7. Problem set on

Algorithms, Data Structures and Data Abstraction WS 18/19

Wolfgang Mulzer

Due on 07. December 2018, 10:00, in the respective TA's mailbox

Problem 1 Hashing: Implementation *10 points*

(a) Read the documentation of `hashCode()` in `java.lang.Object`. Which properties need to be satisfied? Why should you overwrite the method in your own classes?

(b) Implement hashing with chaining, as described in class. Use `hashCode()`. For the linked lists, you may use the Java-library. The hash-table should grow dynamically as soon as a certain load factor is reached.

Perform experiments with your implementation. How do the access times depend on the load factor? Compare two different hash functions (these may be hard-coded for a fixed table size).

Problem 2 Analysis of Skip Lists *10 points*

Let L be a skip list with n entries. Show the following:

(a) The expected number of nodes in L is $O(n)$.

(b) With probability at most $n/2^{j-1}$, the list L has at least j horizontal lists.

Hint: For events $A_1, \ldots, A_\ell$, we have $\Pr[A_1 \cup \cdots \cup A_\ell] \leq \sum_{i=1}^{\ell} \Pr[A_i]$ (this is called the *union bound*).

Problem 3 Implementation of an Ordered Dictionary *10 points*

(a) Write a Java-interface for the abstract data type `OrderedDictionary` from class. Provide comments to specify the operations. Implement the interface using a skip list.

Hint: You can find pseudocde for the skip list operations on the website.

(b) Read the specification of `java.util.Iterator`. Write methods `iterator()` and `reverseIterator()` that create an `Iterator`-Object. `iterator()` should process the entries in sorted order, `reverseIterator()` should process the entries in reverse order.

Make sure your implementation follows the specification laid out in the Java-API-documentation. You do not need to implement the `remove()`-method.

(c) (*voluntary, 5 additional points*)

What happens if the skip list is changed while an `Iterator`-object is active? Describe a possible problem.

One solution uses *fail-fast* iterators: as soon as the skip list is changed through `put` or `remove`, *all* active `Iterator`-objects become invalid. Later method calls result in a `ConcurrentModificationException`. Iterators that were created after the modification should still work. How can this behavior be implemented? Read about the *observer pattern*, and describe how to use it for the problem at hand.

You do not need to write any code.

Problem 4 Cryptographic Hash Functions *voluntary, 10 additional points*

(a) What is a crytopgraphic hash function? Name three heuristic candidates.

(b) Which cryptographic hash functions are available in Java? How can they be used?

(c) Adapt your hash table from Problem 1(b), so that it uses cryptographic hash functions. Perform appropriate experiments.

Give an advantage and a disadvantage of this implementation.