Algorithm Design - Homework 1

Academic year 2019/2020

Instructor: Prof. Stefano Leonardi
Teaching Assistants: Philip Lazos and Rebecca Reiffenhäuser

Deadline: December 5, 2019

Make sure that the solutions are typewritten and clear to read. A complete answer consists of a clear description of an algorithm (an English description is fine), followed by an analysis of its running time and a proof that it works correctly.

Hand in your solutions and keep a copy for yourself. Solutions will be posted or presented after due date. In the final exam, you will be asked to explain your solutions and/or to go over your mistakes.

Collaboration policy. You can work on the exercises in teams of two people, and hand them in accordingly. However, make sure that each one of you has fully understood every solution you present. Do not copy any work from other students, the internet or other sources, and do not share your work with others outside your team. If at any point, any part of the exercises you hand in is apparent to be a copy of other work, this will result in the following consequences: All of your exercises, previous as well as upcoming ones, will be treated as if you did not hand them in at all, and you will have to participate in the written exam to make up for this. Please note that there will be no exception made, even if you are the original author of work someone else copied, or if your exercise partner is the one responsible. Therefore, please make sure to only choose a partner that you trust, and do not hand out your exercise solutions to others.

Late policy: Every homework must be returned by its due date. Homeworks that are late will lose 10% of the grade if they are up to 1 day (24h) late, 20% if they are 2 days late, 30% if they are 3 days late, and they will receive no credit if they are late by more than 3 days.

Formal Requirements: Please typeset your solutions (11 pt at least), and state the names of both team members at the beginning of each sheet you hand in. Make sure to name the exact exercise each part of the solution refers to, otherwise it will not be graded. Please start a new page for each main exercise in the assignment (i.e. exercise 1, 2, etc., but not for each subquestion in them). Make sure your solution takes no more than one page for each main exercise, because everything after the first page will not be taken into account. So, for example, when the assignment has four exercises, please hand in four pages, one for each exercise. All solutions have to be sent via email to

lazos@diag.uniroma1.it **or** rebeccar@diag.uniroma1.it

with subject line 'Algorithm Design - Homework 1'.

Office Hours: Philip and/or Rebecca are available for questions in room A102 at DIAG in via Ariosto 25 every Tuesday, from 12:00 to 14:00. Please feel free to come by.

Please refer to course's Web page for updates about the above aspects.

Exercise 1. Philip decided to supplement his income by participating in the popular game show 'Open the Boxes and keep the Best!'. The game is played in turns. At every turn, the host shows Philip a box that can be opened by paying $c_i > 0$. The box will contain a random prize: in particular its value could be anything in $\{0, 1, 2, \ldots, n\}$, uniformly at random.

At every step, Philip can stop playing and keep *only one prize, the best found so far*. Of course, the costs he paid are not refunded. All of the boxes are known in advance, as well as the order the host will present them. Help Philip find the optimal strategy and calculate his expected payoff, which is the prize he keeps minus the total cost paid.

Goal: Given as input the number of boxes k, the number of different rewards n as well as the cost c_i (which is guaranteed to be integer) of every box:

1. Design a $O(n^2 \cdot k)$ algorithm to find the expected optimal reward.
2. Improve the complexity to $O(n \cdot k)$.

Hint: One way is to use dynamic programming.

Exercise 2. When automatically checking coding exercises, we often run the programs in question against *test cases*, which are made up of one input and one output file. As expected, the program is fed the input file and then its output is compared with the output file. If these match for all test cases, we deem the program correct.

Coming up with these test cases is tricky. Specifically, we want to create test cases for the following problem: 'Given a complete weighted graph G, compute its minimum spanning tree T'. We have already created the desired output files containing the different minimum spanning trees, we now want to find the corresponding input files.

It is known that a graph can have many different MST's. To make testing easier, we want to ensure that the minimum spanning tree in every output file is unique for the graph described in the respective input file. The test cases also need to be able to tell apart algorithms that are wrong, but happen to find the correct MST by chance. For example, if all edges not in the MST have very large weights, a naive algorithm could find the MST by including only the light edges. Of course this algorithm would be incorrect in general. To avoid this, we want the input files to contain complete graphs G whose sum of edge weights is minimum. You need to design a program that can generate these input files.

Goal: Given a weighted tree T with n nodes, find the complete graph G *of minimum weight* such that $T \subseteq G$ and T is the unique minimum spanning tree of G. Assume all edge weights are integer.

1. Find an algorithm whose run time is *polynomial* in n.
2. Improve the complexity to $O(n \cdot \log n)$.

Hint: One way to do this is by thinking of Kruskal's algorithm and the cut property of minimum spanning trees.

Exercise 3. Federico now deals in exquisite chocolates, which he manufactures at home. He has a complicated distribution network set up in the university to sell his merchandise, but

every one of his friends $f \in F$ is only willing to handle a fixed amount n_f every week. He knows for every pair $(f_1, f_2) \in F$ if these two people see each other regularly, and of course, he also knows if he himself will meet them in person or not. Finally, some of his friends are able to sell a fixed amount $s_f \leq n_f$ to customers every week, but can only hand the rest of it over to other people.

a) Model the problem as a flow problem in a graph G, only consisting of regular vertices, one source, one sink, and capacitated edges to find out how many chocolates Federico should actually make every week. Give a formal definition of your network, and also draw a small example, e.g. for $|F| = 4$.

b) In the new semester, each friend in Federico's network will be attending courses in a certain building, according to their respective field of study, i.e. every $f \in F$ will have an associated building $b_f \in B$, where B are all the university buildings. Federico is worried this will affect his sales, since in every building $b \in B$, there will be only a certain regular number of customers available, limiting the weekly amount of chocolate sold there to c_b. Adjust your network with the added restrictions in order to find out if and how this impacts Federico's business.

Exercise 4. Giovanni has only k free hours until the huge party he is planning. He is now noticing that a lot of work still needs to be done. For example, invitations have to be sent, finishing up the decoration will take some hours, different food has to be prepared, more chairs have to be brought to the room, et cetera. For each of the n tasks in the task set J, he knows both an earliest time s_j to start task j, a deadline d_j when it has to be finished, and the length $l_j \in \mathbb{N}$ that specifies how long it will take a single person to do task j. Tasks, once started, cannot be paused and then finished later.

He has summed up all the l_j, and found out that just judging by those, he should be able to finish by the time the party starts, i.e. $\sum_{t \in T} l_j \leq k$. Now, he asks you to help with the matter. Your options are as follows: Either you and some others help him do the work, or, clearly preferable: You convince him he can do it alone. For that, you would need to find a way to order the jobs that will allow him to finish everything in time. You have had a quick glance at the very long TO-DO-list and are in doubt if you even should start looking for a feasible one-person-solution. Prove that indeed, the question if one exists is (in general) not one you can expect to solve with a quick algorithm.