CS41 Lab 10

November 9, 2021

The lab this week focuses on network flow and reductions. The purpose of this lab is to gain practice thinking about using the Ford-Fulkerson algorithm to solve other problems using **reductions**.

You should solve your algorithm design problems in lab this week by **reducing to network flow**. You should **use the Ford-Fulkerson algorithm as a subroutine** rather than making an entirely new algorithm of your own. You should not modify the Ford-Fulkerson algorithm; use it as a black box.

1. **Flow variant.** In the standard flow problem, we get an input $G = (V, E)$ a directed graph and edge capacities $c_e \geq 0$ limiting how much flow can pass along an edge. Consider the following two variants of the maximum flow problem.

   (a) It might be that each junction where water pipes meet is limited in how much water it can handle (no matter how much the pipes can carry). In this case, we want to add *vertex* capacities to our problem. The input is a directed G (with source s and sink $t \in V$), edge capacities $c_e \geq 0$, and vertex capacities $c_v \geq 0$ describing the upper limit of flow which can pass through that vertex. Give a polynomial-time algorithm to find the maximum $s \rightsquigarrow t$ flow in a network with both edge and vertex capacities. Your algorithm should be a reduction to the Ford-Fulkerson algorithm.

   (b) It might be that there are multiple sources and multiple sinks in our flow network. In this case, the input is a directed G, a list of sources $\{s_1, \ldots, s_x\} \subset V$, a list of sinks $\{t_1, \ldots, t_y\} \subset V$, and edge capacities $c_e \geq 0$.
   Give a polynomial-time algorithm to find the maximum flow in a network with multiple sources and multiple sinks. Your algorithm should be a reduction to the Ford-Fulkerson algorithm.

2. **Advertising contracts** (K& T 7.16)

   Back in the euphoric early days of the Web, people liked to claim that much of the enormous potential in a company like Yahoo! was in the "eyeballs"—the simple fact that millions of people look at its pages every day. Further, by convincing people to register personal data with the site, a site like Yahoo! can show each user an extremely targeted advertisement whenever the user visits the site, in a way that TV networks or magazines couldn't hope to match. So if a user has told Yahoo! that she is a 21-year-old computer science major at Swarthmore, the site can present a banner ad for apartments in Philadelphia suburbs; on the other hand, if she is a 50-year-old investment banker from Greenwich, CT, the site can display a banner ad pitching luxury cars instead.

   But deciding on which ads to show to which people involves some serious computation behind the scenes. Suppose that the managers of a popular site have identified k distinct *demographic groups* $G_1, G_2, \ldots, G_k$. (Some may overlap.) The site has contracts with m different advertisers to show a certain number of copies of their ads to users of the site. Here's what a contract with the i^{th} advertiser looks like:

   - For a subset $X_i \subseteq \{G_1, \ldots, G_k\}$ of the demographic groups, advertiser i wants ads shown only to users who belong to at least one of the groups listed in X_i.

- For a number r_i, advertiser i want its ads shown to at least r_i users each minute.

Consider the problem of designing a good *advertising policy* — a way to show a single ad to each user of the site. (Imagine a world where each user saw only *one* ad per site.) Suppose at a given minute, there are n users visiting the site. Because we have registration about each of the users, we know that user j belongs to a subset U_j of the demographic groups.

The problem is: is there a way to show a single ad to each user so that the site's contracts with each of the m advertisers is satisfied for this minute?

Give an efficient algorithm to decide if this is possible, and if so, to actually choose an ad to show each user.

3. **Optimization vs Decision Problems**. Recall that a decision problem requires a YES/NO answer, and an optimization problem requires the "best possible answer", which often means maximizing or minimizing over some *cost* or *score*.

   For most optimization problems, there is an obvious analogue as a decision problem. For example, consider the following problem:

   VERTEX-COVER-OPT: Given a graph $G = (V, E)$, return the size of the smallest vertex cover in G.

   VERTEX-COVER-OPT has a natural decision problem, namely VERTEX-COVER. In fact, every optimization problem can be converted to a decision problem in this way.

   (a) Show that VERTEX-COVER $\leq_{\mathrm{P}}$ VERTEX-COVER-OPT.

   (b) Let B be an arbitrary optimization problem, and let A be the decision version of B. Show that
   $$A \leq_{\mathrm{P}} B \ .$$

   (c) Show that VERTEX-COVER-OPT $\leq_{\mathrm{P}}$ VERTEX-COVER.