CS41 Lab 10: Network Flow

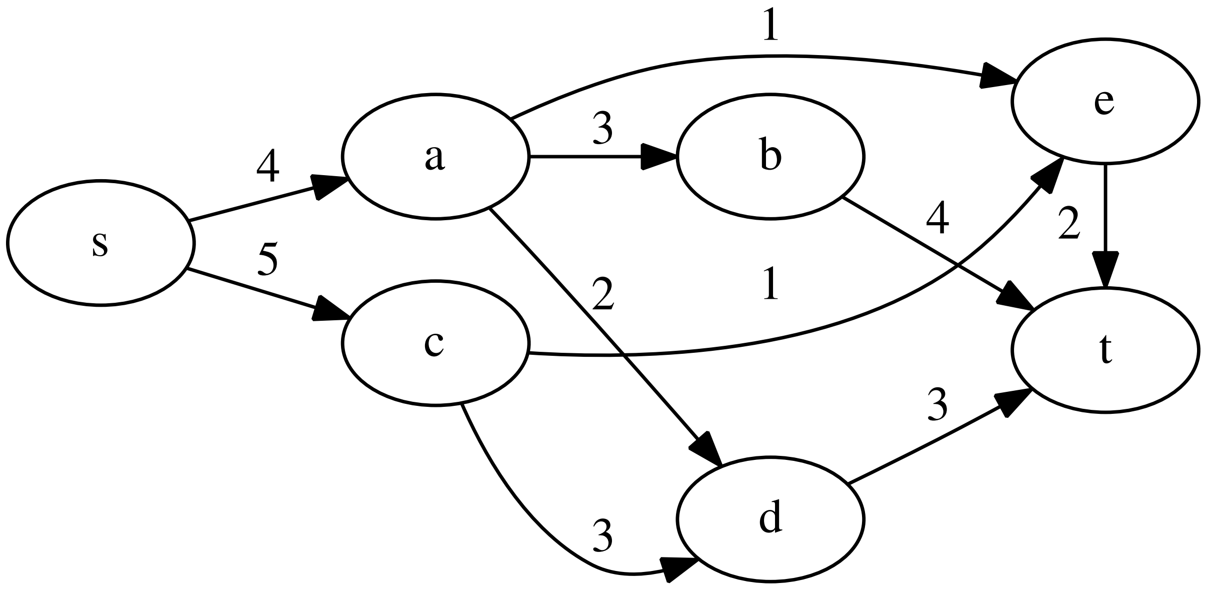
(section 2)

Last week's material had us consider how to reduce problems to each other. This concept is powerful and pervasive in algorithm design, not just for proving NP-completeness but for *reusing* already-known algorithms to solve new problems. For this week (and from now on!) you should be considering how to reduce any new problem you encounter to a problem you already know how to solve.

Blanket hint for the rest of the semester: reduce to a problem you know an algorithm for.

This week, we've considered the problem of finding maximum flow through a network.

1. **Maximum flow.** Find the maximum flow from s to t and the minimum cut between s and t in the network below. Show the residual network at intermediate steps as you build the flow.



2. **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, but there is no limit to the amount of flow that can pass *through* a vertex. This is not very realistic for flow networks in the real world, as for example, each junction where water pipes meet is limited in how much water it can handle (no matter how much the pipes can carry).

   Let's consider an extension to the maximum flow problem. The input is a directed G, 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 flow in a network with both edge and vertex capacities. (Hint: do a reduction!)

3. (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 and 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.