DIT960 – Datastrukturer

exam 2017-06-03

Time: 8:30 - 12:30. Place: SB.

Course responsible: Fredrik Lindblad, tel. 031-772 2038. Will visit at around 9:30 and 11:00.

Allowed aids: One hand-written sheet of A4 paper. You may use both sides. You may also bring a dictionary.

The exam consists of six questions. For each question you can get a U, a G or a VG. To get a G on the exam, you need to answer at least three questions to G or VG standard. To get a VG on the exam, you need to answer at least five questions to VG standard.

A fully correct answer for a question will get a VG. An answer with small mistakes will get a G. An answer with large mistakes will get a U.

Begin each question on a new page. Write your anonymous code (not your name) on every page.

When a question asks for pseudocode, you don't have to write precise code, such as Java. But your answer should be well structured. Indenting and/or using brackets is a good idea. Apart from being readable your pseudocode should give enough detail that a competent programmer could easily implement the solution, and that it's possible to analyse the time complexity.

When the exams have been graded they are available for review in the student office on floor 4 in the EDIT building. If you want to discuss the grading, please contact the course responsible and book a meeting withing three weeks after the grading is finished. In that case, you should leave the exam in the student office until after the meeting.

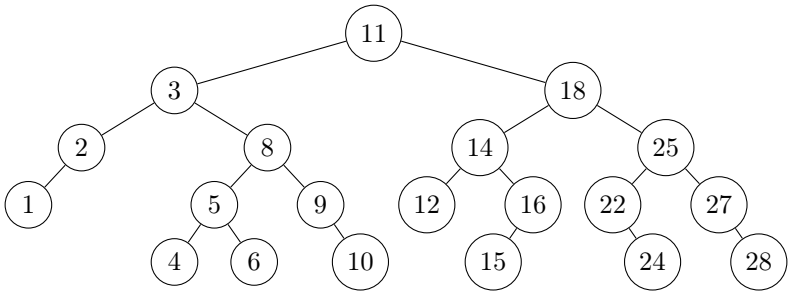
---

1. The following code takes as input two arrays with elements of the same type. The arrays are called `a` and `b`. The code returns a dynamic array which contains one copy of all elements which are in both `a` and `b`.

```
s = new empty set implemented as an AA tree
c = new empty dynamic array

for every element x in a
  s.insert(x)

for every element x in b
  if s.member(y) then
    c.append(y)
    s.remove(y)

return c
```

What is the big-O time complexity of this code? Express it in terms of n, the maximum of the lengths of the arrays. You should express the complexity in the simplest form possible. Apart from the final result you should also describe how you reached it, i.e. show your complexity analysis of the code.

2. Let T be the following binary search tree (BST):



The order is the normal one for integers. For a **G** you need to solve the first sub question below. For a **VG** you need to solve both sub questions.

(a) Assume that T is an unbalanced BST. How does the tree look after deleting the number 18? You only need to show the resulting tree, not the intermediate steps.

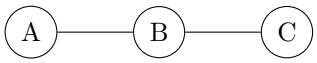
(b) Assume that T is an AVL tree. How does the tree look after inserting the number 7? You only need to show the resulting tree. Notice that you should start with T, not the resulting tree in the previous question.

3. The following Java type is used to represent an undirected graph with string labels on the nodes/vertices and no labels on the edges:

```
HashMap<String, ArrayList<String>>
```

It is a adjecency list representation. Remember that for undirected graphs there are two references for each edge between a node i and a node j; one from from i to j and one from j to i. This is part of the invariant of the representation.

As an example, the graph



is e.g. represented by the object $\{\mathrm{A} \to [\mathrm{B}], \mathrm{B} \to [\mathrm{A}, \mathrm{C}], \mathrm{C} \to [\mathrm{B}]\}$.

Implement the operation

```
boolean isTree(HashMap<String, ArrayList<String>> g)
```

which determines whether the graph `g` is a tree, i.e. is connected and acyclic. You can assume that `g` satisfies the invariant.

You may either use pseudocode or Java code. If you use pseudocode then make sure to follow the general guidelines at the beginning of the exam.

You may use the standard data structures covered in the course without explaining how they work. You may also use sorting algorithms without explanation, but the implementation of graph algorithms must be showed explicitly.

For a **VG** you should also analyse the time complexity of your code. You may state the complexity of standard operations covered in the course (except graph algorithms) without motivation.

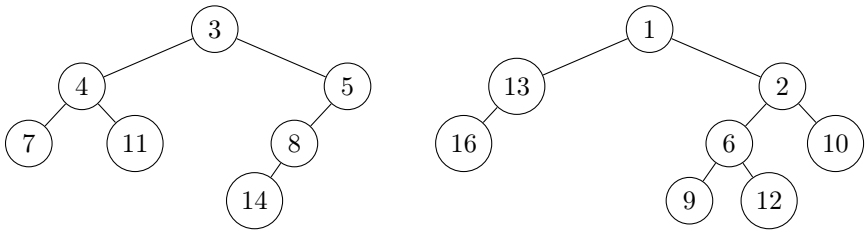
4. For a **G** you need to solve either one of the sub questions below. For a **VG** you need to solve both sub questions. In both sub questions the order of the elements is the normal one for integers.

   (a) The following array represents a binary heap in the standard way:

5	7	9	11	8	12	15	17	15	20	22	16	17	19	17	18	20	17	18

   What does the array look like after performing a delete-min on it?

   (b) The following two trees represent skew heaps:



   How does the resulting tree look after (skew heap) merging the two trees?

5. Implement an operation that takes a non-empty array of elements with defined equality and returns a value which occurs at least as many times as any other value. As an example, for the array $\{5, 2, 3, 8, 2, 1, 3, 12, 3, 2\}$ the answer would be 2 or 3. The implementation should be generic. It should not be specialised to arrays of integers.

   For a **G** the time complexity of the operation should be $O(n \log n)$, where n is the size of the array.

   For a **VG** the time complexity should be $O(n)$ provided that the appropriate conditions for the element type are satisfied. For a **VG** you should also motivate why the complexity requirement is satisfied.

   You may either use pseudocode or Java code. If you use pseudo code then make sure to follow the general guidelines at the beginning of the exam.

   You may use the standard data structures and algorithms covered in the course without explaining how they work.

6. Implement a data structure for a collection of elements which has the following constructor and operations:

   `empty` which creates a collection containing no elements

   `add(x)` which inserts element `x` into the collection

   `deleteOldestMin()` which returns the minimum element and deletes it from the collection. If there are several minimum elements then the operation should return and delete the element, among the minimum ones, that was first added to the collection.

   The implementation should be generic, but you can of course assume that the elements can be pairwise compared for less-than, equal, greater-then.

   For a **G** the time complexity of `add` and `deleteOldestMin` should be $O(n)$ (or better), where n is the number of elements in the collection.

   For a **VG** the time complexity of `add` and `deleteOldestMin` should be $O(\log n)$.

   You may either use pseudocode or Java code. If you use pseudocode then make sure to follow the general guidelines at the beginning of the exam.

   You may use the standard data structures and algorithms covered in the course without explaining how they work.