

1 貪心法

貪心法的作法常常是從一股直覺得出來的，但直覺總有錯的時候，我們不能隨便就說一個作法是對的，而是要好好證明才行。

1.1 直接反證

貪心法的證明思路不外乎就是把我們的演算法得出的解和其他解比較一下，確認我們的解不會比較差（別的解可以一樣好！）。一種比較好理解的方式是：假設我們的貪心演算法得到了一組解 S ，然後有任意一組最佳解 S^* ，我們嘗試對 S^* 作一些改變，在保持它不會變差的同時，最終把 S^* 變成 S 。這樣，我們就知道 S 是一組最佳解了。

複習一下影片中提到的「誰先晚餐」問題，經簡化的題目是這樣的：

例題 1

有 n 位同學要吃晚餐，第 i 位同學想吃的飲食需要 C_i 時間才能煮好，而他吃掉飲食所花的時間為 E_i 。廚師同一時間只能煮一份飲食。請證明，讓吃飯時間 (E_i) 越久的越先吃，一定可以讓「廚師開始煮菜」到「最後一位同學吃完」所經過的時間最短。

解答

令我們的演算法得到的吃飯的順序為 a_1, a_2, \dots, a_n ，根據我們的作法，它滿足 $\forall 1 \leq i < n, E_{a_i} \geq E_{a_{i+1}}$ 。假設有一組最佳解吃飯的順序是 b_1, b_2, \dots, b_n ：

- 如果它跟 a_1, a_2, \dots, a_n 一樣，那我們的解當然就是最佳解了；
- 如果它們長得不一樣，我們就知道存在相鄰的人 b_i, b_{i+1} ，使得在 a_1, \dots, a_n 裡面， b_i 出現的位置比 b_{i+1} 後面，根據剛剛說的性質，我們知道 $E_{b_i} \leq E_{b_{i+1}}$ 。

考慮將這兩個人的吃飯順序對調，則第 j 個人吃飯結束的時間（對調前為 $t_1(j)$ ，對調後為 $t_2(j)$ ），會有以下四種情況：

(1) $j < i$ ：

$$\text{對調前，結束的時間為 } t_1(j) = \sum_{k=1}^j C_{b_k} + E_{b_j} ;$$

$$\text{對調後，結束的時間為 } t_2(j) = \sum_{k=1}^j C_{b_k} + E_{b_j} .$$

(2) $j = i$ ：

$$\text{對調前，結束的時間為 } t_1(j) = t_1(i) = \sum_{k=1}^{i-1} C_{b_k} + C_{b_i} + E_{b_i} ;$$

$$\text{對調後，結束的時間為 } t_2(j) = t_2(i) = \sum_{k=1}^{i-1} C_{b_k} + C_{b_{i+1}} + C_{b_i} + E_{b_i} .$$

(3) $j = i + 1$:

對調前，結束的時間為 $t_1(j) = t_1(i+1) = \sum_{k=1}^{i-1} C_{b_k} + C_{b_i} + C_{b_{i+1}} + E_{b_{i+1}}$ ；

對調後，結束的時間為 $t_2(j) = t_2(i+1) = \sum_{k=1}^{i-1} C_{b_k} + C_{b_{i+1}} + E_{b_{i+1}}$ 。

(4) $j > i + 1$:

對調前，結束的時間為 $t_1(j) = \sum_{k=1}^j C_{b_k} + E_{b_j}$ ；

對調後，結束的時間為 $t_2(j) = \sum_{k=1}^j C_{b_k} + E_{b_j}$ 。

我們要比較的是 $\max\{t_1(j)\}$ 和 $\max\{t_2(j)\}$ ($1 \leq j \leq n$)，可以發現會讓 $t_1(j)$ 和 $t_2(j)$ 不同值的只有 $j = i$ 和 $j = i+1$ ，而且 $t_1(i+1) \geq t_2(i)$ (因為 $E_{b_{i+1}} \geq E_{b_i}$)， $t_1(i+1) \geq t_2(i+1)$ ，所以 $\max\{t_1(j)\} \geq \max\{t_2(j)\}$ 。也就是說，對調之後最後吃完的時間一定不會比對調前差。

經過不斷的對調，一定可以把 b_1, b_2, \dots, b_n 變成 a_1, a_2, \dots, a_n ，而且不會比一開始差，得出 a_1, a_2, \dots, a_n 是一組最佳解。

其實以上的證明還有一點點不足的地方。

先提一下一個值得注意的點：我們剛才要對調時找的 b_i, b_{i+1} 是「在 a_1, \dots, a_n 之中出現順序相反的相鄰兩人」，而不是 $E_{b_i} < E_{b_{i+1}}$ 的相鄰兩人，這麼做是為了考慮可能會有不同人有一樣的吃飯時長，若按照我們的寫法，只要這樣不斷對調，最後就會自然而然把 b_1, \dots, b_n 變得跟 a_1, \dots, a_n 一模一樣了。如果寫成 b_i, b_{i+1} 是 $E_{b_i} < E_{b_{i+1}}$ 的相鄰兩人，那頂多只能說最後 b_1, \dots, b_n 會是照吃飯時間排序好的而已。雖然這是我們的演算法在做的事情，但這樣其實要多費一點字數來說「任意一種照吃飯時間大到小排序好的解都一樣好」。這也沒有不行，只是以上用了筆者認為比較乾淨的寫法。

那不足的地方到底在哪裡？一直交換 b_1, \dots, b_n 裡面「在 a_1, \dots, a_n 之中出現順序相反的相鄰兩人」，真的最後就會自然而然得到跟 a_1, \dots, a_n 一樣的解嗎？雖然只要跟 a_1, \dots, a_n 長得不一樣，就一定找得到出現順序相反的相鄰兩人，但會不會其實這個交換過程永無止境呢？答案是不會，最直觀的理解方式是其實我們在做的事情是 bubble sort，目標是「將 b_1, \dots, b_n 按照每個人在 a_1, \dots, a_n 裡的出現順序排序」，直接 bubble sort 最後就會把 b_1, \dots, b_n 變成 a_1, \dots, a_n 了。

要正式的寫出來的話，「交換兩個相鄰的出現順序錯了的東西」會讓逆序數對數量少 1，這裡的逆序數對指的是對於 $i < j$ ，要是 b_i, b_j 的出現順序跟我們預期的相反（在 a_1, \dots, a_n 裡面是先 b_j 再 b_i ），那 i, j 就是一個逆序數對。如果你證明方法那份作業有認真寫，或許會想到可以對逆序數對數量數歸，寫下來就是：

- 如果有一組最佳解 b_1, \dots, b_n 的逆序數對數量是 0，那麼它跟 a_1, \dots, a_n 一樣，做完了。
- 如果有一組最佳解 b_1, \dots, b_n 的逆序數對數量是 $k > 0$ ，假設若有一組最佳解的逆序數對數量是 $k - 1$ ，就代表 a_1, \dots, a_n 是一組最佳解。根據上面說的，我們可以找到相鄰的逆序數對 b_i, b_{i+1} ，交換之後的解不會變差，還是最佳解，而且逆序數對有 $k - 1$ 個，因此 a_1, \dots, a_n 是最佳解。所以，有一組最佳解逆序數對數量是 k 時，也代表 a_1, \dots, a_n 是最佳解。

根據數學歸納法，只要有一組最佳解 b_1, \dots, b_n 的逆序數對數量是 $k \geq 0$ ， a_1, \dots, a_n 就是最佳解。最佳解一定存在、都滿足這個條件，所以 a_1, \dots, a_n 是最佳解。

太麻煩了吧？也可以不用那麼麻煩，這個數學歸納法並不是必要的，只要我們一開始就假設 b_1, \dots, b_n 是逆序數對數量最少的最佳解，我們就可以直接說：

- 如果逆序數對數量是 0，那 a_1, \dots, a_n 跟它長一樣，是一組最佳解。
- 如果逆序數對數量 > 0 ，那可以用上述的方式讓逆序數對數量變少，得到一組逆序數對數量比較少的最佳解，跟 b_1, \dots, b_n 是逆序數對最少的最佳解矛盾。

所以，逆序數對數量只能是 0， a_1, \dots, a_n 是最佳解。

整理一下，這種證明方法的步驟是：假設我們的解是 S 、有一組跟 S 長得盡可能像的最佳解 S^* ，要是 S^* 跟 S 直接長得一樣那就證完了；否則，我們就對 S^* 做某種操作，讓它跟 S 長得更像，從而得出矛盾。這個「長得有多像」的依據可以像剛剛那樣是逆序數對數量，也可以是最長共同前綴的長度之類的。

1.2 貪心選擇與最佳子結構

雖然剛剛說的反證法已經不錯用了，但有時候比較難用在更複雜的題目上，畢竟有時候不太好定義「兩組解長得有多像」這件事。

如果讀者在一些演算法教科書上看過貪心法的證明，可能會長得像是這樣子：

- 要解的問題有**最佳子結構** (optimal substructure)，解決問題的過程是一系列的決策，「子結構」的意思是，在做完第一個決策後，問題可以被轉換成一個全新的、跟本來一樣形式的問題，這個問題是本來問題的一個子問題，把解決這個子問題的決策依樣畫葫蘆套用在本來問題上，就會得到本來問題的一組解。**最佳子結構**的意思是，在做出第一步決策後，接著做出子問題的最佳決策，只要第一步決策選得好，就可以得到原問題的最佳解。

- 要解的問題有一個**貪心選擇性質** (greedy choice property)，意思是當我們在做第一步決策的時候，有某一種特別的性質，只要第一步決策滿足這個性質，它就是最佳子結構要求的「選得好的第一步決策」。更正式地說，是對於任意一個滿足這個性質的決策，都存在一組最佳解，滿足它是第一步決策。

通常子結構都有明顯的子問題會越來越小的性質，所以一直做決策總是會到終點，而我們的貪心演算法就是不斷做出滿足 greedy choice property 的決策，最後就會得出最佳解。

還記得影片裡有說「整個決策的過程可以畫成一棵樹」嗎？這棵樹就是所謂的子結構，greedy choice property 會指示我們往某個子節點走，而那個分支裡一定存在最佳解。

以上的說明看起來可能有一點抽象，我們用這樣的格式來寫一次誰先晚餐的證明看看：

- 最佳子結構：通常會寫成一個遞迴式，而決策過程就跟演算法做的事一樣。令 $f(C, E)$ 是每個人煮飯時間、吃飯時間分別是 $C = [C_1, \dots, C_n]$ 和 $E = [E_1, \dots, E_n]$ 的話，所有人都吃完至少要花多少時間。決策就是決定現在要煮誰的飯，如果現在決定好要煮第 i 個人的飯，那剩下來的問題就是怎麼讓其他人盡快吃完，因此子問題是 $C' = [C_1, \dots, C_{i-1}, C_{i+1}, \dots, C_n]$ 和 $E' = [E_1, \dots, E_{i-1}, E_i, \dots, E_n]$ 構成的只有 $n - 1$ 個人的問題，做出這個決策後能夠得出的最佳解是 $\max(C_i + E_i, C_i + f(C', E'))$ ，而原問題的最佳解自然就是從所有第一步決策之中挑最好的，因此

$$f(C, E) = \min_{1 \leq i \leq n} \{\max(C_i + E_i, C_i + f(C', E'))\}$$

做到剩下 0 個人就做完了。

- Greedy choice property：我們想要的性質是「第一步決策一定可以是選吃飯時間最長的那個人，先煮他的飯」，假設有一組最佳解煮飯的順序是 b_1, \dots, b_n ，令 i 是任意一個 E_{b_i} 最大的 i ，這也代表 $\forall 1 \leq j < i, E_{b_j} \geq E_{b_i}$ ，也就是 b_i 前面的人都吃得比較快或一樣快、而本來 b_i 的飯又是 b_1, \dots, b_i 之中最晚做好的，所以在本來的策略中， b_1, b_2, \dots, b_i 之中最晚吃完的人是 b_i ，時間是 $\sum_{j=1}^i C_{b_j} + E_{b_i}$ 。如果 $i = 1$ 那選 b_i 當第一個當然可以是最佳策略，否則，考慮交換 b_1, b_i ，在交換之後，

- b_i 吃完的時間是 $C_{b_i} + E_{b_i} < \sum_{j=1}^i C_{b_j} + E_{b_i}$ ，
- 對於 $1 < k < i$ ，交換之後 b_k 吃完的時間是 $C_{b_i} + \sum_{j=2}^k C_{b_j} + E_{b_k} \leq \sum_{j=1}^i C_{b_j} + E_{b_k} \leq \sum_{j=1}^i C_{b_j} + E_{b_i}$ 。
- b_1 吃完的時間是 $\sum_{j=1}^i C_{b_j} + E_{b_1} \leq \sum_{j=1}^i C_{b_j} + E_{b_i}$ 。

後面的人吃完的時間都不變。交換之後不會更差，還是最佳解，所以存在最佳策略是第一個煮 b_i 的飯。

證明 greedy choice property 的方法跟前面用的直接反證法¹有一點像，都是想辦法把最佳解變成我們想要的樣子。這樣的證明方式跟演算法的過程比較相近一點，通常子結構都非常直覺，只要把剩下來還沒決定好的東西當作一個新的問題做就好了。而要證明那是一個最佳子結構，就像這樣寫出一個遞迴式就好，通常都可以很直覺接受那是對的。其實最佳子結構本身是數學歸納法的包裝，所以這跟我們一開始用的方法相去不遠，只是思路不太一樣。(可以試試看把直接反證法中「兩個解有多像」的依據定為最長共同前綴的長度，會發現跟這個作法的相似度很高！)

或許從這個例子感受不太到這個證明方法的好處（至少筆者認為前面的直接反證法比較乾淨也比較好懂）。在複雜的問題上，這個方法會有比較顯著的優勢：

例題 2

有 n 種字元，第 i 種字元會出現 f_i 次，求最優編碼樹。就是影片裡說的霍夫曼編碼，可以想成是要蓋一棵二元樹，每個字元要被標在相異的葉節點上，如果第 i 種字元所在的葉節點深度是 d_i ，則總成本是 $\sum_{i=1}^n f_i d_i$ ，目標是最小化成本。

解答

從影片可以知道，作法是先拿出出現次數最少的兩個字元 x, y ，把 x, y 合併成同一種字元、出現次數是 $f_x + f_y$ ，解決完剩下的問題後，把 x, y 吊在那個它們合併後的字元對應的葉節點下面。

最佳子結構其實已經差不多說完了！答案的樹上總是會有兩個葉節點互為兄弟，可以視為是先選兩個字元 x, y 當兄弟，接下來就把它們頭上的父節點當成是它們合併後的字元、假裝它們兩個自己不存在，剩下的部分取最佳解後，再放回這兩個兄弟，就是最佳解，成本是子問題的成本再加上 $f_x + f_y$ ，子問題是字元種類數為 $n - 1$ 的問題，做到剩 1 種字元就做完了。

Greedy choice property 是 x, y 肯定可以選出現次數最少的那兩個字元，換個角度說，是出現次數最少的兩個字元 x, y 肯定可以是樹上的兄弟。考慮任意一組最佳解的樹，如果 x, y 在上面是兄弟，那就證完了，否則，深度最深的那些點之中一定有一對兄弟 a, b 。

如果 x, y 有其中一人在 a, b 之中，假設是 $x = a$ 、 $y \neq b$ ，把 y, b 交換，成本的變化量是

$$\begin{aligned} & -(f_y d_y + f_b d_b) + (f_y d_b + f_b d_y) \\ & = f_y(d_b - d_y) + f_b(d_y - d_b) \\ & = (f_y - f_b)(d_b - d_y) \end{aligned}$$

¹這是筆者自己取的名字

如果 x, y 都不在 a, b 之中，假設 $f_a \leq f_b$ 、 $f_x \leq f_y$ ，我們把 x, a 交換、 y, b 交換的話，成本的變化量是

$$\begin{aligned} & - (f_x d_x + f_y d_y + f_a d_a + f_b d_b) + (f_x d_a + f_y d_b + f_a d_x + f_b d_y) \\ & = f_x(d_a - d_x) + f_y(d_b - d_y) + f_a(d_x - d_a) + f_b(d_y - d_b) \\ & = (f_x - f_a)(d_a - d_x) + (f_y - f_b)(d_b - d_y) \end{aligned}$$

$f_x - f_a$ 跟 $f_y - f_b$ 一定 ≤ 0 、 $d_a - d_x$ 跟 $d_b - d_y$ 一定 ≥ 0 ，所以變化量都 ≤ 0 ，得出成本不會變高，因此一定存在一組最佳解，滿足 x, y 在樹上是兄弟。

這題要是用直接反證法，就比較難定義出「兩個解長得有多像」，就算硬是定義出來可能也不好寫出乾淨的證明，用這個方式證明反而直覺很多。

習題

本作業的總分不含整潔分數是 130 分，若你的得分超過 120 分，會以 120 分計算成績。整潔分數另計，所以這份作業的最高得分是 125。

1. (20 pts) 考慮以下的可分割背包問題：你有一個承重最多 L 的背包，並有 n 個物品，其中第 i 個有重量 $w_i > 0$ 與價值 $c_i \geq 0$ ，每個物品都可以分割，也就是說，對於任意第 i 個物品，你可以選擇要只帶其中 p ($0 \leq p \leq 1$) 的部分。此時帶這個物品的重量與效用就分別是 pw_i 與 pc_i 。

請證明以下策略為最優解：將物品以 $\frac{c_i}{w_i}$ 由大至小排序，由前往後，如果能取完整個物品則取，不能（背包滿了）則只取剛好讓背包滿的那部分。

2. 在影片中提到了換零錢的題目，敘述如下：已知有 n 種貨幣面額 c_1, c_2, \dots, c_n ，不失一般性可以假設 $c_1 < c_2 < \dots < c_n$ 。對於任意兩種面額 c_i, c_j ($i < j$)，滿足 c_j 一定是 c_i 的倍數，也就是 $c_i|c_j$ ，而且為了確保所有整數的金額都可以湊出， $c_1 = 1$ (例如台灣的硬幣去掉 20 元就滿足這些要求)。對於任意正整數 x ，請問如何使用盡量少的零錢個數湊出 x 元呢？

(a) (30 pts) 請證明在題目的條件之下，盡量使用面額較高的零錢可以得到最佳解。意即要湊出 x ，可以先使用一枚面額不大於 x 的最大面額零錢 c_i ，剩下的 $x - c_i$ 元用相同的方法湊出。

(b) (5 pts) 在不滿足 $\forall i < j, c_i|c_j$ 的情況下，第一題的貪心算法仍然可以得到最佳解嗎？如果可以，請證明之（可以的意思是不論面額和要湊出的金額為多少，貪心算法得到的解都是正確的）；如果不可以，請給出造成反例的面額 $c_1 \dots c_n$ 、欲湊出的金額 x 、使用貪心法得到的解以及能比貪心法得到的解使用更少個零錢的解。請注意，構造反例時也要符合 $c_1 = 1$ 的條件，且硬幣種類數 n 要是有限的喔。

(c) (15 pts) 在不滿足 $\forall i < j, c_i|c_j$ 的情況下，第一題的貪心算法一定不能得到最佳解嗎？如果不可以，請證明之；如果可以，請給出一組貪心法適用的面額 $c_1 \dots c_n$ ，並證明在該組面額上，使用貪心法湊出任意金額都可以得到最佳解。

3. (30 pts) 已知 n 個區間 $[l_1, r_1], [l_2, r_2], \dots, [l_n, r_n]$ ，希望可以挑選出盡量多個不重疊的區間（端點碰到也不行）。請證明先將區間照右界 (r_i) 由小到大排序後，依序挑選與當前挑出的區間不重疊的區間，如此可以挑選出最多的區間。

(例：有四個區間 $[1, 4], [2, 5], [3, 7], [6, 8]$ ，則上述貪心演算法會依序挑選 $[1, 4]$ 和 $[6, 8]$ ，得到最多可以挑選出 2 個區間。當然最佳解不唯一，演算法也可以改成照區間左界由大到小排序，此時挑選出的區間就是 $[6, 8]$ 和 $[2, 5]$ ，但區間數量仍然是最大值。)

4. (30 pts) 還記得證明方法那份作業裡有個投票的問題嗎？我們把那個問題再加難一點：有 $n \geq 2$ 個人要進行投票，每個人手上擁有的票數是不一樣的，第 i 個人手上握有 b_i 張票，他可以選擇每張票要投給誰，可以投給不同的人也可以全部投給同一個人，唯獨

不能投給他自己，最後第 i 個人得到了 a_i 張票，告訴你 n 、 a_1, \dots, a_n 、 b_1, \dots, b_n ，保證 $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ ，請設計一個演算法，找出一種投票方式（每一張票要投給誰）得到這個結果，如果不存在任何可能的投票方式，你的演算法要回報不可能。

演算法過程只要簡單敘述即可，可以直接使用任何 C++ STL 裡有的東西²，並且請嚴謹證明正確性並簡單分析時間複雜度。令總票數為 $T = \sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ ，你的執行時間應該要在 $O(n \times T)$ 以內，超過的話你會得到 0 分，舉例來說，直接窮舉所有的投票方式是一個會得到 0 分的作法。只要是在這個時間複雜度之內，你的演算法多有效率不會影響你的得分。

以下的題目不計分、不須繳交為作業，不過有興趣的學員可以自己嘗試。

5. (0 pts) 現代的電腦通常都有很大的記憶體，不幸的是，雖然存取記憶體中的資料比存取硬碟中的資料來得快，但還是有一點慢。為此，電腦中有個東西叫作快取 (cache)，如它的名字所示，存取它很快，只要把常常需要被存取的資料放在快取裡面，就可以大幅增加程式的執行速度！然而，什麼叫作常常需要存取呢？這是一個好問題，我們沒有要討論這個。

在記憶體裡面有 n 個資料，你的快取有 k 個欄位，一開始每個欄位都沒有放任何資料，接下來有 q 個讀取事件依序發生，第 i 個讀取事件想要拿到記憶體中的第 a_i 個資料，要是這個資料已經在快取裡了，那就可以快樂地直接存取它，否則，你必須要把那筆資料先從記憶體中讀取、存到快取的某個欄位（一定要做這件事，不管以後還有沒有要用這筆資料），本來已有資料的欄位可以被覆蓋，但一個欄位同時只能放一個資料。你已經知道 a_1, a_2, \dots, a_q 了，你的目標是決定每次讀取事件時，要存取的資料不在快取裡的話，要把資料放在快取的哪一個欄位，才能最小化從記憶體中讀取資料的次數。

請證明這是最佳解：如果在第 i 個事件時， a_i 已經在快取裡了，那什麼也不用做。否則，找到快取中下一次被使用的時間最晚的資料（如果有沒放資料的欄位，那隨便選一個沒放資料的），把 a_i 放到那個欄位。

備註：

- 一種比較生活化的題目敘述可以參考 TIOJ 1221 炒菜問題。
- 不幸的是，在真實世界裡我們不能預知未來，這個方法並不能真正的用在電腦裡。

²請不要直接貼程式碼，你可以選擇使用純文字描述或使用 pseudocode 並搭配一些簡略描述。只要確定批改者看得出你想做什麼、能夠接受這個演算法能在多項式時間內執行完就好。