CS 577: Introduction to Algorithms Fall 2020

Homework 8

Instructor: Dieter van Melkebeek TA: Collin Dedrick

This homework covers network flow applications. **Problems 1 and 3 must be submitted for grading by 11:59pm on 11/9.** Problem 1 will be discussed in your discussion section on Friday. Please refer to the homework guidelines on Canvas for detailed instructions.

Warm-up problems

1. [Graded, 2 points.] Your friends are attending a convention and want to ask questions to the panelists. They worked together to create a set S of n different topics they would like to ask about. Each of your m friends has a VIP pass that guarantees them the ability to ask at most three questions each.

   Unfortunately, due to the dense subject matter of the convention, not all of your friends understand every topic well enough to ask about it. For each friend $i = 1, 2, \dots, m$, they have a set S_i of topics they are capable of asking about. Finally, to make sure each topic is thoroughly addressed, the friends want to ask at least k different questions about each of the n topics (Note that one person should not ask about the same topic more than once). They are having trouble figuring out who should ask about which topics.

   (a) Design a polynomial-time algorithm that takes the input to an instance of this problem (the n topics, the sets S_i for each of the m friends, and the parameter k) and decides whether there is a way to ask k questions about each of the n topics, while each friend asks at most three questions each.

   (b) You show your friends a solution computed by your algorithm from (a), but to your surprise, one of them exclaims, “That won’t do at all– the first topic is only asked about by Optimists!” You hadn’t heard anything about optimists; this is an extra wrinkle they neglected to mention earlier.

   Each of your friends is either an Optimist or a Pessimist. This refers to their general attitude- they each fall into exactly one of these two categories and will behave that way for the entire convention, regardless of the topic they are asking about. To keep the panel’s responses as fair as possible, your friends agree that there should be no topic for which all k questions come from people with the same attitude.

   Describe how to modify your algorithm from (a) into a new polynomial-time algorithm that decides whether there exists a solution satisfying all of the conditions from (a), plus the new requirements about optimists and pessimists.

2. The Packers are playing the Bears tonight, and you’d like to invite some of your friends to watch the game at your place. All of your friends love football, and are either Packers or Bears fans, but some of them are known not to get along very well. In order to avoid possible trouble, you do not want to invite two people who are on bad terms with each other *and* root for a different team. (Having people who are on bad terms but root for the same team is OK, as is any of the other two combinations.) Also, although you like all of your friends, you like some better than others, and you have assigned a positive value to each of your friends.

Design a polynomial-time algorithm to figure out which friends to invite so as to maximize their total value under the above constraints.

Regular problems

3. [Graded, 3 points] You're a homeowner who wants to revitalize your backyard. You make a list of n landscaping projects you want completed. There are two local landscaping companies that can do these jobs for you: Abed's Azaleas and Britta's Botanical Boutique. You can designate any subset of the tasks to each company, but every job should be done exactly once. Abed charges a_i to do job i, and Britta charges b_i for the same job. You'd like to spend as little money as possible, but you also need to account for the incompatibilities between the styles of the different suppliers. If you get jobs i and j done by different companies, there is an incompatibility cost of $c(i, j)$.

   (a) Design an efficient algorithm to determine which landscaper should do each task so as to minimize the sum of the purchase costs and incompatibility costs.

   (b) It's a slow season for the gardening industry, so these two companies are competing hard for your business. You'll be completing the jobs in the order of your list, and Abed and Britta both have a copy. They're also fiercely staking out each other's businesses, so they'll know when you purchase a service from their competitor. To try to win you back and keep the industry competitive, they both start offering you the same discount: if you recently hired their competitor, they'll offer you a discount d off your next purchase. More formally, if you purchase job i from one company, the competing company will subtract d from their price to complete job $(i + 1)$. They aren't being too generous, though: you can assume d is less than the cost of all jobs, and is also less than all $c(i, j)$.

   Describe what efficient modification you can make to your algorithm from (a) so that your minimum sum of purchase costs and incompatibility costs accounts for this discount.

4. Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems like the only option. So they decide to carpool to work.

   Unfortunately, they all hate to drive, so they want to make sure that any carpool arrangement they agree upon is fair, and doesn't overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

   Here's one way to define fairness. Let the people be labeled $S = \{p_1, \ldots, p_k\}$. We say that the *total driving obligation* of p_j over a set of days is the expected number of times that p_j would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the carpool plan lasts for d days, and on the ith day a subset $S_i \subseteq S$ of the people go to work. Then the above definition of the total driving obligation Δ_j for p_j can be written as $\Delta_j = \sum_{i:p_j \in S_i} \frac{1}{|S_i|}$. Ideally, we'd like to require that p_j drives at most Δ_j times; however, Δ_j may not be an integer.

   So let's say that a driving schedule is a choice of a driver for each day -— i.e., a sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ with $p_{i_t} \in S_t$ -— and that a fair driving schedule is one in which each p_j is chosen as the driver on at most $\lceil \Delta_j \rceil$ days.

(a) Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

(b) Design an algorithm to compute a fair driving schedule in time polynomial in k and d.

5. A *vertex cover* of a graph $G = (V, E)$ is a collection of vertices $C \subseteq V$ such that every edge $e \in E$ has at least one vertex in C.

   Show that for bipartite graphs, the minimum size of a vertex cover equals the maximum size of a matching.

Challenge problem

6. Here is a variant of the game "Six Degrees of Kevin Bacon."

   You start with a set X of n actresses and a set Y of n actors, and two players P_0 and P_1. Player P_0 names an actress $x_1 \in X$, player P_1 names an actor y_1 who has appeared in a movie with x_1, player P_0 names an actress x_2 who has appeared in a movie with y_1, and so on. Thus, P_0 and P_1 collectively generate a sequence $x_1, y_1, x_2, y_2, \ldots$ such that each actor/actress in the sequence has costarred with the actress/actor immediately preceding. A player $P_i (i = 0, 1)$ loses when it is P_i's turn to move, and she cannot name a member of her set who hasn't been named before.

   Suppose you are given a specific pair of such sets X and Y, with complete information on who has appeared in a movie with whom. A *strategy* for P_i in our setting is an algorithm that takes a current sequence $x_1, y_1, x_2, y_2, \ldots$ and generates a legal next move for P_i (assuming it's P_i's turn to move). Design a polynomial-time algorithm that, given some instance of the game, decides at the start of the the game which of the two players can force a win.

Programming problem

7. SPOJ problem Bank robbery (problem code BANKROB).