Algorithms Re-Exam TIN093/DIT602

Course: Algorithms

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

Date, time: 27th August[1] 2020, 14:00–18:00

Building: online

Responsible teacher: Peter Damaschke, Tel. 5405, email `ptr@chalmers.se`

Examiner: Peter Damaschke

Exam aids: all (home exam)

Time for questions: at any time during the exam

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.

[1] Month corrected, there was a mistake in the original document.

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 page.
- Submit your solutions as **one PDF**, preferably produced with some text editor. If you scan handwritten pages, they must be legible. Unreadable solutions will not get points. Submit in Canvas and, for additional safety, also mail a PDF attachment to `ptr@chalmers.se`.
- 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 (9 points)

This warm-up exercise is more an extremal value problem rather than a genuinely algorithmic problem, but it revolves around the understanding of O-notation and the complexity of arithmetic calculations.

Background:
Suppose that, for some scientific application, we have to multiply real numbers with a rather high precision. The factors are given with many decimal digits, but for the sake of speed we want to use only some limited number of digits in the multiplications.

Some definitions:
If we have an approximate value v' for some true value v, we define the *relative error* of v' by $|v - v'|/v$. Let $v[k]$ be the number obtained from v by truncation after k digits (starting from the first non-zero digit). That is, $v[k]$ consists of the first k digits of v.
Example: If $v = 3.14159265358979323\ldots$ and $k = 6$ then $v[k] = 3.14159$.

1.1. We claim that the relative error of $v[k]$ is $O(10^{-k})$. Briefly explain why this is true. (3 points)

1.2. Suppose that, instead of two exact factors x and y, only approximate values x' and y' are available, where x' has relative error r, and y' has relative error s. Instead of $x \cdot y$ we can only compute $x' \cdot y'$. Show that $x' \cdot y'$ has relative error $O(r + s)$. (2 points)

Now suppose that x and y are two given factors, and we are interested in their product $x \cdot y$. From 1.1 1and 1.2 it follows that $x[k] \cdot y[m]$ has relative error $O(10^{-k} + 10^{-m})$. Two numbers with k and m digits can be multiplied in $O(km)$ time. This raises a natural question: How many digits of x and y should we use in order to compute $x \cdot y$ as precisely as possible, within a given time budget t? This leads to the final question:

1.3. Given a number t, how would you choose numbers k and m such that $km = t$, and the expression $E := 10^{-k} + 10^{-m}$ is minimized? Motivate your answer – an informal but clear argument is enough.
Hint: Perhaps the easiest way is to notice that $E = 1/10^{10^{\log k}} + 1/10^{10^{\log m}}$ and $\log k + \log m = \log t$ is fixed. (4 points)

Problem 2 (8 points)

An $(m+1) \times (n+1)$ grid consists of all points with integer coordinates (i, j), where $0 \leq i \leq m$ and $0 \leq j \leq n$. Suppose that such a grid is given, with two kinds of points: obstacles and free points. Assume that $(0, 0)$ and (m, n) are free points. For some reason we want to find a path from $(0, 0)$ to (m, n) with the following properties:

- The path uses only free points, it avoids obstacles.
- The path is monotone, that is: From a point (i, j) one can go to $(i+1, j)$ or $(i, j+1)$, but not to $(i-1, j)$ or $(i, j+1)$.
- The path has a minimum number of *bends*, i.e., changes of directions, among all possible monotone paths from $(0, 0)$ to (m, n).

All monotone paths from $(0, 0)$ to (m, n) have the same length $m + n$. But here we are interested in the "simplest" path, not in the shortest path.
Example: The path $(0, 0)$, $(0, 1)$, $(0, 2)$, $(1, 2)$, $(2, 2)$, $(3, 2)$, $(4, 2)$, $(4, 3)$, $(5, 3)$, $(6, 3)$ has exactly 3 bends.

2.1. Give an efficient algorithm that computes a monotone path from $(0, 0)$ to (m, n) that uses only free points and has a minimum number of bends. (The destination might be unreachable due to the obstacles, and in this case the algorithm must report that no such path exists at all.)
Hints: Use dynamic programming. Your "OPT function" may need a third parameter besides i and j, or you may define a function on the edges (rather than the points) of the grid. Define your function and state how it is computed. You need not describe standard parts like initial values and backtracing. (6 points)

2.2. Give a time bound for your algorithm, with motivation. (2 points)

Problem 3 (12 points)

Let n be a given positive integer. As you should know, an unknown positive integer $k \leq n$ can be found by binary search, using $\log_2 n$ comparisons of the unknown number k with carefully chosen numbers i. (Comparisons have the form "if $k < i$ then ... else ..."). When we have reasons to expect the unknown number k to be much smaller than n, it is nicer to have a good time complexity as a function of k rather than n. In fact, it is possible to determine k by at most $2\log_2 k + O(1)$ comparisons as follows:
(i) First compare k with 2^j, for $j = 0, 1, 2, 3 \ldots$, until the smallest exponent j with $k < 2^j$ is found.
(ii) Do usual binary search in the interval of integers i between 2^{j-1} and 2^j.

3.1. Show that this algorithm needs, in fact, no more than $2\log_2 k + O(1)$ comparisons. – Note carefully that O-notation is not used for the entire expresssion, because here we also care about the constant factor. And you are advised not to apply recurrences here; this would make things unnecessarily complicated. (5 points)

Next, amazingly, the constant factor 2 can be improved to 1 as follows:
(i) First compare k with 2^{2^j}, for $j = 0, 1, 2, 3 \ldots$, until the smallest exponent j with $k < 2^{2^j}$ is found.
(ii) Then do binary search in the interval of integers between 2^{j-1} and 2^j, in order to find the exponent i with $2^{i-1} \leq k < 2^i$. (For clarity: Only numbers of the form 2^i are compared with k in this phase, where $2^{2^{j-1}} \leq 2^i \leq 2^{2^j}$.)
(iii) Finally do usual binary search in the interval of integers between 2^{i-1} and 2^i.

3.2. Similarly as above, show that this algorithm needs no more than $\log_2 k + O(\log \log k)$ comparisons. (7 points)

Problem 4 (16 points)

Let $G = (V, E)$ be an undirected graph and ℓ a function that assigns to every edge $e \in E$ a positive length $\ell(e)$. An *embedding* of G in a geometric space is a function μ that maps the nodes of V to points, such that for every edge $e = (u, v)$ the distance between the points $\mu(u)$ and $\mu(v)$ equals the given length $\ell(e)$. Note that it is allowed to map several nodes to the same point: If nodes u and v are not joined by an edge, then $\mu(u) = \mu(v)$ is permitted. Now we consider the special case where G is merely a cycle and the geometric space is merely the real line, with the usual distance.

Example: The cycle with nodes u, v, w, x, y, z and edge lengths given by $\ell(u, v) = 6$, $\ell(v, w) = 3$, $\ell(w, x) = 2$, $\ell(x, y) = 6$, $\ell(y, z) = 5$, $\ell(z, u) = 2$ has an embedding in the line, for instance: $\mu(u) = 0$, $\mu(v) = 6$, $\mu(w) = 3$, $\mu(x) = 1$, $\mu(y) = 7$, $\mu(z) = 2$. (You can check that the six distances are exactly as required.)

Accordingly we define the decision problem Embed-Cycle: Given a cycle with positive edge lengths, can we embed it in the real line?
Furthermore, it is known that Half-Half Subset Sum is NP-complete. This problem asks: Given a set of n positive integers w_i $(i = 1, \ldots, n)$, is there some subset S such that $\sum_{i \in S} w_i = \sum_{i=1}^{n} w_i/2$?
Here is a reduction from Half-Half Subset Sum to Embed-Cycle: Given n positive integers w_i $(i = 1, \ldots, n)$, we construct a cycle with n nodes and n edges having these lengths w_i, in arbitrary order.

4.1. Show: If the given instance of Half-Half Subset Sum has a solution, then the constructed instance of Embed-Cycle has a solution. (6 points)

4.2. Show the converse: If the constructed instance of Embed-Cycle has a solution, then the given instance of Half-Half Subset Sum has a solution. (6 points)

4.3. Does 4.1 and 4.2 imply that Embed-Cycle is also NP-complete? Motivate your answer. Note that you can also answer if you did not manage 4.1 and 4.2. (4 points)

Problem 5 (15 points)

One main problem in decision making is to choose a sequence of actions that maximizes the total reward using limited resources. Abstraction leads to the following graph problem:
We are given a DAG $G = (V, E)$ with n nodes and m edges, a start node $s \in V$, two functions r and c that assign to every edge $e \in E$ a positive reward $r(e)$ and a positive cost $c(e)$, and a budget b. All these numbers are integers. Find a directed path P with these properties: P starts in s, the total reward on the traversed edges $\sum_{e \in P} r(e)$ is maximized, but their total cost does not exceed the budget: $\sum_{e \in P} c(e) \leq b$. The end node of P is arbitrary.

5.1. Give an algorithm that solves the above problem using dynamic programming. Make sure that you describe all ingredients, i.e., a clear definition of your optimization function, and its computation. You can skip the backtracing part. (10 points)

5.2. Give a time bound for your algorithm, with a careful motivation. The time bound should be polynomial in n (or m) and b. (5 points)

Solutions (attached after the exam)

1. One possible way to argue: For the relative error it doesn't matter where the decimal point is. For any fixed k, in the worst case, v has the form $1.0\ldots0999\ldots$ (with $k-1$ zeros), because this minimizes v and maximizes the truncated amount. By straightforward calculation, the relative error is roughly 10^{-k} in this case. (3 points)

1.2. Assume $x' < x$ and $y' < y$; the other cases are similar. Then we have: $x' = (1-r)x$, $y' = (1-s)y$, hence $x'y' = (1-r)x(1-s)y = (1-r-s+rs)xy$, and the assertion follows. (2 points)

1.3. Choose $k = m = \sqrt{t}$ (rounded to integers). Optimality is seen as follows. The sum $\log_{10} k + \log_{10} m$ is fixed. When we increase the smaller (and hence decrease the larger) of $\log_{10} k$ and $\log_{10} m$, then E decreases. Thus, E is minimized when $k = m$. (4 points)

2.1. We define $OPT(i, j, H)$ to be the minimum number of bends needed for a path from $(0, 0)$ to (i, j), where the last step is horizontal. We define $OPT(i, j, V)$ similarly, except that the last step is vertical. Furthermore, these values are infinite if no such path exists. Since a bend is added if and only if the direction changes, we have the calculation formula
$OPT(i, j, H) = \min\{OPT(i-1, j, H), OPT(i, j-1, V)+1\}$ for all free points (i, j), and a similar formula holds for $OPT(i, j, V)$. (6 points)

2.2. The time is $O(mn)$, simply because this is the number of values to compute, and each of them needs constant time. (2 points)

3.1. Since $j = \log_2 k + O(1)$ holds for the final j, phase (i) needs this number of comparisons. Phase (ii) is binary search in a range of at most $2k$ integers, hence it needs $\log_2 k + O(1)$ comparisons, too. Together these are $2\log_2 k + O(1)$ comparisons. (5 points)

3.2. Perhaps the most concise argument is: In phases (i) and (ii) we search for the exponent of the power of 2 that is closest to k. This is the same algorithm as in 3.1, but applied to the exponents, i.e., to logarithms. Due to the result of 3.1, this needs $\log_2 \log_2 k + O(1) = O(\log \log k)$ comparisons. Phase (iii) is again ordinary binary search using $\log_2 k + O(1)$ comparisons. (7 points)

4.1. Let S be a solution to the Subset Sum instance. We place an arbtrary node at some arbitrary start point, then we traverse the cycle and place the nodes, obeying the following rule. Let u be the current node and let v be the next node on the cycle. If $\ell(u, v)$ is in S, then we set $\mu(v) := \mu(u) + \ell(u, v)$, else we set $\mu(v) := \mu(u) - \ell(u, v)$. (Go to the right/left if the edge length is inside/outside S. Note that some edge lengths can be equal, but there is no risk of confusion, as the edges are uniquely identified by their indices.) Since $\sum_{i \in S} w_i = \sum_{i \notin S} w_i$, we return exactly to the start point when the traversal is completed, hence we have correctly embedded the cycle. (6 points)

4.2. If the constructed instance of Embed-Cycle has a solution, then we traverse the cycle, and whenever we go to the right on the real line, we put the edge length in S. Since the cycle returns to its start point, the sums of lengths of all edges going to the left and to the right are equal, thus we obtain a correct solution S. (6 points)

4.3. Yes. The reduction runs in linear time, and due to the equivalence of the instances, 4.1 and 4.2 establish a polynomial-time reduction of Half-Half Subset Sum to Embed-Cycle. Furthermore, Embed-Cycle is in NP, because we can verify a given affirmative solution in polynomial time. (4 points)

5.1. For every node $v \in V$ and every non-negative integer $k \leq b$ we define $OPT(v, k)$ to be the maximum reward of a directed path from s to v with total cost at most k. Then we have

$$OPT(v, k) = \max\{OPT(u, k - c(u, v)) + r(u, v)\},$$

where the maximum is taken over all nodes u for which a directed edge (u, v) exists. (The treatment of special cases like $k < c(u, v)$ is omitted here.) This holds true because every directed path to v must use some predecessor u, and for every u and every possible budget it suffices to use some most rewarding path that ends in u. The initial values are $OPT(s, k) = 0$ for all values k. (10 points)

5.2. Some valid time bound is $O(mb)$, for these reasons: Every edge is processed only once for every non-negative $k \leq b$, and every calculation of a value $OPT(v, k)$ takes $O(1)$ time. The vertices v may be considered in a topological order that can be computed in advance in $O(m)$ time. (5 points)