CS3510 Design & Analysis of Algorithms Section A

Homework 3 Solutions

Released: 7pm, Wednesday Nov 8, 2017

This homework has a total of 4 problems on 4 pages. Solutions should be submitted to GradeScope before 3:00pm on Monday Nov 6.

The problem set is marked out of 20, you can earn up to $21 = 1 + 6 + 5 + 4 + 5$ points.

If you choose not to submit a typed write-up, please write neat and legibly.

Collaboration is allowed/encouraged on problems, however each student must independently complete their own write-up, and list all collaborators. No credit will be given to solutions obtained verbatim from the Internet or other sources.

Updates to this homework since version 0 **are**:

- Question 2 (changed in v2), renamed the goal form of the linear program as the text book already defines standard forms differently.
- Question 3 (changed in v1), capacity of edge $a \to b$ should be 1.
- Question 3 (changed in v1), clarified that we're considering the variant of Ford-Fulkerson algorithm where as much flow is pushed along a path as the capacities allow.
- Question 3 (changed in v3), changed the number of iterations to the number of times that the flow is modified. This makes it explicit that the last step (where no $s \to t$ path is found in the residual graph, and we terminate with a cut) is **not counted**.
- Question 4 (a) (changed in v3): clarified that it's asking for a proof (that works for any graph), and explicitly allows Hall's theorem.
- Question 4 (b) (changed in v2), runtime of our algorithm changed to $O(n^5)$ (from $O(n^6)$).
- Question 4 (b) (changed in v3): clarified that part b) is easier than part a).

0. **[1 point, only if all parts are completed]**

   (a) Submit your homework to Gradescope.

   (b) Student id is the same as on T-square: it's the one with alphabet + digits, NOT the 9 digit number from student card.

   (c) Pages for each question are separated correctly.

   (d) Words on the scan are clearly readable.

1. (6 points) Formulating Linear Programs

   The goal of this question is to formulate problems as linear programs, not solve them.

   (a) (2 points) Recall the absolute value function $|\cdot|$:

   $$|y| := \max\{y, -y\}.$$

   Show that the constraint

   $$x_2 \geq |x_1|.$$

   is equivalent to two linear inequalities.

   SOLUTION:
   It's equivalent to

   $$x_2 \geq x_1$$
   $$x_2 \geq -x_1$$

   (b) (4 points) Consider the system of linear equations:

   $$a + b = 123$$
   $$4a + 3b = 234$$
   $$5a + 4b = 345$$

   This system of equations is clearly not solvable: adding together the first two equations gives $5a + 4b = 357$, which is different than the value given in the third equation.
   Instead, we will approximately solve the above problem by minimizing the total absolute value error of the first two equations, aka.

   $$|a + b - 123| + |4a + 3b - 234|,$$

   ($|\cdot|$ is the absolute value function as defined in part a)), while ensuring the third one,

   $$5a + 4b = 345.$$

   Formulate this as a linear program. Note that the absolute value function is not a linear function.

   SOLUTION:

$$\begin{aligned} \min_{a,b,x,y} \quad & x + y \\ \text{subject to:} \quad & x \geq a + b - 123 \\ & x \geq -a - b + 123 \\ & y \geq 4a + 3b - 234 \\ & y \geq -4a - 3b + 234 \end{aligned}$$

2. (5 points) Converging Linear Programs to ~~Standard~~ Garage Forms

It is often convenient to represent linear programs in 'standard forms', which have fewer types of constraints. One example of such a form is on page 210 of the text book, and sadly, different text books define such forms differently.

For this problem, we consider a form of linear programs which we'll term the **garage form**. For variables $\{x_1, x_2, \ldots, x_n\}$, the garage form has:

$$\begin{array}{ll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{subject to} & a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n \leq b_1 \\ & a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n \leq b_2 \\ & \ldots \\ & a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n \leq b_m \\ & x_j \geq 0 \qquad \text{for all } 1 \leq j \leq n \\ & 5a + 4b = 345 \end{array}$$

Specifically every variable x_j is non-negative, and every constraint is of the form of

$$\sum_j a_{i,j}x_j \leq b_i.$$

Convert the linear program:

$$\begin{array}{ll} \text{minimize} & a - 10c \\ \text{subject to} & a + 5b - c = 10 \\ & 5b - 4d \geq -2 \end{array}$$

the garage form as specified above.

Note that any number z can be written as the difference of two non-negative numbers. That is, for any z there are z_+ and z_- such that

$$z = z_+ - z_-$$

and

$$z_+, z_- \geq 0.$$

SOLUTION:
We turn each of the variables into two copies, forming

$$a_-, a_+, b_-, b_+, c_-, c_+, d_-, d_+.$$

Then we can solve for

$$\begin{aligned}
\textbf{maximize} \quad & -a_+ + a_- + 10c_+ - 10c_- \\
\textbf{subeject to:} \quad & a_+ - a_- + 5b_+ - 5b_- - c_+ + c_- \leq 10 \\
& -a_+ + a_- - 5b_+ + 5b_- + c_+ - c_- \leq -10 \\
& -5b_+ + 5b_- + 4d_+ - 4d_- \leq 2 \\
& a_-, a_+, b_-, b_+, c_-, c_+, d_-, d_+ \geq 0
\end{aligned}$$

3. (4 points) Slow case of the Ford-Fulkerson algorithm.

   In class we mentioned that the Ford-Fulkerson algorithm takes $O(F)$ iterations, where F is the value of the maximum flow. Note that the value of F can be much larger than n and m.

   In this problem we will see that this can happen, even when we push as much flow as possible along the $s \to t$ path found in the residual graph.

   Let Z be an arbitrarily large integer. Consider the directed network on 4 vertices, s, a, b, and t with (capacitated) edges

   - $s \to a$ with capacity Z.
   - $s \to b$ with capacity Z.
   - $a \to t$ with capacity Z.
   - $b \to t$ with capacity Z.
   - $a \to b$ with capacity 1.

   The maximum flow is $2Z$: sending Z units each along the two paths $s \to a \to t$ and $s \to b \to t$. Furthermore, if we return $s \to a \to t$ as our first augmenting path, we will route Z units on it, so the Ford-Fulkerson algorithm can potentially terminate in 2 iterations (plus 1 more to output the minimum cut).

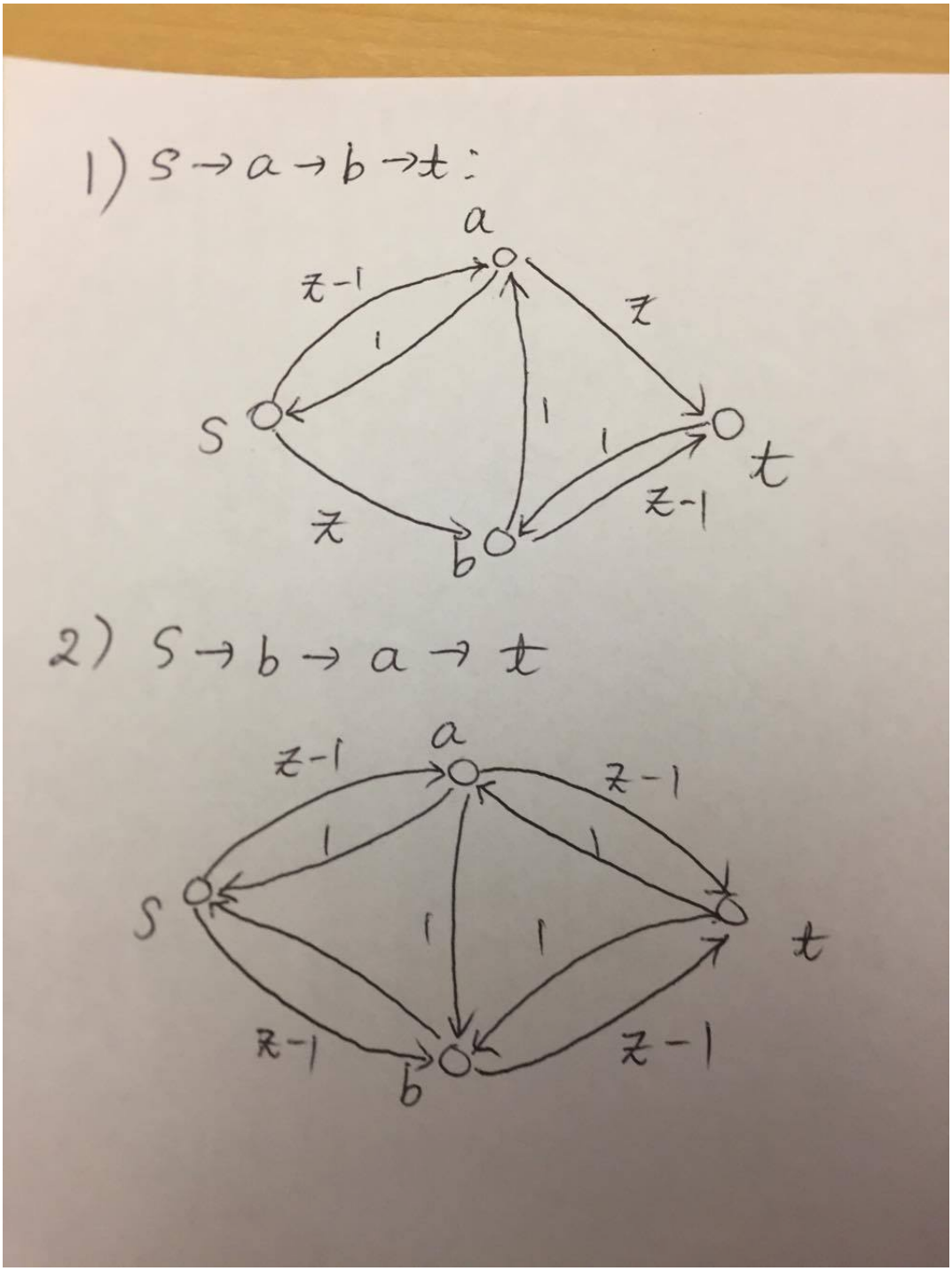
   Instead, suppose our path finding algorithm in the residual graphs always finds a path involving $a \to b$ or $b \to a$ if it exists.

   Draw the first 2 residual graphs, and infer from them (with a brief justification) the number of iterations (in terms of Z) that the Ford-Fulkerson algorithm may take on this graph.

   SOLUTION:
   First residual graph has 1 units on $s \to a$, $a \to b$, $b \to t$. So the $a \to b$ arc in the residual graph is now $b \to a$.

   Second residual graph has 1 units on $s \to a$, $s \to b$, $a \to t$, $b \to t$. So the arc is $a \to b$, and everything else are bi-directional.

   Residual graphs are in the figure below.



This process will then repeated itself Z times before the $s \to a$ and $s \to b$ edges get saturated. Thus at total of $2Z$ flow pushing steps, and 1 more at the end to terminate.

4. (5 points) Decomposing d-regular bipartite graphs

   A bipartite graph $G = (V, E)$ is one where V can be partitioned into A and B such that all edges are between some vertex in A and some vertex in B (there are no edges contained within A or B).

   A graph is d-regular if all vertices have degree d. It can be shown that if a bipartite graph on n vertices is d-regular, then the two bipartitions must have $n/2$ vertices each.

   A matching is a set of edges that do not share vertices. On this graph, a perfect matching is one that involves all vertices, or equivalently, has $n/2$ edges. In this problem we will show that a d-regular bipartite graph can be decomposed into d perfect matchings.

   (a) (3 points) Prove using either the maxflow / mincut theorem, Hall's theorem, or some other method of your choice, that a d-regular bipartite graph will always have a perfect matching.

   SOLUTION:

   The proof is by contradiction. Suppose not, then by Hall's theorem there exists a set $S \subseteq A$ that's adjacent to fewer than $|S|$ vertices.

   The number of edges leaving S is $d|S|$. For these edges to be distributed to fewer than $|S|$ neighbors, some vertex must have degree at least $d+1$. A contradiction with the assumption that the graph is d-regular.

   (b) (2 points) (**note:** this is the easier part of the problem. You may assume part a) in your solution as well) Devise an algorithm that decomposes a d-regular bipartite graph into d perfect matchings in $O(n^{200})$ time. The algorithm in our solution runs in $O(n^5)$ time.

   SOLUTION:

   Consider an algorithm that:

   i. Find a perfect matching,

   ii. remove it, repeat on the resulting graph.

   Since the perfect matching has degree 1 at each vertex, removing it results in a graph with degrees $d - 1$ at the vertices. So by Part a) we always can find such a matching.

   As $d \leq n$, and perfect matchings take up to $O(n^2 m) = O(n^5)$ to find, the runtime of this is $O(n \cdot n^4)$, which is $O(n^5)$.