Algorithms: CSE 202 — Homework II

Problem 1: Nesting Boxes (CLRS)

A d-dimensional box with dimensions $(x_1, x_2, \ldots, x_d)$ *nests* within another box with dimensions $(y_1, y_2, \ldots, y_d)$ if there exists a permutation π on $\{1, 2, \ldots, d\}$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(d)} < y_d$.

1. Argue that the nesting relation is transitive.
2. Describe an efficient method to determine whether or not one d-dimensional box nests inside another.
3. Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Describe an efficient algorithm to determine the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k}\rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1\rangle$. Express the running time of your algorithm in terms of n and d.

Problem 2: Business plan

Consider the following problem. You are designing the business plan for a start-up company. You have identified n possible projects for your company, and for, $1 \leq i \leq n$, let $c_i > 0$ be the minimum capital required to start the project i and $p_i > 0$ be the profit after the project is completed. You also know your initial capital $C_0 > 0$. You want to perform at most k, $1 \leq k \leq n$, projects before the IPO and want to maximize your total capital at the IPO. Your company cannot perform the same project twice.

In other words, you want to pick a list of up to k distinct Projects, $i_1, \ldots, i_{k'}$ with $k' \leq k$. Your *accumulated capital* after completing the project i_j will be $C_j = C_0 + \sum_{h=1}^{h=j} p_{i_h}$. The sequence must satisfy the constraint that you have sufficient capital to start the project i_{j+1} after completing the first j projects, i.e., $C_j \geq c_{i_{j+1}}$ for each $j = 0, \ldots, k'-1$. You want to maximize the final amount of capital, $C_{k'}$.

Problem 3: Scheduling to minimize average completion time (CLRS)

Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time. Let c_i be the *completion time* of task a_i, that is, the time at which task a_i completes processing. Your goal is to minimize the average completion time, that is, to minimize $(1/n)\sum_{i=1}^{n} c_i$. For example, suppose there are two tasks, a_1 and a_2, with $p_1 = 3$ and $p_2 = 5$, and consider the schedule in which a_2 runs first, followed by a_1. Then $c_2 = 5$, $c_1 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$.

1. Give an algorithm that schedules the tasks so as to minimize the average completion time. Each task must run non-preemtively, that is, once task a_i is started, it must run continuously for p_i units of time. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.
2. Suppose now that the tasks are not all available at once. That is, each task has a *release time* r_i before which it is not available to be processed. Suppose also that we allow *preemption*, so that a task can be suspended and restarted at a later time. For example, a task a_i with processing time $p_i = 6$ may start running at time 1 and be preempted at time 4. It can then resume at time 10 but be preempted at time 11 and finally resume at time 13 and complete at time 15. Task a_i has run for a total of 6 time units, but its running time has been divided into three pieces. We say that the completion time of a_i is 15. Give an algorithm that schedules the tasks so as to minimize the average completion time in this new scenario. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.

Problem 4: Shortest wireless path sequence (KT 6.14)

A large collection of mobile wireless devices can naturally form a network in which the devices are the nodes, and two devices x and y are connected by an edge if they are able to directly communicate with each other (e.g., by a short-range radio link). Such a network of wireless devices is a highly dynamic object, in which edges can appear and disappear over time as the devices move around. For instance, an edge (x, y) might disappear as x and y move far apart from each other and lose the ability to communicate directly.

In a network that changes over time, it is natural to look for efficient ways of *maintaining* a path between certain designated nodes. There are two opposing concerns in maintaining such a path: we want paths that are short, but we also do not want to have to change the path frequently as the network structure changes. (That is, we'd like a single path to continue working, if possible, even as the network gains and loses edges.) Here is a way we might model this problem.

Suppose we have a set of mobile nodes V, and at a particular point in time there is a set E_0 of edges among these nodes. As the nodes move, the set of edges changes from E_0 to E_1, then to E_2, then to E_3, and so on, to an edge set E_b. For $i = 0, 1, 2, \ldots, b$, let G_i denote the graph (V, E_i). So if we were to watch the structure of the network on the nodes V as a "time lapse", it would look precisely like the sequence of graphs $G_0, G_1, G_2, \ldots, G_{b-1}, G_b$. We will assume that each of these graphs G_i is connected.

Now consider two particular nodes $s, t \in V$. For an s-t path P in one of the graphs G_i, we define the *length* of P to be simply the number of edges in P, and we denote this $\ell(P)$. Our goal is to produce a sequence of paths $P_0, P_1, \ldots, P_b$ so that for each i, P_i is an s-t path in G_i. We want the paths to be relatively short. We also do not want there to be too many *changes*–points at which the identity of the path switches. Formally, we define $changes(P_0, P_1, \ldots, P_b)$ to be the number of indices i $(0 \leq i \leq b-1)$ for which $P_i \neq P_{i+1}$.

Fix a constant $K > 0$. We define the cost of the sequence of paths $P_0, P_1, \ldots, P_b$ to be

$$cost(P_0, P_1, \ldots, P_b) = \sum_{i=0}^{b} \ell(P_i) + K \cdot changes(P_0, P_1, \ldots, P_b).$$

1. Suppose it is possible to choose a single path P that is an s-t path in each of the graphs $G_0, G_1, \ldots, G_b$. Give a polynomial-time algorithm to find the shortest such path.

2. Give a polynomial-time algorithm to find a sequence of paths $P_0, P_1, \ldots, P_b$ of minimum cost, where P_i is an s-t path in G_i for $i = 0, 1, \ldots, b$.

Problem 5: Selling shares of stock (KT 6.25)

Consider the problem faced by a stockbroker trying to sell a large number of shares of stock in a company whose stock price has been steadily falling in value. It is always hard to predict the right moment to sell stock, but owning a lot of shares in a single company adds an extra complication: the mere act of selling many shares in a single day will have an adverse effect on the price.

Since future market prices, and the effect of large sales on these prices, are very hard to predict, brokerage firms use models of the market to help them make such decisions. In this problem, we will consider the following simple model. Suppose we need to sell x shares of stock in a company, and suppose that we have an accurate model of the market: it predicts that the stock price will take the values $p_1, p_2, \ldots, p_n$ over the next n days. Moreover, there is a function $f(\cdot)$ that predicts the effect of large sales: if we sell y shares on a single day, it will permanently decrease the price by $f(y)$ from that day onward. So, if we sell y_1 shares on day 1, we obtain a price per share of $p_1 - f(y_1)$, for a total income of $y_1 \cdot (p_1 - f(y_1))$. Having sold y_1 shares on day 1, we can then sell y_2 shares on day 2 for a price per share of $p_2 - f(y_1) - f(y_2)$; this yields an additional income of $y_2 \cdot (p_2 - f(y_1) - f(y_2))$. This process continues over all n days. (Note, as in our calculation for day 2, that the decreases from earlier days are absorbed into the prices for all later days.)

Design an efficient algorithm that takes the prices $p_1, \ldots, p_n$ and the function $f(\cdot)$ (written as a list of values $f(1), f(2), \ldots, f(x)$) and determines the best way to sell x shares by day n. In other words, find natural numbers $y_1, y_2, \ldots, y_n$ so that $x = y_1 + \ldots + y_n$, and selling y_i shares on day i for $i = 1, 2, \ldots, n$ maximizes the total income achievable. You should assume that the share value p_i is monotone decreasing, and $f(\cdot)$ is monotone increasing; that is, selling a larger number of shares causes a larger drop in the price.

Your algorithms running time can have a polynomial dependence on n (the number of days), x (the number of shares), and p_1 (the peak price of the stock).

Example Consider the case when $n = 3$; the prices for the three days are 90, 80, 40; and $f(y) = 1$ for $y \leq 40,000$ and $f(y) = 20$ for $y > 40,000$. Assume you start with $x = 100,000$ shares. Selling all of them on day 1 would yield a price of 70 per share, for a total income of 7,000,000. On the other hand, selling 40,000 shares on day 1 yields a price of 89 per share, and selling the remaining 60,000 shares on day 2 results in a price of 59 per share, for a total income of 7,100,000.