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.)

Source: Book of Kleinberg and Tardos [2], Exercice 9 page 320.

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 $s_1 > s_2 > s_3 > \cdots > s_n$. (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 $x_1, x_2, x_3, \ldots, x_n$ and given the profile of your system as expressed by $s_1, s_2, \ldots, s_n$ (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
s	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 $x_1, x_2, \ldots, x_n$ and $s_1, s_2, s_3, \ldots, s_n$ and returns the total number of terabytes processed by an optimal solution.

Solution

	Day 1	Day 2	Day 3	Day 4	Day 5
x	10	10	10	10	10
s	8	4	3	2	1

No reboot: value of the solution $= 8 + 4 + 3 + 2 + 1 = 18$.
Reboot on Day 2 and on Day 4: value of the solution $= 8 + 0 + 8 + 0 + 8 = 24$.

Why the solution is optimal:
if you wait two days before rebooting, instead of $0 + 8$, you get $4 + 3$, which is less, and it goes on decreasing if you wait longer for rebooting.

Suggestion: when asked for an example, take it as simple as possible.

An observation
As $s_1 > s_2 > s_3 > \cdots > s_n$, the value is always better with a reboot (choose any day) than if you never reboot.

Algorithm
$j \leftarrow 1$
$\text{DATA}^0 = 0$
Reboot before Day 1 (no reboot on Day 1): $\text{DATA}^1 = \min\{s_1, x_1\}$
$t \leftarrow 2$
$\star - - \star$ Two cases to consider:
$\star - - \star$ *(i)* you reboot on the previous day (Day $t-1$) or
$\star - - \star$ *(ii)* you reboot on a previous day (Day j)
$\text{DATA}^t \leftarrow \max\left\{\text{DATA}^{t-2} + \min\{s_1, x_t\}; \text{DATA}^{t-1} + \min\{x_t, s_j\}\right\}$
$\star - - \star$ Update the index of the reboot day accordingly
if $\text{DATA}^t = \text{DATA}^{t-2} + \min\{s_1, x_t\}$
then $j \leftarrow 1$
else $j \leftarrow j + 1$

Complexity: $O(n)$

Grading Scheme

Example: 5 points (3 for an example that satisfies the requirements, 2 for the optimality reasoning)
Algorithm and its justification: 10 points
Complexity (including its justification): 5 points.

Question 2. (20 points.)

Source: Book of Goodrich and Tamassia [1], Exercice A-10.1 page 300.

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 $X = \{x_1, x_2, \ldots, x_n\}$ 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 you algorithm does provide the optimal value. What is the complexity of your algorithm?

Solution

See Exercise 6.6 in the coursepack, a different story, but it corresponds to the same mathematical problem. Although the solution in the coursepack does not provide a pseudo-code, make sure you are able to write one.

Grading Scheme

Greedy algorithm: 5 points
Proof of optimality: 10 points
Complexity (including its justification): 5 points.

Be careful: you may not have a continuum of paintings. For instance, in the following case

$$x_1 = 0, x_2 = 7, x_3 = 7.1, x_4 = 7.3, x_5 = 10,$$

you need three guards.
A first possible location (given potentially by your greedy algorithm is:

$$g_1 = 1, g_2 = 8, g_3 = 11.$$

Another optimal solution is:

$$g'_1 = 0, g'_2 = 7, g'_3 = 10.$$

Question 3. (20 points)

Source: Book of Goodrich and Tamassia [1], Exercice A-16.4 page 471.

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.

Solution

See Exercise 7.1 in the coursepack, in particular the node transformation described in Figure 7.2

Grading Scheme

Node transformation: 5 points
Ford & Fulkerson's Algorithm: 10 points (5 points for the algorithm, 5 points for explaining how to construct the reduced graph)
Complexity (including its justification): 5 points.

Question 4. (20 points)

Source: Book of Goodrich and Tamassia [1], Exercice A-9.7 page 281.

Suppose that 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.

Solution

See Exercise 5.1 in the coursepack.

Grading Scheme

Definition of the weighted median problem. 2 points
Show that the above problem reduces to the weighted median problem, see question **d** in Exercise 5.1 of the coursepack: 8 points
Algorithm for computing the weighted median problem, see question **c** in Exercise 5.1 of the coursepack: 8 points
Complexity and its justification: 2 points.

Question 5. Recurrence relation. (20 points)

Source: Book of Goodrich and Tamassia [1], Exercice A-11.1 page 320.

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 dies 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 exemple, 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 his 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 the number of comparisons. Solve it using the method with the characteristic equation.

Indeed, the above problem amounts to answer the following (short) question:
Suppose we wish to find the minimum and maximum items in a list of numbers. How many comparisons does it take?

Solution

A natural approach is to try a divide and conquer algorithm.

Split the list into two sublists of equal size. (Assume that the initial list size is a power of two.) Find the maxima and minima of the sublists. Two more comparisons then suffice to find the maximum and minimum of the list.

Hence, if $T(n)$ is the number of comparisons, then $T(n) = 2T(n/2) + 2$. (The $2T(n/2)$ term comes from conquering the two problems into which we divide the original; the 2 term comes from combining these solutions.) Also, clearly $T(2) = 1$. Solving the recurrence relation with the method with the characteristic roots and a change of variables, we find $T(n) = (3n/2) - 2$, for n a power of 2.

Grading Scheme

Recurrence relation + reasoning for setting it: 5 points
Change of variable ($n = 2^k$): 2 points
General solution of the non homogeneous recurrence relation: 5 points
Particular & General Solution Non Homogeneous recurrence relation: 5 points
Change of variable (coming back to n): 3 points.

References

[1] M.T. Goodrich and R. Tamassia. *Algorithm Design and Applications.* Wiley, 2014.

[2] J. Kleinberg and E. Tardos. *Algorithm Design.* Addison Wesley, Boston, 2005.