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.

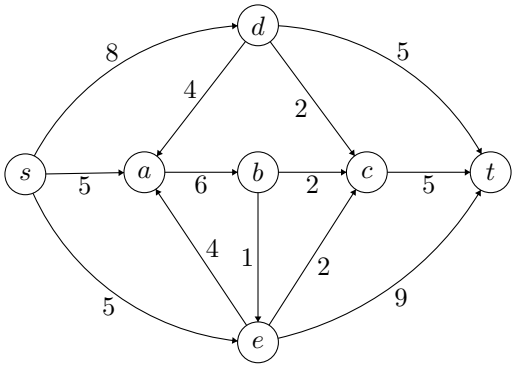
1. Try the algorithm by yourself! [10 points]

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.



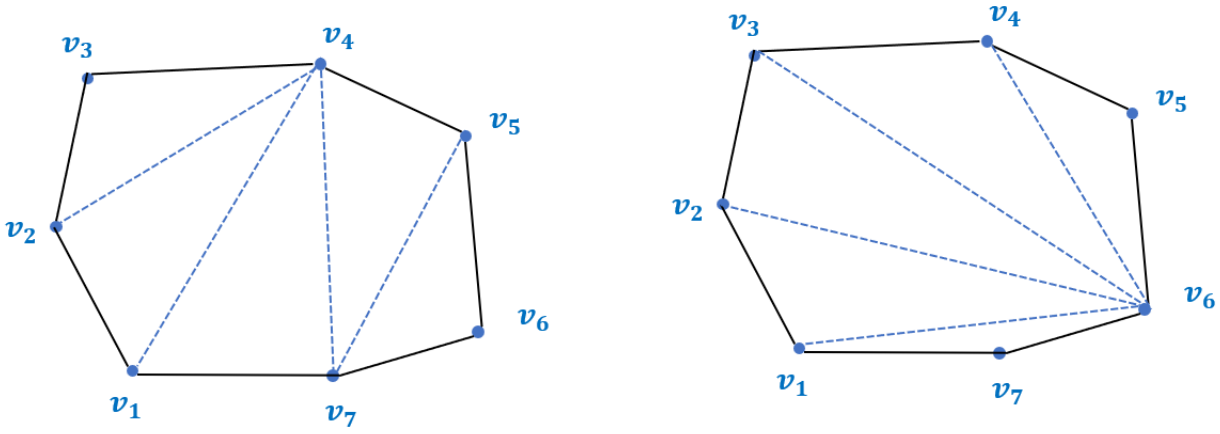
2. Cheap Field Fencing [25 points]

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. Get Out! [25 points]

In emergency planning we often need to set up escape routes ahead of time in case of a disaster. In this problem you are given a directed graph $G = (V, E)$ representing a set of possible 1-way roads. Each node in V is either a populated node in P, a safe node in S, or an intersection node in I. (These are disjoint sets of nodes whose union is V.)

(a) An escape plan consists of a collection C of directed paths in G such that

- each path in C begins at a node in P and ends at a node in S,
- for each node $u \in P$, there is exactly one path in C that begins at u, and
- no two paths in C share any edges of G.

Give a polynomial-time algorithm to determine whether or not an escape plan exists.

(b) Suppose also that every node v not in P has a capacity c_v. A congestion-free escape plan is an escape plan in which

- the directed path beginning at node $u \in P$ does not go through any other populated node, and
- for every node v not in P, the total number of paths containing v is at most c_v.

Show how to determine in polynomial-time whether or not such a congestion-free escape plan exists and produce such a plan if one does.

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.