Homework 7: Going with the flow

Due Date: The problems from this assignment are due at 11:59 PM on **Friday** March 1st. Problem resubmissions are due on **Monday** March 4th.

Collaboration: You are allowed (and encouraged!) to discuss these problems at a high level with others. But, please read the full collaboration policy to ensure you are keeping your discussions at the right level, and remember to cite any collaborators.

Problems to Submit: We will count your 1 best mechanical question and your 3 best long-form questions.
We strongly recommend you skim all the problems in a section before attempting them. While we try to make problems in a given section of approximately equal difficulty, you may find some to be easier than others.

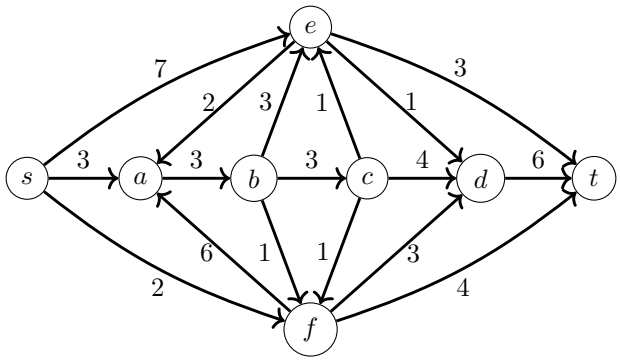
Directions Unless otherwise noted, you are allowed to use algorithms described in class (or prerequisite courses) as though you had library implementations of them. For example, you can say "run the BFS-based 2-coloring algorithm from class on the graph G" or "run the bipartite checking algorithm from lecture 5 on G." We also have a list of data structures and algorithms you can use from 373 here.

Since this is a course about efficient algorithms, algorithms that are slower than necessary are unlikely to get "E" scores, but (unless otherwise noted) we do not care about constant factors. In general, an algorithm that is correct but (mildly) slower than optimal will get an equal or higher score to an algorithm that is fundamentally incorrect, but fast.

Mechanical Problem

1. Ford-Fulkerson

You are given the following graph where s is the source vertex (where the water starts), t is the sink vertex (where the water ends), and the capacities are written on each edge.



(a) Perform the Ford-Fulkerson algorithm on the graph. Draw the residual graph after each augmenting step (i.e. after each iteration of the while loop from class). And draw the final flow

(b) What is the min s, t cut? List both the two sets of vertices and the capacity of the cut (i.e. the value of the cut).

Long-Form Problems

2. Humans & Robots

In the near future where humans and robots cohabit the planet, you are now a manager of a large production company. You need to assign humans and robots to finish different tasks. You want to assign the w total workers (humans and robots) to the t total tasks, subject to the following conditions:

- Each task needs **exactly** s total workers.
- Each task needs **at least** two humans and **at least** three robots.
- Each worker (human or robot) will report to you which task(s) they can do. You must never assign a worker to a task they cannot do.
- Nobody may be assigned to more than 3 tasks (3 or fewer, down to 0, is acceptable).
- Nobody may be assigned to the same task more than once.

As an example, suppose you get a list like:
Worker 0, human, can do tasks 0 and 1.
Worker 1, human, can do tasks 0 and 1.
Worker 2, robot, can do task 0.
Worker 3, robot, can do tasks 0 and 1.
Worker 4, robot, can do tasks 0 and 1.
Worker 5, robot, can do task 1.
Worker 6, robot, can do task 1.

For $s = 5$, you could report (for example, multiple assignments are possible):
"Assign workers 0,1,2,3,4 to task 0"
"Assign workers 0,1,4,5,6 to task 1"

For $s = 6$ you should report:
"No valid assignment is possible."

(a) Describe a graph you could use for a max-flow computation in solving this problem. Remember to give capacities and directions for edges, and choose your source and target!

(b) How many vertices does your graph have (in terms of $w, s,$ and/or t whichever of those you need).

(c) Describe how you will interpret a flow in the graph as an answer to the assignment problem. Be sure to mention both finding an assignment and handling there not being an assignment possible.

(d) Explain (in 1-2 sentences each) why any assignment you return must meet all 5 of the conditions in the bullets above.

3. Release The Kraken

In class, we saw how to tell if the Mariners could still win the division. In this problem, we're going to see if the Kraken can still win the President's Trophy. The President's Trophy is awarded to the hockey team that finishes the season with the best record. Unlike in baseball, where every game ends in a win or a loss and the team with the most wins is the champion, hockey uses "points" to decide on standings. A hockey game can end in three different ways[1]:

- In regulation: the winner of the game gets 2 points in the standings and the loser gets 0.
- In overtime: the winner of the game gets 2 points in the standings, the loser gets 1 point in the standings.
- As a draw: both teams get 1 point in the standings.

Note that the number of points awarded per game can vary! Either 2 or 3 points total could be awarded.

The team with the most points in the standings wins the President's Trophy. Unlike in class, we want to see if the Kraken can win the trophy "outright" – it is not enough to have as many points as all other teams, we want them to have *strictly* more points than all other teams.

You'll be given:

- A list of n teams (including the Kraken), and their current number of points.
- A list of k games remaining to be played.

You should return `true` if the Kraken can still finish the season with strictly more points than any other teams and `false` otherwise.

(a) Describe a graph you could use for a max-flow computation in solving this problem. Remember to give capacities and directions for edges, and choose your source and target!

(b) Describe the runtime of the algorithm and briefly justify your answer. Hint: Think about how many edges and vertices you have.

(c) Describe how you will interpret a flow in the graph as an answer to whether an assignment is possible or not. If an assignment is possible, how do you read it from the maximum flow?

(d) Explain (in 1-2 sentence each) why any assignment you return meets all 3 of the conditions given in the top bullets.

[1] Observant hockey fans may notice the information below is quite out-of-date (as the third option was eliminated with the introduction of shootouts in the mid '00's). Consider this problem a historical analysis.

4. Real World: Ethical Implications of P vs. NP

4.1. Ethics

The Association of Computing Machinery (the largest professional organization for computer scientists) has published a code of ethics to aid computer scientists in "reflect[ing] upon the wider impacts of their work [and] consistently support[ing] the public good." We will use their framework to consider what one should do if one has proved **P** = **NP**, as a significant number of real-world systems are built assuming **P** $\neq$ **NP**.[2]

4.2. Applying The Code

We'll walk through the code's process for considering the ethics of a particular application. We recommend you read the process on page 13 (number as shown on the page) of the linked pdf and look at the case studies on the following pages for examples. We'll expect you to write a few sentences for each part (as the case studies in the document do).

Here is your scenario: You are working for a logistics company that uses SAT solvers as part of their scheduling system. Your team spent significant time rewriting the SAT solvers in the system, and produced massive improvements. While preparing a report for your supervisors on the new method, you realize you might have actually designed an efficient algorithm for 3-SAT (and thus proved **P** = **NP**). You must decide who to tell about the breakthrough[3] (and what to tell them).

(a) The first step is to **consider** who would be affected. Who would be affected by the revelation of the method behind the code? You should think about both those affected directly by a decision, and those affected more indirectly.

(b) Now **analyze** the effects of telling others about the code. What rights are impacted and what principles of the code are relevant (you don't need to read the entire code, but you should read each of the large, numbered principles in the document, and mention at least one explicitly in your answer).

(c) Next, **review**: what responsibilities do you think you have in this scenario? What action(s) should you take?

(d) Finally, **evaluate** and summarize your conclusions. How do the principles interact to give you a course of action? Is there other information that you would need to decide on how to proceed?

4.3. P vs. NP in the real world

(a) Think of at least one potential **negative** side-effect of proving **P** = **NP**. Describe what **NP** problem you can now solve (by describing inputs, outputs, and the certificate) and what negative effect might result (2-3 sentences).

(b) Think of at least one potential **positive** side-effect of proving **P** = **NP**. Describe the **NP** problem you can now solve (by describing inputs, outputs, and the certificate) and what positive effect might result (2-3 sentences).

(c) For the most part we live our lives as though **P** $\neq$ **NP** (we don't know any efficient algorithm for an **NP**-complete problem, so it's pretty easy to live as though they don't exist). Suppose someone actually proved $P \neq NP$. Would the ACM Code apply to publicizing this proof (as it did for proving $P = NP$)? Briefly describe why or why not (2-3 sentences).

[2] To be clear, the scenario of accidentally proving **P** = **NP** is **extraordinarily** unlikely. But the consequences would be so significant that it makes for an easy place to start.

[3] For example, you could tell just your supervisors possibly keeping the code within the company, or you could publicly post the code so the whole world can use it, or you could decide to downplay the importance of the breakthrough to your supervisors and try to keep it totally secret.

5. A Proof Problem

Note: as the title suggests, this problem may be comparatively more challenging than other proof problems you've seen thus far in this course. Unless you've taken prior proof-based courses, we recommend trying the other long-form problems first.

Recall from lecture the minimum vertex cover problem, which is to find a minimum-sized set of vertices S such that every edge has at least one of its endpoints in S. We want to solve this problem for bipartite graphs, which if you'll recall, are graphs whose vertices can be split up into two disjoint sets X and Y in such a way that every edge has one endpoint in X and the other in Y. (Equivalently, no two vertices in X have an edge between them, and likewise with Y.)

In this problem, we will guide you through the proof of an algorithm for finding a minimum vertex cover of a bipartite graph using max-flow and min-cut.

Some sections are marked with a dagger (†). **All such sections are optional** and will not be graded, though you may find it informative or helpful to do them. You are welcome to discuss these parts of the proof during office hours as well. **You may take these statements as fact in subsequent parts of the proof!**

We understand if the length of the problem or the number of parts looks daunting, but the problem is structured this way to make things easier for you, not harder. We've broken the proof down into smaller, more manageable pieces, and each piece does not necessarily need to have a lengthy response.

Suppose G is our input bipartite graph with bipartite partition X and Y. We will define the vertex set of H to be the vertex set of G with s and t appended to it, where s and t are source and sink/target vertices respectively that aren't present in G.

H will have directed edges (s, x) and (y, t) for all x in X and y in Y, each with a capacity of 1. In addition, H will also have a directed edge (x, y) for each undirected edge $\{x, y\}$ in the edge set of G, each with a capacity of ∞.

Our ultimate goal will be to prove that the capacity of our minimum s-t cut on H is equal to the size of a minimum vertex cover on G, which will also give us a way to come up with the minimum vertex cover itself.

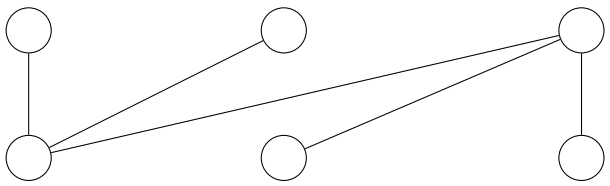
(a) Our goal in this part will be to show that, given a *minimum* s-t cut (A, B) of H, we can construct a vertex cover of G whose number of vertices is equal to $\text{cap}(A, B)$, which is the number of edges leaving A and entering B.

(i) **Propose a vertex cover of G given in terms of some unions of X_A, X_B, Y_A, and Y_B, where:**

$$X_A = X \cap A \qquad Y_A = Y \cap A$$
$$X_B = X \cap B \qquad Y_B = Y \cap B$$

("$\cup$" denotes union, while "$\cap$" denotes intersection.)

Hint: Consider the example bipartite graph to the right. Find a minimum vertex cover for this example. Then, explicitly build H for this particular G and run Ford-Fulkerson to come up with a min cut. Try out different unions of the 4 sets above until you can find one that has the same vertices as the vertex cover.



(ii) **Argue that there are no vertices from X_A to Y_B in H.**
(If you are concerned that this might not hold true for the H that you constructed, please reach out to the staff via the discussion board or office hours. We can help get you pointed in the right direction.)

Hint: try using a proof by contradiction. On the one hand, the capacities of the edges leaving s (or entering t) provide an *upper* bound on the maximum flow in H. On the other hand, the capacity of any one edge going from A to B provides a *lower* bound on $\text{cap}(A, B)$, which is by definition the capacity of a min cut.

(iii) **Prove that the set of vertices you proposed in (i) is indeed a vertex cover of G.**

Hint: start by taking an arbitrary edge $\{x, y\}$ in G, then consider 2 cases, one where x is in A, and one where x is in B. Show that at least one of x or y is in the vertex cover in either case. The result from (ii) will come in handy for one of the cases.

(iv)† **Suppose that (a, b) is an edge in H with endpoints a in A and b in B. Argue that it must hold true that either $a = s$ or a belongs to Y_A.**

Hint: try using a proof by contradiction where you start by assuming that a comes from X_A. (Ask yourself, why did we suggest X_A in particular?) It may help to observe that A is the disjoint union of $\{s\}$, X_A, and Y_A, and that likewise, B is the disjoint union of $\{t\}$, X_B, and Y_B. Use (ii) and a property of X to disqualify certain possibilities of which set b could come from, then show that the only remaining possibility would result in a contradiction. The term "independent set" may come in handy.

(v)† **Prove that any edge going from A to B is either of the form (s, x_b) or (y_a, t), where x_b in X_B and y_a in Y_A.**

Hint: start by supposing that you have an edge (a, b) in H, where a is in A and b is in B. Apply the result from (iv) so that you can do proof by cases.

(vi) **Show that the number of vertices in your vertex cover is equal to** $\text{cap}(A, B)$**.**

Hint: apply the result from (v) to simplify the sum that $\text{cap}(A, B)$ is defined in terms of. At some point, you'll need to use the disjointness of two sets to be able to say that the cardinality of the union is the sum of the individual cardinalities.

(b) **It turns out that it is not too hard but somewhat tedious to show that given an arbitrary vertex cover; one can always construct an s-t cut, the capacity of which is equal to the size of that vertex cover. You may use this fact without proof in your solution.** Now we will be running the Ford-Fulkerson algorithm on your graph H. This gives us the following sets of vertices: $X \cap A$, $X \cap B$, $Y \cap A$, and $Y \cap B$. **Propose a minimum vertex cover in terms of the above sets.** (Look at part (a). Déjà vu?) **Argue that this is a minimum vertex cover.**

6. Programming: Sauerkraut Signage

Robbie's Sauerkraut restaurant wants to up its marketing and has purchased a kit to create some eye-catching signs outside the restaurant. The kit consists of some number of LED displays that can each display one letter of the alphabet at a time. However, due to issues during shipping, each display is limited in which letters it is capable of showing. Given these partially functioning displays, Robbie wants you to figure out whether or not a certain catchphrase can be assembled for a sign.

More precisely, you are given a list of displays (where each display is itself a list of letters) and a phrase you want to output (which is a list of letters of that phrase). You then output whether or not you can select one letter from some of the displays such that they match the phrase. You may assume that each letter is a capital letter and that there are no spaces in the phrase. Also, you do not need to use every display to output the phrase.

For example, you could get:
Display 1 can display "A", "B", "C", "D", "E".
Display 2 can display "E", "F", "G".
Display 3 can display "E", "F", "G", "H".
Display 4 can display "Z".

Given the phrase "BEG": output True.
Given the phrase "BEEF": output False.

For this problem you will be required to use a graph and network flow to solve it. Your algorithm should set up a graph model and run a network flow algorithm once. You are **not** allowed to brute force over every possible combination of display letter choices. The library you will use for the graph and the network flow algorithm is JgraphT so you don't need to implement your own. If you want to debug on your own system you will need to download the `.zip` or `.tar.gz` file from the latest release section and add the `.jar` files to your project.

Here is a guide on using JgraphT in a project on various IDEs, you should only need the `jgrapht-core-1.5.1.jar` file as a dependency.

The starter code gives you all the graph and network flow tools to solve this problem but if you want to learn more about the JgraphT library the library overview, Graph Javadoc and Flow Javadoc are good resources to start with.