Data Structures and Algorithms I — Tue 7.3.2017

Exercise 7

No mandatory X-task this week, but there will be on last week. Draw a picture of all exercises.

In the following exercises we implement abstract data type queue according to interface *java.util.Queue*. First using a ready linked list, then using a plain array, and finally using dynamic linked nodes as the storage of the elements. Interface *Queue* contains some redundancy, but you can implement one operation and call it from the other one. Much of *Collection* functionality will be inherited from *AbstractSequentialList*. Implementation of *Iterable* can be non-functional.

Take a skeleton and test program from course www-page. Rename the same skeleton for each of the three different implementations. You can use the same test program to test all the implementations.

37. Implement interface *Queue* using *java.util.LinkedList* as a storage structure and using its operations. Even if *LinkedList* already implements *Queue*, you must implement them as we do not extend *LinkedList*.

38-39. Implement interface *Queue* using a plain array as the storage structure and using the array as a ring buffer. Now you must allocate the array and remember where the queue elements are stored in the array. Operations must be (average) unit time.

40. Add to previous exercise 38-39 automatic doubling the storage space when queue runs out of space.

41-42. Implement interface *Queue* using dynamic allocation of linked nodes. The queue is an object holding references to first and last nodes of the queue. Each node contains an element and a reference to the next node. Use an internal class to represent the linked nodes of queue storage. Each add will allocate a new node and link it as the last node of the queue. Do not use ready linked list implementations.