CS161 Summer 2022 Problem Session 6

Problem 1: Do You Even Prove?

Recall the "Do You Even Lift?" problem from Problem Session 5. Brutus is trying to load up a 45 lb. bench press bar with weight plates to form a desired **exact** target weight. The available plate sizes, in pounds, are: 45, 35, 25, 10, 5, 2.5, and there are infinitely many of each available; for safety, plates must be added in identical pairs. Brutus wants to use as **few** plates as possible.

Waverly devised the following algorithm: start with the empty bar, and keep adding the heaviest pair of plates possible that doesn't cause the total weight so far to exceed n. In the original problem, we found a counterexample that showed that this was wrong.

Suppose that Waverly had tried to prove the correctness of her algorithm as follows:

- **Claim**: At any point during the algorithm, there is at least one optimal solution (an ordered set of choices) that we have not ruled out. That is, our choices so far can be extended to reach an optimal solution.
- **Base Case**: At the start of the algorithm, before we have even done anything, then at least one optimal solution necessarily exists, and we have not ruled it out because we have not even done anything yet.
- **Inductive step**: Suppose we are at some stage of the algorithm and we have not yet hit the target weight. Inductively, there still exists at least one optimal solution S^* that matches our choices so far. If S^* makes the same choice at this stage that we did, we are fine.

  Now suppose that S^* chooses a pair of plates that differs from what our algorithm would choose, i.e., S^*'s pair of chosen pair of plates weighs less. But then we could alter S^* to have it use the heaviest available pair of plates instead; call this new solution S^{**}. We know this is a legal choice because our algorithm never chooses a pair of plates that would overload the bar. And this swap can only get the solution closer to the goal, so S^{**} is no worse than S^*.

  Therefore we have shown that an optimal solution – namely, S^{**} – still exists, completing the proof of the claim.

Waverly's proof is structured more or less correctly for a greedy algorithm proof, but we know that it has to be wrong because the algorithm itself is wrong. Please identify the flaw with the argument.

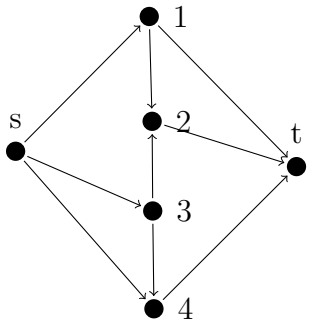
Solution to Problem 1

The setup is all correct except for this sentence: "And this swap can only get the solution closer to the goal, so S^{**} is no worse than S^{*}." This is handwavy, and like many handwavy things, it turns out to be wrong. Being closer to the goal is not necessarily a good thing – that assumption is the whole problem with this greedy algorithm!

For example, we saw that when the target weight is 165 (120 without the bar), the greedy algorithm chooses a pair of 45s, dropping the remaining weight needed to 30. But then it has to choose a pair of 10s and a pair of 5s. The better solution starts by choosing a pair of 35s, which leaves it farther away from the goal (50), but then chooses a pair of 25s, finding a solution that is better than Waverly's.

Problem 2: Neverending Ford-Fulkerson

In lecture, I mentioned that Ford-Fulkerson might fail to terminate if edges have non-integer weights. Here is one example.



The relevant edges are 12, 32, and 34. The idea is that this graph will unwittingly compute successive terms of the sequence defined by

- $x_0 = 1$
- $x_1 = \frac{\sqrt{5}-1}{2}$, hereafter denoted k
- $x_n = x_{n-2} + x_{n-1}$, for $n \geq 2$

Observe that term x_i is the i-th power of k. Suppose that we currently have residual capacities x_1, 0, and x_0 on 12, 32, and 34. Assume that edge 32 has capacity large enough that we can send x_1 units of flow along it in reverse, and therefore we can find an augmenting path that sends x_1 units along the path s-1-2-3-4-t. Then the residual capacities become $x_1 - x_1 = 0$, x_1, and $x_0 - x_1 = x_2$.

Now we send x_1 units along the path s-3-2-1-t. Then our capacities are x_1, 0, x_2. Can you see where to go from here to finish up the demonstration that the algorithm goes on forever?

Solution to Problem 2

We proceed as follows:

Next we repeat the s-1-2-3-4-t path, this time with x_2 units of flow, to get capacities $x_1 - x_2 = x_3, x_2, x_2 - x_2 = 0$.

Then we use the only other type of path we need, s-4-3-2-t. We send x_2 units, ending with capacities $x_3, 0, x_2$.

But now we are back where we started, except two terms farther ahead in the x_i sequence! We can repeat this series of moves arbitrarily many times, reaching tinier and tinier x_i values, but there will always be another flow to find.

Problem 3: Ties

Can you extend the "can my team still win the league" problem from Lecture 13 to handle ties? Specifically, I mean a system in which a win is 2 points, a tie is 1 point (per team), and a loss is 0 points.

(Some sports leagues disincentivize playing it safe by making a win 3 points and a tie 1. I haven't figured out how to adapt the flow setup to do this; I'm not sure it's even possible in the language of flow. If you find a way, please let us know on Ed!)

Problem 4: MST3K

Suppose that an undirected weighted graph G (with 3000 edges) has all unique edge weights: 1, 2, ..., 3000.

(a) Is the edge with weight 1 necessarily in every MST of G? (Provide a brief proof or counterexample.)

(b) Is the edge with weight 2 necessarily in every MST of G? (Provide a brief proof or counterexample.)

(c) Is the edge with weight 3 necessarily in every MST of G? (Provide a brief proof or counterexample.)

(d) Assuming that Prim's Algorithm always starts in the same place, does it always find the same spanning tree in G?

(e) You may wonder whether the starting place matters. Can there actually be more than one spanning tree in G? (This is a challenge problem! It involves a moderately meaty proof. You would not be expected to do this kind of thing on the final.)

Solution to Problem 3

When a tie is half a win, it turns out to be not too hard to extend the example from class to handle ties. Instead of pushing losses through the graph, we think of it as pushing "loss points", where a tie gives each team a loss point. As before, we figure out how many "loss points" we need to assign to each threat team so that we surely finish the season ahead of them. Then we construct the same flow graph as before, but with 2 points involved in each game instead of 1.

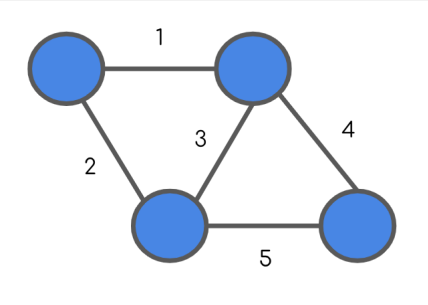
This does the right thing – suppose that we have a node representing that teams A and B have 3 more games to play against each other. Then Ford-Fulkerson might end up assigning 0 loss points to team A and 6 to team B, or 3 to team A and 3 to team B, and so on. All of these represent actual ways that the three individual games could go, and every possible set of outcomes of the three games is covered.

Solution to Problem 4

(a) Yes. Suppose, heading for a contradiction, that there were some minimum spanning tree T of G that did not use the weight-1 edge. Now imagine adding the weight-1 edge to G, creating a cycle C. We know we can delete any edge from C without disconnecting the graph, since (informally, as mentioned in lecture) there were two ways to get to any vertex in C to begin with, and we have removed at most one of them. So we choose to delete something other than the weight-1 edge, and now we have a spanning tree with even lower weight than T, contradicting T's status as a minimum spanning tree.

(b) Yes. The proof is the same as in part (a). The only difference is that C now might contain the weight-1 edge. But C has to involve at least three edges (since this graph is undirected, the smallest possible cycle is a little triangle), so there is some edge in C other than the weight-1 edge, and we delete that one.

(c) No. Now the proof's strategy falls apart, because there can be situations like this in which we definitely don't want all three of the lowest-weight edges:



(d) Yes. Prim's only has a choice to make when the "frontier" of possible edges to add has multiple edges of the same weight. Since that can't happen here, it doesn't even matter whether our implementation of Prim's is deterministic or randomized.

(e) Jeff Erickson says this better than I could – see: `https://jeffe.cs.illinois.edu/teaching/algorithms/book/07-mst.pdf`