Algorithms Exam
TIN093/DIT093/DIT602

Course: Algorithms

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

Date, time: 23rd October 2021, 14:00–18:00

Building: L

Responsible teacher: Peter Damaschke, Tel. 5405, email `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.

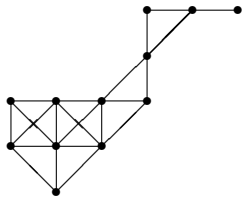
Inspection of grading (exam review): to be announced.

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)

Let $G = (V, E)$ be an undirected, connected graph whose nodes are grid points; these are points (x, y) with integer coordinates x, y in the plane. Two nodes $(x, y) \neq (x', y')$ are adjacent if and only if $|x - x'| \leq 1$ and $|y - y'| \leq 1$. We call G hole-free if, for any two nodes $u, v \in V$ on the same horizontal or vertical line, all grid points on this line between u and v are also nodes in V. (The picture shows an example.) The coordinates of any node v are denoted (x_v, y_v).

Given a hole-free grid graph $G = (V, E)$ and two nodes $s, t \in V$ with $x_s \leq x_t$ and $y_s \leq y_t$, we want to find a shortest path from s to t in G. We can assume $s = (0, 0)$, otherwise we move the coordinate system.

Below we formulate a greedy algorithm that shall find a shortest path. Let z be a variable for nodes in V, representing the position of a "piece" that we want to move from s to t.

(0) Initially let $z := s = (0, 0)$.
(1) If $x_z = x_t$ or $y_z = y_t$, then move directly (vertically or horizontally) to t and stop.
(2) While $x_z < x_t$ and $y_z < y_t$, do the following.
(2a) If $(x_z + 1, y_z + 1) \in V$, then move there; that is, $z := (x_z + 1, y_z + 1)$.
(2b) If $(x_z + 1, y_z + 1) \notin V$, then move to either $(x_z + 1, y_z)$ or $(x_z, y_z + 1)$.

More informally, step (2) says: If possible, make a diagonal step to get closer to t. Otherwise, go to the upper or the right neighbor.

We want to prove that our piece with variable position z will in fact traverse some shortest path from s to t. Since the correctness of step (1) is evident, only the correctness of (2) must be shown. Thus we can assume $x_t > 0$ and $y_t > 0$. Moreover, only the first step (when $z = s = (0, 0)$) needs to be correct, because all other steps obey the same greedy rule.

1.1. Let $z = (0, 0)$. We claim that, in case (2b) where $(1, 1) \notin V$, in fact, exactly one of the two nodes $(1, 0)$ and $(0, 1)$ exists in V. Explain why this is true, by using that the graph is hole-free. (4 points)

In the following we use these notations: Let P be some fixed shortest path from s to t. On P, let q be the first node with both $x_q > 0$ and $y_q > 0$. (Since P must finally reach t, such a node exists.) Let p be the node on P preceding q. Clearly, $x_p = 0$ or $y_p = 0$. The next two statements 1.2 and 1.3 imply the correctness of step (2a) and (2b), respectively.

1.2. Consider the case $(1, 1) \in V$. Assume $x_p = 0$. (Case $y_p = 0$ is similar). Take P and construct, by an exchange argument, a shortest path from s to t that visits $(1, 1)$ in the first step. (4 points)

1.3. Consider the case $(1, 1) \notin V$. We may assume that $(0, 1) \in V$ and $(1, 0) \notin V$. (The opposite case is similar.) Take P and construct, by an exchange argument, a shortest path from s to t that visits $(0, 1)$ in the first step. (4 points)

Do not think complicated. Drawing some pictures should make the situation quite obvious, in all 3 sub-exercises.

Problem 2 (15 points)

As you should know, the time bound $O(nW)$ for the standard dynamic programming algorithm for Knapsack is not truly polynomial, since the capacity W can be huge compared to the number n of items. Therefore, special cases that allow more efficient solutions can be of interest.

Specifically, consider the Knapsack problem where the weights (sizes) of items are restricted to integers $1, \ldots, s$, where s is some fixed limit.

2.1. Give an algorithm with time bound $O(n^2 s)$ for this case. Hint: Simply use the known algorithm and its time bound carefully. (5 points)

2.2. For each of the integer weights $j \leq s$, let R_j denote the set of items whose weight equals j. Show that, whenever an optimal solution contains any item from R_j, it must also contain *all* items from R_j with strictly larger values. (6 points)

2.3. The observation in 2.2 suggests the following algorithm: Present the items in some ordering where every R_j is sorted by decreasing values. Run the standard dynamic programming algorithm on this ordering, but whenever you decide not to take some item, say from R_j, then ignore all later items from R_j, too. (This does not improve the worst-case time bound $O(n^2 s)$, but it can save many calculation steps in concrete instances.)
Does this modified algorithm still work properly, or did we overlook some detail that invalidates it? Either give a short informal motivation why you think it is correct, or clearly point out the issue if you think it is incorrect. (4 points)

Problem 3 (10 points)

Let f be some function, whose arguments x are integers in the interval $[1, n]$, and whose function values $f(x)$ are integers in the interval $[1, m]$. Imagine the following setting: We do not know the function f, but we know that f is monotone increasing. That means, for any $x \leq z$ we always have $f(x) \leq f(z)$. Furthermore, we are able to do "experiments" of the following kind: We can choose any argument x and any value y, and test whether $f(x) \leq y$ or $f(x) > y$. (To avoid misunderstandings: The experiment does not yield the actual value $f(x)$, but we can only see the binary result of the comparison with y.) The goal is to determine the unknown function f by doing a small number of such experiments.
As an application scenario, suppose we want to figure out the forces y that materials of various thicknesses x can resist. Naturally, we expect thicker materials to be more robust, but the precise correlation must be determined empirically. Therefore we expose materials of different thicknesses to different forces and test whether they break or resist.

For every x, we may determine the value $f(x)$ by comparison to values y selected as in binary search. This way we need $O(n \log m)$ experiments in total, even if f is not monotone. Intuitively, it should be possible to use the prior knowledge that f is monotone, and thus be faster.
We propose the following divide-and-conquer style algorithm. First figure out $y = f(\lfloor n/2 \rfloor)$ by binary search. Then determine the "halves" of f recursively, once for all arguments in $[1, \lfloor n/2 \rfloor]$ and values in $[1, y]$, and once for all arguments in $[\lfloor n/2 \rfloor, n]$ and values in $[y, m]$.

3.1. Briefly argue why this algorithm determines f correctly and completely. (3 points)

More interestingly, the number of experiments is only $O(n + m)$, however the analysis would be quite technical. Moreover, and at first glance a bit surprisingly, the same efficiency for this problem can be achieved without divide-and-conquer, in a more "incremental" way:

3.2. Give an algorithm that determines the unknown monotone function f with $O(n + m)$ experiments. Hint: First compare $f(1)$ with 1. Then, in every step, increase x or y in a smart way, depending on the results of the experiments. Formulate your rules for that and argue why they are correct. Finally explain the $O(n + m)$ bound. (7 points)

Problem 4 (14 points)

4.1. Bundles of wooden beams shall be stored in a narrow corridor. No bundles can be stored side by side. Moreover, there are obstacles in the corridor that cannot be shifted. Hence there is space for bundles only between these obstacles. The lengths of all bundles and of the free spaces are known.

This leads to the following problem: Given a set of "beams" and a set of tall "spaces", all with known lengths, assign the beams to the spaces such that they all fit. That means, the sum of lengths of all beams in a space must not exceed the length of that space. The lengths are given as numbers (e.g., in meters) in usual decimal notation.

Is this problem solvable in polynomial time or NP-complete? Give either an algorithm or a reduction, in order to prove your claimed answer. (Of course, only one of these options can be correct.) In either case, argue also why the problem is in NP. (7 points)

4.2. We define a graph problem "Smash" as follows. Given a graph G and two integers c and k, delete k nodes and all edges incident to them, such that, in the remaining graph, every connected component has at most c nodes.

Is Smash solvable in polynomial time or NP-complete? Give either an algorithm or a reduction, in order to prove your claimed answer. (Of course, only one of these options can be correct.) In either case, argue also why the problem is in NP. – Hint: Look at the special case $c = 1$ first. (7 points)

Problem 5 (9 points)

A graph $H = (U, F)$ is called an induced subgraph of a graph $G = (V, E)$ if there exists a function $m : U \to V$ such that:

- m is injective, that is, any two nodes $u \neq u'$ from U are mapped to nodes $m(u) \neq m(u')$ in V.
- For any two nodes $u, u' \in U$ we have: u and u' form an edge in F, if and only if $m(u)$ and $m(u')$ form an edge in E.

Informally: Distinct nodes are mapped to distinct nodes, edges are mapped to edges, and non-edges are mapped to non-edges. Even more informally: G contains an exact copy of H.

The degree of a graph is the largest number of nodes adjacent to any single node. Let H be any fixed connected graph and Δ any fixed positive integer. After these definitions we are ready to formulate our problem:

Given a graph G of maximum degree Δ as input, we want to decide whether H is an induced subgraph of G. Provide an algorithm that solves this problem in $O(n)$ time, where n is the number of nodes in G.
Hints: Use BFS and the fact that H is connected. For the time analysis, take advantage of the assumption that H and Δ are fixed, and remember that O-notation ignores constant factors, even if they are "huge".

Solutions (attached after the exam)

1.1. We have $(0,0) \in V$ and $(1,1) \notin V$, and t has two positive coordinates. Assume that both $(1,0) \in V$ and $(0,1) \in V$ exist. Some path from $(0,0)$ to t must contain some point $(1,y)$ with $y > 1$, or some point $(x,1)$ with $x > 1$. Since G is hole-free, either case implies $(1,1) \in V$, a contradiction. Finally, if none of $(1,0)$, $(1,1)$, $(0,1)$ is in V, then no path from z to t can exist at all, which contradicts connectivity. (4 points)

1.2. We observe $y_p \geq 0$ and $x_q = 1$. Since $(1,1) \in V$ and $q = (1,y_q) \in V$, all nodes on the vertical line segment between them are also in V. The part of P until q has at least $y_p + 1$ edges. By going instead from s to $(1,1)$ and upwards to q we need only y_q edges. Since $y_q \leq y_p + 1$, we can replace the part of P until q with this path. In particular, we can go from s directly to $(1,1)$ in the first step. (4 points)

1.3. Since $(1,0) \notin V$, no vertices $(x,0)$ with $x > 0$ can exist either. Thus $p = (0,y_p)$. Now we conclude as above: The part of P until q has at least $y_p + 1$ edges. By going instead from s upwards to $(0,y_q - 1)$ and then to q we need only y_q edges. Since $y_q \leq y_p + 1$, we can replace the part of P until q with this path. In particular, we can go from s to $(0,1)$ in the first step. (4 points)

2.1. First compare W and ns. If $W \geq ns$, then all items fit in the knapsack, and this is clearly the optimal solution, "computed" in $O(n)$ time. If $W < ns$ then $nW < n^2 s$, such that applying the known algorithm with time bound $O(nW)$ yields the claim. (5 points)

2.2. This is seen by an exchange argument: Let A and B be items in R_j, where A has the larger value. If B is in the solution but A is not, we can replace B with A. The new solution is still valid, since A and B have the same weight, but the total value has increased. (6 points)

2.3. It is correct. Whenever we skip an item but choose a later one with equal weight and smaller value, the underlying dynamic programming algorithm would later "detect" that this choice was not optimal (due to 2.2). All what we do here is to avoid these unnecessary comparisons beforehand. (4 points)

3.1. We can argue inductively. The algorithm determines first $f(\lfloor n/2 \rfloor)$ and then the two "halves" of f recursively, hence it eventually determines all function values. Moreover, since f is monotone increasing, it suffices to search the function values in $[1, y]$ and $[y, n]$, respectively. (3 points)

3.2. Initially let $x := 1$ and $y := 1$. If the experiment says $f(x) > y$ then increment y (to find the higher value). If the experiment says $f(x) \leq y$ then return the result $f(x) = y$ and increment x. Repeat these steps until $x = n$ and $y = m$. The returned values are all correct, for the following reasons. For $x = 1$ we find the smallest y with $f(1) \leq y$, hence this y equals $f(1)$. Whenever we have incremented x, we know for the previous argument (which is now $x - 1$) and for the current y that $f(x - 1) = y$. Since f is monotone, $f(x)$ must be at least the current y, and we find the correct value due to the same argument as in case $x = 1$. Since x and y can only grow, and one of them strictly grows in every step, the bound $O(n + m)$ follows immediately, regardless of the "shape" of f. (7 points)

Extra remark (not part of the solution): It might seem paradoxical that finding the correct y-values by climbing up is more efficient than binary search, however, we are interested in the total time bound, rather than in minmizing the worst-case time spent on every single x.

4.1. Given a placement of the beams, one can easily add and compare the corresponding numbers in polynomial time. This shows membership in NP. The problem is NP-complete via the following polynomial-time reduction: Given an instance of Subset Sum, simply "create" beams whose lengths are the given numbers, and two spaces whose lengths are (1) the sum W, and (2) the sum of all beams' lengths minus W. Equivalance of the two instances is evident. (7 points)

4.2. Given a solution, we can check in polynomial time whether the graph after removal of that node set has only connected components with at most c nodes. This shows membership in NP. Moreover, for $c = 1$ this is exactly the Vertex Cover problem: delete at most k nodes such that the remaining graph is an independent set. Hence the following simple reduction shows NP-completeness: Given a graph G and an integer k as an instance of the Vertex Cover problem, take the same G and k, and $c := 1$. This yields an equivalent instance of Smash, in polynomial time. (7 points)

5. For every node v of G, we start BFS in v, in order to find a copy of H that contains v. Let h be the number of nodes of H. Since H is connected, such a copy of H must entirely be in the first h BFS layers. In particular, we can stop BFS after h layers. Since h and Δ are constant, these layers comprise only a constant (albeit perhaps large) number of nodes. Now we can search for a copy of H therein. Even naive exhaustive search needs only $O(1)$ time. Since we do this n times, the overall time is $O(n)$, admittedly with a large constant. (9 points)