Technische Universität München
Fakultät für Informatik
Lehrstuhl für Algorithmen und Komplexität
Prof. Dr. Harald Räcke
Richard Stotz

Winter term 2019/20
Assignment 07
November 25, 2019

Efficient Algorithms and Data Structures I

*Deadline: December 2, 10:15 am in the **Efficient Algorithms** mailbox.*

Homework 1 (3 Points)

In an ancient book on elephant mathematics, the elephant Amadeus Cage, discovers Algorithm 1. The description has almost faded, but seems to be `elephantnutco`, its subroutine being called `wicked`. The description also indicates that the algorithm takes a boolean array A and an integer k as input.

Algorithm 1: `elephantnutco`(A, k)

```
Algorithm elephantnutco(A, k)
    for j = 0; j < k; j ← j + 1 do
        wicked(A)

Procedure wicked(A)
    i ← 0
    while i < A.length and A[i] == TRUE do
        A[i] ← FALSE
        i ← i + 1
    if i < A.length then
        A[i] ← TRUE
```

(a) Describe briefly what the procedure `wicked` does.

(b) Using a suitable potential function, show that the amortized running time of procedure `wicked` during an execution of `elephantnutco` is $\mathcal{O}(1)$, **if the array A is initially false at every position**.

Homework 2 (5 Points)

Consider a binary search tree that supports `find-by-rank` as described in the Slides 190ff.

(a) Write an efficient method Rank(x) that returns the rank of a node x, given pointer to node x.

(b) Suppose we store the subtreerank of the node with the node (just as we store the size field). The subtreerank of a node x is the rank of x within the subtree of root x. Show how to maintain the subtreerank field during a left-rotation without using the size field.

Homework 3 (6 Points)

A minqueue is a dynamic data structure that supports the following operations:

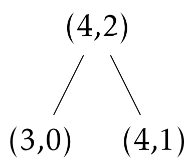
- ENQ(x) : Inserts the number x into the minqueue.
- DEQ(): Removes the element that has been in the minqueue for the longest time.
- FMin(): Returns the smallest value in the minqueue but does NOT remove it.

You may assume that all numbers that are enqueued are distinct.
You are given two doubly linked lists (called A and B) to store elements, as well as some constant amount of additional storage. Describe how to implement a minqueue such that all operations are supported in amortized constant time.
Describe precisely how the operations are implemented. Then analyze their running times using the accounting method or a suitable potential function. You do not need to prove that the operations work correctly.

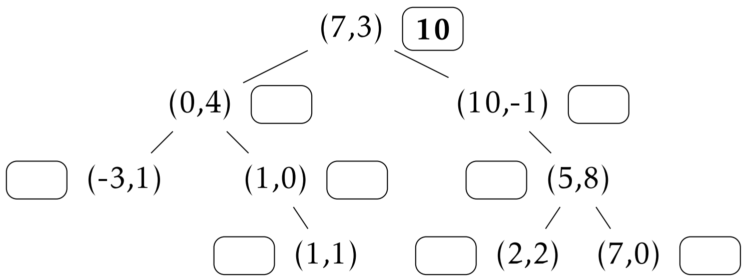
Hint: Try to implement DEQ() and FMin() in worst-case constant time. Use list A to store all elements in your data structure. Use list B to store all elements that can still become minimum.

Homework 4 (6 Points)

A *Bavarian Search Tree* is a Binary Search Tree in which each node v contains a *base* as well as an additional value called *addend*. The addend of a node v is implicitly added to each base in the subtree rooted at v. Let (*base*, *addend*) denote the contents of any node v. For example, the following tree contains the elements $\{5, 6, 7\}$:



(a) In the following Bavarian Search Tree, write the value of each node next to it. The value of the root is 10.



(b) Let h be the height of some Bavarian Search Tree. Describe how to perform the following operations in $O(h)$ time:

- FIND(x): return YES if value x is stored in tree T.
- INSERT(x): inserts element x in tree T.

- SUPERADD(x,k): add k to all elements $\geq x$ currently in the tree.

(c) Show how to perform a right-rotation in a Bavarian Search Tree in constant time.

For parts (b) and (c), show that your implementation reaches the required time bounds. You do not need to prove correctness.

Tutorial Exercise 1

Describe a dynamic data structure for storing intervals. Your data structure should support the following operations:

- INSERT($[q_\ell, q_r]$): Stores the interval with left endpoint q_ℓ and right endpoint q_r into the data structure. Returns a handle I for referencing the interval.
- DELETE(I): Deletes the interval referenced by I from the data structure.
- OVERLAP(): Returns true if at least two intervals in the data structure overlap and false otherwise.

You may assume that all endpoints of intervals are distinct. Your data structure should perform INSERT and DELETE in $O(\log n)$ time and OVERLAP in constant time, where n is the number of intervals currently stored in your data structure.
Describe precisely how your data structure is implemented and prove correctness and running time of all operations.

Hint: Use an augmented red-black tree that stores the intervals sorted by their left endpoint.
Hint 2: Observe that OVERLAP must take constant time. This limits the possibilities for augmenting the tree.

Tutorial Exercise 2

Suggest how to use a skip list so that given a pointer to a node with key x, we can return a pointer to a node with key $y < x$ in $O(\log k)$ expected time where k is the distance between the nodes with values y and x in L_0. Prove that your method works!

```
Computer science, I think, differs
from other fields most in that it
constantly jumps levels - from look-
ing at something in the small to
looking at it in the large.
                       - D.E. Knuth
```