Algorithms Re-Exam TIN093/DIT093

Course: Algorithms

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

Date, time: 27th August 2026, 14:00–18:00

Place: Johanneberg

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

Examiner: Peter Damaschke

Exam aids: dictionary, printout of the Summary of Course Contents (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): in Canvas room.

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 complex algorithms.
  Instead *explain* in your words how the algorithms work.
- 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 (10 points)

Some municipality is planning to build a bridge over a river. The relevant section of the river is straight and has width b everywhere. A set of n candidate points on both sides of the river indicates possible ends of the bridge. The cost of the bridge will be proportional to its length (apart from some constant extra amount). All candidate points have mutual distances of at least c. Moreover, all candidate points have distances of at most d to the river. The mentioned numbers b, c, d are positive constants. You work for the municipality and you are asked to determine the cheapest possible bridge.

Describe an algorithm that accomplishes this task in $O(n \log n)$ time. Prove its correctness and time complexity, carefully taking the given assumptions into account.

Problem 2 (20 points)

In the following, a "string" always means a binary string with k characters 1 and $n-k$ characters 0, where n and k are fixed. A swap is an exchange of two neighbored characters. Now, let A and B be two given strings. For some reason, we want to transform A into B by a minimum number of swaps.

An example with $n = 6$ and $k = 3$:
$A = 011010$ can be transformed into $B = 110001$ by 3 swaps as follows.
$011010 \to 101010 \to 110010 \to 110001$.

We will derive a greedy algorithm that computes such a shortest sequence of swaps for any given A and B. The task is already divided into small sub-exercises. (Note that, even if you do not manage some sub-exercises, you can still continue and use their claims.)
In a string C, we define $c_1 < c_2 < \ldots < c_k$ to be the positions of the characters 1 from left to right. Let $a_1 < a_2 < \ldots < a_k$ and $b_1 < b_2 < \ldots < b_k$ be these positions in A and B, respectively. In the example above, we have: $a_1 = 2$, $a_2 = 3$, $a_3 = 5$, $b_1 = 1$, $b_2 = 2$, $b_3 = 6$.

2.1. Prove that any transformation of A into B needs at least $\sum_{i=1}^{k} |a_i - b_i|$ swaps. (5 points)

2.2. Prove that, for every string $C \neq B$, there exists some swap which decreases $\sum_{i=1}^{k} |c_i - b_i|$ by 1. (5 points)

2.3. Based on 2.1 and 2.2, give an algorithm for the problem and prove its correctness. (5 points)

2.4. Show that the number of swaps is $O(n^2)$, and that this worst-case bound cannot be improved. (5 points)

Problem 3 (8 points)

Approximating a complicated curve by a sequence of straight-line segments (as in linear regression) is one application of the Segmentation problem. We may want to do the same also for closed curves. Abstraction leads to the following problem that we call Cyclic Segmentation:

A cyclic sequence is a sequence $(x_1, \ldots, x_n)$ of items on a cycle. That is, it does not matter where it begins, hence $(x_1, \ldots, x_n)$ is considered equal to $(x_{1+s}, \ldots, x_n, x_1, \ldots, x_s)$ for any integer $s$with $0 < s < n$. We are given a cyclic sequence and penalties for all segments $(x_i, \ldots, x_j)$ with $i \leq j$ and for all segments $(x_i, \ldots, x_n, x_1, \ldots, x_j)$ with $i > j$. The goal is to partition the cyclic sequence into segments with a minimum sum of penalties.

Describe an algorithm that solves the Cyclic Segmentation problem in $O(n^3)$ time. You may refer to the known Segmentation algorithm for usual (linear) sequences and make use of it, without repeating its details.

Problem 4 (12 points)

It should be well known that a particular element in a sorted array of size n can be found by binary search in $O(\log n)$ time. But if some element is misplaced, binary search can fail, if you incidentally look up the misplaced element. (Imagine a long book shelf where the books are sorted alphabetically by authors, but some careless user has taken one book and put it back at an arbitrary position.)

4.1. Give an algorithm for the Search problem in an array that is sorted in ascending order, except for at most one misplaced element. It should still work in $O(\log n)$ time. – Hint: Keep the idea of binary search, but enhance it by some error correction routine. (10 points)

4.2. One may get the idea to check in a preprocessing phase whether the array is correctly sorted (or detect a misplaced element), and then safely apply the original binary search algorithm. But how much time would this preprocessing take? Motivate your answer. (2 points)

Problem 5 (10 points)

A definition first: A Hamiltonian path in a graph is a path that visits every node exactly once. The Shortest Hamiltonian Path problem asks whether a given graph with edge weights has a Hamiltonian path whose total length does not exceed a given threshold.

As you know, a minimum spanning tree (MST) in a graph with edge weights can be found in polynomial time. Now we consider a related problem with stronger demands: Given a graph G with edge weights and an integer k, we wish to find an MST T where every node has a degree at most k in T. (Note carefully that the degree of a node in T can be smaller than its degree in G, since T contains only a subset of the edges of G.) The decision version asks for a spanning tree T where all node degrees in T are at most k and the sum of all edge lengths in T does not exceed a given threshold.

5.1. Show that this problem is NP-complete. You may use, without proof, the fact that the Shortest Hamiltonian Path problem is NP-complete.
(8 points)

5.2. One may get the idea to modify Kruskal's algorithm: We could maintain a partitioning of the node set into disjoint trees of degree at most k, and in every step insert a cheapest edge that joins two different trees and does not increase the degree of any node beyond k. Can we expect this algorithm to always produce an optimal solution? Motivate your answer. (2 points)

Solutions (attached after the exam)

1. We can move our coordinate system such that the river is parallel to the y-axis. The new coordinates of all candidate points are computable in $O(n)$ time. We sort them by their y-coordinates in $O(n \log n)$ time. Going through the sorted list, we determine a pair of candidate points from both sides of the river that has a minimum vertical distance v (i.e., minimum difference of y-coordinates). These two points have a distance at most $b + 2d + v$. (The Pythagorean theorem yields a better bound, but this simpler one is sufficient for our purpose.) Hence, it suffices to compute the distance of every candidate point P to all candidate points Q on the other side which might be closer to P than $b+2d+v$. In particular, their vertical distances to P must be between v and $b+2d+v$. Thus, all suitable points Q are located in two rectangles of size $d \times (b+2d)$. Since all points Q have mutual distances at least c and b, c, d are constants, only $O(1)$ points Q remain for each P, hence $O(n)$ distance calculations solve the problem. – So, the situation is similar to the conquer step in the closest-pair algorithm, however, with several modifications. (10 points)

2.1. Swapping 00 or 11 does not change the string. Hence, an optimal solution would swap only pairs 01 or 10. It follows that the characters 1 preserve the ordering they had in the initial string A. Therefore, the ith character 1 must move from a_i to b_i. Moreover, every swap moves only one character 1 by one step. This implies the claimed minimum number of swaps. (5 points)

2.2. There exists some index i with $c_i \neq b_i$. We can assume $c_i < b_i$; the opposite case is symmetric. Let i be the largest such index. Then, the character at position $c_i + 1$ in C is a 0. Thus, we have found a pair 10 in C where swapping moves the 1 one step closer to its destination in B.
(5 points)

2.3. Initially let $C := A$. Simply pick any pair as in 2.2 and swap it. This step is repeated until the current string C equals B. The number of swaps equals the lower bound from 2.1, hence it is optimal. (5 points)

2.4. Each of the $k < n$ characters 1 moves in only one direction, hence only $O(n^2)$ swaps are done. The following pair actually needs that many swaps: $A = 1 \ldots 10 \ldots 0$, $B = 0 \ldots 01 \ldots 1$, where $n = 2k$. (5 points)

3. For each of the n positions P between the items of the cyclic sequence, we guess that P is the end and start, respectively, of two neighbored segments of an optimal solution. We cut the cyclic sequence at P and obtain a usual linear sequence. We solve the problem for this sequence in $O(n^2)$ time by the known algorithm. We do this n times (for all P) and finally take the overall optimal result. (8 points)

4.1. If an element gets misplaced, it will become smaller than its left neighbor or larger than its right neighbor. Moreover, one misplaced element causes only one such corrupted pair of neighbors that are in the wrong order. Now we can search for an element x as in binary search, with some modifications: Whenever we compare x to some y in the array, we also check whether y is in a corrupted pair. In that case, we ignore the corrupted pair and instead compare x to some neighbor of it, knowing that this comparison yields a correct answer. In this way, each of the $O(\log n)$ steps of binary search still uses only $O(1)$ comparisons. (10 points)

4.2. Unfortunately, this would take $O(n)$ time, since all elements must be read. By skipping one element we could already overlook a possibly misplaced element. (2 points)

5.1. Our problem is in NP, since for a given tree T we can check the degrees of all nodes and compare the sum of the lengths of all edges in T to the given threshold. Furthermore, we can reduce the Shortest Hamiltonian Path problem to our problem in polynomial time, even for fixed $k = 2$: A spanning tree where all degrees are at most 2 is the same as a Hamiltonian path. That is, equivalence is evident here and does not need a separate proof. The reduction merely consists of copying the weighted graph and the threshold. (8 points)

5.2. No (unless P=NP), because this algorithm would run in polynomial time, but the problem is NP-complete due to 5.1. (2 points)