Homework 7

Winston (Hanting) Zhang

Friday, November 4, by 23:00

General Instructions. The following assignment is meant to be challenging, and we anticipate that it will take most of you at least 10–15 hours to complete, so please allow yourself plenty of time to work on it. We highly recommend reading the entire assignment right away — you never know when inspiration will strike. Please provide a formal mathematical proof for all your claims, and present runtime guarantees for your algorithms using asymptotic (big-$O/\Omega/\Theta$) notation, unless stated otherwise. You may assume that all basic arithmetic operations (multiplication, subtraction, division, comparison, etc.) take constant time.

Collaboration. Please carefully check the collaboration policy on the course website. When in doubt, ask an instructor.

Consulting outside sources. Please carefully check the policy regarding outside sources on the course website. Again, when in doubt, ask an instructor.

Submission. Homework submission will be through the Gradescope system. Instructions and links have been provided through the course website and Piazza. The only accepted format is PDF. Starting with this homework, only typed solutions will be accepted, and we highly recommend producing your solutions using LaTeX (the text markup language we are also using for this assignment).

Recommended practice problems (do not hand in): KT Problems 7.6, 7.7, 7.9, 7.12, 7.13, 7.16, 7.19, 7.22, 7.23, 7.25, 7.27, 7.35

Problem 1. [15]

When we defined flows in class, we did not rule out that there might be cycles in the flow. Having cycles in your flow seems kind of pointless at best, and a nuisance at worst. Here, you are going to prove a formal version of this intuition, by designing an algorithm that proves that you do not need cycles to get good flows. To be precise, when we say that a flow f "contains cycles", what we mean is that the set of edges with positive flow, i.e., $\{e \in E\} f_e > 0$, contains a cycle.

Your algorithm will be given a graph $G = (V, E)$ with non-negative edge capacities c_e, a source s, sink t, and a valid flow f. Also, there will be no edge into s or out of t; otherwise, the definition of the value of a flow is a bit strange. It is supposed to run in polynomial time (not pseudo-polynomial) and output a new flow f' with $f'_e \leq f_e$ for all edges e (so you cannot add flow to any edges), of the same value $\nu(f') = \nu(f)$, and such that f' is acyclic.

Give and analyze a polynomial-time algorithm for finding such an f'.

Algorithm 1 Remove cycles from flow.

```
procedure FIND-CYCLES-AT(G, v)
  let q ← {v} (queue with one element)
  let visited ← {} (empty set)
  let prev ← {} (empty map)
  let cycles ← {} (empty list)
  while not q.empty do
    let w ← q.pop
    visited[w] = true
    if w = v then
      let C ← BACKTRACK(prev, w)
      cycles.push()
    for each edge e out of v do
      if ahh then

  return
procedure REMOVE-CYCLES-AT(G, f, v)
  let cycles ← FIND-CYCLES-AT(G, v)
  for C in cycles do
    let e* ← edge with minimum cost in C.edges
    for e in C.edges do
      let f_e ← f_e − f_e*
  return f
procedure REMOVE-CYCLES(G, f)
  for v in V do
    let f ← REMOVE-CYCLES-AT(G, f, v)
  return f
```

Definition 1. *Let f, f' be flows on G. Define $f \leq f'$ if $f_e \leq f'_e$ for all $e \in E$.*

Proposition 1. *Given a finite graph $G = (V, E)$ and vertex $v \in V$,* FIND-CYCLES-AT*(G, v) outputs a list of all cycles in G containing the vertex v in $O(E^2)$ time complexity.*

Proof. This is a rare *proof by piazza post*. Kempe states in a post: "If you want to detect cycles, you need to outline it at a level of detail that a good 104 student could implement it from your description."

A good 104 student should recognize that FIND-CYCLES-AT(G, v) is a BFS procedure. The condition for termination is hitting the start node v again in our search, at which point we backtrack through a map of previous pointers to construct a cycle. This is a full BFS that only terminates when all nodes are explored, possibly triggering many termination calls to save multiple detected cycles.

We claim that since this is a full BFS, we detect all possible cycles that contain v. We point to Kempe's comment again and omit a full proof of this claim. The cost of searching with BFS is $O(V + E)$. The termination condition can only be triggered when we we explore a edge going into e; this number is bounded by E. Furthermore, the cost of constructing individual cycles is also bounded by E, since we can backtrack at most E times. Hence the cost of termination and constructing cycles is $O(E^2)$. In total, the runtime complexity is $O(V + E + E^2) = O(E^2)$, as desired. □

Proposition 2. *Let f be a valid flow on G. Given $v \in V$, let $f' =$ REMOVE-CYCLES-AT(G, f, v). Then*

1. *f' is a valid flow on G;*
2. *$f' \leq f$;*
3. *f' does not contain any cycles containing v.*

Proof. We begin by indexing the cycles obtained by FIND-CYCLES-AT(G, v) with $C_1, C_2, \ldots, C_k$. The outer for loop executes its body from $C = C_1$ to $C = C_k$.

Now we define a sequence of modifications to the original flow f as follows. Start with $f_0 = f$, then at each iteration i of the outer loop, we modify f_{i-1} and produce a new flow f_i, ending with $f_k = f'$ (which we output). We claim that for every $1 \leq i \leq k$, f_i is a valid flow, $f_i \leq f$, and f_i does not contain any of the cycles C_j for $j \leq i$.

Indeed, we proceed by induction on i. The base case $i = 0$ is trivial: $f_0 = f$ is obviously a valid flow and $f \leq f$. Since C_0 does not exist, f_0 trivially does not contain C_0.

Now assume for the sake of induction that f_{i-1} is a valid flow such that $f_{i-1} \leq f$ and and f_{i-1} does not contain any of the cycles C_j for $j \leq i - 1$. We want to show that f_i is a valid flow such that $f_i \leq f$ and f_i does not contain any of the cycles C_j for $j \leq i$.

We analyze the inner loop that modifies f_{i-1} into f_i. Consider each node w in C_i. Since C_i is a cycle, we know that w has exactly two edges e_{in} and e_{out} going in and out. When we loop through all edges in C_i in our algorithm, we subtract f_{i,e^*} from e_{in} and e_{out}, setting $f_i^{\text{in}}(w) = f_{i-1}^{\text{in}}(w) - f_{i,e^*}$ and $f_i^{\text{out}}(w) = f_{i-1}^{\text{out}}(w) - f_{i,e^*}$. By our induction hypothesis, f_{i-1} is a valid flow! Thus $f_{i-1}^{\text{in}}(w) = f_{i-1}^{\text{out}}(w)$, and hence we may conclude that $f_i^{\text{in}}(w) = f_i^{\text{out}}(w)$. This hold for all vertices w which we modify within C_i, showing that f_i conserves the in-out flow at each vertex.

Next, we are substracting flow, so clearly $f_i \leq f_{i-1}$. By our induction hypothesis, $f_{i-1} \leq f$, so $f_i \leq f_{i-1} \leq f$. Furthermore, since f_{i,e^*} is the minimum flow in C_i, we can guarantee that all modified edges are still non-negative. Thus this shows that f_i is valid and $f_i \leq f$.

Finally, because $f_i \leq f_{i-1}$, f_i definitely cannot contain any cycles that f_{i-1} does not have. Thus by our induction hypothesis, f_i does not contain any of the cycles C_j for $j \leq i - 1$. Furthermore, $e^* \in C_i$ and we must have $f_{i,e^*} = 0$, therefore f_i does not contain C_i as well, proving that f_i does not contain any of the cycles C_j for $j \leq i$, as desired. This completes the induction!

The result of our induction gives $f_k = f'$ is a valid flow such that $f_k = f' \leq f$ and $f_k = f'$ does not contain any of the cycles $C_1, \ldots, C_k$. These are exactly the three conditions that complete the proof. □

Proposition 3. *Let f be a valid flow on G and $f' =$ REMOVE-CYCLES(G, f). Then*

1. *f' is a valid flow on G;*
2. *$f' \leq f$;*
3. *f' is acyclic.*

Proof. Index the vertices $v_1, \ldots, c_n$. Again, we follow the execution of the loop and define a sequence of modifications to the original flow f as follows. Start with $f_0 = f$, then at each iteration i of the loop, we modify f_{i-1} and produce a new flow f_i, ending with $f_n = f'$ (which we output). We claim that for every $1 \leq i \leq n$, f_i is a valid flow such that $f_i \leq f$ and f_i does not contain any cycles containing v_j for $j \leq i$.

We proceed by induction on i. The base case $i = 0$ is trivial: $f_0 = f$ is obviously a valid flow and $f \leq f$.

Now assume for the sake of induction that f_{i-1} is a valid flow such that $f_{i-1} \leq f$ and f_{i-1} does not contain any cycles containing v_j for $j \leq i - 1$. We want to show that f_i is a valid flow such that $f_i \leq f$ and f_i does not contain any cycles containing v_j for $j \leq i$.

Indeed, the body simply sets $f_i = \text{REMOVE-CYCLES-AT}(G, f_{i-1})$. Combining Proposition 2 with our induction hypothesis, we can immediately conclude that f_i is a valid flow such that $f_i \leq f_{i-1} \leq f$ and f_i does not contain any cycles containing v_j for $j \leq i-1$ plus v_i itself. This completes the induction.

Thus $f_n = f'$ is a valid flow such that $f_n = f' \leq f$ and $f_n = f'$ does not contain any cycles, as desired. □

Problem 2. [15]

Imagine that you are starting a food delivery startup with the motto "We know you hate when you must wait. Herewith we state that it's not great when we are late, so it's our fate to not get paid.[1]" The business model is as follows: you have m delivery drivers. Customers place food orders from restaurants of their choosing, and state a deadline of how much maximum they are willing to wait for their food. They pay the restaurant for the food, and possibly you for delivery. If your company accepts an order from a customer, you promise to deliver it within the specified time limit; otherwise, the (flat) delivery fee of \$10 is waived.[2] Your high-level goal is now to select as many orders to accept as possible (and thus to make as much money as possible), but subject to not being late on any of them.

To be more precise, you are choosing from among n potential customers. For each customer i, you are told where they live (let's call that location L_i), and where the restaurant is that they are ordering from (let's call that R_i). You are also told how many minutes they are willing to wait; this is a number $t_i > 0$. In addition, you know the locations of your m delivery drivers (let's call the location of the j-th driver D_j).

Each delivery driver can only be assigned at most one order (even though some orders may be close together, or close to where the driver lives). If a driver is in charge of an order, they first drive from their location to the restaurant, where they pick up the food. You can assume that picking up food always takes exactly one minute. From the restaurant, they then drive to the customer's house. You have a route planning software (think Google Maps) into which you or your program can enter any pair of locations (restaurants, customer locations, driver locations) and get a precise (and always completely accurate) answer of how long it will take to drive from the first location to the second. Remember that for each order that you serve on time, you get \$10, and for any other order (which you decline or for which you are late), you get nothing.

Give and analyze a polynomial-time algorithm for maximizing the profit that your company makes. Your algorithm should output both the profit and the assignment of drivers to orders that gives that profit.

Problem 0.

Chocolate Problem: 1 chocolate bar

Problem 7.51 from the textbook.

[1] There wasn't enough budget to hire someone to come up with a good marketing slogan.

[2] The customer still pays for the food, but since that money goes directly to the restaurant, it doesn't help you.