

# Introduction to Algorithms and Data Structures

IT University of Copenhagen

June 4, 2003

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.

CLRS refers to “Introduction to Algorithms” by Cormen, Leiserson, Rivest and Stein, Second Edition, 2001. 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 and heaps. The representation and notation for binary trees is similar to section 10.4 pp. 214-215 in CLRS.

We assume that all nodes  $x$  in the trees we consider have a field  $key[x]$  which contains an integer. Furthermore,  $A$  is an  $n$ -element max-heap as defined in Chapter 6 in CLRS, initially empty before the call to `TREETOHEAP`. The procedure uses the operation `MAX-HEAP-INSERT` as defined on page 140 in CLRS. Consider the following procedure, which as input takes the root  $x$  of a binary tree with  $n$  nodes.

```
TREETOHEAP( $x$ )
1 if  $x \neq \text{NIL}$ 
2   then MAX-HEAP-INSERT( $A, key[x]$ )
3     TREETOHEAP( $right[x]$ )
4     TREETOHEAP( $left[x]$ )
```

**a)** Give the time complexity of the procedure `TREETOHEAP( $x$ )` in  $O$ -notation.

In the subexercises b) and c) below we assume that the tree with root  $x$  satisfy the *binary-search-property*.

**b)** Describe an efficient algorithm for finding the closest pair of nodes in a tree  $T$  with root  $x$ , *i.e.*, return two nodes  $y$  and  $z$  in  $T$  such that the difference between  $key[y]$  and  $key[z]$  is smallest possible among all pairs of nodes in  $T$ . Give the time complexity of your algorithm and justify your answer.

**c)** Describe an efficient algorithm for a procedure `SEARCHTREETOHEAP( $A, x$ )` that like `TREETOHEAP` inserts all keys from the tree to heap  $A$  using the `MAX-HEAP-INSERT` operation  $n$  times. Give the time complexity of your algorithm and justify your answer.

Consider the data structure  $S$  that supports the following three operations.

`INSERT( $x$ )` inserts element  $x$  into  $S$  with key  $key[x]$ .

`DELETE( $x$ )` removes element  $x$  from  $S$ .

`CLOSESTPAIR()` returns two different elements  $x, y \in S$  such that  $key[x] - key[y]$  is least possible.

**d)** Make a data structure that supports the above three operations in worst-case time  $O(\log n)$  per operation, where  $n$  denotes the actual size of  $S$ . Justify your answer.

## Exercise 2

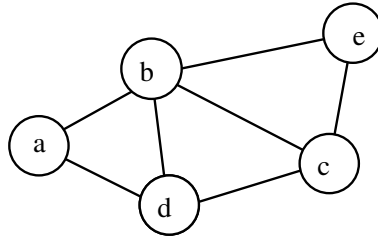


Figure 1: The graph  $H$

a) Draw a spanning tree of the above graph  $H$ .

In subexercises b), c) and d) below we consider a graph  $G$  with  $n$  fixed nodes  $V = \{1, 2, \dots, n\}$ , and initially without any edges. This graph is maintained by a number of operations given below. Furthermore, we assume that each edge in  $G$  is colored either blue or white. Consider the following four operations.

INSERTEDGE( $u, v$ ) inserts a *white* edge between node  $u$  and  $v$ .

COLORAROUNDNODE( $v$ ) colors all white edges incident to  $v$  *blue*.

COLOREDGE( $u, v$ ) colors the edge  $(u, v)$  *blue* (keep it blue if it already is blue).

BLUE( $u, v$ ) returns true if and only if the edge  $(u, v)$  is *blue*.

b) Give an efficient algorithm that supports the above four operations aiming at fastest possible amortised time per operation. Give both worst-case and amortised time complexity of your algorithm and justify your answer.

We say that two nodes  $u$  and  $v$  in  $G$  are *connected by a blue path* if there is a path between  $u$  and  $v$  with only blue edges. Furthermore, the *blue component of  $v$*  is the set of nodes which are connected to  $v$  by a blue path including  $v$ . Consider the following additional operation.

BLUEPATH( $x, y$ ). Return *true* if and only if  $x$  and  $y$  are connected by a blue path.

c) Give an efficient algorithm that supports all of the above five operations aiming at fast amortised time per operation. Give both worst-case and amortised time complexity of your algorithm and justify your answer.

Consider the following additional operation.

`REMOVEBLUECLOUD( $x$ )`. Delete all *blue* edges between pairs of nodes in the blue component of  $x$ .

**d)** Give an efficient algorithm that supports all of the above six operations aiming at fast amortised time per operation for operation sequences of length  $m \leq n$ . Give both worst-case and amortised time complexity of your algorithm and justify your answer.

## Exercise 3

This exercise is about weighted, directed graphs. We consider a special case of such graphs, which we will call *1-2 layered graphs*. A 1-2 layered graph is a directed graph  $G = (V, E)$  where the nodes except two special nodes  $s, t \in V$  can be partitioned into  $k$  disjoint sets  $L_1, L_2, \dots, L_k$  called *layers* such that the following holds. For  $1 \leq i \leq k-1$ , each node  $v$  in layer  $i$  has an edge to exactly two nodes in layer  $L_{i+1}$ , and there is no other edge from  $v$  to a node elsewhere. Furthermore, node  $s$  has an edge to all nodes in layer  $L_1$  and no else, and all nodes in layer  $L_k$  have an edge to node  $t$  and no else. The graph in Figure 2 is thus an example of an 1-2 layered graph with three layers of nodes.

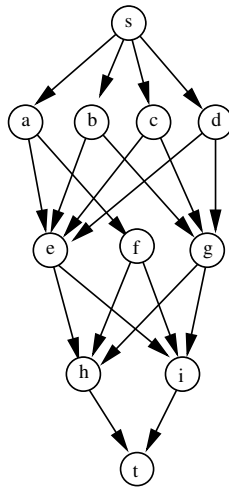


Figure 2: An example of an 1-2 layered graph  $H'$

**a)** List the nodes in a sequence corresponding to a *depth-first* traversal of the  $H'$  starting from  $s$ .

In subexercise b) and c) below, Dijkstra's algorithm refers to the version described in CLRS Chapter 24.3 pp. 595-599.

**b)** Give the time complexity for finding the shortest path between  $s$  and  $t$  in an 1-2 layered graph with  $n$  nodes using Dijkstra's algorithm assuming  $E$  only has positive edge weights.

**c)** Give a counterexample of an 1-2 layered graph with both positive and negative edge weights, where Dijkstra's algorithm fails to find the shortest path.

**d)** Given a graph  $G = (V, E)$ , make an efficient algorithm that decides if  $G$  is an 1-2 layered graph. Give the time complexity of your algorithm and justify your answer.

e) Describe an efficient algorithm that finds a subset  $E' \subseteq E$  of edges of an 1-2 layered graph  $G = (V, E)$  forming a *shortest path tree* with root  $s$  such that the sum of edge weights in  $E'$  is smallest possible. Give the time complexity of your algorithm and justify your answer.