - 4)$. Proceed as follows:

a) (Discussion Topic) Try to determine the maximum height h of an AVL tree with F_k leaves. Has an AVL tree with maximum height in this case also the maximum internal path length?
In the following part, we assume that this is proved.

b) Using induction, show the above inequality.
Hint: Be sure you start with the right base case(s). For the induction step use the definition of the internal path length and your knowledge about the number of internal nodes in the tree.

c) What does this tell us about the *average search path length* $D(t)$ in such a tree (in terms of its height)?

Exercise 2: ($2 + 2 + 1$ points) AVL trees

a) Insert sequentially the keys $1, 2, ..., 7$ into the initial empty AVL tree. Show after each insertion the originated tree. Which structure has the originated AVL tree?

b) Insert further the keys $8, 9, ..., 15$ into the constructed tree and show the resulting tree.

c) Do you have a presumption, which structure the insertion operation for AVL trees produces, if $2^n - 1$ keys were inserted into the initial empty tree?

Exercise 3: (5 points) Fibonacci trees
Given a fibonacci tree t_h with height h and $Fib(h + 2)$ leaves.
Show the fibonacci tree t_5 and the originated tree, if the delete operation for AVL trees is applied to the root of t_5.
(t_h can be defined inductive: A variant of a binary tree where a tree t_h ($h > 1$) has a left subtree t_{h-1} and a right subtree t_{h-2}. The fibonacci tree t_0 has no nodes, and tree t_1 has 1 node.)

Exercise 4: ($4 + 2$ points) Representation of trees
In a *k-ary tree* each node may have up to k child nodes. In order to handle such trees, two different representations are commonly used:

- In the *standard* representation, each node stores the pointers to its children in an array **children[]** of size k, plus a variable **rank** that indicates the actual number of children.
- In the *child-sibling* representation, we also have the **rank** variable indicating the number of children. However, each node stores only one pointer **child** to its leftmost child. Each node also has a pointer **sibling** pointing to its right sibling (or $null$, if there is no such sibling).

a) For both representations, describe (in your words, pseudo-code, or Java) efficient methods for

   - accessing the i-th child (from the left) of a given node N
   - inserting a new child for node N (assuming N has less than k children before the insertion)
   - deleting a given child node C of node N

   For each method, determine the worst-case time complexity (lowest asymptotic bound in O-notation). Explain your answers.

b) Can we improve any of the worst-case times in the child-sibling representation by using *doubly-linked lists* (i.e. each node contains pointers to its left and right sibling)?