

Introduction to Algorithms and Data Structures

IT University of Copenhagen

June 17, 2002

This exam consists of 3 exercises containing in total 13 subexercises. Each of these 13 subexercises is given the same weight in the evaluation. The exam consists of 6 pages. You have 4 hours to complete the exam. Remember to number the pages and write your name and CPR number on every page.

For exercises in which algorithms must be specified, the asymptotic time complexity of the specified solution will be taken into account when grading. Exercises asking for time complexity must be answered using O -notation. It is weighted in the evaluation that growth rates in the O -notation are expressed with least possible asymptotic growth.

Exercise 1

This exercise is about rooted binary trees. The representation and notation for binary trees is similar to section 10.4 pp. 214-215 in CLRS.

Consider the following procedure, which as input takes the root x of a binary tree. We assume that all nodes x have a field $size[x]$ which contains an integer.

ZERO(x)

```
1 if  $x \neq \text{NIL}$ 
2   then  $size[x] \leftarrow 0$ 
3       ZERO( $right[x]$ )
4       ZERO( $left[x]$ )
```

The idea is that after the execution of the procedure all fields $size[x]$ will be 0.

a) Give the time complexity of the procedure ZERO(x) in O -notation, where x is the root of a tree with n nodes.

In the following subexercises we let $T(x)$ denote the *subtree rooted at x* in tree T cf. CLRS pp. 1087. Assume now that we want a procedure INITSIZE(x), which given the root x of a tree with n nodes initializes $size[y]$ to be the number of nodes in the subtree $T(y)$ for all nodes y in T .

b) Give the pseudocode for INITSIZE(x) such that the running time is $O(n)$.

For a tree T , we say that an edge (u, v) with u parent to v is *green* if the number of nodes in $T(u)$ is at least two times the number of nodes in $T(v)$. That is, after the execution of INITSIZE, $size[u] \geq 2size[v]$ for all green edges (u, v) .

c) Describe a procedure which calculates the number of green edges for the entire tree T . Give the time complexity of your procedure and argue for your answer. Do also argue for the correctness of your algorithm.

d) Using O -notation, give an upper bound on the number of green edges on a path from a node to the root of T . Argue for your answer.

Consider the following procedure.

GREENPATHSUM(x)

1 **if** $x \neq \text{NIL}$ and $\text{left}[x] \neq \text{NIL}$ and $\text{right}[x] \neq \text{NIL}$

2 **then for** $i \leftarrow 1$ **to** $\text{size}[x]$

3 **do** $\text{timecount} \leftarrow \text{timecount} + 1$

4 **if** $\text{size}[x] \geq 2\text{size}[\text{left}[x]]$

5 **then** GREENPATHSUM($\text{left}[x]$)

6 **else** GREENPATHSUM($\text{right}[x]$)

e) Give the time complexity of GREENPATHSUM(x). Argue for your answer.

Exercise 2

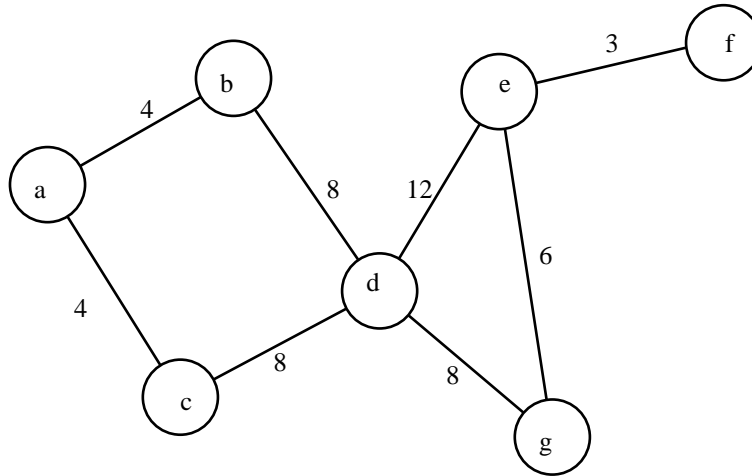


Figure 1: The graph H

a) Give the edges in a minimum spanning tree for the graph H in figure 1.

b) Give the adjacency-list representation for H in the same way as Figure 22.1 pp. 528 in CLRS.

Let G be a connected, undirected graph with edge weights $w(u, v)$ defined as the product of the degrees (as defined on page 1081 in CLRS) of the nodes u and v . That is, $w(u, v) = \deg(v) \cdot \deg(u)$, where $\deg(u)$ and $\deg(v)$ denotes the degree of node u and node v respectively. In the graph H in figure 1 the edge weights are defined in this way.

c) Give an efficient algorithm to compute a minimum spanning tree for graphs where the edge weights are defined as in G above.

d) Let G be a given *directed* graph with positive edge weights and let v be a given node v in G . Describe an algorithm that computes the shortest simple cycle in G (i.e., *simple cycle* as in CLRS p. 1081) that contains v . Give the time complexity of your algorithm and argue for the correctness of your algorithm.

Exercise 3

A heap with five elements with the values 1, 3, 4, 6 og 12 is shown in Figure 2.

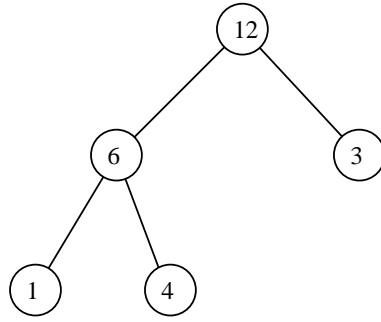


Figure 2: The heap B

a) Draw the heap B after execution of a HEAP-EXTRACT-MAX operation.

The following subexercises b) - d) are about extending the number of operations on heaps to include operations other than the ones described on page 129 in CLRS. The existing operations must still be supported by the heap within the same time complexity as on page 129 in CLRS. In the following we assume that the values of the elements are positive real numbers for which we can perform addition and division in constant time.

Consider the following operation.

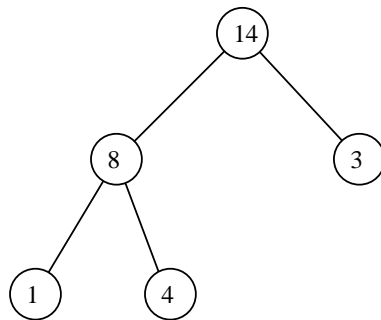
FUSION (A, x, y): Removes from heap A the elements x and y (i.e., with the value $A[x]$ and $A[y]$) and adds a new element with the value $A[x] + A[y]$.

b) Give an algorithm for the operation **FUSION**. Argue for the running time.

Consider the following new heap operation.

ADDUPTO (A, q, v): Adds the positive value v to all elements y in the heap A , where $A[y] \geq q$.

As an example, performing **ADDUPTO**($B, 6, 2$) on the heap B in figure 2 will give the new heap:

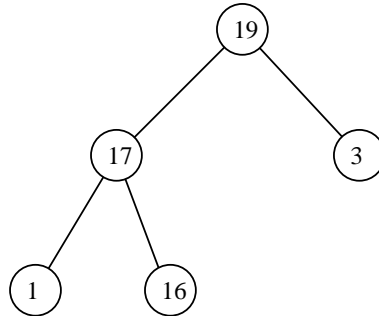


c) Describe an algorithm for the operation `ADDDUPTO` such that the time complexity is $O(k)$, where k is the number of elements which change value after the execution of the operation.

Consider the following operation:

`ADDFUNNY`(A, k): Adds the value $\frac{60}{A[x]}$ to the value of an element x if x is among the k largest elements in the heap A .

As an example, after the execution of `ADDFUNNY`(3) the heap B in figure 2 will contain five elements with the values 1, 3, 16, 17 and 19 since the three previously largest elements 12, 6 and 4 become 17, 16 and 19 respectively. After executing this operation, the heap could look as follows:



d) Describe an algorithm for the operation `ADDFUNNY`(k) such that the time complexity is $O(k \lg k)$. The time complexity of the other operations must be maintained.