

1. Occurrences of Cyclic Sub-Strings

Let T be a string of size n , and S be a string of size m . You have learned how to count how many times S appears in T in $O(n)$ time. In this problem, we are going to make this more interesting. We want you to count how many substrings of T are cyclical isomorphic to S . Two strings are called *cyclical isomorphic* if one can cyclically rotate one string to get the other one. For instance, “reet” is cyclical isomorphic to “tree”, but “rete” is not. “ringst” is cyclical isomorphic to “string”, but “srtng” is not. A string is also cyclical isomorphic to itself. For simplicity, you can assume that all m cyclical isomorphic strings of S are distinct.

Example 1: $T = \text{“banana”}$, $S = \text{“an”}$. The answer is 4: an, na, an, na.

Example 2: $T = \text{“Mississippi”}$, $S = \text{“sis”}$. The answer is 5: iss, ssi, sis, iss, ssi.

The naïve algorithm, where you consider all strings isomorphic to S , count their occurrences, and add them up, takes $O(nm)$ time. We want you to find an $O(n + m)$ algorithm.

(Hint: All of the cyclical isomorphic strings of S are substrings of one particular string. Can you figure out what that string is?)

2. Domino Tiling

You get a job paving the floors for a grand hall. You can think of the floor plan as an $m \times n$ grid where each cell is a square of shape 1×1 . You can use two kinds of tiles: 1×2 tiles and 1×1 tiles. You notice that paving a 1×2 tile is cheaper than two separate 1×1 tiles, so you aim at **maximizing the number of 1×2 tiles used**.

In the hall, there are pillars and statues occupying some of the grid cells. No tile should be placed on these grid cells. The other cells must be fully covered. You are provided an input of a $m \times n$ matrix $F[m][n]$. each entry is either 0 (unoccupied) or 1 (occupied). Provide a network flow algorithm that, in polynomial time, outputs a paving solution that uses the maximum number of 1×2 tiles.

Explain your ideas on how to construct a network flow problem based on the settings. Show the relation between the flow problem and the original paving problem. Write down which algorithm you then used to solve the flow problem. You **do not** need to give detailed pseudocode, but you **do** need to show why your flow problem is equivalent to the original problem.

3. Fairness

There is much interest in algorithmic fairness nowadays, and here’s a basic problem. Your company has n employees. You all have m tasks to handle, each of which is done by a task-force, which is some given (non-empty) $S_i \subseteq \{1, 2, \dots, n\}$ of the employees. Each task-force must be led by some member $j \in S_i$. You want the leadership to be divided as fairly as possible. Your task in this problem is to give an algorithm to do this efficiently.

The fairness criterion is the following: Say that person p is in some k of the sets, which have sizes n_1, n_2, \dots, n_k , respectively. Person p should really have to be the leader for $\frac{1}{n_1} + \frac{1}{n_2} + \dots + \frac{1}{n_k}$ of these task-forces. Of course this number may not be an integer, so let’s round it up to an integer. This quantity $\lceil \frac{1}{n_1} + \frac{1}{n_2} + \dots + \frac{1}{n_k} \rceil$ is called that person’s *fair cost*. A *fair solution* is an assignment of leadership positions to members of these task-forces so that that each person is leader for no more task-forces than their fair cost.

Say the first set has Alice and Charlie, the second has Alice, Bob, and Eve, the third has Charlie and Eve, and the fourth has just Eve. 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 (of respective group leaders) Alice, Eve, Charlie, Eve, or Alice, Bob, Eve, Eve, or Alice, Bob, Charlie, Eve, are all fair.

Note that it is not *a priori* clear there even *exists* a fair solution. Give a polynomial-time algorithm that, given any instance S_1, S_2, \dots, S_m , *always* finds a fair solution, and explain why your algorithm is correct. (This will also show that there always exists a fair solution.)

Again, first illustrate your solution on the example above, and then explain the general solution.

(Hint: first, try to show that there *exists* a fair solution. Can you construct a maximum flow that is not necessarily integral?)

4. Circuit Board Wiring.

There are n terminals on a circuit board. Each terminal is either positive or negative. A wire will be attached to each terminal. It must be the case that for every pair of terminals of opposite polarity the two wires must be long enough so that they can be made to touch.

E.g., if there's a positive terminal at point $a = (0, 0)$, and a negative terminal at $b = (0, 10)$, and another negative terminal at $c = (10, 0)$ then one solution is to put a wires of length 5 at all three terminals. Or to put a wire of length 10 at a and length zero at the others. Or, for any $x \in [0, 10]$, wires of length $\geq 10 - x$ at a , and length $\geq x$ at both b and c .

So given the locations of the terminals (and their polarities), the problem is to compute the shortest total length of wires needed to satisfy the stated requirement.

- (a) Consider the following specific instance of the problem. All the terminals are along the x -axis. There are positive terminals at 0, 110, 111, 112. And there are negative terminals at 99 and 100. What's the optimal solution to the problem? (You don't need to prove your solution is optimal.)
- (b) We will show that, surprisingly, it is the *dual* of this problem that can be converted to maximum flow!
 - i. Formulate this problem as an LP.
 - ii. Write the dual of the LP in part (i).
 - iii. Show that the dual problem of part (ii) can be formulated as a min-cost flow problem. (Hint: think about bipartite matching.)
- (c) **Extra credit:** You may code up a solver for the LP for **extra credit**. The following [python notebook](#) in Google Colab contains starter code which generates and plots synthetic data (terminals and polarity).

We recommend using the `cvxopt` library. The documentation can be found [here](#) and an example of how to use it can be found [here](#). We have also provided some starter code if you choose to use `cvxopt`.

Give the result for the minimum total length of wires needed for the generated data. Please write down the numerical answer here, and **include your code at the end of your homework submission**.