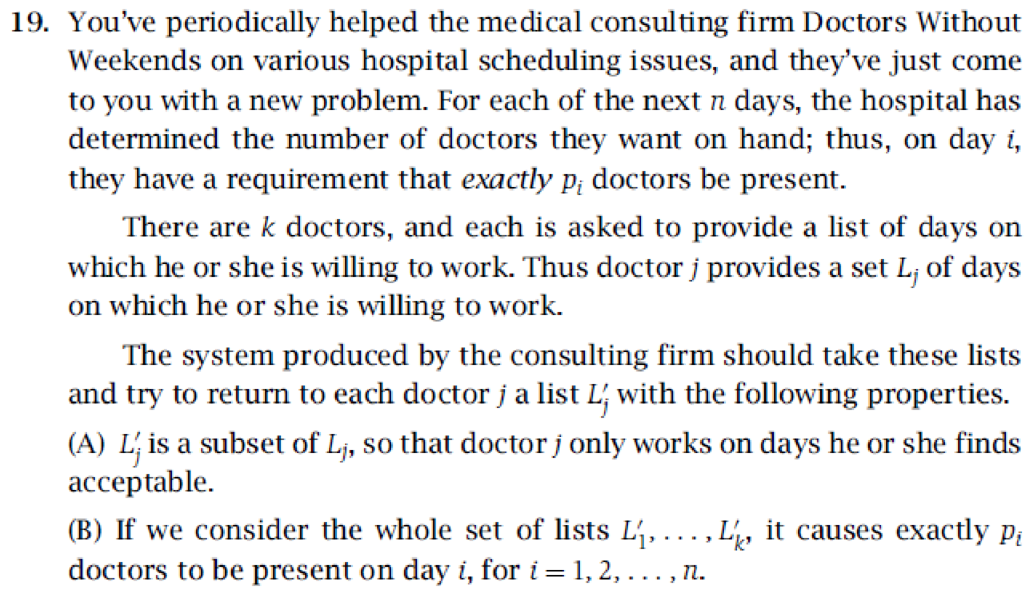
CSCE 586 HW 4

Marvin Newlin

29 November 2018

1. **Chapter 7 Problem 19 part a:**



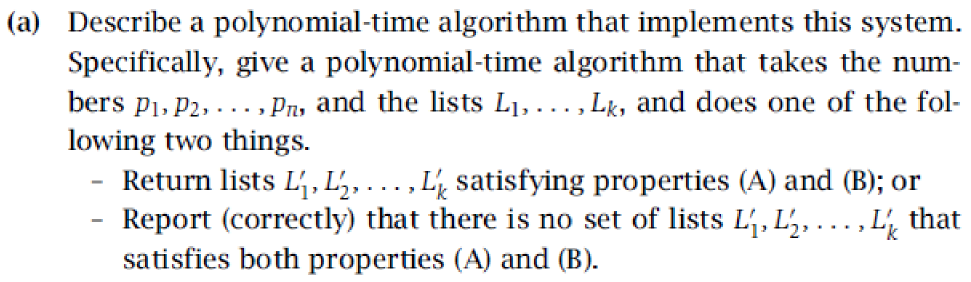
19. You've periodically helped the medical consulting firm Doctors Without Weekends on various hospital scheduling issues, and they've just come to you with a new problem. For each of the next n days, the hospital has determined the number of doctors they want on hand; thus, on day i, they have a requirement that *exactly* p_i doctors be present.

There are k doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus doctor j provides a set L_j of days on which he or she is willing to work.

The system produced by the consulting firm should take these lists and try to return to each doctor j a list L'_j with the following properties.

(A) L'_j is a subset of L_j, so that doctor j only works on days he or she finds acceptable.

(B) If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.



(a) Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers $p_1, p_2, \ldots, p_n$, and the lists $L_1, \ldots, L_k$, and does one of the following two things.

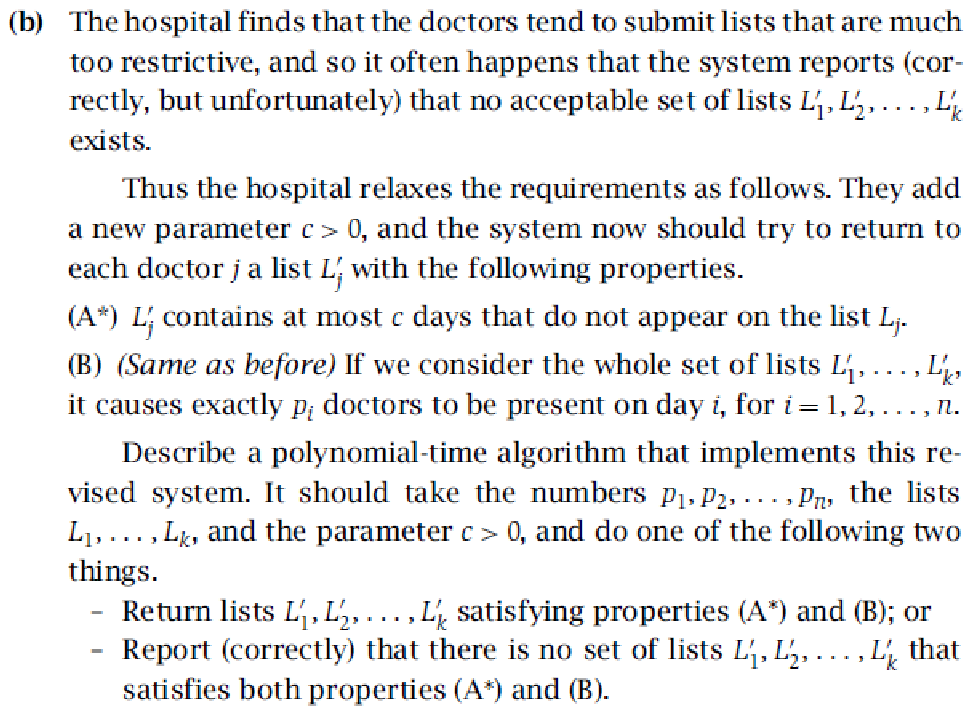
- Return lists $L'_1, L'_2, \ldots, L'_k$ satisfying properties (A) and (B); or
- Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties (A) and (B).

Sol'n:
Description: Let d_i be the number of doctors that can work on day i. Then the algorithm is as follows:
If $d_i < p_i$ then there is no possible assignment since we don't have enough available that day.
Else, pick p_i of the d_i doctors and return that as the assignment.

Analysis: This algorithm runs in $O(n)$ time. In the worst case, the algorithm makes an assignment for each of the n days and so it runs n times, yielding $O(n)$. In the best case, $d_1 < p_1$ so the algorithm terminates after one iteration. In the average case we will run through some fraction of the n iterations before terminating so the runtime will still be $O(n)$.

Difficulty: 4, time spent: 30 minutes

2. **Chapter 7 Problem 19 part b:**



(b) The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no acceptable set of lists $L'_1, L'_2, \ldots, L'_k$ exists.

Thus the hospital relaxes the requirements as follows. They add a new parameter $c > 0$, and the system now should try to return to each doctor j a list L'_j with the following properties.

(A*) L'_j contains at most c days that do not appear on the list L_j.

(B) *(Same as before)* If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

Describe a polynomial-time algorithm that implements this revised system. It should take the numbers $p_1, p_2, \ldots, p_n$, the lists $L_1, \ldots, L_k$, and the parameter $c > 0$, and do one of the following two things.

- Return lists $L'_1, L'_2, \ldots, L'_k$ satisfying properties (A*) and (B); or
- Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties (A*) and (B).

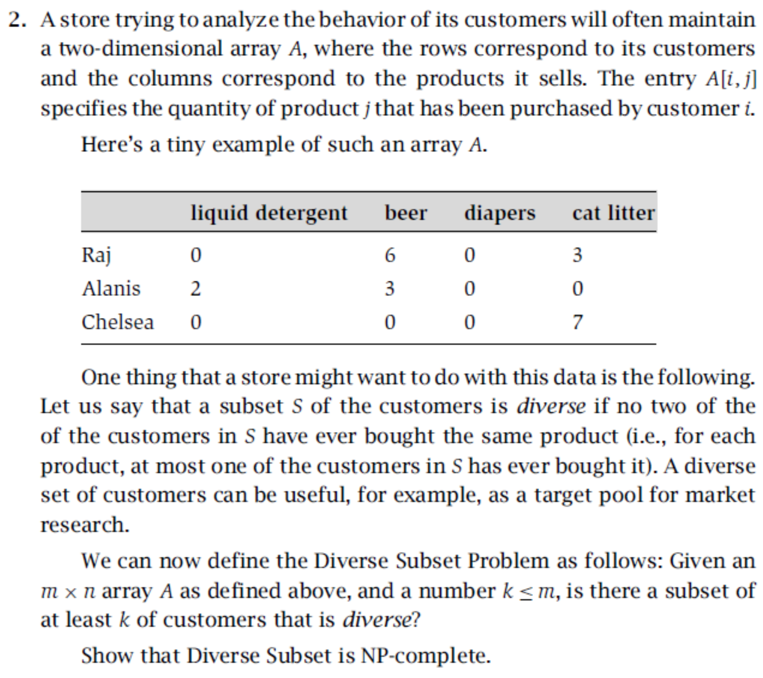
Sol'n:
Description: As before, if $p_i >= d_i$ we don't have a problem. However, if $d_i < q_i$, then we must find a way around this.
We can see that this is a version of bipartite matching so to set this up we construct a network $G = (V, E)$ with a node u_i for each doctor i, and

a node v_j for each day j. To build a network, we add an edge (s, u_i) for each u_i and edge (v_j, t) for each node v_j.
Then we add an edge for each (u_i, v_j) such that doctor i doesn't want to work on day j. This edge has a capacity of 1. Here we see that we have a bipartite matching problem in which the u_i's are X and v_j's are Y. We can then use Thm 7.37's result in [1] that a maximum matching is equal to the value of the maximum flow across the network. We know that we have a satisfying assignment if our max flow saturates all of the incoming edges into t. We return the lists $L'_1, ..., L'_k$ where each list $L'_i = \{(u_i, v_j) \mid (u_i, v_j) \notin E\}$.
Proof: By thm 7.37 in [1], we know that the max matching is equal to the value of the maximum flow. We must show that this matching provides a satisfying assignment. If we have a deficit on the given day then we need to find an assignment works around this. On each day i, we need $p_i - d_i$ doctors. If the max flow across the network doesn't saturate the incoming edges of t, then there is some day v_j that doesn't get flow to it even though each doctor can only specify c days they don't want to work. This means that there is no satisfying assignment.

Analysis: Since Thm 7.37 utilizes the Ford-Fulkerson algorithm to find the equivalence of bipartite matching and maximum flow, we know that it has the same runtime. Thus, the runtime is O(mn) per Thm 7.38 in [1].

Difficulty: 7, time spent: 90 minutes

3. **Chapter 8 Problem 2:**



2. A store trying to analyze the behavior of its customers will often maintain a two-dimensional array A, where the rows correspond to its customers and the columns correspond to the products it sells. The entry $A[i, j]$ specifies the quantity of product j that has been purchased by customer i.

Here's a tiny example of such an array A.

	liquid detergent	beer	diapers	cat litter
Raj	0	6	0	3
Alanis	2	3	0	0
Chelsea	0	0	0	7

One thing that a store might want to do with this data is the following. Let us say that a subset S of the customers is ***diverse*** if no two of the of the customers in S have ever bought the same product (i.e., for each product, at most one of the customers in S has ever bought it). A diverse set of customers can be useful, for example, as a target pool for market research.

We can now define the Diverse Subset Problem as follows: Given an $m \times n$ array A as defined above, and a number $k \leq m$, is there a subset of at least k of customers that is ***diverse***?

Show that Diverse Subset is NP-complete.

Proof: We can see that *Diverse Subset* is in NP since if we are given a certificate instance of *Diverse Subset*, we can verify that no customer purchased duplicate items, so *Diverse Subset* $\in NP$.

Looking closely at this problem we can see that it is a special case of the *Set Packing Problem* which is $NP-$complete according to Thm 8.7 in [1]. Therefore, we show that *Set Packing Problem* $\leq_p$ *Diverse Subset*.
Proof: Suppose we have an instance of the *Set Packing Problem* such that our collection of sets $U = \{S_1, S_2, ..., S_m\}$. For each S_i, $i = 1, ..., m$, let $S_i[k]$ denote the k^{th} element of S_i where $k = 1, ..., n$ and the value of $S_i[k] = j$, where $j \geq 0$. We claim that $\exists$ a collection of at least k of these sets that do not intersect if and only if, there exists a diverse subset of at least k customers.
Proof ($\rightarrow$): Suppose that we have at least k sets of the collection U that do not intersect. Then we can represent this set as

$$S = \{S_l \mid S_l[i] \neq S_m[i], \forall m \neq l, i = 1, ...n\}$$

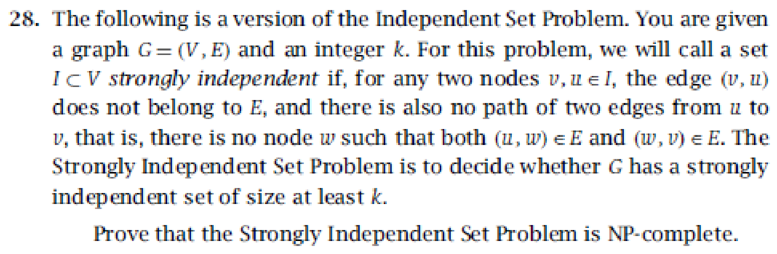
We can now let the elements of U represent each customer, so we can build our original Array A as

$$A = \begin{pmatrix} S_1 \\ \vdots \\ S_m \end{pmatrix}, \text{ where } S_i = \text{ customer } i.$$

We can then see that our set S having no intersecting elements is the same as the having a diverse subset since no two customers (S_i's) have the same $S_i[k]$ so then S is a diverse subset of size at least k.
Proof ($\leftarrow$): The proof holds in the reverse direction beginning with defining the rows (customers) of the matrix A as the collection U and defining the intersection property as shown above.
Therefore, since *Diverse Subset* $\in NP$ and *Set Packing* $\leq_p$ *Diverse Subset*, *Diverse Subset* $\in NP$-complete.

Difficulty: 4, time spent: 30 minutes

4. **Chapter 8 Problem 28:**



28. The following is a version of the Independent Set Problem. You are given a graph $G = (V, E)$ and an integer k. For this problem, we will call a set $I \subset V$ *strongly independent* if, for any two nodes $v, u \in I$, the edge (v, u) does not belong to E, and there is also no path of two edges from u to v, that is, there is no node w such that both $(u, w) \in E$ and $(w, v) \in E$. The Strongly Independent Set Problem is to decide whether G has a strongly independent set of size at least k.

    Prove that the Strongly Independent Set Problem is NP-complete.

Proof: Since a Strongly Independent Set is also an Independent Set, we know that it is in NP, but given a Graph $G = (V, E)$ and a certificate instance of the *Strong Independent Set* problem, T, we can very its Strong Independence by checking that any path between two nodes within T has length at least 3. Thus, *Strong Independent Set* $\in NP$.

Now, we must pick an NP-complete problem and show that it poly-time reduces to *Strong Independent Set*. Naturally, we show that *Independent Set* $\leq_p$ *Strong Independent Set*.
Proof: Suppose we have a graph $G = (V, E)$ and an instance of *Independent Set*, $S \subseteq V$. Now, we define a new graph $G' = (V', E')$ defined as follows: For every pair of nodes u, v in our instance of S that have a path between them of length two, i.e. $\exists w \in V$ s.t. $(u, w), (w, v) \in E$, we insert a new node r into V' and connect it between u and w s.t. $(u, w) \in E$ is now $(u, r), (r, w) \in E'$. Now, taking our original *Independent Set* instance S on new the vertex set V' produces an instance of *Strongly Independent Set* $S' \subseteq V'$ since now all of the nodes in S' have a path of at least length 3 between them by our construction. Thus, *Independent Set* $\leq_p$ *Strongly Independent Set*.
Therefore, since *Strongly Independent Set* $\in NP$ and *Independent Set* $\leq_p$ *Strongly Independent Set*, then *Strongly Independent Set* $\in NP$-complete.

Difficulty: 5, time spent: 60 minutes

5. **References**

[1] J. Kleinberg and E. Tardos, Algorithm Design. Pearson, 2006.