Department of Mathematics and Computer Science
University of Southern Denmark, Odense

September 15, 2016
KSL

# Part 1 of the exam project in DM803 – Advanced Data Structures

Kim Skak Larsen
Fall 2016

## Introduction

In this note, we describe one part of the exam project that must be solved in connection with the course DM803 – Advanced Data Structures, Fall 2016. It is important to read through the entire project description before starting the work on the project; also the sections on requirements and how to turn in your solution.

## Deadline

The deadline for this part of the project is

Wednesday, October 26, 2016 at 12:00 (noon).

## Implementation of Scapegoat Trees and Skip Lists

The aim is to implement scapegoat trees and skip lists, investigate properties of these separately and compare them. Note that in the general rules below, we state some implementation requirements in addition to the ones given in this section.

Both data structures must be implemented as outlined in the articles describing them, and in such a way that the same time and space complexity bounds hold, and both must support at least the operations `search`, `insert`, and `delete`.

After implementing them, you should investigate their properties. Your philosophy should be that you want to investigate and understand, and then you want to

communicate the results as clearly as possible to someone else afterwards. Thus, in particular, you should include graphs showing the connections for all your findings; not just tables with measurements. Below are some requirements and ideas, but you can supplement with additional questions.

Instead of measuring time, you must count comparisons, i.e., in the following when we refer to the time complexity of operations, we mean the number of times two keys are compared.[1] As a starting point, investigate the issues below when searches are uniformly random. An easy way to handle this would be the following: Decide on an $n$, insert the keys $1$ through $n$ in uniformly random order, and then search a number of times, choosing the key to search for every time uniformly at random from $\{1, \ldots, n\}$. You can of course also investigate other distributions or worst-case ordering if you wish, e.g., what happens if you insert the numbers $1$ through $n$ in sorted order?

Investigate the following:

- Average search complexity.

  For both structures, illustrate this using a graph where you display the search complexity as a function of $n$. Can you demonstrate that the search time is logarithmic? Can you find out what the constant is in front of the logarithm, assuming that we use base 2?

  For skip lists, try to investigate the connection between the choice of $p$ and the constant in front of the logarithm. You can do this for a fixed, large enough $n$. Does it behave as predicted theoretically?

- Variation in search complexity.

  For a large number of searches, record for each $i$ how many of these searches had time complexity $i$, and illustrate by graphing the percentage of searches of complexity $i$ as a function of $i$. If this does not give a good illustration, consider accumulating, i.e., at $i$, give the percentage of searches of complexity $i$ or smaller.

- For scapegoat trees, investigate frequency and size (number of nodes involved) of restructurings, and the relationship between the two.

[1] These data structures are extremely fast, which means that it is basically impossible to get meaningful results by simply timing operations. If trying to time extremely fast operations, actions of the underlying operating system disturb the results so much that results become meaningless. To work around this, one can decide to work with a very large $n$, so that running times get slowed down. However, $n$ has to be so large that data storage effects, such as cache misses at different levels, start ruining the measurements instead.

Compare and discuss your findings and supplement with you own ideas, if any.

For the report, other than the investigations described above, you need only discuss possible important choices in your implementation. There is no reason to go through code that is simply implemented as outlined in the papers.

Remember to read the general rules, where there are also explicit requirements to programs and reports.

### Specific Execution Requirements and Input/Output Formats

You can let your programs take options such that you can define your own behaviors as well. However, when executed without any options, the programs should follow the directions below.

Your programs must read from `stdin` and write to `stdout`. Keys are integers.

Input is a number of lines, each line consists of one capital letter, I, D, or S, as the first character on the line, followed by one space, and then a key. The letters symbolize insertion, deletion, and search, respectively. Output is the same number of lines as the input and each line is a response to the corresponding input line. The response is either S or F, for success or failure, as the first character on the line. You may add additional explanation on the same line following the letter S or F. For insertion, S means that you successfully inserted the key, for deletion, S means that you successfully deleted the key, and for search, S means that you found the key. In all cases, F is the negation of the statements above.

After the processing of each input line, you must flush the output.

Fig. 1 contains an example of an input file and the corresponding output file (you cannot see the white space characters), with a few optional example explanations.

## General Requirements and Rules

Here we list general requirements, procedures for turning in, and exam rules.

### Exam Rules

This is an exam project, and cooperation is not permitted. It will be considered cheating and will be treated as such. You have a duty to keep your notes private and protect your files against reading and copying by others. Both parties involved in a possible plagiarism can be held responsible.

| Input | Output |
|---|---|
| I 42 | S |
| I 43 | S |
| I 44 | S |
| S 43 | S |
| S 41 | F |
| D 43 | S |
| S 43 | F |
| I 41 | S |
| I 42 | F - key already present |
| S 41 | S |
| S 42 | S |
| D 42 | S |
| S 42 | F |
| D 42 | F - key not present |

Figure 1: Input example with corresponding output.

## On Making the Deadline

I always set deadlines so you have very long time for a project compared to how long it takes, enabling you to schedule your work where it is most convenient for you. I do not give extensions without formal and objective reasons such as long-term, documented illness, for instance. However, if you notice a problem with the deadline shortly after it has been announced, please alert me and I will consider changing it. If you are interested in why I try to follow this policy and why I think it is a matter of fairness, you can read about that here:

`http://www.imada.sdu.dk/~kslarsen/Blog#18`

## The Report

The front page of your report must include your full name, the first 6 digits of your CPR number, and student e-mail address (the prefix to `@student.sdu.dk`).

There are not supposed to be any of the following, but if there are possible omissions, known errors, etc. they should be described in the report. It is often a good idea to do this in a separate section instead of mixing it in with the rest of the report.

The report should in the best possible manner account for the entire solution, i.e., if

your solution includes programs, the report must contain a description of the most important and relevant decisions that have been made in the process of developing the solution and reasons must be given where this is appropriate. You must also explain how possible programs have been tested. Test examples or references to test examples and test runs can and should be included in the report to the extent that this is helpful to understand your solution and to be convinced that it is working correctly.

## Programs

When you have to implement algorithms, you must implement them in such a way that you obtain the asymptotic complexities claimed in the course material. If you happen to find a library, a package, or similar that implements the algorithms that you are supposed to implement, then you are of course not allowed to use them. You should make the implementation yourself from scratch using the basic features of the programming language you are working with. The safest approach is not to use libraries and avoid non-trivial built-in features.

Files and directories should be named and organized logically. Programs must be well-structured with appropriately chosen names and indentation and tested sufficiently.

Programs that are turned in must compile and run on IMADA's machines.

The preapproved programming languages you can choose from are the following:

- C or C++
- Java
- Python

If you have other preferences, you could likely obtain permission to use an alternative. Contact the lecturer.

You are very welcome to develop your programs at home, but it is your responsibility. This includes technical problems at home, lack of access to relevant software, moving data to IMADA via e-mail, USB keys, etc. and converting to the correct format, e.g., between Windows and Linux.

You must turn in a file `manual.txt`. Here you must explain how to compile and run your code on IMADA's computers. Be careful not to hardwire references to your home directory etc. into the code such that it will not be possible for us

to compile or run your program directly. Your goal should be to make it as easy as possible for us to run your program on additional tests to the ones you have provided. You have to make everything really clear, e.g., which directory one should be in, the order in which commands should be carried out, etc. You are advised to make things as simple as possible. The safest is usually to have all source files and executables in one directory. Of course, your report and test files can be in subdirectories.

## Turning In

You must turn in everything, i.e., the report in pdf-format, named `report.pdf`, andall relevant programs, test files, and `manual.txt`.

You upload your files using "SDU Assignment" in Blackboard (which will give you a receipt). You should avoid Danish (and other non-ascii) characters (such as æ, ø, and å) in your directory and file names (Blackboard does not handle this well). To be safe, also avoid spaces and all special characters not normally occurring in file names.

You may upload your files individually or collect your files into one (archive) file (recommended) before uploading. If you choose to do the latter, you may use `zip`, `bzip2`, or `tar` (with or without `gzip`).