Lab Discussion 7

EXERCISE 1. Explore the class definitions BST and AVLTREE:

(i) Create binary search trees and AVL trees with the same elements, compare the trees, the height of subtrees, the balances of nodes.

(ii) Run *find* and *delete* operations on your trees.

(iii) Implement methods to determine the minimum and the maximum element in BSTs and AVL trees.

(iv) Implement a method to return the median of the elements in an AVL tree.

(v) Given x and e, implement an operation that returns the elements e in an AVL tree with $x \leq e \leq z$.

EXERCISE 2.

(i) Write a program that checks, if a given binary tree with labelled vertices is a BST.

(ii) Write a program that checks, if a given binary tree with labelled vertices is an AVL tree.

EXERCISE 3.

(i) Modify the implementation of the class TRIE used in the lectures such that it suffices to store unique prefixes of minimum length of the keys in a TRIE object.

(ii) Use this idea to store the elements of a set S that is in one-one correspondence with the set K of keys in a TRIE object.

(iii) Modify the *insert* and *find* operations accordingly.

EXERCISE 4. A *Fibonacci tree* of height h is an AVL tree with a minimum number of nodes among all AVL trees of height h.

(i) Familiarise yourself with the notion of Fibonacci tree by creating Fibonacci trees of height $1, 2, 3, 4$.

(ii) Write a program that checks, if a given binary tree with labelled vertices is a Fibonacci tree.

(iii) Determine the number of Fibonacci trees of height h (up to isomorphism).

(iv) Give a criterion, whether it is possible to create a Fibonacci tree with n given elements.

(v) Implement a method that turns an AVL tree into a Fibonacci tree, if this is possible.

(vi) Implement a class FIBTREE of Fibonacci trees with a method *insert*, which according to (iv) must insert several elements. Exploit (v) for the definition of the *insert* method.