COMS20010 — Problem sheet 7

You don't have to finish the problem sheets before the class — focus on understanding the material the problem sheet is based on. If working on the problem sheet is the best way of doing that, for you, then that's what you should do, but don't be afraid to spend the time going back over the quiz and videos instead. (Then come back to the sheet when you need to revise for the exam!) I'll make solutions available shortly after the problem class. As with the Blackboard quizzes, question difficulty is denoted as follows:

- ⋆ You'll need to understand facts from the lecture notes.
- ⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in unfamiliar situations.
- ⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes and also move a bit beyond them, e.g. by seeing how to modify an algorithm.
- ⋆⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in a way that requires significant creativity. You should expect to spend at least 5–10 minutes thinking about the question before you see how to answer it, and maybe far longer. At most 10% of marks in the exam will be from questions set at this level.
- ⋆⋆⋆⋆⋆ These questions are harder than anything that will appear on the exam, and are intended for the strongest students to test themselves. It's worth trying them if you're interested in doing an algorithms-based project next year — whether you manage them or not, if you enjoy thinking about them then it would be a good fit.

If you think there is an error in the sheet, please post to the unit Team — if you're right then it's much better for everyone to know about the problem, and if you're wrong then there are probably other people confused about the same thing.

This problem sheet covers week 7, focusing on linear programming and network flows.

1. (a) [⋆⋆] Convert the following linear programming problem into standard form.

$$\begin{aligned} 4x - 3y + z &\to \min \text{ subject to} \\ x + y &\leq 4, \\ 3x - 2y - z &\geq 2, \\ x,\, y &\geq 0. \end{aligned}$$

   (b) [⋆⋆⋆] Some linear programming algorithms prefer to take their input in another form, known as *slack form*. In slack form, we are given a linear objective function $f\colon \mathbb{R}^n \to \mathbb{R}$, an $m \times n$ matrix A, and an m-dimensional vector $\vec{b} \in \mathbb{R}^m$. The desired output is a vector $\vec{x} \in \mathbb{R}^n$ maximising $f(\vec{x})$ subject to $A\vec{x} = \vec{b}$ and $\vec{x} \geq 0$; thus slack form is the same as standard form, except that the upper bound constraint $A\vec{x} \leq \vec{b}$ is replaced by the equality constraint $A\vec{x} = \vec{b}$.

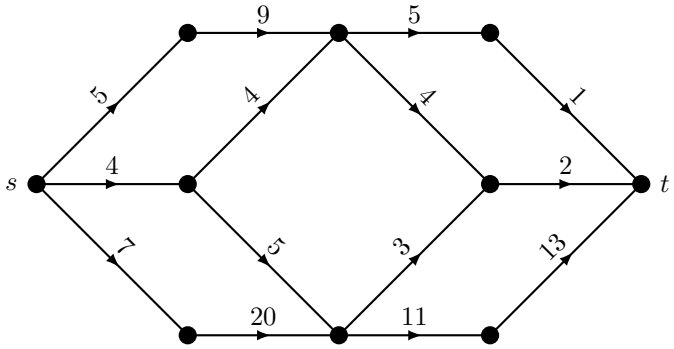
   Put the linear programming problem from part (a) into slack form, and explain how to apply your method to convert any problem in standard form into slack form. (**Hint:** You may find it useful to add new variables.)

2. Are the following statements true or false? For each one, give either a short explanation (if it's true) or a counterexample (if it's false).

   (a) [⋆⋆] If f is a maximum flow in a flow network (G, c, s, t), then $f(e) = c(e)$ for all edges e incident to the source s.

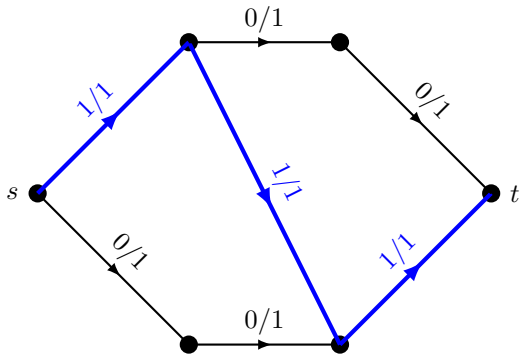
(b) [★★★] If (A, B) is a minimum cut in a flow network (G, c, s, t), and we add 1 to the capacity of every edge, then (A, B) is still a minimum cut.

3. Consider the following flow network with capacities indicated:



(a) [★★] Run the Ford-Fulkerson algorithm on this flow network to find a maximum flow, showing how the residual network evolves as the current flow changes.

(b) [★★] Now consider the following flow network with a flow.



Continue running the Ford-Fulkerson algorithm on this flow network to find a maximum flow, showing how the residual network evolves as the current flow changes.

Hint: How are reverse edges useful here?

4. [★★★★] John thinks backward edges sound unnecessarily complicated, and so he decides to implement the Ford-Fulkerson algorithm using only forward edges. (In other words, his algorithm only chooses augmenting paths that increase flow along every edge, and terminates when no such paths exist.) He reasons that while it won't always give a maximum flow, it will at least give a decent approximation. Show that John is wrong by giving an example on which his algorithm may output a flow whose value is at most one hundredth of that of a maximum flow. You may make whatever assumptions you like about how John's algorithm chooses its augmenting paths. (**Hint**: Your example may have too many vertices to draw in full.)

5. (a) [★★★] Prove that the approximation algorithm for vertex cover given in lectures works, outputting a vertex cover with weight at most twice the minimum possible.

(b) [★★★] In the *weighted* vertex cover problem, we are given a graph $G = (V, E)$ and a weight function $w \colon V \to \mathbb{N}$, and we must output a vertex cover with minimum total weight. Adapt the unweighted algorithm given in lectures to solve this problem approximately, and prove it works.

6. Consider the folowing linear programming problem:

$$\begin{aligned} 7x + 8y &\to \max \text{ subject to} \\ -0.5x + 2 &\leq y \\ x - 15 &\leq y \\ -0.2x + 25 &\geq y \\ 0.8x + 16 &\geq y \\ x, y &\geq 0 \end{aligned}$$

   (a) [⋆⋆] Convert this problem into standard form.

   (b) [⋆⋆] Plot these constraints. What do they look like geometrically?

   (c) [⋆⋆] Evaluate the objective function at each vertex to determine the optimal values of x and y for this problem.

7. (a) [⋆⋆] Consider the following linear programming problem:

$$\begin{aligned} 3(x_1 + x_2 + x_3 + x_4 + x_5 + x_6) &\to \max \text{ subject to} \\ 5x_1 + 4x_2 + 2x_3 + x_4 &\leq 10, \\ x_3 + 2x_4 + 4x_5 + 5x_6 &\leq 25, \\ x_1, \ldots, x_6 &\geq 0. \end{aligned}$$

   By subtracting the left-hand sides of both constraints from the objective function, prove that the optimal solution has value at most 35.

   (b) [⋆⋆⋆] Instead of subtracting the left-hand sides of both constraints once each, we could have subtracted a constant multiple a_1 of $5x_1+4x_2+2x_3+x_4$ and a constant multiple a_2 of $x_3+2x_4+4x_5+5x_6$ and then kept the rest of the argument the same, yielding a different and potentially better upper bound. Formulate the problem of trying to find the best possible upper bound with this method as a two-variable linear program in a_1 and a_2.

   (c) [⋆⋆] Solve this new linear program (e.g. using the fact that the solution must be at a corner of the feasible polytope) to prove that the original problem's optimal solution has value at most 33.75.
   Plugging in $(\frac{3}{4}, \frac{3}{2})$, $(1, 1)$ and $(\frac{3}{2}, \frac{3}{4})$ into our objective function $10a_1 + 25a_2$, we see that it is minimised when $a_1 = \frac{3}{2}$ and $a_2 = \frac{3}{4}$, giving a value of 33.75.

   (d) [⋆⋆⋆⋆] Generalise this method to give an upper bound for an arbitrary linear programming problem in standard form. (In fact, this "upper bound" will always be *equal* to the optimal value — the new linear programming problem formed is called the "dual" of the original "primal" problem. This is a key ingredient in almost every algorithm for solving linear programming problems.)

8. (a) [⋆⋆⋆] You are working in the upper echelons of MI6 during the Cold War. The organisation's spies in Russia communicate via dead drops, parcels left in predetermined locations on a predetermined schedule. For security reasons, each pair of nearby spies has their own dedicated dead drop which no-one knows about but them. SPECTRE is trying to find out where these dead drops are, so they can intercept the parcels. M is concerned that if SPECTRE finds enough dead drops, they may be able to completely disconnect her spy network, so that at least one pair of spies is completely unable to get in contact with each other even via intermediaries. Give a polynomial-time algorithm which takes as input a list of pairs of spies with dead drops between them and outputs the minimum k such that SPECTRE can disconnect the spy network by compromising k dead drops. (**Hint:** Try reducing the problem to finding minimum cuts in suitably-constructed flow networks.)

   (b) [⋆⋆⋆⋆] James Bundt suggests another measure of the spy network's robustness: the minimum C such that every pair of spies is joined by C "paths" of dead drops, with no dead drop appearing in more than one path. Show that these two measures coincide, in the sense that $C = k$ in every possible spy network. (**Hint:** Try greedily building these paths from the maximum flows of part (a).)