Algorithms and Complexity
2017
Mästarprov1: Algorithms

Mästarprov 1 should be solved individually in written form and presented orally. No collaboration is allowed.

Written solutions should be handed in latest on **Wednesday, February 22 17.00,** to Johan or Jonas in written or printed form personally or in physical mailbox (CSC expedition). Be sure to save a copy of your solutions. Mästarprov 1 is a mandatory and rated part of the course. The test consists of four tasks. The test is roughly graded as follows: Two task correctly solved give an E. Three tasks correctly solved give a C and all tasks correctly solved give an A. You can read more about the grading criteria and the final grade on the course web page. The report should be written in English.

In all problems you should give an analysis of the time complexity of your algorithm and you should be able to argue for its correctness.

1. Improved Dijkstra

You analyze a graph that represents a project. The nodes represent different *stages* in the project and an arrow from i to j tells you that you can go from stage i directly to stage j. There is a positive weight $w(i, j)$ on the arrow that tells you the textitcost (whatever that is) of going from i to j. For stages that are not neighbors you define the cost of going between the stages as the length of the shortest path (sum of weights) between the stages. In this way you have a regular shortest path problem and you realize that you could use Dijkstra's algorithm for finding costs as distances. But then you notice one thing. The costs are integers in the set $\{1, 2, 3, ..., 17\}$.You remember that Dijkstra's algorithm (in it's simplest form) has time complexity $O(|V|^2 + |E|)$. You also realize that your graph is rather sparse so that $|E| < 5|V|$ (but $|V|$ can still be large). If you run normal Dijkstra you would still get $O(|V|^2)$ as time complexity. But maybe you could use the previous observation about the range of the costs and modify Dijkstra's algorithm and get a lower time complexity?

Your task is to modify Dijkstra's algorithm and use the fact that the costs (distances) are integers in $[1, 17]$ so you get a time complexity $O(|V|)$. You must argue for the correctness of your algorithm.

2. Flat seqeuences

We will study sequences of postitive numbers $x_1, x_2, ..., x_n$ of a special type. We will (just in this problem) call a sequence *flat* if , for all k, s such that $1 \leq k \leq n$ and $1 \leq s < k$, we have $0 < \frac{x_k - x_s}{k-s} < 1$. We want to find long consecutive subsequences that are flat. In order to do so, we use dynamic programming and define $FL(k)$ as the length of the longest consecutive flat sequence ending in x_k.

(For instance, if $FL(k) = 4$, then this sequence is $x_{k-3}, x_{k-2}, x_{k-1}, x_k$ which is assumed to be flat. This sequence can, of course, be the initial part of a longer flat sequence end in x_{k+m}, say.)

Your task is to find a recursion formula for $FL(k)$ as a function of $FL(k-1)$ and perhaps some of the previous x_i:s. You will probably have to consider separate cases and some base case. When you have stated your algorithm, write a program that solves the problem in dynamic programming-style (bottom-up). The program should take the number sequence and an integer k such that $1 \leq k \leq n$ as input and return $FL(k)$. You must argue for the correctness of your algorithm.

3. Maximum Flow?

Let us assume that we have a computer network with connections in form of a directed graph G. On all edges e we have capacities $c(e)$. The communication between the computers can then be considered to be a network flow (as in the lectures) from a sorce s to a sink t. . We can monitor the actual flow on the edges and we suspect that the network us fully used in the sense that we have a maximum flow. But how do we know for sure? We can first find the flow from s, let us say it is F_0. We can then use the Flow algorithm to find a maximum flow in G ans see if it is F_0 or not. This algorithm has a time complexity $O(|V||E|^2)$. But we claim that since we already have a given flow, we can check in much shorter time if this flow is maximal. But how?

Your task is to give an algorithm which takes a graph G (directed, with s and t and capacities) and a flow F (given by $f(e)$ for all $e \in E$) as input and decides if F is maximal in G or not. Find the time complexity of your algorithm. It should be no worse than $O(|E|)$. (Yes, that low.) You must argue for the correctness of your algorithm.

4. Bad railroad

In this problem we have a railroad which we can represent as a real line with $n+1$ points $x_0, x_1, x_2, ..., x_n$ corresponding to stations s_0 = start position, $s_1, s_2, ..., s_n$ = end postion. We have a passenger train going from s_0 to s_n. There are m passengers boarding the train at s_0 and they want to go to different stations along the line. Normally, the train would stop at all n stations, s_0 excluded. But on a particular day, for some reasons the train can just stop at $k < n$ stations between s_0 and s_n. The staff on the train must decide at which stations the stops should be, guided by the demands of the passengers. They collect information on the form $g_1, g_2, ..., g_m$ where g_i is the index of the station passenger i wants to go to. The staff should find a set $h_0, h_1, h_2, ..., h_k$ of indices of the stations the train will stop at and $0 = h_0 < h_1 < h_2 < ... < h_k < n$. We can see that passengers with $g_j = n$ do not cause any problems. The staff now employs the policy that if for any passenger, say j, the train doesn't stop at g_j, the passenger is asked to get of the train at the largest h_i such that $h_i < g_j$. (This includes the possibility that the passenger "get off" the train at h_0, i.e., does not border the train at all.)

Of course, this will make some passengers very angry. How angry? The mathematically inclined driver of the train makes the following estimate. The anger $anger(j)$ of passenger j will be 0 if the train stops at station number g_j. Otherwise, if the passenger is forced to get of at h_i we have the estimate $anger(j) = A + B\sqrt{x_{g_j} - x_{h_i}}$, where A and B are some positive constants.

The staff wants to find $h_0, h_1, h_2, ..., h_k$ such that $\sum_j anger(j)$ is minimal. Find an algorithm that solves this problem. The algorithm must be polynomial in n, m, k. You must argue for the correctness of your algorithm.