CS 577: Introduction to Algorithms Fall 2020

Homework 6

Instructor: Dieter van Melkebeek TA: Andrew Morgan

This homework covers the greedy algorithms and exchange arguments. **Problems 1 and 3 must be submitted for grading by 11:59pm on 10/26.** 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.] A small photocopying service with a single large machine to do its photocopying faces the following scheduling problem. Each morning they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer i's job will take t_i time to complete. Given a schedule (i.e., an ordering of the jobs), let C_i denote the finishing time of job i. For example, if job j is the first to be done, we would have $C_j = t_j$; and if job j is done right after job i, we would have $C_j = C_i + t_j$. Each customer i also has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

   Design an $O(n \log n)$ algorithm to solve this problem. That is, you are given a set of n jobs with a processing time t_i and a weight w_i for each job. You want to order the jobs so as to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

   Example: Suppose there are two jobs: the first takes time $t_1 = 1$ and has weight $w_1 = 10$, while the second job takes time $t_2 = 3$ and has weight $w_2 = 2$. Then doing job 1 first would yield a weighted completion time of $10 \cdot 1 + 2 \cdot 4 = 18$, while doing the second job first would yield the larger weighted completion time of $10 \cdot 4 + 2 \cdot 3 = 46$.

2. You see the following special offer by the convenience store: "A bottle of cola for every 3 empty bottles returned.", and you want to find out the maximum number of colas you can drink if you buy N bottles of cola.

   For example, consider the case where $N = 8$. The straightforward strategy is as follows: After finishing your 8 bottles of cola, you have 8 empty bottles. Take 6 of them and you get 2 new bottles of cola. Now after drinking them you have 4 empty bottles, so you take 3 of them to get yet another new cola. Finally, you have only 2 bottles in hand, so you cannot get new cola any more. Hence, you have enjoyed $8 + 2 + 1 = 11$ bottles of cola.

   You can actually do better! You first borrow an empty bottle from your friend, then you can enjoy $8 + 3 + 1 = 12$ bottles of cola! Of course, to be fair, you have to return your remaining empty bottle back to your friend which you can do because you will have one left over bottle after drinking the final cola you get from the store.

   Design an algorithm to finds the largest number of bottles you can get when buying N of them and can borrow as many bottles as you want. Your algorithm should run in time bounded by a polynomial in the bit length of N, i.e., a polynomial in $\log N$.

Regular Problems

3. [Graded, 3 points] There are n items you want to buy. Each item i is advertised at a price of $p_i \geq 0$ dollars. However, there is a catch: to get item i at that price, you must first buy it at a cost of $c_i \geq p_i$ dollars and then redeem a rebate coupon. You may redeem the rebate for an item right after you buy it, but you do have to first buy the item at full cost c_i in order to qualify. What is the smallest initial amount of funding that you need in order to buy all n items? In what order should you buy them?

   Design an $O(n \log n)$ algorithm for this problem.

4. In a city there are n bus drivers. There are also n morning bus routes and n evening bus routes, each with various lengths. Each driver is assigned one morning route and one evening route. For any driver, if his total route length for a day exceeds d, he has to be paid overtime for every hour after the first d hours at a fixed rate per hour. Your task is to assign one morning route and one evening route to each bus driver so that the total overtime amount that the city authority has to pay is minimized.

   Design an $O(n \log n)$ algorithm for this problem.

5. Each morning you drive from your house to work. The route decomposes to n parts each of length $l_i > 0$. At each part there is a different speed limit $v_i > 0$. As usual, you are late for work and decide to break the speed limit in order to arrive faster. Since you do not want to risk much you decide to only break the speed limit by v at each part and only for k parts overall. For example, your route may be the list

   $$[(30\text{mi}, 20\text{mph}), (40\text{mi}, 70\text{mph}), (10\text{mi}, 30\text{mph})],$$

   where each tuple contains the length l_i and the speed limit v_i of each part. You decide that you only want to break the speed limit 2 times, that is $k = 2$ and by speed $v = 10$mph. A possible way to do this route is for example to drive 30 mph at l_1 and 40 mph at l_3. Remember that since you always travel at a *constant* speed during each part of your route you the time t_i to travel part i is l_i/v_i if you travel according to the speed limit or $l_i/(v_i + v)$ if you decide to use the extra speed v. Since you want to optimize the time of your commute, you want to find at which parts of the route you should break the speed limit and use the extra speed v. You have many ideas on how you should use the extra speed.

   (a) A reasonable choice is to use it on the longest parts of the route, that is sort the list with respect to l_i (largest first) and use the bonus speed on the first k parts of the sorted list. Provide a counter-example for this greedy strategy.

   (b) Another option is to use it on the parts of the route with smallest speed limit v_i, that is sort the list with respect to v_i and use the bonus speed on the first k parts of the sorted list. Provide a counter-example for this greedy strategy.

   (c) Design an $O(n \log n)$ algorithm that takes as input the list of lengths and speed limits

   $$L = [(l_1, v_1), \ldots (l_n, v_n)]$$

   and two positive integers k, v and computes the minimum time to go from your house to work. You need to provide a proof that the algorithm finds the minimum travel time.

Challenge problem

6. Suppose you are given a connected graph G in which the edge weights depend on a parameter t, where t ranges over the reals. More specifically, the cost of an edge e is given by a function of the form $c_e(t) = a_e t^2 + b_e t + d_e$, where a_e, b_e, d_e are reals depending on e, and $a_e > 0$. Your goal is to find the minimum cost of a minimum spanning tree of G over all values of t. Your algorithm should run in time polynomial in the number of nodes and edges of G. You may assume that you can perform standard arithmetic operations (including computing square roots) at unit cost.

   Would your approach also work for the shortest paths problem (assuming the weight functions are nonnegative over their entire range)?

Programming problem

7. SPOJ problem Island Hopping (problem code ISLHOP)