Due: 6 pm on Wednesday, March 12

Assignment 3

1 Commuting to Maximize Profit

Suppose you run a company with two offices, one in Beijing and the other in Shanghai. Each week, you must choose where to work, and your choice will affect your profit: in week i, you will earn a_i CNY (China Yuan) if you work in Beijing, or b_i CNY if you work in Shanghai. While you clearly would like to work in the location with the higher profit, each train ticket between Beijing and Shanghai (for either direction) costs 500 CNY. Given the lists $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, your goal is to find a schedule that maximizes your total profit, taking commuting costs into account. Since you live in Beijing, your schedule must start and end there.

1) A natural greedy strategy is to always work in the office with the higher profit. Give a (minimal) example showing that this strategy is not always optimal.

2) Design a polynomial-time algorithm in n that finds the maximum total profit. Justify your algorithm's **correctness** and establish its **running time**. To get full credit for your solution, you should make your running time as small as possible.

2 Investing in the Stock

Suppose you are a wise investor looking at n consecutive days of a given stock, from some point in the past. The days are numbered $i = 1, 2, \ldots, n$; for each day i, there's a price $p(i)$ per share for the stock on that day. For certain (possibly large) values of k, you want to study a so-called *k-shot strategies*. The strategy is a collection of m pairs of days $(b_1, s_1), \ldots, (b_m, s_m)$, where $0 \leqslant m \leqslant k$ and $1 \leqslant b_1 < s_1 < b_2 < s_2 < \ldots < b_m < s_m \leqslant n$. We view these as a set of up to k non-overlapping intervals, during each of which you will buy 1000 shares of the stock (on day b_i) and then sell it (on day s_i). The *return* of a given k-shot strategy is simply the profit obtained from the m buy-sell transactions, namely,

$$1000 \sum_{i=1}^{m} \left(p(s_i) - p(b_i)\right).$$

Your goal is to design an efficient algorithm that determines, given the sequence of prices, the k-shot strategy with the maximum possible return. Prove the **correctness** and analyze the **running time** of your algorithm. Your running time should be polynomial in both n and k; it should not contain k in the exponent.

3 Number of Shortest Paths

To assess how "well-connected" two nodes in a directed graph are, we may not only look at the length of the shortest path between them, but can also count the number of shortest paths. Suppose we are given a directed graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, with cost c_e on each edge $e \in E$. The costs may be positive or negative, but every cycle in the graph has strictly positive cost. We are also given two nodes $s, t \in V$. Give an $O(mn)$ algorithm that computes the number of shortest s-t paths in G. (You don't have to list all the paths; just the number suffices.)

4 Running a Sales Company

You are running a company that sells trucks, and predictions tell you the quantity of sales to expect over the next n months. Let d_i denote the number of sales you expect in month i. We'll assume that all sales happen at the beginning of the month, and trucks not sold are *stored* until the beginning of the next month. You can

store at most S trucks, and it costs C to store a single truck for a month. You receive shipments of trucks by placing orders for them, and there is fixed ordering fee of K each time you place an order (regardless of the number of trucks you order). You start out with no trucks. The problem is to design an algorithm that decides how to place orders so that you satisfy all the demands $\{d_i\}$, and minimize the costs. In summary:

- There are two parts of the cost: 1) storage—it costs C for every truck on hand that is not needed that month; (2) ordering fees—it costs K for every order placed.
- In each month you need enough trucks to satisfy the demand d_i, but the number left over after satisfying the demand for the month should not exceed the inventory limit S.

Give an algorithm that solves this problem in $O(nS)$ time.