



# Greedy

Lecture by LittleCube  
Credit by WiwiHo

Sprout



## 開始之前

- 影片看了嗎？

Sprout



## 目錄

- 影片複習
- 更多題目
- 再談證明
- 還有題目

Sprout



# 影片複習

Sprout



## 什麼是 Greedy ?

- 問題的每一組方案都是一組決策
- 決策的過程：決策樹
  - 每個分枝都是一個決定
  - 走到葉節點就是一組最終決策
- Greedy 演算法的性質：  
每次選擇的分支總是會保證還是能走的到其中一組最佳決策。

Sprout



# 更多題目

Sprout



## 暖身 — Stick Divisions

### Problem CSES Stick Divisions

你想要蓋一棟自己的房子，根據工程圖你需要  $n$  隻棍子，第  $i$  隻的長度要是剛好  $d_i$ ，所以你買了一根長度為  $x = \sum d_i$  的棍子來切。

拿起一根棍子切成兩段的費力是那根棍子的長度，你最少需要花多少力氣才能切出你想要的棍子？

Example:  $d = \{2, 3, 3\}$

- $\{8\} \rightarrow \{5, 3\} \rightarrow \{2, 3, 3\} \Rightarrow 8 + 5 = 13$
- $\{8\} \rightarrow \{6, 2\} \rightarrow \{2, 3, 3\} \Rightarrow 8 + 6 = 14$

Sprout



## 暖身 — Stick Divisions

- 切點的越平均感覺越輕鬆

Sprout



## 暖身 — Stick Divisions

- 切點的越平均感覺越輕鬆
- $d = \{1, 4, 5, 8\}$

Sprout



## 暖身 — Stick Divisions

- 切點的越平均感覺越輕鬆
- $d = \{1, 4, 5, 8\}$
- $\{18\} \rightarrow \{\textcolor{red}{9}, \textcolor{blue}{9}\} \rightarrow \{\textcolor{red}{1}, \textcolor{red}{8}, \textcolor{blue}{4}, \textcolor{blue}{5}\} \Rightarrow 18 + 9 + 9 = 36$
- $\{18\} \rightarrow \{8, 10\} \rightarrow \{8, 5, 5\} \rightarrow \{8, 5, 1, 4\} \Rightarrow 18 + 10 + 5 = 33$

Sprout



## 暖身 — Stick Divisions

- 把最大的先切走

Sprout



## 暖身 — Stick Divisions

- 把最大的先切走
- $d = \{1, 1, 1, 1\}$

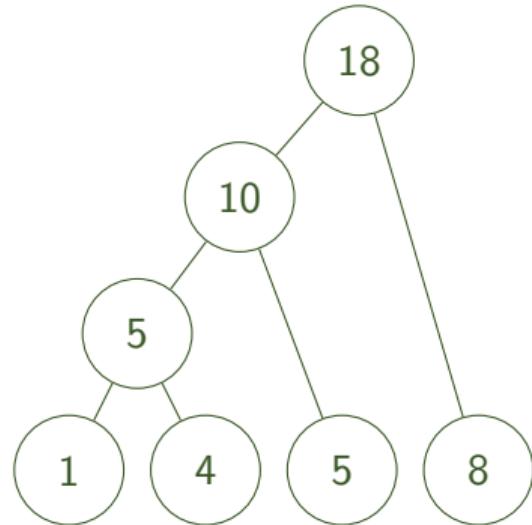
Sprout



## 暖身 — Stick Divisions

切棍子每次都是切成兩根，根本就是樹狀圖

$$d = \{1, 4, 5, 8\}$$

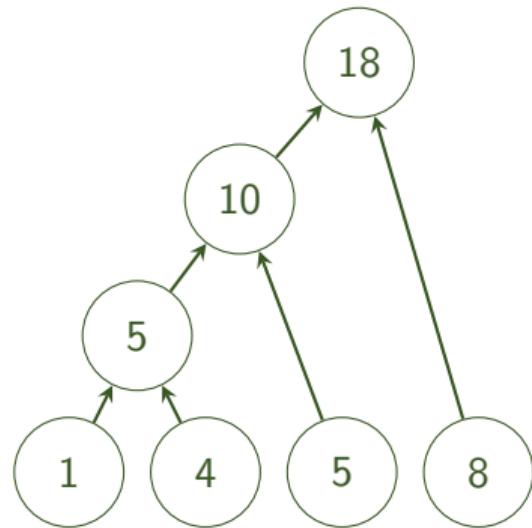


Sprout



## 暖身 — Stick Divisions

與其正著亂切一通，反過來合併不就是我們知道的霍夫曼編碼嗎？



Sprout



## 各種線段問題

### Problem USACO Why Did the Cow Cross the Road

農場裡有  $C$  隻小雞跟  $N$  隻牛，第  $i$  隻小雞可以在  $T_i$  的時間點幫助一隻牛過馬路，而第  $j$  隻牛可以在  $[A_j, B_j]$  的時間點過馬路。最多可以有幾隻牛過馬路？

Sprout



## 各種線段問題

- 每條牛可以選好幾隻小雞，不知道要選哪隻
- 區間有長有短，不知道要怎麼選

Sprout



## 各種線段問題

- 每條牛可以選好幾隻小雞，不知道要選哪隻
- 區間有長有短，不知道要怎麼選
- 反過來從小雞下手：小雞一定排得出時間順序

Sprout



## 各種線段問題

- 每條牛可以選好幾隻小雞，不知道要選哪隻
- 區間有長有短，不知道要怎麼選
- 反過來從小雞下手：小雞一定排得出時間順序
- 考慮最早出的小雞，如果能幫助牛就幫不虧！
- 要幫助哪一隻？

Sprout

## 各種線段問題

- 每條牛可以選好幾隻小雞，不知道要選哪隻
- 區間有長有短，不知道要怎麼選
- 反過來從小雞下手：小雞一定排得出時間順序
- 考慮最早出的小雞，如果能幫助牛就幫不虧！
- 要幫助哪一隻？
- 對這隻小雞能幫的所有牛來說， $A_i$  已經不怎麼重要，但是  $B_i$  小的會先失去機會

Sprout

## 各種線段問題

- 每條牛可以選好幾隻小雞，不知道要選哪隻
- 區間有長有短，不知道要怎麼選
- 反過來從小雞下手：小雞一定排得出時間順序
- 考慮最早出的小雞，如果能幫助牛就幫不虧！
- 要幫助哪一隻？
- 對這隻小雞能幫的所有牛來說， $A_i$  已經不怎麼重要，但是  $B_i$  小的會先失去機會
- 優先幫助右界最小的！

Sprout



## 各種線段問題

### Problem 經典問題

工廠裡有  $N$  個工作需要負責執行，第  $i$  個工作需要一隻機器人在  $L_i$  到  $R_i$  的時間負責。一台機器人同一時間只能負責一個工作，你最少需要幾台機器人？

Sprout



## 各種線段問題

- 假設現在有很多很多機器人，工作來了要交給誰？

Sprout

## 各種線段問題

- 假設現在有很多很多機器人，工作來了要交給誰？
- 如果用隨便的順序指派工作好像很難維護又很難擬定策略
- 但是，要是工作開始時間遞增，那我們只需要在乎每個機器人做完手上工作的時間

Sprout

## 各種線段問題

- 假設現在有很多很多機器人，工作來了要交給誰？
- 如果用隨便的順序指派工作好像很難維護又很難擬定策略
- 但是，要是工作開始時間遞增，那我們只需要在乎每個機器人做完手上工作的時間
- 目前可以接工作的機器人其實不管什麼時候做完都沒差，反正從現在開始它們都暫時有空
- 直接看現在最早完成的機器人可不可以接，不然就多叫一台機器人！

Sprout

## 各種線段問題

- 假設現在有很多很多機器人，工作來了要交給誰？
- 如果用隨便的順序指派工作好像很難維護又很難擬定策略
- 但是，要是工作開始時間遞增，那我們只需要在乎每個機器人做完手上工作的時間
- 目前可以接工作的機器人其實不管什麼時候做完都沒差，反正從現在開始它們都暫時有空
- 直接看現在最早完成的機器人可不可以接，不然就多叫一台機器人！
- 拖延到必須作出決定再做有時候是很好的 greedy 策略
- 可以試試看怎麼嚴謹證明，不過後面會介紹更簡潔的證明方法

## 不顯然的 Greedy — 物品堆疊

### Problem APCS 物品堆疊

有  $N$  個物品，其中第  $i$  個的重量是  $w_i$ ，要被使用的次數是  $f_i$ 。你要決定某個順序  $a_1, a_2, \dots, a_N$  由上而下把它們疊起來。不過你每次要拿某個東西時需要先把上面的物品都抬起來，也就是說總花費是

$$\sum_{i=1}^N \left( f_{a_i} \times \sum_{j=1}^{i-1} w_{a_j} \right)$$

這個花費最小可以是多少？

## 不顯然的 Greedy — 物品堆疊

- 越上面的東西，重量會被算越多次，那就把重的放下面好了

Sprout

## 不顯然的 Greedy — 物品堆疊

- 越上面的東西，重量會被算越多次，那就把重的放下面好了
  - $(f_i, w_i) = \{(2, 1), (100, 1)\}$

Sprout

## 不顯然的 Greedy — 物品堆疊

- 越上面的東西，重量會被算越多次，那就把重的放下面好了
  - $(f_i, w_i) = \{(2, 1), (100, 1)\}$
- 那我把用比較多次的放上面總行了吧

Sprout

## 不顯然的 Greedy — 物品堆疊

- 越上面的東西，重量會被算越多次，那就把重的放下面好了
  - $(f_i, w_i) = \{(2, 1), (100, 1)\}$
- 那我把用比較多次的放上面總行了吧
  - $(f_i, w_i) = \{(1, 100), (1, 2)\}$

Sprout

## 不顯然的 Greedy — 物品堆疊

- 越上面的東西，重量會被算越多次，那就把重的放下面好了
  - $(f_i, w_i) = \{(2, 1), (100, 1)\}$
- 那我把用比較多次的放上面總行了吧
  - $(f_i, w_i) = \{(1, 100), (1, 2)\}$
- 不要亂猜！

Sprout

## 不顯然的 Greedy — 物品堆疊

- 對於只有兩個物品 ( $N = 2$ ) 的時候

- $a = \{1, 2\} \Rightarrow f_2 \times w_1$
- $a = \{2, 1\} \Rightarrow f_1 \times w_2$

Sprout

## 不顯然的 Greedy — 物品堆疊

- 對於只有兩個物品 ( $N = 2$ ) 的時候
  - $a = \{1, 2\} \Rightarrow f_2 \times w_1$
  - $a = \{2, 1\} \Rightarrow f_1 \times w_2$
- 不管上面下面疊了什麼，我們都沒有改變他們的花費

Sprout

## 不顯然的 Greedy — 物品堆疊

- 對於只有兩個物品 ( $N = 2$ ) 的時候
  - $a = \{1, 2\} \Rightarrow f_2 \times w_1$
  - $a = \{2, 1\} \Rightarrow f_1 \times w_2$
- 不管上面下面疊了什麼，我們都沒有改變他們的花費
- 如果有兩個相鄰的人換了會變好，就交換！

Sprout



## 不顯然的 Greedy — 物品堆疊

- 一定會換到停下來嗎？

Sprout

## 不顯然的 Greedy — 物品堆疊

- 一定會換到停下來嗎？
- 比較  $f_2 \times w_1$  跟  $f_1 \times w_2$ ，實際上是比較  $\frac{f_2}{w_2}$  跟  $\frac{f_1}{w_1}$ ！

Sprout

## 不顯然的 Greedy — 物品堆疊

- 一定會換到停下來嗎？
- 比較  $f_2 \times w_1$  跟  $f_1 \times w_2$ ，實際上是比較  $\frac{f_2}{w_2}$  跟  $\frac{f_1}{w_1}$ ！
- 想像每個人都有這樣一個隱藏的優先值，用這個方法 greedy 排序的答案就是好的嗎？
- 回想影片，跟最佳解不一樣的話必定有地方可以交換，不過上面的式子保證從最佳解換過來不可能變更差！

Sprout



## 不顯然的 Greedy — 物品堆疊

記得要怎麼自定義排序嗎？

```
bool cmp(int i, int j)
{
    return w[i] * f[j] < w[j] * f[i];
}

int main()
{
    vector<int> v = {1, 2, 3};
    sort(v.begin(), v.end(), cmp);
}
```

roul

## 不顯然的 Greedy — 物品堆疊

也可以用 lambda function !

```
int main()
{
    vector<int> v = {1, 2, 3};
    sort(v.begin(), v.end(),
        [&](int i, int j) { return w[i] * f[j] < w[j] * f[i]; });
}
```

- [] 代表不能使用外面的變數
- [&] 代表使用 reference
- [=] 代表使用值（會複製，小心複製到大東西複雜度爆炸）



## 不顯然的 Greedy — 字串串接

Problem Codeforces Smallest String Concatenation

有  $N$  個字串，在所有把他們串在一起的方法中，最小字典序的方法是什麼？  
A 字典序比 B 小的定義是比較第一個  $A, B$  不同的位置  $k$  的時候， $A_k < B_k$ 。  
 $N \leq 50\,000$ , 字串長度  $\leq 50$

Sprout

## 不顯然的 Greedy — 字串串接

- 對於只有兩個字串 ( $N = 2$ ) 的時候，比較  $s_1s_2$  與  $s_2s_1$
- 這樣的排序保證是好的嗎？

Sprout

## 不顯然的 Greedy — 字串串接

- 對於只有兩個字串 ( $N = 2$ ) 的時候，比較  $s_1s_2$  與  $s_2s_1$
- 這樣的排序保證是好的嗎？
- 如果字串是一個數字，那  $s_1s_2 = s_1 \times 10^{s_2}$  的位數 +  $s_2$

Sprout

## 不顯然的 Greedy — 字串串接

- 對於只有兩個字串 ( $N = 2$ ) 的時候，比較  $s_1s_2$  與  $s_2s_1$
- 這樣的排序保證是好的嗎？
- 如果字串是一個數字，那  $s_1s_2 = s_1 \times 10^{|s_2|}$  的位數 +  $|s_2|$
- 嘗試把字串寫成數字，如果有  $\Sigma$  種字元就可以寫成

$$s_1 \times \Sigma^{|s_2|} + |s_2| \leq s_2 \times \Sigma^{|s_1|} + s_1$$

移項就可以寫成

$$s_1 \times \frac{1}{\Sigma^{|s_1|} - 1} \geq s_2 \times \frac{1}{\Sigma^{|s_2|} - 1}$$

Sproul

## 拆包裝 — 基地台

### Problem 基地台

有一條數線，上面有  $N$  個給定的點，你最多可以設置  $K$  座基地台，並且決定一個數字  $D$ ，使得對於每一個給定的點，離它最近的基地台距離都不超過  $\frac{D}{2}$ 。求  $D$  至少要是多少。

Sprout

## 拆包裝 — 基地台

- 直接想辦法解出對於  $D$  的最佳化問題，好像有點困難

Sprout



## 拆包裝 — 基地台

- 直接想辦法解出對於  $D$  的最佳化問題，好像有點困難
- 還記得影片裡的抄書問題嗎？

Sprout

## 拆包裝 — 基地台

- 直接想辦法解出對於  $D$  的最佳化問題，好像有點困難
- 還記得影片裡的抄書問題嗎？
- 對答案二分搜！
- 在  $D$  固定的情況下，我可以只設  $K$  座基地台嗎？

Sprout



## 拆包裝 — 基地台

- 從最左邊的給定點開始枚舉
- 假如一個點沒有被任何基地台蓋到，那勢必要興建一個新的
- 因為沒有點在這個點的左邊，所以一定是越右邊越好！
- 最後檢查可不可以不超過  $K$  座

Sprout

## 預判條件 — 包裝問題

### Problem 包裝問題

你有很多彩色鉛筆，其中第  $i$  枝的顏色是  $a_i$ 。現在你要把他們兩個兩個一組包成禮物送給朋友，但是你希望禮物裡兩枝鉛筆的顏色都不一樣，你能不能做到這件事情？

Sprout



## 預判條件 — 包裝問題

### Problem 包裝問題

你有很多彩色鉛筆，其中第  $i$  枝的顏色是  $a_i$ 。現在你要把他們兩個兩個一組包成禮物送給朋友，但是你希望禮物裡兩枝鉛筆的顏色都不一樣，你能不能做到這件事情？

- 什麼樣的狀況必定不會滿足條件？

Sprout



## 預判條件 — 包裝問題

- 如果一種顏色過半，必定不可能達成目標
- (當然還有鉛筆數量是奇數)

Sprout



## 預判條件 — 包裝問題

- 如果一種顏色過半，必定不可能達成目標
- (當然還有鉛筆數量是奇數)
- 如果還沒有任何一種顏色超過一半就一定可以嗎？

Sprout

## 預判條件 — 包裝問題

- 如果一種顏色過半，必定不可能達成目標
- (當然還有鉛筆數量是奇數)
- 如果還沒有任何一種顏色超過一半就一定可以嗎？
- 我們不喜歡某種顏色出現太多，每次就選最多的那兩種包成一包
- 原本少於一半的還是不會過半，等於一半的一定會被用到，而且至多兩種！

Sprout

## 奇怪的 Greedy — 炒菜問題

### Problem 炒菜問題

你有 2 個炒菜鍋，一開始都是乾淨的。接下來你要炒  $N$  道菜，如果這個炒菜鍋是乾淨的或是上次炒的菜跟這次炒的菜是一樣的就能繼續炒，否則就必須要洗鍋子，不然味道會很奇怪。

不過洗鍋子費時費力又不好玩，你想要洗盡量少次，最少可以是多少次？

$$N \leq 10^5$$

Sprout

## 奇怪的 Greedy — 炒菜問題

### Problem 炒菜問題

你有 2 個炒菜鍋，一開始都是乾淨的。接下來你要炒  $N$  道菜，如果這個炒菜鍋是乾淨的或是上次炒的菜跟這次炒的菜是一樣的就能繼續炒，否則就必須要洗鍋子，不然味道會很奇怪。

不過洗鍋子費時費力又不好玩，你想要洗盡量少次，最少可以是多少次？

$$N \leq 10^5$$

- 遇到同樣的菜必定是用同樣的鍋子

Sprout

## 奇怪的 Greedy — 炒菜問題

### Problem 炒菜問題

你有 2 個炒菜鍋，一開始都是乾淨的。接下來你要炒  $N$  道菜，如果這個炒菜鍋是乾淨的或是上次炒的菜跟這次炒的菜是一樣的就能繼續炒，否則就必須要洗鍋子，不然味道會很奇怪。

不過洗鍋子費時費力又不好玩，你想要洗盡量少次，最少可以是多少次？

$$N \leq 10^5$$

- 遇到同樣的菜必定是用同樣的鍋子
- 遇到都不一樣的菜應該要洗掉哪一鍋？



## 奇怪的 Greedy — 炒菜問題

- 常常用來感覺 Greedy 正確性或思考方法的準則：會不會遇到一樣好的選擇？一樣好的選擇可以隨便選嗎？
- 兩道菜乍看之下一樣好，兩道都可以亂洗嗎？

Sprout



## 奇怪的 Greedy — 炒菜問題

- 常常用來感覺 Greedy 正確性或思考方法的準則：會不會遇到一樣好的選擇？一樣好的選擇可以隨便選嗎？
- 兩道菜乍看之下一樣好，兩道都可以亂洗嗎？
- abcacb 在第一個 c 的時候應該要留著 a 不要洗掉

Sprout

## 奇怪的 Greedy — 炒菜問題

- 常常用來感覺 Greedy 正確性或思考方法的準則：會不會遇到一樣好的選擇？一樣好的選擇可以隨便選嗎？
- 兩道菜乍看之下一樣好，兩道都可以亂洗嗎？
- abcacb 在第一個 c 的時候應該要留著 a 不要洗掉
- 猜測隱藏條件：在未來會先出現的不該洗掉

## 奇怪的 Greedy — 炒菜問題

- 假設最佳解告訴我們某一步的時候就應該反過來洗
- 不失一般性假設鍋子上是 a 跟 b，接下來 a 比較早出現
- 而最佳解說第一個鍋子要  $a \rightarrow c \rightarrow X$  而第二個鍋子要  $b \rightarrow Y$ 。

Sprout

## 奇怪的 Greedy — 炒菜問題

- 假設最佳解告訴我們某一步的時候就應該反過來洗
- 不失一般性假設鍋子上是  $a$  跟  $b$ ，接下來  $a$  比較早出現
- 而最佳解說第一個鍋子要  $a \rightarrow c \rightarrow X$  而第二個鍋子要  $b \rightarrow Y$ 。
  - Case 1 :  $Y$  以  $b$  開頭  
這時候  $X$  必定可以寫成  $SaT$  且  $S$  當中沒有  $a$ ，那換成炒  $a \rightarrow a \rightarrow T$  與  $b \rightarrow c \rightarrow S \rightarrow Y$ 。  
分析： $a - c, S - a$  兩次變為  $b - c, S - Y$  至多兩次，不會變差。



## 奇怪的 Greedy — 炒菜問題

- 假設最佳解告訴我們某一步的時候就應該反過來洗
- 不失一般性假設鍋子上是  $a$  跟  $b$ ，接下來  $a$  比較早出現
- 而最佳解說第一個鍋子要  $a \rightarrow c \rightarrow X$  而第二個鍋子要  $b \rightarrow Y$ 。
  - Case 1 :  $Y$  以  $b$  開頭  
這時候  $X$  必定可以寫成  $SaT$  且  $S$  當中沒有  $a$ ，那換成炒  $a \rightarrow a \rightarrow T$  與  $b \rightarrow c \rightarrow S \rightarrow Y$ 。  
分析： $a - c, S - a$  兩次變為  $b - c, S - Y$  至多兩次，不會變差。
  - Case 2 :  $Y$  不以  $b$  開頭  
直接換成  $a \rightarrow Y$  與  $b \rightarrow c \rightarrow X$   
分析： $a - c, b - Y$  兩次變為  $b - c, a - Y$  至多兩次，不會變差。

**Sprout**

## 奇怪的 Greedy — 炒菜問題

- 假設最佳解告訴我們某一步的時候就應該反過來洗
- 不失一般性假設鍋子上是  $a$  跟  $b$ ，接下來  $a$  比較早出現
- 而最佳解說第一個鍋子要  $a \rightarrow c \rightarrow X$  而第二個鍋子要  $b \rightarrow Y$ 。
  - Case 1 :  $Y$  以  $b$  開頭  
這時候  $X$  必定可以寫成  $SaT$  且  $S$  當中沒有  $a$ ，那換成炒  $a \rightarrow a \rightarrow T$  與  $b \rightarrow c \rightarrow S \rightarrow Y$ 。  
分析： $a - c, S - a$  兩次變為  $b - c, S - Y$  至多兩次，不會變差。
  - Case 2 :  $Y$  不以  $b$  開頭  
直接換成  $a \rightarrow Y$  與  $b \rightarrow c \rightarrow X$   
分析： $a - c, b - Y$  兩次變為  $b - c, a - Y$  至多兩次，不會變差。
- 洗鍋子的次數只會變少，Greedy 策略是正確的！
- 實際上鍋子任意多都是對的，有興趣可以參考 Bélády's optimal page replacement policy



# 再談證明

Sprout



## 教科書上的 Greedy

How can we tell whether a greedy algorithm will solve a particular optimization problem? No way works all the time, but the **greedy-choice property** and **optimal substructure** are the two key ingredients.

---

*Introduction to Algorithms*

Sprout

## 教科書上的 Greedy

- Greedy-choice property

我們只要一直選取當下的最佳策略，不需要考慮未來的操作或是回頭，就能走到其中一組最佳決策。換句話說就是什麼樣的性質讓我們可以這樣「貪心」。

- Optimal substructure

就是當我們做完決策之後，問題會變成比較小的原本的問題，稱為子問題。只要在這些子問題中不斷做出最佳決策就好。

一些例子：Huffman Coding

- Greedy-choice property  
最小的兩個節點一定可以當鄰居。
- Optimal substructure  
把兩個節點合併之後，問題就變成比原本少一個點的問題了。

Sprout

### 一些例子：炒菜問題

- Greedy-choice property

洗掉最晚出現的鍋子一定是其中一個最佳策略。

- Optimal substructure

炒完下一道菜之後，問題就變成少一道菜的問題了，不過我們可能要把鍋子的狀態也列為問題的輸入之一。



## 教科書上的 Greedy

其實就只是把影片中遞迴證明法的兩個重要性質寫出來而已

當我們有 Greedy-choice property，就可以做出決策把問題變小，而問題有 Optimal substructure，所以變小之後一樣可以回來做我們貪心的操作。

Sprout



## 影片上看到的證明方法

- 直接反證法
- 數學歸納法
- 遞迴證明法

這真的是三種不同的證明嗎？

Sprout

## 數學歸納法 vs 遞迴證明法

1. 定義  $P_n$  為輸入規模為  $n$  時的問題
2. 證明當  $n = 1$  時，命題成立
3. 證明已知包含  $P_{n-1}$  的解的情況下，貪心策略正確
4. 證明存在  $P_n$  的解包含  $P_{n-1}$  的解
5. 得證

Sprout

## 數學歸納法 vs 遞迴證明法

根本就是數學歸納法！

1. 證明當  $n = 1$  時，命題成立
2. 假設  $P_{n-1}$  的所有問題，貪心策略都正確
3. 考慮任何一個  $P_n$  的問題
4. 證明存在  $P_n$  的問題可以用同樣的策略變成最好的  $P_{n-1}$  問題
5. 根據數學歸納法得證

Sprout

## 數學歸納法 vs 遷迴證明法

根本就是數學歸納法！

1. 證明當  $n = 1$  時，命題成立
2. 假設  $P_{n-1}$  的所有問題，貪心策略都正確  
⇒ Optimal substructure
3. 考慮任何一個  $P_n$  的問題
4. 證明存在  $P_n$  的問題可以用同樣的策略變成最好的  $P_{n-1}$  問題  
⇒ Greedy-choice property
5. 根據數學歸納法得證



## 直接反證法

- 一定要像影片一樣用相鄰逆序對嗎？
- 不同的解一定會有相鄰逆序對嗎？
  - 假如決策的數量或是選擇可以不一樣就爆掉了
  - 就算一樣，這部分可能還是需要證明才完整
  - 最終還是需要對逆序對數學歸納法

Sprout

## 直接反證法

- 一定要像影片一樣用相鄰逆序對嗎？
- 不同的解一定會有相鄰逆序對嗎？
  - 假如決策的數量或是選擇可以不一樣就爆掉了
  - 就算一樣，這部分可能還是需要證明才完整
  - 最終還是需要對逆序對數學歸納法
- 有沒有更簡單的證明？

Sprout



## 經典問題 — Minimum Segment Cover

### Problem Minimum Segment Cover

你是伺服器管理員，你需要請工人來幫忙監控你的伺服器。現在有  $N$  個候選人，第  $i$  個可以從時間點  $L_i$  工作到時間點  $R_i$ 。

如果伺服器要從時間 0 開始運轉到時間  $T$ ，你最少需要請多少人來監控伺服器？

Sprout

## 經典問題 — Minimum Segment Cover

- 假設你已經請了一些工人，但是伺服器會在時間  $T'$  沒有人顧
- 這時候你必定需要再請一個人滿足  $L_i \leq T' \leq R_i$
- $L_i$  是多少其實不重要，只要不超過  $T'$  就好
- $R_i$  越大越好，貪心的選最大的！
- 一旦抵達  $T$  就停止

Sprout

## 經典問題 — Minimum Segment Cover

- 假設你已經請了一些工人，但是伺服器會在時間  $T'$  沒有人顧
- 這時候你必定需要再請一個人滿足  $L_i \leq T' \leq R_i$
- $L_i$  是多少其實不重要，只要不超過  $T'$  就好
- $R_i$  越大越好，貪心的選最大的！
- 一旦抵達  $T$  就停止
- 感覺很好，我們盡量拖延決策，並且做出當下最好的決策
- 不過不足以說明這一定是最佳解

Sprout



## 經典問題 — Minimum Segment Cover

嘗試套用投影片的方法...

- 集合裡的東西根本就可以不一樣，逆序數對是 0 也不代表一樣
- 甚至不知道要怎麼排序

Sprout



## 經典問題 — Minimum Segment Cover

證明草稿：

選某個最佳解  $S^*$ ，然後說明我們得出來的解  $S$  可以換成這個  $S^*$  所以不會變差。

- 對於兩個解不一樣的元素們，怎麼知道哪些要換成哪些？
- 要是能找到某個順序，例如工作的順序就會容易許多

**Sprout**



## 經典問題 — Minimum Segment Cover

### Observation 觀察

最佳解必不包含兩個人  $i, j$  使得  $L_i \leq L_j \leq R_j \leq R_i$ 。

這件事情相當顯然，因為  $j$  這個人拿掉就會是嚴格更好的解。

Sprout

## 經典問題 — Minimum Segment Cover

- 對每個工作人員開始的時間排序，拿任意一個最佳解  $S^*$  與我們的解  $S$  來比較
- 如果  $S^*$  是  $S$  的前綴，那們  $S^* = S$ ，因為我們的演算法在拿到  $S^*$  的最後一個就會中止。
- 如果第一個不一樣的人最佳解選擇了  $s'$  而我們選擇了  $s$ ，考慮直接把  $s'$  換成  $s$ ，最佳解仍然合法嗎？

## 經典問題 — Minimum Segment Cover

- 對每個工作人員開始的時間排序，拿任意一個最佳解  $S^*$  與我們的解  $S$  來比較
- 如果  $S^*$  是  $S$  的前綴，那們  $S^* = S$ ，因為我們的演算法在拿到  $S^*$  的最後一個就會中止。
- 如果第一個不一樣的人最佳解選擇了  $s'$  而我們選擇了  $s$ ，考慮直接把  $s'$  換成  $s$ ，最佳解仍然合法嗎？

### Proof

假設最後一個一樣的人是  $t$ ，如果  $t$  存在那麼會有  $L_{s'} \leq R_t$ ，如果  $t$  不存在就會有  $L_{s'} \leq 0$ 。

根據我們 greedy 演算法的性質，如果  $t$  存在  $s$  必定是  $L_s \leq R_t$  中  $R_s$  最大的，不存在則是  $L_s \leq 0$  中  $R_s$  最大的。

所以我們必定有  $R_{s'} \leq R_s$ ，因此換過去必定不會使這個解不合法



## 經典問題 — Minimum Segment Cover

- 對每個工作人員開始的時間排序，拿任意一個最佳解  $S^*$  與我們的解  $S$  來比較
- 如果  $S^*$  是  $S$  的前綴，那們  $S^* = S$ ，因為我們的演算法在拿到  $S^*$  的最後一個就會中止。
- 如果第一個不一樣的人最佳解選擇了  $s'$  而我們選擇了  $s$ ，考慮直接把  $s'$  換成  $s$ ，最佳解仍然合法嗎？

### Proof

假設最後一個一樣的人是  $t$ ，如果  $t$  存在那麼會有  $L_{s'} \leq R_t$ ，如果  $t$  不存在就會有  $L_{s'} \leq 0$ 。

根據我們 greedy 演算法的性質，如果  $t$  存在  $s$  必定是  $L_s \leq R_t$  中  $R_s$  最大的，不存在則是  $L_s \leq 0$  中  $R_s$  最大的。

所以我們必定有  $R_{s'} \leq R_s$ ，因此換過去必定不會使這個解不合法

□

- 只不過這樣還沒有辦法好好證明，我們仍然逃不了數學歸納法

善用反證法！

- 假設沒有任何一個最佳解與我們的解相同
- 對每個工作人員開始的時間排序，拿與我們的解  $S$  **有最長共同前綴的最佳解  $S^*$**  來比較
- 如果  $S^*$  是  $S$  的前綴，那麼  $S^* = S$ ，因為我們的演算法在拿到  $S^*$  的最後一個就會中止，矛盾
- 所以必定有一個位置  $i$  使得  $S_i^* \neq S_i$ ，設  $S_i^* = s'$  而  $S_i = s$
- 我們嘗試製造另一個解  $S' = S^* \cup \{s\} \setminus \{s'\}$
- $S'$  也是其中一個最佳解（根據剛剛的證明）
- 這樣就能更快得到矛盾： $S'$  與  $S$  的共同前綴比  $S^*$  多 1！



## 證明方法

- 數學歸納法就沒用了嗎？
- 只有這兩種證明方法嗎？

Sprout



還有題目

Sprout



## 更多 Greedy

### Problem

你有  $n$  個食物，第  $i$  個食物的過期日期是  $e_i$ ，你可以在它過期前吃它，並且你接下來的  $d_i$  天內會很開心。你可以在任何你想要的時間吃任何東西（沒過期的話，且每個食物只能吃一次），求你最多可以開心多少天。

設計一個  $O(n \log n)$  的演算法並證明正確性。

Sprout

### Problem Stable Marriage

有  $n$  個男生跟  $n$  個女生要兩兩配對結婚，每個男生都對  $n$  位女生分別列出從最喜歡到最不喜歡，反過來女生也為男生分別列了這個列表。

能不能找到一組一男一女配對的方法，使得不會有一個男生  $A$  跟一個女生  $B$ ，滿足男生  $A$  比較喜歡女生  $B$  勝過於匹配到的女生，而且女生  $B$  比較喜歡男生  $A$  勝過於匹配到的男生？



## Tree Matching

### Problem Tree Matching

有一棵  $n$  個點的樹，如果兩個點間有一條邊就可以把他們配起來。在一個點只能被配到一次的情況下，最多可以匹配幾組點？

Sprout