Homework 8

Winston (Hanting) Zhang

Friday, November 11, 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]

You are trying to pick out the classes to take during your computer science major. There are n classes available, and for each of them, you know how rewarding it would be to take the class. That is captured by the integer number r_i, which could be positive or negative. (Some classes are not very rewarding. Of course, CSCI 270 has the largest r_i of all of them.) Sometimes, classes build on each other. While the program does not enforce prerequisites, if you are missing a prerequisite for a class, you have to spend extra time studying to catch up, which reduces the reward.

More specifically, for each pair of classes i, j such that i is a prerequisite for j, you are given an integer quantity $p_{i,j} > 0$ which gives you the amount of extra studying you will have to do for class j if you didn't also take class i. This $p_{i,j}$ is subtracted from your total reward if you take class j but not class i. We assume that prerequisites do not contain cycles: if you look at the graph of all edges (i, j) with $p_{i,j} > 0$, this graph contains no cycles.

To summarize the preceding discussion: if you take the set $S \subseteq \{1, \ldots, n\}$ of classes, your total reward is $R(S) := \sum_{i \in S} r_i - \sum_{j \notin S, i \in S} p_{j,i}$. We assume that there is no bound on the number of classes you are allowed to take.

Give (and analyze) a polynomial-time algorithm for finding a set S of classes maximizing $R(S)$.

Hint: In addition to the class material, you might find Section 7.11 in the textbook useful.

Algorithm 1 Class selection with prerequisite weights.

```
let V' ← V ∪ {s, t}
let E' ← {} (empty set of edges)
let c ← {} (empty map of capacities)
for i ∈ V do
  if r_i > 0 then
    E'.add((s, i))
    c[(s, i)] ← r_i
  else
    E'.add((i, t))
    c[(i, t)] ← −r_i
for e = (i, j) ∈ E do
  E'.add((j, i))
  c[(j, i)] ← p_{j,i}
let f ← EDMOND-KARP(G', c)
let A* ← MIN-CUT(G', f)
return A* − {s}
```

Construct graph G' by taking $V' = V \cup \{s, t\}$. For each node i with $r_i > 0$, we add edge (s, i) with capacity r_i. For each node i with $r_i < 0$, we add edge (i, t) with capacity $-r_i$. For each edge $e = (i, j) \in E$ with penalty p_e, we add edge (j, i) (note that this is reversed!) with capacity p_e. Denote the maximum capacity of G' to be $C = \sum_{i, r_i > 0} r_i$. Note that this mirrors the construction we have in Algorithm 1.

Lemma 1. *The capacity of the cut (A, B), as defined from a set of classes $S = A - \{s\}$, is*

$$c(A, B) = C - \left(\sum_{i \in S} r_i - \sum_{j \notin S, i \in S} p_{j,i} \right).$$

Proof. The edges of G' can be split into 3 categories: those leaving the source s, those entering the sink t, and those from the original graph G (now reversed).

1. The edges leaving the source contribute $\sum_{i \notin S, r_i > 0} r_i = C - \sum_{i \in S, r_i > 0} r_i$.
2. The edges entering the sink contribute $\sum_{i \in S, r_i < 0} -r_i$.

3. The edges leaving S and entering $B - \{t\}$ are exactly the ones leaving $B - \{t\}$ and entering S in G, since all edges were reversed. Thus we are counting over all the prerequisite classes for S, which contributes

$$\sum_{j \notin S, i \in S} p_{j,i}$$

The capacity of the cut $c(A, B)$ is the sum of these three terms, which gives

$$\begin{aligned} c(A,B) &= C - \sum_{i \in S, r_i > 0} r_i + \sum_{i \in S, r_i < 0} -r_i + \sum_{j \notin S, i \in S} p_{j,i} \\ &= C - \left(\sum_{i \in S} r_i - \sum_{j \notin S, i \in S} p_{j,i} \right), \end{aligned}$$

as desired. □

Proposition 2. *If (A, B) is a minimum cut in G', then $S = A - \{s\}$ is an optimal selection of classes to take.*

Proof. Lemma 1 shows that every cut of capacity at most C defines a valid set of classes to take. Furthermore, every set of classes S defines a cut $(S \cup \{s\}, (G - S) \cup \{t\})$ of capacity at most C. Thus we see that the cuts (A, B) of capacity at most C are in one-to-one correspondence with feasible sets of classes to take. The capacity of any such cut is

$$c(A, B) = C - R(S),$$

so the cut which minimizes $c(A, B)$ clearly maximizes $R(S)$, as desired. □

Proposition 3. *Algorithm 1 is correct.*

Proof. **Correctness:** The first part of the algorithm exact constructs the needed graph G'. Then we simply run Edmond-Karp on G' to compute the min-cut (A^*, B^*), and output $S = A^* - \{s\}$, which we have proven is the optimal solution.

Termination and Runtime: It depends on the implementation, but constructing G' takes at worst case $O(n+m) = O(m)$ and obviously terminates. Then running Edmond-Karp takes $O(m^2 \log C)$. Then finding the min cut from the returned flow takes $O(m)$. So the total runtime is $O(m) + O(m) + O(m^2 \log C) = O(m^2 \log C)$. □

Problem 2. [5+10=15]

In class, we used a duality-based proof to show that the Ford-Fulkerson algorithm returns a maximum flow and a minimum cut. Here, you will explore another duality-based proof of optimality, for a completely different problem (or rather, two completely different problems).

In fact, one of the problems is one we already solved, way in the beginning of class: unweighted interval selection. Just for concreteness, here it is again: you are given intervals $I_1 = [s_1, f_1], I_2 = [s_2, f_2], \ldots, I_n = [s_n, f_n]$. Your goal is to select as many of these intervals as possible without any selected pair overlapping. Formally, your goal is to select a set J of indices such that $I_j \cap I_{j'} = \emptyset$ for all $j, j' \in J, j \neq j'$. The goal is then to make $|J|$ as large as possible.

Here is the second problem: Interval Hitting. The input is the same, namely intervals $I_1, \ldots, I_n$. Your goal now is to select a set of points $T = \{t_1, t_2, \ldots, t_k\}$, such that for each interval, you have included at least one point. That is, $T \cap I_j \neq \emptyset$ for all j. Your goal is to achieve this with a smallest possible T. For a motivation, think about the last week of the semester. You want to go thank all of your CPs for their amazing help. So you want to visit SAL at a few different points in time, so as to hit the office hours of all CPs. Sometimes, office hours overlap, so you can thank multiple CPs with just one visit. Your goal is to thank all of them with as few visits as possible.

1. Prove the following weak duality lemma:

   Lemma 4. *For all inputs $I_1, \ldots, I_n$, and any valid solution J for interval selection and T for interval hitting, we have $|J| \leq |T|$.*

   Proof. Consider the map $f : J \to T$ which takes some interval I_j and outputs some t_i. We know that at least one such t_i must exist because T covers all intervals. Now for any $j_1 \neq j_2$, I_{j_1} are I_{j_2} are disjoint, thus they cannot share the same time point. Thus $f(I_{j_1}) \neq f(I_{j_2})$, and we know that f is injective – we conclude that $|J| \leq |T|$. □

2. Use Lemma 4 to give an alternative proof (not using Greedy Stays Ahead or even explicit induction) that the greedy algortihm for Interval Selection finds an optimum solution.

   Just as with our analysis of Ford-Fulkerson, you should achieve this by modifying the algorithm to output not just the selected intervals (i.e., the set J), but also some set T of time points, and then relating the two.

Algorithm 2 Intervals with weak duality.

```
let valid ← set of all intervals
SORT valid by finish time
let J ← {} (empty set of intervals)
let T ← {} (empty set of points)
while valid is not empty do
  let I_j = [s_j, f_j] ← interval with smallest finishing time in valid
  valid.remove I_j
  J.add(I_j)
  valid.remove (I_{j'} = [s_{j'}, f_{j'}] such that s_{j'} ≤ f_j)
  T.add(f_j)
return J, T
```

Proposition 5. *Algorithm 2 is correct.*

Proof. We follow the execution of the loop and define a sequence of modifications such that at the end of each iteration i, J_i, T_i, and valid$_i$ are the values of J, T, and valid. We want to show that the final values J_n and T_n are correct (for some n we don't know since it is a while loop).

We claim that each iteration i maintains the invariant that $|J_i| = |T_i|$, J_i is a set of disjoint intervals, valid$_i$ is disjoint from J_i, and T_i covers all intervals removed from valid$_i$. We proceed by induction.

For the base case, clearly $|J_0| = |T_0| = 0$, J is clearly disjoint and trivially disjoint from valid$_0$, and nothing has been removed from valid so the condition on T_0 holds.

Now assume for the sake of induction that we have finished th i^{th} iteration with $|J_i| = |T_i|$, J_i is a set of disjoint intervals, valid$_i$ is disjoint from J_i, and T_i covers all intervals removed from valid$_i$. We want to show that this all holds for the $i + 1^{\text{th}}$ iteration.

Indeed, in the $i + 1^{\text{th}}$ iteration we set $J_{i+1} = J_i \cup \{I_j\}$ and $T_{i+1} = T_i \cup \{s_j\}$. Thus $|J_{i+1}| = |J_i| + 1 = |T_i| + 1 = |T_{i+1}|$.

By assumption, valid$_i$ is disjoint from J_i, so the interval I_j is disjoint from J_i. Thus J_{i+1} is itself disjoint.

Next in the algorithm, we remove all intervals that intersect with J_{i+1} by removing I_j itself along with all intervals that begin before it ends. Denote the removed intervals by R_{i+1}. Thus we maintain that valid$_{i+1}$ = valid$_j \cup R_{i+1}$ continues to be disjoint from J_{i+1}.

By our induction hypothesis, T_i covers all of valid$_i$. Note that for every $I = (s, f) \in R_{i+1}$, we have $f_j < f$ we chose the *smallest* finishing time in valid$_i$. We also know that I must intersect with I_j somewhere, we deduce that $s < f_j$. Thus $f_j \in I$. This means that f_j covers every interval in R_{i+1}, so $T_i \cup \{f_j\} = T_{i+1}$ covers valid$_i \cup R_{i+1}$ = valid$_{i+1}$. Thus this completes the induction.

At the end of the loop, valid is empty, so we know that $|J_n| = |T_n|$, J_n is disjoint, and T_n covers all intervals. Thus J_n is a valid solution for interval selection and T_n is a valid solution for interval hitting; by weak duality, $|J_n| = |T_n|$ implies that J_n and T_n must be the maximal and minimal solutions, respectively. This is exactly what our algorithm returns, so it is correct. □

Problem 0. [0]

Chocolate Problem: 2 chocolate bars

Suppose that you are playing a simple kind of attack-defense game, as follows. There are n targets, which it is your goal to keep alive. Each target starts with a shield of strength c (the same for all targets). Then, in each round, two things happen: (1) the attacker distributes 1 unit of damage over the shields of the n targets in any way he likes. (2) You get to pick exactly one target, and fully restore its shield, to its original strength c. If at any point, any of the shields reaches strength 0, you lose. If you can keep all of your targets alive, you win.

The way you would normally play this is that your opponent (the attacker) gets to see your actions (i.e., which shields you restore) before making their next choice. In that case, it is not very difficult to show that you need[1] $c \geq H_n$ to have a chance against a perfect opponent, and something around $c = H_n$ is also enough if you follow the obvious greedy strategy of always fixing a shield of maximum damage. (It may amuse you to prove these two, though they are worth any chocolate. The fact that you need $c \geq H_n$ is quite easy to prove, while the other one is about medium difficulty.)

Now suppose that you are conducting espionage, and turn the tables. You can look into the future, and know exactly how your opponent is going to distribute the damage in every round. So for each target i and each time $t = 1, 2, \ldots, T$ (where T is the end of the game), you know the damage $d_{t,i}$ your opponent will do to target i at time t. This being a valid strategy for your opponent, you have $d_{t,i} \geq 0$ for all i, t, and $\sum_{i=1}^{n} d_{t,i} = 1$ for all t.

Show that for this version, if $c \geq 3$, that is sufficient for you to win, never mind how large n is. (In fact, I believe that $c = 2$ is sufficient, but a former CP convinced me that my proof of that fact had a small bug.)

[1] $H_n = \sum_{i=1}^{n} 1/i$ denotes the n-th Harmonic number, which is approximately $\ln(n)$.