C/C++ 進階班

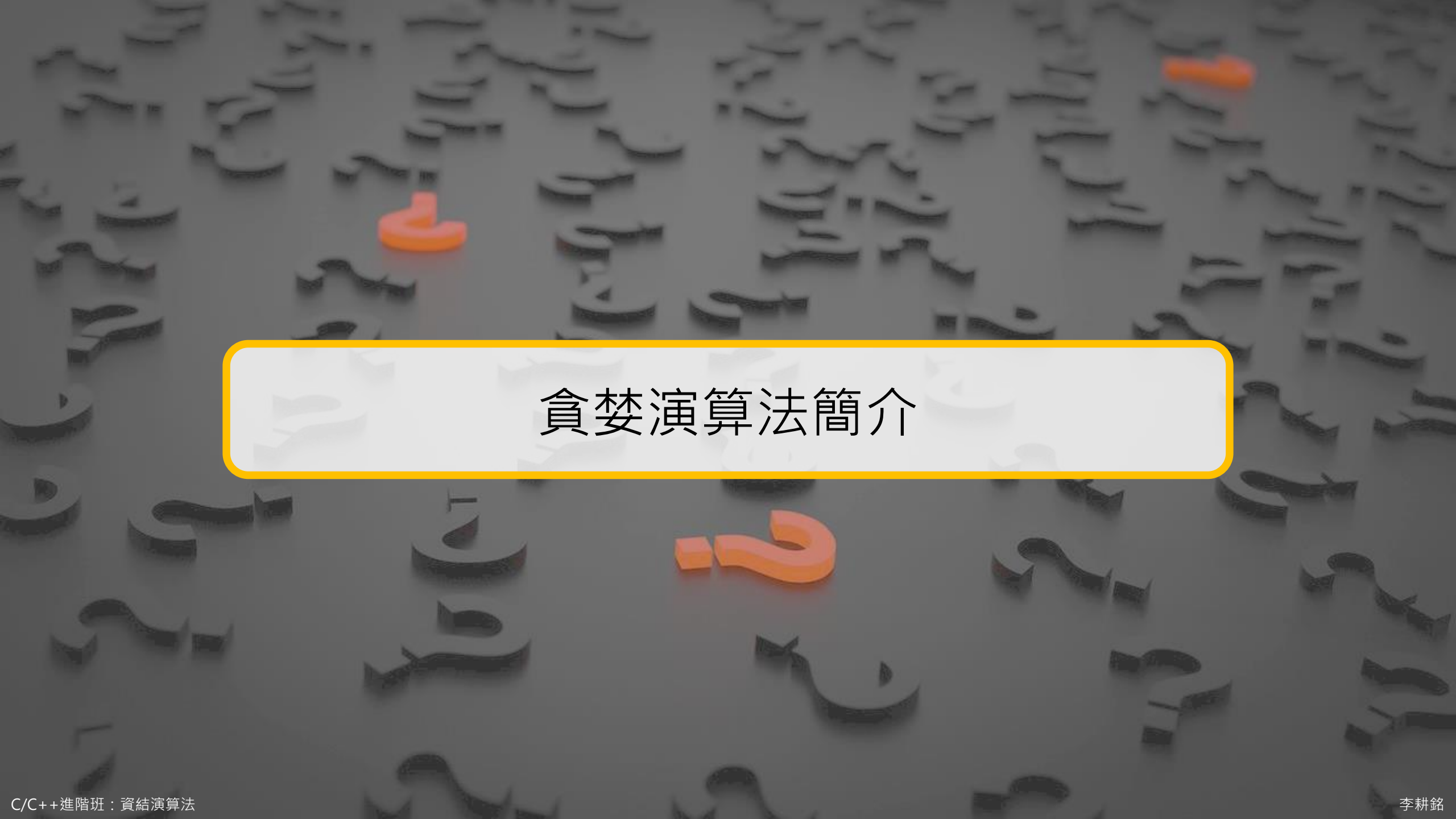
資結演算法

貪婪演算法 (Greedy Algorithm)

李耕銘

課程大綱

- 貪婪演算法簡介
- 貪婪演算法應用
  - 找錢問題 (Coin Changing)
  - 中途休息 (Breakpoint Selection)
  - 活動選擇問題 (Activity–Selection Problem)
  - 背包問題 (Fractional Knapsack Problem)
  - 工作排程 (Task Scheduling)
  - 最小延遲 (Scheduling to Minimize Lateness)
- 貪婪演算法總結
- 實戰練習



貪婪演算法簡介

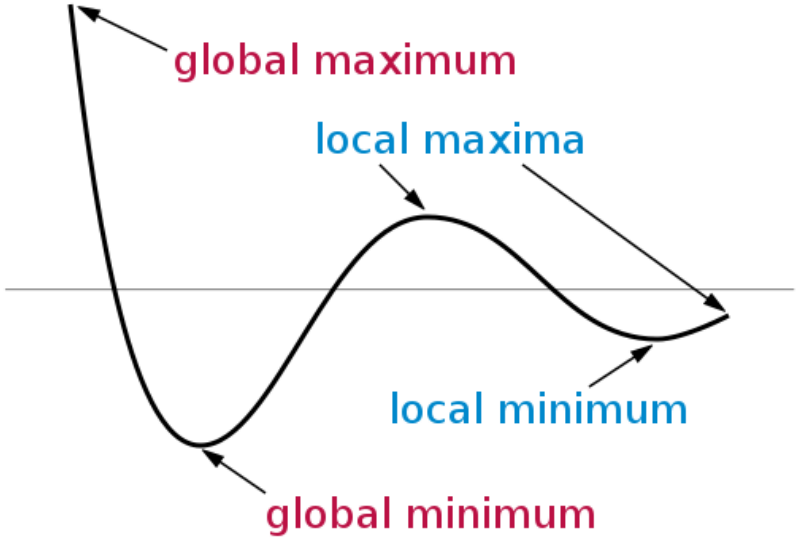
貪婪演算法簡介

- 短視/近利/偷懶/貪婪的想法
- 在每一步選擇時選當下局部最佳解
  - ➢ 非本科系想在最短時間內當上有名的律師
  - ➢ 就去選能最快考律師執照的方式：法律學分班

貪婪演算法簡介

- 短視/近利/偷懶/貪婪的想法
- 在每一步選擇時選當下局部最佳解
  - 登山迷路時想回到山下的城市
  - 一步步選擇下坡路往下走

貪婪演算法簡介

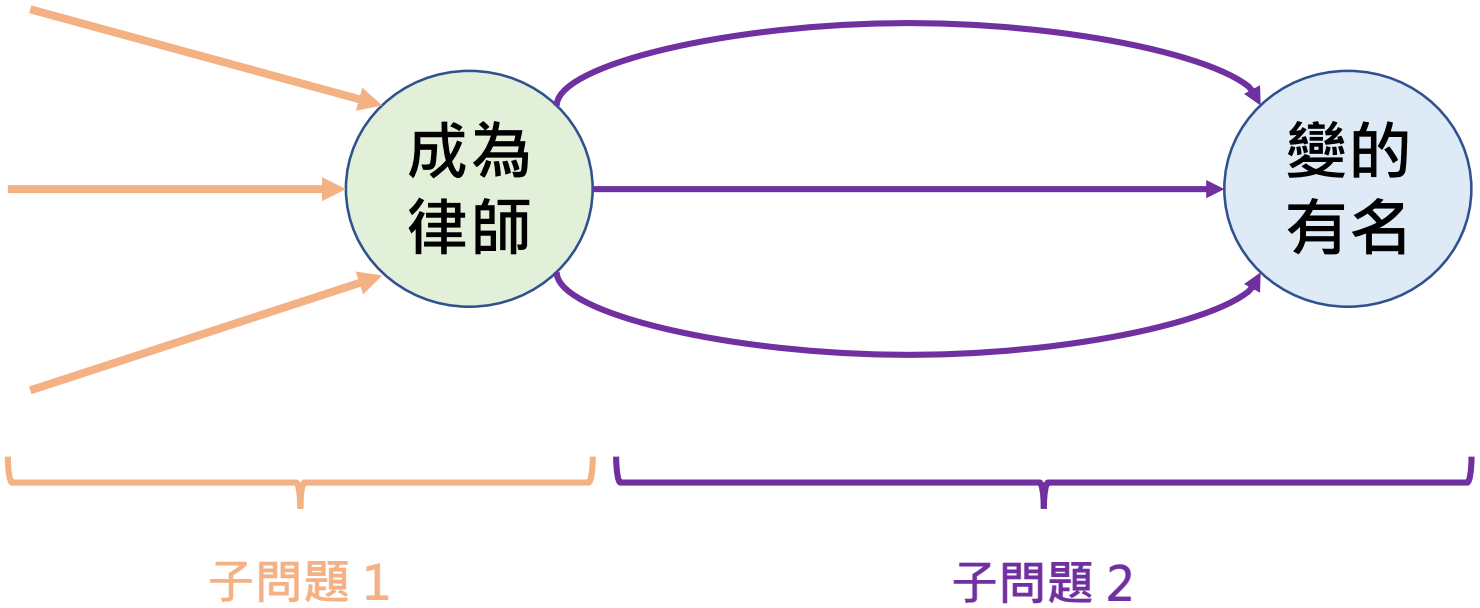


登山迷路時，選擇下坡路走就能回到山下？

- 可行嗎？為什麼不？
  - ✓ 最後可能會被困在山谷中
  - ✓ 往下坡的路不一定通往城市
  - ✓ 走的每一步都會影響到下一步
  - ✓ 選擇的次序也會有影響！
  - ✓ 最終會掉到局部最低點 (Local Minimum)

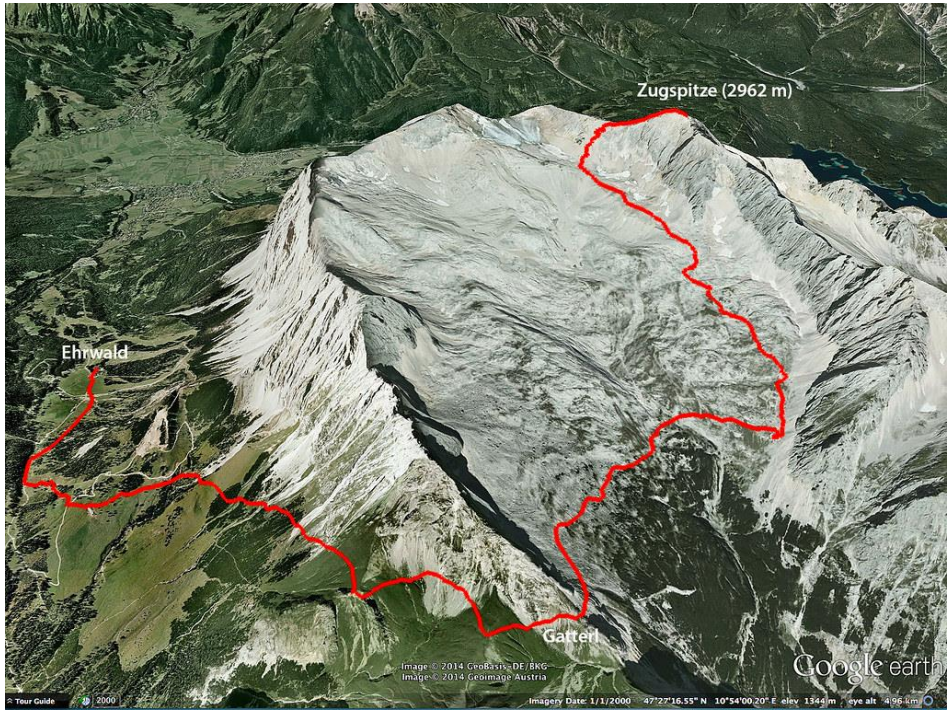
貪婪演算法簡介

- 為什麼大律師問題可以用貪婪解？
  - 大律師問題可以拆解成兩個子問題
    1. 成為律師
    2. 變的有名
  - 子問題 1 的選擇不會影響到子問題 2



貪婪演算法簡介

- 為什麼登山問題不可以用貪婪解？
  - ➢ 登山問題可以拆解成眾多子問題
    - ✓ 每次叉路的選擇就是一個子問題
  - ➢ 子問題的選擇會影響到下個子問題



貪婪演算法簡介

- 大律師問題跟下山問題有甚麼不同？
  - 大律師問題可以拆解成兩個子問題
    1. 成為律師
    2. 變的有名
    - 每一步都互相獨立
  - 下山問題可以拆解成眾多子問題
    - ✓ 每一個叉路要往哪裡走
    - 每一步都互相相關、彼此影響

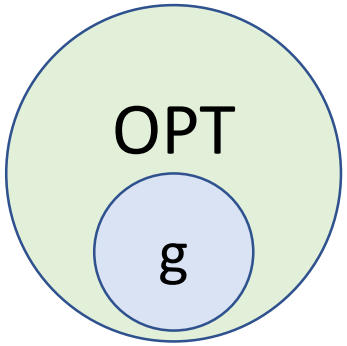
貪婪演算法簡介

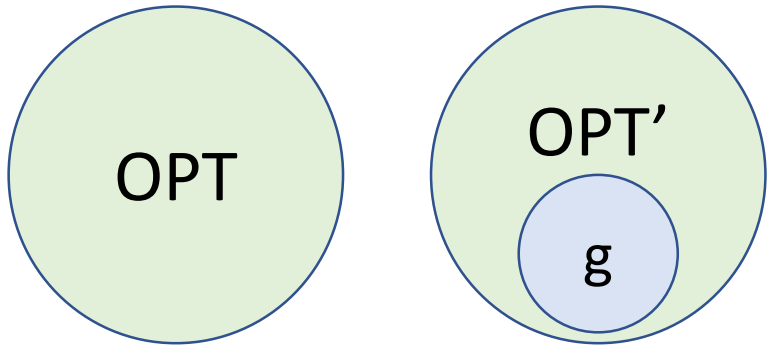
1. 我想要賺大錢！
   A. 工作選擇
      a) 跑 UbXX 賺 250元/hr
      b) 跑 熊X 賺 220元/hr
   B. 行程選擇
      a) 好好讀書考國考 0元/hr
      b) 去打工賺 200元/hr
2. 哪個可以用貪婪？

貪婪演算法簡介

- 適用貪婪演算法的特性
  1. 最佳化子結構
     - ✓ 子問題的最佳解一定在原問題的最佳解內
  2. 適用貪婪選擇
     - ✓ 局部最佳解能夠組成全域最佳解

貪婪演算法證明





- 利用 Exchange Arguments 證明貪婪演算法
  - 目前子問題的貪婪最佳解為 g
  - 任意最佳解為 OPT
  - 產生兩種情形
    1. g 在 OPT 中→沒問題
    2. g 不在 OPT 中

       →修改 OPT 成為 OPT’ 且 OPT’ 包含 g

       A. 如果 OPT’ 比 OPT 好，反證

       B. 如果 OPT’ 跟 OPT 一樣好，得證

貪婪演算法簡介

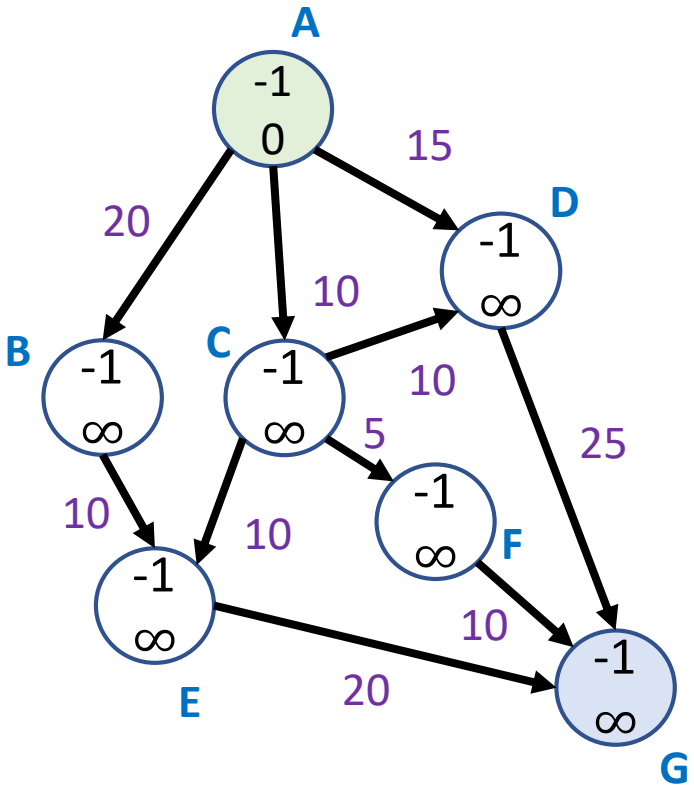
- 貪婪演算法的步驟
  1. 把原問題切出至少一個子問題
     - ✓ 解決這個子問題
  2. 不斷切割出子問題後解決之
  3. 把所有子問題的答案拼湊起來
  4. 證明局部子問題的答案會構成全域最佳解

貪婪演算法簡介

貪婪演算法總結

- 解決最佳化問題時，通常很有效率且簡單
  - 按照順序一步步找出目前最佳解就可以了
- 適用貪婪演算法的狀況
  - 子問題的最佳解一定在母問題的最佳解
- 貪婪演算法並不能解決所有最佳化問題
  - 問題間的選擇有相關性
  - 每一步選擇會影響到下一步

Dijkstra's Algorithm



1. 初始化 Distance 與 Predecessor
2. 分成已經找到最短路徑跟還沒找到最短路徑兩組
3. 從還沒找到最短路徑中找到 Distance 最小的頂點 (V)
4. Relax V，V 就可放入已經找到最短路徑組
5. 重複步驟 2~4 共 |V| 次
6. 直到所有頂點都被放入已經找到最短路徑組

最短路徑組

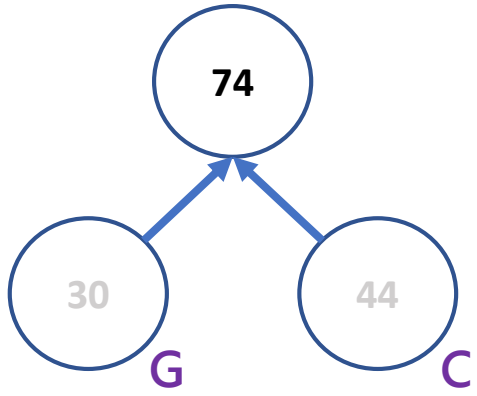
- A(-1,0)

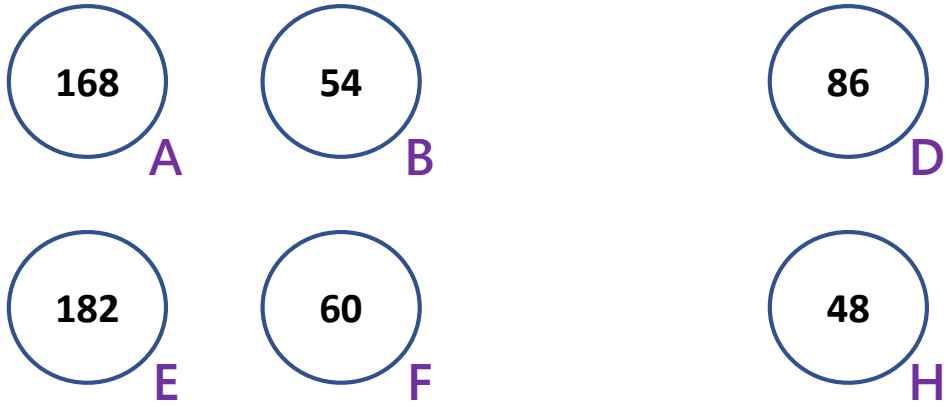
尚未是最短路徑

- B(-1,∞),C(-1,∞),D(-1,∞),E(-1,∞),F(-1,∞),G(-1,∞)

霍夫曼編碼 (Huffman Coding)

- 霍夫曼編碼步驟
  1. 建立字頻表
  2. 每個字元與頻率視為一個節點
  3. 最小的兩節點依序聚合成子樹
     - 父節點為子節點的頻率和



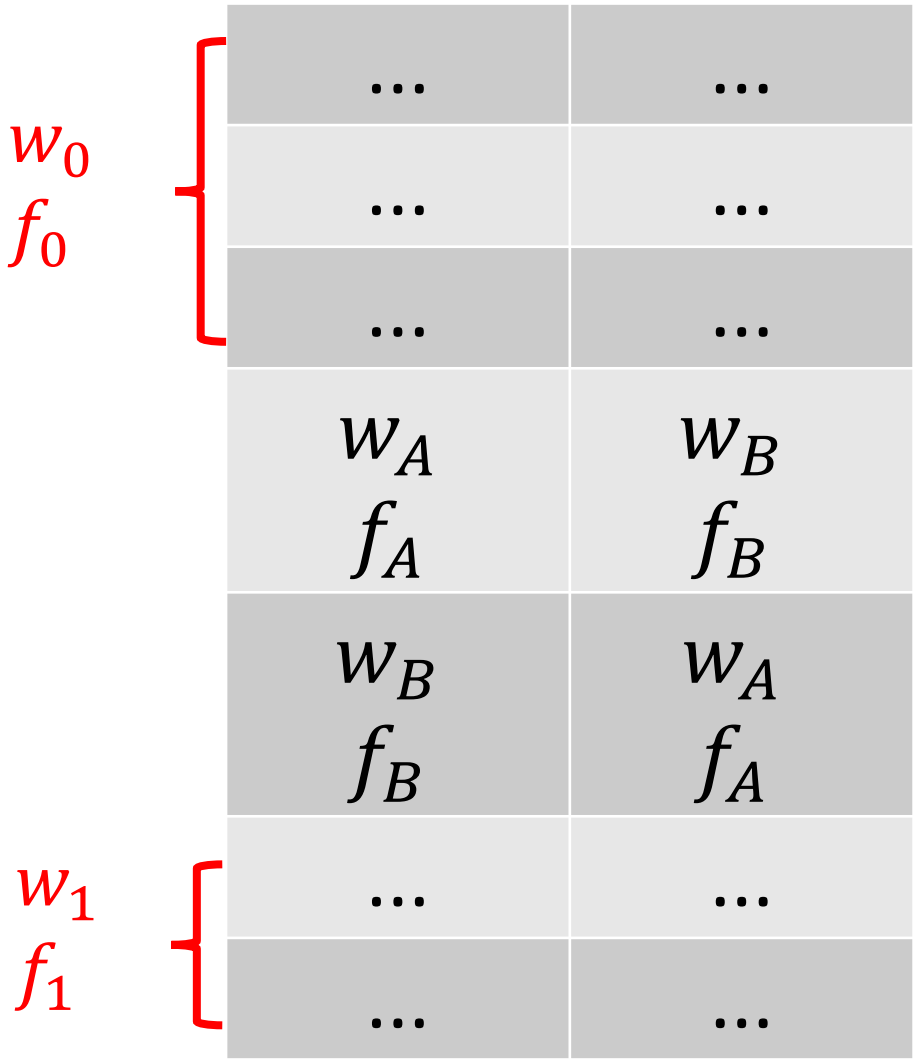


APCS: 物品堆疊

某個自動化系統中有一個存取物品的子系統，該系統是將 N 個物品堆在一個垂直的貨架上，每個物品各佔一層。系統運作的方式如下：每次只會取用一個物品，取用時必須先將在其上方的物品貨架升高，取用後必須將該物品放回，然後將剛才升起的貨架降回原始位置，之後才會進行下一個物品的取用。

每一次升高某些物品所需要消耗的能量是以這些物品的總重來計算，在此我們忽略貨架的重量以及其他可能的消耗。現在有 N 個物品，第 i 個物品的重量是 w(i)而需要取用的次數為 f(i)，我們需要決定如何擺放這些物品的順序來讓消耗的能量越小越好。舉例來說，有兩個物品 w(1)=1、w(2)=2、f(1)=3、f(2)=4，也就是說物品 1 的重量是 1 需取用 3 次，物品 2 的重量是 2 需取用 4 次。

APCS: 物品堆疊



Given 2 object A and B :
W_A *is the weight of* A
f_A *is the frequency of* A
W_B *is the weight of* B
f_B *is the frequency of* B

Condition 1: *B is under A*

$$Total\ cost(1) = f_1 \times (w_0 + w_A + W_B) + f_B \times (w_0 + w_A) + f_A \times (w_0)$$

Condition2: *A is under B*

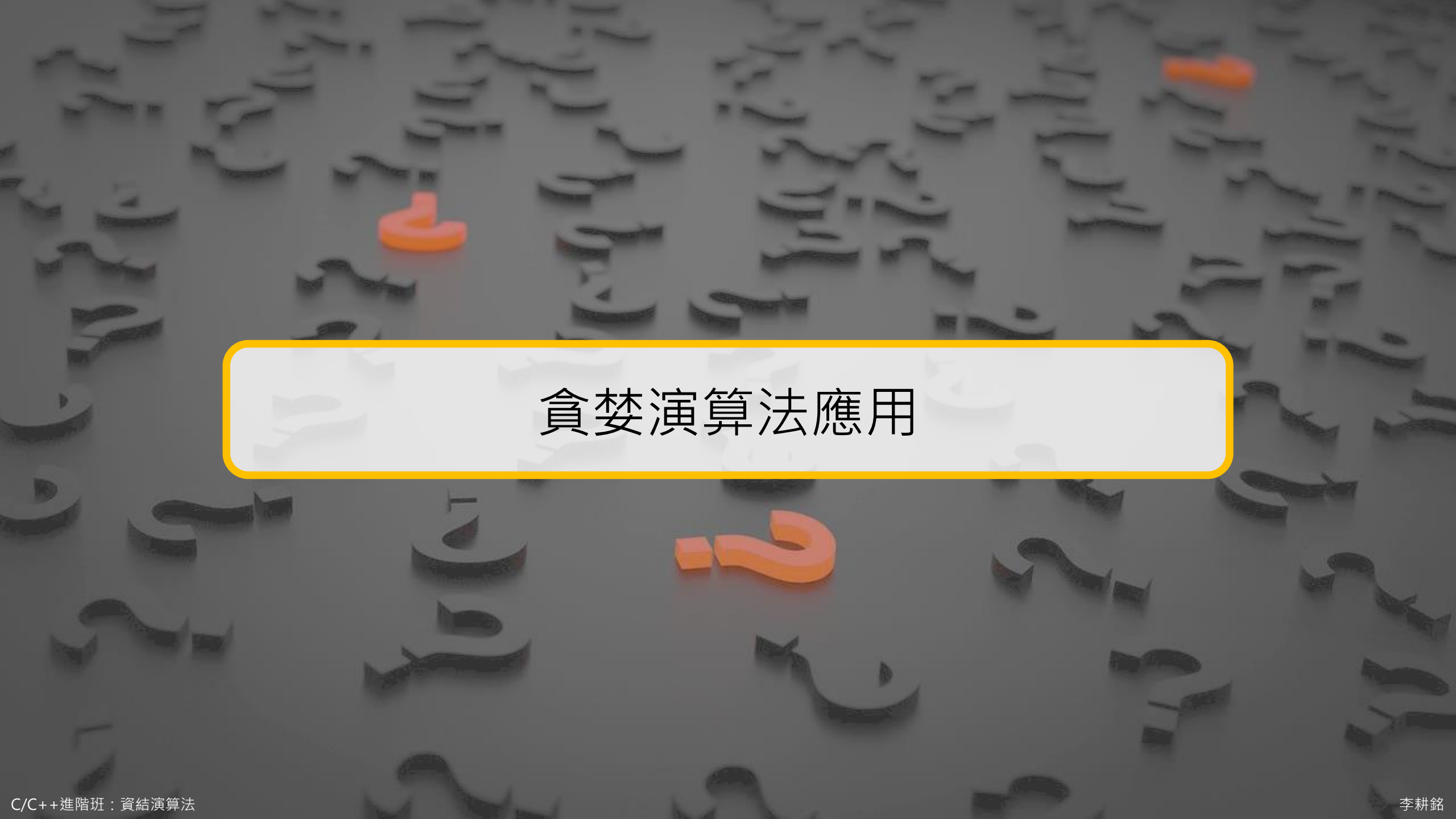
$$Total\ cost(2) = f_1 \times (w_0 + w_A + W_B) + f_A \times (w_0 + w_B) + f_B \times (w_0)$$

$$Total\ cost(1) - Total\ cost(2) = f_B w_A - f_A w_B$$

if $f_B w_A > f_A w_B$:
put A under B

if $f_B w_A < f_A w_B$:
put B under A

Conclustion :
Sort the objects by $f_B w_A$



貪婪演算法應用

找錢問題

顧客需要找 n 元，而目前有四種硬幣：

1. 1 元
2. 5 元
3. 10 元
4. 50 元

如何找錢可以讓找的硬幣「數目」最少？

找錢問題

貪婪演算法：每次都從四枚硬幣中最大幣值找起

有 128 元要找：

1. 50 元硬幣找 2 枚
2. 10 元硬幣找 2 枚
3. 5 元硬幣找 1 枚
4. 1 元硬幣找 3 枚

找錢問題

利用反證法證明：

假設第一次用找 50 元硬幣兩枚並不是最佳解。

1. 50 元只能用 0 枚或 1 枚
2. 10 元硬幣只能用 0 ~ 4 枚
   - 否則可以用 1 枚 50 元替代 5 枚 10 元
3. 5 元硬幣只能用 0 枚或 1 枚
   - 否則可以用 1 枚 10 元替代 2 枚 5 元
4. 1 元硬幣只能用 0 ~ 4 枚
   - 否則可以用 5 元替代 5 枚 1 元
5. 最多只能找：$50 \times 1 + 10 \times 4 + 5 \times 1 + 1 \times 4 = 99$

找錢問題

假設 C(n) 是 n 元問題所需的硬幣數，OPT 是最佳解，

故有以下四種狀況：

1. 裏頭有一元硬幣
   - OPT - $coin_1$ 也是 C(n-1) 的最佳解
2. 裏頭有五元硬幣
   - OPT - $coin_5$ 也是 C(n-5) 的最佳解
3. 裏頭有十元硬幣
   - OPT - $coin_{10}$ 也是 C(n-10) 的最佳解
4. 裏頭有五十元硬幣
   - OPT - $coin_{50}$ 也是 C(n-50) 的最佳解

$$coin(i) = \min_j(1 + coin(i - v_j))$$

Practice

Mission

Try LeetCode #860. Lemonade Change

At a lemonade stand, each lemonade costs $5. Customers are standing in a queue to buy from you, and order one at a time (in the order specified by bills).

Each customer will only buy one lemonade and pay with either a $5, $10, or $20 bill. You must provide the correct change to each customer, so that the net transaction is that the customer pays $5.

Note that you don ‘t have any change in hand at first. Return true if and only if you can provide every customer with correct change.

Ref : https://leetcode.com/problems/lemonade-change/

Practice

Mission

Try LeetCode #322. Coin Change

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Ref：https://leetcode.com/problems/coin-change/

找錢問題

顧客需要找 25 元，而目前有三種硬幣：

1. 1 元
2. 8 元
3. 10 元

如何找錢可以讓找的硬幣「數目」最少？

貪婪演算法可行嗎？

找錢問題

顧客需要找 25 元，而目前有三種硬幣：1、8、10 元

貪婪演算法：

1. 10 元硬幣 2 枚
2. 8 元硬幣 0 枚
3. 1 元硬幣 5 枚

2 + 0 + 5 = 7，共七枚

找錢問題

顧客需要找 25 元，而目前有三種硬幣：1、8、10 元

另一種找錢法：

1. 10 元硬幣 0 枚
2. 8 元硬幣 3 枚
3. 1 元硬幣 1 枚

0 + 3 + 1 = 4，共四枚

找錢問題

1. 四種硬幣：1、5、10、50 元
2. 三種硬幣：1、8、10 元

為何第一種情況可以用貪婪，第二種卻不行？

第一種狀況多枚小硬幣都可以用大硬幣替代

所以子問題間不相關！

用貪婪前請檢查子問題間是否相依！

相依的話請用動態規劃

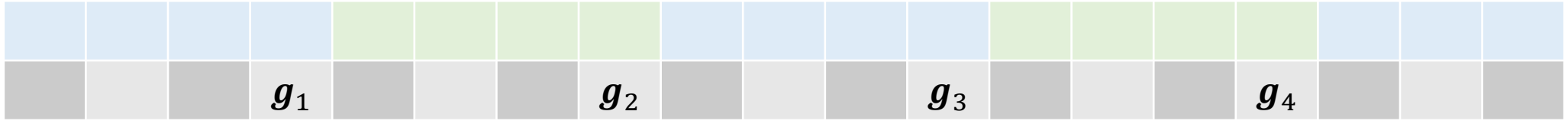
中途休息

有一段距離為 d 的旅途需要行駛，油箱的容量每次只能連續行走 c，給定一連串的加油站位置 g_i，每次加油後都能夠繼續行走 c，該如何安排加油的地點可以讓停下來的次數最少？

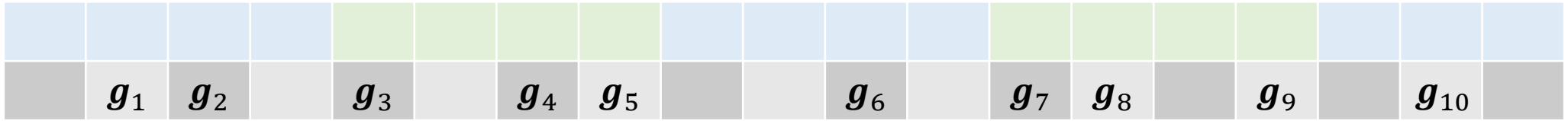
貪婪演算法：每次加油之前盡可能地走越遠越好

中途休息

理想狀況：每次耗盡油量時剛好都有加油站

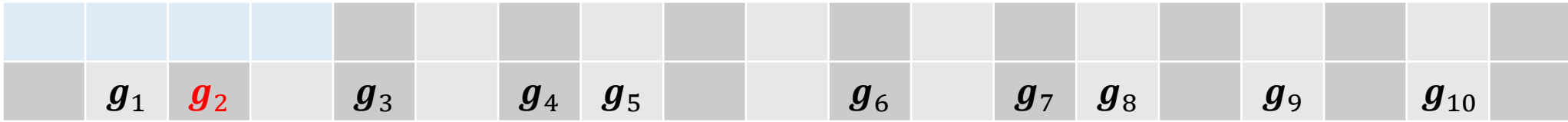


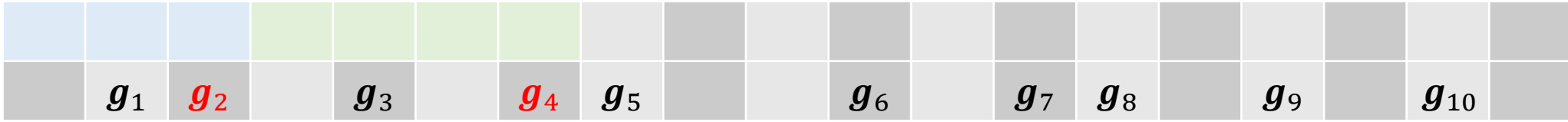
現實狀況：每次耗盡油量時不一定有加油站

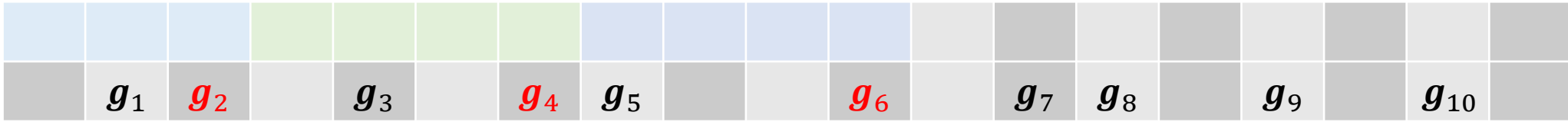


中途休息

現實狀況：選每次耗盡油量前的最後一個加油站

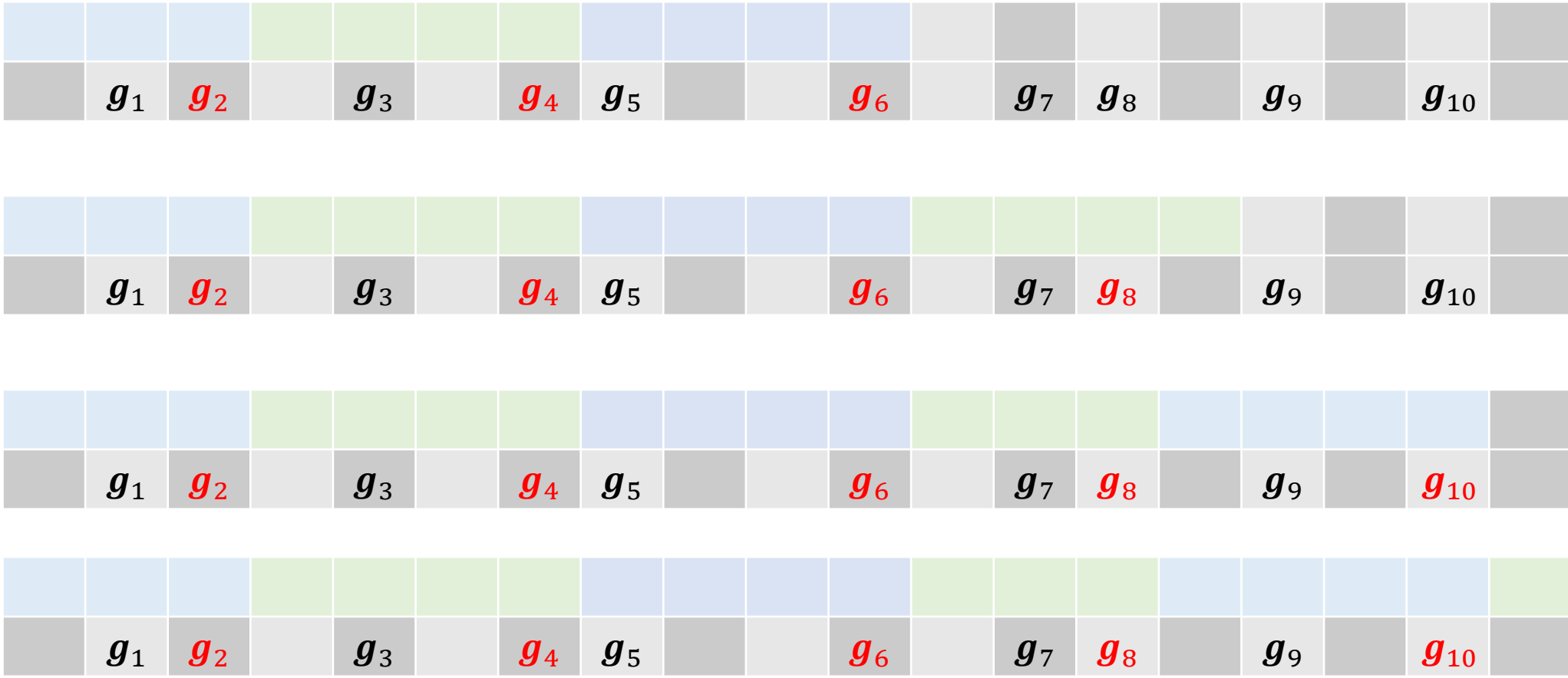




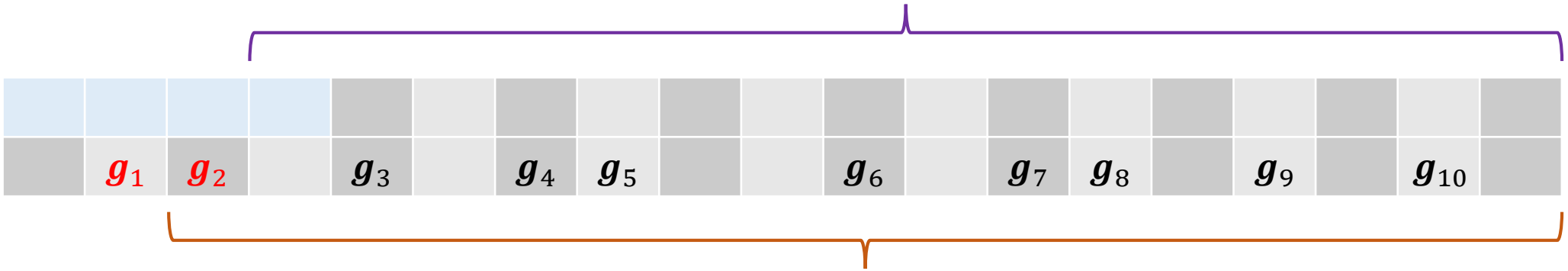


中途休息

現實狀況：選每次耗盡油量前的最後一個加油站



中途休息



貪婪的選擇：g_2

1. 選擇 g_2 後的 → 汽油用罄，不夠
2. 選擇 g_2 前的 → 必可以用 g_2 取代

Example Code

Mission

讓使用者輸入每次加滿油可行走的距離 c、旅途總長度 d、加油站數目 n 以及每個加油站的座標，請輸出這趟旅途中至少要停下來加油幾次。

```
Please enter c, d, n:
4 19 10
2 3 5 7 8 11 13 14 16 18
Stop @3
Stop @7
Stop @11
Stop @14
Stop @18
Stops = 5
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;


int main() {
    int c,d,n;
    cout << "Please enter c, d, n:" << endl;
    cin >> c >> d >> n;
    vector<int> gas_station(n);

    for(int i=0;i<n;i++)
        cin >> gas_station[i];

    sort(gas_station.begin(),gas_station.end());
    int stops = 0;
    // Code from this line!

    cout << "Stops = " << stops << endl;

    return 0;
}
```

活動選擇問題

學校只有一個音響，但有 n 個活動需要它，今天給定每個活動的時間長度，如何讓舉辦的活動數目最多？

	Days									
1										
2										
3										
4										
5										

活動選擇問題

活動日程完全在另一活動之內的，優先考慮之

優先考慮活動 5 不考慮活動 1！

	Days									
1										
2										
3										
4										
5										

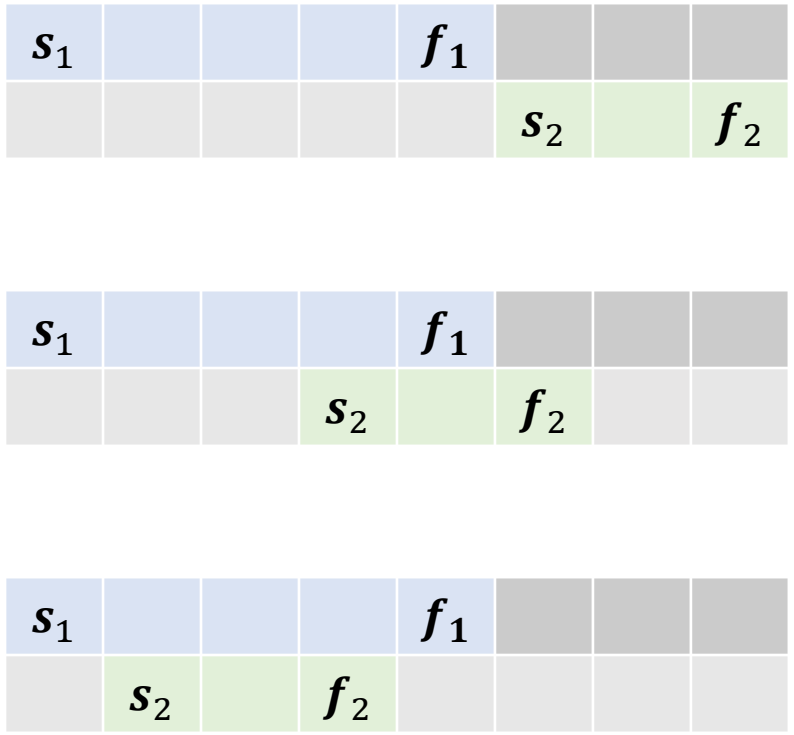
活動選擇問題

貪婪演算法步驟

1. 把活動依照開始日期進行排序！

	Days									
1										
2										
3										
4										
5										

活動選擇問題



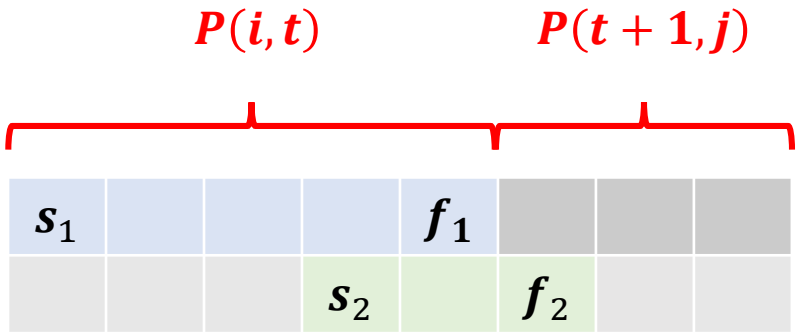
貪婪演算法步驟

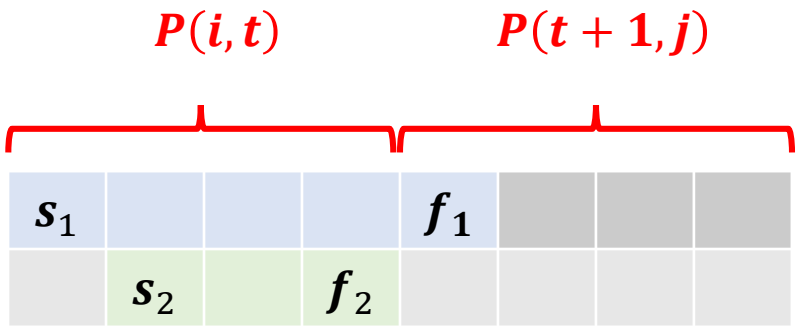
2. 檢查第一個開始活動 a_1 的區間，起點 s_1、終點 f_1
3. 檢查第二個開始活動 a_2 的區間，起點 s_2、終點 f_2
   - A. 若 $s_2 > f_1$，採納活動 a_1
   - B. 若 $s_2 \leq f_1$
     - A. 若$f_2 > f_1$採納活動 a_1
     - B. 若$f_2 \leq f_1$採納活動 a_2

當活動有重疊的時候，每次都選最早結束的那個。

這樣對嗎？

活動選擇問題





有兩個衝突，究竟該選誰？

- 重疊的區域中，最早結束的時間為 t
  1. 把問題 $P(i,j)$ 切割成 $P(i,t)$ 與 $P(t+1,j)$
  2. 選擇最小的 t 可以讓 $P(t+1,j)$ 區間最大化
  3. 最大化的 $P(t+1,j)$ 能夠塞入最多的活動
  4. 故：$1+P(t+1,j)=P(i,j)$
- 這裡不給嚴謹的證明。

Practice

Mission

Try LeetCode #435. Non-overlapping Intervals

Given an array of intervals intervals where intervals[i] = [$start_i$, end_i], return the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

- Example 1:
  - Input: intervals = [[1,2],[2,3],[3,4],[1,3]]
  - Output: 1
  - Explanation: [1,3] can be removed and the rest of the intervals are non-overlapping.

Ref：https://leetcode.com/problems/non-overlapping-intervals/

活動選擇問題

如果活動有權重 (重要性有別) 的話呢？

- 請使用動態規劃！

		Days									
1	w_1										
2	w_2										
3	w_3										
4	w_4										
5	w_5										

背包問題

給定固定的背包負重 W，以及每一個物品的重量 w_i 及價值 v_i，如何在不超過負重的狀況下，讓背包裏頭的物品價值最貴重？

w_i	4	5	2	6	3	7
v_i	6	7	5	12	5	18

背包問題

背包問題分類：

1. 0/1 Knapsack Problem
   - ➢ 每項物品只能拿一個
2. Unbounded Knapsack Problem
   - ➢ 每項物品可以拿無限多個
3. Multidimensional Knapsack Problem
   - ➢ 背包空間有限
4. Multiple Choice Knapsack Problem
   - ➢ 每一類物品最多拿一個
5. Fractional Knapsack Problem
   - ➢ 物品可以只拿部分

背包問題

Fractional Knapsack Problem

可以只拿一部分的物品，不用全部都拿，但每個物品的上限最多只能拿一個。

- 依序從單位價值最高的物品放

w_i	4	5	2	6	3	7
v_i	6	7	5	12	5	18
$\frac{v_i}{w_i}$	1.5	1.4	2.5	2	1.67	2.57

背包問題

假設背包負重為 16：

1. 編號 6 物品拿 7，value = 18，剩餘負重 = 9
2. 編號 3 物品拿 2，value = 23，剩餘負重 = 7
3. 編號 4 物品拿 6，value = 35，剩餘負重 = 1
4. 編號 5 物品拿 1，value = $36\frac{2}{3}$，剩餘負重 = 0

$index$	1	2	3	4	5	6
$\boldsymbol{w_i}$	4	5	2	6	3	7
$\boldsymbol{v_i}$	6	7	5	12	5	18
$\frac{\boldsymbol{v_i}}{\boldsymbol{w_i}}$	1.5	1.4	2.5	2	1.67	2.57

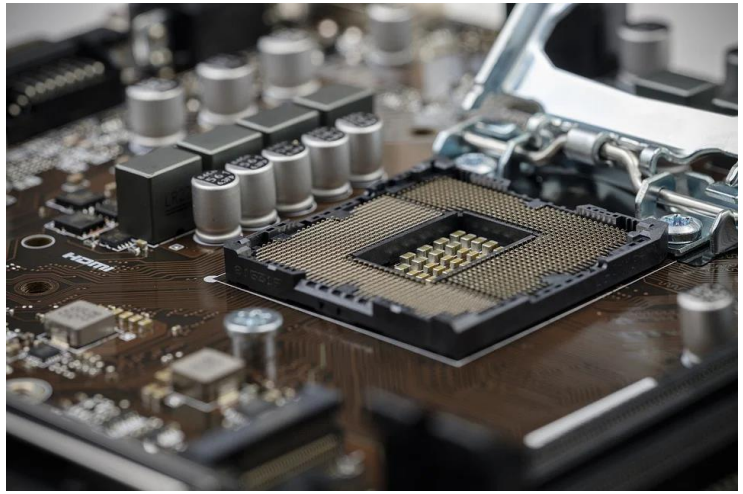
背包問題

貪婪的證明：假設 F-K(n) 是負重 n 的背包所能有的價值，OPT 是問題的最佳解

1. CP 值最大的物品 $\mathbf{Obj}(w_i, v_i)$ 在 OPT 中
   - $\mathbf{Obj}(w_i, v_i)$ 移走後仍為 F-K(n-w_i) 的最佳解
2. CP 值最大的物品 $\mathbf{Obj}(w_i, v_i)$ 不在 OPT 中
   - 可以把 OPT 中的物品用 $\mathbf{Obj}(w_i, v_i)$ 替換
   - OPT 不為最佳解，反證之

- **其他的背包問題可以這樣解嗎？**

工作排程



給定一連串回家作業 H_i，每份作業都有各自的繳交期限 D_i、遲繳的扣分 P_i，已知每份作業都需要一天的時間完成，該如何安排寫作業的順序可以讓遲繳的扣分最少？

D_i	4	5	4	2	3	2
P_i	8	3	2	6	5	2

工作排程

當繳交順序 $S_i > D_i$ 時，就需要接受扣 P_i 分。

以下圖為例，需要扣 6 + 5 +2 = 13 分

S_i	1	2	3	4	5	6
D_i	4	5	4	2	3	2
P_i	8	3	7	6	5	2

工作排程

當繳交順序 $S_i > D_i$ 時，就需要接受扣 P_i 分。

來貪婪一下，以死線的順序來寫作業呢？

需要扣 7 +3 = 10 分

S_i	1	2	3	4	5	6
D_i	2	2	3	4	4	5
P_i	6	2	5	8	7	3

工作排程

當繳交順序 $S_i > D_i$ 時，就需要接受扣 P_i 分。

改以扣分的大小順序來寫作業呢？

需要扣 6 + 5 + 2 = 13 分，反而更差了！

S_i	1	2	3	4	5	6
D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

工作排程

S_i	1	2	3	4	5	6
D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

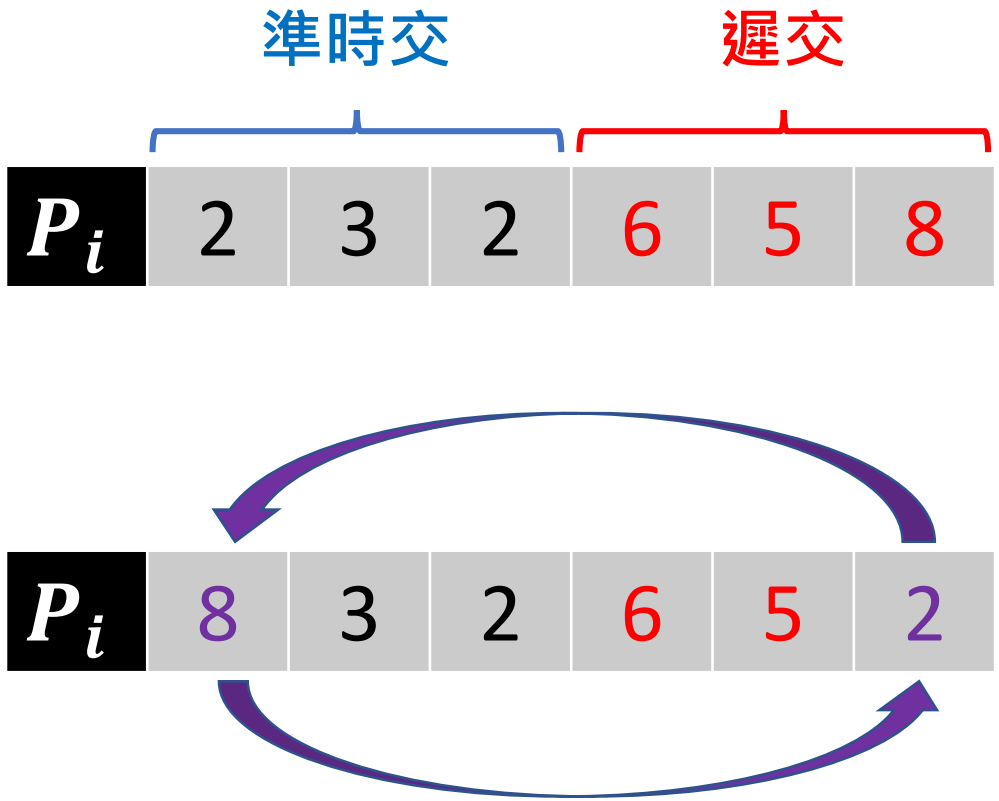
準時交　　遲交

S_i	1	2	3	4	5	6
D_i	4	4	5	2	3	2
P_i	8	7	3	6	5	2

觀察一下問題的特色：

- 遲交就是遲交，不管晚多久都是遲交
- 把作業分成兩組：
  1. 準時交
  2. 遲交
  - ➢ 準時交或遲交的順序不影響扣分

工作排程



換個方法做貪婪：

- 每次想辦法準時交扣分最重的作業

Proof：

假設 P_i 為扣份最重，假設 H_i 不在最佳解中，如果能透過兩兩交換順序讓 H_i 準時交必可以減少總扣分。

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i						
P_i						

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i				4		
P_i				8		

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i			4	4		
P_i			7	8		

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i		2	4	4		
P_i		6	7	8		

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i	3	2	4	4		
P_i	5	6	7	8		

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i	3	2	4	4	5	
P_i	5	6	7	8	3	

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

工作排程

D_i	4	4	2	3	5	2
P_i	8	7	6	5	3	2

S_i	1	2	3	4	5	6
D_i	3	2	4	4	5	2
P_i	5	6	7	8	3	2

實際步驟：

1. 依照 P_i 大小排序
2. 依照 P_i 大小把 H_i 放到死線 D_i 前
   A. 有衝突就往前放
   B. 放滿了代表只能遲交，往最後放

只需要扣 2 分！

Practice

Mission

Try LeetCode #621. Task Scheduler

Given a characters array tasks, representing the tasks a CPU needs to do, where each letter represents a different task. Tasks could be done in any order. Each task is done in one unit of time. For each unit of time, the CPU could complete either one task or just be idle.

However, there is a non-negative integer n that represents the cooldown period between two same tasks (the same letter in the array), that is that there must be at least n units of time between any two same tasks.

Return the least number of units of times that the CPU will take to finish all the given tasks.

Ref：https://leetcode.com/problems/task-scheduler/

Task Scheduler

只有一個工作：A，出現次數：5，且 n = 4

$$\boldsymbol{Total}$$
$$= (\mathbf{4}+\mathbf{1})(\mathbf{5}\text{-}\mathbf{1})+\mathbf{1}$$
$$= (\mathbf{n}+\mathbf{1})(\mathbf{t}-\mathbf{1})+\mathbf{1}$$

出現次數最多為 t 次

中間可以塞：$\boldsymbol{t}\times(\boldsymbol{n}-\mathbf{1})$

總長為：$(\mathbf{n}+\mathbf{1})(\boldsymbol{t}-\mathbf{1})+\mathbf{1}$

Task Scheduler

兩個工作：AB，出現次數：5，且 n = 4

$$Total$$
$$= (4 + 1)(5\text{-}1) + 2$$
$$= (n + 1)(t - 1) + 2$$

出現次數最多為 t 次
有 counts 個工作出現 t 次
總長為：$(n + 1)(t - 1) + counts$

Task Scheduler

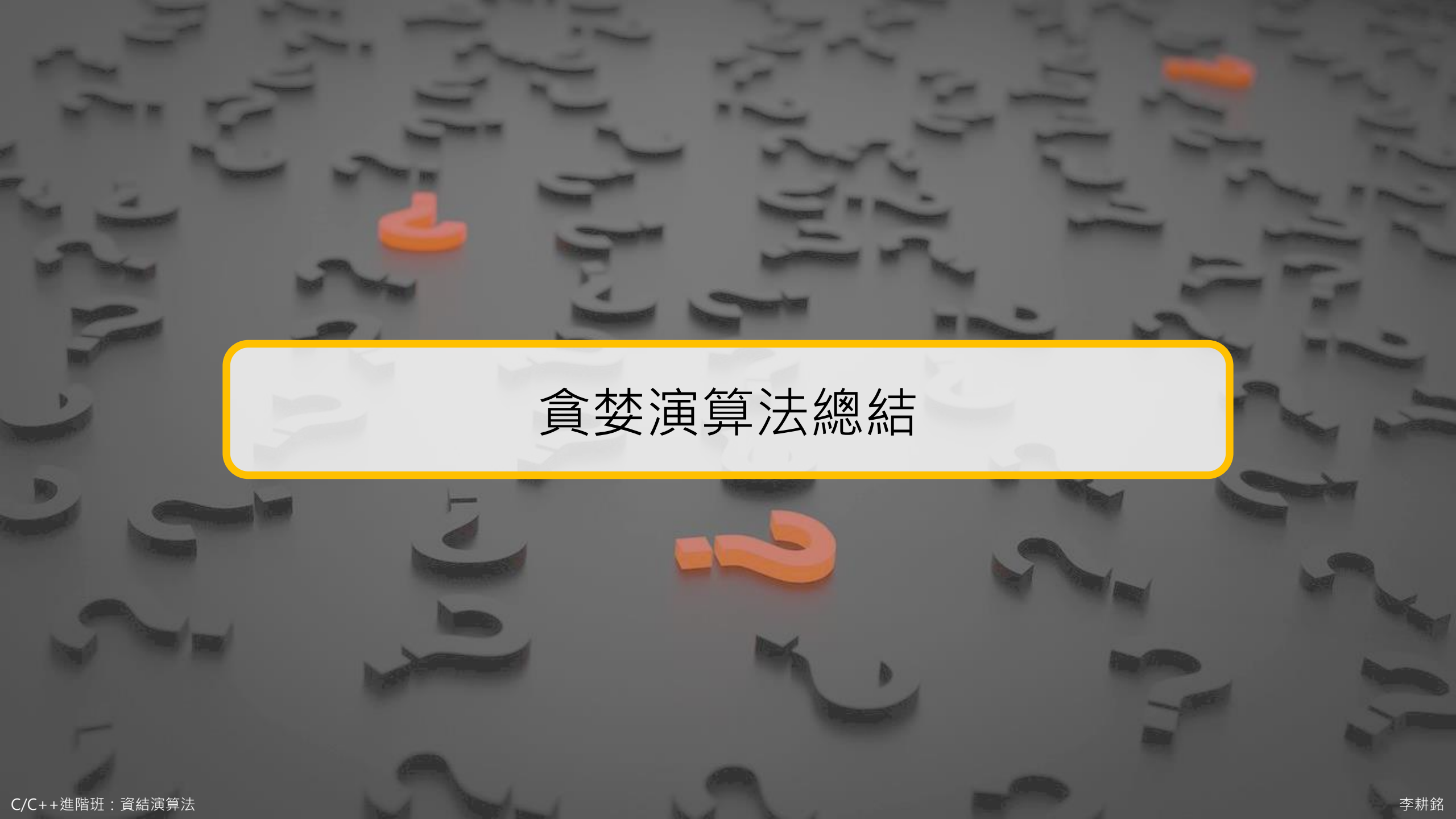
6份工作：ABCDEF，出現次數：5，且 n = 4

$$Total = 6 \times 5$$

總長為：工作的數目

出現次數最多為 t 次
有 counts 個工作出現 t 次
總長為：$(n + 1)(t - 1) + counts$

$$Total = max(taskAmount, (n + 1)(t - 1) + counts)$$



貪婪演算法總結

貪婪演算法簡介

貪婪演算法

- 解決最佳化問題時，通常很有效率且簡單
  - 按照順序一步步找出目前最佳解就可以了
- 適用貪婪演算法的狀況
  - 子問題的最佳解一定在母問題的最佳解
- 貪婪演算法並不能解決所有最佳化問題
  - 問題間的選擇有相關性
  - 每一步選擇會影響到下一步

實戰練習

Practice

Mission

Try LeetCode #1217. Minimum Cost to Move Chips to The Same Position

We have n chips, where the position of the i^{th} chip is position[i].

We need to move all the chips to the same position. In one step, we can change the position of the i^{th} chip from position[i] to:

- position[i] + 2 or position[i] - 2 with cost = 0.
- position[i] + 1 or position[i] - 1 with cost = 1.

Return the minimum cost needed to move all the chips to the same position.

Ref：https://leetcode.com/problems/minimum-cost-to-move-chips-to-the-same-position/

Practice

Mission

Try LeetCode #1221. Split a String in Balanced Strings

Balanced strings are those that have an equal quantity of 'L' and 'R' characters. Given a balanced string s, split it in the maximum amount of balanced strings. Return the maximum amount of split balanced strings.

Ref：https://leetcode.com/problems/split-a-string-in-balanced-strings/

Practice

Mission

Try LeetCode #1710. Maximum Units on a Truck

You are assigned to put some amount of boxes onto one truck. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]:

- numberOfBoxesi is the number of boxes of type i.
- numberOfUnitsPerBoxi is the number of units in each box of the type i.

You are also given an integer truckSize, which is the maximum number of boxes that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize.

Return the maximum total number of units that can be put on the truck.

Ref：https://leetcode.com/problems/maximum-units-on-a-truck/

Practice

Mission

Try LeetCode #1094. Car Pooling

You are driving a vehicle that has capacity empty seats initially available for passengers. The vehicle only drives east (ie. it cannot turn around and drive west.)

Given a list of trips, trip[i] = [num_passengers, start_location, end_location] contains information about the i-th trip: the number of passengers that must be picked up, and the locations to pick them up and drop them off. The locations are given as the number of kilometers due east from your vehicle's initial location.

Return true if and only if it is possible to pick up and drop off all passengers for all the given trips.

Ref：https://leetcode.com/problems/car-pooling/

Car Pooling

	Location									
1	5	5	5	5	5					
2						3	3	3		
3								4	4	4
4					2	2	2			
5		1	1	1						

Car Pooling

	Location									
1	+5				-5					
2						+3		-3		
3								+4		-4
4					+2		-2			
5		+1		-1						