Homework 7: Going with the flow

Due Date: The written parts of this assignment (problems 1-4) will be due at 11:59 PM on **Friday** December 2nd. As always, that means that the TAs will grade **every** written problem you submit to gradescope before that deadline. You will also be able to resubmit two written problems with every homework; the resubmission for this week is due on **Monday** December 5th.

The coding problems may be submitted at any time through the last day of classes (Friday Dec. 9), but we strongly recommend you finish any you intend to do by the due date for the written assignment so you have time to do next week's problems next week.

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.

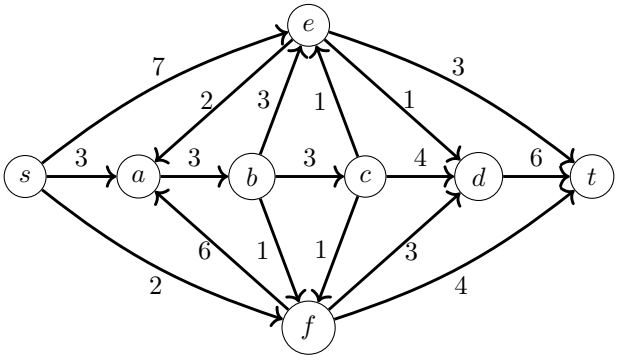
Directions Unless otherwise noted, you are allowed to use algorithms described in class (or prerequisite courses) as though you had library implementations of them. For example, you can say "run the BFS-based 2-coloring algorithm from class on the graph G" or "run the bipartite checking algorithm from 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. Assignment

Honeycomb Outdoor Adventures (HOA) needs your help. They are looking to send out every single tour they have available each day (to avoid tour cancellations) and are asking you to create an assignment of guides and customers to allow this to happen! They want to assign their p total participants (guides and customers) to the t tour activities they provide each day, subject to the following conditions:

- Each tour needs **exactly** s total participants.
- Each tour needs **at least** two guides and **at least** three customers.
- Each participant (guide or customer) will report to HOA which tours they are interested in. You must never assign a participant to any tour they are not interested in.
- No participant may go on more than 3 tours each day (3 or fewer, down to 0, is acceptable).
- No participant may go on the same tour more than once.

As an example, suppose you get a list like:
Participant 0, who is a guide, finds tours 0,1 acceptable.
Participant 1, who is a guide, finds tours 0,1 acceptable.
Participant 2, who is a customer, finds tour 0 acceptable.
Participant 3, who is a customer, finds tours 0,1 acceptable.
Participant 4, who is a customer, finds tours 0,1 acceptable.
Participant 5, who is a customer, finds tour 1 acceptable.
Participant 6, who is a customer, finds tour 1 acceptable.

For $s = 5$, you could report (for example, multiple assignments are possible):
"Assign participants 0,1,2,3,4 to tour 0"
"Assign participants 0,1,4,5,6 to tour 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 $p, s,$ 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) Describe (in 1-2 sentences each) why any assignment you return must meet all 5 of the conditions in the bullets above.

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

3.1. 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 $P \neq NP$.[1]

3.2. 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[2] (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?

3.3. 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).

[1] 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.

[2] 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.

4. 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 end points 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 at office hours at well. **You may use the statements as facts in future 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 in order to make things easier for you, not harder. We've broken the proof down into smaller, more manageable pieces, and each individual piece does not necessarily need to have a lengthy response.

To start with, we want to construct a directed graph H such that we can get a minimum s-t cut (A, B). This construction of H and the result of finding a minimum s-t cut will reveal the following four sets of vertices in H: $X \cap A$, $X \cap B$, $Y \cap A$, $Y \cap B$. The minimum vertex cover of a bipartite graph found in this way will be some union of 2 or more of these 4 sets.

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) **Give a construction of the graph H. Make sure to describe where the edges are directed towards (G is undirected so you can impose directions on the edges for H) and specify capacities for the edges.**

Hint 1: If you build your graph properly, the maximum flow will be finite, and so the capacity of your minimum s-t cut should also be finite.

Hint 2: Consider applying ∞ capacities to the edges in H that correspond to edges in G.

Hint 3: Your source vertex s should be in A in your construction, and it is possible that s sends edges directly to vertices in B.

Hint 4: Think about how you could get a construction for H such that once we obtain a min s-t cut, there are only edges between certain sets. For example, there could be edges from $X \cap A$ to $Y \cap A$, $X \cap A$ to $Y \cap B$, $X \cap B$ to $Y \cap B$, and $X \cap B$ to $Y \cap A$. Consider how we might eliminate one of these sets of edges by designing H carefully. (Think about infinite capacities again.)

Hint 5: Come to office hours!

(b) 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 combination of X_A, X_B, Y_A, and Y_B, where:**

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

Hint: We strongly recommend making a small example bipartite graph (say 4-5 vertices on each side) and finding a max-flow and min-cut and guessing from there.

(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 will do our best to get you pointed in the right direction.)

Hint: try using a proof by contradiction. It may help to think about how the capacities of the edges leaving a certain vertex in H provide an upper bound on the maximum flow in H.

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

Hint: use (ii).

(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: apply the result from (iv) so that you can use proof by cases.

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

Hint: use (v).

(c) In this part, our goal will be to show that, given a minimum vertex cover, we can construct an s-t cut whose capacity is equal to the size of that minimum vertex cover.

(i) **Given a minimum vertex cover W, propose an s-t cut (Q, R), where Q contains s and R contains t. Both should be given in terms of some combination of $\{s\}$, $\{t\}$, X, Y, W, W_X, and W_Y, where $W_X = W \cap X$, $W_Y = W \cap Y$ and both Q and R are of the form $? \cup ? \cup ?$.**

Hint: the sets being "union-ed" together should form disjoint unions of Q and R respectively. You may also want to consider using the set-difference operation.

(ii)† Define $f(U_1, U_2)$ to be the set of edges in H that are directed from a vertex in U_1 toward a vertex in U_2. To be more precise:

$$f(U_1, U_2) = \{(u_1, u_2) \in E_H : u_1 \in U_1 \text{ and } u_2 \in U_2\}$$

Briefly argue why each of the following statements are true.

(about 1 sentence each unless we say otherwise, probably 2 sentences at most)

- (s, t) is not an edge in H.
- $f(\{s\}, Y - W) = \emptyset$
- $f(X - W, \{t\}) = \emptyset$
- $f(X - W, W_X) = \emptyset$
- $f(W_Y, W_X) = \emptyset$
- $f(W_Y, Y - W) = \emptyset$
- $|f(\{s\}, W_X)| = |W_X|$
- $|f(W_Y, \{t\})| = |W_Y|$
- $f(X - W, Y - W) = \emptyset$ (around 3 sentences)

  Hint: consider using proof by contradiction combined with a certain definition.

(iii) **Prove that $\text{cap}(Q, R) = |W|$.**

Hint: the bottom two bullet points in the middle column above will be the most useful to you, but each of the other statements listed above are (technically) necessary to simplify the summation. Don't sweat the technical details too much though.

(d) 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 $Y \cap B$. **Give a minimum vertex cover in terms of the above sets.** (Look at part (b). Déjà vu?) **Argue that this is a minimum vertex cover.**

Hint: Both (b) and (c) started with us proving smaller things leading up to some big result. Use the big results from (b) and (c) to show that there cannot be a vertex cover with a smaller size.

5. 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.