Assignment 7

David Blumenthal, Florian Hofer,
Werner Nutt, Daniele Porello

Priority Queues and Lists

1. Priority Queues

In this coursework we realize so-called priority queues using heaps. Priority queues are a an abstract data-type that allows one to insert new elements into a set and to extract the maximum of a set. Priority queues are a crucial building block of many complex algorithms. They form an *abstract* data type because we do not specify how to implement them.
You have learnt about heaps as a data structure that allows one to retrieve the maximum in a set of numbers in constant time. Moreover, a heap can be maintained in logarithmic time if we insert a new element at the root. Your task will be to realize priority queues using heaps in arrays.
Implement priority queues as instances of the class

```
public class PriorityQueue{
    int maxSize;
    int currentSize;
    int[] queue;

public PriorityQueue(int maxSize){
    this.maxSize = maxSize;
    queue = new int[maxSize];
    currentSize = 0;
  }
}
```

An instance of this class can hold a heap of up to `maxSize` integers. Initially, the heap has size 0, which means that none of the elements of the array `queue` is considered to be a heap element.

You are now asked to implement the following methods for this class:

1. `boolean empty()`, which returns `true` iff the heap is empty;
2. `boolean full()`, which returns `true` iff the number of integers in the heap is equal to `maxSize`;
3. `int extractMax()`, which returns the maximum element of the heap, deletes it from the heap, and reorganizes the array so that it is a heap of the remaining elements; after this operation the `currentSize` of the priority queue is one less than it was before; clearly, the operation can only be applied if the priority queue is not empty, otherwise, an exception has to be raised;
4. `void insert(int n)`, which inserts a number n and reorganizes the array so that it is a heap of the new, larger set of elements; after this operation the `currentSize` of the priority queue is one greater than it was before; clearly, the operation can only be applied if the priority queue is not full, otherwise, an exception has to be raised.

Your task is to develop and implement efficient algorithms for these methods and to test them. Explain all your solutions and answers to questions. You find a template for the class `PriorityQueue` in the zip file `DSA_A7.zip`.

1. Develop JUnit tests for the methods above, covering both special and general cases. You may want to write tests that combine several calls to the methods you want to write. For instance, you may want to insert several numbers and then execute `extractMax`.
2. Write pseudocode for the methods `extractMax` and `insert` and for auxiliary methods that you may need.
3. What is the asymptotic worst-case complexity of your algorithms?
4. Implement the four methods listed above.

(12 Points)

2. Operations on Linked Lists

Implement a data type `List` that realizes linked lists consisting of nodes with integer values, as discussed in the lecture. Remember that:

- an object of type `Node` has two fields, an integer `val` and (a pointer to) a Node `next`;
- an object of type `List` has one field, (a pointer to) a Node `head`;

The type List will have the methods specified below. For some methods we have developed implementations in the lecture. In these cases, writing the code is only an exercise to help you get more acquainted with algorithms for recursive data structures. For some other methods, we have developed only an iterative or a recursive implementation in the lecture. In those cases, you are asked to developed a recursive or an iterative version, respectively.
In the cases where it is specified whether the method should be recursive or iterative, please, follow the specification. If there is no specification, use the style you find most convenient.

1. `boolean isEmpty();`
2. `int length();`
   returns the number of nodes in the list, which is 0 for the empty list (choose the iterative or the recursive version);
3. `void print()`
   print the content of all nodes (choose the iterative or the recursive version);
4. `void addAsHead(int i)`
   creates a new node with the integer and adds it to the beginning of the list;
5. `int popHead()`
   returns the value of the head of the list and removes the node, if the list is nonempty, otherwise throws an Exception `NULL`;
6. `void addAsTailRec(int i)`
   creates a new node with the integer and adds it to the end of the list (implement as a recursive algorithm);
7. `void addSorted(int i)`
   creates a new node with the integer and adds it behind all nodes with a `val` less or equal the `val` of the node, possibly at the end of the list (choose the iterative or the recursive version);

8. `Node find(int i)`
   returns the first node with `val i` (implement both an iterative and a recursive version);

9. `void reverse()`
   reverses the list (choose the iterative or the recursive version);

10. `void removeFirstRec(int i)`
    removes the first node with `val i` (implement as a recursive algorithm);

11. `void removeAll(int i)`
    removes all nodes with `val i` (choose the iterative or the recursive version);

12. `void addAll(List l)`
    appends the list `l` to the last element of the current list, if the current list is nonempty, or lets the `head` of the current list point to the first element of `l` if the current list is empty (choose the iterative or the recursive version);

13. `List toList(int[] A)`
    returns a list that contains the elements of array `A`, in the order in which they occur in the array; (choose the iterative or the recursive version);

14. `int[] toArray()`
    returns an array of integers that contains the elements of the list in the order in which they occur in the list (choose the iterative or the recursive version).

(18 Points)

Deliverables.

1. A short report for Question 1 that explains how you implemented the methods `extractMax` and `insert`, and why and how you chose your JUnit tests;

2. Your implementation of the classes in Exercises 1 and 2.

Combine all deliverables into one zip file, which you submit via the OLE website of the course. Please, follow the "Instructions for Submitting Course Work" on the Web page with the assignments, when preparing your coursework.
Submission: Until Mon, 9 May 2017, 23:55 hrs, to the OLE submission page of

Lab A / Lab B / Lab C