演算法設計報告

B1275015 孫語辰

一、題目重新描述

為了以下報告之簡潔與直觀性，對於題目中一些複雜的計算式與描述法，我於此處重新作出部分描述與定義。請注意對於題目中有指定範圍的變數，以下所有描述皆嚴格遵守範圍限制並不再對其範圍作出描述。

我們的目標為最大化單日租車營收，此處定義為V，考慮對於車型i在車站j對 V 的影響：首先，由於車站 j 多了一台車，很顯然的，V 應該會增加：$R_i * [W_{i,j} - \Delta W_j]$，其中，$R_i$ 表示車型 i 的每小時費用、$W_{i,j}$表示車型 i 於車站 j 的稼動率、ΔW_j表示車站 j 截至目前為止受到競食效應的影響(此處取正)。但在車站 j 停入新的車的同時，由於他會改變其他車站的ΔW，其他車站對 V 的影響也會有所變動。

而要找出每次推入車站 j 對其他車站的影響，我們必須要找出其他車站分別與車站j的距離等級$K \in \{0,1,2,3\}$，於是我們維護集合$N_j^{(K)}$，表示與車站j距離在等級K的所有車站。而推入車站j將對V造成影響為：

$-\sum_{t=0}^{3}\sum_{j' \in N_j^t}\sum_{e \in I_{j'}} R_e * Q_t$ ，其中$I_{j'}$表示在 j' 這個車站中的所有車(以車型表示)、Q_t代表在第 t 個距離等級，稼動率下降的量。

也就是說，最終每一次放入一台新的車時 V 的變化為：

$$R_i * [W_{i,j} - \Delta W_j] - \sum_{t=0}^{3}\sum_{j' \in N_j^t}\sum_{e \in I_{j'}} R_e * Q_t$$

我們將其定義為ΔV。

二、算法設計

我們重複好幾次迴圈，每次都枚舉所有車站，找到能帶來使 V 增加最多的車站。那麼問題就在於我們如何決定車型，我們可以觀察上式，當我們給定車站 j ，跟車型 i 有關的只有$R_i * W_{i,j}$而它當然越大越好，所以我們只要找到能使這個值最大的車型 i 即可，這個任務是簡單的，因為$R_i * W_{i,j}$都是一開始就給定的定值，只要針對車站 j 枚舉所有車型，在還沒用完的車型

中找到$R_i * W_{i,j}$最大的即可。以下再詳細敘述演算法的步驟。

算法開始：

1. 準備輸入資料
2. 製作前述之集合 N ，方法不限但原則上一個$N_j^{(K)}$可簡單的用多維陣列維護，像是$N[j][K][(all\ stations\ effected\ by\ station\ j\ in\ range\ K)]$。
3. 進入迴圈(while)，迴圈內容如下：
   枚舉所有車站，令當前枚舉的車站為 j ，如果 j 還沒滿，就找出最佳的車型 i 。有了這些資訊就可以算出ΔV，在所有的車站中找出最大的ΔV並記錄對應的車型和車站。如果ΔV_{max}小於 0 迴圈結束，否則更新答案與ΔW(更新ΔW用集合 N即可)。
4. 輸出答案

只要完成上述算法即可達到 75 分，但我的程式大約花了 4400ms 才執行完畢，我認為可以再更好於是我針對上述兩畫線處進行優化。

三、時間優化

(i)

在上面我們提到，對於每個車站，我們都可以找到對應的最佳車型。我們可以描述的在詳細一點：對於車站 j ，我們假設的車型 i_j^r 為 $R_i * W_{i,j}$ 中第r大的車型，所以我們應該先取 i_j^1 ，如果用完就找 i_j^2 …，以此類推。重點在於說這個順序是固定的，所以我們可以先維護好這個順序，問題就只剩下用完與否了。

在最初的實作中，我對於每次枚舉到車站 j 時，我都枚舉所有車型，來找到還未用完的最佳車型 i 。但根據我的推論，我們只要對於每個車站 j ，維護陣列 i_j^r，也就是做 j 個陣列，每個陣列將 i 根據 $R_i * W_{i,j}$排序就行了。接下來在迴圈中每次枚舉到車站 j 時，我們都檢查 i_j^r 的開頭是否用完，如果用完就將其從陣列推出，直到 i_j^r 的開頭是一個未被用完的車型為止。由於每個車站 j 只將所有車型推出一次，原本每次迴圈都做$n * m$次，變成每次迴圈只做 m 次且額外做最多$n * m$次的推出陣列，明顯優化許多。
進行此優化後，程式執行時間為 888ms

(ii)

事實上，這個演算法可以再優化，我們再次觀察ΔV：

$$R_i * [W_{i,j} - \Delta W_j] - \sum_{t=0}^{3} \sum_{j' \in N_j^t} \sum_{e \in I_{j'}} R_e * Q_t$$

我們發現其中的$\sum_{e \in I_{j'}} R_e$並不用每次重算，因為即便於車站 j 放入新的車，$\sum_{e \in I_{j'}} R_e$也不受影響。也就是說，我們可以維護一個m大小的陣列 S 其中 S_j 表示目前停在車站 j 的所有車輛的每小時費用的總和。每次在確定ΔV_{max}後更新S(更新ΔV_{max}對應的車站)，即可將迴圈跑過 $I_{j'}$ 的時間變成單純一次的取值。最後可以將ΔV表示為：

$$R_i * [W_{i,j} - \Delta W_j] - \sum_{t=0}^{3} \sum_{j' \in N_j^t} S_{j'} * Q_t$$

進行此優化後，程式執行時間大約為 400ms