Algorithms Re-Exam TIN093/DIT093/DIT602

Course: Algorithms

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

Date, time: 29th August 2024, 14:00–18:00

Place: Johanneberg

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

Examiner: Peter Damaschke

Exam aids: dictionary,
printouts of the Lecture Notes (which may contain own annotations),
one additional handwritten A4 paper (both sides).

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

Solutions: will be published after the exam.

Results: will appear in ladok.

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

Inspection of grading (exam review): to be announced.

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 does not only cost time.
  It may also obscure the actual solutions.
- But motivate all claims and answers.
- Strictly avoid code for describing a complex algorithm.
  Instead *explain* in your words how the algorithm works.
- If you cannot manage a problem completely, still provide your approach or partial solution to earn some points.
- Facts from the course material can be assumed to be known.
  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 simply look for an algorithm that counts objects and then outputs their number in binary representation. Let n denote the unknown number of objects that we want to determine. Let x be some variable for an integer, written in binary representation. Our elementary operations are: any operation with a binary digit in x, and moving any single object.

One idea is to start with $x = 0$, move the objects one by one to another place, and always add 1 to x. Clearly, the final value of x is then n. This method may need $O(n \log n)$ digit operations, because even the addition of 1 can change $\log_2 n$ digits of x in the worst case. (For instance, note that $1111 + 1 = 10000$.)

Show that, actually, $O(n)$ time is enough. (5 points)

Hint: One way is a more careful analysis of this counting algorithm. But it is much simpler (and recommended here!) to change the algorithm: Group the objects in pairs. In the end, one object is left over or not. Keep only one object from every pair, and repeat the procedure recursively. Explain how this yields the binary digits of n, and why only $O(n)$ elementary operations are carried out.

Problem 2 (18 points)

In this problem, we continue moving certain objects, but with a different goal:

In a storage hall, k identical boxes standing at one place have to be carried to k mutually distinct destination points. The boxes can be moved on certain routes, but only if these routes are not blocked by other boxes.

The scenario can be described abstractly as a graph problem: In an undirected graph with $n > k$ nodes, k tokens are initially located at some node s, while all other nodes are initially free of tokens. With the exception of s, every node can host at most one token at any time. We can successively select and move tokens, one at a time. Any token can be moved along an edge to some selected neighbored node that is currently free. We assume that every such step must be completed; it is not possible to interrupt it and wait in the middle of the edge. We want to move the k tokens to k given destination nodes, exactly one to each of these nodes, using a minimum total number of steps.

2.1. Describe an algorithm that outputs such an optimal schedule for moving all tokens to their destinations. (5 points)

2.2. Prove that your algorithm computes, in fact, a schedule that never lets any tokens collide on a node (other than s), and that uses the minimum total number of steps. (5 points)

2.3. State and motivate a time bound for your algorithm. – Note that we ask here for the time complexity of the algorithm computing the schedule. Do not confuse this with the number of steps used by the schedule itself. (5 points)

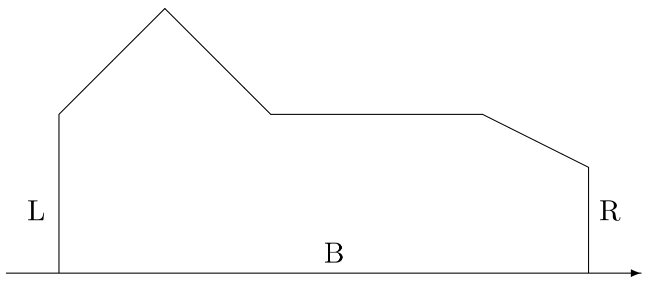
2.4. Consider the same problem on a directed graph, that is, tokens can be moved on every edge only in the given direction. Briefly discuss: Can you still apply your algorithm, or some modified version of it? What is different? (3 points)

Problem 3 (12 points)

In an instance of Weighted Interval Scheduling, let $I_1, \ldots, I_n$ denote the intervals sorted by their right endpoints: $f_1 \leq \ldots \leq f_n$. We had defined $OPT(j)$ as the maximum weight of a solution to the instance $\{I_1, \ldots, I_j\}$. It was not required that this solution contains I_j. Alternatively, we may choose to define $OPT(j)$ as the maximum weight of a solution to the instance $\{I_1, \ldots, I_j\}$ which does contain I_j. This would considerably change the details of the dynamic programming algorithm:

3.1. Give and explain a formula for the computation of this re-defined function $OPT(j)$. (7 points)

3.2. How can we construct the final solution (not only its value!) from these new $OPT(j)$ values? Give specific details of the construction, such that a person who wants to implement the algorithm would know what to do. But you need not go into data structure details and time complexity. (5 points)



Problem 4 (10 points)

Consider the skyline problem generalized to the case when buildings can have pitched roofs. Formally, in an x, y-coordinate system we are given a set of geometric objects of the following type: Every object is a polygon with one side B on x-axis (the bottom line), two vertical sides L and R incident to B that go upwards, and a chain of straight line segments (the "roof") connecting the upper ends of L and R. In other words, the roof can be seen as a piecewise linear function of x. The polygons are given by the coordinates of all their corners. Let n denote the total number of corners of all given polygons.

Propose an $O(n \log n)$-time algorithm that computes the union of the interiors of all the given polygons. Provide enough motivation of the correctness and time bound. (10 points)

Problem 5 (15 points)

A perfect matching in an undirected graph $G = (V, E)$ with an even number n of nodes is a set $M \subseteq E$ of exactly $n/2$ edges that contain every node of V exactly once. The Perfect Matching (PM) problem asks whether a given graph has some perfect matching. PM can be solved in polynomial time. Such an algorithm was not in the course. But imagine that a polynomial-time algorithm for PM is at your disposal (e.g., from a library), and you can trust its correctness. We remark that PM is a famous problem with many applications, like job assignment.

Now we get a more general problem called Perfect Matching with Loops (PML). We are given an undirected graph $G = (V, E)$ where some edges can be loops: A loop is an edge that is attached to only one node, rather than joining two nodes. Still, we want to decide whether G has a perfect matching, that is, a set of normal edges and loops that contain every node exactly once.

Luckily, PML can be reduced to PM as follows. Let $G = (V, E)$ be a graph with n nodes, m normal edges, and ℓ loops. For every node v with a loop, we replace this loop with an edge vv' to some fresh node v'. (For every such v we create an own node v'.) If $n - \ell$ is odd, then we create yet another fresh node. All fresh nodes are connected to a clique, by adding all possible edges between them. Let H be the graph obtained in this way.

5.1. Show that G has a perfect matching if and only if H has a perfect matching. (10 points)

5.2. Does this result imply that PML is solvable in polynomial time? Briefly explain your (positive or negative) answer. (5 points)

Solutions (attached after the exam)

1. The unknown n has the form $n = 2k + 1$ if one object is left over, and $n = 2k$ if not. This yields the last binary digit of n. Then, the algorithm continues with $\lfloor n/2 \rfloor$ and produces, in the same way, the other binary digits from right to left. The number of times we access any objects is at most $n + n/2 + n/4 + n/8 + \ldots = O(n)$. (5 points)

2.1. Run BFS with start node s. This also yields the distances $d(s, v)$ to all nodes v. Sort the k destination nodes v by decreasing $d(s, v)$. Always take the next node v in this ordering, and send a token from s to v on some shortest path. (5 points)

2.2. While a token is travelling from s to some node v on a shortest path, all nodes on this path are free, because all previous tokens are now placed at nodes u with $d(s, u) \geq d(s, v)$, and v itself is still free as well. Furthermore, at least $d(s, v)$ steps are trivially needed to move a token from s to v, and the algorithm uses only this number of steps. Hence the sum over all v is optimal, too. – Remark: This is an optimality proof by a simple lower bound, i.e., no exchange argument was needed here. (5 points)

2.3. Let m denote the number of edges. During BFS we can construct a BFS tree which contains a shortest path from s to every node. The nodes are already sorted by their distances from s, since BFS detects them in this ordering. Hence, all this can be managed in $O(m)$ time. (5 points)

2.4. The whole algorithm and proof goes through litereally for directed graphs, when we interprret the $d(s, v)$ as lengths of shortest directed paths. (3 points)

3.1. We may indroduce a dummy interval I_0 with $w_0 = 0$, being to the left of all given intervals. Then we have $OPT(0) = 0$ and $OPT(j) = \max_{i | f_i < s_j} OPT(i) + w_j$. Namely, since I_j is in the solution, we always gain its value w_j, but the rest of the solution must have its last interval I_i ending before s_j. We must take an index i that yields the best $OPT(i)$. If no interval ends before s_j, the dummy interval yields the maximum value which is still 0 in this case. (7 points)

3.2. Backtracing works as follows: Find an index j with maximum $OPT(j)$. Add I_j to the solution. Next, find some index i with maximum $OPT(i)$, among all $i < j$ that satisfy $f_i < s_j$. Add I_i to the solution. Set $j := i$, and iterate the procedure. (5 points)

4. First off, in general we cannot divide the given set of polygons into two halves with the same number of corners. Instead, we first subdivide the polygons by vertical lines into smaller polygons each having only one straight line segment as its roof. We need at most n additional lines, and this step does not change the union. Then we apply the divide-and-conquer algorithm for the skyline problem for rectangles. The merging of two skylines can still be done in $O(n)$ time, since the graphs of the two piecewise linear functions can intersect in only $O(n)$ points, and each of them can be computed in $O(1)$ time. The rest works as in the algorithm for the skyline problem for rectangles. (10 points)

5.1. Define $p = 1$ if $n - \ell$ is odd, and $p = 0$ if $n - \ell$ is even. Thus, H contains $\ell + p$ fresh nodes. Suppose that G has a perfect matching M that contains x loops. We replace every loop in M, say at node v, with the corresonding edge vv'. The number of fresh nodes not yet covered by M is now $(\ell + p) - x = (n - x) - (n - \ell) + p$. Since $n - x$ is even, this number is always even. Since the fresh nodes form a clique, we can therefore add pairwise disjoint edges to M, so as to cover all nodes. Thus we have found a perfect matching in H. Conversely, let M be a perfect matching in H. We replace every edge vv' in M with the loop at v. These loops together with the normal edges from M in G form a perfect matching in G. (10 points)

5.2. Yes, it does. The reduction from PML to PM obviously works in polynomial time, and PM was polynomial. Hence, by a known theorem, PML is polynomial. As a remark, we can even find an existing perfect matching in polynomial time, and not only solve the decision problem.
(5 points)