

# Algorithms Re-Exam

## TIN093/DIT093/DIT602

**Course:** Algorithms

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

**Date, time:** 25th August 2022, 14:00–18:00

**Building:** Johanneberg

**Responsible teacher:** Peter Damaschke, Tel. 5405, email [ptr@chalmers.se](mailto:ptr@chalmers.se)

**Examiner:** Peter Damaschke

**Exam aids:** dictionary,  
printouts of the Lecture Notes (possibly with own annotations),  
one additional 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 (12 points)

We are given an undirected connected graph  $G = (V, E)$  and a subset  $P \subset V$  of  $2k$  nodes. The problem is to pair up the nodes in  $P$ , that is, form  $k$  pairs of nodes, and connect every such pair  $\{u, v\}$  by some path with end nodes  $u$  and  $v$ . The only condition is that no two paths may share edges.

(Do not worry about the motivation of this problem. It appears as a subtask in some other route planning problems.)

Surprisingly, there always exists a solution, and it can be found by a greedy-like algorithm: Start with an arbitrary pairing and an arbitrary system of paths for it. Then choose two arbitrary paths that share some edge  $x - y$ , such as:

$s - \dots - x - y - \dots - t$  and  $u - \dots - x - y - \dots - v$ .

Replace them with two new paths

$s - \dots - x - \dots - u$  and  $t - \dots - y - \dots - v$

which do not use the edge  $x - y$  anymore. Repeat this step until no two paths share edges anymore.

1.1. Show that this algorithm always terminates and produces some valid solution in polynomial time. You need not prove a “good” specific time bound here (this would be quite laborious); it suffices to argue why the algorithm is correct and the complexity is polynomial.

Hint: It might be useful to study how the total length (number of edges) of all paths behaves. (8 points)

1.2. Finally we make the problem stricter: As before, the  $k$  paths in the solution must not share any edges. Moreover, every path in the solution must be a shortest path between its end nodes. Modify the above algorithm such that it solves this stricter problem, still in polynomial time, and motivate why your modification works correctly. (4 points)

## Problem 2 (9 points)

Let  $X = (x_1, x_2, \dots, x_n)$  be a given sequence of numbers, and let  $J$  be some positive constant. A subsequence of  $X$  is any selection of numbers from  $X$ , not necessarily consecutive, but appearing in the given order. More formally, we may choose  $s$  indices  $k_1 < k_2 < \dots < k_s \leq n$ , and the subsequence with these indices is then  $(x_{k_1}, x_{k_2}, \dots, x_{k_s})$ . A subsequence is called monotone if  $x_{k_1} \leq x_{k_2} \leq \dots \leq x_{k_s}$ . A jump is a pair of neighbored elements in the subsequence whose difference is at least  $J$ . The problem is now to find a monotone subsequence of  $X$  with a maximum number of jumps.

Example: Let  $X = (1, 3, 4, 7, 6, 12, 8, 15, 13, 18)$  and  $J = 4$ . Then  $(1, 6, 12, 18)$  is monotone, and every pair of neighbors is a jump. Another solution with 3 jumps is  $(3, 8, 13, 18)$ , and there are more possible solutions. But no subsequence with 4 jumps exists.

2.1. Develop a dynamic programming algorithm for the problem. As the idea is perhaps not totally obvious, we propose already a definition of a function: For every index  $k$ , let  $OPT(k)$  be the maximum number of jumps in a monotone subsequence that ends exactly with  $x_k$ . – Now describe how you compute the values  $OPT(k)$  efficiently, and argue why your method is correct. You need not describe the backtracing. (6 points)

(Please follow this hint and do not attempt to find a simple greedy algorithm instead, as this is doomed to failure.)

2.2. Give and motivate a time bound for your computation. Of course, it should be a “small” polynomial bound. You can assume arithmetic operations with numbers to be elementary operations. (3 points)

### Problem 3 (10 points)

Let  $G = (V, E)$  be an undirected graph with  $n$  nodes and  $m > n \log_2 n$  edges. Moreover, suppose that every node in  $V$  is represented by a point in the plane, with known coordinates. Let the length of any edge  $(u, v)$  in  $E$  be defined as the usual Euclidean distance of the points  $u$  and  $v$  in the plane. Now we want to find the shortest edge in  $E$ . Note that the lengths of the edges are not explicitly given, we can only compute them from the coordinates of their nodes. Since  $m > n \log_2 n$ , it would be nice to avoid naive computation and comparison of all edge lengths.

Take the known  $O(n \log n)$ -time algorithm for the Closest Points problem and modify it such that it solves the above problem within the same time bound. It suffices to explain informally what you change in the algorithm, and why this modification is correct and does not increase the time bound.

### Problem 4 (15 points)

A triangle in a graph is a set of three nodes  $u, v, w$  such that all edges  $uv, vw, vw$  exist (in other words, a clique of three nodes). A graph without triangles is called triangle-free. We want to show that the Independent Set problem remains NP-complete when it is restricted to triangle-free graphs. To this end we use the following reduction. Let  $G = (V, E)$  be any undirected graph with  $n$  nodes and  $m$  edges. We replace every edge  $(u, v)$  in  $E$  with a path  $u - x - y - v$ , where  $x$  and  $y$  are two fresh nodes. (For clarity: for every edge in  $E$ , two additional nodes are created; these are together  $2m$  new nodes.) Let  $H$  denote the resulting graph.

Prove the following statements:

- $H$  is triangle-free.
- The reduction runs in polynomial time.
- $G$  has an independent set with at least  $k$  nodes if and only if  $H$  has an independent set with at least  $k + m$  nodes.

Finally argue why these statements together imply the NP-completeness of Independent Set for triangle-free graphs.

### Problem 5 (14 points)

We are given a tree  $T$  with  $n$  nodes, where every edge has some given positive length. A vehicle is able to traverse the edges of  $T$ , in either direction and arbitrarily often. Starting from some fixed node  $r$ , we want the vehicle to visit every node of  $T$  at least once and finally return to  $r$ . The goal is to minimize the total length of such a tour. We denote this problem  $TOUR(T, r)$ .

Let  $r_1, \dots, r_d$  denote the nodes adjacent to  $r$ , and let  $T_i$  be the subtree that would be obtained by deleting the edge  $rr_i$ , and which contains the node  $r_i$ . The following recursive algorithm for  $TOUR(T, r)$  is proposed:

For  $i = 1$  to  $d$ , go from  $r$  to  $r_i$ , then perform an optimal tour that solves  $TOUR(T_i, r_i)$ , and go from  $r_i$  back to  $r$ .

5.1. Argue why this algorithm yields indeed a valid and optimal solution to the  $TOUR(T, r)$  problem. (12 points)

Hint: It is advisable to argue in a few steps (and motivate every step):

- (a) Why is the solution valid, i.e., satisfies the constraints of the problem?
- (b) How often does this algorithm traverse every edge?
- (c) In an arbitrary(!) valid solution, how often do we have to traverse every edge at least?
- (d) Now conclude optimality. But remember that the edges have some given lengths.

5.2. What is the running time of this algorithm, and why? (2 points)