CS261: Problem Set #2

Due by 11:59 PM on Tuesday, February 9, 2016

Instructions:

(1) Form a group of 1-3 students. You should turn in only one write-up for your entire group.

(2) Submission instructions: We are using Gradescope for the homework submissions. Go to www.gradescope.com to either login or create a new account. Use the course code 9B3BEM to register for CS261. Only one group member needs to submit the assignment. When submitting, please remember to add all group member names in Gradescope.

(3) Please type your solutions if possible and we encourage you to use the LaTeX template provided on the course home page.

(4) Write convincingly but not excessively.

(5) Some of these problems are difficult, so your group may not solve them all to completion. In this case, you can write up what you've got (subject to (3), above): partial proofs, lemmas, high-level ideas, counterexamples, and so on.

(6) Except where otherwise noted, you may refer to the course lecture notes *only*. You can also review any relevant materials from your undergraduate algorithms course.

(7) You can discuss the problems verbally at a high level with other groups. And of course, you are encouraged to contact the course staff (via Piazza or office hours) for additional help.

(8) If you discuss solution approaches with anyone outside of your group, you must list their names on the front page of your write-up.

(9) Refer to the course Web page for the late day policy.

Problem 7

A *vertex cover* of an undirected graph (V, E) is a subset $S \subseteq V$ such that, for every edge $e \in E$, at least one of e's endpoints lies in S.[1]

(a) Prove that in every graph, the minimum size of a vertex cover is at least the size of a maximum matching.

(b) Give a non-bipartite graph in which the minimum size of a vertex cover is strictly bigger than the size of a maximum matching.

(c) Prove that the problem of computing a minimum-cardinality vertex cover can be solved in polynomial time in bipartite graphs.[2]

[Hint: reduction to maximum flow.]

(d) Prove that in every bipartite graph, the minimum size of a vertex cover equals the size of a maximum matching.

[1] Yes, the problem is confusingly named.

[2] In general graphs, the problem turns out to be NP-hard (you don't need to prove this).

Problem 8

This problem considers the special case of maximum flow instances where edges have integral capacities and also

(*) for every vertex v other than s and t, either (i) there is at most one edge entering v, and this edge (if it exists) has capacity 1; or (ii) there is at most one edge exiting v, and this edge (if it exists) has capacity 1.

Your tasks:

(a) Prove that the maximum flow problem can be solved in $O(m\sqrt{n})$ time in networks that satisfy (*). (As always, m is the number of edges and n is the number of vertices.)

[Hint: proceed as in Problem 4, but prove a stronger version of part (a) of that problem.]

(b) Prove that the maximum bipartite matching problem can be solved in $O(m\sqrt{n})$ time.

[Hint: examine the reduction in Lecture #4.]

Problem 9

This problem considers approximation algorithms for graph matching problems.

(a) For the maximum-cardinality matching problem in bipartite graphs, prove that for every constant $\epsilon > 0$, there is an $O(m)$-time algorithm that computes a matching with size at most ϵn less than the maximum possible (where n is the number of vertices). (The hidden constant in the big-oh notation can depend on $\frac{1}{\epsilon}$.)

[Hint: ideas from Problem 8(b) should be useful.]

(b) Now consider non-bipartite graphs where each edge e has a real-valued weight w_e. Recall the greedy algorithm from Lecture #6:

Greedy Matching Algorithm

```
sort and rename the edges $E = \{1, 2, \ldots, m\}$ so that $w_1 \geq w_2 \geq \cdots w_m$
$M = \emptyset$
for $i = 1$ to $m$ do
    if $w_i > 0$ and $e_i$ shares no endpoint with edges in $M$ then
        add $e_i$ to $M$
```

How fast can you implement this algorithm?

(c) Prove that the greedy algorithm always outputs a matching with total weight at least 50% times that of the maximum possible.

[Hint: if the greedy algorithm adds an edge e to M, how many edges in the optimal matching can this edge "block"? How do the weights of the blocked edges compare to that of e?]

Problem 10

This problem concerns running time optimizations to the Hungarian algorithm for computing minimum-cost perfect bipartite matchings (Lecture #5). Recall the $O(mn^2)$ running time analysis from lecture: there are at most n augmentation steps, at most n price update steps between two augmentation steps, and each iteration can be implemented in $O(m)$ time.

(a) By a *phase*, we mean a maximal sequence of price update iterations (between two augmentation iterations). The naive implementation in lecture regrows the search tree from scratch after each price update in a phase, spending $O(m)$ time on this for each of up to n iterations. Show how to reuse work from previous iterations so that the total amount of work done searching for good paths, in total over all iterations in the phase, is only $O(m)$.

[Hint: compare to Problem 2(a).]

(b) The other non-trivial work in a price update phase is computing the value of Δ (the magnitude of the update). This is easy to do in $O(m)$ time per iteration. Explain how to maintain a heap data structure so that the total time spent computing Δ over all iterations in the phase is only $O(m \log n)$. Be sure to explain what heap operations you perform while growing the search tree and when executing a price update.

[This yields an $O(mn \log n)$ time implementation of the Hungarian algorithm.]

Problem 11

In the *minimum-cost flow problem*, the input is a directed graph $G = (V, E)$, a source $s \in V$, a sink $t \in V$, a target flow value d, and a capacity $u_e \geq 0$ and cost $c_e \in \mathbb{R}$ for each edge $e \in E$. The goal is to compute a flow $\{f_e\}_{e \in E}$ sending d units from s to t with the minimum-possible cost $\sum_{e \in E} c_e f_e$. (If there is no such flow, the algorithm should correctly report this fact.)

Given a min-cost flow instance and a feasible flow f with value d, the corresponding *residual network* G_f is defined as follows. The vertex set remains V. For every edge $(v, w) \in E$ with $f_{vw} < u_{vw}$, there is an edge (v, w) in G_f with cost c_e and residual capacity $u_e - f_e$. For every edge $(v, w) \in E$ with $f_{vw} > 0$, there is a reverse edge (w, v) in G_f with the cost $-c_e$ and residual capacity f_e.

A *negative cycle* of G_f is a directed cycle C of G_f such that the sum of the edge costs in C is negative. (E.g., $v \to w \to x \to y \to v$, with $c_{vw} = 2$, $c_{wx} = -1$, $c_{xy} = 3$, and $c_{yv} = -5$.)

(a) Prove that if the residual network G_f of a flow f has a negative cycle, then f is not a minimum-cost flow.

(b) Prove that if the residual network G_f of a flow f has no negative cycles, then f is a minimum-cost flow.

[Hint: look to the proof of the minimum-cost bipartite matching optimality conditions (Lecture #5) for inspiration.]

(c) Give a polynomial-time algorithm that, given a residual network G_f, either returns a negative cycle or correctly reports that no negative cycle exists.

[Hint: feel free to use an algorithm from CS161. Be clear about which properties of the algorithm you're using.]

(d) Assume that all edge costs and capacities are integers with magnitude at most M. Give an algorithm that is guaranteed to terminate with a minimum-cost flow and has running time polynomial in $n = |V|$, $m = |E|$, and M.[3]

[Hint: what would the analog of Ford-Fulkerson be?]

Problem 12

The goal of this problem is to revisit two problems you studied in CS161 — the minimum spanning tree and shortest path problems — and to prove the optimality of Kruskal's and Dijkstra's algorithms via the complementary slackness conditions of judiciously chosen linear programs.

[3]Thus this algorithm is only "pseudo-polynomial." A polynomial algorithm would run in time polynomial in n, m, and $\log M$. Such algorithms can be derived for the minimum-cost flow problem using additional ideas.

(a) For convenience, we consider the maximum spanning tree problem (equivalent to the minimum spanning tree problem, after multiplying everything by -1). Consider a connected undirected graph $G = (V, E)$ in which each edge e has a weight w_e.

For a subset $F \subseteq E$, let $\kappa(F)$ denote the number of connected components in the subgraph (V, F). Prove that the spanning trees of G are in an objective function-preserving one-to-one correspondence with the 0-1 feasible solutions of the following linear program (with decision variables $\{x_e\}_{e \in E}$):

$$\max \sum_{e \in E} w_e x_e$$

subject to

$$\begin{aligned} \sum_{e \in F} x_e &\le |V| - \kappa(F) \qquad \text{for all } F \subseteq E \\ \sum_{e \in E} x_e &= |V| - 1 \\ x_e &\ge 0 \qquad \text{for all } e \in E. \end{aligned}$$

(While this linear program has a huge number of constraints, we are using it purely for the analysis of Kruskal's algorithm.)

(b) What is the dual of this linear program?

(c) What are the complementary slackness conditions?

(d) Recall that Kruskal's algorithm, adapted to the current maximization setting, works as follows: do a single pass over the edges from the highest weight to lowest weight (breaking ties arbitrarily), adding an edge to the solution-so-far if and only if it creates no cycle with previously chosen edges. Prove that the corresponding solution to the linear program in (a) is in fact an optimal solution to that linear program, by exhibiting a feasible solution to the dual program in (b) such that the complementary slackness conditions hold.[4]

[Hint: for the dual variables of the form y_F, it is enough to use only those that correspond to subsets $F \subseteq E$ that comprise the i edges with the largest weights (for some i).]

(e) Now consider the problem of computing a shortest path from s to t in a directed graph $G = (V, E)$ with a nonnegative cost c_e on each edge $e \in E$. Prove that every simple s-t path of G corresponds to a 0-1 feasible solution of the following linear program with the same objective function value:[5]

$$\min \sum_{e \in E} c_e x_e$$

subject to

$$\begin{aligned} \sum_{e \in \delta^+(S)} x_e &\ge 1 \qquad \text{for all } S \subseteq V \text{ with } s \in S,\ t \notin S \\ x_e &\ge 0 \qquad \text{for all } e \in E. \end{aligned}$$

(Again, this huge linear program is for analysis only.)

(f) What is the dual of this linear program?

(g) What are the complementary slackness conditions?

[4] You can assume without proof that Kruskal's algorithm outputs a feasible solution (i.e., a spanning tree), and focus on proving its optimality.

[5] Recall that $\delta^+(S)$ denotes the edges sticking out of S.

(h) Let P denote the s-t path returned by Dijkstra's algorithm. Prove that the solution to the linear program in (e) corresponding to P is in fact an optimal solution to that linear program, by exhibiting a feasible solution to the dual program in (f) such that the complementary slackness conditions hold.

[Hint: it is enough to use only dual variables of the form y_S for subsets $S \subseteq V$ that comprise the first i vertices processed by Dijkstra's algorithm (for some i).]