Homework 7: Max Flow Min Cut

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting, you should not take those estimates as hard length-limitations.

Our solutions for any individual problem will fit in approximately one page or less. **If you submit an answer substantially longer than 2 pages, the TAs are allowed to stop reading at the end of page 2.**

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to gradescope; we have a different box for each problem, please give yourself time to submit.

Note on running times
When analyzing running times on this homework, you may assume that creating a graph with n vertices and m edges takes $\mathcal{O}(n+m)$ time, as long as it is constant time to determine whether an edge exists (e.g., because you're reading it from input, or using a very simple rule to decide if edges exist).

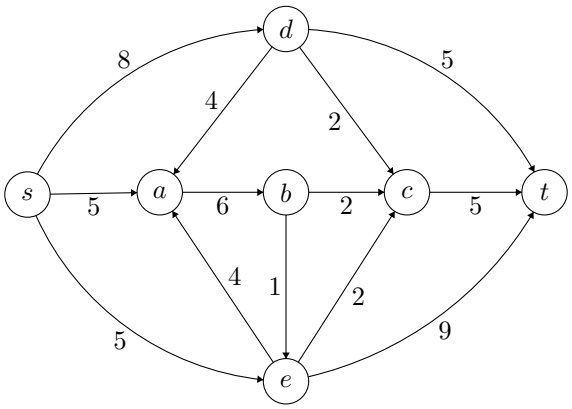
1. Try the algorithm by yourself! [10 points]

Execute the Ford-Fulkerson algorithm on the following graph. When there are multiple augmenting paths available, pick the shortest one (that is the one with the fewest edges). If multiple shortest paths are available, choose the one which increases the flow the most.

Remark: with this rule for choosing the augmenting path, you are actually running the Edmonds-Karp algorithm!).

At the end, you'll submit the following on Gradescope:

- the maximum flow (i.e., the amount of flow on each edge).
- The value of the maximum flow.
- The minimum cut (this should be formatted as two sets of vertices)
- The capacity of the minimum cut.



2. Playoffs, you kiddin' me?

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 Seahawks can still win their division. Unlike in baseball, where every game ends in a win or loss, a football game can end in a tie. A tie is treated as $1/2$-win and $1/2$-loss for both teams. Additionally, the teams in the Seahawks division will play not just each other, but other teams in the NFL. These games count toward the standings the same,

but we only want to check if the Seahawks finish above every team in their division. Finally, we want the Seahawks to finish alone in first place (meaning they have strictly more wins than every other team in their division). You'll be given:

- A list of d teams (including the Seahawks) in the Seahawks division, and the current number of wins for each (which may be an integer or an integer plus $1/2$).
- A list of n teams outside the Seahawks division, and the current number of wins for each (which may be an integer or an integer plus $1/2$).
- A list of k games remaining to be played. (involving all $n + d$ teams).

You should return `true` if the Seahawks can still finish the season with strictly more wins than any other team in their division and `false` otherwise.

(a) Describe a graph you can run a max-flow algorithm on. Be sure to mention edge directions and capacities.

(b) Briefly justify correctness. We aren't expecting a formal proof here, but you should have a sentence or two for each of the restrictions given in the problem.

(c) Describe how you'll tell whether an assignment is possible or not. If an assignment is possible, how do you read it from the maximum flow?

(d) Describe the runtime of the algorithm in terms of n, d, k. Briefly justify why the running time is the value that you state.

3. 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.

4. Chess-Box-Compassion

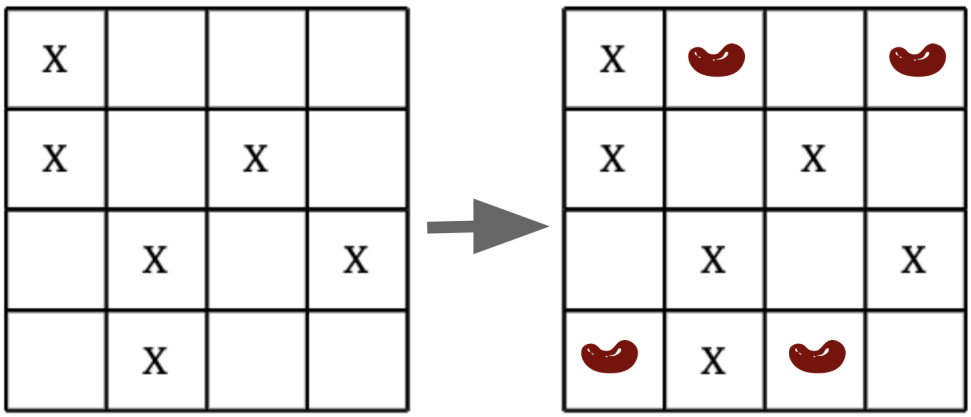
The 421 staff has rented out the CSE atrium, which we will think of as $n \times n$ grid. The staff intends to host the 1st Annual Paul G. Allen School of Computer Science & Engineering Chess-Boxing Tournament[1]. In this tournament, each match must use their own "chess-boxing field" which we can represent as a 1×1 boxing ring attached to a 1×1 chess-table, thus an entire field has size 2×1 or 1×2 (i.e. we can rotate the fields as well).
In the atrium, there are already some permanent fixtures (such as columns, coffee bar, stairs, etc) where you cannot place any part of the field.

You, as a 421 student, noticed that your entire study group seems to be unable to attend your weekly study session since they will be chess-boxing and you, feeling compassionate, want to ensure that none of your friends feel sorrowful should they lose at chess-boxing.

You decide that you will sabotage the tournament. You intend to do this by placing giant beans on some of the squares in the atrium to ensure that there are no free blocks of size 2×1 or 1×2 thus ensuring the tournament must be postponed[2]. Since the giant beans are heavy, you want to only place the minimum you need to ensure that the atrium is untenable for the tournament. Determine the minimum number of giant beans you need to place to ensure that the tournament will not be held.

More formally, you're given an $n \times n$ representation of the atrium, where certain elements are pre-marked as "full". You need to decide where to place the minimum amount of beans such that after the placement, the organizers will be unable to place any 1×2 (or 2×1) tiles in the atrium. For simplicity, you may return just the number of beans (rather than the exact locations where they go).

One example is shown on the table on the left, where before you place any beans there are already 6 locations that are taken. Your algorithm should output 4, corresponding to the four squares marked with the beans on the right.



(a) Give an efficient algorithm that outputs the minimum number of extra beans that we would need to place to guarantee that we cannot place any chess-boxing field on the remaining board. (You must use a max-flow/min-cut-based algorithm in this problem).

(b) Explain why your algorithm works. We don't need a full proof here, but you should have an explanation of why the object you're finding in the last part corresponds to beans placed out, and why the one you find gives you the minimum number of additional squares.

[1] PGASCSECBT for short.
[2] and thereby ensuring you can discuss the Ford-Fulkerson algorithm with your study group

(c) Describe the runtime of the algorithm above. Let the board be $n \times n$, let the initial board have s squares that are already marked with an X, and suppose you find the answer is a additional squares must be marked. Briefly justify why the running time is correct.