Eidgenössische
Technische Hochschule
Zürich

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

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

8th December 2016

Data Structures & Algorithm Solutions to Sheet P11 AS 16

Solution P11.1 *Binary trees.*

The solution was mostly outlined in the task description and it was just important to work out the details. See the example solution for a discussion of those.

Generally, every operation on a given value had to compare the value to current `key` and either work on the current node, or recursively call itself on the left subtree or right subtree. The only exception was `delete` of a node X with two children, where finding the nearest larger key can be done by going from X once to right child and then repeatedly to left child until you hit a node Y without a left child – that node always has the nearest larger key than X.

Keeping the values of subtree `size` updated could be done either explicitly with a function like `node.fixSizesToRoot()` that recomputes the sizes on the path from *node* all the way to the root, or even directly in the `insert` or `delete` recursive functions, as the differences are always just +1 or -1 (as hinted in the task text).

Note that keeping the subtree sizes is a simple example of *subtree aggregation*. You could also easily maintain, for example, the sum or product of the keys of every subtree, the maximum depth (as in AVL trees), the average of the keys, the smallest and largest key, the number of even keys etc.

Solution programs On the lecture website, you can find a the solution sources with further comments on the implementation.

Data Similarly to the `test*` cases, test cases `judge2` and `judge3` first inserted 30 000 random unique elements in such an order to have a tree of depth cca 30. Then they inserted another 10 000 random elements while asking for their presence before and after the insert. Then 10 000 random elements were deleted, again asking for them before and after. Finally, `judge3` then asked for 10 000 elements by their rank, occasionally deleting some of them (to prevent solutions that would just make an array of the elements for rank lookups). `judge1` was smaller and contained no deletes. The trees were printed cca 5 times in every testcase (printouts are large and slow, so only few were used).