TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in algorithms
Spring Semester, 2012/2013

Homework 4, May 28, 2013

Due on June 11.

Comments on previous homework: (a) As pointed out by some of you there is a trivial solution to homework 3 question 2, just by scaling down the flow by a factor of Δ/Γ. This may require precision which is too large if that algorithm of Goldberg and Rao would do it in each iteration. The point is that you can do it easily over the integers in linear time. Please try to do it. You can solve question 2b in homework 2 using the same technique. Just use a topological order with respect to the arcs that carry flow and use it carefully to return the flow. There is no need to hand it in again. Just make sure you understand it.
(b) A correction to question 4 in homework 3 appears as question 1 below.

1. (This is the corrected version of question 4 in homework 3.) In this question we develop in two stages an algorithm that finds a blocking flow from s to t in a directed acyclic graph $G(V, E)$ in $O(m \log(\frac{n^2}{m}))$ time where $n = |V|$ and $m = |E|$.

Each vertex in $V \setminus \{s\}$ is either *blocked* or *unblocked*. Initially all vertices in $V \setminus \{s\}$ are unblocked. We also maintain a preflow f which is initialized by setting $f(s, v) = u(s, v)$ for all edges outgoing from s. The excess $e(v)$ of a vertex v is the difference between the flow incoming to v and the flow outgoing of v and is maintained nonnegative for all v.

A vertex v with $e(v) > 0$ (more flow entering v than outgoing of v) is *active*. We denote by $r(v, w)$ the residual capacity of an arc (v, w) with respect to the current preflow.

The algorithm makes use of the three operations push, pull, and block. A push operation applies to an arc (v, w) such that v is active, v and w are unblocked, and $r(v, w) > 0$. It consists of increasing $f(v, w)$ by $\min\{e(v), r(v, w)\}$. A pull operation applies to an arc (v, w) such that w is active and blocked and $f(v, w) > 0$. It consists of decreasing $f(v, w)$ by $\min\{e(w), f(v, w)\}$. A block operation applies to an unblocked vertex v such that, for every arc (v, w), either $r(v, w) = 0$ or w is blocked. It consists of making v blocked.

A *discharge* operation applies to an active vertex v. The operation either pushes on an arc (v, w), blocks v, or pulls on an arc (u, v), whichever applies, and repeats this until v becomes inactive.

We may assume that s has no incoming arcs and let T be a topological order of the graph that starts with s. Let T^r be the reverse of T. Let L be a concatenation of T^r and T (we can remove one of the two consecutive copies of s in L). The *first-active blocking flow algorithm* uses the list L to determine the order in which to process active vertices. A blocked active vertex will be marked in T^r and a non blocked active vertex will be marked in T.

The algorithm consists of repeating the following three steps until there are no active vertices: Select the first marked vertex in L, say v. Apply a discharge operation to v. If the discharge has

made v inactive then unmark v. If the discharge made v blocked while it remains active then we unmark v in T and mark it in T^r. While discharging v we may make other vertices active. We mark each such vertex either in T or in T^r as appropriate.

a. Prove carefully and formally that this algorithm finds a blocking flow in $O(n^2)$ time. (For the analysis divide the discharge operations into phases and argue that in each phase there is one nonsaturating push/pull out of each vertex.)

b. Show how to combine the first-active blocking flow algorithm with dynamic trees and finger search trees to reduce the running time to $O(m \log \frac{n^2}{m})$. Argue formally that this is indeed the running time of your algorithm.

2. Show how to improve the order maintenance data structure presented in class such that the amortized time per operation is $O(1)$.

3. Prove a bound of $O(mn^{2/3} \log n)$ time on the cycle detection algorithm that uses a topological order and two way search presented in class.

4. Consider the dynamic graph connectivity problem **on an undirected forest** where we allow only two operations:
a) *connected*(v, w) that returns "yes" if and only if v and w are in the same connected component, and
b) *delete*(v, w) that deletes the edge (v, w) from the forest.
Assume we perform k such operations on a forest with n nodes. Describe an efficient (in an amortized sense) algorithm for the problem. (Note that from the material we learned in class easily follows a solution that runs in $O(k \log n)$ time, but better solutions exist in particular if $k >> n$.)