Assignment No. 6: The Josephus Permutation

Allocated time: 4 hours

Implementation

You are required to implement **correctly** and **efficiently** an *O(nlgn)-time* algorithm which outputs the *(n,m)-Josephus permutation*, for when m is not constant (problem 14.2 (b) 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). First, you have to *build* a balanced BST containing the keys 1,2,...n (*hint:* use a divide and conquer approach). Then, at each step you have to *select* and *delete* the k-th element from the tree.

The pseudo-code for the algorithm:

```
JOSEPHUS(n,m)
        T = BUILD_TREE(n)
        j ←1
        for k ←n downto 1 do
                j ←(( j + m − 2) mod k) + 1
                x ←OS-SELECT(root[T], j )
                print key[x]
                OS-DELETE(T, x)
```

The pseudo-code for the OS-SELECT procedure can be found in Chapter 14.1 from the book[1]. For OS-DELETE, you may use the deletion from a BST, without increasing the height of the tree (why don't you need to rebalance the tree?). You have to be careful that, at each step, the *size* information in each node be correct. 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). For BUILD_TREE, you have to write a procedure which builds a balanced BST from the keys 1, 2, …, n. Make sure you initialize the *size* field in each tree node in the BUILD_TREE procedure.

Evaluation

! Before you start to work on the algorithms evaluation code, make sure you have a correct algorithm! You will have to prove your algorithm(s) work on a small-sized input: i.e. for

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

Josephus(7,3), pretty print the augmented BST after each step of your algorithm (i.e. initial tree build, then after each OS_DELETE).

Once you are sure your program works correctly, vary n from 100 to 10000 with step 100; for each n, choose the value of m as n/2.

Evaluate the computational effort as the sum of the comparisons and assignments performed by your algorithm on each size. Is your algorithm running in O(nlgn) ?