

C/C++ 進階班

資結演算法

動態規劃 (Dynamic Programming)

李耕銘

課程大綱

- 動態規劃簡介
 - 斐波那契數列
 - 找錢問題
 - 最大子數列問題
 - 活動選擇問題
- 動態規劃常見應用
 - 郵票問題
 - 切割問題
 - 背包問題
 - 矩陣鏈乘
 - 最長遞增子序列 (LIS)
 - 最長共同子序列 (LCS)
- 動態規劃總結
- 實戰練習

動態規劃簡介

動態規劃簡介

- 與分治法類似，動態規劃會把母問題切成許多子問題
 - 再把子問題的答案組回母問題的答案
- 跟分治法不同的是
 - 動態規劃中，每個子問題通常環環相扣
 - 因為會用到每個子問題的答案，因此需要加以記錄
- 動態規劃會把較小的子問題答案記錄下來
 - 用空間換取時間
 - 凡走過必留下痕跡

動態規劃簡介

```
int fibo(int n) {  
    int *data = (int*) malloc(sizeof(int)*n);  
    data[0] = 1;  
    data[1] = 1;  
    for(int i=2;i<n;i++){  
        data[i] = data[i-1] + data[i-2];  
    }  
    int result = data[n-1];  
    free(data);  
    return result;  
}
```

```
int fibo(int n) {  
    if(n <= 2)  
        return 1;  
    else  
        return fibo(n-1) + fibo(n-2);  
}
```

動態規劃

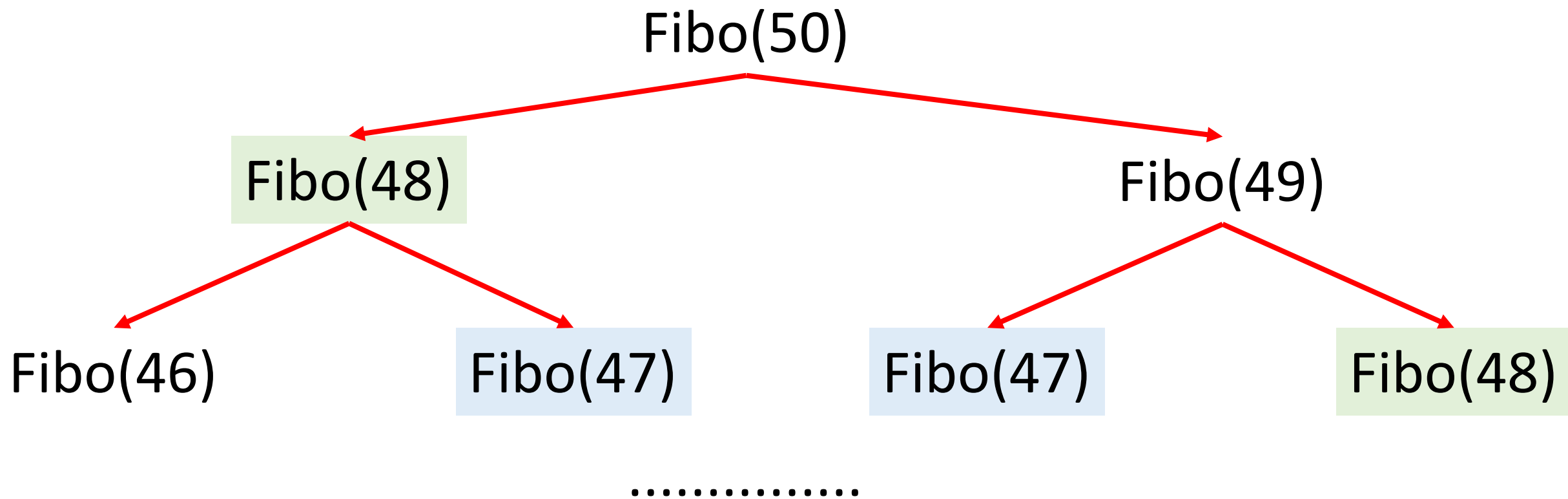
分治法

分治法

`fibonacci(1) = 1`

`fibonacci(2) = 1`

`fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)`

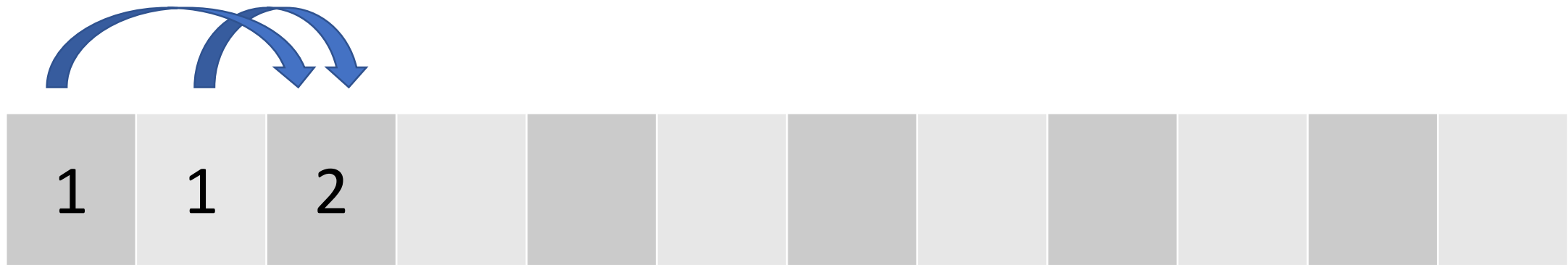


動態規劃簡介

1

1

動態規劃簡介



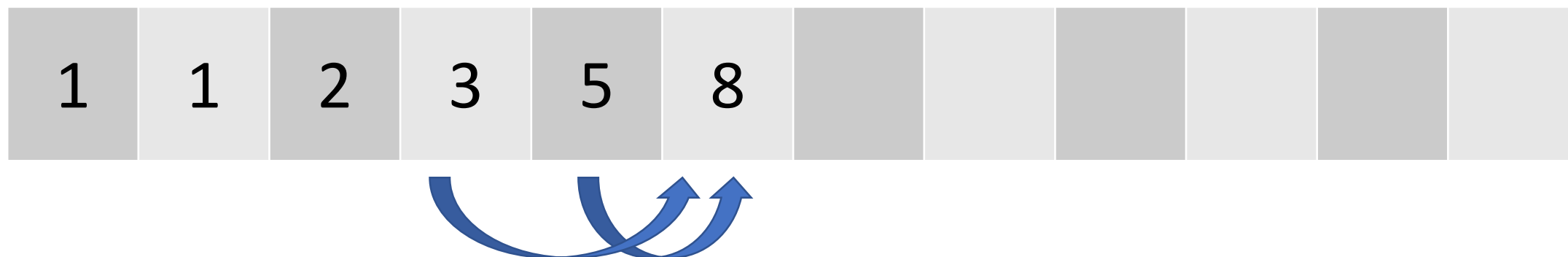
動態規劃簡介



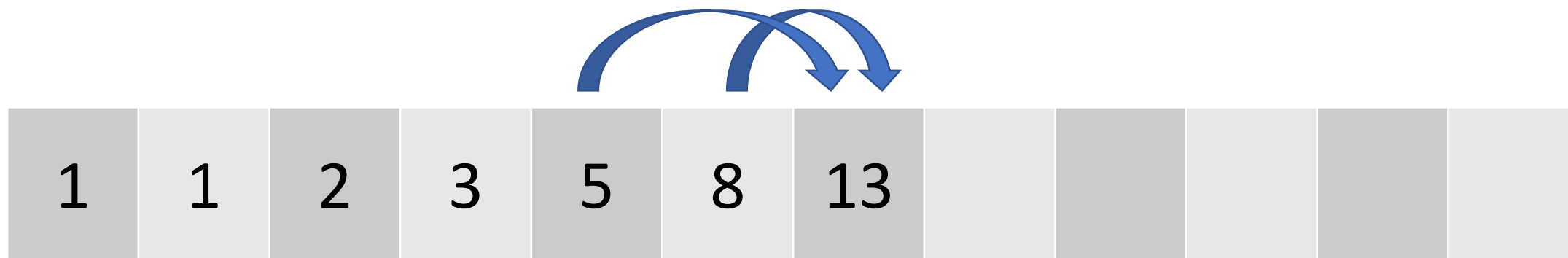
動態規劃簡介



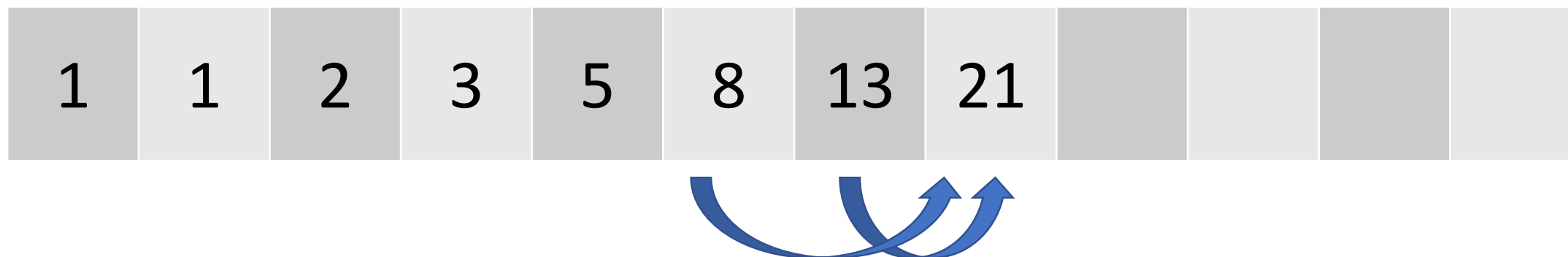
動態規劃簡介



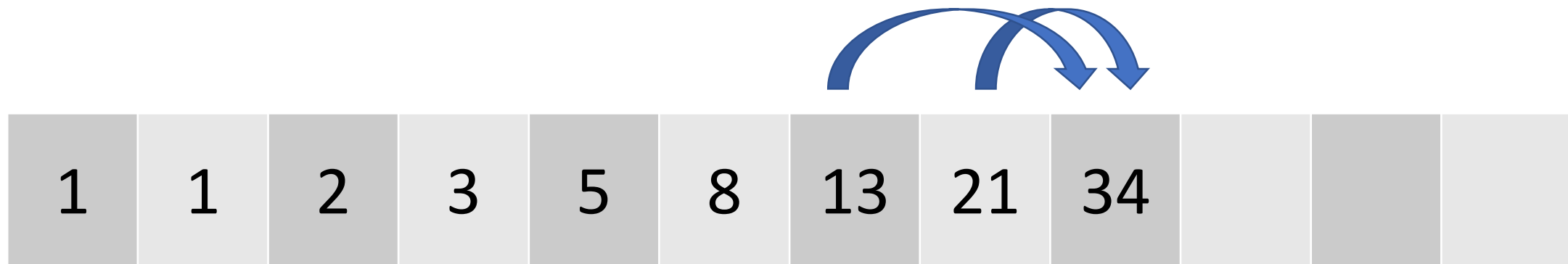
動態規劃簡介



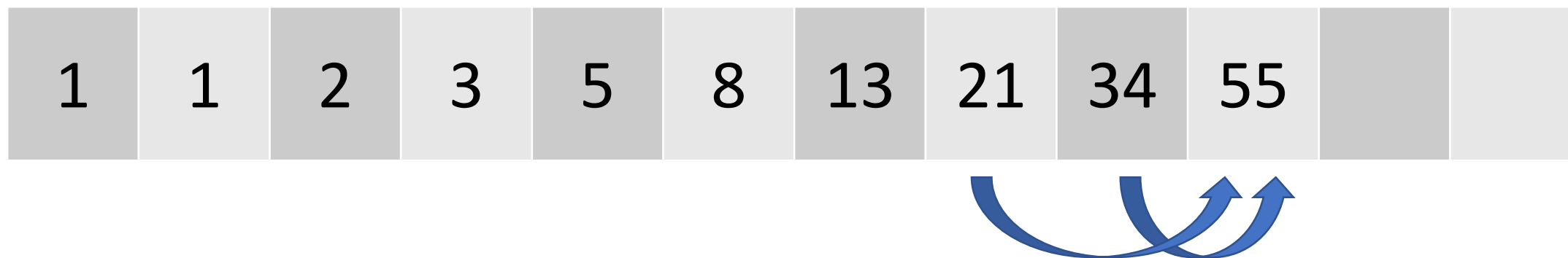
動態規劃簡介



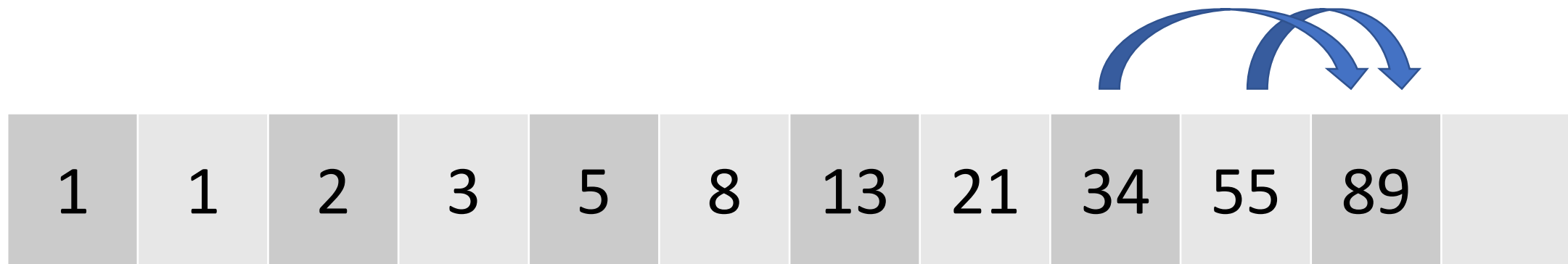
動態規劃簡介



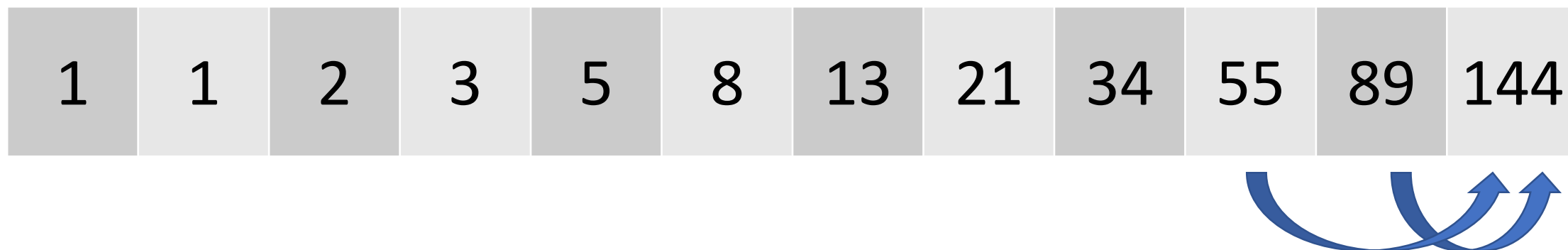
動態規劃簡介



動態規劃簡介



動態規劃簡介



Practice

Mission

已知有一函式 $f(n)$ 改良自斐波那契數列如下，
請找出 $f(30)$ 的值：

$$f(n) = \begin{cases} 1, & \text{if } n \leq 3 \\ f(n-1) + f(n-2) + f(n-3) \end{cases}$$

Practice

Mission

Try LeetCode #746. Min Cost Climbing Stairs

You are given an integer array `cost` where `cost[i]` is the cost of *i*th step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1. Return the minimum cost to reach the top of the floor.

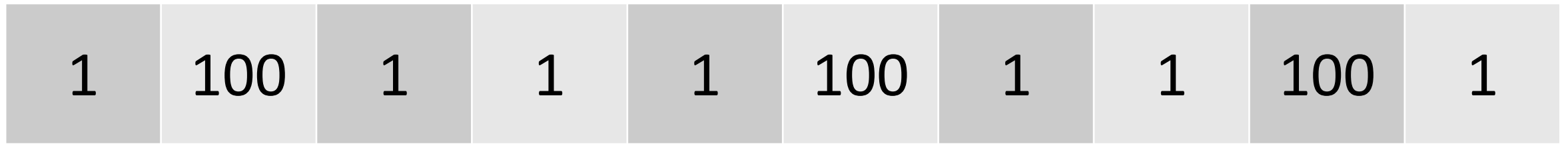
Ref : <https://leetcode.com/problems/min-cost-climbing-stairs/>

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100								
---	-----	--	--	--	--	--	--	--	--

Practice



$$\begin{aligned} &Step[2] \\ &= \min(Step[0] + cost[2], Step[1] + cost[2]) \\ &= \min(Step[0], Step[1]) + cost[2] \\ &= \min(1, 100) + 1 \\ &= 2 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

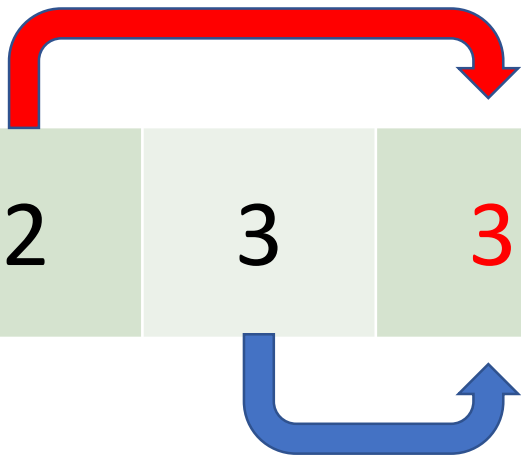


$$\begin{aligned} &Step[3] \\ &= \min(Step[1], Step[2]) + cost[3] \\ &= \min(100, 2) + 1 \\ &= 3 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100	2	3	3					
---	-----	---	---	---	--	--	--	--	--

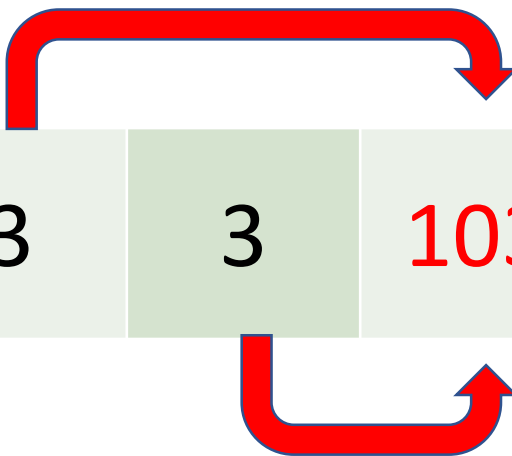


$$\begin{aligned} &Step[4] \\ &= \min(Step[2], Step[3]) + cost[4] \\ &= \min(2, 3) + 1 \\ &= 3 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100	2	3	3	103				
---	-----	---	---	---	-----	--	--	--	--

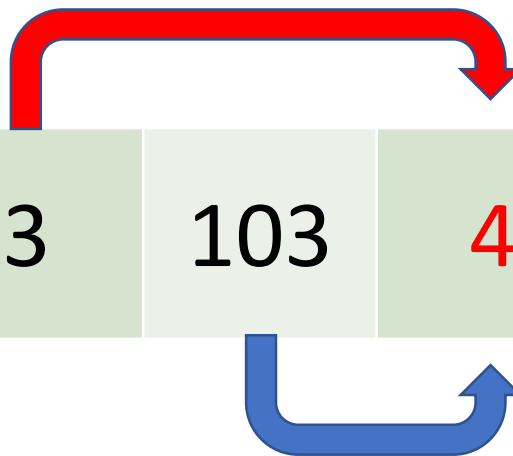


$$\begin{aligned} &Step[5] \\ &= \min(Step[3], Step[4]) + cost[5] \\ &= \min(3, 3) + 1000 \\ &= 103 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100	2	3	3	103	4			
---	-----	---	---	---	-----	---	--	--	--

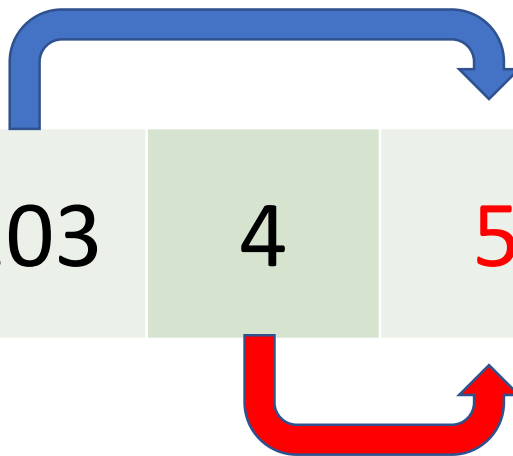


$$\begin{aligned} &Step[6] \\ &= \min(Step[4], Step[5]) + cost[6] \\ &= \min(3, 103) + 1 \\ &= 4 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100	2	3	3	103	4	5		
---	-----	---	---	---	-----	---	---	--	--

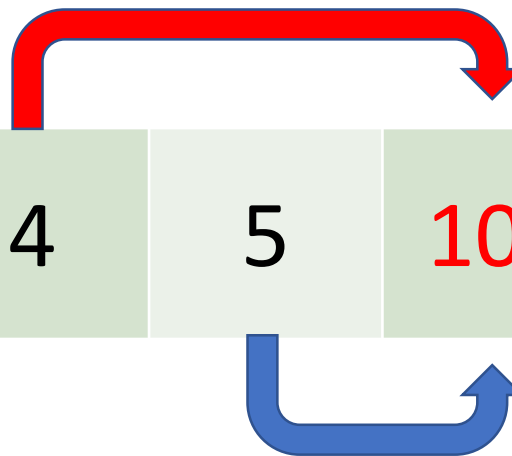


$$\begin{aligned} &Step[7] \\ &= \min(Step[5], Step[6]) + cost[7] \\ &= \min(103, 4) + 1 \\ &= 5 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100	2	3	3	103	4	5	104	
---	-----	---	---	---	-----	---	---	-----	--

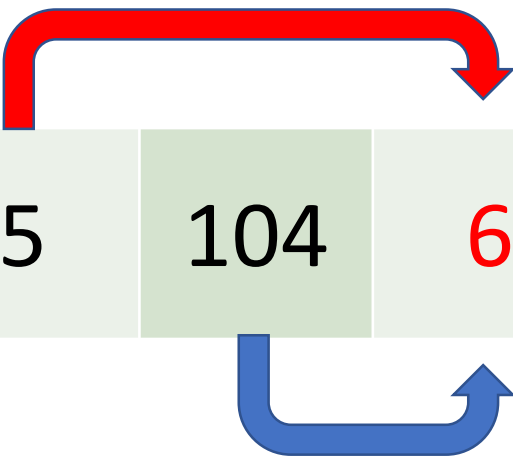


$$\begin{aligned} &Step[8] \\ &= \min(Step[6], Step[7]) + cost[8] \\ &= \min(4, 5) + 100 \\ &= 104 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1
---	-----	---	---	---	-----	---	---	-----	---

1	100	2	3	3	103	4	5	104	6
---	-----	---	---	---	-----	---	---	-----	---



$$\begin{aligned} &Step[9] \\ &= \min(Step[7], Step[8]) + cost[9] \\ &= \min(5, 104) + 1 \\ &= 6 \end{aligned}$$

Practice

1	100	1	1	1	100	1	1	100	1	0
---	-----	---	---	---	-----	---	---	-----	---	---



插入以處理邊界條件

1	100	2	3	3	103	4	5	104	6	
---	-----	---	---	---	-----	---	---	-----	---	--

動態規劃簡介

1. 最佳化子結構

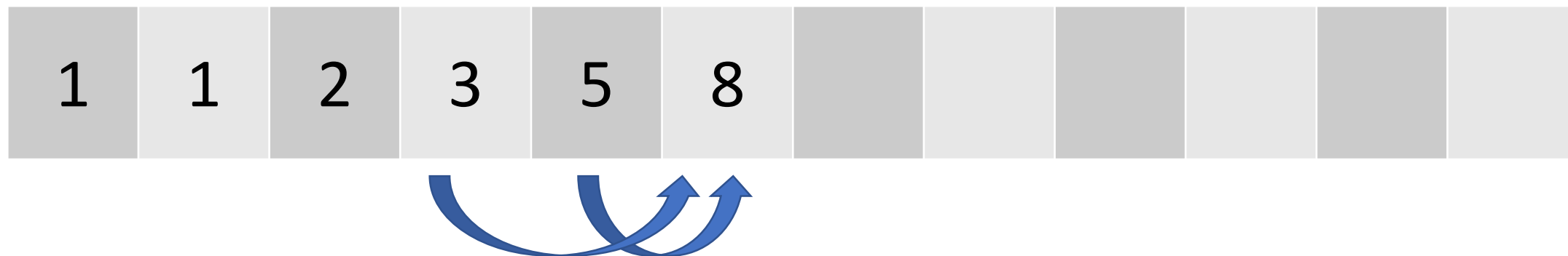
- 能把母問題切割成數個子問題
- 能夠推算出子問題的答案
- 子問題的答案能夠拼湊出母問題的答案

2. 重疊子問題

- 在求解子問題時，會使用到其餘子問題的答案

3. 無後效性

- 每次解決子問題時，該子問題的解答只跟之前求解過的子問題有關
- 這些子問題的答案可以視作一個有向無環圖(DAG)



動態規劃簡介

- 動態規劃名詞簡介：

1. 狀態

- 切割後每個子問題的特性、資料或解答

2. 階段

- 把性質類似並且可同時處理的狀態集合在一起

3. 決策

- 在每個階段中所下的選擇

4. 狀態轉移方程式

- 透過其它子問題的狀態求出另一子問題狀態的解

- 複雜度

- 時間：通常為狀態總數 \times 每次狀態轉移方程式的複雜度
- 空間：因未來可能被使用而需記錄下的所有狀態數目

動態規劃簡介

- 狀態轉移方程式

1. 斐波那契數列

- $f(n) = f(n - 1) + f(n - 2)$

- $f(n) = f(n - 1) + f(n - 2) + f(n - 3)$

2. Min Cost Climbing Stairs

- $Step[i] = \min(Step[i - 2], Step[i - 1]) + cost[i]$

- 找出狀態轉移方程式就差不多結束了！

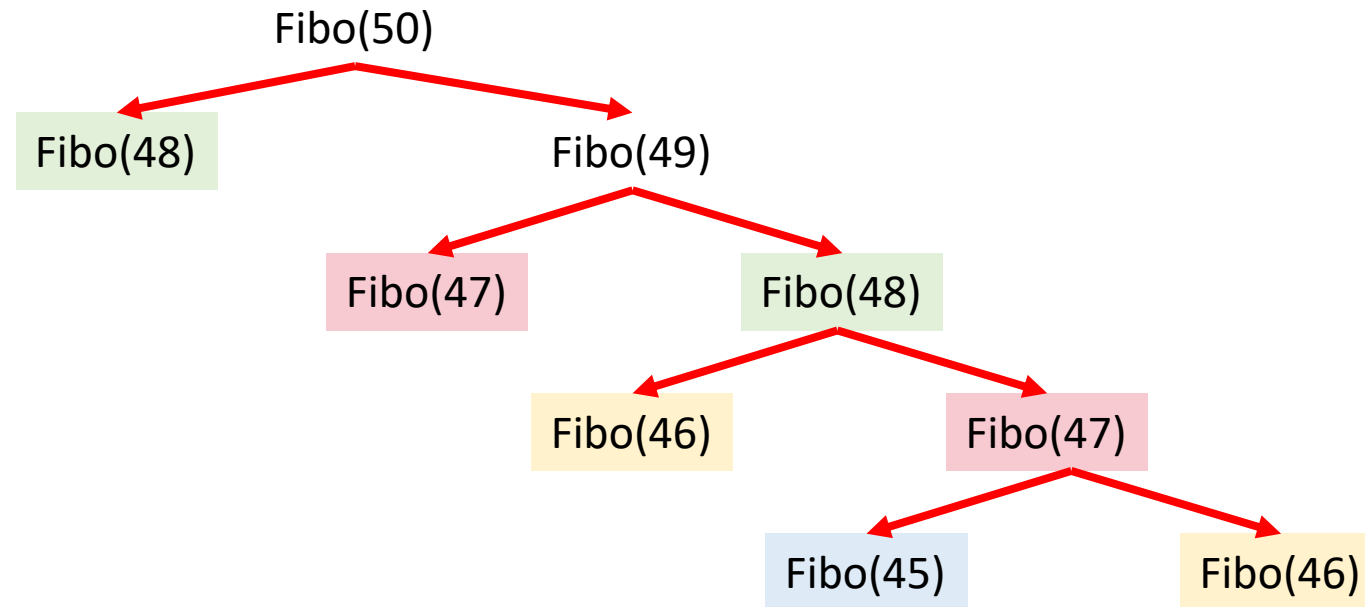
動態規劃簡介

- 動態規劃法求解的兩方式
 1. Top-down with memoization
 - 跟遞迴類似
 - ✓ 從母問題一路往下解
 - ✓ 但每次得到子問題的答案就記下來
 2. Bottom-up method
 - 從小的地方一路算到大的
 - 通常使用 Bottom-up

動態規劃簡介

Top-down with memoization

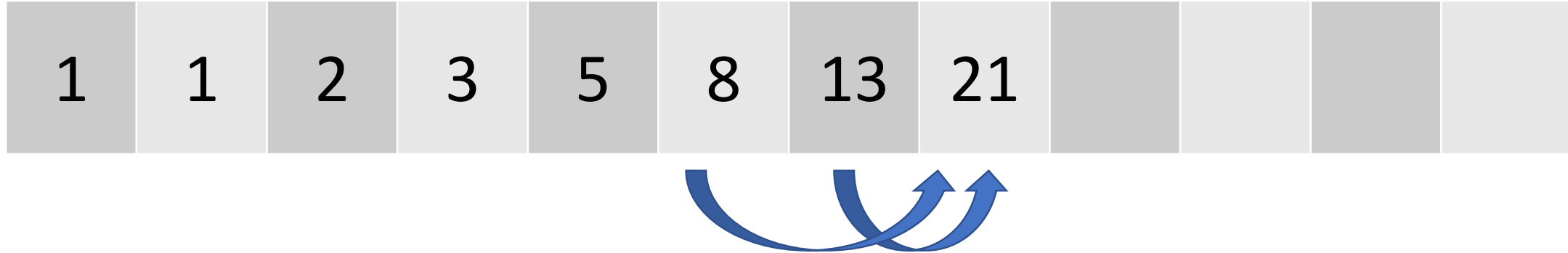
```
int Fibo_Top_Down(int n){  
    if(n<=2)  
        Fibo[n-1] = 1;  
    else if(Fibo[n-1]==0)  
        Fibo[n-1] = Fibo_Top_Down(n-1) + Fibo_Top_Down(n-2);  
    return Fibo[n-1];  
}
```



動態規劃簡介

Bottom-up

```
int Fibo_Bottom_Up(int n){  
    Fibo[0] = Fibo[1] = 1;  
    for(int i=2;i<n;i++)  
        Fibo[i] = Fibo[i-1] + Fibo[i-2];  
    return Fibo[n-1];  
}
```



動態規劃簡介

- 動態規劃法求解的兩方式
 1. Top-down with memoization
 - 通常用遞迴來解
 - ✓ 每次呼叫函式需佔用記憶體
 - ✓ 因此較耗費記憶體空間
 2. Bottom-up method
 - 通常用疊代來解決
 - ✓ 逐步把狀態填上
 - ✓ 狀態通常存在於陣列中

找錢問題



顧客需要找 20 元，而目前有三種硬幣：

1. 1 元
2. 5 元
3. 8 元

如何找錢可以讓找的硬幣「數目」最少？

貪婪演算法可行嗎？**不可行**

找錢問題

目前有三種硬幣：1 元、5 元、8 元

假設需要找 x 元時至少需要 $Coin(x)$ 枚硬幣

已知：

$$Coin(1) = 1$$

$$Coin(2) = 2$$

$$Coin(3) = 3$$

$$Coin(4) = 4$$

$$Coin(5) = 1$$

$$Coin(6) = ?$$

找錢問題

6 元硬幣可能有兩種情形：

1. 至少有一枚 1 元硬幣： $1 + \textit{Coin}(5)$
2. 至少有一枚 5 元硬幣： $1 + \textit{Coin}(1)$

選這兩種狀況中比較少的：

$$\textit{Coin}(6) = \min(1 + \textit{Coin}(5), 1 + \textit{Coin}(1))$$

找錢問題

x 元硬幣可能有三種情形，從中選硬幣數最少的：

1. 至少有一枚 1 元硬幣： $1 + \text{Coin}(x - 1)$
2. 至少有一枚 5 元硬幣： $1 + \text{Coin}(x - 5)$
3. 至少有一枚 8 元硬幣： $1 + \text{Coin}(x - 8)$

狀態轉移方程式

$\text{Coin}(x)$

$$= \min(1 + \text{Coin}(x - 1), 1 + \text{Coin}(x - 5), 1 + \text{Coin}(x - 8))$$

$$= 1 + \min(\text{Coin}(x - 1), \text{Coin}(x - 5), \text{Coin}(x - 8))$$

找錢問題

Coin(1)~Coin(5)



$$\text{Coin}(6) = 1 + \min(\text{Coin}(5), \text{Coin}(1), \text{Coin}(-2))$$



$$\text{Coin}(7) = 1 + \min(\text{Coin}(6), \text{Coin}(2), \text{Coin}(-1))$$



$$\text{Coin}(8) = 1 + \min(\text{Coin}(7), \text{Coin}(3), \text{Coin}(0))$$



$$\text{Coin}(9) = 1 + \min(\text{Coin}(8), \text{Coin}(4), \text{Coin}(1))$$



$$\text{Coin}(10) = 1 + \min(\text{Coin}(9), \text{Coin}(5), \text{Coin}(2))$$



找錢問題

1	2	3	4	1	2	4	3	2	2	3								
1	2	3	4	1	2	4	3	2	2	3	4							
1	2	3	4	1	2	4	3	2	2	3	4	2						
1	2	3	4	1	2	4	3	2	2	3	4	2	3					
1	2	3	4	1	2	4	3	2	2	3	4	2	3	3				
1	2	3	4	1	2	4	3	2	2	3	4	2	3	3	4	3		
1	2	3	4	1	2	4	3	2	2	3	4	2	3	3	4	3	3	4
1	2	3	4	1	2	4	3	2	2	3	4	2	3	3	4	3	3	4

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/>

最大子數列問題

給定一陣列，陣列的值有正有負，請找出一區間[a,b]可以使區間內的**元素總和**最大，並回傳該**元素總和**。

8	-5	-1	4	-3	6	2	-2	3	4
---	----	----	---	----	---	---	----	---	---

暴力解：

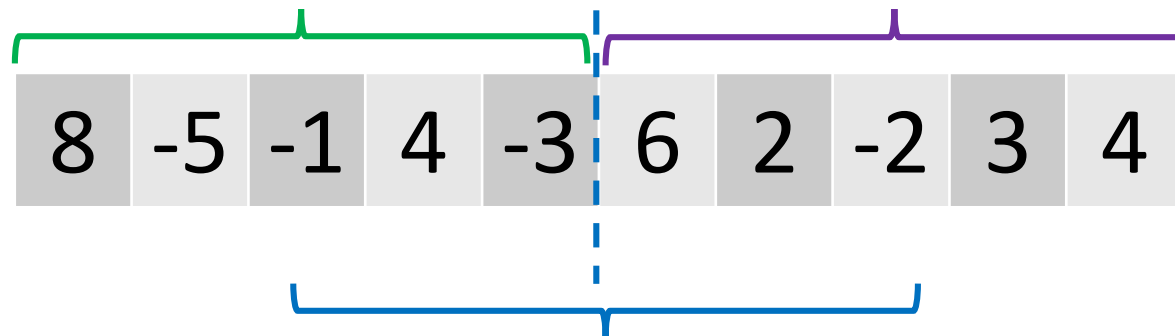
$O(n^3)$

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int n = nums.size();
        int sum[n][n]; // Should use malloc or vector instead
        int maximum = -2147483648;
        for(int start=0;start<n;start++){
            for(int finish=start;finish<n;finish++){
                sum[start][finish] = 0;
                for(int k=start;k<=finish;k++){
                    sum[start][finish] += nums[k];
                }
                if(sum[start][finish]>maximum){
                    maximum = sum[start][finish];
                }
            }
        }
        return maximum;
    }
};
```

$O(n^3)$

最大子數列問題

$max_subarray(0, \lfloor \frac{n}{2} \rfloor)$ $max_subarray(\lfloor \frac{n}{2} \rfloor, n)$



$max_cross_array(0, n)$

1. $max_subarray(0, \lfloor \frac{n}{2} \rfloor)$

2. $max_subarray(\lfloor \frac{n}{2} \rfloor, n)$

3. $max_cross_array(0, n)$

回傳其中的最大值！

最大子數列問題

判斷該數是否加入此最大數列：

1. 如果加了比較大，就加 (-10)
2. 如果沒有比較大，就用自己的 (-2)

-8	-2	3	4	-3	6	-3	4	-3	1
----	----	---	---	----	---	----	---	----	---

$DP[i]$

只算自己的就好

$$= \max(\underbrace{DP[i-1] + Data[i]}_{\text{把到目前為止的最大值}}, \underbrace{Data[i]}_{\text{只算自己的就好}})$$

把到目前為止的最大值

狀態轉移方程式

最大子數列問題

-8	-2	3	4	-3	6	-3	4	-3	1
----	----	---	---	----	---	----	---	----	---

$$DP[0] = \max(\textcolor{red}{DP}[-1] + Data[0], Data[0])$$

-8									
----	--	--	--	--	--	--	--	--	--

$$DP[1] = \max(DP[0] + Data[1], Data[1])$$

-8	-2								
----	----	--	--	--	--	--	--	--	--

$$DP[2] = \max(DP[1] + Data[2], Data[2])$$

-8	-2	3							
----	----	---	--	--	--	--	--	--	--

$$DP[3] = \max(DP[2] + Data[3], Data[3])$$

-8	-2	3	7						
----	----	---	---	--	--	--	--	--	--

$$DP[4] = \max(DP[3] + Data[4], Data[4])$$

-8	-2	3	7	4					
----	----	---	---	---	--	--	--	--	--

最大子數列問題

-8	-2	3	4	-3	6	-3	4	-3	1
----	----	---	---	----	---	----	---	----	---

$$DP[5] = \max(DP[4] + Data[5], Data[5])$$

-8	-2	3	7	4	10				
----	----	---	---	---	----	--	--	--	--

$$DP[6] = \max(DP[5] + Data[6], Data[6])$$

-8	-2	3	7	4	10	7			
----	----	---	---	---	----	---	--	--	--

$$DP[7] = \max(DP[6] + Data[7], Data[7])$$

-8	-2	3	7	4	10	7	11		
----	----	---	---	---	----	---	----	--	--

$$DP[8] = \max(DP[7] + Data[8], Data[8])$$

-8	-2	3	7	4	10	7	11	8	
----	----	---	---	---	----	---	----	---	--

$$DP[9] = \max(DP[8] + Data[9], Data[9])$$

-8	-2	3	7	4	10	7	11	8	9
----	----	---	---	---	----	---	----	---	---

$O(n)$

最大子數列問題

-8	-2	3	4	-3	6	-3	4	-3	1
----	----	---	---	----	---	----	---	----	---

$$DP[5] = \max(DP[4] + Data[5], Data[5])$$

-8	-2	3	7	4	10				
----	----	---	---	---	----	--	--	--	--

$$DP[6] = \max(DP[5] + Data[6], Data[6])$$

-8	-2	3	7	4	10	7			
----	----	---	---	---	----	---	--	--	--

$$DP[7] = \max(DP[6] + Data[7], Data[7])$$

-8	-2	3	7	4	10	7	11		
----	----	---	---	---	----	---	----	--	--

$$DP[8] = \max(DP[7] + Data[8], Data[8])$$

-8	-2	3	7	4	10	7	11	8	
----	----	---	---	---	----	---	----	---	--

$$DP[9] = \max(DP[8] + Data[9], Data[9])$$

-8	-2	3	7	4	10	7	11	8	9
----	----	---	---	---	----	---	----	---	---

$O(n)$

Practice

Mission

Try LeetCode #53. Maximum Subarray

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

- Example 1:
 - Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`
 - Output: 6
 - Explanation: `[4,-1,2,1]` has the largest sum = 6.

Ref : <https://leetcode.com/problems/maximum-subarray/>

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

s_1				f_1			
					s_2		f_2

s_1				f_1			
			s_2		f_2		

s_1				f_1			
	s_2		f_2				

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

活動選擇問題



如果活動有權重 (重要性有別) 的話呢？

- Weighted Interval Scheduling Problem
- 請使用動態規劃！
- 輸出最好選擇時的權重

		Days									
1	v_1										
2	v_2										
3	v_3										
4	v_4										
5	v_5										

活動選擇問題



把活動依照**結束**日期進行排序！

		Days									
1	v_1										
2	v_2										
3	v_3										
4	v_4										
5	v_5										

活動選擇問題

分成兩種選擇

1. 選擇活動 5

➤ $wis(10) = wis(7) + v_5$

2. 不選擇活動 5

➤ $wis(10) = wis(9)$

s_i 可換成在活動開始前最後結束的活動

$$wis(f_i) = \max(wis(s_i) + v_i, wis(f_i - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5								3		

$wis(s_i)$
 v_i

活動選擇問題

$$wis(f_i) = \max(wis(s_i) + v_i, wis(f_i - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5								3		

Days	1	2	3	4	5	6	7	8	9	10
Value										

活動選擇問題

$$wis(n) = \max(wis(s_i) + v_i, wis(n - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5								3		

Days	1	2	3	4	5	6	7	8	9	10
Value	0	0	0							

活動選擇問題

$$wis(n) = \max(wis(s_i) + v_i, wis(n - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5								3		

$\underbrace{\hspace{1.5cm}}_{wis(s_i)} \quad \underbrace{\hspace{1.5cm}}_{v_i}$

Days	1	2	3	4	5	6	7	8	9	10
Value	0	0	0	3						

$$\begin{aligned}
 & wis(4) \\
 &= \max(wis(1) + 3, wis(3)) \\
 &= 3
 \end{aligned}$$

活動選擇問題

$$wis(n) = \max(wis(s_i) + v_i, wis(n - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5							3			

v_i

Days	1	2	3	4	5	6	7	8	9	10
Value	0	0	0	3	4					

$$\begin{aligned}
 & wis(5) \\
 &= \max(wis(0) + 4, wis(4)) \\
 &= 4
 \end{aligned}$$

活動選擇問題

$$wis(n) = \max(wis(s_i) + v_i, wis(n - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5								3		

$wis(s_i)$

v_i

Days	1	2	3	4	5	6	7	8	9	10
Value	0	0	0	3	4	4	4			

$$\begin{aligned}
 & wis(7) \\
 &= \max(wis(4) + 1, wis(6)) \\
 &= 4
 \end{aligned}$$

活動選擇問題

$$wis(n) = \max(wis(s_i) + v_i, wis(n - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5							3			

$wis(s_i)$

v_i

Days	1	2	3	4	5	6	7	8	9	10
Value	0	0	0	3	4	4	4	6		

$$\begin{aligned}
 & wis(8) \\
 &= \max(wis(5) + 2, wis(7)) \\
 &= 6
 \end{aligned}$$

活動選擇問題

$$wis(n) = \max(wis(s_i) + v_i, wis(n - 1))$$

		Days									
		1	2	3	4	5	6	7	8	9	10
1	v_1		3								
2	v_2	4									
3	v_3					1					
4	v_4						2				
5	v_5								3		

$wis(s_i)$

v_i

Days	1	2	3	4	5	6	7	8	9	10
Value	0	0	0	3	4	4	4	6	6	7

$$\begin{aligned}
 & wis(10) \\
 &= \max(wis(7) + 3, wis(9)) \\
 &= 7
 \end{aligned}$$

Practice

Mission

Try LeetCode #1235. Maximum Profit in Job Scheduling

We have n jobs, where every job is scheduled to be done from $\text{startTime}[i]$ to $\text{endTime}[i]$, obtaining a profit of $\text{profit}[i]$.

You're given the startTime , endTime and profit arrays, return the maximum profit you can take such that there are no two jobs in the subset with overlapping time range.

If you choose a job that ends at time X you will be able to start another job that starts at time X .

Ref : <https://leetcode.com/problems/maximum-profit-in-job-scheduling/>

動態規劃常見應用

郵票問題

需要付 n 元的郵資，而且已知每個郵票的價格為

v_i	1	3	10	12	18
-------	---	---	----	----	----

該如何安排郵票的貼法，可以讓郵票張數最少？



郵票問題

需要付 n 元的郵資，而且已知每個郵票的價格為

v_i	1	3	10	12	18
-------	---	---	----	----	----

該如何安排郵票的貼法，可以讓郵票張數最少？

$$Stamp(n) =$$

$$\begin{aligned} & \min(1 + Stamp(n - 1), 1 + Stamp(n - 3), 1 + Stamp(n - 10), 1 + Stamp(n - 12), 1 + Stamp(n - 18)) \\ &= 1 + \min(Stamp(n - 1), Stamp(n - 3), Stamp(n - 10), Stamp(n - 12), Stamp(n - 18)) \end{aligned}$$

跟找錢問題一樣！

切割問題



有一長度為 n 的木頭，並且知道市場上的木材
長度與價格的表格如下：

l_i	1	2	3	4	5
p_i	10	22	35	45	56

該如何切割木頭以求利潤最大化？

切割問題

暴力解：

長度為 n 的木頭，總共會有 $n-1$ 個切割點

每個切割點都可以選擇切或不切。

$$O(2^{n-1})$$



切割問題

分治法：

長度為 n 的木頭，總共會有 $n-1$ 個切割點

每個切割點都可以選擇切或不切，從中選最大的

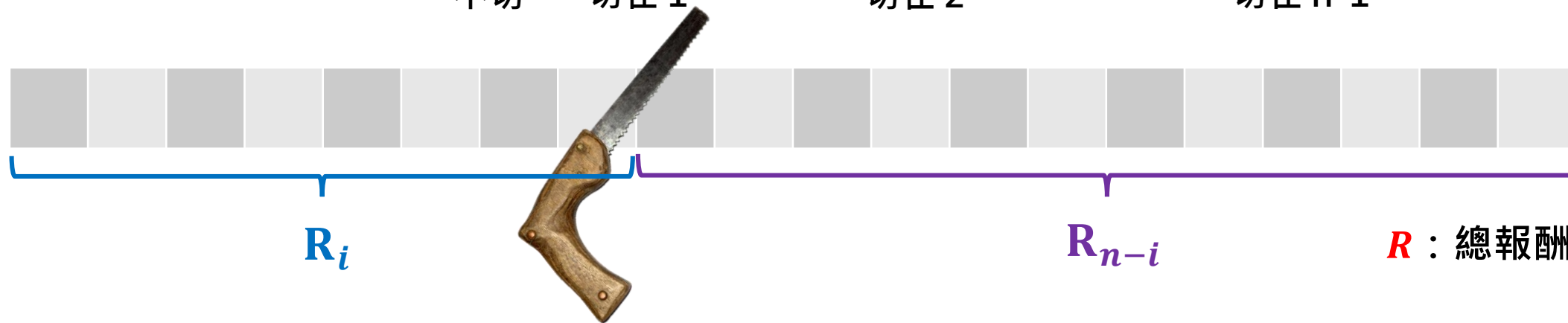
$$R_n = \max(p_n, R_1 + R_{n-1}, R_2 + R_{n-2}, \dots, R_{n-1} + R_1)$$

不切

切在 1

切在 2

切在 $n-1$



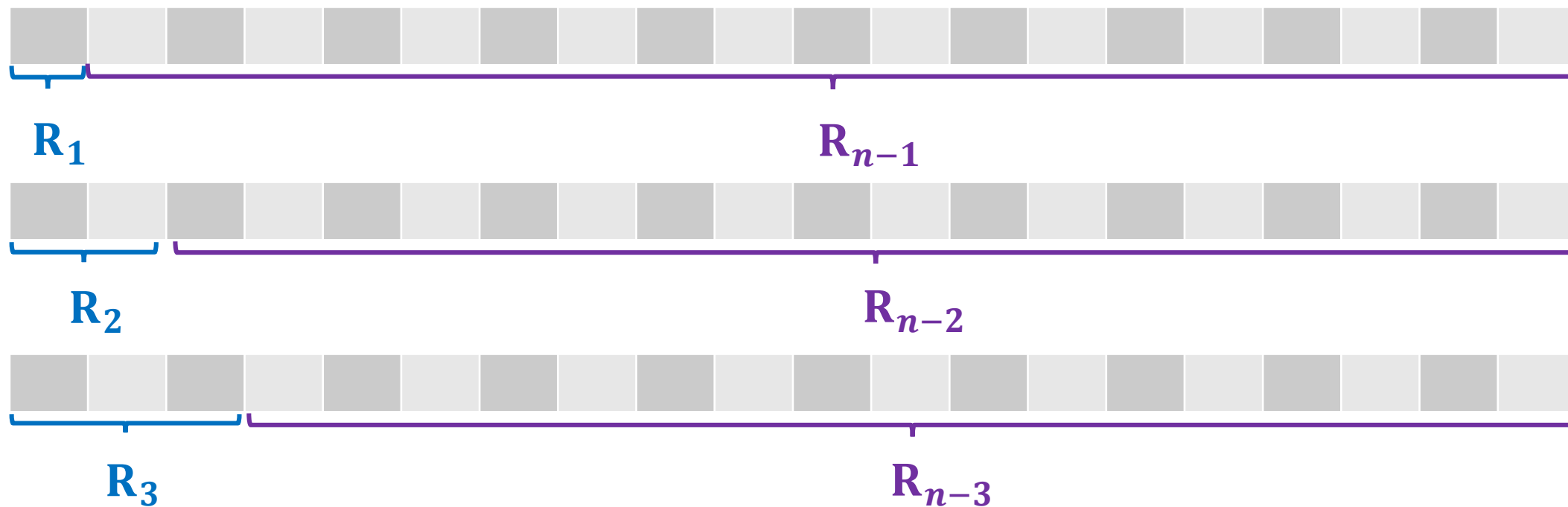
$$R_n = \max_{0 \leq i \leq n-1} (R_i + R_{n-i})$$

切割問題

分治法：

為了減少子問題的數目，每次都從左邊切割一段下來

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

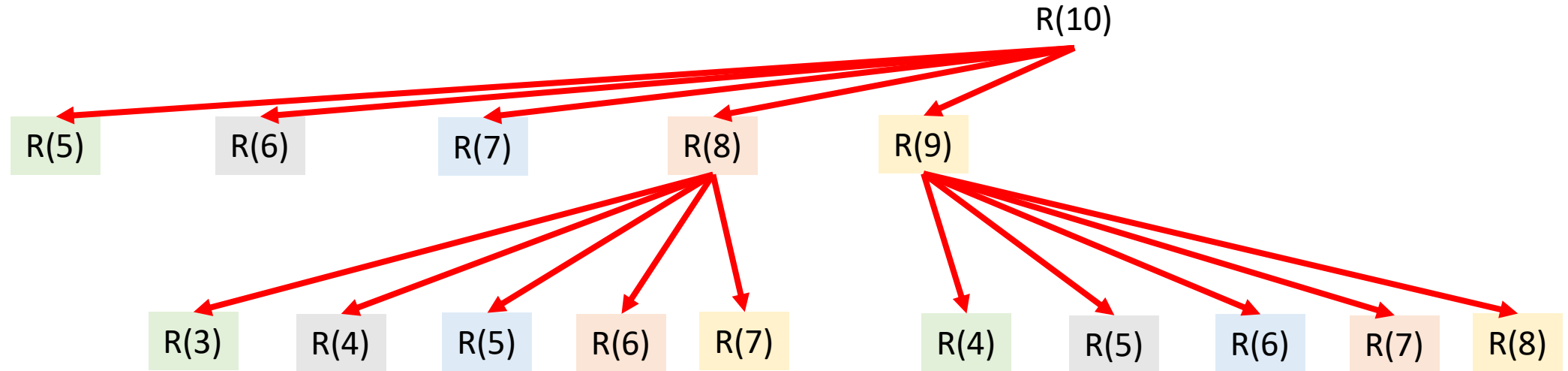


切割問題

這是動態規劃嗎？

```
int Cut_Rod(int* p, int p_len, int n){
    if(n==0)
        return 0;
    int revenue = -2147483648;
    for(int i=0;i<p_len;i++){
        if(n>=i+1){
            int revenue_i = p[i] + Cut_Rod(p,p_len,n-i-1);
            revenue = revenue>revenue_i?revenue:revenue_i;
        }
    }
    return revenue;
}
```

切割問題



$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ \Theta(1) + \sum_{i=1}^{len_p} T(n - l_i), & \text{otherwise} \end{cases}$$

沒有存結果導致效率低落！

切割問題

這是動態規劃嗎？

```
int Cut_Rod(int* p, int p_len, int n){
    if(n==0)
        return 0;
    int revenue = -2147483648;
    for(int i=0;i<p_len;i++){
        if(n>=i+1){
            int revenue_i = p[i] + Cut_Rod(p,p_len,n-i-1);
            revenue = revenue>revenue_i?revenue:revenue_i;
        }
    }
    return revenue;
}
```

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n																			

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n	10																		

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n	10	22																	

$$\begin{aligned} R_2 &= \max(R_1 + R_1, R_2 + R_0) \\ &= \max(10 + 10, 22 + 0) \\ &= 22 \end{aligned}$$

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n	10	22	35																

$$\begin{aligned} R_3 &= \max(R_1 + R_2, R_2 + R_1, R_3 + R_0) \\ &= \max(10 + 22, 22 + 10, 35 + 0) \\ &= 35 \end{aligned}$$

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n	10	22	35	45															

R_4

$$= \max(R_1 + R_3, R_2 + R_2, R_3 + R_1, R_4 + R_0)$$

$$= \max(10 + 35, 22 + 22, 35 + 10, 45 + 0)$$

$$= 45$$

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n	10	22	35	45	57														

$$\begin{aligned} R_5 &= \max(R_1 + R_4, R_2 + R_3, R_3 + R_2, R_4 + R_1, R_5 + R_0) \\ &= \max(10 + 45, 22 + 35, 35 + 22, 45 + 10, 56 + 0) \\ &= 57 \end{aligned}$$

切割問題

$$R_n = \max_{1 \leq i \leq \text{len}_p} (R_{p_i} + R_{n-p_i})$$

l_i	1	2	3	4	5
p_i	10	22	35	45	56

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
R_n	10	22	35	45	57	70													

$$\begin{aligned} &R_6 \\ &= \max(R_1 + R_5, R_2 + R_4, R_3 + R_3, R_4 + R_2, R_5 + R_1) \\ &= \max(10 + 57, 22 + 45, 35 + 35, 45 + 22, 57 + 10) \\ &= 70 \end{aligned}$$

切割問題

```
int Cut_Rod(int* p, int p_len, int n){
    if(n==0)
        return 0;
    int revenue_array[n+1] = {0};
    for (int i=1;i<=n;i++)
    {
        int max_revenue = -2147483648;
        for (int j=0;j<p_len;j++){
            if(i<=j)
                break;
            int revenue_j = p[j]+revenue_array[i-j-1];
            max_revenue = max_revenue>revenue_j?max_revenue:revenue_j;
        }
        revenue_array[i] = max_revenue;
    }

    return revenue_array[n];
}
```

Practice

Mission

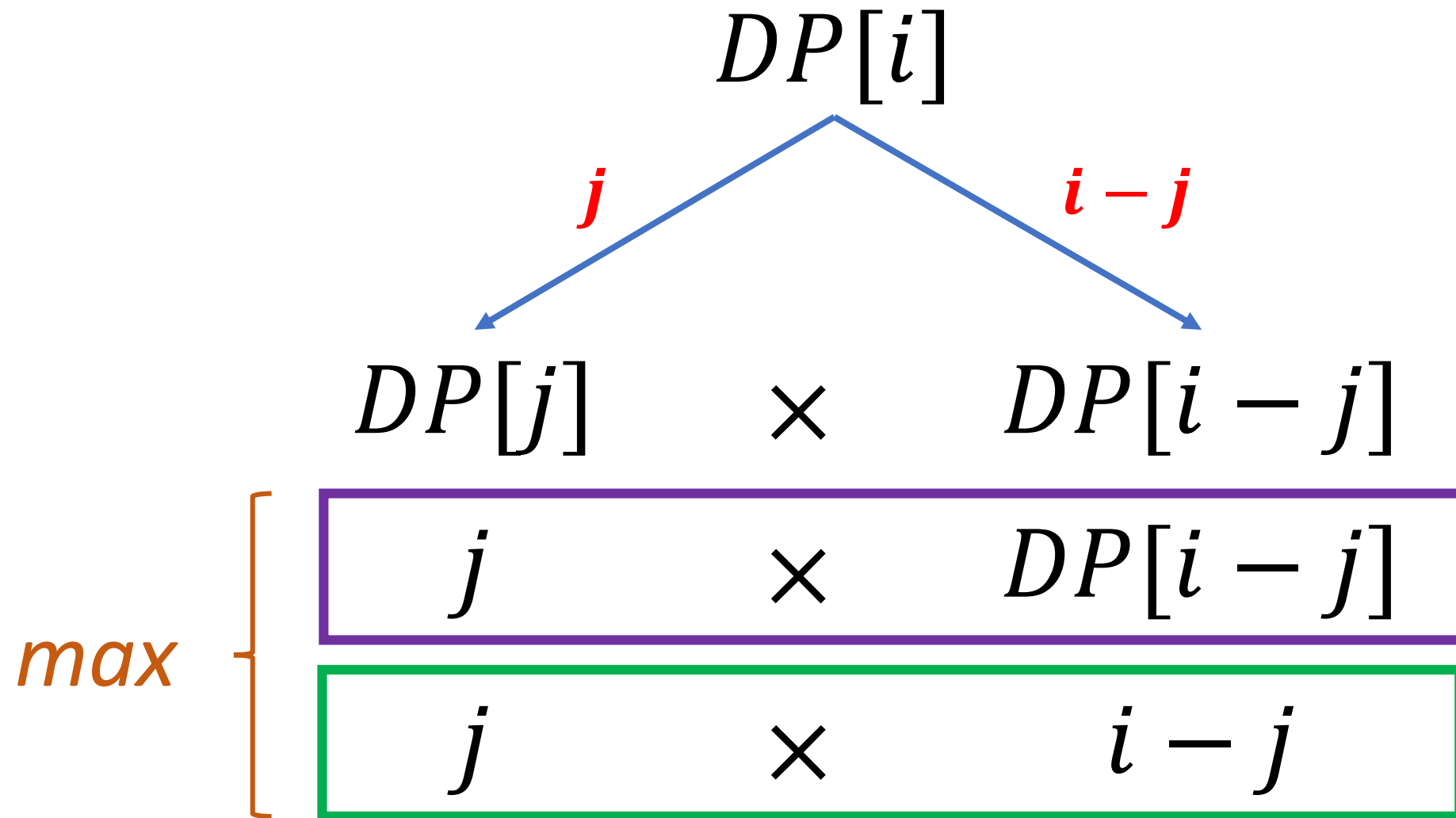
Try LeetCode #343. Integer Break

Given an integer n , break it into the sum of k positive integers, where $k \geq 2$, and maximize the product of those integers. Return the maximum product you can get.

- Example:
 - Input: $n = 2$
 - Output: 1
 - Explanation: $2 = 1 + 1, 1 \times 1 = 1$.

Ref : <https://leetcode.com/problems/integer-break/>

Practice



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

以下牽涉到二維的 DP 比較抽象，可自行斟酌服用

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



限制

- 每個物品最多只能放一個
 - ✓ Unbounded Knapsack Problem 沒有限制

典型的 NP-complete 問題

- 無法快速求得最佳解
- 當範圍不大可以用動態規劃快速求得最佳解

暴力解：

- 物品數量 N ，所有選擇的子集合總共 $O(2^N)$

背包問題

$DP[i][j]$ ：前 i 件物品放入容量 j 的背包所能產生的最大價值

1. 不放第 i 件物品 \rightarrow 前 $i-1$ 件物品所能產生的最大價值

➤ $DP[i-1][j]$

2. 放第 i 件物品 \rightarrow 前 $i-1$ 件物品放入 $j-w_i$ 的背包中

➤ $DP[i-1][j-w_i] + v_i$

狀態轉移方程式：

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0						
1	2						
2	5						
3	8						

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0					
2	5	0					
3	8	0					

放不下

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2				
2	5	0					
3	8	0					

$$\begin{aligned} & DP[1][1] \\ &= \max(DP[0][1], DP[0][0] + 2) \\ &= \max(0, 2) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2				
2	5	0	2				
3	8	0					

$$\begin{aligned} & DP[2][1] \\ &= \max(DP[1][1], DP[1][-1] + 5) \\ &= \max(2, X) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2				
2	5	0	2				
3	8	0	2				

$$\begin{aligned} & DP[3][1] \\ &= \max(DP[2][1], DP[2][-2] + 8) \\ &= \max(2, X) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2			
2	5	0	2				
3	8	0	2				

$$\begin{aligned} & DP[1][2] \\ &= \max(DP[0][2], DP[0][1] + 2) \\ &= \max(0, 2) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2			
2	5	0	2	5			
3	8	0	2				

$$\begin{aligned} & DP[2][2] \\ &= \max(DP[1][2], DP[1][0] + 5) \\ &= \max(2, 0 + 5) = 5 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2			
2	5	0	2	5			
3	8	0	2	5			

$$\begin{aligned} & DP[3][2] \\ &= \max(DP[2][2], DP[2][-1] + 8) \\ &= \max(5, X) = 5 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2		
2	5	0	2	5			
3	8	0	2	5			

$$\begin{aligned} & DP[1][3] \\ &= \max(DP[0][3], DP[0][2] + 2) \\ &= \max(0, 0 + 2) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2		
2	5	0	2	5	7		
3	8	0	2	5			

$$\begin{aligned} & DP[2][3] \\ &= \max(DP[1][3], DP[1][1] + 5) \\ &= \max(2, 2 + 5) = 7 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2		
2	5	0	2	5	7		
3	8	0	2	5	8		

$$\begin{aligned} & DP[3][3] \\ &= \max(DP[2][3], DP[2][0] + 8) \\ &= \max(7, 0 + 8) = 8 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2	2	
2	5	0	2	5	7		
3	8	0	2	5	8		

$$\begin{aligned} & DP[1][4] \\ &= \max(DP[0][4], DP[0][3] + 2) \\ &= \max(0, 0 + 2) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2	2	
2	5	0	2	5	7	7	
3	8	0	2	5	8		

$$\begin{aligned} & DP[2][4] \\ &= \max(DP[1][4], DP[1][2] + 5) \\ &= \max(2, 2 + 5) = 7 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2	2	
2	5	0	2	5	7	7	
3	8	0	2	5	8	10	

$$\begin{aligned} & DP[3][4] \\ &= \max(DP[2][4], DP[2][1] + 8) \\ &= \max(7, 2 + 8) = 10 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2	2	2
2	5	0	2	5	7	7	
3	8	0	2	5	8	10	

$$\begin{aligned} & DP[1][5] \\ &= \max(DP[0][5], DP[0][4] + 2) \\ &= \max(0, 0 + 2) = 2 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2	2	2
2	5	0	2	5	7	7	7
3	8	0	2	5	8	10	

$$\begin{aligned} & DP[2][5] \\ &= \max(DP[1][5], DP[1][3] + 5) \\ &= \max(2, 2 + 5) = 7 \end{aligned}$$

背包問題

w_i	1	2	3
v_i	2	5	8

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

w_i	v_i	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	2	0	2	2	2	2	2
2	5	0	2	5	7	7	7
3	8	0	2	5	8	10	13

$$\begin{aligned} & DP[3][5] \\ &= \max(DP[2][5], DP[2][2] + 8) \\ &= \max(7, 5 + 8) = 13 \end{aligned}$$

背包問題

$$DP[i][j] = \begin{cases} DP[i-1][j], & j < w_i \\ \max(DP[i-1][j], DP[i-1][j-w_i] + v_i), & j \geq w_i \end{cases}$$

Homework !

w_i	v_i	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0
4	6	0										
5	7	0										
2	5	0										
6	12	0										
3	5	0										
7	18	0										

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



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

Practice

Mission

Try LeetCode #518. Coin Change 2

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

Return the number of combinations that make up that amount. If that amount of money cannot be made up by any combination of the coins, return 0.

You may assume that you have an infinite number of each kind of coin. The answer is guaranteed to fit into a signed 32-bit integer.

Ref : <https://leetcode.com/problems/coin-change-2/>

Practice

$DP[i][j]$: 用前 i 種硬幣去找 j 元

$$DP[i][j] = \begin{cases} DP[i-1][j] & \text{不使用第 } i \text{ 種硬幣, 只使用前 } i-1 \text{ 種} \\ + \\ DP[i][j - \text{coin}[i-1]] & \text{使用至少一個第 } i \text{ 種硬幣} \end{cases}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1										
2	1										
5	1										

Practice

$DP[i][j]$: 用前 i 種硬幣去找 j 元

$$DP[i][j] = \begin{cases} DP[i-1][j] & \text{不使用第 } i \text{ 種硬幣, 只使用前 } i-1 \text{ 種} \\ + \\ DP[i][j - \text{coin}[i-1]] & \text{使用至少一個第 } i \text{ 種硬幣} \end{cases}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1									
2	1										
5	1										

Practice

$DP[i][j]$: 用前 i 種硬幣去找 j 元

$$DP[i][j] = \begin{cases} DP[i-1][j] & \text{不使用第 } i \text{ 種硬幣, 只使用前 } i-1 \text{ 種} \\ + \\ DP[i][j - \text{coin}[i-1]] & \text{使用至少一個第 } i \text{ 種硬幣} \end{cases}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1									
2	1	1									
5	1										

Practice

$DP[i][j]$: 用前 i 種硬幣去找 j 元

$$DP[i][j] = \begin{cases} DP[i-1][j] & \text{不使用第 } i \text{ 種硬幣, 只使用前 } i-1 \text{ 種} \\ + \\ DP[i][j - \text{coin}[i-1]] & \text{使用至少一個第 } i \text{ 種硬幣} \end{cases}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1									
2	1	1									
5	1	1									

Practice

$DP[i][j]$: 用前 i 種硬幣去找 j 元

$$DP[i][j] = \begin{cases} DP[i-1][j] & \text{不使用第 } i \text{ 種硬幣, 只使用前 } i-1 \text{ 種} \\ + \\ DP[i][j - \text{coin}[i-1]] & \text{使用至少一個第 } i \text{ 種硬幣} \end{cases}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1								
2	1	1									
5	1	1									

Practice

$DP[i][j]$: 用前 i 種硬幣去找 j 元

$$DP[i][j] = \begin{cases} DP[i-1][j] & \text{不使用第 } i \text{ 種硬幣, 只使用前 } i-1 \text{ 種} \\ + \\ DP[i][j - \text{coin}[i-1]] & \text{使用至少一個第 } i \text{ 種硬幣} \end{cases}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1								
2	1	1	2								
5	1	1									

Practice

$$\begin{aligned} & DP[3][1] \\ &= DP[2][1] + DP[3][-4] \\ &= DP[1][1] + DP[3][-4] + DP[2][-1] \\ &= \textcolor{red}{DP[0][1]} + DP[3][-4] + DP[2][-1] + DP[1][0] \\ &= DP[3][-4] + DP[2][-1] + DP[1][0] \end{aligned}$$

v_i	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1									
2		1									
5		1									

Practice

$$DP[i][j] = \begin{cases} DP[i-1][j] \\ + \\ DP[i][j - coin[i-1]] \end{cases}$$
$$DP[i-1][j] = \begin{cases} DP[i-2][j] \\ + \\ DP[i-1][j - coin[i-2]] \end{cases}$$
$$DP[i-2][j] = \begin{cases} DP[i-3][j] \\ + \\ DP[i-2][j - coin[i-3]] \end{cases}$$

.....

$$DP[1][j] = \begin{cases} DP[0][j] = 0 \\ + \\ DP[1][j - coin[0]] \end{cases}$$

$$DP[len_{coin}][j] = \sum_{i=1}^{len_{coin}} DP[i][j - coin[i-1]]$$

$DP[0] = 1$
 $for\ i = 0 \sim len_{coin} - 1$
 $for\ j = coin[i] \sim amount$
 $DP[j] += DP[j - coin[i]]$

動態規劃總結

動態規劃總結

- 最佳化問題先想想能否用貪婪解
- 無法用貪婪的狀況下再考慮動態規劃/分治法
 - 子問題間不相干→分治法
 - 子問題間相干→動態規劃

動態規劃總結

分治法與動態規劃的差異

	分治法	動態規劃
適用時機	子問題間並不交疊 (overlap)	子問題間交疊 (overlap)
演算過程	Top-Down 居多	Bottom-Up 居多
記憶體空間	不須額外空間	需要額外空間

動態規劃簡介

- 分治法
 - 把問題拆解成獨立或相鄰的子問題
 - 分別解決這些子問題的答案後再組回母問題的答案
 - ✓ 通常利用遞迴解決，有**可能重複計算**
 - ✓ 運算次數通常較多
- 貪婪演算法
 - 在解決問題過程中，依序選擇子問題的最佳解
 - 實作較為簡單，遇最佳化問題可優先考慮
- 動態規劃法
 - 把問題拆解成相關或交疊的子問題
 - 分別解決這些子問題的答案後再組回母問題的答案
 - ✓ 通常會儲存每個子問題的答案**避免重複計算**
 - ✓ 需要的記憶體空間通常較多