

# Introduction to Algorithms and Data Structures

IT University of Copenhagen

10 June 2005

This examination assignment consists of 4 exercises with a total of 12 subexercises. Subexercises are given equal weight in the grading. You have 4 hours to answer all 12 subexercises. Remember to write the page number, your name and your CPR number on each page of your written answer. The complete assignment consists of 7 numbered pages, including this one.

CLRS refers to “Introduction to Algorithms” by Cormen, Leiserson, Rivest and Stein, Second Edition, 2001.

For exercises that ask for efficient algorithms the asymptotic complexity of the specified solution will be taken into account when grading. Exercises asking for time complexity must be answered using  $O$ -notation with least possible asymptotic growth.

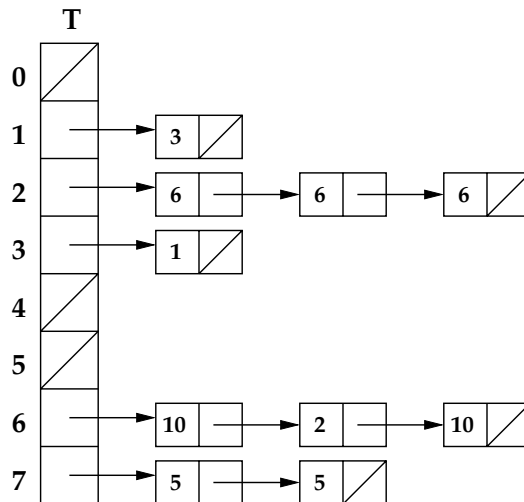
# 1 Multisets

In this exercise we consider the problem of storing a *multiset*. A multiset is a collection of elements where each element can occur one or more times, i.e. it is similar to a set, but in a set each element only occurs once.

We want a data structure that supports the operations  $\text{INSERT}(x)$ ,  $\text{DELETE}(x)$ , and the query  $\text{ISINMULTISET}(x)$ .  $\text{INSERT}(x)$  inserts element with key  $x$  into the multiset,  $\text{DELETE}(x)$  deletes one copy of the element with key  $x$  from the set (provided  $x$  is in the multiset), and  $\text{ISINMULTISET}(x)$  returns 'true' if there is at least one element with key  $x$  in the multiset, and 'false' otherwise. There is no additional information stored together with the keys.

One way to implement the data structure is to use a hash table. Assume that the keys are positive integer numbers and that the size of the hash table is  $N$  and the number of elements in the multiset is at most  $m$ . Use the hash function  $h(x) = 3 \cdot x \pmod N$ .

The data structure for the multiset  $M = \{1, 2, 3, 5, 5, 6, 6, 6, 10, 10\}$ , and  $N = 8$  may look like this:



**1.1** Draw the data structure after the following operations have been performed (in the given order):  $\text{INSERT}(7)$ ,  $\text{DELETE}(4)$ ,  $\text{INSERT}(9)$ ,  $\text{INSERT}(4)$ ,  $\text{DELETE}(10)$ . Insertion inserts the element first in the list. Deletion deletes the first occurrence in the list.

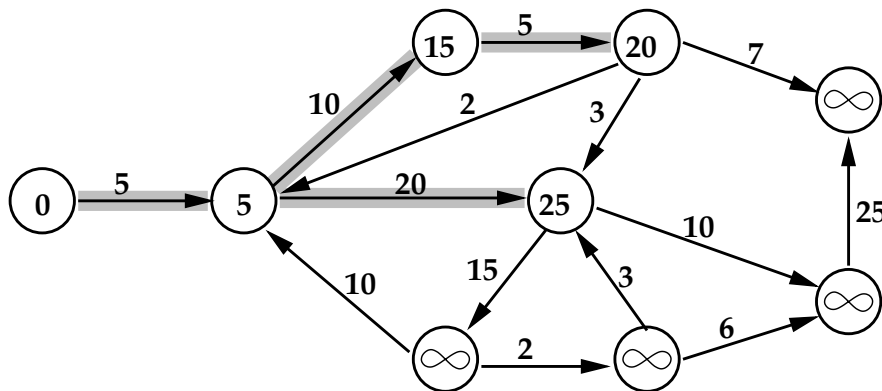
**1.2** Write the pseudocode for  $\text{ISINMULTISET}(x)$ , and give the worst case time complexity in terms of  $m$  and  $N$ . Choose your own implementation of linked lists.

**1.3** The above data structure may be inefficient if we have many equal values. Describe an alternative data structure using linear space in  $m$  and worst case time  $O(1)$  for  $\text{INSERT}(x)$ ,  $\text{DELETE}(x)$ , and  $\text{ISINMULTISET}(x)$ , when the number of possible keys is  $O(m)$  (or in other words, all keys have values between 1 and  $O(m)$ ). Note that we only store keys and no associated information.

## 2 Around Graphs

In the course we have seen different algorithms for finding shortest paths in graphs, Breadth-First Search (BFS), the Bellman-Ford algorithm, and Dijkstra's algorithm.

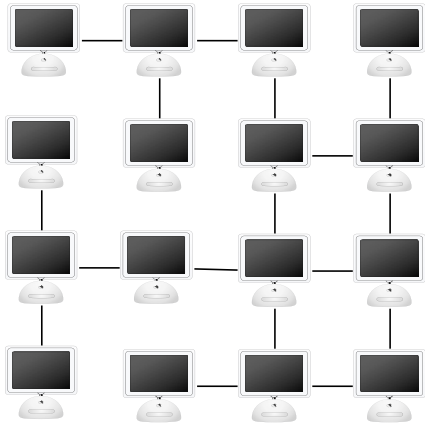
**2.1** In Dijkstra's algorithm a nested loop relaxes edges in a specific order. The following figure illustrates a graph during the execution of Dijkstra's algorithm, in the same way as figure 24.6, on page 596 in CLRS. Perform the next iteration of the while-loop, lines 4-8 in the pseudocode for Dijkstra's algorithm, on page 595 in CLRS, and draw the resulting graph, including  $d$  values and  $\pi$  values (shaded edges). The edges shaded so far are  $(0,5)$ ,  $(5,15)$ ,  $(15,20)$ , and  $(5,25)$ .



**2.2** Below three shortest path problems are described. For each of them, choose one of the above algorithms that you recommend to use to solve the problem. Give a short motivation for each of your choices. (No motivation gives no points.)

**Problem 1:** Consider a graph that models a landscape. Vertices represent a set of locations, and edges model the difference in altitude (altitude is defined as the position of given point above the sea level). We want to decide the difference in altitude from one specified location to all other locations.

**Problem 2:** 10000 computers are connected in a network. They are organized in a 100 by 100 mesh (see figure of 4 by 4 mesh), so that a computer is connected to at most 4 other computers. Some of the wires are broken, and we want to compute the shortest route for one computer to send a message to any of the other computers. Assume that the time to send a message along a route is proportional only to the number of computers it passes on its way, and does not depend on the lengths of the physical wires.

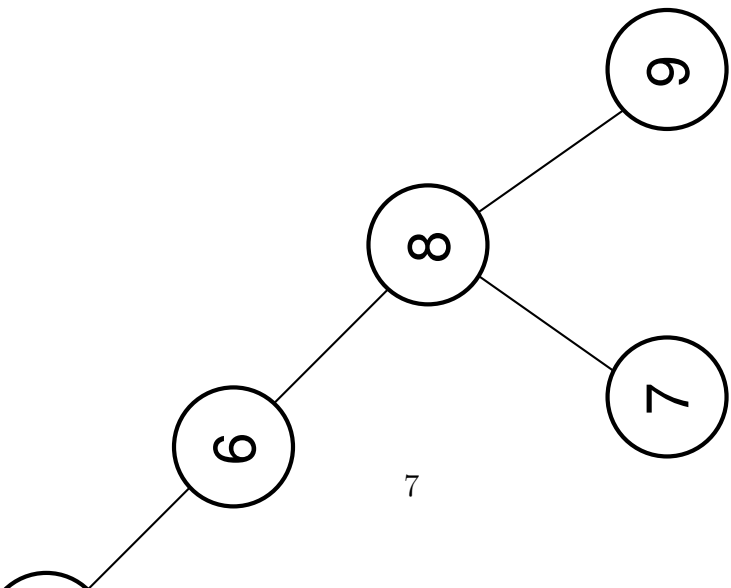
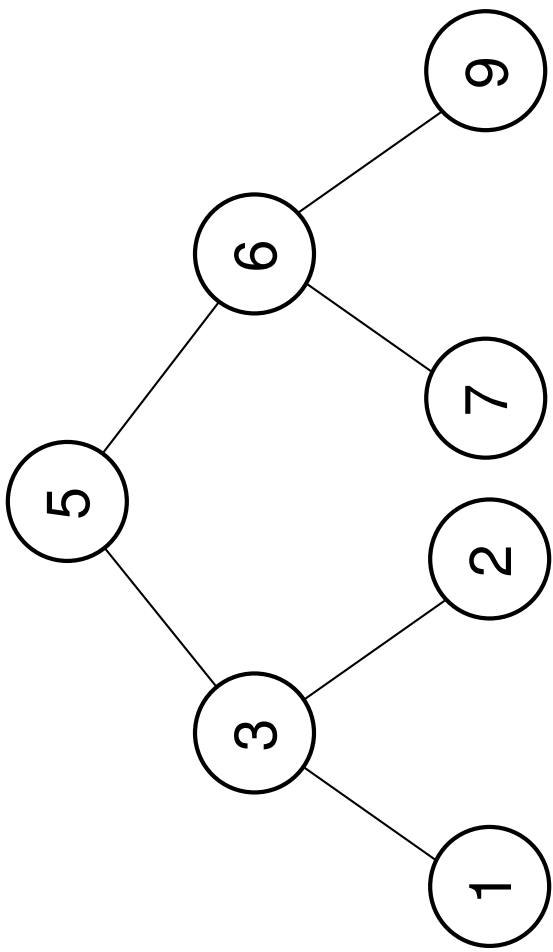


**Problem 3:** An airline company has a graph that models the routes it offers, where vertices represent airports and edges direct connections. The weights of the edges represent the time it takes to fly from one airport to the other. Note that the time to go from A to B does not have to be the same as going from B to A. We want to compute the shortest flying time (excluding the time spent waiting for the next plane at an airport) from Copenhagen Airport to all other airports.

**2.3** The diameter of a graph is defined as the longest distance between two vertices in a graph. In an unweighted graph the distance means the number of edges. Describe an algorithm for computing the diameter of an unweighted and connected graph in time  $O(VE)$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Describe the algorithm without using pseudocode. An analysis of the running time should be included. An algorithm with running time higher than  $O(VE)$  will also give you some points.

### 3 Trees and Colours

**3.1** Which of the following binary trees are AVL trees? Motivate your answer briefly, for all the trees that fail to be AVL trees.



(c)

We are interested in colouring trees. A good colouring of a tree is such an assignment of colours to nodes that any two neighbours (child-parent pairs) in the tree are coloured by different colours. Any tree can be well coloured using just two different colours.

**3.2** Assume that an insertion has been made into a well-coloured AVL tree that has caused some nodes to be unbalanced. If this tree is well coloured, and a single rotation needs to be applied, will it preserve the goodness of colouring? (or will it make the tree not well coloured?) How about double rotation? Give arguments why.

A multitude of cars is parked at the parking lot in front of ITU every day. One of the researchers is very keen on monitoring how many car colours are represented on the lot. She wants to built a system that would automatically count the number of colours present.

Unfortunately the researcher does not have access to any fancy equipment. The only useful device in her possession is an old-fashioned web-camera that cannot properly assess colours. Nevertheless the camera, when mounted on the side of the fancy ITU building, can distinguish whether any two cars have the same colours or not.

**3.3** Help the researcher to design a car park watching system: design an algorithm NUMBER-OF-COLOURS that given an array  $C$  of  $n$  cars returns the number of colours  $k$  that can be found among these cars. Remember that the only test you can make on colours of two cars is checking whether they are equal or not. Assume that the cars are not moving during the operation of the algorithm. Give the asymptotic time complexity of your algorithm.

## 4 Around Sorting

**4.1** What is the worst case asymptotic running time of RADIX-SORT (CLRS section 8.3), if we replace COUNTING-SORT by MERGE-SORT in its implementation? Express your answer in terms of  $n$ —the number of integers to sort,  $d$ —the number of digits in each of the integers, and  $k$ —the number of values that each digit can take. Explain your reply briefly.

**4.2** Let  $S_1, S_2, \dots, S_m$  be arrays of integer numbers from 1 to  $k$ , such that:  $\sum_{i=1}^m |S_i| = k$  (the sum of all their sizes is  $k$ ). Modify COUNTING-SORT in order to obtain a sorting algorithm, which is able to sort the entire sequence of arrays in total time  $O(k)$ .

You are given a collection of  $n$  bolts (Fig. 1) of different widths and  $n$  corresponding nuts. There are no two bolts of the same size, nor two nuts of the same size. However each bolt enjoys a nut with a corresponding size in the collection.

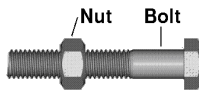


Figure 1: Domain description: a bolt and a nut.

The difference sizes of nuts and bolts can not be decided by looking at them since the differences in sizes are very subtle. However by taking any nut and any bolt you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. There is no direct way to compare two nuts together or two bolts together.

**4.3** What is the most efficient algorithm that you can propose, which matches each bolt to its nut? Describe your algorithm in English or using pseudocode. Remember to comment on the correctness of your algorithm.