Homework 6: Dynamic Programming and Network Flow

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.

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 will have a different box for each problem, so please give yourself extra time to submit.

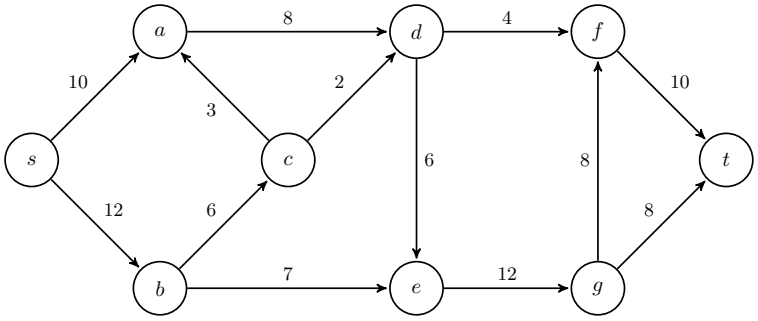
1. Try the algorithm by yourself! [10 points]

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.



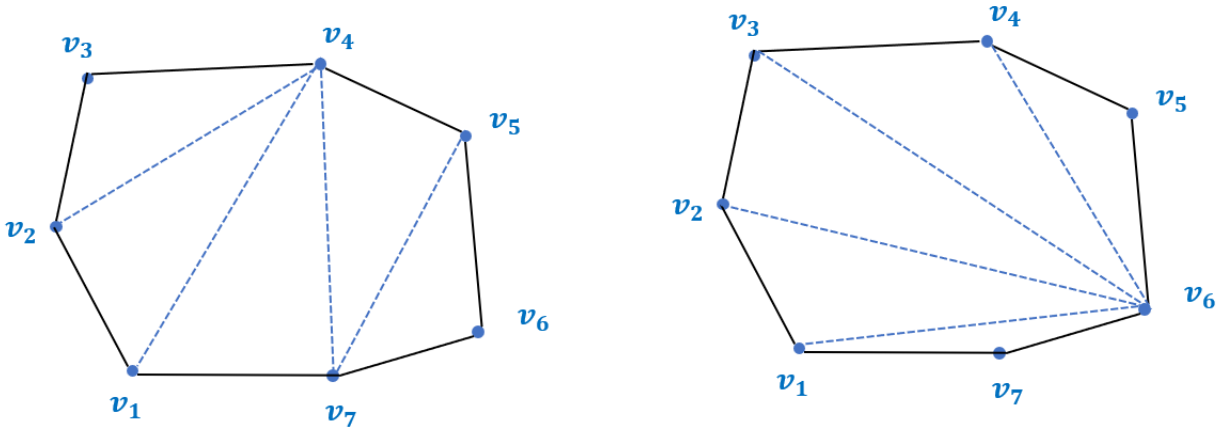
2. Cheap Field Fencing [25 points]

You are helping out a hobby farmer in a very flat area who has a property with n very heavy-duty fence-posts sunk in concrete on the perimeter of the property at locations given by coordinates $v_1 = (x_1, y_1), \ldots, v_n = (x_n, y_n)$ They have installed straight-line fencing between adjacent fence-posts v_i and v_{i+1} for every i as well as between v_n and v_1. Wherever you are inside the fenced off area you can see every bit of the exterior fencing.

Your hobby farmer friend wants to grow many different kinds of crops and raise many different kinds of animals that they need to separate from each other in different portions of their property, but they don't want to add any more heavy-duty fence-posts or use any more fencing than is absolutely necessary.

You get excited when you realize that you can create $n - 2$ different zones that are all 3-sided simply by running straight-line fences inside the field between pairs of existing v_i and v_j. (Of course those fences can't cross.)

When you think some more, you realize that there are many possible ways to do it and some of them result in using much less fencing than others:



This gets you even more excited because you realize that you can apply the dynamic programming ideas you learned in your algorithms course to figure out what one would use the least fencing! In particular, you start wondering what the best choice of the third vertex v_k would be for the 3-sided region that has the 2 other vertices v_1 and v_n. This gives you some ideas of how to proceed.

Apply all the standard steps associated with dynamic programming to get an efficient algorithm to the find shortest length of fencing necessary.

3. Risky Business [25 points]

Suppose that you have a flow network G with source s and sink t.

This problem is about understanding something more that will help to narrow down the nodes/edges that might be potentially involved in bottlenecks in such a network.

We say that a node v is:

- *upstream* of any st-bottleneck iff for every minimum st-cut of G, v is on the side of the cut that includes s,
- *downstream* of any st-bottleneck iff for every minimum st-cut of G, v is on the side of the cut that includes t, and
- *risky* otherwise.

Only the edges that touch risky nodes are of potential concern with respect to bottlenecks.

Give an algorithm that runs in time within a constant factor of a single a maxflow computation that classifies every node of G as either upstream, downstream, or risky.

4. Don't let the Big10 Take Over! [25 points]

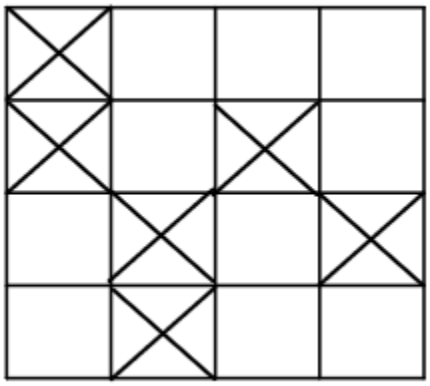
As part of UW's commitment in joining the Big10, UW is now required to field teams in other sporting competitions. For one of these new competitions, *chess-boxing*[1], the UW Athletics department has requisitioned the Allen Center atrium for a tournament and marked it off using an $n \times n$ square grid.

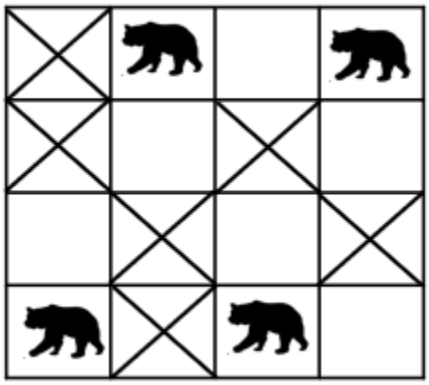
Each match in a chess-boxing tournament requires its own "chess-boxing field" which we can represent as a 1×1 boxing ring attached to an adjacent 1×1 chess table, thus an entire field is a 2×1 or 1×2 rectangle (i.e. we can rotate the fields as well).
In the atrium, there are already some permanent fixtures (such as columns, stairs, etc) where it is impossible to place any part of any chess-boxing field.

You decide that you want to prevent this takeover of the CSE atrium so that the athletic department cannot run even a single match of their tournament in CSE. You intend to do this by placing heavy bear sculptures on some of the squares in the atrium to ensure that there are no free blocks of size 2×1 or 1×2. Since the bear sculptures 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 bears you need to place to ensure that no matches can take place there.

More formally, you're given an $n \times n$ representation of the atrium, where certain entries are pre-marked as off-limits. You need to decide where to place the minimum number of bear sculptures such that after the placement, the organizers will be unable to place any 1×2 (or 2×1) chess-boxing fields in the atrium. For simplicity, you may return just the number of bears (rather than the exact locations where they go).

One example is shown on the diagram on the left, where before you place any bears there are 6 squares that are already off limits for the tournament. Your algorithm should output 4, corresponding to the four squares marked with the bears on the right, which is one of two ways that the bears can be placed to achieve this number.





(a) Give an efficient algorithm that outputs the minimum number of bear scupltures 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 bears you would place, and why the one you find gives you the minimum number of bears.

(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 as full, and suppose that you find the answer is a additional squares must be marked. Briefly justify why the running time is correct.

[1]This is a real competition in which players alternate games of chess with boxing rounds. See https://en.wikipedia.org/wiki/Chess_boxing

5. The most likely path [Extra credit]

Suppose that you have a set S of possible labels and are given a directed graph $G = (V, E)$ with a designated start node s with each edge (u, v) having a label $L(u, v)$ from S. (Note that multiple edges out of a node may have the same label.) In addition, each edge has a probability $p(u, v) \geq 0$ of being taken when at u. In other words, for every $u \in V$,

$$\sum_{v:\ (u,v)\in E} p(u, v) = 1.$$

The probability of taking a path beginning at the start node s is the product of the probabilities labeling its edges. Produce an efficient algorithm that takes as input the graph G with its edge labels and edge probabilities and a sequence of labels $a_1, \ldots, a_t$ and determines the most likely path beginning at node s that is consistent with the sequence of labels (and determine the probability of that path). You can assume that arithmetic operations on real numbers have cost 1.