Practice Problems for Midterm 2

Midterm 2 will be on Wednesday, November 1st. The exam will be closed-book and closed notes, but you will be allowed one cheat-sheet (front and back).

Disclaimer: These are practice problems. They do not necessarily reflect the actual length, difficulty, or coverage for the exam. You should also be sure to look over your homework problems (problem set 3 and 4) as well as the in-class practice problems.

Problem 0. You should expect one problem in which you will be asked to work an example of one of the algorithms we have presented in class.

Problem 1. a. Let $G = (V, E)$ be a flow network with source s, sink t, and an integer capacity $c(u, v)$ for each edge $(u, v) \in E$. Let $C_{max(u,v)\in E}c(u, v)$ (C = the maximum capacity edge).
True or False: The minimum cut of G has capacity at most $C * |E|$.

b. **True or False.** The worst-case running time of the Ford-Fulkerson network flow algorithm has a true polynomial run time.

c. Given the initial array $< 2, 4, 1, 3 >$, how many inversions occurred?

d. A solution to a dynamic programming problem is expressed recursively. The top-down approach applies recursion directly to solve the problem. Due to the overlapping nature of the subproblems, what often happens and what technique is utilized to fix this problem?

Problem 2. Recall that a *bipartite graph* is an undirected graph G whose vertex set is partitioned into two sets $U = u_1, u_2, ..., u_m$ and $V = v_1, v_2, ..., v_n$, such that all edges have one endpoint in U and one endpoint in V. Two edges (u_i, v_j) and (u'_i, v'_j) are said to *cross* if either $i < i'$ and $j > j'$ or if $i > i'$ and $j < j'$. A *non-crossing* subset is a subset of edges in G in which no two edges cross one another (see figure below).

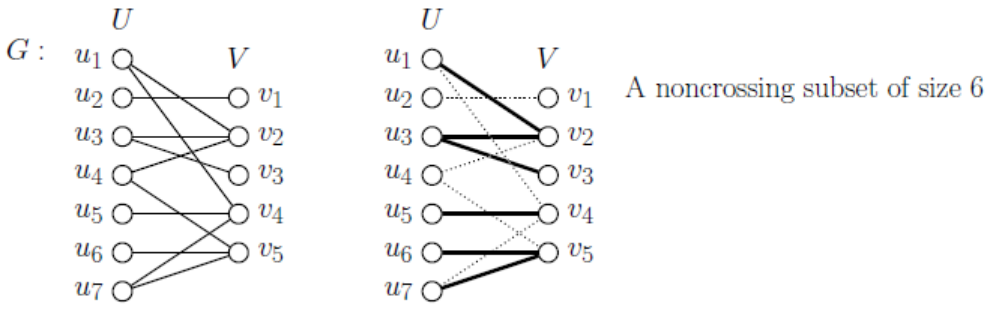

Figure 1: Problem 2.

Give a dynamic programming algorithm, which given a bipartite graph $G = (U, V, E)$, computes the size of the *maximum non-crossing subset* for G. It is sufficient to give just the recursive rule. (Hint: The algorithm is structurally similar to the LCS algorithm.)

Problem 3. Recall that in the longest common subsequence (LCS) problem the input consists of two strings $X =< x_1, ..., x_m >$ and $Y =< y_1, ..., y_n >$ and the objective is to compute the longest string that is a subsequence of both X and Y. For each of the following variations, present a short DP formulation. (It suffices to provide the recursive rule, similar to what we did with the standard LCS problem.)

a. (LCS with wild cards) Each of the strings X and Y may contain a special character "?", which is allowed to match any single character of the other string, except another wild-card character (see the figure below (a)).

b. (LCS with swaps) Any two consecutive characters of either string are allowed to be swapped before matching in the LCS (see the figure below (b)).

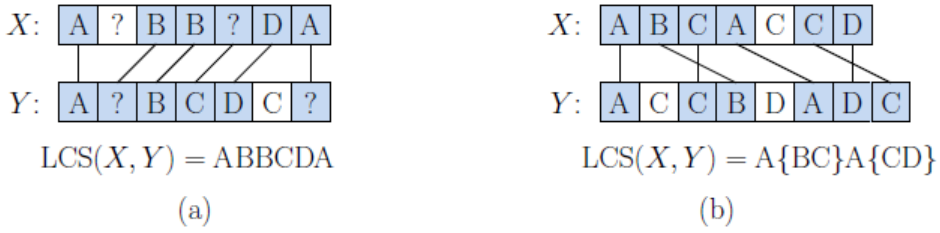

Figure 2: Problem 3.

In all cases, your revised rule should admit an $O(mn)$ time solution.

Problem 4. Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are *equivalent* if they correspond to the same account.
It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.
Their question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester. Show how to decide the answer to their question with only $O(n \ log \ n)$ invocations of the equivalence tester.

Problem 5. Consider the following problem. 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 the 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.

Give a polynomial-time algorithm to solve this problem.