University of Washington
Department of Computer Science and Engineering
CSE 421, Winter 2024

February 14, 2024

Homework 7, Due Friday, February 23, 11:59 pm, 2024

Turnin instructions: Electronic submission on gradescope using the CSE 421 gradescope site. Submit the assignment as a PDF, with separate pages for different numbered problems. Problems consisting of multiple parts (e.g., 2a, 2b) can be submitted on the same page.
For all of these problem, provide a justification as to why your algorithm correctly solves the problem. Give and justify the run time for your algorithms.

Problem 1 (10 points):

(Page 324, Exercise 13.) The problem of searching for cycles in graphs arises naturally in financial trading applications. Consider a firm that trades shares in n different companies. For each pair $i \neq j$, they maintain a trade ratio r_{ij}, meaning that one share of i trades for r_{ij} shares of j. Here we allow the rate r to be fractional; that is, $r_{ij} = \frac{2}{3}$ means that you can trade three shares of i to get two shares of j.

A *trading cycle* for a sequence of shares $i_1, i_2, \ldots, i_k$ consists of successively trading shares in company i_1 for shares in company i_2, then shares in company i_2 for shares i_3, and so on, finally trading shares in i_k back to shares in company i_1. After such a sequence of trades, one ends up with shares in the same company i_1 that one starts with. Trading around a cycle is usually a bad idea, as you tend to end up with fewer shares than you started with. But occasionally, for short periods of time, there are opportunities to increase shares. We will call such a cycle an *opportunity* cycle, if trading along the cycle increases the number of shares. This happens exactly if the product of the ratios along the cycle is above 1. In analyzing the state of the market, a firm engaged in trading would like to know if there are any opportunity cycles.

Give a polynomial-time algorithm that finds such an opportunity cycle, if one exists.

Problem 2 (10 points):

The Chvatal-Sankoff constants are mathematical constants that describe the length of the longest common subsequences of random strings. Given parameters n and k, choose two length n strings A and B from the same k-symbol alphabet, with each character chosen uniformly at random. Let $\lambda_{n,k}$ be the random variable whose value is the length of the longest common subsequence of A and B. Let $E[\lambda_{n,k}]$ denote the expectation of $\lambda_{n,k}$. The Chvatal-Sankoff constant γ_k is defined as

$$\gamma_k = \lim_{n \to \infty} \frac{E[\lambda_{n,k}]}{n}.$$

Experimentally determine (by implementing an LCS algorithm), the smallest value of k, such that $\gamma_k < \frac{2}{5}$. In other words determine how large an alphabet needs to be so that the expected length

of the LCS of two random strings is less than 40% the length of the strings. Submit your code and a table of computed values to justify your answer.

You do not need to implement the memory efficient version of LCS (Section 6.7), but you are welcome to if you want a challenge, and should submit your code if you do so.

Problem 3 (10 points):

Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary flow network, with a source s, a sink t and a positive integer capacity c_e on every edge e; and let (A, B) be a minimum $s-t$ cut with respect to these capacities $\{c_e : e \in E\}$. Now suppose we add 1 to every capacity; then (A, B) is still a minimum $s-t$ cut with respect to these new capacities $\{1 + c_e : e \in E\}$.

Problem 4 (10 points):

You are given a flow network with unit-capacity edges: it consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter k.

The goal is to delete k edges so as to reduce the maximum $s-t$ flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum $s-t$ flow in $G' = (V, E - F)$ is as small as possible subject to this.

Give a polynomial time algorithm to solve this problem, and justify that your algorithm is correct.

Problem 5 (10 points):

In a standard $s-t$ Maximum-Flow Problem, we assume edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant of the Maximum-Flow problem with node capacities.

Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative node capacities $\{c_v \geq 0\}$ for each $v \in V$. Given a flow f in this graph, the flow through a node v is defined as $f^{\text{in}}(v)$, the sum of the flows on the incoming edges to v. We say that a flow is feasible if it satisfies the usual flow-conservation constraints and the node-capacity constraints: $f^{\text{in}}(v) \leq c_v$ for all nodes.

Give a polynomial-time algorithm to find an $s-t$ maximum flow in such a node-capacitated network. Justify the correctness of your algorithm. (Note: While the Ford-Fulkerson algorithm for network flow is not technically polynomial time, section 7.4 describes a *strongly polynomial time* algorithm for network flow, and it can be invoked as a subroutine in your solution.)