Homework 4 (CS 401)

Deadline: **Monday May 2, 11:59 pm CST**

1. (20 pts)

Consider the flow network with source s, sink t, and internal nodes a, b, c, and d. The capacities on the edges are

$$c_{(s;a)} = 16;\ c_{(s;b)} = 13;\ c_{(a;c)} = 12;\ c_{(b;a)} = 4;\ c_{(c;b)} = 9;\ c_{(b;d)} = 14;\ c_{(d;c)} = 7;\ c_{(c;t)} = 20;\ c_{(d;t)} = 4;$$

and the current flow value specified as

$$f_{(s;a)} = 11;\ f_{(s;b)} = 4;\ f_{(a;c)} = 11;\ f_{(b;a)} = 0;\ f_{(c;b)} = 0;\ f_{(b;d)} = 4;\ f_{(d;c)} = 0;\ f_{(c;t)} = 11;\ f_{(d;t)} = 4:$$

a) Draw the flow network and show one step in the Ford-Fulkerson algorithm.
b) Specify the min-cut and its value in this network.

2. (25 pts)

Suppose you and your friend Alanis live, together with $n - 2$ other people, at a popular off-campus cooperative apartment, the Upson Collective. Over the next n nights, each of you is supposed to cook dinner for the co-op exactly once, so that someone cooks on each of the nights. Of course, everyone has scheduling conflicts with some of the nights (e.g., exams, concerts, etc.), so deciding who should cook on which night becomes a tricky task. For concreteness, let's label the people $\{p_1, . . . , p_n\}$,the nights $\{d_1, . . . , d_n\}$; and for person p_i, there's a set of nights $S_i \subset \{d_1, . . . , d_n\}$ when they are not able to cook. A feasible dinner schedule is an assignment of each person in the coop to a different night, so that each person cooks on exactly one night, there is someone cooking on each night, and if p_i cooks on night d_j, then $d_j \in S_i$.

(a) Describe a bipartite graph G so that G has a perfect matching if and only if there is a feasible dinner schedule for the co-op.

(b) Your friend Alanis takes on the task of trying to construct a feasible dinner schedule. After great effort, she constructs what she claims is a feasible schedule and then heads off to class for the day. Unfortunately, when you look at the schedule she created, you notice a big problem. $n - 2$ of the people at the co-op are assigned to different nights on which they are available: no problem there. But for the other two people, p_i and p_j, and the other two days, d_k and d_l, you discover that she has accidentally assigned both p_i and p_j to cook on night d_k, and assigned no one to cook on night d_l. You want to fix Alanis's mistake but without having to recompute everything from scratch. Show that it's possible, using her "almost correct" schedule, to decide in only $O(n^2)$ time whether there exists a feasible dinner schedule for the co-op. (If one exists, you should also output it.)

3. (25 pts)

You are designing an interactive image segmentation tool that works as follows. You start with the image segmentation setup we described in the class (KT 7.10), with n pixels as the set of neighboring pairs. And parameters $\{a_i\}$, $\{b_i\}$, $\{p_{ij}\}$. We will make two assumptions about this instance. First, we will suppose that each of the parameters $\{a_i\}$, $\{b_i\}$, $\{p_{ij}\}$ is a non-negative integer between 0 and d, for some number d. Second, we will suppose that the neighbor relation among the pixels has the property that each pixel is a neighbor of at most four other pixels (so in the resulting graph, there are at most four edges out of each node).

You first perform an initial segmentation $(A_0;B_0)$ so as to maximize the quantity $q(A_0;B_0)$. Now, this might result in certain pixels being assigned to the background, when the user knows that they ought to be in the foreground. So when presented with the segmentation, the user has the option of mouse-clicking on a particular pixel v_1, thereby bringing it to the foreground. But the tool should not simply bring this pixel into the foreground; rather, it should compute a segmentation $(A_1;B_1)$ that maximizes the quantity $q(A_1;B_1)$ subject to the condition that v_1 is in the foreground. (In practice, this is useful for the following kind of operation: in segmenting a photo of a group of people, perhaps someone is holding a bag that has been accidentally labeled as part of the background. By clicking on a single pixel belonging to the bag, and re-computing an optimal segmentation subject to the new condition, the whole bag will often become part of the foreground.)

In fact, the system should allow the user to perform a sequence of such mouse-clicks $v_1, v_2, \ldots, v_t$; and after mouse-click v_i, the system should produce a segmentation $(A_i; B_i)$ that maximizes the quantity $q(A_i; B_i)$ subject to the condition that all of $v_1, v_2, \ldots, v_i$ are in the foreground.

Give an algorithm that performs these operations so that the initial segmentation is computed within a constant factor of the time for a single maximum flow, and then the interaction with the user is handled in $O(dn)$ time per mouse-click.

(Note: Solved Exercise 1 in KT-7 (page 411) is a useful primitive for doing this. Also, the symmetric operation of forcing a pixel to belong to the background can be handled by analogous means, but you do not have to work this out here.)

4. (30 pts)

you are working with a large database of employee records. For the purpose of this question, we'll picture the database as a two-dimensional table T with a set R of m rows and a set C of n columns; the rows correspond to individual employees, and the columns correspond to different attributes.

To take a simple example, we may have four columns labeled *name, phone number, start date, manager's name* and a table with five employees as shown here.

name	phone number	start date	manager's name
Alanis	3-4563	6/13/95	Chelsea
Chelsea	3-2341	1/20/93	Lou
Elrond	3-2345	12/19/01	Chelsea
Hal	3-9000	1/12/97	Chelsea
Raj	3-3453	7/1/96	Chelsea

Given a subset S of the columns, we can obtain a new, smaller table by keeping only the entries that involve columns from S. We will call this new table the projection of T onto S, and denote it by $T[S]$. For example, if $S = \{name, start\ date\}$, then the projection $T[S]$ would be the table consisting of just the first and third columns.

There's a different operation on tables that is also useful, which is to permute the columns. Given a permutation p of the columns, we can obtain a new table of the same size as T by simply reordering the columns according to p. We will call this new table the permutation of T by p, and denote it by T_p.

All of this comes into play for your particular application, as follows. You have k different subsets of the columns $S_1, S_2, \ldots, S_k$ that you're going to be working with a lot, so you'd like to have them available in a readily accessible format. One choice would be to store the k projections $T[S_1], T[S_2], \ldots, T[S_k]$, but this would take up a lot of space. In considering alternatives to this, you learn that you may not need to explicitly project onto each subset, because the underlying database system can deal with a subset of the columns particularly efficiently if (in some order) the members of the subset constitute a prefix of the columns in left-to-right order. So, in our example, the subsets {name, phone number} and {name, start date, phone number,} constitute prefixes (they're the first two and first three columns from the left, respectively); and as such, they can be processed much more efficiently in this table than a subset such as {name, start date}, which does not constitute a prefix. (Again, note that a given subset S_i does not come with a specified order, and so we are interested in whether there is some order under which it forms a prefix of the columns.)
So here's the question: Given a parameter $l < k$, can you find l permutations of the columns $p_1, p_2, \ldots, p_l$ so that for every one of the given subsets S_i (for i = 1, 2, …, k), it's the case that the columns in S_i constitute a prefix of at least one of the permuted tables $T_{p1}, T_{p2}, \ldots, T_{pl}$? We'll say that such a set of permutations constitutes a valid solution to the problem; if a valid solution exists, it means you only need to store the l permuted tables rather than all k projections.

Give a polynomial-time algorithm to solve this problem; for instances on which there is a valid solution, your algorithm should return an appropriate set of l permutations.

Example. Suppose the table is as above, the given subsets are

S1= {name, phone number},
S2 = {name, start date},
S3 = {name, manager's name, start date},

and $l = 2$. Then there is a valid solution to the instance, and it could be achieved by the two permutations

p1 = {name, phone number, start date, manager's name},
p2 = {name, start date, manager's name, phone number}.

This way S_1 constitutes a prefix of the permuted table T_{p1}, and both S2 and S3 constitute prefixes of the permuted table T_{p2}.