Assignment 7 – Selected Model Answers

Exercise 1.

(i) Describe how to determine the minimum and the maximum of the elements in an AVL tree.

(ii) Given a binary tree with labelled vertices describe how to check, if the tree is an AVL tree.

Solution.

(i) To find the minimum we always have to follow the left sucessor pointer until we reach a node v without a left successor. The label of v is the minimum. By definition of a BST all values in the right successor tree of v are larger. Also, if the path from the root is $v_0, \dots, v_k$ with $v_k = v$ then the label of v is smaller than the labels of all v_i with $i < k$, and in the right successor tree of v_i we can only find even larger values.

Analogously, to find the maximum we always have to follow the right sucessor pointer until we reach a node v without a right successor. The label of v is the maximum. By definition of a BST all values in the left successor tree of v are smaller. Also, if the path from the root is $v_0, \dots, v_k$ with $v_k = v$ then the label of v is larger than the labels of all v_i with $i < k$, and in the leftt successor tree of v_i we can only find even smaller values.

(ii) We proceed recursively using an algorithm that takes the binary tree as input and either returns the height in case the tree is an AVL tree or otherwise returns -1. Thus, in case of an empty tree we return 0, and in case of a tree with exactly one vertex we return 1.

For an arbitrary non-empty tree call the algorithm recursively for the left and right successor trees. Let the returned values be h_ℓ and h_r. If both values are ≥ 0 and in addition $-1 \leq h_r - h_\ell \leq 1$ holds, we have an AVL tree and return $\max\{h_\ell, h_r\}$. Otherwise return -1.

Exercise 2. Consider the following retrieval operations on AVL trees.

(i) Describe an operator that exploits a *depth-first search* through the tree and returns the elements in the order of this search.

(ii) Define an operator that exploits a *breadth-first search* through the tree and returns the elements in the order of this search.

(iii) Describe how to determine the median of the elements in an AVL tree.

(iv) Use one of the operators in (i) or (ii) to define a function *range* on AVL trees, which for arguments x and y returns those elements in the tree that satisfy $x \leq e \leq y$.

Solution.

(i) Proceed recursively. For an empty tree return the empty list []. For an arbitrary non-empty tree with root v call the operator on the left successor tree yielding a list ℓ_ℓ and on the right successor tree yielding a list ℓ_r. Then return the concatenated list $\ell_\ell + [\ell(v)] + \ell_r$, where $\ell(v)$ is the label of v.

(ii) For an empty tree return the empty list []. For a non-empty tree proceed using a working list L, which initially contains only the root v, and an output list O, which initially is empty. Then iterate until the working list becomes empty. Remove the first element v from L, append its label to O, and append first the left successor, then the right successor of v to O.

(iii) Use the algorithm in (i) to determine a list L of the elements in the AVL tree. For each node v all labels in the left successor tree of v appear before the label of v, and this appears before all labels in the right successor tree of v. Hence the list L is sorted. We obtain the median as the k'th element of this list L for $k = \lceil n/2 \rceil$, for which a linear scan suffices.

(iv) Analogously to (iii) we use the algorithm in (i) to determine the sorted list L of all elements in the AVL tree. Then scan this list from front to back moving thos elements e satisfying $x \leq e \leq y$ to an output list.

EXERCISE 3.

(i) Prove that the total number of comparisons in a search in an (a, b)-tree with n nodes is bounded by $\lceil \log b \rceil (2 + \log_a((n-1)/2))$.

(ii) Assuming $b \leq 2a$ show that the number in (i) is in $O(\log b) + O(\log n)$.

SOLUTION.

(i) The root of an (a, b)-tree has at least two children, and all other non-leaf nodes have at least a children. Thus, in a tree of height h the number of nodes is at most

$$1 + 2\sum_{i=0}^{h-1} a^i = 1 + 2\frac{a^h - 1}{a - 1} \ .$$

This implies

$$\frac{(n-1)(a-1)}{2} + 1 \geq a^h$$

and hence

$$h \leq \log_a \left(\frac{(n-1)(a-1)}{2} + 1 \right) \leq 1 + \log_a \frac{n-1}{2} + \log_a(a-1) \leq 2 + \log_a \frac{n-1}{2} \ .$$

The height determines the length of a search path, and in each node along the path there are at most b elements stored in an ordered way. For binary search in a node we need at most $\lceil \log b \rceil$ comparisons.

Hence the total number of comparisons is bounded by $\lceil \log b \rceil (2 + \log_a((n-1)/2))$.

(ii) Using the result from (i) the number of comparisons is bounded by

$$\left(\log_a \frac{n-1}{2} + 2\right) \cdot \log_2 b \;=\; \frac{\log_2 b}{\log_2 a} \log_2 \frac{n-1}{2} + 2\log_2 b \;.$$

As we assume $b \leq 2a$, we get $\log_2 b \leq 1 + \log_2 a$, which gives

$$\frac{\log_2 b}{\log_2 a} \log_2 \frac{n-1}{2} + 2\log_2 b \;=\; \left(\frac{1}{\log_2 a} + 1\right) \log_2 \frac{n-1}{2} + 2\log_2 b \;,$$

which is in $O(\log n) + O(\log b)$ as claimed.

EXERCISE 4.

(i) Implement a TRIE data structure. To test your implementation take a text of your choice with a length of around one page and build a dictionary with the words in the text.

(ii) Implement a procedure, which in case that a search for a word in the dictionary fails return suggested alternatives:

(a) Return words from the dictionary that extend the given word (not found in the dictionary) by one symbol.

(b) Return words from the dictionary that are prefixes of the given word (not found in the dictionary) with one or two symbols less.

(c) Return words from the dictionary that differ from the given word (not found in the dictionary) by exactly one symbol.

SOLUTION. See the C++ header and program files in the archive `Ass7_Ex4solution.zip`.