Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Thomas Tschager
Antonis Thomas

23th March 2016

Datenstrukturen & Algorithmen Exercise Sheet 5 FS 16

Exercise 5.1 *AVL Trees.*

a) Draw the resulting tree if you insert the keys $5, 8, 16, 10, 12, 13, 15, 9, 11$ in this order into an initially empty AVL tree.

b) Remove the key 5 from the tree in a) and draw the resulting AVL tree.

c) Let T be an AVL-tree that stores the set S of keys. Is there an insertion order for S, such that starting from an empty tree no insertion operation leads to a rotation in the tree and the resulting tree is exactly T? If yes, prove the statement and give a counterexample otherwise.

Exercise 5.2 *Advanced Search Trees.*

In this exercise, we wish to extend the functionality of a natural search tree. We assume that the tree maintains integers. Modify the search tree such that we can not only find a number, but are also able to answer the following queries.

a) How many elements in the tree are *even* and smaller than a given number k?

b) How many *even* elements in the tree are (strictly) between two given numbers k_1 and k_2 with $k_1 < k_2$?

Discuss which additional information every node has to store and how the insertion and deletion operations have to be changed such that the above queries can be answered efficiently. Provide the running times of all operations.

Exercise 5.3 *Amortized Analysis.*

In this exercise, we consider arrays that grow dynamically on demand (e.g., `java.util.Vector` in the Java standard library). Specifically, we assume that we insert the values one by one. If more than n elements are stored, a new array of fixed length $k > n$ is created, the old contents are copied and the new element is stored. An array of length k can be created in k steps, and the copying of an element is done in constant time.

a) Describe how to choose k so that each insert operation has amortized constant time, and hence the insertion of n elements can be done in time $\Theta(n)$. Prove using amortized analysis that your choice results in constant amortized time per insert operation.

b) Now consider the situation where we allow to delete the last element from the array. We allow any mixture of such insert and delete operations. For memory reasons, it may be useful to also shrink the array sometimes. Describe how would you shrink the array, and show that both insertion and removal require amortized constant time.

Please turn over.

Exercise 5.4 *AVL trees (**Programming Exercise**).*

The goal of this exercise to write your own Java implementation of AVL trees. For this, we provide you with a Java file that you can use for your code. It contains the whole implementation except the following functions that you have to implement yourselves:

- `rotateLeft`. The analogous function for rotating right is given and it should be of help.
- `insert`. The function for inserting an element. It should insert the element to the right place (duplicates are ignored) and then call the `rebalance` function (provided).
- `delete`. The function for deleting an element. In case the node to be deleted has two children, always use the *in-order successor* for key replacement. Finally, call the `rebalance` function.

Input The first line of the input contains only the number t of testcases. Each of the t testcases is then given in the following way: the first line contains $n \leq 100$. After that, we have one line of n non-zero integer numbers. The i-th number x_i represents the insertion of element x_i if $x_i > 0$ and represents the deletion of element $|x_i|$ if $x_i < 0$. For example, $x_i = 10$ means insert element 10 and $x_i = -10$ means delete element 10 (if element 10 exists in the AVL tree).

Output For each testcase, you have to output the *balance vector* of the AVL-tree. That is a vector of length n over $\{-1, 0, 1\}$, where the ith element represents the balance of the ith node of the AVL tree with respect to the in-order traversal. For this, we give you the `printBalance` function.

Example

Input:

```
2
4
7 10 9 1
7
6 8 3 1 9 -6 2
```

Output:

```
0 -1 -1 0
0 0 0 -1 0
```

Remarks The balance of a node is defined as the height of the right subtree minus the height of the left subtree. This is given in the code.

Testsets There are two categories of testsets for a total of 100 points:

- **Inserts**: Only inserting elements. Worths 50 points.
- **InsertsAndDeletes**: Both inserting and deleting elements. Worths 50 points.

Hand-in: Wednesday, 6th April 2016 in your exercise group.