Algorithms Re-Exam TIN093/DIT602*

Course: Algorithms

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

Date, time: 8th January 2019, 8:30–12:30

Building: M

Responsible teacher: Peter Damaschke, Tel. 5405

Examiner: Peter Damaschke

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

Time for questions: around 9:30 and around 11:00.

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.

*version with corrected typo

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 (6 points)

Imagine the following situation: We have solved some instance of Interval Scheduling, using the Earliest End First algorithm, and we got an optimal solution X. After our work, yet another interval is added to the instance. It ends later than all the other n intervals.

Since we got an additional interval, the size of an optimal solution may increase. In order to compute the new optimal solution, of course, we may run Earliest End First again from scratch on the $n+1$ intervals. This would cost again $O(n)$ time (provided that the intervals are already sorted by their right ends). But we can do much better:

Show that the optimal solution can be updated in $O(1)$ time. That is, give a rule to construct the new optimal solution from X, explain why it takes only constant time, and, most importantly, argue why your rule is correct.

Problem 2 (15 points)

Suppose that we have to cover a rectangular stripe of size $2 \times n$ with m tiles of given sizes $1 \times k_i$ $(i = 1, \ldots, m)$. All given numbers are positive integers. Of course, it is assumed that $\sum_{i=1}^{m} k_i = 2n$.

Note that any tile of size 1×2 (if such tiles should exist) can be placed horizontally or vertically, and this freedom of choice adds a slight difficulty to the problem.

Give an algorithm that places the tiles such that they exactly cover the stripe, or reports that such a tiling is not possible for the given instance. Show that your algorithm is correct, i.e., that it never reports failure if a solution exists. Moreover, analyze the time as a function of n. It must be polynomial in n.

Hint: First consider the special case without the troublemakers (tiles of size 1×2), in order to get the right basic idea. Then think about a way to cope correctly with tiles of size 1×2 as well.

Problem 3 (6 points)

A colleague sends you the following message:

"(1) Note that $(10^{n/2}x + y)^2 = 10^n x^2 + 10^{n/2} \cdot 2xy + y^2$.
(2) Hence I can compute the square of an integer with n digits by computing the squares of only 2 integers with $n/2$ digits.
(3) Similary to the divide-and-conquer algorithm for general multiplication, I get the recurrence $T(n) = 2T(n/2) + O(n)$, since now the number of recursive calls is only 2.
(4) The solution is $T(n) = O(n \log n)$.
(5) Finally, squaring cannot be done faster than multiplication.
(6) Hence my result also yields an $O(n \log n)$-time algorithm for the multiplication of two integers."

Write a reply. Discuss sentence by sentence: Which steps in this reasoning are correct, and which steps are wrong (and why)?

Problem 4 (6 points)

We know that both Subset Sum and Knapsack are NP-complete, but this was not proved in the course. Do the following simple part of the job: Give a polynomial-time reduction from Subset Sum to Knapsack. (Remark: It doesn't matter which version of the Subset Sum problem you choose.)

Problem 5 (9 points)

Researchers conjecture that certain problems cannot be solved faster than by trivial exhaustive search, that is: Every algorithm for them needs $O(2^n)$ time, and this upper time bound cannot be improved. Suppose that X is such a problem, and that we can reduce X to some other problem Y of interest, in $O(n)$ time.

5.1. Does this imply that Y cannot be solved faster than in exponential time either? More precisely, the claim is: "There exists some constant $b > 1$, such that no algorithm can solve Y in $O(b^n)$ time, or even faster." True or false? (6 points)

5.2. Can we also conclude that Y cannot be solved faster than in $O(2^n)$ time? (3 points)

Remarks and hints: Note carefully that claim 5.2 is stronger than claim 5.1, since the base b is specified to be 2 there. Also keep in mind that O-notation ignores constant factors. Motivate your positive or negative answers.

Problem 6 (6 points)

Let $G = (V, E)$ be an undirected and connected graph, consisting of n nodes and m edges.

6.1. We assume that all edges have weight 1. Show that a minimum spanning tree can be computed in $O(m)$ time. (In graphs with general edge weights we needed more than linear time.) It is enough to use a known algorithm and apply it properly to this problem. You need not invent a new algorithm. But motivate the correctness of your approach. (3 points)

6.2. An articulation point was defined as a node whose removal disconnects the graph G. We have seen that all articulation points can be found in $O(m)$ time. Now we bring up a much easier problem: Find just *one* node which is *not* an articulation point, in $O(m)$ time. – But be careful: This is not a trivial consequence of the known result, because it does not say that such a node *exists*. (3 points)

Problem 7 (12 points)

Suppose that the nodes of a directed graph model companies. The directed edges model ownership, in the following way. Every directed edge has some weight w with $0 \leq w \leq 1$. A directed edge (u, v) with weight w means that company u owns a fraction w (that is, $100 \cdot w$ %) of company v. Of course, for every node v, the sum of weights of all edges that end in v must not exceed 1. Suppose that the graph has no directed cycles. (Circular ownership might be strange.)

We say that a company u *controls* a company v if:

- u owns more than half of v,
- or u controls a set of other companies that together own more than half of v.

(Note that this definition is inductive – this is not a mistake!)

The problem is: Given a graph with edge weights as introduced above, and a company u, figure out the set of all companies that u controls.

Give an algorithm and a time bound, as a function of the number of nodes and edges in the graph. The time bound should be as good as possible. Also, argue why your algorithm is correct, that is, why it returns exactly the set of controlled companies.

Hint: A topological order should be very helpful.

Solutions (attached after the exam)

1. The update "algorithm" is: Let $[s, f]$ denote the added interval. If $[s, f]$ intersects the last interval of X, then do not change the solution. If $[s, f]$ is disjoint to the last interval of X, then add it to X.
Exacution costs $O(1)$ time, since only one comparison is needed to distinguish the two cases. Correctness follows immediately from these facts: Earliest End First (EEF) is correct on every instance, EFF processes the intervals by increasing right ends, and our update rule is equivalent to doinng one more step of EEF. (6 points)

2. The problem is a variant of Subset Sum. We may solve it from scratch by dynamic programming. (This is, of course, perfectly acceptable.) Instead we present here a reduction working with an exchange argument.
If no tiles of size 1×2 exist, we can simply rephrase the problem as finding a subset of the numbers k_i with a sum exactly n. We can use the known algorithm for it, which runs in $O(mn)$ time. Since $m = O(n)$, this is $O(n^2)$.
Now consider the general case. The stripe consists of two rows of length n. We call every tile of size 1×2 that intersects both rows a barrier. Every barrier divides the solution in two "independent" parts, on both sides of the barrier. Hence we can move the barrier to the left end of the stripe, and obtain another valid solution. In this way we can collect all barriers at one end of the stripe. Now we can rotate any two neighbored barriers and obtain two tiles of size 1×2, each of which is in only one row. There remains at most one barrier (if their number is odd).
This reasoning shows: If there exists a solution at all, then there exists a solution with at most one barrier. Thus the following algorithm is correct: Run the Subset Sum algorithm twice, namely, once on the given instance, and once on the instance with $n := n - 1$, where one number $k_i = 2$ is deleted. The time bound is still $O(n^2)$. (15 points)

3. "(1) is obviously correct. (2) is formally not yet wrong, but you may miss the multiplication xy. (3) is definitely wrong: The multiplication xy cannot be done in $O(n)$ time, unless you know already such a multiplication algorithm, but then your whole result becomes pointless. (4) would be the correct conclusion from (3) by the master's theorem, but your recurrence in (3) does not hold. Your claim in (5) is right; there is a linear-time reduction from multiplication to squaring. (6) would be a correct conclusion, but

actually you have not proved $O(n \log n)$ for squaring, due to the mistake in (3)." (6 points)

4. Let $w_1, \ldots, w_n, W$ be an instance of Subset Sum, aking for a subset that sums up to exactly W. We construct an instance of Knapsack that takes these numbers as weights and capacity, and where $v_i := w_i$ for all i. Trivially this is done in polynomial time. Equivalence is also obvious by construction: The Subset Sum instance has a solution if and only if the Knapsack instance has a solution with total value W. (6 points)

5.1. True. Assume that some algorithm solves Y in $O(b^n)$ time, where b is any constant. It was also assumed that X is reducible to Y in $O(n)$ time. Let c be the "hidden" constant factor in this $O(n)$ time bound. Thus we can solve any instance of X in $O(b^{cn})$ time. Now we choose $b > 1$ small enough such that $b^c < 2$. This would yield an algorithm for X running in $O(2^n)$ time, a contradiction. Hence we have found some $b > 1$ with the claimed property. (6 points)

5.2. This cannot be concluded, because of the unknown constant factor c in the exponent. Complexities $O(b^n)$ are not equal for different bases b.
(3 points)

6.1. Simply run BFS or DFS from an arbitrary start node, and output the BFS tree or DFS tree. This is a spanning tree. Morever, all spanning trees in a graph of n nodes have exactly $n-1$ edges, hence either one is an MST.
(3 points)

6.2. Compute a spanning tree as in 1. Any leaf of this tree is not an articulation point: If we remove the leaf, the rest of the tree remains connected. Hence the graph (which may even contain further edges) remains connected as well. (3 points)

7. By assumption, the graph is acyclic. We construct a topological ordering. Beginning with u we build the set S that contains u and all nodes controlled by u, as follows. We set $S := \{u\}$. Then we traverse the topological order. For every node v we consider the directed edges (s, v) with $s \in S$. We compute the sum of their weights. If this sum is larger than $1/2$ then we add v to S, otherwise we do not. With n nodes and m edges, the algorithm runs in $O(n + m)$ time, since every node and edge is processed only once, during the topological sorting and during the computation of S.

Correctness is seen as follows. We claim by induction that every node v after u is put in S if and only if v is controlled. In fact, if the sum of weights of edges from S to v is larger than $1/2$, then v is controlled, by the induction hypothesis and the definition of "controlled". If the sum is at most $1/2$, then we can be sure that v is not controlled: Nodes that are added later to S cannot have directed edges ending in v, since backward edges do not exist in a topological ordering. That is, the sum cannot further increase. (12 points)