15-451/651 Algorithms, Spring 2017

Homework #5 Due: March 28, 2017

This HW has three regular problems, one programming problem, and one bonus problem. All problems on written HWs are to be done individually, no collaboration is allowed. Solutions to the three written problems should be submitted as a single PDF file using gradescope, with the answer to each problem starting on a new page.

Submission instructions for the programming problem will be posted on Piazza.

1. Pushing Pawns in Parallel. You’re given an $n \times n$ grid of cells. Some cells (not in the left and right columns) are blocked. The left column contains $n$ chess pawns. These pawns don’t move like regular pawns. They still only move one cell in some direction, but now move in parallel. So a “move” consists of selecting a subset of the pawns and a direction for each of them to move (N, S, E or W). The non-selected pawns do not move. The parallel move is okay if all of the destination cells of all of the moving pawns (1) have just one pawn as its destination (2) are not occupied by a non-moving pawn and (3) are not blocked. So specifically a pawn can move to a cell simultaneously vacated by another moving pawn. For example, the first board below can become the second board in one move.

Describe an algorithm which takes as input the $n \times n$ array of blocked and non-blocked cells, and determines if it’s possible to move all the pawns from the left column to the right column. If it is possible, your algorithm should output the minimum number of moves required to do this. The running time of your algorithm should be polynomial in $n$. (We use numbers on the pawns only for illustration purposes, the actual pawns are all identical, so it does not matter which pawns end up where on the right column.)

Hint 1: To get started, devise an efficient test that takes the board as input, along with a move bound $M$, and determines if it’s possible to complete the task in $M$ or fewer moves. How would you use this test to solve the problem?

Hint 2: Try to apply one or more of the algorithms we’ve been covering in lecture.

2. A Fair Carpool. The $n$ employees of the KäärPööl ride-sharing company sometimes carpool to work together in a single large beat-up van. Say there are $m$ days, and $S_i$ is the set of people that carpool together on day $i$. For each set $S_i$, one of the people $j \in S_i$ must be chosen to be the driver that day. Since people would rather not drive, they want the work of driving to be divided as fairly as possible. Your task in this problem is to give an algorithm to do this efficiently.

---

1 Any relation to actual companies of this name is purely coincidental.
The fairness criterion is the following: Say that person $p$ is in some $k$ of the sets, which have sizes $n_1, n_2, \ldots, n_k$, respectively. Person $p$ should really have to drive $\frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k}$ times, because this is the amount of resource that this person effectively uses. 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 their *fair cost*. A fair solution is a schedule for who drives on which day, such that each person drives no more than their fair cost.

For example, say that on day 1, Aram and Bob carpool together, and on day 2, Aram, Celia, and Dorothy carpool together. Aram’s fair cost would be $\lceil 1/2 + 1/3 \rceil = 1$. So Aram driving both days would not be fair. Any solution except that one is fair.

Give a polynomial-time algorithm that, given $S_1, S_2, \ldots, S_m$, computes a fair solution. This will also show that there always exists a fair solution.

3. **What are Widgets, Anyways?** You are in charge of the supply chain for Widgets-r-Us, where you have to find ways to transport materials from their storage warehouses to the factories, subject to capacity constraints on the transportation network. You model it as a directed graph $G = (V, E)$, where each directed edge $e \in E$ represents a one-way road. (Roads capable of carrying traffic in both directions can be thought of as two one-way roads.)

There are $k$ “warehouse” vertices, and $k$ kinds of materials, where the $i^{th}$ material is originally located at the warehouse vertex $s_i \in V$, and the total amount of material $i$ is $D_i$. Each road (directed edge) $e$ has a capacity $u_e$, such that the total amount of material (of all kinds) sent over this road must be at most $u_e$. You may assume that all quantities given as input in this problem are non-negative integers.

You also have $\ell$ “factory” vertices. There are $\ell$ factories producing widgets (the $j^{th}$ factory is located at the $j^{th}$ factory vertex $f_j \in V$). The factory $f_j$ has a request vector $r_j = (r_{1j}, r_{2j}, \ldots, r_{kj})$, such that to produce one unit of widget, it requires $r_{ij}$ amounts of material $i$ for every $i \in \{1, \ldots, k\}$.

(a) Write an LP to figure out how to transport the material from their warehouses to the factories (respecting the road capacity constraints), to maximize the total amount of widgets produced, subject to these constraints. (It is OK if you produce fractional amounts of widgets.)

(b) You find out that widgets produced at different factories sell for different amounts of money: the price per unit of widget produced at factory $j$ is $p_j$. You want to maximize your revenue. Change the LP from the previous part to handle this.

(c) You are informed that two of the roads $e_1 = (u, v)$ and $e_2 = (v, u)$ are special: they represent the two directions of traffic over a bridge. For structural reasons, you have the balance requirement that the absolute value of the difference between the amount of material send in the two directions can be at most $\delta \in \mathbb{Z}_{\geq 0}$. How can you add such a constraint to your LP.

---

Nodes like $s_i$ and $f_j$ can have both incoming and outgoing arcs, and the route taking material $i$ from the warehouse $s_i$ to some factory $f_j$ is allowed to pass through some other warehouse vertex $s_{i'}$ or some other factory vertex $f_{j'}$. You may assume that all the $k + \ell$ warehouse and factory vertices are distinct.
4. **Devious Dominos.** In this programming problem you’re given a rectangular board of square cells. Some of the cells are blocked. You are to compute the maximum number of dominos (1 × 2 tiles) that can be placed on the board. Each domino will be in either a vertical or horizontal orientation, and is placed on two neighboring squares, neither of which is blocked. The time limit is 10 seconds (60 seconds for python).

**Input:**

The first line contains two space-separated integers: \( r \) and \( c \). The next \( r \) lines are strings of length \( c \) comprised of the characters “.” and “x”. The “x” characters denote cells of the board that are blocked. \( 1 \leq r, c \leq 30 \).

**Output:**

Print the maximum number of dominos that can be placed on the board satisfying these constraints. The format of the output is illustrated below.

Input 1:

```
2 3
...
...
```

Output 1:

```
3 dominos
```

Input 2:

```
5 4
.xx.
xx.x
x...
xx.x
.xx.
```

Output 2:

```
1 domino
```

Input 3:

```
2 5
..xx.
.....
```

Output 3:

```
4 dominos
```