Assignment No. 7: Dynamic Order Statistics

Allocated time: 2 hours

Implementation

You are required to implement **correctly** and **efficiently** the management operations of an **order statistics tree** (*chapter 14.1 from the book*[1]).

You have to use a balanced, augmented Binary Search Tree. Each node in the tree holds, besides the necessary information, also the *size* field (i.e. the size of the sub-tree rooted at the node).

The management operations of an **order statistics tree** are:

- BUILD_TREE(n)
    - *builds* a **balanced** BST containing the keys 1,2,...n (*hint:* use a divide and conquer approach)
    - make sure you initialize the size field in each tree node
- OS-SELECT(tree, i)
    - selects the element with the *i*-th smallest key
    - the pseudo-code is available in *chapter 14.1 from the book*[1]
- OS-DELETE(tree, i)
    - you may use the deletion from a BST, without increasing the height of the tree (why don't you need to rebalance the tree?)
    - keep the size information consistent after subsequent deletes
    - there are several alternatives to update the size field without increasing the complexity of the algorithm (it is up to you to figure this out).

Does OS-SELECT resemble anything you studied this semester?

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*

Requirements

1. **BUILD_TREE: correct and efficient implementation (5p)** ✓

You will have to prove your algorithm(s) work on a small-sized input (11)

- pretty-print the initially built tree ✓

2. **OS_SELECT: correct and efficient implementation (1p)** ✓

You will have to prove your algorithm(s) work on a small-sized input (11)

- pretty-print the initially built tree
- execute OS-SELECT for a few elements (at least 3) by a randomly selected index

3. **OS_DELETE: correct and efficient implementation (1p)** ✓

You will have to prove your algorithm(s) work on a small-sized input (11)

- pretty-print the initially built tree
- execute OS-SELECT followed by OS-DELETE for a few elements (at least 3) by a randomly selected index and pretty-print the tree after each execution.

4. **Management operations - BUILD, SELECT, DELETE - evaluation (2p)** ✓

! Before you start to work on the algorithms evaluation code, make sure you have a **correct algorithm**!

Once you are sure your program works correctly:

- vary *n* from 100 to 10000 with a step of 100;
- for each n (don't forget to repeat 5 times),
  - BUILD a tree with elements from *1 to n*
  - perform *n* sequences of OS-SELECT and OS-DELETE operations using a randomly selected index based on the remaining number of elements in the BST,
  - Evaluate the number of operations needed for each management operation (BUILD, SELECT, DELETE)

Evaluate the computational effort as the sum of the comparisons and assignments performed by each individual management operation.

5. **Bonus: Implementation using AVL / Red black tree (1p)** ✗

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*