Exam – Datastrukturer

DIT960 / DIT961, VT-18
Göteborgs Universitet, CSE

Day: 2017-05-31, *Time:* 8:30-12.30, *Place:* SB-MU

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 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, 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

Consider the following algorithm that *sorts* a list of n elements by using a priority queue:

```
public static void sort(List<Integer> xs) {
  PriorityQueue<Integer> pq = new PriorityQueue<>();

  for (Integer x : xs)
    pq.insert(x);

  while (!pq.empty()) {
    Integer x = pq.deleteMin();
    System.out.println(x);
  }
}
```

What is the worst-case time complexity of this algorithm, if the priority queue is implemented using:

a) an ordinary binary search tree?

b) an AVL tree?

c) an unsorted linked list?

d) a 2-3 tree?

e) a sorted array?

You may assume that printing out a value takes constant time.

Recall that you can find the smallest value in a binary search tree by starting at the root and following the left child until you find a node without a left child – this is the node with the smallest value.

The complexity should be expressed in terms of n, the size of the input list. 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.

Exercise 2

Your task is to implement a *map* from keys to values in Haskell using a binary search tree. Your solution should define a type `Map k v` that represents a map from keys `k` to values `v`, together with two functions:

```
data Map k v = ...

lookup :: Ord k => k -> Map k v -> Maybe v
update :: Ord k => k -> v -> Map k v -> Map k v
```

The `lookup` function looks up a key in the map, while `update` adds a key/value pair to the map, or updates the value if the key already exists in the map.

You may like to take inspiration from the following Haskell code which implements a *set* using a binary search tree.

```
data BST a
  = Empty
  | Node (BST a) a (BST a)

member :: Ord a => a -> BST a -> Bool
member x Empty = False
member x (Node l y r)
  | x < y     = member x l
  | x > y     = member x r
  | otherwise = True

insert :: Ord a => a -> BST a -> BST a
insert x Empty = Node Empty x Empty
insert x t@(Node l y r)
  | x < y     = Node (insert x l) y r
  | x > y     = Node l y (insert x r)
  | otherwise = t
```

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 map, and have the following type:

```
delete :: Ord k => k -> Map k v -> Map k v
```

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 define and use a function that finds/deletes the smallest (alternatively the greatest) element when implementing `delete`.

Exercise 3

Which array out of A, B and C represents a binary heap? Only one answer is right.

A

3	12	23	10	15	35	45	15	18	20	21	
0	1	2	3	4	5	6	7	8	9	10	11

B

2	13	20	21	65	54	67	41	30	83	52	
0	1	2	3	4	5	6	7	8	9	10	11

C

3	8	22	11	43	79	87	32	13	50	49	
0	1	2	3	4	5	6	7	8	9	10	11

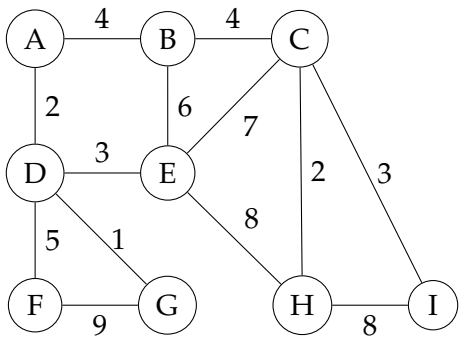
a) Write the heap out as a binary tree.

b) Add 19 to the heap, making sure to restore the heap invariant. How does the array look now?

c) **For a VG only:** Describe the procedure how we can build a heap from an arbitrary array. Answer either in pseudocode or English (or Swedish).

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, $h(x) = x \bmod 10$, where mod is the modulo operator that calculates the remainder of a division.

10	XX	2	33		5	16			19
0	1	2	3	4	5	6	7	8	9

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) 20

D) 38

E) 41

F) 13

b) **For a VG only:**

Hash tables typically have better performance than balanced binary search trees. Even so, both are widely used in practice. One reason is that a hash table does not support all the operations that a BST does.

Give an example of an operation which can be efficiently implemented for a binary search tree but not for a hash table.

Exercise 6

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

- `new`: create a new, empty set
- `insert`: add a string to the set
- `member`: test if a given string is in the set
- `delete`: delete a string from the set
- `deleteLarge`: delete all strings with more than 42 characters

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`,
  $O(n \log n)$ for `deleteLarge`
  (where n is the number of elements in the set)

- **For a VG**:

  as for G but `deleteLarge` must take $O(1)$ time