

CONCORDIA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6651/4: Algorithm Design Techniques

Fall 2014

MidTerm - Close book exam - 2:30 hours

Instructor: Professor B. Jaumard
Office: EV 003.115
Email: bjaumard@cse.concordia.ca

First Name	Last Name	ID#
------------	-----------	-----

Question 1. (Dynamic Programming) (20 points.)

You are helping to run a high-performance computing system capable of processing several terabytes of data per day. For each of n days, you are presented with a quantity of data: on day t , you're presented with x_t terabytes. For each terabyte you process, you receive a fixed revenue, but any unprocessed data becomes unavailable at the end of the day. Your system is running with software that is not totally reliable, and so the amount you can process goes down with each day that passes since the most recent reboot of the system. On the first day after a reboot, you can process s_1 terabytes, on the second day after a reboot you can process s_2 terabytes, and so on down to s_n ; we assume $s_1 > s_2 > s_3 > \dots > s_n$. (Of course you can only process up to x_t terabytes on day t now matter how fast your system is.) To get the system back to peak performance, you can choose to reboot it, but on any day you choose to reboot the system, you can't process any data at all. Given the amounts of available data for the next n days $x_1, x_2, x_3, \dots, x_n$ and given the profile of your system as expressed by s_1, s_2, \dots, s_n (and from a freshly rebooted system on day 1) choose the days on which you're going to reboot so as to maximize the total amount of data you process.

Example. Suppose $n = 4$, and the values of x_i and s_i are given by the following table.

	Day 1	Day 2	Day 3	Day 4
x	10	1	7	7
y	8	4	2	1

The best solution would be to reboot on day 2 only; this way, you process 8 terabytes on day 1, then 0 on day 2, then 7 on day 3, then 4 on day 4, for a total of 19. Note that if you don't reboot at all, you'd process $8 + 1 + 2 + 1 = 12$; and other rebooting strategies give you less than 19 as well.

(a) Give an example of an instance with the following properties.

- There is a surplus of data in the sense that $x_i > s_i$ for every i
- The optimal solution reboots the system at least twice

In addition to the example, you should say what the optimal solution is.

(b) Give an efficient algorithm that takes values for x_1, x_2, \dots, x_n and $s_1, s_2, s_3, \dots, s_n$ and returns the total number of terabytes processed by an optimal solution.

Question 2. (20 points.)

In the art gallery guarding problem, we are given a line L that represents a long hallway in an art gallery. We are also given a set $X = \{x_1, x_2, \dots, x_n\}$ of real numbers that specify the positions of paintings in this hallway. Suppose that a single guard can protect all the paintings within distance at most 1 of his or her position (on both sides). Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard on the paintings with positions in X . Prove that your algorithm does provide the optimal value. What is the complexity of your algorithm?

Question 3. (20 points)

Imagine that you are working on creating a flow for a set of packets in a media stream. So you are given a network G , with a source s and sink t together with bandwidth constraints on each edge, which indicate the maximum speed that the communication link, represented by that edge, can support. Your goal is to produce a maximum flow from s to t , respecting the bandwidth constraints on the edges. Suppose now, however, that you also have a bandwidth constraint on each router in the network, which specifies the maximum amount of information, in bits per seconds, that can pass through that node. Describe an efficient algorithm for finding a maximum flow in the network, G , that satisfies the bandwidth constraints on the edges as well as the nodes. What is the running time of your algorithm? You need to provide a detailed justification of your answers.

Question 4. (20 points)

Suppose that you are the owner of a chain of premium coffee shops that sell high-priced coffee with fancy Italian names to college students. You have learned that there is a street in a large college town that is lined with n dormitories and there currently is no coffee shop on this street. Your goal is to place a new coffee shop on this street, so as to optimize the distance from this shop to the various dormitories. To simplify things, let us model the street as a line and each dormitory as a point, d_i , which is a real number on this line. In addition, we know that the number of people, p_i , who live in each dormitory, d_i . You are interested in finding the location, x , that minimizes the cost function:

$$\sum_{i=1}^n p_i |d_i - x|.$$

Describe an efficient algorithm for finding the point x , where to place your coffee shop, that minimizes this cost. What is the running time of your algorithm?

If you happen to use an algorithm we saw during one of the lectures, you need to cite it, but also to describe in detail.

Question 5. Recurrence relation. (20 points)

A very common problem in computer graphics is to approximate a complex shape with a bounding box. For a set S of n points in 2-dimensional space, the idea is to find the smallest rectangle, R , with sides parallel to the coordinate axes that contains all the points in S . Once S is approximated by such a bounding box, we can often speed up lots of computations that involve S . For example, if R is completely obscured by some object in the foreground, then we don't need to render any of S . Likewise, if we shoot a virtual ray and it completely misses R , then it is guaranteed to miss S . So doing comparisons with R instead of S can often save time. But this savings is wasted if we spend a lot of time constructing R , for a set, S , in the plane. Note that the construction of R can be reduced to two instances of the problem of simultaneously finding the minimum and the maximum in a set of n numbers; namely, we need only do this for the x -coordinates of S and then for the y -coordinates of S . Therefore, design a divide-and-conquer algorithm for finding both the minimum and the maximum element of n elements using no more than $3n/2$ comparisons.

Hint. Establish a divide and conquer recurrence relation whose solution corresponds to the number of comparisons. Solve it using the method with the characteristic equation.