Homework 3

Name: 方嘉聪 ID: 2200017849

Problem 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.

◀

Answer. (1) 考虑 $\{a_i\} = \{1, 400, 1\}$ 和 $\{b_i\} = \{400, 1, 400\}$, 那么朴素贪心算法得到的总收益为 -800, 一直待在北京的总收益为 402, 显然朴素贪心算法不是最优的.

(2) 我们可以用动态规划来解决这个问题. 设 $Opt(i, 0)$ 和 $Opt(i, 1)$ 分别表示第 i 周在北京或上海工作的最大收益. 那么我们有如下的状态转移方程:

$$
\begin{aligned}
&Opt(1,0) = a_1, \quad Opt(1,1) = b_1 - 500, \\
&Opt(i,0) = a_i + \max\{Opt(i-1,0), Opt(i-1,1) - 500\} \quad \text{for } i = 2, 3, \ldots, n, \\
&Opt(i,1) = b_i + \max\{Opt(i-1,1), Opt(i-1,0) - 500\} \quad \text{for } i = 2, 3, \ldots, n.
\end{aligned}
$$

伪代码见下:

Algorithm 1 Maximize Profit by Dynamic Programming

Input: Two lists of profits $\{a_i\}$ and $\{b_i\}$;

Output: The maximum total profit;

```
Opt[1][0] ← a_1, Opt[1][1] ← b_1 − 500
for i ← 2 to n do
    Opt[i][0] ← a_i + max{Opt[i − 1][0], Opt[i − 1][1] − 500}
    Opt[i][1] ← b_i + max{Opt[i − 1][1], Opt[i − 1][0] − 500}
return max{Opt[n][0], Opt[n][1] − 500}
                    // Consider the last week work in Shanghai and then go back to Beijing.
```

时间复杂度为 $O(n)$. 算法正确性: 我们来证明这一问题具有最优子结构性质且无后效性.

注意到 $n = 1$ 时, 最优性显然成立. 假设 $Opt(i-1,0)$ 和 $Opt(i-1,1)$ 是最优的, 那么 $Opt(i,0)$ 和 $Opt(i,1)$ 也是最优的. 否则,

$$\exists\, Opt'(i-1,0), Opt'(i-1,1)\ s.t.\ Opt'(i-1,0) > Opt(i-1,0) \vee Opt'(i-1,1) > Opt(i-1,1)$$

这与假设的最优性矛盾. 又注意到每次分解出的子问题是独立的, 无后效性, 故算法正确性得证.

◁

Problem 2 (Processing Sheet Metal). You run a company that processes (two-dimensional) sheet metal. You have a price list indicating that a rectangular piece of sheet metal of dimensions $x_i \times y_i$ can be sold for v_i dollars, where $i \in \{1, 2, \ldots, n\}$. Starting from a raw piece of sheet metal of dimensions $A \times B$, you would like to make a sequence of horizontal or vertical cuts, each of which cuts a given piece into two smaller pieces, to produce pieces from your price list (and possibly some additional worthless scrap pieces). Design an algorithm that maximizes the total value of the pieces obtained by some sequence of cuts, prove that your algorithm is **correct**, and analyze its **running time**.
Note that:

- Your algorithm should return **both the maximum value that can be obtained and a description of the cuts that should be made to achieve it**.
- The numbers A, B, and x_i, y_i, v_i for $i \in \{1, 2, \ldots, n\}$ are all positive integers.
- You can produce any number of copies of any of the pieces, i.e., no matter how many pieces of $x_i \times y_i$ you have produced, they can always be sold at price v_i.

◀

Answer. 设 $Opt(i,j)$ 表示形状为 $x_i \times y_j$ 的矩形的最大收益, 考虑遍历所有的切割方式, 那么我们有如下的状态转移方程:

$$Opt(i,j) = \max_{1 \le k < i} \{Opt(k,j) + Opt(i-k,j)\}, \quad \forall i \in \{1,2,\ldots,A\}, j \in \{1,2,\ldots,B\}.$$
$$Opt(i,j) = \max_{1 \le k < j} \{Opt(i,k) + Opt(i,j-k)\}, \quad \forall i \in \{1,2,\ldots,A\}, j \in \{1,2,\ldots,B\}.$$

用一个二维数组 Cut 来记录切割的位置, 其中 $Cut[i][j]$ 表示形状为 $x_i \times y_j$ 的矩形的切割位置 (可以建立一个结构体), 那么我们有如下的伪代码:

Algorithm 2 Maximize the Total Value of the Pieces

Input: The dimensions of the raw piece $A \times B$, the price list $\{(x_i, y_i, v_i)\}_{i=1}^n$;
Output: The maximum total value and a description of the cuts;

```
Initialize Opt[i][j] = 0, ∀i ∈ {1,2,...,A}, j ∈ {1,2,...,B};
Initialize Cut[i][j] = ∅, ∀i ∈ {1,2,...,A}, j ∈ {1,2,...,B};
Assign Opt with Base Case {x_i, y_i, v_i}_{i=1}^n;          // Base Case: O(n)
for i ← 1 to A do
    for j ← 1 to B do
        for k ← 1 to i − 1 do
```

```
        Opt[i][j] ← max{Opt[i][j], Opt[k][j] + Opt[i − k][j]}
        Cut[i][j] ← coordinates of the cut that achieves the maximum
    for k ← 1 to j − 1 do
        Opt[i][j] ← max{Opt[i][j], Opt[i][k] + Opt[i][j − k]}
        Cut[i][j] ← coordinates of the cut that achieves the maximum
return Opt[A][B] and Cut[A][B]
```

我们用数学归纳法证明算法的正确性:

1. 归纳奠基: 对于 $Opt(x_i, y_i), \forall i \in \{1, 2, \ldots, n\}$ 显然是符合最优性的.
2. 归纳假设: 假设对于任意尺寸小于等于 $i \times j$ 的矩形, 算法得到的最大收益是最优的, 下面我们证明对于 $i \times (j+1)$ 和 $(i+1) \times j$ 的矩形, 算法得到的最大收益是最优的.
3. 归纳证明: 注意到算法考虑了 $i \times (j+1)$(或 $(i+1) \times j$) 的所有切割后的子问题 $i - k_i \times (j+1-k_j)$, 由归纳假设知这些子问题都已得到最优解, 故算法得到的最大收益是最优的.
4. 终止条件: Opt 二维数组是有限的, 算法显然能终止, 且最终选择了 $Opt[A][B]$ 和 $Cut[A][B]$.

综上所述, 算法的正确性得证. 时间复杂度为 $O\left(\sum_{i=1}^{A}\sum_{j=1}^{B}(i+j)+n\right) = O(AB(A+B)+n)$. ◁

Problem 3 (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 $0 \le m \le k$ and $1 \le b_1 < s_1 < b_2 < s_2 < \cdots < b_m < s_m \le n$. 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. ◀

Answer. 设 $Opt(i, j)$ 为前 i 天进行 j 笔交易的最大收益, 那么对 $\forall i \in \{2, 3, \ldots, n\}, j \in \{1, 2, \ldots, k\}$ 我们有如下的状态转移方程:

$$Opt(i, j) = \max\{Opt(i-1, j), \max_{1 \le l < i}\{Opt(l, j-1) + 1000(p(i) - p(l))\}\}, \tag{1}$$

初始状态为:

$$Opt(1, j) = 0, \quad \forall j \in \{0, 1, 2, \ldots, k\},$$
$$Opt(i, 0) = 0, \quad \forall i \in \{1, 2, \ldots, n\}$$

其中(??)式是由于我们可以选择是否在第 i 天卖出股票, 若是则我们可以通过遍历的方法在前 $i-1$ 天中选择一个最优的买入时机. 若否则即为 $Opt(i-1,j)$. 那么伪代码如下:

Algorithm 3 Maximize the Return of Stock

Input: The sequence of prices $\{p(i)\}$, the number of transactions k;

Output: The maximum return of stock;

1: $Opt[1][j] \leftarrow 0,\ \forall j \in \{0,1,2,\ldots,k\}$
2: $Opt[i][0] \leftarrow 0,\ \forall i \in \{1,2,\ldots,n\}$
3: **for** $i \leftarrow 2$ to n **do**
4: **for** $j \leftarrow 1$ to k **do**
5: $Opt[i][j] \leftarrow \max\{Opt[i-1][j], \max_{1\le l<i}\{Opt[l][j-1]+1000(p[i]-p[l])\}\}$
6: **return** $\max_{j\in\{0,1,2,\ldots,k\}}\{Opt[n][j]\}$

由上述的讨论可知, 该算法的时间复杂度为 $O(n^2k)$. 下面我们用数学归纳法证明算法的正确性:

1. 归纳奠基: $Opt(1,j)=0, \forall j \in \{0,1,\ldots,k\}$ 显然是最优的.
2. 归纳假设: 假设 $Opt(l,j-1), \forall l<i, j\le k$ 是最优的, 下面我们证明 $Opt(i,j)$ 是最优的.
3. 归纳步骤: 注意到 $Opt(i,j)$ 考虑是否在第 i 天卖出股票, 根据归纳假设 $Opt(l,j-1)$ 都是最优的, 通过上述的归纳转移方程知 $Opt(i,j)$ 考虑了所有的买入时机 (l), 故 $Opt(i,j)$ 是最优的.
4. 终止条件: Opt 二维数组是有限的, 算法显然能终止, 且最终选择了 $\max_j\{Opt[n][j]\}$, 考虑了最后一天所有的可能情况的最大收益.

综上所述, 算法的正确性得证.

这里也给出问题的另一种算法:

设 $buy(i,j)$ 表示前 i 天恰好进行 j 笔交易且当前手中持有股票的最大收益, $sell(i,j)$ 表示前 i 天恰好进行 j 笔交易且当前手中不持有股票的最大收益. 那么我们有如下的状态转移方程:

$$buy(i,j) = \max\{buy(i-1,j), sell(i-1,j) - 1000p(i)\}, \tag{2}$$

$$sell(i,j) = \max\{sell(i-1,j), buy(i-1,j-1) + 1000p(i)\}. \tag{3}$$

其中(??)式表示手中的股票是否是在第 i 天买入的, 若否则即为 $buy(i-1,j)$, 若是则对应第 $i-1$ 天不持有股票 (即为 $sell(i-1,j)$, 注意需要扣除 $1000p(i)$), 取二者最大即可. (??)式表示手中的股票是否是在第 i 天卖出的, 若否则即为 $sell(i-1,j)$, 若是则对应第 $i-1$ 天持有股票 (即为 $buy(i-1,j-1)$, 注意需要加上 $1000p(i)$), 取二者最大即可. 伪代码如下:

Algorithm 4 Maximize the Return of Stock

Input: The sequence of prices $\{p(i)\}$, the number of transactions k;

Output: The maximum return of stock;

1: $buy[1][j] \leftarrow -1000p(1),\ \forall j \in \{0,1,2,\ldots,k\}$
2: $sell[1][j] \leftarrow 0,\ \forall j \in \{0,1,2,\ldots,k\}$
3: **for** $i \leftarrow 2$ to n **do**
4: **for** $j \leftarrow 1$ to k **do**
5: $buy[i][j] \leftarrow \max\{buy[i-1][j], sell[i-1][j] - 1000p[i]\}$
6: $sell[i][j] \leftarrow \max\{sell[i-1][j], buy[i-1][j-1] + 1000p[i]\}$

7: **return** $\max_{j\in\{0,1,2,\ldots,k\}}\{sell[n][j]\}$

这一算法的时间复杂度为 $O(nk)$. 算法正确性证明类似上面的归纳法, 这里略去. ◁

Problem 4 (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 maybe 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.) ◀

Answer. 从 *Bellman Ford* 算法出发加以修改. 设 $Opt(i,\omega)$ 为 $s \to \omega$ 的边数恰为 i 的最短路径长度, $N(i,\omega)$ 为上述路径的条数, $Opt(t)$ 为 $s \to t$ 的最短路径长度, $N(t)$ 为对应长度的路径的条数. 那么根据定义有如下初始化条件 (初始条件下, $s \to s$ 的最短路径长度为 0, 路径条数为 1):

$$Opt(i,s) = 0, \quad Opt(i,s') = +\infty, \quad \forall s' \neq s,$$
$$N(i,s) = 1, \quad N(i,s') = 0, \quad \forall s' \neq s.$$

进一步有如下状态转移方程 ($\forall i \in \{1,2,\ldots,n-1\}$):

$$Opt(i,r) = \min_{w,(w,r)\in E}\{Opt(i-1,w) + c_{(w,r)}\}, \quad \forall r \in V, \tag{4}$$

$$N(i,r) = \sum_{w,(w,r)\in E}\{N(i-1,w)\cdot \mathbb{I}(Opt(i-1,w) + c_{(w,r)} = Opt(i,r))\}, \quad \forall r \in V. \tag{5}$$

其中 $\mathbb{I}(\cdot)$ 为指示函数. (**??**)式计算了与已知最短路同长度的路径的个数.
类似 *Bellman Ford* 算法, 时间复杂度为

$$O\left(n\sum_{v\in V} n_v\right) = O(mn)$$

其中 n_v 为节点 v 的出度. 接着有:

$$Opt(t) = \min_i\{Opt(i,t)\}, \tag{6}$$

$$N(t) = \sum_i\{N(i,t)\cdot \mathbb{I}(Opt(i,t) = Opt(t))\}. \tag{7}$$

最后的结果为 $N(t)$. 时间复杂度为 $O(n)$. 伪代码如下:

Algorithm 5 Number of Shortest Paths

Input: A directed graph $G = (V, E)$, two nodes $s, t \in V$

Output: The number of shortest $s - t$ paths

1: $Opt[0][s] \leftarrow 0,\ Opt[0][s'] \leftarrow +\infty, N[0][s] \leftarrow 1,\ N[0][s'] \leftarrow 0,\ \forall s' \neq s$

2: **for** $i \leftarrow 1$ *to* $n-1$ **do**

3: **for** $r \in V$ **do**
4: $Opt[i][r] \leftarrow +\infty$, $N[i][r] \leftarrow 0$
5: $Opt[i][r] \leftarrow \min_{w,(w,r)\in E}\{Opt(i-1,w) + c_{(w,r)}\}$
6: $N[i][r] \leftarrow \sum_{w,(w,r)\in E \wedge Opt[i][r]=Opt[i][w]+c_{(w,r)}} N[i-1][w]$
7: $Opt \leftarrow \min_i\{Opt[i][t]\}$
8: $N \leftarrow \sum_{i\wedge Opt[i][t]=Opt} N[i][t]$
9: **return** N

由上述的讨论可知, 该算法的时间复杂度为 $O(mn)$. ◁

Problem 5 (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. ◀

Answer. 设 $Opt(i,s)$ 表示第 i 个月**开始时**剩余 s 辆车的最小花费. 那么我们所要求的是 $Opt(n,0)$, 总大小是 $n(S+1)$ 的表格. 设第 $i-1$ 天剩余车辆为 z, 下面我们分情况讨论:

1. 初始值: $Opt(1,0) = 0,\ Opt(1,j) = +\infty,\ \forall j \in \{1,2,\ldots,S\}$.
2. $0 \le z < \min\{s+d_i, S\}$ 时, 说明我们需要进货, 那么 ($\forall i \in \{2,3,\ldots,n\}$)
$$Opt_1(i,s) = \min_{z<\min(s+d_i,S)}\{Opt(i-1,z) + K + zC\} \tag{8}$$
3. $s + d_i \le z \le S$ 时, 说明我们不需要进货, 那么 ($\forall i \in \{2,3,\ldots,n\}$)
$$\begin{aligned} Opt_2(i,s) &= \min_{s+d_i\le z\le S}\{Opt(i-1,z) + zC\} \\ \Longrightarrow\ Opt(i,s) &= \min\{Opt_1(i,s), Opt_2(i,s)\} \end{aligned} \tag{9}$$

对于(**??**)式, 注意到不论第 $i-1$ 个月剩余车辆是多少, 我们都需要进货, 那么当 $z=0$ 时取到最小值 (这是由于多出来的货车会额外有储存的成本). 对于(**??**)式, 注意到不论第 $i-1$ 个月剩余车辆是多少, 我们都不需要进货, 那么类似的当 $z = s + d_i$ 时取到最小值. 因此

$$Opt(i,s) = \begin{cases} \min\{Opt(i-1,0)+K, Opt(i-1,s+d_i)+(s+d_i)C\}, & \text{if } s+d_i \le S, \\ Opt(i-1,0)+K, & \text{if } s+d_i > S. \end{cases}$$

综合上述讨论, 我们可以得到如下的伪代码:

Algorithm 6 Minimize the Costs of Ordering Trucks

Input: The demands $\{d_i\}$, the storage limit S, the storage cost C, the ordering fee K

Output: The minimum cost

```
Opt[1][0] ← 0, Opt[1][j] ← +∞, ∀j ∈ {1, 2, ..., S}
for i ← 2 to n do
    for s ← 0 to S do
        if s + d_i ≤ S then
            Opt[i][s] ← min{Opt(i − 1, 0) + K, Opt(i − 1, s + d_i) + (s + d_i)C}
        else
            Opt[i][s] ← Opt[i − 1][0] + K
return Opt[n][0]
```

由于循环内部每次计算 $Opt[i][s]$ 均为常数时间, 故总的时间复杂度为 $O(nS)$. ◁