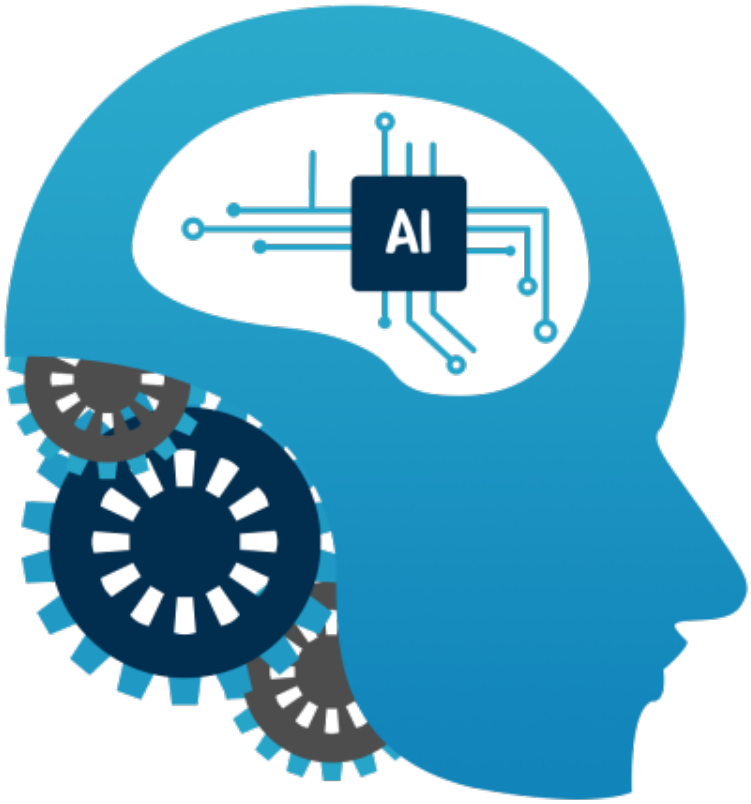
自主學習

初級演算法筆記(By Python)

學校：苗栗高中
作者：彭彥傑



動機

經過約半年的努力，我的APCS實作終於達到了3級(2023.11)，這讓我又驚又喜。而且，我意識到我對於程式設計的興趣與熱情也逐漸增長，也發現這也許是我的專長，因此，我想透過APCS組升學，以更深入地探索和擴展我的興趣為主要動機。

我希望能夠掌握更多演算法，如前綴和(prefix—sum)、二分搜尋(Binary—Search)、貪婪演算法(Greedy)、動態規劃(Dynamic programming)、最長遞增子序列(LIS)、最長共同子序列(LCS)等相關知識，這不僅是為了提升自己在學術上的表現，更是為了滿足我對於程式設計這個領域的好奇心和熱情。雖然這充滿挑戰，但我對於未來的學習和成長充滿信心，我相信只要我堅持努力，就一定能夠達成我的目標。

題目程式碼：https://jeremy960110.github.io/my-website/index.html

練習的題數以及最終選取的題數

1. 前綴和以及差分共5題選取3題
2. 二分搜共15題選取7題
3. 貪婪演算法共10題選取5題
4. 動態規劃共20題選選取8題

目錄

- 動態規劃(Dynamic Programming)

- 心得與反思

前綴和(Prefix-sum)

前綴和是指將一個數組中某個範圍內的元素相加的結果，並將這個結果存儲在另一個數組中，這樣如果需要一個區間範圍的總和，就不需要在慢慢的連加了，只需要進行一次的運算。

i	0	1	2	3	4	5	6
data	2	4	1	9	7	3	5
Presum	2	6	7	16	23	26	31

data[1]~data[5]的總和

一般解法：

```
Sum = 0
for i in range(1,6):
    Sum+= data[i]
print(Sum)
```

前綴和解法

```
Presum = [0]*len(data)
pre = 0
for i in range(len(data)):
    pre+=data[i]
    Presum[i] = pre
#前置作業
print(Presum[5]-Presum[1-1])
```

實戰1—幸運數字

(APCS 2021.9.part3)

題目要求

1. 從數字區間裡面找出最小數字
2. 把數字區間切成左右兩個子區間
3. 找出區間和較大的子區間,兩者相同則選右區間
4. 回到第一步。重複直到剩下一個數字

想法

先把轉成前綴和的形式，而為了之後能快速找到最小值，需要把data的index和value,照value由大到小排序。從尾巴看,就是目前最小值,但不一定在想要的區間(L-R)內，所以只要一直pop,即可找到當前(L-R)範圍內的最小值，再用前綴和求出左邊界跟右邊界的大小，即可找到下一個區間。

差分(difference)

差分就是跟前一項的差，可以用來計算全區間值，而且拿差分去做前綴會發現結果正好就是原來的陣列。

i	0	1	2	3	4	5	6
data[i]	0	2	2	5	5	1	0
diff[i]	0	2	0	3	0	-4	-1
Prediff[I]	0	2	2	5	5	1	0

*頭尾補零以方便計算

– diff[i]=data[i]-data[i-1]

– data[i]=diff[i]+data[i-1]

- data[i]=diff[i]+(diff[i-1]+data[i-2])
- data[i]=diff[i]+diff[i-1]+diff[i-2]...+diff[1]+data[0]
- data[i]=diff[i]+diff[i-1]+diff[i-2]...+diff[1]+diff[0]

實戰1—線段覆蓋長度

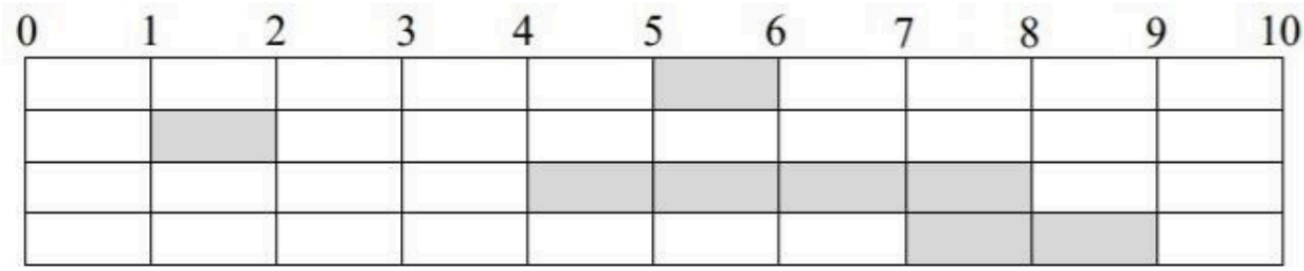
(APCS 2016.3.part3)

題目要求

1. 給定一維座標上一些線段,求這些線段所覆蓋的長度

想法

先照題目的輸入建立一個Map[0~最遠距離]，再把題目給的每一組頭跟尾地方在Map上加一跟減一(頭的地方加一，尾端減一)，接著用差分的方式去做(data[i]=diff[i]+data[i-1])，只要是大於一的地方就是有被覆蓋的，因此求出覆蓋長度。



i	0	1	2	3	4	5	6	7	8	9	10
Map	0	1	-1	0	1	1	-1	1	-1	-1	0
have	0	1	0	0	1	2	1	2	1	0	0

實戰2—生產線

(APCS 2021.11.part3)

題目要求

有n台機器排成一直線，每一個機器都有一個數值t[i]，代表該台機器要產出一單位的資料需要t[i]單位的時間，接下來有m個工作要完成，每一個工作都需要位置在[L[i],R[i]]的機器各生產出w[i]單位資料，現在你可以調換 n 台機器的順序，目標是使得這 m 個工作做完的總時間要最小。

想法

第一步先做差分，只是尾端要再多加長一單位，先把每個位置的生產總單位計算出來，然後某個位置的最大生產單位(line[i])跟最短時間的機器(machine[i])相乘加上第二大的跟第二小的相乘，以此類推，直到所有機器都被算過。

二分搜尋（BinarySearch ）

顧名思義，就是在搜尋時把資料分成兩半(左半邊跟右半邊)，一直切割，直到找到符合條件的資料，但是要怎麼知道左邊還是右邊呢？資料一定要可以排序，例如：[1,2,3,4,5,6,...]、[a,b,c,d,e,...]，這樣才可以確定目標所在的範圍。

二分搜做法

– 先把資料排序好

– 把搜尋的範圍,每次縮小一半

– 排序很費效能,但只要排序一次,之後搜尋的效率都會大幅增加

• 設target=38

data=[2, 11, 24, 35, 38, 39, 44, 47, 51, 55, 72, 75, 82, 85, 91,96]

[2, 11, 24, 35, 38, 39, 44, 47, 51, 55, 72, 75, 82, 85, 91, 96]

[2, 11, 24, 35, 38, 39, 44, 47, 51, 55, 72, 75, 82, 85, 91, 96]

[2, 11, 24, 35, 38, 39, 44, 47,51, 55, 72, 75, 82, 85, 91, 96]

[2, 11, 24, 35, 38, 39, 44, 47,51, 55, 72, 75, 82, 85, 91, 96]

[2, 11, 24, 35, 38, 39, 44, 47,51, 55, 72, 75, 82, 85, 91, 96]

心得

二分搜索是相對簡單的演算法，大家普遍都能夠寫出來。但是即使看似簡單，我在解題時往往容易出現各種Bug。一開始，我也以為這個演算法很容易實作，但在實際解題時，卻發現題目遠比我想像的更複雜。

在學習過程中，我深刻體會到了編測資的重要性。只要**編寫足夠測試資料，就能夠有效提高一次AC的機率**。透過不斷地嘗試各種題目，我逐漸提高了自己對於二分搜索算法的熟練程度，並**能夠在一小時內寫完二分搜的題目。**

總的來說，雖然二分搜可能會帶來一些挑戰和困難，但通過持續的學習和實踐，我相信我能夠克服更困難的問題，並提升我自己的解題能力。

二分搜尋函式庫(Python)------Bisect

```
import bisect
bisect.bisect_left(list,value)
bisect.bisect_right(list,value)
bisect.insort(list,value)
```

＊其實bisect函式不只這些，但是考試裡這些功能就很夠用了

- bisect_left跟bisect_right都會回傳索引，但是不一樣的是bisect_left回傳在list裡大於等於value的最小索引，而bisect_right則回傳list裡小於等於value的最大索引值。

- 至於insort就非常簡單了，就是把value依照順序插入list裡

注意事項

在python3.6.7裡，Bisect不能用在二維陣列，就算是使用key函式也不行，如果有二分搜的題目需要用二維陣列去做，就需要使用一個1D-list和一個2D-list去實作，詳情請翻閱P13、P17。

實戰1—切割費用

(APCS 2021.1.part3)

題目要求

有一根長度為L的棍子，你會把這個棍子切割n次。假設一開始棍子左端放在數線上0的位置，棍子的右端放在數線上L的位置，每次的切割會給定一個介於0到L的數字表示要切割的位置，你要把穿過這個位置的棍子切成兩段，而所需的花費就等於所切割的棍子的長度。

想法

可以定義一個p去紀錄頭跟尾還有每次切割的點，這樣每次切割後的花費就是p[x+1]-p[x-1]，然後用bisect快速找出放入p的適當位置，最後把這些花費計算出來就是答案了。

實戰2—美麗的彩帶

(APCS 2019.6.part3)

題目要求

一條彩帶的美麗度定義為，彩帶所擁有的長度為 m 且當中顏色皆不同的子區間數量。

舉例來說，彩帶為1 2 3 5 4 5 4，m為3。從左向右看，每3個可為一個子區間，分別為 1 2 3、2 3 5、3 5 4、5 4 5、4 5 4。當中有3個子區間內的顏色皆不同。因此美麗度為3。

想法

1.window是紀錄子區間的2D結構如果子區間為[1,3,2,3]，則window=[[1,1], [3,2], [2,1]]而前面有說bisect無法接受2D-array，所以要新增名為win0([x[0] for x in window])的list去紀錄目前子區間的集合。

2.每多一個點，就在win0找看看有沒有，沒有就insort(window跟win0都要)，有就window[idx][1]+=1

3.每少一個點，就在window找這個點，並且window[idx][1]-=1，如果變成0，就pop(window跟win0都要)。

4.len(window)如果等於m，美麗度+1

實戰3—基地台

(APCS 2017.3.part4)

題目要求

某電信公司負責其中 N 個服務點，這 N個服務點位在一條筆直的大道上，其中距離範圍為0~(N-1)。每個基地台的服務範圍必須都一樣，當基地台架設後，與此基地台距離不超過 R (稱為基地台的半徑)的服務點都可以使用無線網路服務，也就是說每一個基地台可以服務的範圍是 D=2R(稱為基地台的直徑)。現在電信公司想要計算，如果要架設K個基地台，那麼基地台的最小直徑是多少才能使每個服務點都可以得到服務。

基地台架設的地點不一定要在服務點上，最佳的架設地點也不唯一，但本題只需要求最小直徑即可。

想法

剛開始看到題目根本沒有想到會用到二分搜尋，但在紙上模擬後，就發現了規律。

1. 假如D=n，那n+1也一定可以覆蓋全部的基地台，而n-1則不一定。

2. 承1，發現了二分搜的規律，這樣就需要去寫一個函示去實現，如果m無法覆蓋全部，L = m+1，否則，R = m(不能m-1，因為有可能此時的m剛好是最小覆蓋半徑)。

實戰4—牆上海報

(APCS 2022.1.part4)

題目要求

有高度不一的柵欄，和多張寬度不同的海報(有規定順序)，最高能貼多高。

想法

跟基地台那題類似，都具有單調性，也就是說，若某高度x可以貼,那x以下也一定可以貼，若某高度x不能貼,那x以上也一定不能貼，如果找出x可以貼,x+1不能貼,那x一定就是最佳解。

而難的地方是如何知道當前高度是否可以貼，首先需要把p(廣告長度)reverse，以便之後p2做pop時可以提高效率，接著在函示past裡宣告一個p2 = p.copy()，之後pop就不會引響到原來的p了，然後用迴圈跟一些if-else，就可以求出是否貼得上。

實戰5—支點切割

(APCS 2018.2.part3)

題目要求

輸入一個大小為N的一維整數陣列p[i]，要找其中一個所謂的最佳切點將陣列切成左右兩塊，然後針對左右兩個子陣列繼續切割，切割的終止條件有兩個：子陣列範圍小於3或切到給定的層級K就不再切割。

舉例來說，有數列[2,4,1,3,14]，K=3。左右端點2、14不可為切點。依序嘗試4,1,3為切點。

以4為切點時，左區間有值2與切點的距離為1，因此左區間乘積(值x距離)總和為2x1=2；右區間乘積(值x距離)總和為1x1+3x2+14x3=49; 差異(差的絕對值)為47

以1為切點時，左區間乘積總和為2x2+4x1=8; 右區間乘積總和為3x1+14x2=31; 差異23

以3為切點時，左區間乘積總和為2x3+4x2+1x1=15; 右區間乘積總和為14x1=14; 差異1

想法

不難發現，這題就是標準的遞迴分治題型，而且合力矩值(左力矩-右力矩)也是遞增，這樣就可以知道這題是考二分搜。

而之後在實作的過程中發現，用前綴和好像也可以，只是要做兩次前綴和。

實戰6—雷射測試

(APCS 2022.6.part3)

題目要求

一個雷射光從 (0,0) 向右邊發射，平面上有很多個鏡子，問雷射光會反射幾次，保證輸入沒有無限循環的情形。鏡子用三個數字表示 ，代表座標在(x,y,t)，t=0代表/角度的鏡子，t=1代表\角度的鏡子。輸出雷射光共反射幾次。

想法

因為雷射光是直線前進的，所以每個x座標相同的點都以y排序，每個y座標相同的點都以x排序。這樣就可以很快以二分搜找到某點的相鄰點(上下左右)，而mirX[i]是記錄所有x值為i的鏡子資料，而mirXy[i]是記錄所有x=i的鏡子的y值，且有照大小排序；mirY跟mirYx則相反。

實戰7—圓環出口

(APCS 2020.7.part3)

題目要求

有n個房間排成一個環，編號分別是0到n−1。

房間之間僅有單向的路徑，編號 i 的房間僅可以走到下一個號碼的房間，且最後一個房間(n-1號)可以通到第一個房間(0號)。離開每個房間可獲得對應點數。你的任務是走過必要的房間，收集到大於等於指定的點數，並算出最後停留的房間編號。

對於每次的任務，一開始所在房間即可獲得點數，每進入又離開一個房間又可獲得該房點數，當累計點數大於等於指定的點數時，則完成此一任務並停在下一個房間(尚未獲得此房間點數)。

假設n,m=4,3 ；p=1, 3, 5, 7; q=4, 2, 2

1. 一開始位於0號房(1點)，走到1號房(3點)時即收集到4點，完成第一個任務，停留在2號房
2. 從2號房(5點)開始即超過所需2點，完成第二個任務，停留在3號房
3. 從3號房(7點)開始即超過所需2點，完成第三個任務，停留在0號房
4. 所有任務完成，輸出為0

想法

前面做了那麼多題目，這題一看就知道是考前綴和啦！而要快速找到目標，就需要再配合bisect，找到第一個剛好大於目標的presum

貪婪演算法(Greedy)

1. 是一種**符合人類直覺**的抽象演算法

2. **只考慮目前狀態最佳的選擇**，且之前所選擇的解答不會影響後面所選擇的解答，不斷的選擇局部的最佳解，全部選取結束後就獲得整體的最佳解。**若可以舉出反例，就證明此題不能用Greedy。**

學習心得

在學習Greedy算法的過程中，我遇到了蠻多挫折，這讓我感受到了自己在邏輯和數學方面的一些不足。很多題目一開始對我來說根本難以解決，讓我相當沮喪。不過，我發現自己在**排程問題**上相對較擅長，這點給了我一些信心。

除了排程問題外，其他類型的題目要麼需要我花費超過一個小時的時間去思考，不然就根本無法解決，只好去參考別人的程式，而且說不定還要花一兩天才能理解背後的邏輯。這樣的情況讓我深刻地意識到了自己在Greedy方面的學習還有很大的提升空間。

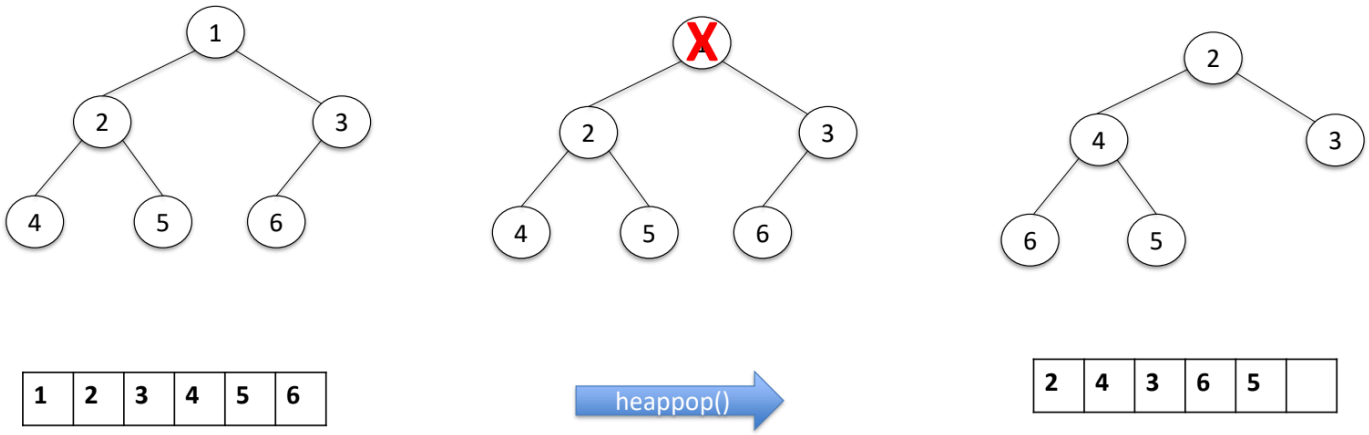
但儘管遇到了挫折，這段學習過程還是讓我受益良多。它不僅提升了我的邏輯思維能力，還加深了我對數學問題的理解。我相信，通過我的努力和學習，我一定能夠克服這些困難，更加熟練地使用Greedy解決各種題目。

優先佇列函式庫(Python)----heapq

```
import heapq
heapq.heapify(list)
heapq.heappop(heap)
heapq.heappush(heap,value)
```

＊APCS裡會用到heapq的題目少之又少，但一旦遇到，它就會變的很重要

- 是一種Queue,增刪元素要用push跟pop
- 能快速的找到跟刪除最小值
  - data[0]就是最小值
  - heapq.heappop(data)就是把最小值丟出來並刪除。然後重整結構
- push元素進去時,會自動維持一個樹狀結構,確保下次要處理最小值很快



實戰1—Add All

題目要求

我們的任務是把一些數加起來，但不是要得到加總後的值，而是執行任務所需要的最低成本(cost)。

兩個數相加的**成本**定義是，2個數的總和。譬如1+2的成本為3。當多個數要相加時，必須要兩兩相加來計算成本。

舉例來說，1、2、3這三個數相加，可以用1+2+3、1+3+2、2+3+1的不同相加順序來估算成本如下

- **1+2+3:**1+2=3, cost1=3 => 3+3=6, cost2=6。total cost=9
- **1+3+2:**1+3=4, cost1=4 => 4+2=6, cost2=6。total cost=10
- **2+3+1:**2+3=5, cost1=5 => 5+1=6, cost2=6。total cost=11

最低成本為9

想法

既然要最低成本，又同一組測資要做的加法次數都是一樣的，所以要盡可能讓每次的成本越少越好，因次要讓目前data的最小兩個值相加，再放回data裡，但這就需要排序，所以就需要用bisect來維持data的順序了。

實戰2－書籍印刷裝訂

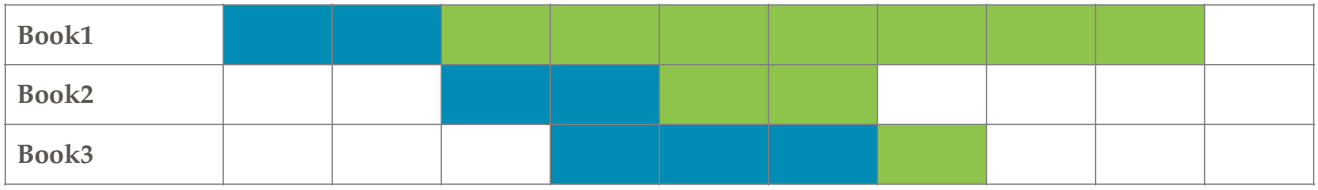
題目要求

現有n本書需印刷裝訂。每一本書必須先印刷再裝訂。工廠有n 台裝訂機但只有一台印刷機。印刷機同時只能印一本書，必需印完一本書後才能印下一本書。現給定每一本書的印刷時間和裝訂時間，請計算所有書最快要多久才能印刷裝訂完畢。

想法

因為印刷是稀缺資源，而裝訂是無限的資源，**無論先印誰，總印刷時間都不變**，而裝訂時間長的則是越早印越好，長的先完成，才不會浪費，總不可能把裝訂時間長的放在最後面吧！

而我們能不能把**所有印刷時間加起來再加最小裝訂時間呢？**答案是不行，因為有可能某個裝訂時間非常大，大到超過所有印刷時間加最小裝訂時間(如下圖)。



藍色：印刷時間;綠色：裝訂時間

實戰3—死線排程

題目要求

給 N 個作業，每個作業分別有deadline與profit，分別以d_i和p_i表示，一天表示，一天可選擇一項作業來完成。每項作業必須在dealine之前(含)完成，否則就得放棄，已完成的作業則可獲得其profit。譬如$d_1=2,p_1=5$，表示第1項作業若在第2天前完成，可獲得5點利益。因為時間有限，有可能必須放棄部份作業。請幫忙安排時程，達到最大利益(profit)。

舉個例子：

- d=2 3 4 4 5 5
- p=5 6 4 2 3 1
- 可依序每日完成第1,2,3,4,5項作業，第6項作業放棄，可獲得(5+6+4+2+3)=20點最大利益

想法

如果可完成的作業是一個list，會發現每個元素的**deadline一定大於索引(也可以說是deadline大於等於到deadline時的子區間長度)**，而照常理來說，deadline一定是由小到大，而profit不能確定，所以一開始先依deadline大小排序data，再新增一個heapq(在Python標準函示庫，正式名稱是優先佇列)，而如果要放入作業的deadline大於該heapq的長度，則直接把該作業的profit放入此heapq;而如果等於heapq的長度，代表如果要寫這個作業必須犧牲掉其中一個作業，這時候如果heapq裡的最小profit小於該作業的profit，則做抽換，以達最大利益。

實戰4—物品堆疊

(APCS 2017.10.part4)

題目要求

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。我們有兩個可能的擺放順序(由上而下)：

- (1,2)，也就是物品 1 放在上方，2 在下方。那麼，取用 1 的時候不需要能量，而每次取用 2 的能量消耗是 w(1)=1，因為 2 需取用 f(2)=4 次，所以消耗能量數為w(1)*f(2)=4。
- (2,1)，也就是物品 2 放在 1 的上方。那麼，取用 2 的時候不需要能量，而每次取用1的能量消耗是w(2)=2，因為1需取用f(1)=3次，所以消耗能量數=w(2)*f(1)=6。

在所有可能的兩種擺放順序中，最少的能量是4。再舉一例，若有三物品而 w(1)=3、w(2)=4、w(3)=5、f(1)=1、f(2)=2、f(3)=3。假設由上而下以(3,2,1)的順序，此時能量計算方式如下：取用物品 3 不需要能量，取用物品 2 消耗 w(3)*f(2)=10，取用物品 1 消耗

(w(3)+w(2))*f(1)=9，總計能量為 19。如果以(1,2,3)的順序，則消耗能量為3*2+(3+4)*3=27。事實上，我們一共有 3!=6 種可能的擺放順序，其中順序(3,2,1)可以得到最小消耗能量 19。

想法

由題目敘述可知如果要用暴力法，最糟情況有可能n!種情況都要算完，所以先不考慮，而這題應該是我用第~~六~~感數感解出來的。

直觀上來說，如果要減少能量消耗，必須讓越重的物品在下面，而兩個物品比較是依W跟F相乘的結果—>W_A*F_B 和W_B*F_A，在數學上又可以得出 W_A/F_A 和 W_B/F_B ，所以就可以用sort(key=lambda)做排序，W/F值較大者會在data右方(下面)，較小者則在data左方(上面)，這樣就可以由左到右去累加重量並且計算總能量消耗了。

實戰5—過橋問題

題目要求

有n個人想要在晚上過橋，橋上每次最多只能容許2個人行走。由於全部只有一支手電筒，所以每次2個人拿著手電筒過橋後，必須有一人再把手電筒拿回來，這樣後面的人才能繼續過橋。

每個人走路的速度不同，過橋所需的時間也因此不同。而每次過橋的那2個人，其花費的時間以較慢的那個人計算。你的任務是寫一個程式，安排這n個人過橋，並使得總共花費的時間最少。

想法

第一次看到這個問題我想了一個下午，結果還是無法解出，於是只能在網路上求助了，**以下是我的理解。**

先把人依速度做排序，如果只有兩人以下就是選最大的，而三個人(a,b,c)呢？那就是c(c跟a一起)+a(a跑回來)+b(b跟a一起)。但四個以上就有點複雜了，這裡舉四個人的例子(a,b,y,z)可以把a,b當成運輸手電筒的人，y,z為最慢的人，如果四人以上，先把y,z運走，以產生新的y,z，直到小於三個人，換句話說，每次把最慢的兩個人運送到對面。

方法1：az過，a回來接y過去，a再回來。共耗z+a+y+a

方法2：ab過，a回來，yz過，b回來。共耗b+a+z+b

每一輪所需時間就是min(z+a+y+a,b+a+z+b)，並且把最後兩項pop，直到len(data)<4。

動態規劃DynamicProgramming

1. 一種分治法

2. 把大問題拆解成小問題,小問題的解答放在表裡面

3. **建表**是DP主要的特色

—>**找出狀態轉移方程式**,建出完整的表

學習心得

一開始，我對DP的學習過程比起Greedy來的更加艱難。因為DP的核心在於找到適當的狀態轉移方程式，而如果這一步無法達成，則解決問題會變得非常困難。這段時間或許是我學習編程以來遇到最大的挫折，每個練習題目都讓我感到無助。然而，我很慶幸隨著時間的推移、透過頻繁不斷地練習，我開始找到動態規劃的感覺，並有了一定的進步。

實戰1—最小路徑

題目要求

現在有一張地圖，出發點在地圖的左上角，在移動時，所以只能往右或下走，終點在右下角，凡是走過某一個格子，都會消耗體力，地圖上的數字為消耗值，左上角跟右下角必為 0，走過的點上的數字加總即為總消耗。

以不同路徑走到終點會有不同的總消耗，請輸出最小值。

Sample 1:

0 7 8 9

1 5 1 1

2 4 10 **0**

在此例的最小值為8(1+5+1+1)

想法

因為只能往右跟下走，代表除了左邊界跟上邊界以外的地方都是由左方或上方來的。

左邊界：M[i][0] = M[i-1][0] + data[i][0]

上邊界：M[0][j] = M[0][j-1] + data[0][j]

其他 ： M[i][j] = min(M[i-1][j], M[i][j-1])+data[i][j]

實戰2—採蘑菇攻略問題

題目要求

有一個益智遊戲用鍵盤方向鍵控制主角,想辦法採到好蘑菇以得高分。但要小心不是每個蘑菇都會加分,採到壞的蘑菇會扣分!每個蘑菇以一個整數來表示正或負的分數。正數代表好蘑菇及其得分,負數代表壞蘑菇及其扣分。例如: 有9個蘑菇由左至右排列如下:

-2, 1, -3, 4, -1, 2, 1, -5, 4

採蘑菇的方式限制為由左至右連續採,但不一定要從第一個採,也不需要採到最後一個,也可以通通不採。以上例而言,最佳的採法得分為6分,出現在採第4到第7這四個蘑菇中(4-1+2+1=6),其他的採法(要連續)的得分都比6小。 請寫一程式計算出最高得分。

想法

根據題目敘述，最小值必定為0，所以先宣告一個名為M的list，負責紀錄到該點所有可能中最大值，既然是最大值，就一定是用max()函式了，而這樣也可以確保答案一定可以大於等於0。接下來就可以很直觀到寫出

1. M[i-1]>=0的情況，M[i] = M[i-1]+data[i]

2. M[i-1]<0的情況，M[i] = 0(負數只會對後面的數字造成負擔)

狀態轉移方程式：M[i] = max(M[i],M[i-1]+data[i])

實戰3—超級馬拉松賽

題目要求

一個超級馬拉松比賽將開始。在遊戲中，選手每天需要跑不同的路徑。假設遊戲全部有 n 條路徑; 每個路徑得分可以是不同的。如果一名選手不能在規定時間內完成一條路徑，他該路徑得到零分;如果玩家完成了一條路徑在一個規定的時間，他得到該路徑設定的得分;如果玩家完成了一條路徑，用較短的時間，他可以得兩倍分數。

小愛想參加這個比賽，她如果在一條路徑上按正常速度來跑，就只能拿到原始分數，如果他加速跑，就能拿到兩倍分數，不過她就會需要在加速跑完後的下一條路徑上休息而速度變慢得到0分，請寫一個程式幫助小愛計算哪些路徑應該加速得到兩倍分數而能獲得最高的總得分。

想法

- 如果這次要兩倍，上次必須0或者1
- 如果這次是一倍，上次必須1或者0
- 如果這次是零倍，上次必須2或者1

利用以上關係，就可以求出狀態轉移方程式

—status[i][0] = max(status[i-1][1],status[i-1][2])

—status[i][1] = max(status[i-1][0],status[i-1][1])+data[i]

—status[I][2] = max(status[i-1][0],status[i-1][1])+data[i]*2

實戰4—What Goes Up

題目要求

寫一個程式從一連串的整數序列中選出最長的嚴格遞增子序列（strictlyincreasing subsequence）。例如：在 1, 3, 2, 2, 4, 0 中最長的嚴格遞增子序列為 1,3, 4 或者 1, 2, 4。

首先輸出一列最長的嚴格遞增子序列的長度是多少。然後一列僅含有一個減號（dash character, '-'）。然後接下來為這個最長的嚴格遞增子序列的內容，每個整數一列。

如果有不止一個最長的嚴格遞增子序列，請輸出在輸入中最後出現的。例如在 1, 3, 2, 2, 4, 0 中，應該輸出 1, 2, 4 而不是 1, 3, 4。

想法

這是標準的LIS題型，而且是嚴格遞增，我們都會做，但如何找出最後出現的呢？答案一定是增加長度的前面一個，如下圖：

i	0	1	2	3	4	5	6	7	8	9
data	1	4	2	5	3	7	4	9	2	5
LISlen	1	2	2	3	3	4	4	5	2	5

從LISlen裡最大的LISlen[9]往後找，比它(5)小一個長度的第一個是LISlen[6]，再接著往後找，是LISlen[4]，以此類推，直到找到值為1的。最後總共會找到data[9],data[6],data[4],data[2],data[0]，也就是[5,4,3,2,1]，最後再反轉就好了。

實戰5—飛黃騰達

(APCS 2021.1.part4)

題目要求

有隻特別的飛黃一開始在座標 (0, 0) 的位置，而且你知道它只會往右上方移動，也就是移動的時只可以走到 x 座標跟 y 座標都不比原本小的位置。

現在座標平面的第一象限上有n個位置有果實，給定這n個果實的座標，你想要知道這隻特別的飛黃最多可以吃到幾個果實（它必須移動到果實所在的座標才可以吃到果實）。

輸出一個數字表示最多可以吃到多少果實。

想法

題目有說飛黃只會往右方、上方、右上方移動，所以先把data按照X跟Y的大小做排序，然後把Y的部分取出，代替原本的data，雖然裡面的Y有可能值是一樣的，但所對應到的X不可能相同，因為題目有說：「x 座標跟 y 座標都不比原本小」，同理，換成取出X也是一樣的道理，而這樣就可以知道Y非嚴格遞增，所以新的data是NLDS(非嚴格遞增子序列)，詳情請看程式碼。

實戰6—基因序列密碼問題

題目要求

基因序列是由四個鹼基A、C、G、T組合而成，例如ACGT和CTA 的最長共同子序列是CT。請注意subsequence和substring不同，subsequence的字母不需要在原來字串裡鄰近出現，只需要保持字母的順序。你的任務就是要寫一個程式找出兩個基因列序的最長共同子序列。若兩個基因列序的不只有一個最長共同子序列，選擇以第一個序列靠左字母先出現的解。舉例來說，ACG跟CAG比對，有AG跟CG皆為最長共同子序列，選AG為解。因為第一個序列為(A)CG，(A)G具有最靠左的字母A。

想法

如果可能性只有一種，那就是LCS的標準做法。

—若data1[i-1]==data2[j-1]，status[i][j]=status[i-1][j-1]+1
—若data1[i-1] != data2[j-1]，status[i][j]=max(status[i-1][j],status[i][j-1])
—>status[len(data1)][len(data2)]即為LCS長度。

status	j	0	1	2	3	4	
i		ε	**B**	**D**	**C**	**A**	data2
0	ε	0	0	0	0	0	
1	**A**	0	0	0	0	1	
2	**B**	0	1	1	1	1	
3	**C**	0	1	1	2	2	
	data1						

但是題目要求是「以第一個序列靠左字母先出現的解」，所以要先紀錄status[i][j]的值是從哪個方向來的，例如左=1，上=2，左上=3，這時就要建立一個名為path的list去紀錄，之後再由path[-1][-1]依照方向往回推。

status

i		j	0	1	2	3
			data2	**B**	**A**	**C**
0	data1		0	0	0	0
1	**A**		0	0	1	1
2	**B**		0	1	1	1
3	**C**		0	1	1	2

path

i		j	0	1	2	3
			data2	**B**	**A**	**C**
0	data1		0	0	0	0
1	**A**		0	2	**3**	1
2	**B**		0	**3**	2	2
3	**C**		0	2	2	**3**

建表之後就會發現，如果path[i][j]=3，則data1[i-1]=data2[j-1]，意思是有可能是答案要求的LCS，但是要怎麼知道答案要求的是哪一個？在建path時，如果path[i][j]左跟上的值都一樣，那就優先選上，因為這樣會用接近data1的左邊，才能滿足「以第一個序列靠左字母先出現的解」的需求。

實戰7—開心的金明

題目要求

金明今天很開心，家里購置的新房有一間他自己專用的房間。媽媽昨天對他說：“你的房間需要購買哪些物品，怎麼布置，你說了算，只要總價不超過N元錢就行”。今天金明就開始做預算，但是他想買的東西太多了，肯定會超過媽媽限定的N元。於是，他把每件物品規定了一個重要度，分為5等：用整數1~5表示，第5等最重要。他希望在不超過N元（可以等於N元）的前提下，**使每件物品的價格與重要度的乘積總和最大。**

請你幫助金明設計一個滿足要求的購物單。

想法

這題就是背包問題的變形，以 i 軸為每項物品，j 軸為價格，令二維DB的表格為staus，status[j]代表在考慮第 i 項物品後價格為 j 時的開心度(價格與重要度的乘積)總和，然候迭代每個物品。

—價格(p)大於 j 代表無法改變，繼承上一次的值

status[i][j] = status[i-1][j]

—價格(p)小於 j 代表可以從stats[i][j-p]加上來去做比較

status[i][j] = max(status[i-1][j],status[i-1][j-p]+p*imp)

最後答案為max(status[-1])

另外還有一維DP的解法，那就是以 j 軸為價格，status[j]代表價格為j時的開心度(價格與重要度的乘積)總和，而因為status[j] = max(stasts[j],status[j-p]+p*imp)是由前面為基礎做加法，所以for j in range()時必須反向迭代以防止影響前面的值，換句話說就是避免重複拿到同一個東西。

實戰8—換零錢

題目要求

貝茜在某國商店工作。這個國家很奇怪，銅板的種類跟幣值經常在更動，請你算出貝茜最少能用幾個硬幣找錢給顧客。

譬如有5種不同的幣值1、3、4、5、6，要找20元給顧客，最少可以用4個硬幣。包括(5+5+5+5)或(6+4+6+4)。

你需要用N（1<=N<=10）種不同的硬幣數提供C（1<=C<=1000）元給顧客。所有的C值都可以用此N種硬幣湊出來。

想法

這種題目如果幣值是可以連續整除的話，可以用Greedy，否則，就只能用DP，而這題我用一維DP的解，但是跟上一題「開心的金明」不一樣，一維DP的 j 軸不需要反向迭代，原因是同一種硬幣可以重複取用。令status[j]為價格為j時的最小硬幣數量。

—幣值(coin)大於所需價格(j)，不需做更動

pass

—幣值小於等於所需價格，以status[j-coin]為基礎加一個硬幣做比較

status[j] = min(status[j],status[j-coin]+1)

最後答案為status[C]

心得與反思

自主學習基礎演算法的過程的確是一段充滿挑戰的旅程。在解題的過程中，挫折感時常會出現，因為每一道問題都可能需要花費相當長的時間來思考和解決，而且常常還需要花費大量的時間來除錯。

然而，就是這些挑戰和困難塑造了我對演算法的理解和能力。通過不斷地思考、測試和糾正錯誤，我漸漸地發現自己在解決問題上變得更加熟練和自信。每當我克服了一個困難，找到了一個有效的解決方案時，成就感便油然而生，激勵著我繼續前進。

更重要的是，這段學習過程讓我意識到持之以恆的重要性。雖然遇到困難可能會感到沮喪，但只要堅持不懈，最終總能找到解題的方法。這種堅持不懈的精神不僅在學習演算法中發揮作用，也同樣適用於生活中的各個方面。

總的來說，雖然學習基礎演算法的過程充滿了挑戰和困難，但正是這些挑戰讓我成長和進步。我深知這段學習過程只是漫漫學海中的一小步，但我將以更大的熱情和毅力繼續探索演算法的世界，不斷提升自己的技能和能力。

題目程式碼：https://jeremy960110.github.io/my-website/index.html