15-750: Algorithms in the Real World **CMU, Spring 2023**

HW #4 (out Tuesday March 14, 2023) **Due:** Tuesday March 28, 2023

Please limit yourself to groups of two (or three) people. We adhere to a "whiteboard" collaboration policy: you should discuss in your group on whiteboards (or equivalents) but then should go away and write down your solutions yourself. You must not share written work. **You must write down the names of the person(s) with whom you collaborate.** In general, you should solve the homework problems using only to material we refer you to (or put on the course page), and not books or other online resources. If you have reason to use any resources we point you to (except the course notes), please cite them.

A word of advice: please try to solve the problem by yourself before working with your group. That is the way to improve your problem-solving skills.

Homeworks will be due at 11:59pm on the due date on `gradescope`. Corrections and changes will appear on the course webpage and on Piazza; please check them regularly.

Problems

In these problems, you may cite the fact that there is a polynomial-time algorithm for: (i) Max-Flow (and even its Min-Cost version); (ii) Max-st-Cut; (iii) Linear Programming. Moreover, for (Min-Cost) Max-Flow, you may use the fact that the capacities need not be integers, but if they *are* all integers, then the algorithm will return an optimal solution in which all flow amounts are integers.

1. This problem concerns *fairness*. Your company has n employees. The company has m tasks to handle, each of which is done by a task-force. The task-force for task i is a nonempty subset $S_i \subseteq \{1, 2, \ldots, n\}$ of the employees. These task-force sets are the input to the problem.

   Each task-force must be led by some member $j \in S_i$. Since being the leader of a task-force is a burdensome job, you want the leadership roles to be divided as fairly as possible. The fairness criterion is the following: Say that person p is in some k of the task-force sets, and these have sizes $n_1, n_2, \ldots, n_k$. It seems reasonable that person p should have to be the leader for $\frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k}$ of these task-forces. Of course this number may not be an integer, so let's round it up to an integer. This quantity $\lceil \frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k} \rceil$ is called that person's *fair cost*. A *fair solution* is an assignment of a leader for each task-force so that person is the leader for no more task-forces than their fair cost.

   Say the 1st set has Alice and Charlie, the 2nd has Alice, Bob, and Deniz, the 3rd has Charlie and Deniz, and the 4th has just Deniz. Alice's fair cost would be $\lceil 1/2 + 1/3 \rceil = 1$. So Alice leading both her sets would not be fair. But the solutions "1. Alice, 2. Deniz, 3. Charlie, 4. Deniz", or "1. Alice, 2. Bob, 3. Deniz, 4. Deniz", or "1. Alice, 2. Bob, 3. Charlie, 4. Deniz" are all fair.

   Note that it is not at all obvious that there always *exists* a fair solution. Nevertheless, describe and analyze the correctness and running time of a polynomial-time algorithm that, given any instance $S_1, S_2, \ldots, S_m$, *always* finds a fair solution. After describing your algorithm, first illustrate how it works on the above small example before explaining the general analysis.

2. This problem is about "denoising" images. Suppose we are given a black-and-white image where some of the pixels have been corrupted. We want to remove these corruptions. Beforehand we have no particular idea which pixels are correct and which are corrupted. So we

hypothesize that: (a) neighboring pixels are likely to be identical; (b) the image has a small number of corruptions, so that most pixels are correct.

With these hypotheses in mind, here is a concrete strategy: We view the image as an $m \times n$ rectangular grid graph $G = (V, E)$, with each node $p \in V$ representing a pixel (i, j) (with $1 \le i \le m$, $1 \le j \le n$), and two pixels $p = (i, j)$ and $p' = (i', j')$ having an edge between them if $|i - i'| + |j - j'| = 1$. For each pixel p we are given an original color $c_p \in \{0, 1\}$. We want to find a color $x_p \in \{0, 1\}$ for each pixel p so as to minimize

$$\text{obj} := K \sum_p |x_p - c_p| + \sum_{(p,p') \in E} |x_p - x_{p'}|.$$

Here $K > 0$ is a parameter that we may choose, which tunes how attentive we are to matching the input.

Describe and analyze a polynomial-time algorithm that takes as input an image and a parameter K, and outputs a "denoised" image (x_p values) that minimizes "obj".

3. This problem is about an algorithmic subroutine that plays a key role in Traveling Salesperson algorithms.

   Given a graph $G = (V, E)$, a *cycle-fitting* is a collection of cycles $C_1, C_2, \dots, C_t$ in the graph such that every vertex of the graph lies in exactly one of these cycles. (Put another way, it is a subset of edges such that the degree of every vertex is exactly 2.)

   (a) Suppose that G is bipartite with the same number of vertices on both sides; i.e., V is partitioned as $L \sqcup R$, all edges go between L and R, and $|L| = |R|$. Describe and analyze a polynomial-time algorithm that correctly decides whether or not G has a cycle-fitting.

   (b) Suppose each edge e of G has a given cost c_e. Explain how to modify your previous solution to get an efficient algorithm that finds the cycle-fitting of least total cost (if one exists).

   (c) Now suppose that G is not necessarily bipartite. Describe and analyze a polynomial-time algorithm to find a min-cost cycle-fitting in G (if one exists). For this part, we slightly change the definition of a cycle-fitting: here, a "cycle" is allowed to be "two copies of an edge" (which arguably can be thought of as a cycle of length 2). (Hint: transform G into a bipartite graph H in some natural way, and think about matchings in H.)

4. (This problem is intentionally kind of tedious, but it is meant to illustrate the huge power of Linear Programming for modeling.)

   Your company is building new economic stoves that reduce the amount of greenhouse gas emissions. You have k different kinds of stoves you can build at your factory. There are L different basic components that are assembled together to build these stoves, and these are numbered $1, 2, \dots, L$. For each $i \in \{1, 2, \dots, k\}$ and $j \in \{1, 2, \dots, L\}$, stove type i uses up $A_{ij} \ge 0$ amounts of component j. Moreover it requires t_i person-hours to assemble, and you have a total of T person-hours available each year.

   There are other contractual obligations: (a) you must build at least d_i units of stove i; (b) the number of stoves of type 1 and 2 built can differ by at most 2% of the total number of stoves built by your company; (c) if your company uses more than M units of component 1 then it must install an air filter which costs F.

Finally, the cost per unit of each component j is c_j, and the assembly cost for each stove of type i is λ_i. (This need not be related to the number of person hours t_i.) The sale price of stove i is S_i.

Describe a polynomial-time solution for computing the number of stoves of each type needed to maximize profit. If you explain the meaning of each variable and constraint in an LP (hint), and you also explain the goal of each (hint) LP you solve, then it is fine to only have a couple of sentences more about how/why you collate your answers to get the final solution.

Note: even though it doesn't quite make sense, assume in this problem that you are allowed to build and sell "fractional" (non-integer) amount of stoves. (You might imagine that the numbers involved are in the thousands or more, so that rounding solutions to integers has a negligible effect on optimality.)