基礎動態規劃(I)

2021/07/10

by林品安

初步介紹

動態規劃的題型、核心觀念

動態規劃?
看這名字根本看不出它要拿來幹嘛

動態規劃?
看這名字根本看不出它要拿來幹嘛

也因此很多人學完動態規劃，還是不知道甚麼題目要用動態規劃?

動態規劃?
看這名字根本看不出它要拿來幹嘛

也因此很多人學完動態規劃，還是不知道甚麼題目要用動態規劃

動態規劃主要是拿來解:

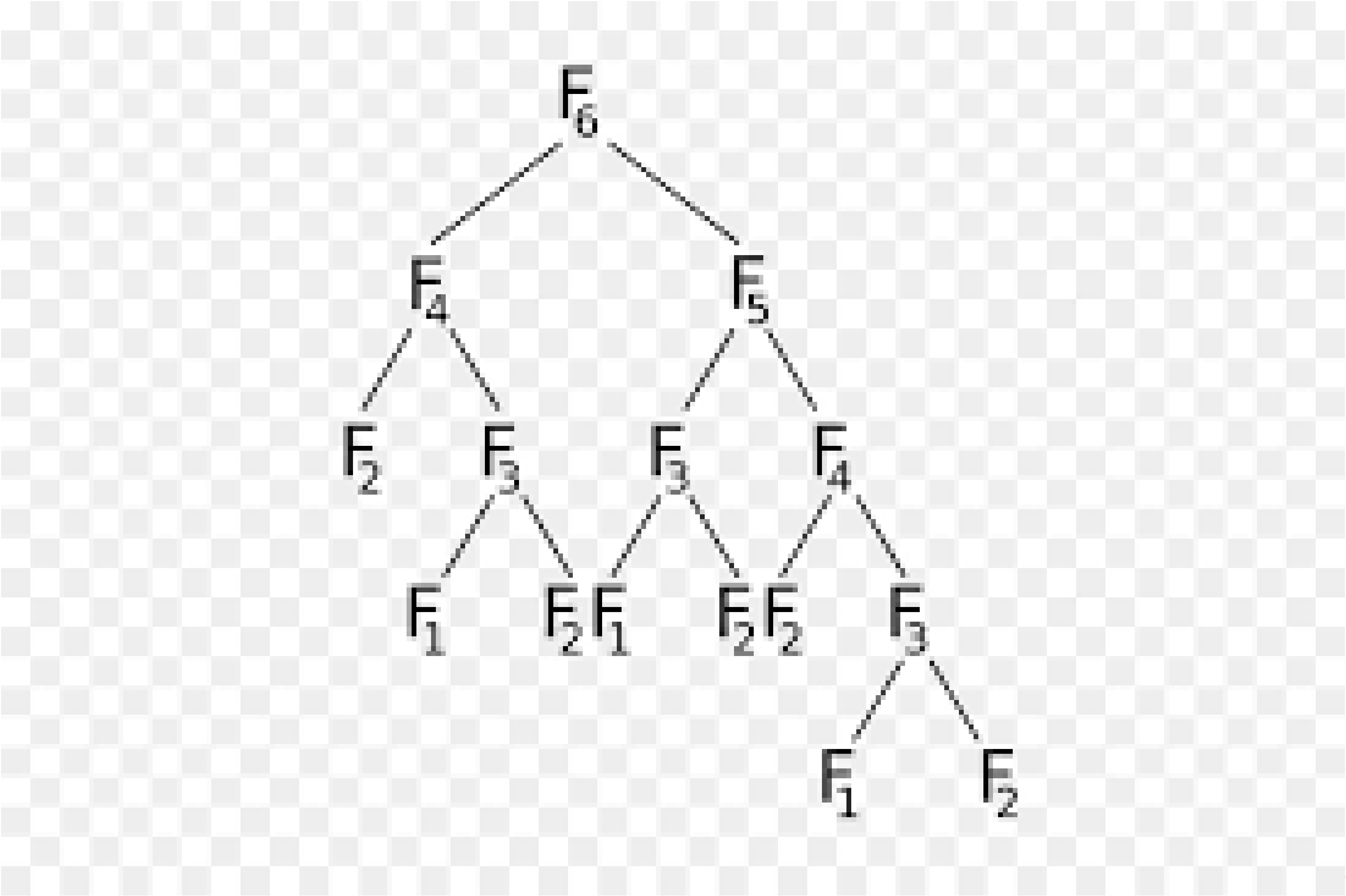
1. 記憶化搜索
2. 最佳化問題
3. 記數問題

甚麼是記憶化搜索?

給你一個遞迴式 ： $F(x) = F(x-1) + F(x-2)$

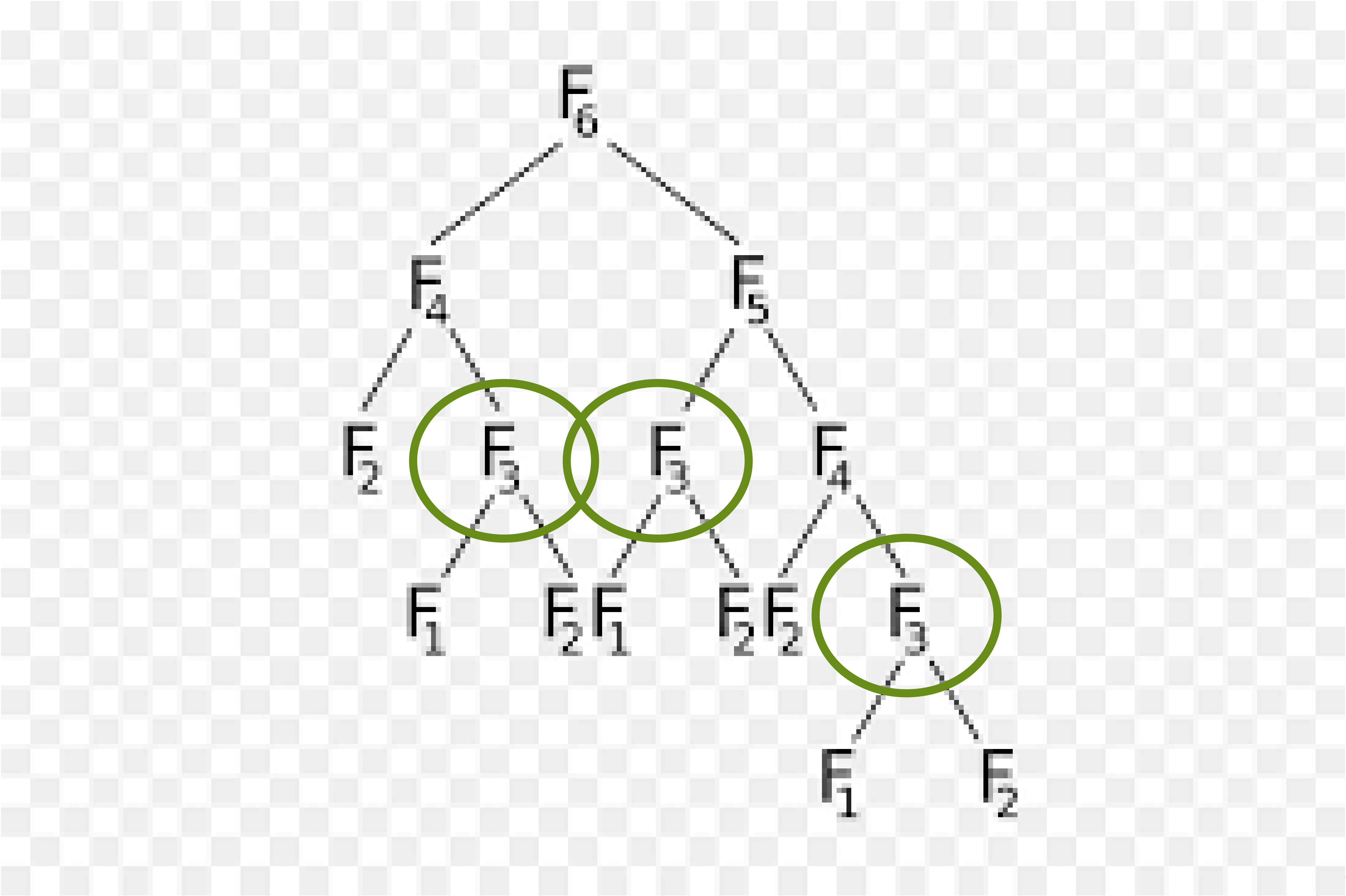
甚麼是記憶化搜索?

給你一個遞迴式 ： $F(x) = F(x-1) + F(x-2)$



甚麼是記憶化搜索?

給你一個遞迴式 ： $F(x) = F(x-1) + F(x-2)$



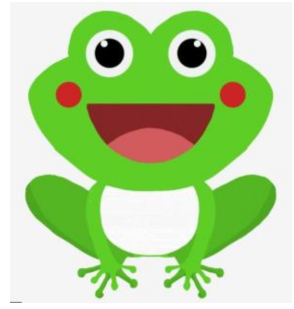
甚麼是最佳化問題?

有一隻青蛙在最左邊的藍色寶珠，
每次他可以往右跳一顆石頭或是往右跳兩顆石頭，
請問他所經過的數字總和最大是多少?

甚麼是最佳化問題?

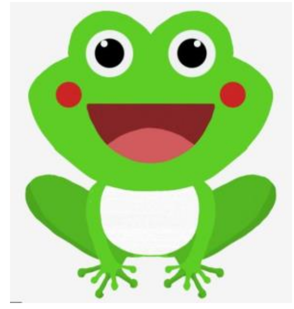
給你一個背包，你只背得動 W 公斤的東西，
現在有 N 個商品，每個商品有重量和價值，
問你最多可以背走多少價值的商品?

有一隻青蛙在最左邊的藍色寶珠，
每次他可以往右跳一或二顆石頭，
請問他所經過的數字總和**最大**是多少?



5　3　7　2　4

每次都跳一格就好啦…？



5 3 7 2 4

最佳化問題一定要用動態規劃來解決嗎?

答案是否定的，很多最佳化問題用Greedy更容易解決，通常時間複雜度也更好。

這也是導致動態規劃讓初學者棘手的地方…

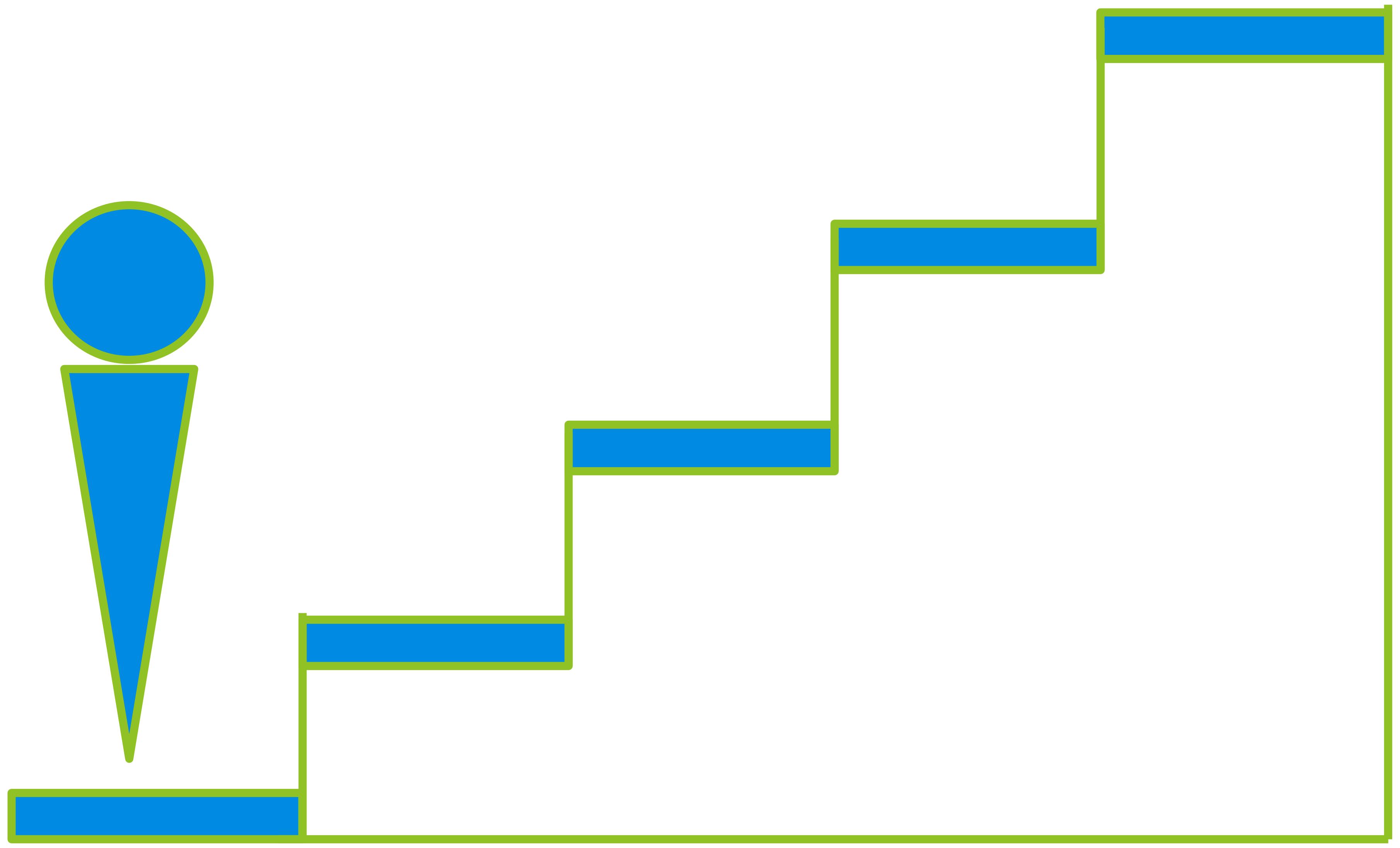
這題我到底要用 greedy 還是 dp ?

這個問題是沒有答案的，甚至很多題目 greedy 跟 dp 都可以解。

多做題目培養感覺，才是最好的解決之道!

甚麼是記數問題?

有一個 N 階的樓梯，每次你可以往上走一或二層，
請問有幾種不同的走法?



核心觀念

狀態、轉移、基底

核心觀念

動態規劃的核心觀念離不開 “記憶已經算過的東西”，

因為該算的還是要算，只是我們 “不做重複的計算”，

所以我們又可以說動態規劃是 “優雅的暴力”。

核心觀念

以費式數列的遞迴當作例子：

$$F(x) = F(x-1) + F(x-2)$$

```
void F(int x){
    if(x == 1 || x == 2) return 1;
    return F(x-1) + F(x-2);
}
```

核心觀念

既然會重複呼叫到 $F(x)$ 很多次，那就把它紀錄起來。

以此來避免重複的計算(也就是遞迴)。

```
int dp[100];
void F(int x){
    if(x == 1 || x == 2) return 1;
    if(dp[x] != 0) return dp[x];
    dp[x] = F(x-1) + F(x-2);
    return dp[x];
}
```

核心觀念

做動態規劃的三大步驟:

1. 列出狀態
2. 導出轉移
3. 打好基底

核心觀念

做動態規劃的三大步驟:

1. 列出狀態 - 想好你的DP表格要記錄甚麼

核心觀念

做動態規劃的三大步驟:

1. 列出狀態 – 想好你的DP表格要記錄甚麼
2. 導出轉移 – 每一格DP表格的值怎麼算

核心觀念

做動態規劃的三大步驟:

. 列出狀態 – 想好你的DP表格要記錄甚麼
2. 導出轉移 – 每一格DP表格的值怎麼算
3. 打好基底 – 初始狀態有哪些?

線性DP

線性的動態規劃

線性DP

線性 DP，是屬於 DP 裡面比較簡單的類型，往往比較直觀好思考。

線性DP–試題演練

70. Climbing Stairs

Easy 7175 223 Add to List Share

You are climbing a staircase. It takes `n` steps to reach the top.

Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

https://leetcode.com/problems/climbing-stairs/

線性DP−試題演練

狀態：

$dp[i]$ 代表走到第 i 階的方法數

轉移：

因為只能從前一階、前兩階走過來，所以方法數是兩者兩個相加

$dp[i] = dp[i-1] + dp[i-2]$

基底：

共一種方法走到起點：$dp[0] = 1$

線性DP−前綴和

我們定義前綴和 S_i 是整數序列 A_i 的前i個元素的總和。

我們可以利用前綴和在 $O(1)$ 的時間得出區間和，即兩個前綴和的差值:

$$sum(l,r) = S_r - S_{l-1}$$

線性DP-前綴和

用人話來說:

a[ ]=	5	2	3	1	2	2
S[ ]=	5	7	10	11	13	15

所以 $S[i] = S[i-1] + a[i]$，前面的總和加上現在這格的值。

線性DP-前綴和

用人話來說:

a[ ]=	5	2	3	1	2	2
S[ ]=	5	7	10	11	13	15

當我們要求出 $a_l \sim a_r$ 的總和時只要扣掉不需要的部分就好囉!

線性DP−前綴和

用人話來說:

			l		r	
$A=$	5	2	3	1	2	2
$S_r=$	5	7	10	11	13	15
$S_{l-1}=$	5	7	10	11	13	15

黃色的部分是 $a_1 \sim a_r$ 的總和，但我們不需要 $a_1 \sim a_{l-1}$ 所以要扣掉喔~

線性DP–前綴和

https://zerojudge.tw/ShowProblem?problemid=a693

輸入說明
多組測資以 EOF 結束 每組測資開始有兩個正整數 n,m (n,m <= 100000) 接下來一行有 n 個不超過一千的正整數依序代表每個食物的飽足度 接下來 m 行每行有兩個數字 l,r (1 <= l <= r <= n) 代表你想要吃掉第 l 個到第 r 個食物

輸出說明
對每組測資輸出 m 行，代表總飽足度

範例輸入 #1
3 3 1 2 3 1 3 1 2 2 3

範例輸出 #1
6 3 5

線性DP−前綴和

狀態：

設 $dp[i]$ 代表前 i 個元素的和。

轉移：

因為前 i 個元素的和是前$i-1$個元素的和+現在的元素。

$$dp[i] = dp[i-1] + A[i]$$

基底：

前 0 個元素的和是0。

$$dp[0] = 0;$$

線性DP−前綴和

#時間複雜度

對於每個 i 求一次 DP 值…就$O(N)$阿w

線性DP−爬樓梯進階版

#問題

樓梯共有 N 階，每次你可以往上走最多 k 階、最少 1 階，請問有幾種方法?

#輸入

輸入的第一行有兩個數字 N, k $(1 \le N \le 100000, 1 \le k \le N)$，
分別代表樓梯共有幾階、阿東一次最多走幾階。

#輸出

輸出方法總數%1000000007，代表答案。

線性DP−爬樓梯進階版

狀態：

設 $dp[i]$ 代表走到第 i 階樓梯共有幾種走法。

轉移：

因為走到第 i 階，由他的前k層走過來，所以得到轉移式

$$dp[i] = dp[i-k] + dp[i-k+1] + \cdots + dp[i-1]$$

基底：

走到第 0 階的方法數是 1。

$$dp[0] = 1;$$

線性DP–爬樓梯進階版

轉移：

因為走到第 i 階，由他的前k層走過來，所以得到轉移式

$$dp[i] = dp[i-k] + dp[i-k+1] + \cdots + dp[i-1]$$

這好像是……？ 區間和？

線性DP–爬樓梯進階版

#時間複雜度分析

每一階的方法數我們需要用前 k 階的方法數來算，
而得到前 k 階的方法數的總和可以用前綴和將複雜度壓到$O(1)$，
故總體複雜度依舊是$O(N)$

線性DP–爬樓梯進階版

這種技巧稱之為 DP優化，
寫 DP 題目的時候一定要先想出正確的狀態、轉移，
即使複雜度不好也沒關西，或許我們可以想辦法優化它，
就算不能優化，至少也會有部分分！

背包問題DP

背包問題的兩種轉移

背包問題

https://atcoder.jp/contests/dp/tasks/dp_d

D - Knapsack 1 Editorial

Time Limit: 2 sec / Memory Limit: 1024 MB

Score : 100 points

Problem Statement

There are N items, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), Item i has a weight of w_i and a value of v_i.

Taro has decided to choose some of the N items and carry them home in a knapsack. The capacity of the knapsack is W, which means that the sum of the weights of items taken must be at most W.

Find the maximum possible sum of the values of items that Taro takes home.

Constraints

- All values in input are integers.
- $1 \leq N \leq 100$
- $1 \leq W \leq 10^5$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^9$

背包問題

Sample Input 1 Copy

```
3 8
3 30
4 50
5 60
```

Sample Output 1 Copy

```
90
```

Sample Input 3 Copy

```
6 15
6 5
5 6
6 4
6 6
3 5
7 2
```

Sample Output 3 Copy

```
17
```

背包問題

狀態：

設 $dp[i][j]$ 代表僅拿前 i 種商品，並且總共裝了 j 公斤的物品的最大價值。

轉移：

某商品的重量價值分別是 W_i、V_i ，

1. 如果我們拿完此商品後總重量是 j ，那原本的重量是 $j-W_i$，且還沒拿第i件商品，

故取$dp[i-1][j-W_i]+V_i$的值代表拿當前商品的最大價值。

2. 如果我們不拿此商品但總重量一樣是 j，那當然是用前 $i-1$ 件商品裝的，

故取 $dp[i-1][j]$ 的值代表不拿當前商品的最大值。

那用前 i 件商品去湊出重量 j 的最大價值就是上述兩個的值較大的那個了！

背包問題

基底:

我們會發現，前 0 件商品怎麼湊重量都是0，

所以 $dp[0][0] = 0$

而前0件商品也湊不出任何重量，所以我們初始成0，代表沒價值，

所以 $dp[0][j] = 0, j \leq X$

背包問題

時間複雜度：

對於$dp[i][j]$每一格都要算一次，所以複雜度$O(nX)$

背包問題

https://atcoder.jp/contests/dp/tasks/dp_e

E - Knapsack 2 Editorial

Time Limit: 2 sec / Memory Limit: 1024 MB

Score : 100 points

Problem Statement

There are N items, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), Item i has a weight of w_i and a value of v_i.

Taro has decided to choose some of the N items and carry them home in a knapsack. The capacity of the knapsack is W, which means that the sum of the weights of items taken must be at most W.

Find the maximum possible sum of the values of items that Taro takes home.

Constraints

- All values in input are integers.
- $1 \leq N \leq 100$
- $1 \leq W \leq 10^9$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^3$

背包問題

Sample Input 1 Copy

```
3 8
3 30
4 50
5 60
```

Copy

Sample Output 1 Copy

```
90
```

Copy

Items 1 and 3 should be taken. Then, the sum of the weights is $3 + 5 = 8$, and the sum of the values is $30 + 60 = 90$.

Sample Input 2 Copy

```
1 1000000000
1000000000 10
```

Copy

Sample Output 2 Copy

```
10
```

Copy

背包問題

狀態：

設 $dp[i][j]$ 代表僅拿前 i 種商品，並且總共裝了 j 公斤的物品的最大價值。
總共做多會有 X 公斤的物品，所以要開的陣列是 $dp[n][10^9]$????

炸啦！ 所以我們得換個做法

背包問題

狀態:

設 $dp[i][j]$ 代表僅拿前 i 種商品，並且湊出總價值為 j ，最少需要幾公斤。
因為商品總價值最多 $n * \max(v_i)$ ，所以陣列開的下!

轉移:

1. 如果拿第 i 件商品，則總價值從 $j - v[i]$ 變成 j，
   故取 $dp[i-1][j-v[i]] + w[i]$ 的值代表拿第i件商品的最小重量
2. 如果不拿第 i 件商品，則總價值不便，從前i-1件商品的最大值轉移，
   故取 $dp[i-1][j]$ 的值代表不拿第i件商品的最小重量

背包問題

基底：

我們會發現，前 0 件商品怎麼湊都湊不出任何價值，
所以我們將 $dp[0][i]$ 初始化為無限大，表示湊不出來。

特別的是 $dp[0][0] = 0$，因為湊的出來(?

背包問題

時間複雜度：

對於$dp[i][j]$每一格都要算一次，所以複雜度$O(n * n * V_{max})$

經典問題

LCS、LIS

LCS(最長共同子序列)

- 子序列是在一個序列中，有序但不連續的部分
  - 例如 : $A[] = 1\,3\,5\,9\,5\,7$
  - 其中 357 就是 A 的子序列
- 共同子序列就是兩個序列 A、B 都有的子序列
  - 例如 : $A[] = 1\,3\,7\,4\,1\,5$, $B[] = 1\,7\,2\,4\,5$
  - 其中 17 是他們的共同子序列
  - 而 1745 是他們的最長共同子序列

LCS(最長共同子序列)

- 轉換到字串，假設:
  - $A = “abcdba”$
  - $B = “cbdaba”$
  - 則A、B的LCS為 $“bdba”$

LCS(最長共同子序列)

- 怎麼求 *LCS*?
- 一樣，先想狀態、再想轉移、然後打好基底

LCS(最長共同子序列)

- 狀態:

  令$dp[i][j]$ 為

  「A的前i個的字元組成的字串」跟「B的前j個字元組成的字串」的LCS

LCS(最長共同子序列)

- 狀態:

  令$dp[i][j]$ 為

  「A的前i個的字元組成的字串」跟「B的前j個字元組成的字串」的LCS

- 轉移:

  想想 $dp[i][j]$ 是怎麼來的?

  如果 $A[i]$ == $B[j]$ 的話，LCS要+1，那既然 i 去匹配 j 了，

  顯然的我們要從更短的兩個子序列來轉移，

  所以

$$\boldsymbol{if(A[i] == B[j])dp[i][j] = dp[i-1][j-1]+1;}$$

LCS(最長共同子序列)

- 狀態:

  令$dp[i][j]$ 為

  「A的前i個的字元組成的字串」跟「B的前j個字元組成的字串」的LCS

- 轉移:

  想想 $dp[i][j]$ 是怎麼來的?

  如果 $A[i]\,!=\,B[j]$ 的話，LCS要從比較大的那一邊轉移過來。

  $$\boldsymbol{if(A[i]\ !=\ B[j])\ dp[i][j]\ = max(dp[i-1][j], dp[i][j-1]);}$$

LCS(最長共同子序列)

用圖來說比較好懂

X	Y	0	a 1	s 2	w 3	v 4
	0	0	0	0	0	0
a	1	0	↖ **1**	← 1	← 1	← 1
r	2	0	↑ 1	↑ 1	↑ 1	↑ 1
s	3	0	↑ 1	↖ **2**	← 2	← 2
w	4	0	↑ 1	↑ 2	↖ **3**	← 3
q	5	0	↑ 1	↑ 2	↑ 3	← 3
v	6	0	↑ 1	↑ 2	↑ 3	↖ **4**

LCS(最長共同子序列)

- 基底?

我們會發現，既然還沒匹配那長度當然是0，所以$dp[0][0] = 0$。

再者，如果其中一邊還沒用到當然LCS長度也是0， $dp[i][0] = dp[0][i] = 0$。

LCS(最長共同子序列)

1143. Longest Common Subsequence

Medium 3461 40 Add to List Share

Given two strings `text1` and `text2`, return *the length of their longest **common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

https://leetcode.com/problems/longest-common-subsequence/

課後練習

- https://zerojudge.tw/ShowProblem?problemid=d212
- https://atcoder.jp/contests/dp/tasks/dp_a
- https://atcoder.jp/contests/dp/tasks/dp_b
- https://atcoder.jp/contests/dp/tasks/dp_e
- https://zerojudge.tw/ShowProblem?problemid=b184
- https://leetcode.com/problems/longest-palindromic-subsequence/