Algorithms & Datastructures
Laboratory Exercise Sheet 6

Wolfgang Pausch <wolfgang.pausch@uibk.ac.at>
Heiko Studt <heiko.studt@uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>
Tomas Vitvar <tomas.vitvar@uibk.ac.at>

April 27th, to be discussed on May 4th

Exercise 1) Tree Traversal The tree iterator from the lecture only supports traversal of trees, but did not allow to modify the underlying data-structure. For this exercise you should also implement the `remove()`-method in the iterator with the following conventions (adapted from the Java Iterator-Interface).

- `remove` may only be invoked after a call to `next`.
- If `remove` is called, then the last element that has been returned by `next` is removed from the tree. It is not allowed to call `remove` twice without an intermediate call to `next`.
- If `remove` is called wrongly, an `IllegalStateException` should be thrown.
- After a (legal) call to `remove`, the iterator should proceed normally, i.e., in total, all elements must be traversed and no element should be returned twice.

For example, if there is a tree with elements 4,7,8,10,11 then after calling `next` twice, the iterator may return first the 4 and then the 8 (or other values, depending on the structure of the tree and the traversal-order). A call of `remove` should now delete the 8 from the tree. Two more calls to `next` return the values 11 and 7. Another call to `remove` is possible and removes the 7. Finally, one more call to `next` is possible and it returns the element 10.

1. In the sources to this exercise sheet you already find a slightly simplified binary tree implementation from the lecture where the iterator only supports inorder-traversal. Modify this implementation such that the iterator supports the `remove`-operation. (You might also need to change other methods, please indicate which methods have been changed.)
2. Would your implementation also work correct for preorder-traversal or for postorder-traversal? If not, try to illustrate the problem (think about removal of nodes with two children).

Exercise 2) AVL Trees

1. Insert the values 3,7,9,1,2,8,6,4,5 into an empty AVL-tree (in the given order). Give the tree after each insertion.
2. In the picture on slide 163 the new `hdiff`-values for nodes `n` and `p` are missing. Determine these values by only looking at the other `hdiff`-values, but without computing the heights of the trees.
3. Implement the method `rotateLeft` for AVL-tree nodes. This method should update all references to children, parents, etc., but should ignore the `hdiff`-values.
4. Implement `balGrow` where you can omit the symmetric cases where `n = p.right`. Moreover, you can assume that both `rotateLeft` and `rotateRight` are available.

An incomplete implementation of AVL-trees is given in the sources of this exercise sheet.