Algorithms Exam TIN093/DIT602

Course: Algorithms

Course code: TIN 093 (CTH), DIT 602 (GU)

Date, time: 26th October 2019, 14:00–18:00

Building: L

Responsible teacher: Peter Damaschke, Tel. 5405

Examiner: Peter Damaschke

Exam aids: one A4 paper (both sides), dictionary,
printouts of the Lecture Notes (possibly with own annotations).

Time for questions: around 15:00 and around 16:30.

Solutions: will appear on the course homepage.

Results: will appear in ladok.

Point limits: CTH: 28 for 3, 38 for 4, 48 for 5; GU: 28 for G, 48 for VG;
PhD students: 38. Maximum: 60.

Inspection of grading (exam review):
Time will be announced on the course homepage.

Instructions and Advice:

- First read through all problems, such that you know what is unclear to you and what to ask the responsible teacher.
- Write solutions in English.
- Start every new problem on a new sheet of paper.
- Write your exam number on every sheet.
- Write legible. Unreadable solutions will not get points.
- Answer precisely and to the point, without digressions.
  Unnecessary additional writing may obscure the actual solutions.
- Motivate all claims and answers.
- Strictly avoid code for describing a complex algorithm.
  Instead *explain* how the algorithm works.
- If you cannot manage a problem completely, still provide your approach or partial solution to earn some points.
- Facts that are known from the course material can be used.
  You don't have to repeat their proofs.

Remark: The number of points is not always "proportional" to the length or difficulty of a solution, but it may also be influenced by the importance of the topics and skills.

Good luck!

Problem 1 (5 points)

Suppose that we have to compute the product y of n integers $x_1, \ldots, x_n$. Each of them has at most k digits.

For the complexity analysis we assume that operations with single digits are the elementary operations. We also assume that a standard algorithm for the multiplication of *two* integers is used, which costs $O(ab)$ time when the two factors have a and b digits, respectively. (There exist faster multiplication algorithms, but they are practical only for huge numbers.)

We also multiply the given factors in the most obvious way:

```
$y := 1$
for $i = 1$ to $n$ do $y := y \cdot x_i$
return $y$
```

Your task: Show that this algorithm takes $O(n^2k^2)$ time.

Hint: Think carefully. What can we say about the number of digits of the product of two integers?

Problem 2 (15 points)

Imagine a factory building consisting of just one long corridor. Think of the building simply as an interval $[s, f]$ on the real line. The building has no windows, but you can turn lights on and off. The i-th lamp ($i = 1, \ldots, n$) illuminates a certain segment (fornally: an "interval") $[s_i, f_i]$ of the corridor. The light switches for the i-th lamp are always located at s_i and f_i, and both can be used to turn the lamp on and off. No other light switches exist, and no remote control either. But you know the plan of the building, i.e., the set of the mentioned segments and hence also the locations of the switches.

Now you want to go through the whole corridor, from one end to the other end (in order to do some work steps or to fetch or put down some objects). For security reasons it is allowed to walk in illuminated segments only. Moreover, initially all lights are switched off, and when you leave the building, all lights must be switched off again.

Your task: Give an algorithm for traversing the corridor and switching the lights on and off, that minimizes the number of switches you have to press.

Some more clarifications: You can assume that some interval begins at s and some interval ends at f (otherwise, trivially, no solution is possible). The given segments can overlap, and a place is illuminated when it lies in at least one illuminated segment. All $2n$ numbers s_i and f_i are distinct, that is, no two light switches are at the same point.

2.1. First propose your algorithm. The easiest way is to describe it in a text. Well commented pseudocode is also acceptable, but please do not give code. In either case, the description must be comprehensible and unambiguous. (5 points)

2.2. Prove that your algorithm outputs a valid result and will, in fact, minimize the number of switches that must be pressed. The proof does not have to be formalized, but the crucial arguments must be there. (6 points)

2.3. How much computation time does your algorithm need? Provide enough details to explain your time bound. It should be a "small" bound, at most some polynomial in n. (4 points)

Problem 3 (10 points)

Consider the following generalized version of the String Editing problem: Besides "regular" symbols from an alphabet we also have *wildcard symbols*. A wildcard symbol can be aligned with any symbol from the alphabet, and this does not count as a mismatch. However, a wildcard aligned with a gap symbol is a mismatch. The goal is still to find an alignment of two given strings (consisting of regular symbols and perhaps wildcards) with a minimum number of mismatches.

Your task: Give an algorithm for String Editing with wildcards. It should run in $O(mn)$ time, for strings of lengths m and n.

You need not describe the entire algorithm and analysis from scratch. It suffices to say how you modify the known algorithm for String Editing, why this is correct, and why the time bound still holds true.

Problem 4 (10 points)

Looking at Problem 1 again, the visionary computer scientist Stanley Qubit gets the conjecture that multiplying n integers $x_1, \dots, x_n$, each with at most k digits, may work faster by divide-and-conquer. He proposes the following alternative algorithm:

For simplicity assume that n is a power of 2; the general case can then be handled by additional factors 1 that do nothing. Compute the two products $x_1 \cdot \ldots \cdot x_{n/2}$ and $x_{n/2+1} \cdot \ldots \cdot x_n$ recursively, and finally multiply them.

As in Problem 1, let us use only the simple standard algorithm that multiplies two integers with a and b digits in $O(ab)$ time.

4.1. For the analysis we consider the special case $k = 1$ first. That is, the problem is to multiply n single digits. Then the time needed by the algorithm is described by a recurrence $T(n) = 2T(n/2) + O(...)$. What is this last term? Plug in the correct term and explain. (5 points)

4.2. What is the result $T(n)$ of this recurrence? And why? (2 points)

4.3. We claim that the time bound for the general case (factors with k digits) is at most k^2 times the previous result. Briefly argue why this claim is true. (3 points)

(Now you can also see whether the divide-and-conquer algorithm is faster, slower, or neither of these.)

Problem 5 (10 points)

A *feedback vertex set* in a directed graph $H = (V, E)$ is a subset $U \subset V$ of nodes such that the removal of U (and of all incident edges) destroys all directed cycles in the graph.

The Feedback Vertex Set problem is defined as follows: Given a directed graph and an integer k, does the graph contain a feedback vertex set with at most k nodes?

Now we present a reduction from Vertex Cover to Feedback Vertex Set:
We are given an undirected graph G. We arrange the nodes of G arbitrarily in a linear order, say, the nodes of G are $v_1, \ldots, v_n$ from left to right. We replace every undirected edge (v_i, v_j), $i < j$, in G with a directed edge that goes from left to right, that is, from v_i to v_j. (Note that the graph is now a DAG.) For every directed edge $e = (v_i, v_j)$ we create a new node w_e and two directed edges: from v_j to w_e, and from w_e to v_i. (Informally: On every given edge we "put a triangle" which is also a directed cycle.) Let H denote the directed graph constructed in this way.

5.1. Show the following equivalence: G has a vertex cover with k nodes if and only if H has a feedback vertex set with k nodes. (6 points)

5.2. Does this reduction work in polynomial time? Why, or why not?
(2 points)

5.3. Is it correct to conclude from this reduction that Feedback Vertex Set is NP-complete? Why, or why not? (2 points)

(Note that you can answer 5.2 and 5.3 even if you did not manage 5.1.)

Problem 6 (10 points)

Imagine that we have some system being capable of n different states, in every time unit we can get from the current state to some possible successor states only, and we want to be exactly in some particular state exactly at some desired time. The states and the possible transitions can be modeled as the nodes and edges of a directed graph. This leads to the following graph problem. (You may even skip this first paragraph and start reading **here**.)

Given a directed graph with n nodes, two nodes u and v, and an integer k, we want to figure out whether we can walk from u to v in exactly k steps.

Accordingly, we define a function $f(u, v, k)$ with Boolean value, which indicates whether this is possible or not.

For clarity: By a walk with k steps we mean any sequence of nodes $u = v_0, v_1, v_2, \ldots, v_k = v$ such that every pair (v_i, v_{i+1}) is a directed edge. Nodes and edges are allowed to appear several times in a walk.

6.1. Suppose that you have already computed $f(u, v, k)$ for some fixed integer k and for all pairs of nodes u and v. How can you compute $f(u, v, k+1)$ for all pairs of nodes, and how much time does this take? Clearly, you cannot beat $O(n^2)$ time, because that many values must be computed. But is $O(n^2)$ time enough, or do you need more? Your time bound does not have to be optimal, but what you claim should be correct. (5 points)

The procedure in 6.1 can be used as the basis of a dynamic programming algorithm. But we can also do divide-and-conquer instead. We do not discuss the complete algorithms here, but only the essential step:

6.2. Suppose that you have already computed $f(u, v, k)$ and $f(u, v, m)$ for some fixed integers k and m, and for all pairs of nodes u and v. How can you compute $f(u, v, k + m)$ for all pairs of nodes, and how much time does this take? (5 points)

Solutions (attached after the exam)

1. The product of any two integers with a and b digits, respectively, has $a + b + O(1)$ digits. For the given simple algorithm it follows that, after i iterations of the for-loop, y has $O(ik)$ digits. Hence the i-th iteration costs $O(ik \cdot k) = O(ik^2)$ time. Summation over all $i \leq n$ yields $O(n^2k^2)$ time. (5 points)

2.1. The proposed algorithm is to follow these rules: Always go to the right, never back to the left. First turn on the light at s. Then repeat: In the currently illuminated part to the right of your position, go to the switch s_i with the rightmost f_i and turn on the light at this s_i. But whenever you pass any f_j that is switched on, turn off the light at f_j. (5 points)

2.2. One can, e.g., argue like this: Initially the worker stands at the left end of the leftmost segment, which is now illuminated. At every moment, the worker stands at some illuminated point p, and some interval to the right, say $[p, q]$, is illuminated, too. The worker should next press the button $s_i \in [p, q]$ with the rigthmost f_i, since this extends the illuminated part to the right as much as possible. Some button in $[p, q]$ must be pressed, and every other choice could be replaced by this one, without making the solution worse. (This is the exchange argument!) It is also useless to go to the left and to press a better switch s_j, since the worker comes from the left and could have pressed s_j already when (s)he passed that point. Finally, since the worker never goes back, it is safe to turn off the light at every f_i for which $[s_i, f_i]$ was illuminated. In particular, all lights are switched off in the end. (6 points)

2.3. Maintain the coordinate of the current position p and of the rightmost illuminated point q. Furthermore, store in a priority queue all $s_i \in [p, q]$ and their corresponding values f_i. The algorithm requests the $s_i \in [p, q]$ with maximum f_i, at most n times, and the data structure can answer every such request in $O(\log n)$ time. As p and q can only grow, every interval enters and leaves the priority queue only once. These are n insert and delete operations. In total this yields $O(n \log n)$ time. – A worse time bound with less data structure efforts is also acceptable, but polynomial time in n is mandatory. (4 points)

3. The only change in the algorithm is to re-define δ_{ij}. We set $\delta_{ij} = 1$ if a_i and b_j are regular symbols with $a_i \neq b_j$, and $\delta_{ij} = 0$ if $a_i = b_j$ or if some of them is a wildcard. Every insertion and deletion step still yields one mismatch. Altogether, the original formula for dynamic programming remains therefore correct with these new δ_{ij}. The time analysis does not change at all, hence the time bound is not affected either. (10 points)

4.1. The product of n single digits has $O(n)$ digits. Hence the last step after the recursion is to multiply two integers of length $O(n)$. Using the standard multiplication algorithm this costs $O(n^2)$ time. Thus the last term should be $O(n^2)$. (5 points)

4.2. Here we can simply apply the master theorem, which yields
$T(n) = O(n^2)$ since $2^2 > 2$. (2 points)

4.3. The product of n integers with $O(k)$ digits has $O(nk)$ digits, hence the last term is now $O(n^2k^2)$. This causes an extra factor k^2 also in the result of the recurrence. Altogether, the total time is $O(n^2k^2)$. Conclusion: The new algorithm is neither faster nor slower. (3 points)

5.1. Suppose that G has a vertex cover with k nodes. Reomoval of the same nodes in H destroys all triangles that we created (since a vertex cover is incident to every edge in G). Hence this also destroys all directed cycles in H. Thus H has a feedback vertex set with k nodes.
Conversely, suppose that H has a feedback vertex set F with k nodes. F may contain some new nodes w_e. But we can replace every such node in F with one of the nodes of e. Then it still destroys the triangle of e, and maybe more triangles. After these replacements we get a set of at most k original nodes, and obviosuly they form a vertex cover of G. (6 points)

5.2. Yes, it even runs in linear time, since every node and every edge of G is processed only once. (2 points)

5.3. Yes. Feedback Vertex Set is trivially in NP. Vertex Cover is known to be NP-complete, and we reduced from it. Correctness holds according to 5.1, and the reduction runs in polynomial time according to 5.2. (2 points)

6.1. $f(u, w, k + 1)$ is true if and only if there exists some node w such that $f(u, w, k)$ is true and the directed edge (w, v) exists. One possibility is to check all n nodes w. Since this must be done for all $O(n^2)$ pairs of u and v, we get a time bound $O(n^3)$. (5 points)

6.2. Similarly, $f(u, w, k + 1)$ is true if and only if there exists some node w such that $f(u, w, k)$ and $f(w, v, m)$ are true. By the same argument, the calculations can be done in $O(n^3)$ time. (5 points)