演算法 DIY

B09502019 石博允

May 3, 2022

我從高中升大學的暑假開始，跟幾個高中同學一起寫了一系列考高中科學/數資班的參考書，並放在蝦皮上賣。而在我們之中，我是負責包裝並寄書的人，因爲我家離印書的地方有一點距離，所以我常常懶得去捕貨，這使得常常會發生有人下訂單，但我的庫存不夠，沒辦法全部寄出的情況。遇到這種情況，我就會想辦法寄出盡量多的訂單，其他訂單只好等我去補貨後再出貨。這種情況在最近特別常發生(因爲數資班考試快到了)，剛好上次又聽教授講到 DIY 要寫用演算法解決現實遇到的問題，所以我就想到了這件事。

首先先定義好問題：假設我們目前有 n 本不同的書在販售（實際上 $n = 12$)，我目前每本的庫存是 $a_1 \sim a_n$，此時我接到了 m 筆有時間先後順序的訂單，每單都有可能有任意數量任意種類的書，方便起見，令第 k 單中總共有 b_k 本，且其中第 1 本到第 n 本書分別需要 $b_{k1} \sim b_{kn}$ 本，目標是找到一個訂單的集合 S，使得 S 滿足我決定寄書優先順序的標準。我有以下兩種可行的寄書標準：

1. 依照下訂單的時間順序，先訂的優先，並以目前有的庫存完成最多份訂單。
2. 不論順序，以目前有的庫存完成最多份訂單。
3. 不論順序，以目前有的庫存寄出最多本書。

下面討論使用這兩種標準時，分別能用怎樣的演算法來得到如何選擇訂單。

1 依照時間，寄出最多訂單

以這種標準來決定時，只需要利用 greedy 的方式，將所有訂單依照時間排序，接著從第一張訂單開始檢查，若對於所有 n，$b_{1n} < a_n$，則把此訂單加入 S，並讓 $a_n = a_n - b_{1n}$，反之則跳過此訂單，以此類推，把每張訂單都檢查過一次，就可以得到目標的 S。過程中 m 張訂單都要檢查一次，每次要檢查 n 本書，因此時間複雜度爲 $O(mn)$。

2 不論順序，寄出最多訂單

以這種標準決定寄書順序的方法比較難，我想了蠻久的，但沒有想到特別快的方法。如果直接使用暴力解法，把所有情況窮舉，會需要考慮 2^m 種情況（每張訂單選或不選），其中每種情況還需要檢查 n 本書，因此時間複雜度爲 $O(2^m \times n)$。

另一種我想到的方法是利用 DP 的方法，令

$$T(m, a_1, a_2, \cdots, a_n)$$

代表當第一本書到第 n 本書分別有 $a_1 \sim a_n$ 時，在第 1 張到第 m 張訂單選出的最佳解共有幾筆訂單。則可以寫出遞迴式

$$\begin{aligned} &T(m, a_1, \cdots, a_n) \\ =&\max\left(T(m-1, a_1, \cdots, a_n), T(m-1, a_1 - b_{m1}, \cdots, a_n - b_{mn}) + 1\right), \end{aligned}$$

boundary condition 爲

$$\begin{cases} T(0, a_1, a_2, \cdots, a_n) = 0 & , a_i \geq 0 \ \forall i \\ T(m, a_1, a_2, \cdots, a_n) = -\infty & , a_i < 0 \text{ for some } i. \end{cases}$$

上式代表對於每筆訂單，都有選或不選兩種選擇，如果選了就可以多一單，但庫存要減少，不選的話就代表要從剩下的訂單中選出最好的

結果，當庫存不夠時，T 的值會是負無窮大，因此不會選擇讓庫存不夠的訂單。

利用這個遞迴關係配合 memoization，最多會需要填

$$m \times a_1 \times a_2 \times \cdots \times a_n$$

個格子。填入每個格子需要的時間是 $O(1)$，因此總時間複雜度爲 $O(m \times \Pi_{i=0}^{n} a_i)$。

這個時間複雜度在實際上應該會比暴力解法還要快，因爲我家裡的每本書庫存都在 30 本以內，也就是 $a_i < 30 \ \forall i$，且實際上的書本品項 $n = 12$，是已知的值，相比之下，唯一不會被現實情況 bound 住的只有訂單數量 m，而這個複雜度和 m 是一次的關係，應該是相對較優的。

3 不論順序，寄出最多本書

以這種標準決定寄書順序時，其實跟第 2 種標準蠻類似的，相當於每個訂單都被給了一個權重而已，除了暴力解法外，可以直接將前面使用的 DP 遞迴式修改成

$$\begin{aligned} &T(m, a_1, \cdots, a_n) \\ =&\max\left(T(m-1, a_1, \cdots, a_n), T(m-1, a_1 - b_{m1}, \cdots, a_n - b_{mn}) + b_m\right), \end{aligned}$$

紅色部分的修改相當於每張訂單都利用那張訂單包含的書本數量做加權，除此之外都可以利用與前面相同的方法，獲得一樣的結果，時間複雜度仍然是 $O(m \times \Pi_{i=0}^{n} a_i)$。

心得

其實這份 DIY 從 DIY 1 之前我就開始做了，開學時教授講到要做 DIY 我馬上就想到這個問題，但那時候覺得大概隨便 sort 一下，然後選的好一點就可以解決，結果實際在做時認眞想一遍才發現沒有那麼容易。那時候完全還沒有關於 DP 這種方法的知識，想到的也只有暴力法和最直接的 greedy，但不管依照什麼方式 greedy 好像都得不到最佳解，後來我發現這好像是整數線性規劃的問題，似乎沒有多項式時間的解，所以我就暫時擱置了。直到課程上到 Dynamic Programming，我才又再思考這種方法能不能用在這個問題上，稍微想了想覺得好像有機會，但期中有點忙，所以就拖到 DIY 3 之前才寫出來。

另外，分析複雜度之後，我其實覺得蠻神奇的，雖然複雜度算不上好，但依照我實際的經驗，在參數都還在 $2, 30$ 以內時，人的腦袋可以很直覺的就找到最佳解，好像在處理小的數字時，就算很複雜，人腦也可以輕鬆得到答案，我猜想可能是因爲人腦可以更好的進行平行處理吧。