Exam – Datastrukturer

DIT960 / DIT961, VT-18
Göteborgs Universitet, CSE

Day: 2018-08-24, *Time:* 8:30-12.30, *Place:* J

Course responsible

Alex Gerdes, tel. 031-772 6154. 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.

Grading

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 *four* questions to G or VG standard. To get a VG on the exam, all answers need to be correct and you need to answer at least five questions to VG standard.

A fully correct answer for a question, including the parts labelled "For a VG", will get a VG. A correct answer, without the "For a VG" parts, will get a G. An answer with minor mistakes might be accepted, but this is at the discretion of the marker. An answer with large mistakes will get a U.

Inspection

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. In that case, you should leave the exam in the student office until after the meeting.

Note

- 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.
- Excessively complicated answers might be rejected.
- *Write legibly!* Solutions that are difficult to read are not evaluated!

Exercise 1

Here is a Haskell function to test if two lists `xs` and `ys` are disjoint (contain no elements in common) and uses an AVL-tree:

```haskell
disjoint :: [Int] -> [Int] -> Bool
disjoint xs ys = null bs
 where
  tree = foldr insert empty xs
  bs   = filter (flip member tree) ys

empty  :: AVL                 -- create an empty AVL tree
insert :: Int -> AVL -> AVL   -- add an integer to the tree
member :: Int -> AVL -> Bool  -- does the tree contain the specified integer
```

What is the worst-case time complexity of the `disjoint` function? You may assume that `xs` and `ys` have the same length, n. The complexity should be expressed in terms of n, the size of the input list(s). You should express the complexity in the simplest form possible. Apart from the final result you should also describe how you reached it, that is, show your complexity analysis.

For a VG only:

a) Don't assume that `xs` and `ys` have the same length. Give the complexity in terms of m and n, where m is the length of `xs` and n is the length of `ys`.

b) Suppose that we run the function twice, with the following inputs. Which run would you expect to go faster, if either?

1. `xs` is an array of 1000000 elements, `ys` is an array of 10 elements
2. `xs` is an array of 10 elements, `ys` is an array of 1000000 elements

Exercise 2

Suppose we are given the following type of binary search trees in Haskell:

```
data Tree a = Nil | Node a (Tree a) (Tree a)
```

a) Implement a Haskell function

```
greatest :: Ord a => Tree a -> Maybe a
```

that returns the greatest element in a non-empty binary search tree.

The complexity of your function should be $O(\text{height of tree})$, i.e., $O(\log n)$ for balanced trees, $O(n)$ for unbalanced trees.

b) **For a VG only:**

Write a Haskell function to delete an element. It should take two parameters, which are the element to delete and the tree, and has the following type:

```
delete :: Ord a => a -> Tree a -> Tree a
```

The complexity of your function should be $O(\text{height of tree})$, i.e., $O(\log n)$ for balanced trees, $O(n)$ for unbalanced trees.

Hint: it will help to use `greatest` when implementing `delete`.

Exercise 3

Perform a quicksort-partitioning of the following array:

53	88	32	44	23	90	67	71	42
0	1	2	3	4	5	6	7	8

Show the resulting array after partitioning it with the following pivot elements:

a) first element

b) middle element

c) last element

Also, highlight which subarrays that need to be sorted using a recursive call. (for example by drawing a line under each subarray)

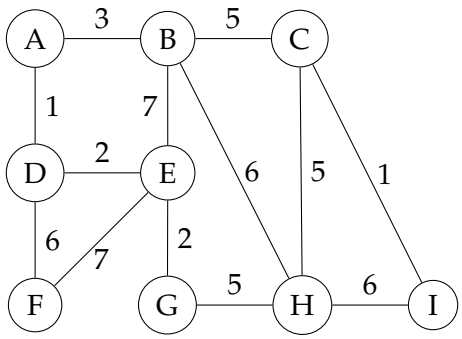
For a VG only:

d) Quicksort has an average case runtime of $O(n \log n)$, but a worst case complexity of $O(n^2)$. Describe how the choice of a pivot element influences the performance of the quicksort algorithm. What precautions are possible?

e) Compare quicksort to mergesort, what are the similarities and differences?

Exercise 4

You are given the following undirected weighted graph:



a) Compute a minimal spanning tree for the following graph by manually performing Prim's algorithm using A as starting node.

Your answer should be the set of edges which are members of the spanning tree you have computed. The edges should be listed in the order they are added as Prim's algorithm is executed. Refer to each edge by the labels of the two nodes that it connects, e.g. DF for the edge between nodes D and F.

b) **For a VG only:** Suppose we perform Dijkstra's algorithm starting from node F. In which order does the algorithm visit the nodes, and what is the computed distance to each of them?

Exercise 5

Consider the following hash table implemented using linear probing, where the hash function is the identity, $h(x) = x \bmod 12$, where mod is the modulo operator that calculates the remainder of a division.

24	XX		15		29	18		20		10	35
0	1	2	3	4	5	6	7	8	9	10	11

a) The value that was previously at index 1 has been deleted, which is represented by the XX in the hash table.

Which value might have been stored there, before it was deleted? There may be several correct answers, and you should write down all of them.

A) 26

B) 9

C) 21

D) 38

E) 11

F) 36

b) **For a VG only:**

if we want a hashtable that stores a set of strings, one posible hash function is the string's length, $h(x) = x$.length. Is this a good hash function? Explain your answer.

Exercise 6

Design a data structure for storing a set of integers. It should support the following operations:

- `new`: create a new, empty set
- `insert`: add an integer to the set
- `member`: test if a given integer is in the set
- `delete`: delete a given integer from the set
- `increaseBy`: add a given integer to all the integers in the set

For example, calling `increaseBy(2)` on a set containing the values $1, 2, 3, 4, 5$ should give a set containing the values $3, 4, 5, 6, 7$.

You may use existing standard data structures as part of your solution – you don't have to start from scratch.

Write down the data structure or design you have chosen, plus *pseudocode* showing how the operations would be implemented. The operations must have the following time complexities:

- **For a G**:

  $O(1)$ for `new`,
  $O(\log n)$ for `insert/member/delete`,
  $O(n)$ for `increaseBy`
  (where n is the number of elements in the set)

- **For a VG**:

  as for G but `increaseBy` must be $O(1)$