Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Dynamic Programming

Outline

- Intro to DP
- Knapsack problem
- Longest Increase Subsequence (LIS)
- Longest Common Subsequence (LCS)

What is DP ?

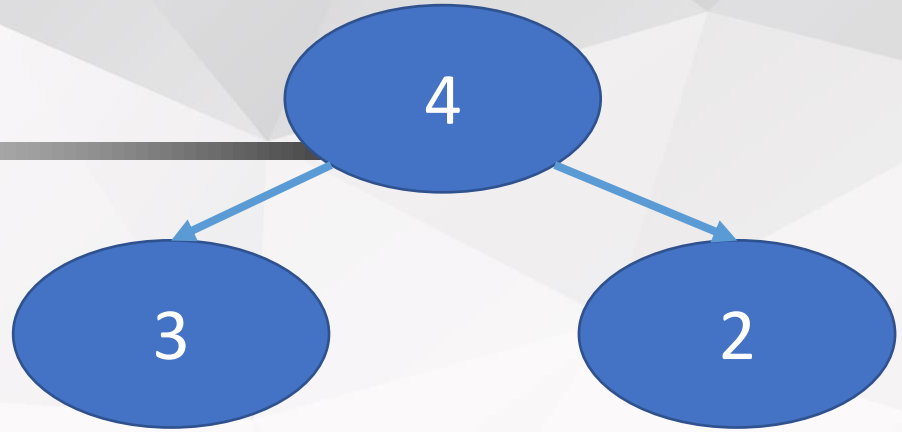
DP 是啥能吃嗎?

Intro to DP

- 計算費伯納契數列

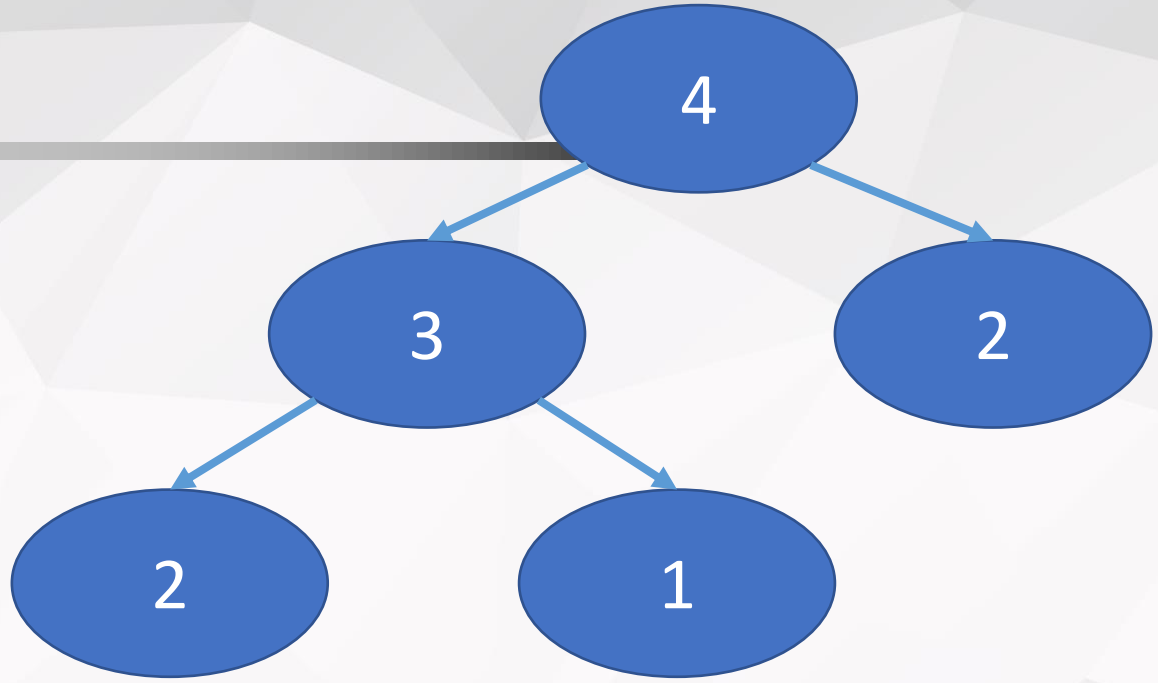
Intro to DP

- 計算費伯納契數列



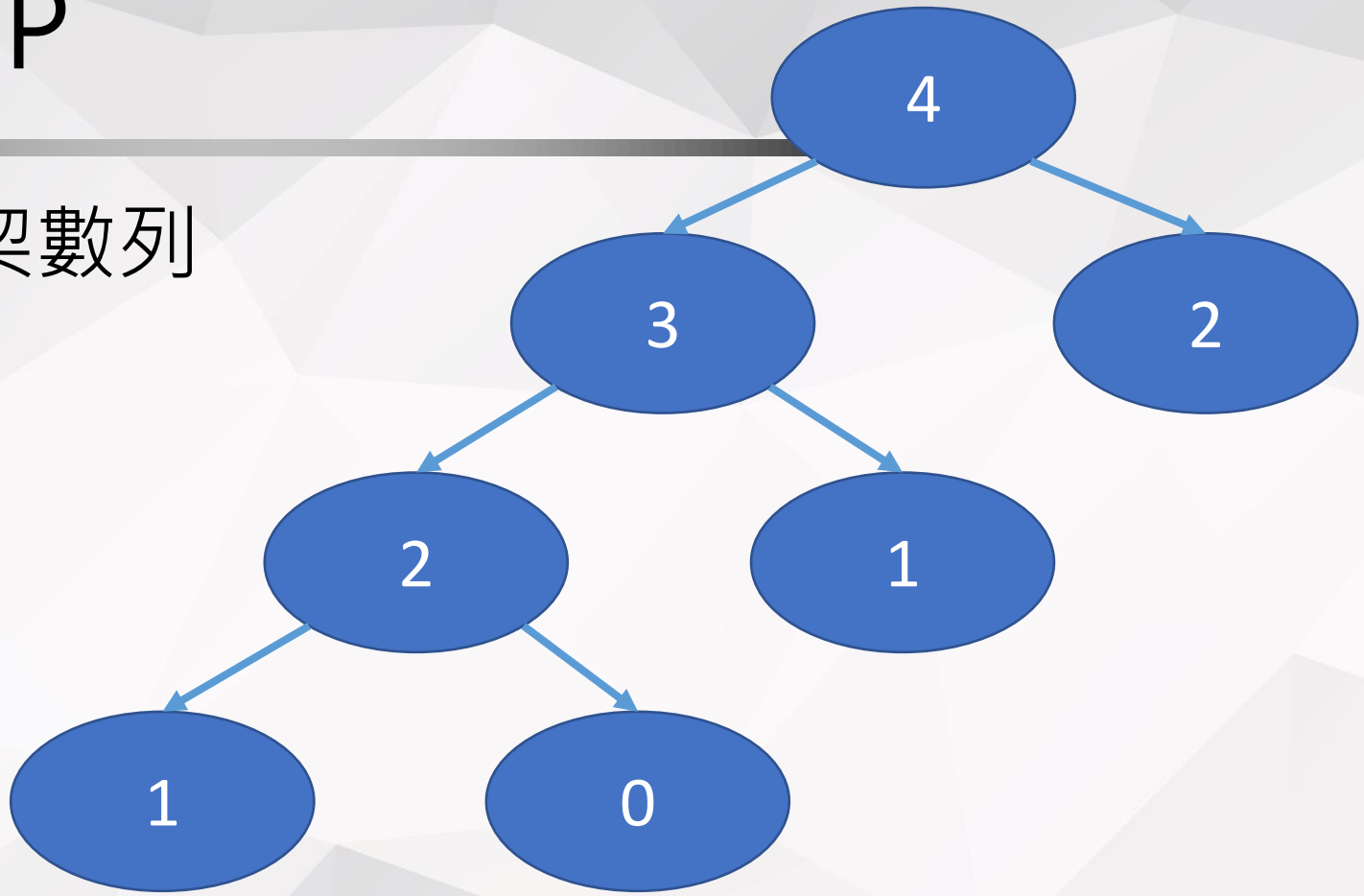
Intro to DP

- 計算費伯納契數列



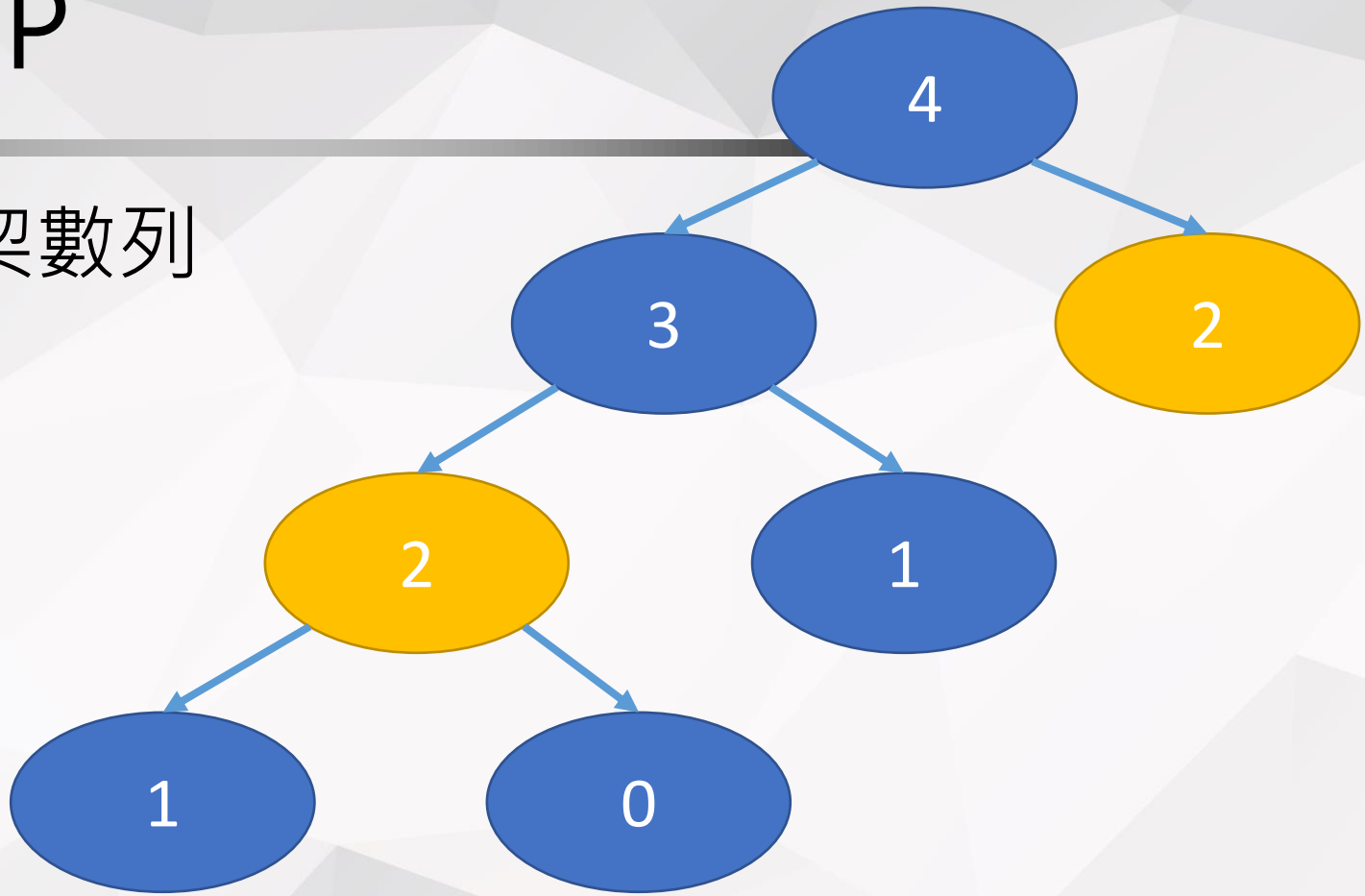
Intro to DP

- 計算費伯納契數列



Intro to DP

- 計算費伯納契數列



Intro to DP

- 動態規劃 = 分治 + 記憶化
- 三個重要性質
  - 最優子結構
  - 重複子問題
  - 無後效性

最優子結構

- 問題的解，可以由子問題的解推得。
  例如費氏數列 f[n] = f[n - 1] + f[n - 2]。

重複子問題

- 有很多子問題可歸為同樣的問題
- -> 引入記憶化(把已取得的解儲存)

無後效性

- 確定的子問題解，並不會受到其他決策影響

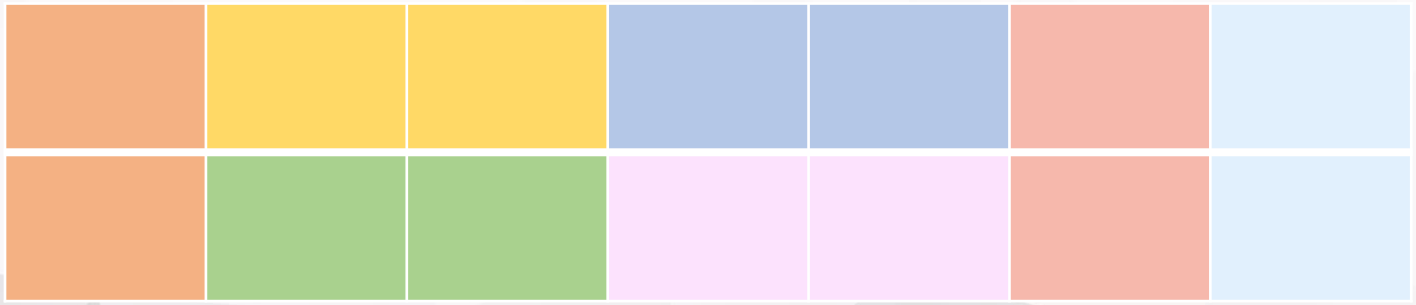
經典 DP 問題

學習經典不如學習創造經典

1 x 2 格填充 (弱)

1 x 2 格填充 (弱)

2 x N 的格子，
用 1 x 2 與 2 x 1 的格子填滿的方法數

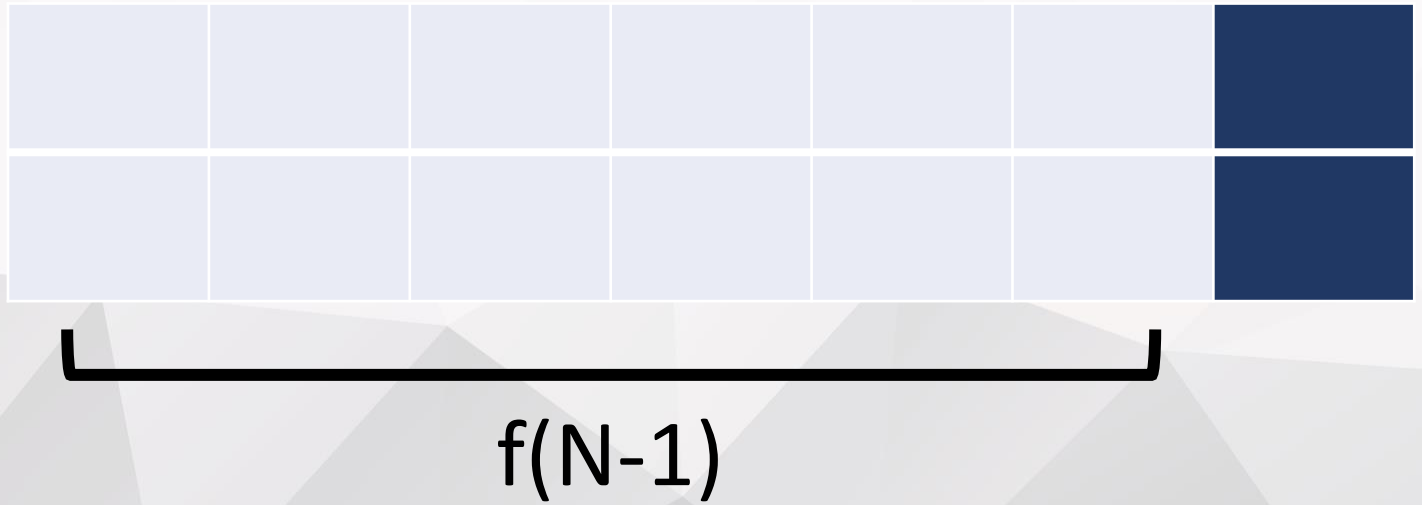


1 x 2 格填充 (弱)

定義: f(N) 為 2 x N 格的方法數

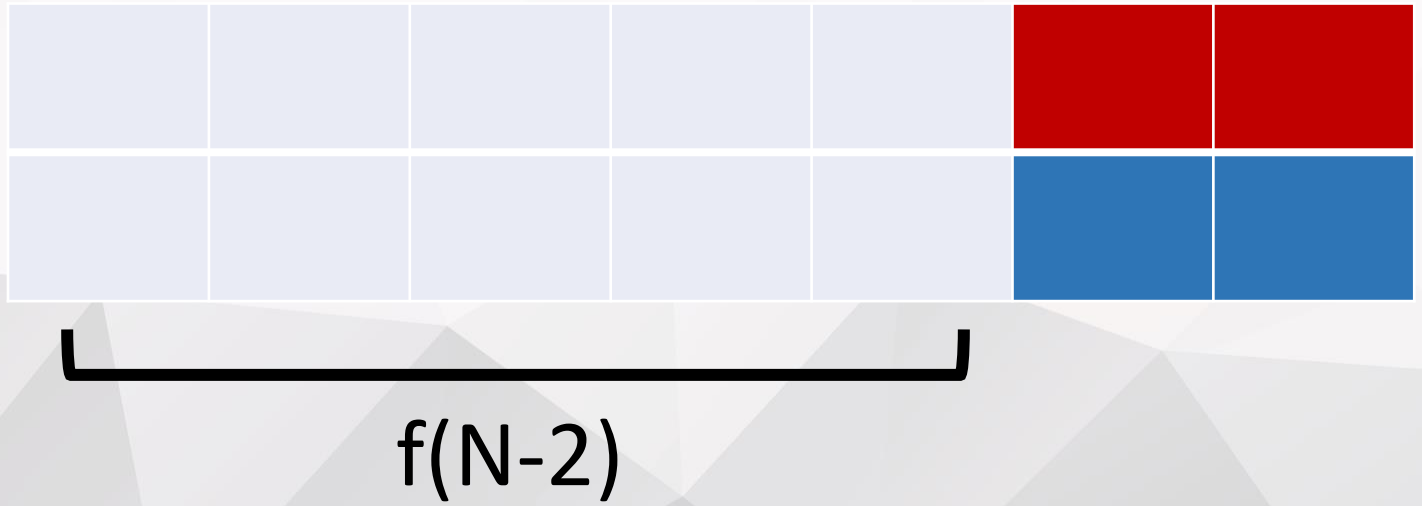
1 x 2 格填充 (弱)

狀態轉移: f(N) = f(N-1) +



1 x 2 格填充 (弱)

狀態轉移: f(N) = f(N-1) + f(N-2)



1 x 2 格填充 (弱)

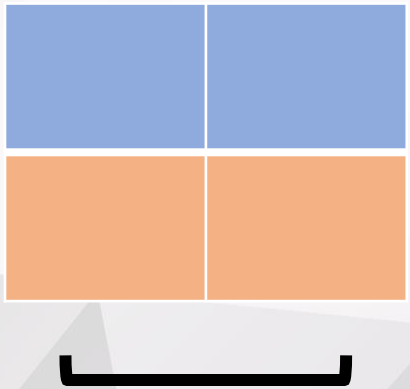
邊界: f(1) = 1

f(1)

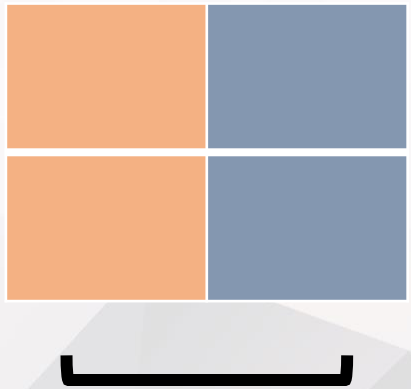
1 x 2 格填充 (弱)

邊界: f(1) = 1

f(2) = 2



f(2)



f(2)

紅綠藍

紅綠藍

1 x N 格子塗上紅、綠、藍三顏色，
且**藍綠不可相鄰**，有幾種塗法?

紅綠藍

如果定義: g(n) 為方法數
狀態轉移: 太難了
邊界: g(1) = 3

紅綠藍

填了一格就得看**前一次**填的顏色，

這一格填什麼 很重要 (狀態的考慮要納入)

紅綠藍

重新定義：

- f(n,0) 最後一格紅 方法數
- f(n,1) 最後一格綠 方法數
- f(n,2) 最後一格藍 方法數

紅綠藍

狀態轉移：

- f(n,0) = f(n-1, 0) + f(n-1, 1) + f(n-1, 2)
- f(n,1) = f(n-1, 0) + f(n-1, 1)
- f(n,2) = f(n-1, 0) + f(n-1, 2)

紅綠藍

邊界:

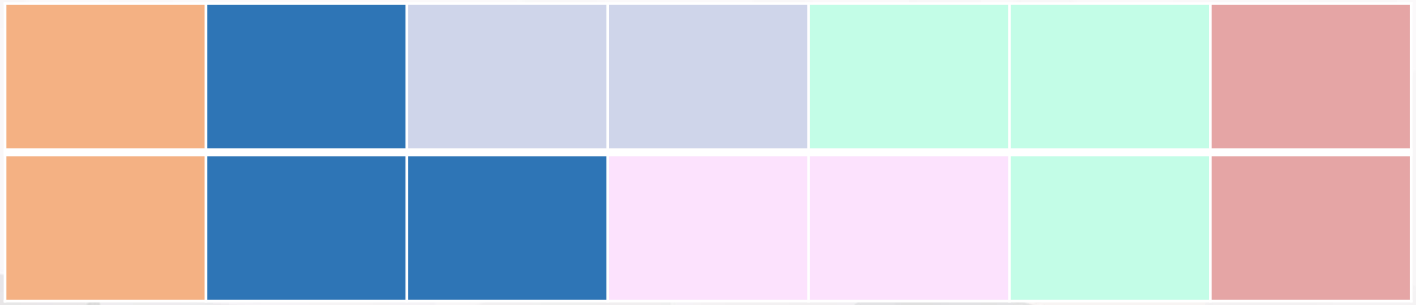
- f(1,0) = f(1,1) = f(1,2) = 1

- 最後把結果相加 g(n) = f(n,0) + f(n,1) + f(n,2)

1 x 2 格填充 (強)

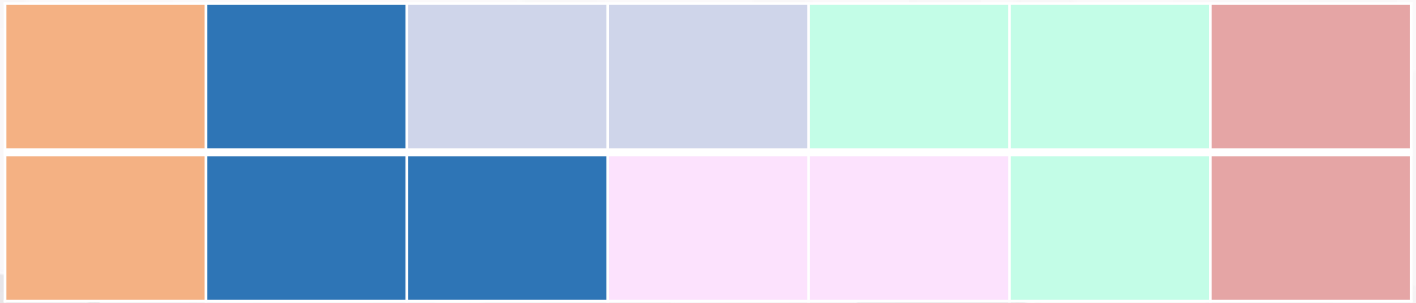
1 x 2 格填充 (強)

2 x N 的格子，
用 1 x 2 與 2 x 1 的格子
以及 **L 型格子** 填滿的方法數



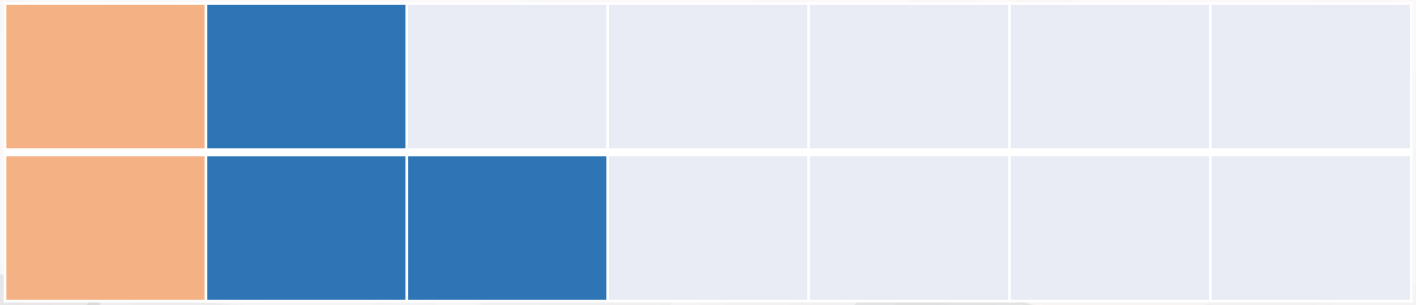
1 x 2 格填充 (強)

定義: g(N) 方法數
狀態轉移: 太難了



觀察問題

放上 L 型後 就會有一個**缺口**，
剩下**未填滿**的也是 L 形狀的
兩個 L 型**拼起來**，會變為 2 * i 格子



1 x 2 格填充 (強)

狀態：
f(N, I): 2 x N 格的方法數
f(N, L): 2 x N + **上或下凸一格**的方法數

f(N, I)

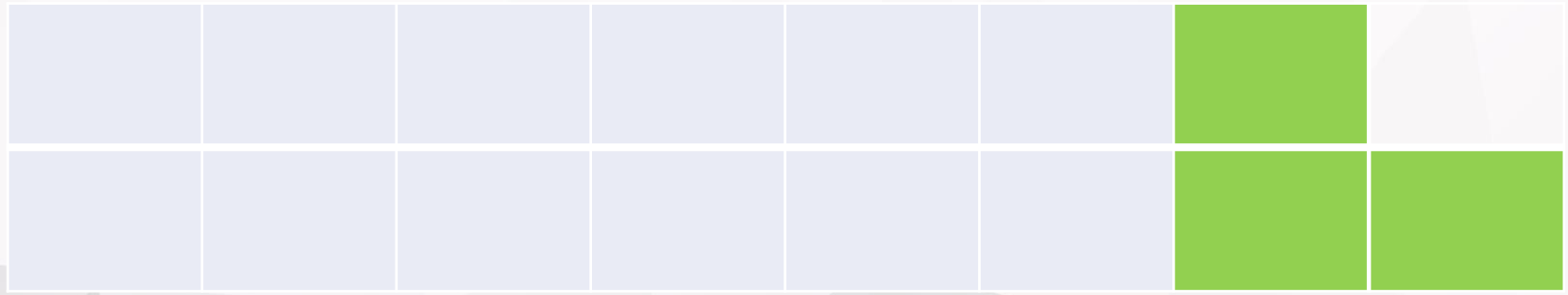
f(N, L)

1 x 2 格填充 (強)

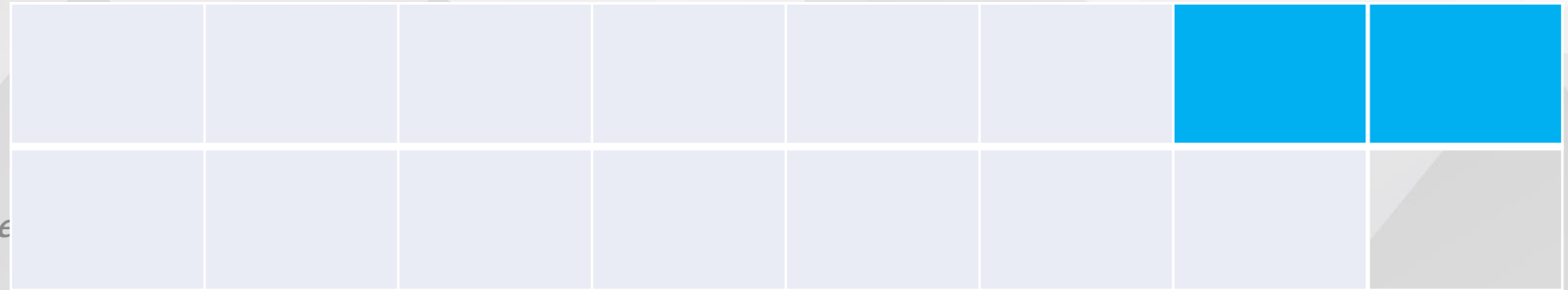
狀態轉移：

f(N, L) = f(N-1, l)*2 + f(N-1, L)

有可能會有以下兩種填法



還有另一個方向

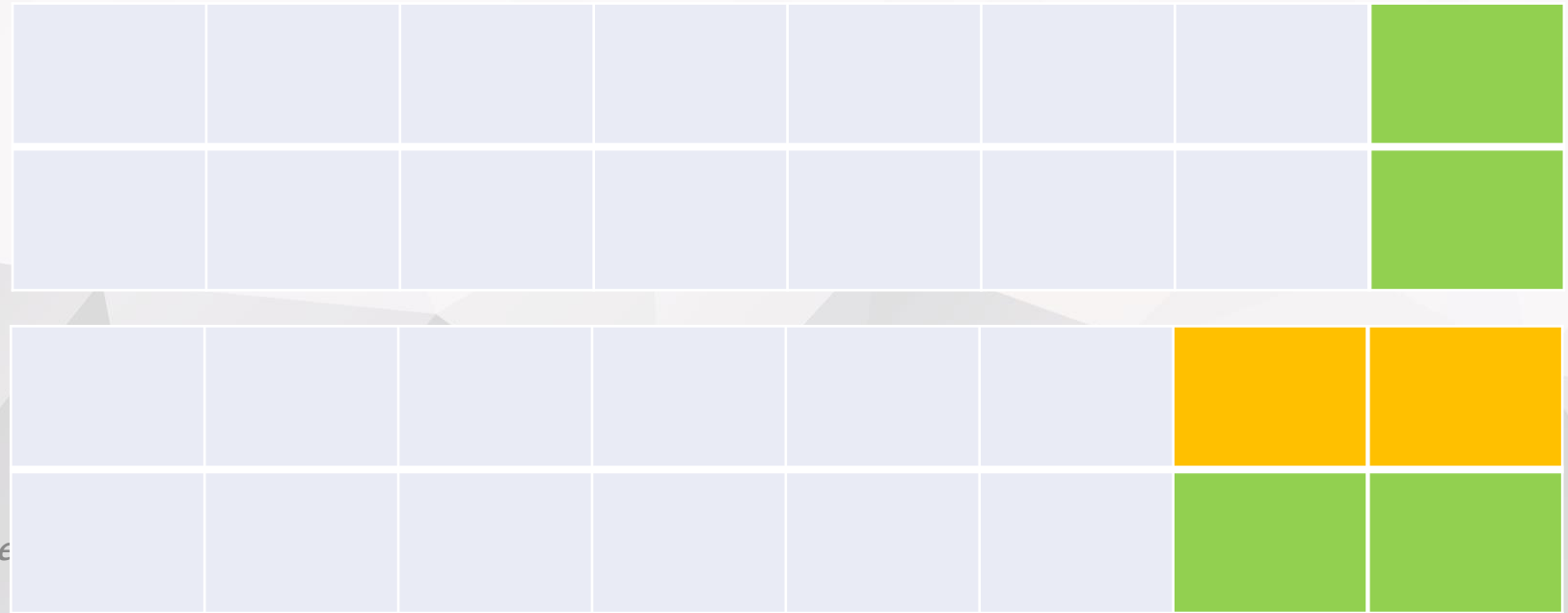


1 x 2 格填充 (強)

狀態轉移：

f(N, l) = f(N-1, l) + f(N-2, l) +

有可能會有以下兩種填法

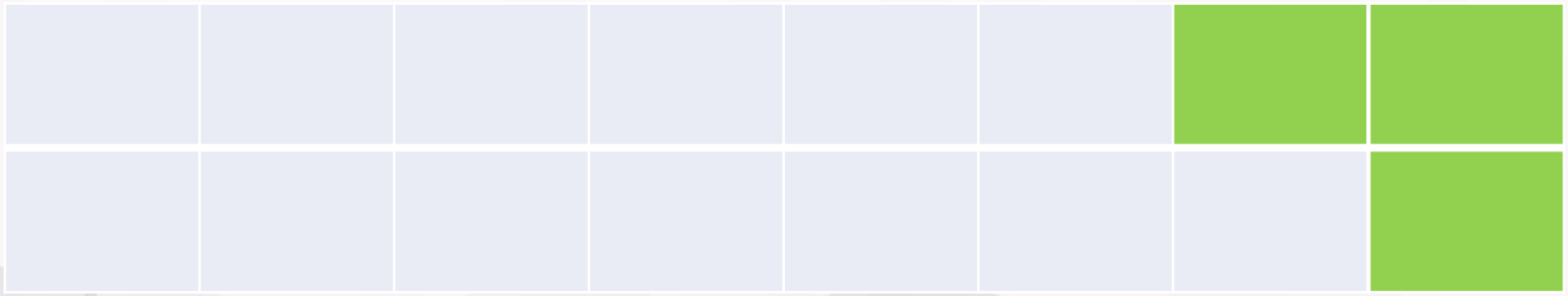


1 x 2 格填充 (強)

狀態轉移：

$f(N, l) = f(N-1, l) + f(N-2, l) + f(N-2, L)$

還有可能會有以下填法(在凸一格的情況下塞 L 型)



1 x 2 格填充 (強)

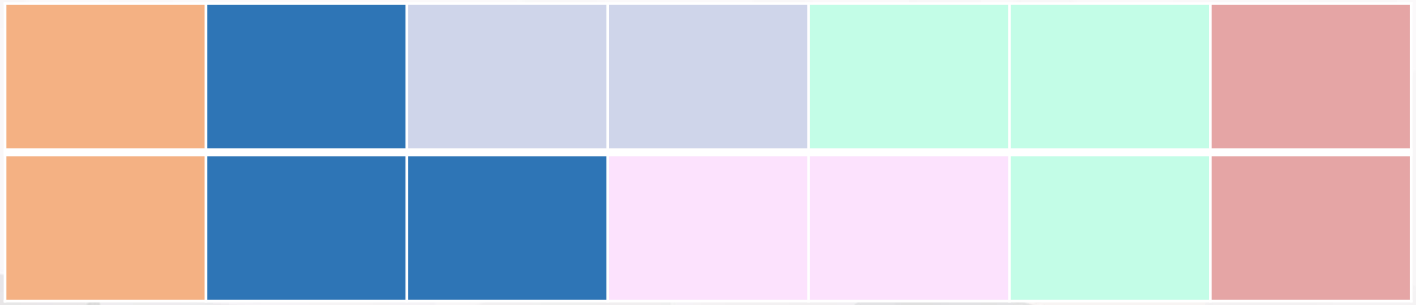
邊界：

f(1, I) = 1
f(2, I) = 2
f(1, L) = 2

另一種解法(供參考)

1 x 2 格填充 (強)

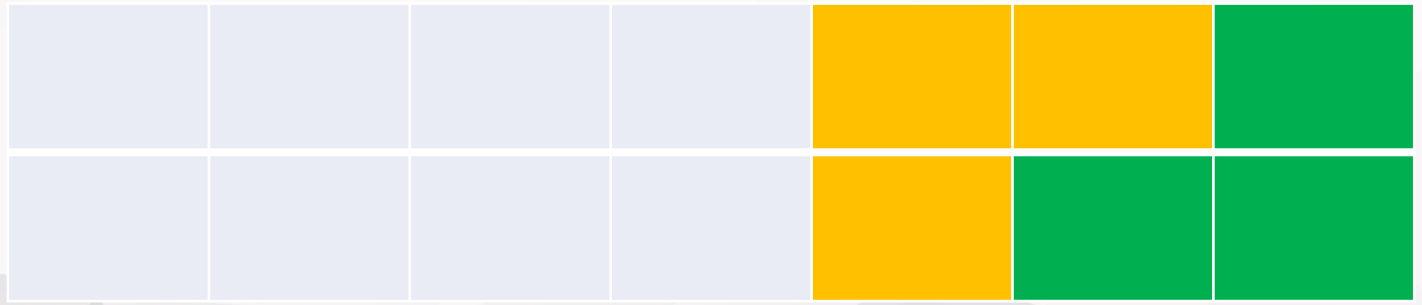
狀態: g(N) 方法數
狀態轉移: 太難了.... 才怪!



1 x 2 格填充 (強)

狀態轉移:

$$g(N) = g(N-1) + g(N-2) + 2(g(N-3) + g(N-4) + \dots + g(1))$$



g(N-3)

1 x 2 格填充 (強)

狀態轉移:

$$g(N) = g(N-1) + g(N-2) + 2(g(N-3) + g(N-4) + \dots + g(1))$$

g(N-4)

1 x 2 格填充 (強)

狀態轉移:

$$g(N) = g(N-1) + g(N-2) + 2(g(N-3) + g(N-4) + \dots + g(1))$$

$g(N-5)$

1 x 2 格填充 (強)

狀態轉移:

$$g(N) = g(N-1) + g(N-2) + 2(g(N-3) + g(N-4) + \dots + g(1))$$

這個轉移需要 O(N)

1 x 2 格填充 (強)

$g(N) = g(N-1) + g(N-2) + \underline{2(g(N-3) + \ldots + g(1))}$

利用 g(N-1) 這個狀態

$g(N-1) = g(N-2) + g(N-3) + 2(g(N-4) + \ldots + g(1))$

$g(N-1) + \mathbf{g(N-3)} - g(N-2) = \underline{2(g(N-3) + \ldots + g(1))}$

$g(N) = g(N-1) + g(N-2) + g(N-1) + g(N-3) - g(N-2)$

$= 2g(N-1) + g(N-3)$

1 x 2 格填充 (強)

$g(N) = 2g(N-1) + g(N-3)$

這個轉移只要 O(1)

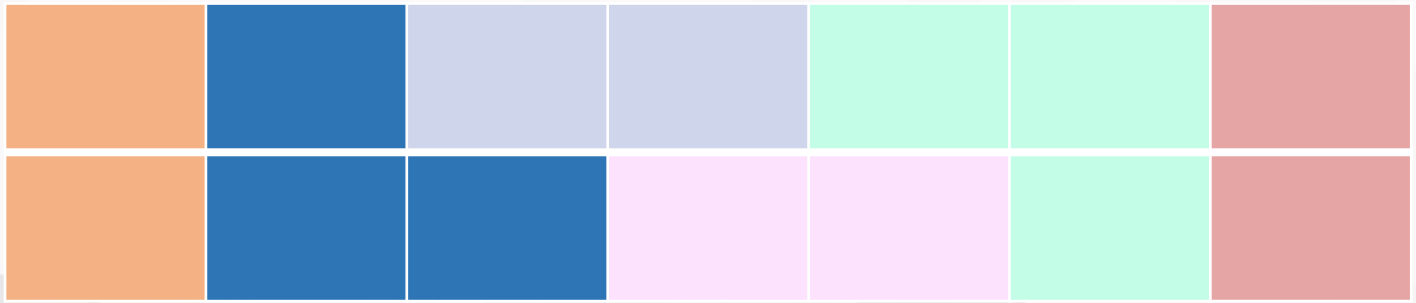
1 x 2 格填充 (強)

邊界:

$g(1) = 1$

$g(2) = 2$

$g(3) = 1 + 2 + 2$



背包問題 Knapsack problem

Knapsack problem

- 背包問題：
  給定一個固定大小的背包，
  以及各種不同大小和價值的物品，
  問如何放置物品使得背包中總價值最大。

Knapsack problem

- 聽起來很貪心?
- 來看個例子
- 假設背包容量 = 8

價值	體積
10	2
80	3
110	4
150	5
200	6

Knapsack problem

- 經典背包問題
  - 無限背包
  - 01 背包
  - 多重背包

無限背包問題

- 對於每一**種**物品，其個數是無限多個

價值	體積
10	2
80	3
110	4
150	5
200	6

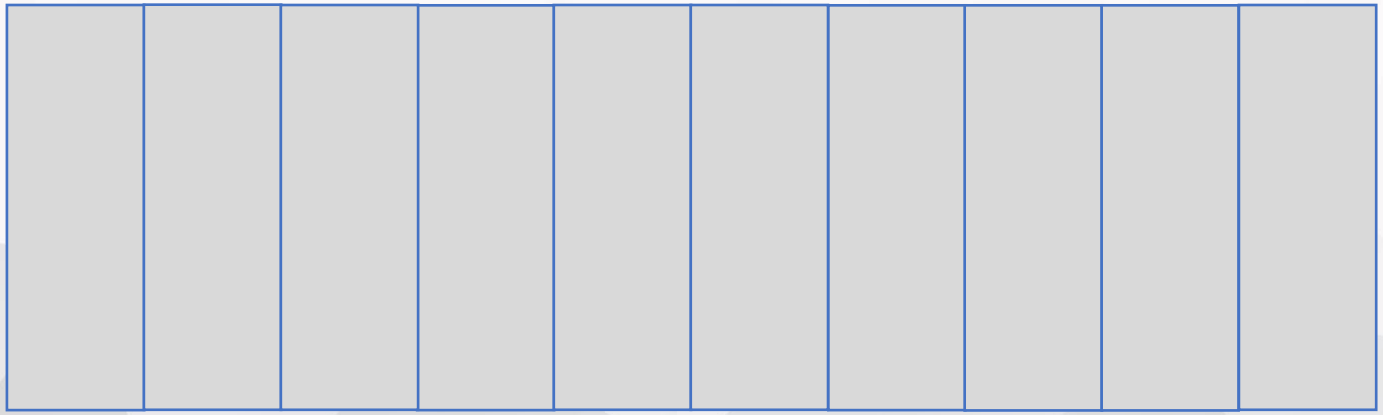
無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- 定義 P[i] : 第 i 個物品的價值
- 定義 V[i] : 第 i 個物品的體積

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- 初始化為 0

0	0	0	0	0	0	0	0	0	0

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = ?

0	0	0	0	0	0	0	0	0	0

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

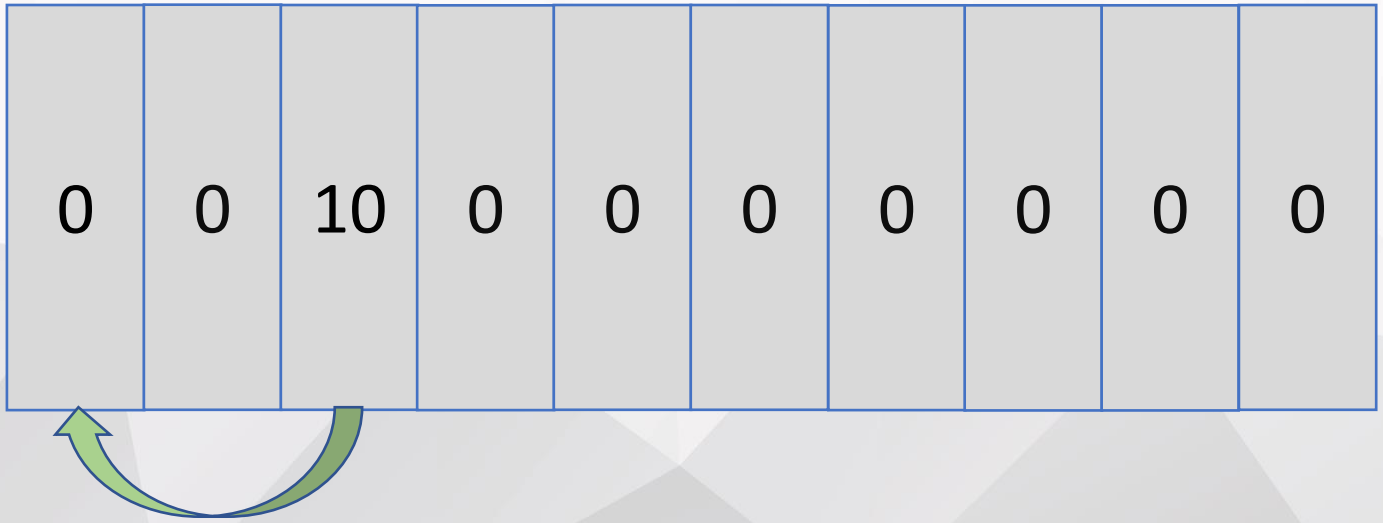
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	0	0	0	0	0	0	0	0

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

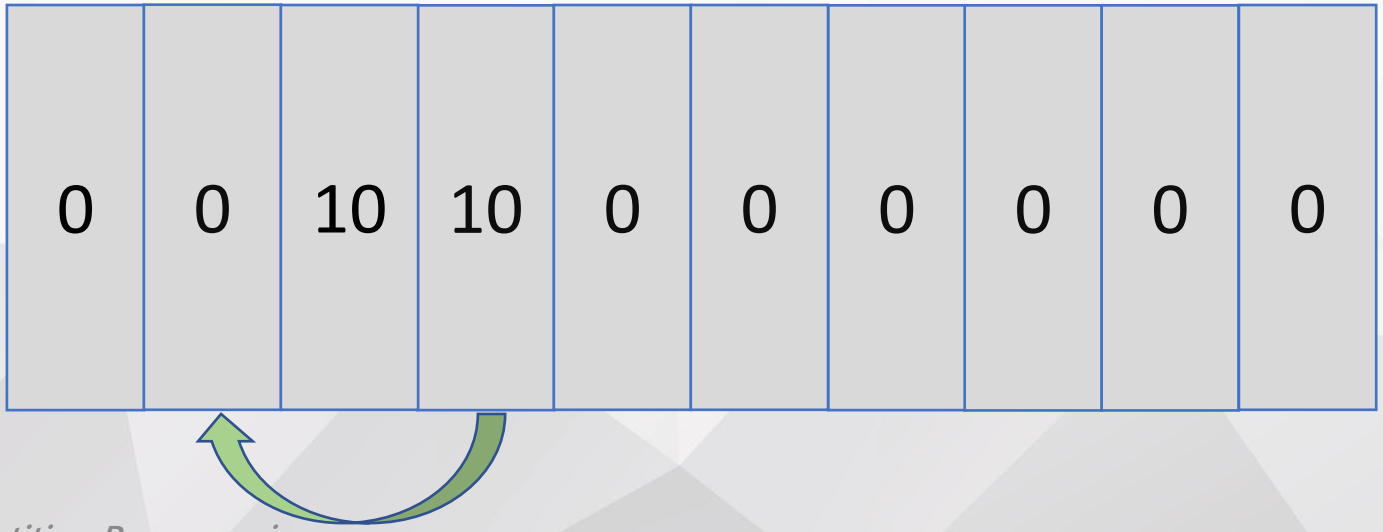
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

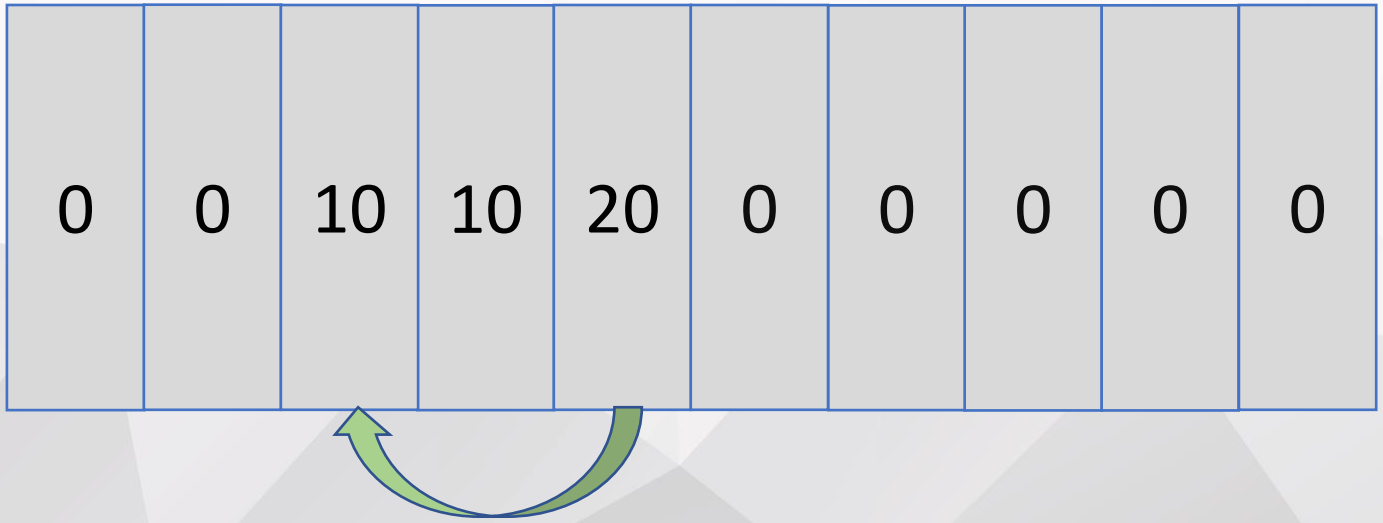
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

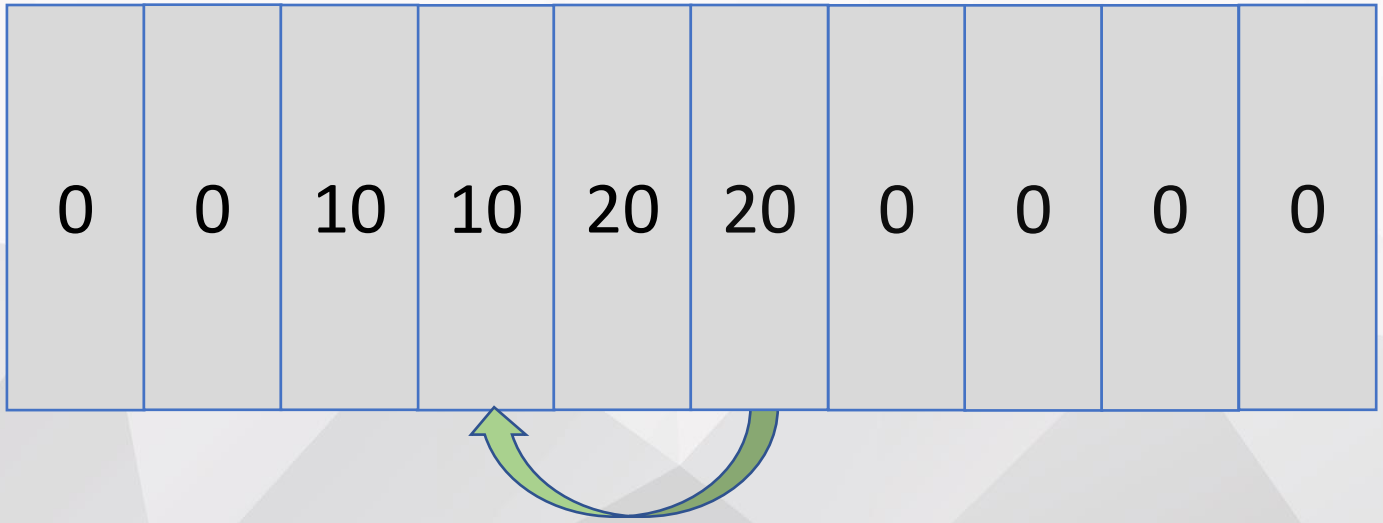
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

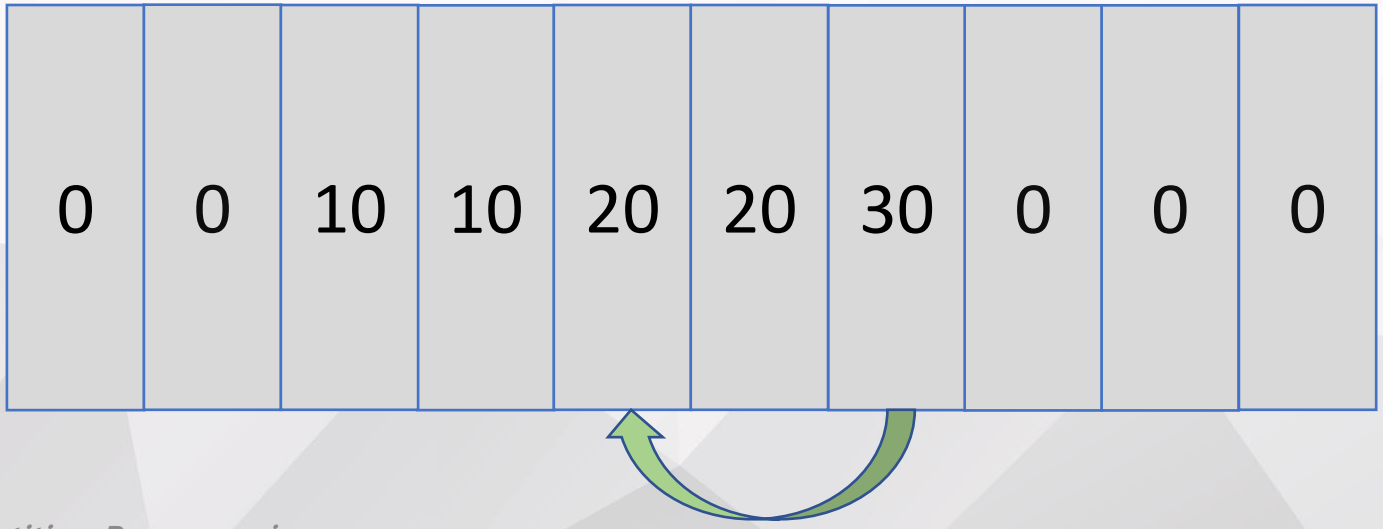
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	10	20	20	30	30	0	0

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

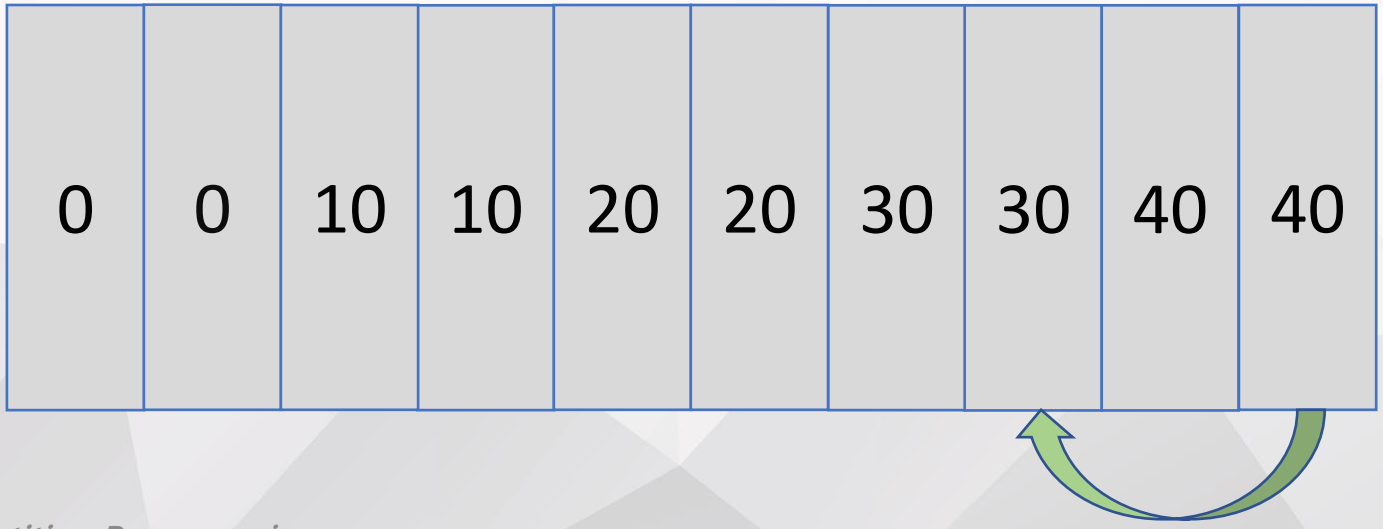
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	10	20	20	30	30	40	0

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

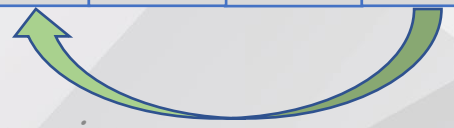
0	0	10	~~10~~ 80	20	20	30	30	40	40

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	~~20~~ 80	20	30	30	40	40



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	80	~~20~~ 90	30	30	40	40

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

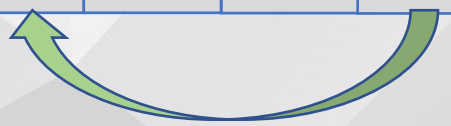
0	0	10	80	80	90	~~30~~ 160	30	40	40

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	80	90	160	~~30~~ 160	40	40



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

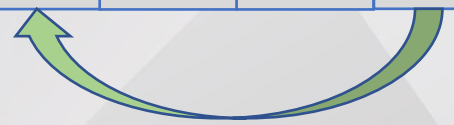
0	0	10	80	80	90	160	160	~~40~~ 170	40

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

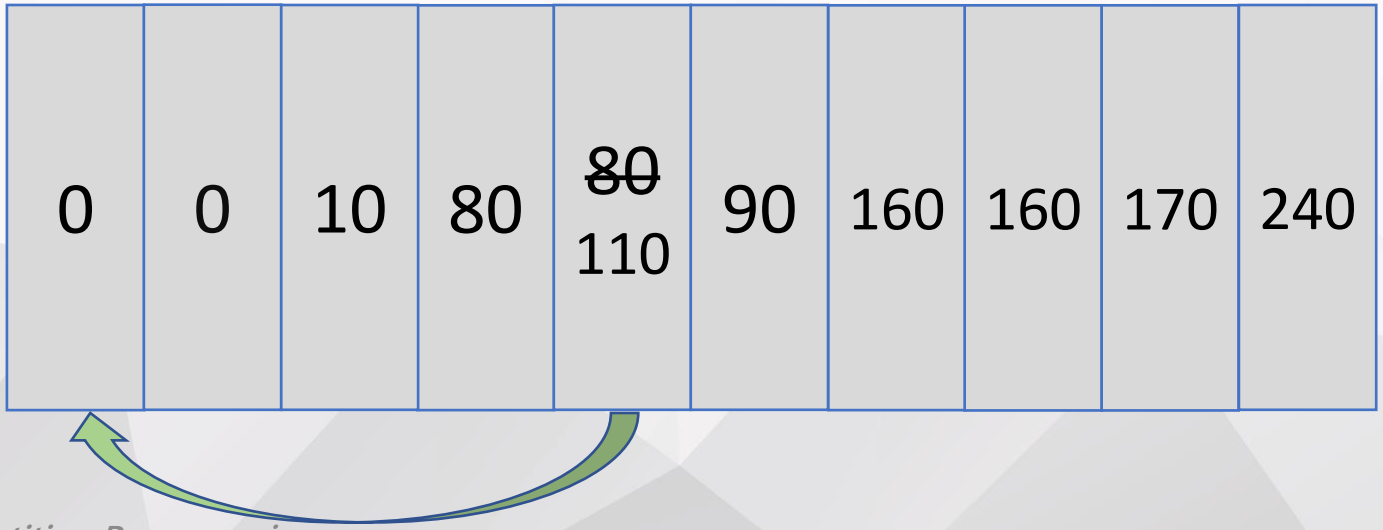
0	0	10	80	80	90	160	160	170	~~40~~ 240



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

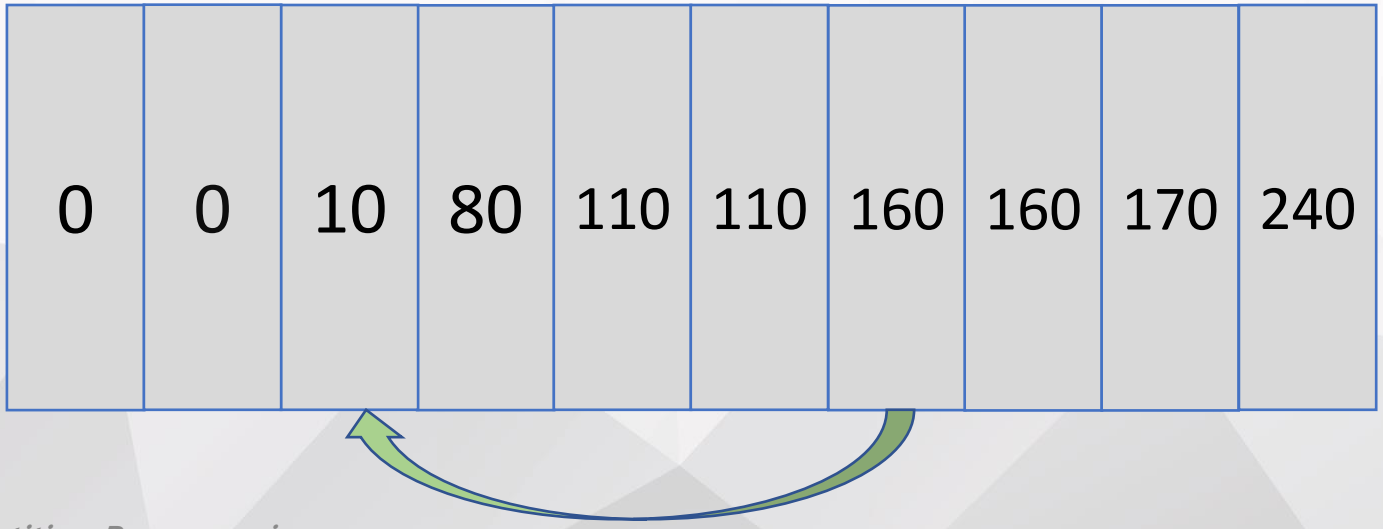
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	~~90~~ 110	160	160	170	240

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

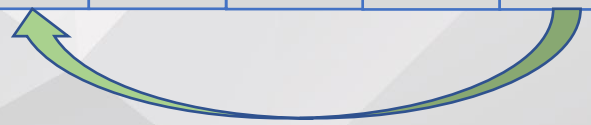
0	0	10	80	110	110	160	~~160~~ 190	170	240

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

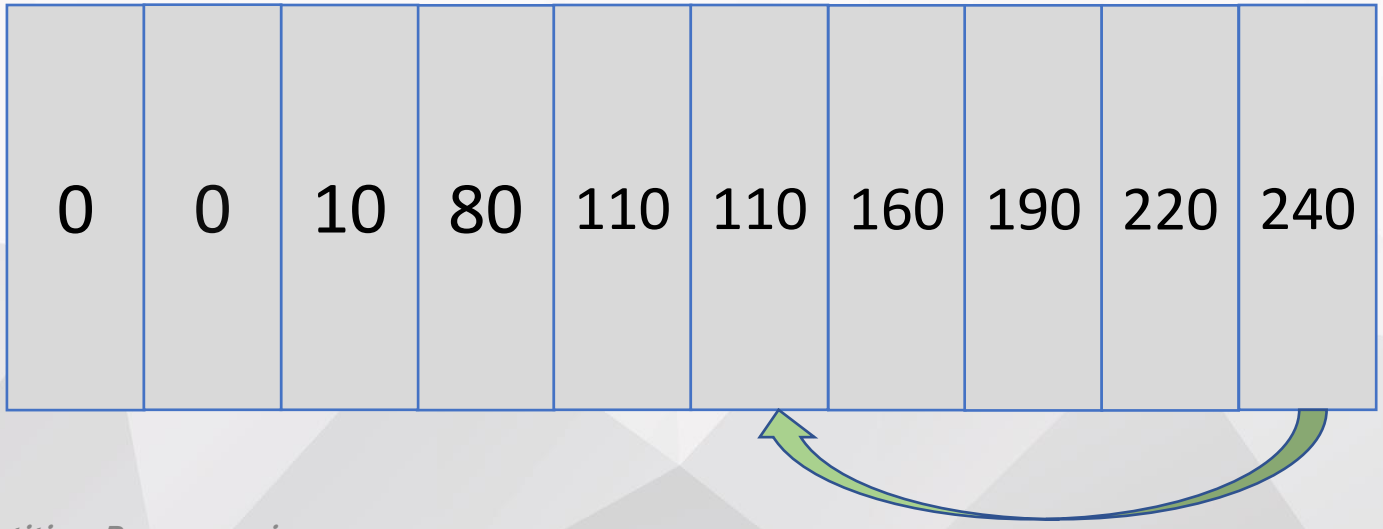
0	0	10	80	110	110	160	190	~~170~~ 220	240



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	110	160	190	220	240

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	150	160	190	230	260

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	150	160	190	230	260

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	150	200	200	230	280

價值	體積
10	2
80	3
110	4
150	5
200	6

無限背包問題

```
for (int i = 0; i < 物品數量; ++i) {
  for (int j = V[i]; j <= 背包容量; ++j) {
    S[j] = max(S[j], S[j - V[i]] + P[i]);
  }
}
```

無限背包問題

- UVa OJ 10980 Lowest Price in Town
- POJ 2063 Investment

01 背包問題

- 對於每一**種**物品，至多拿**一個**
- 做法和無限背包相似，差在順序
- 策略是由後向前更新

01 背包問題

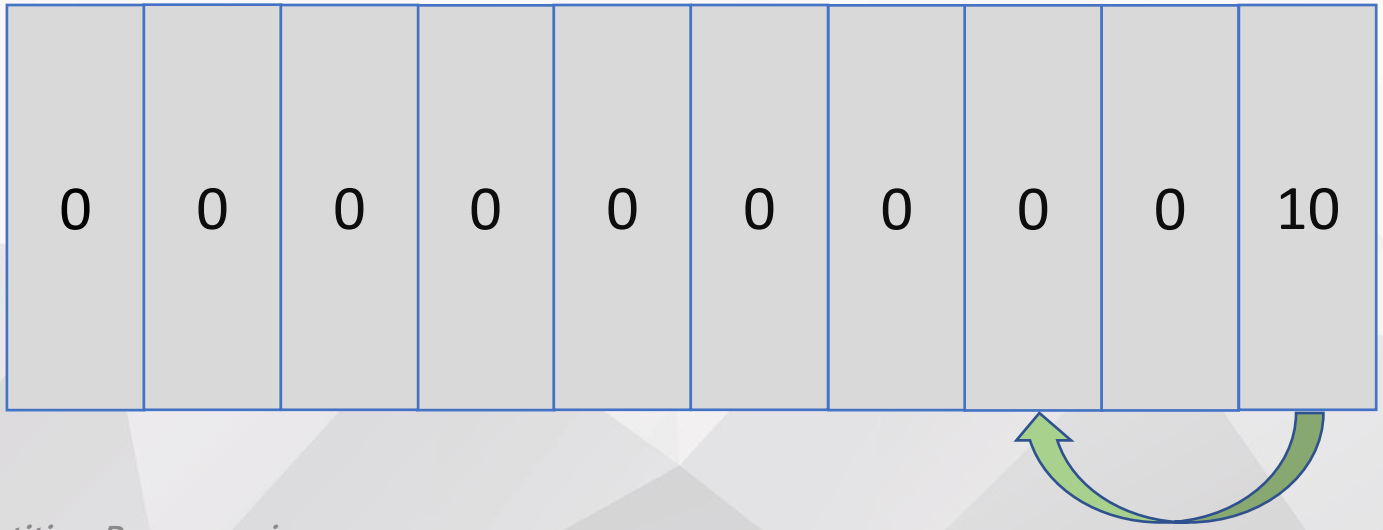
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	0	0	0	0	0	0	0	0

價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

價值	體積
10	2
80	3
110	4
150	5
200	6

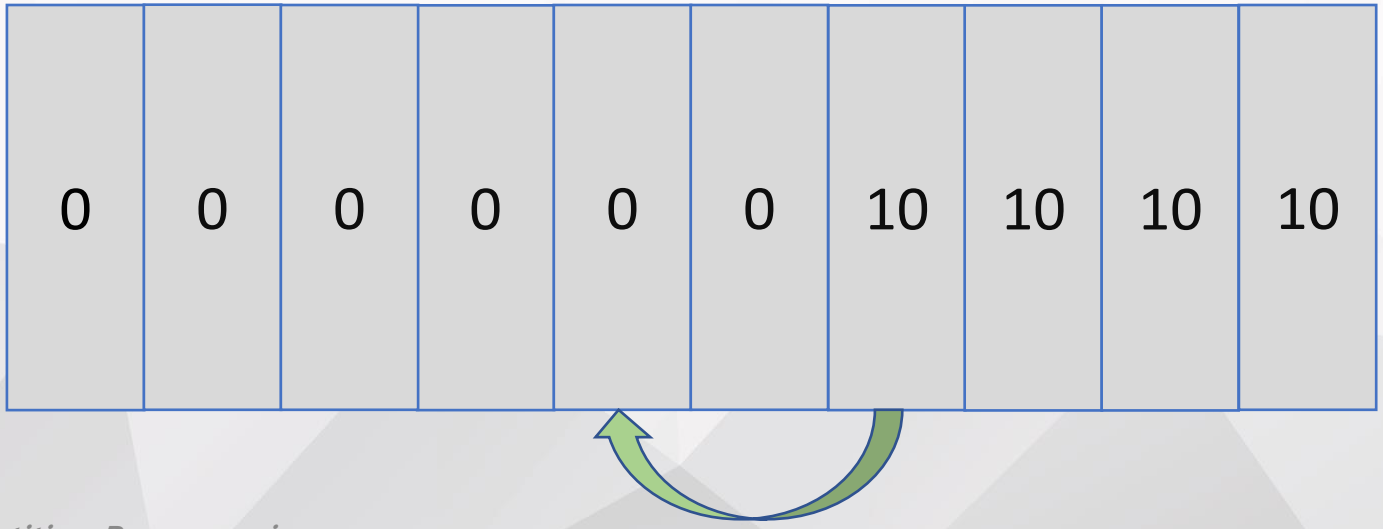
01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

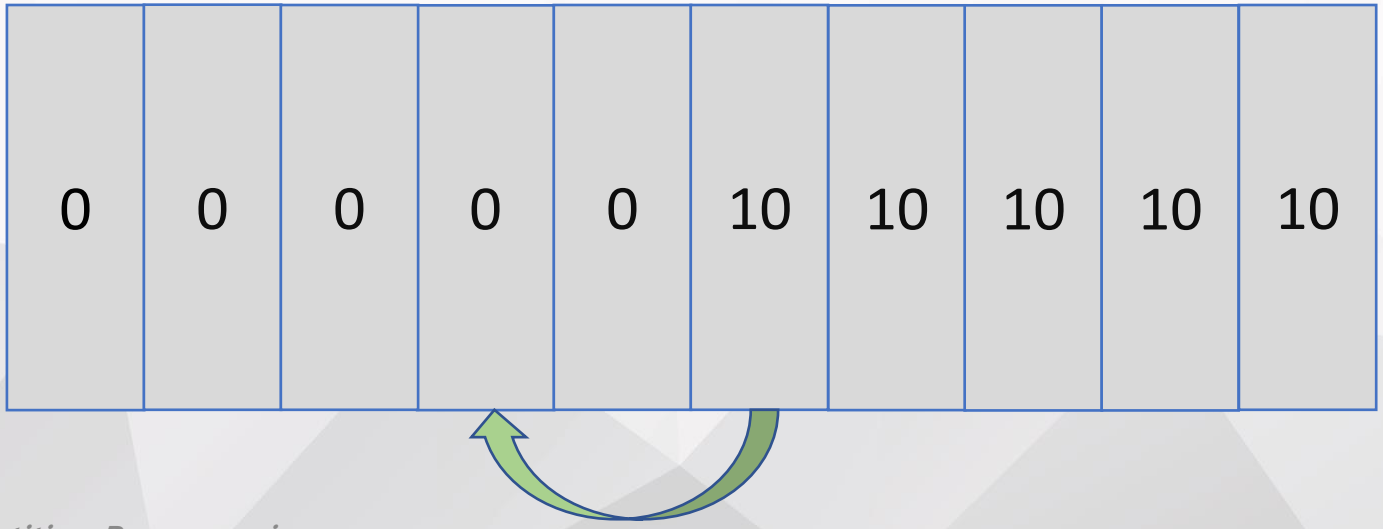
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

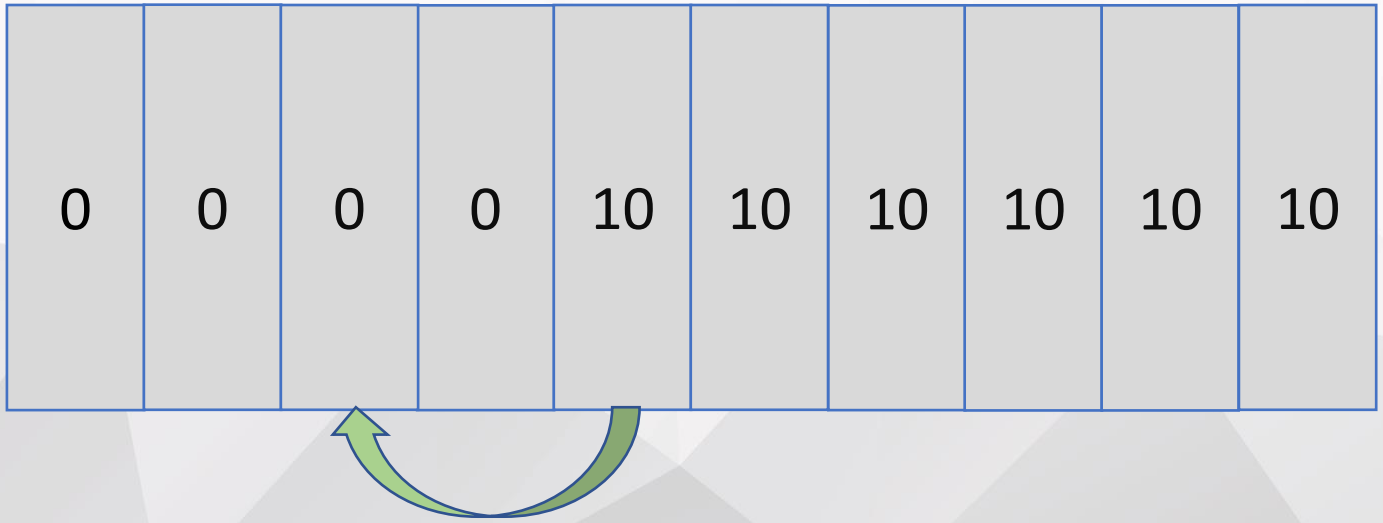
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

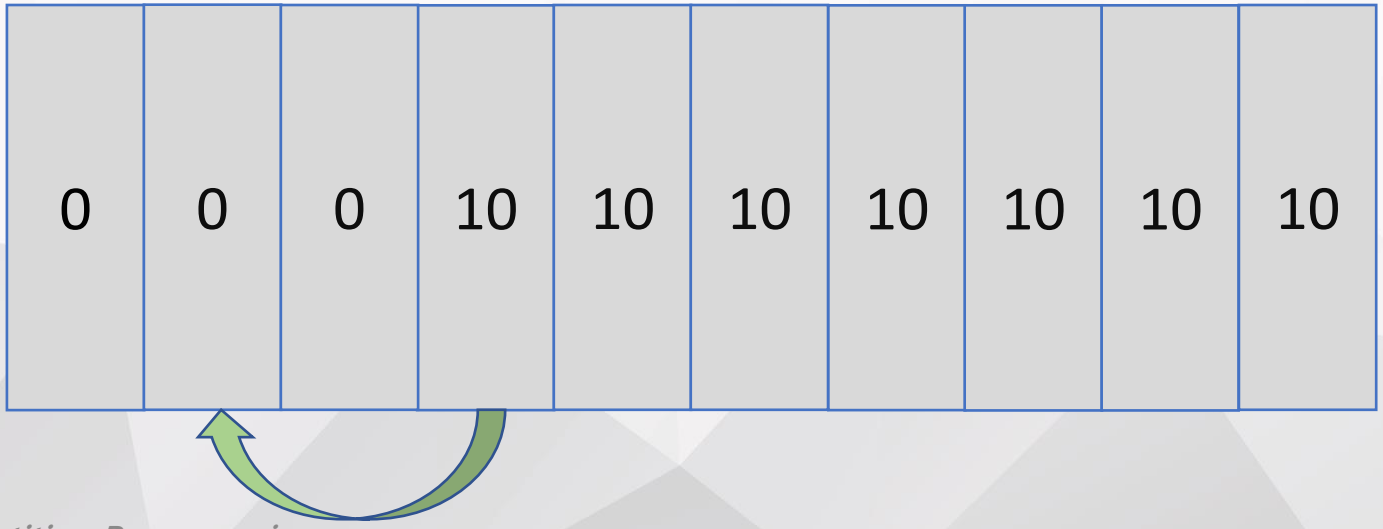
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

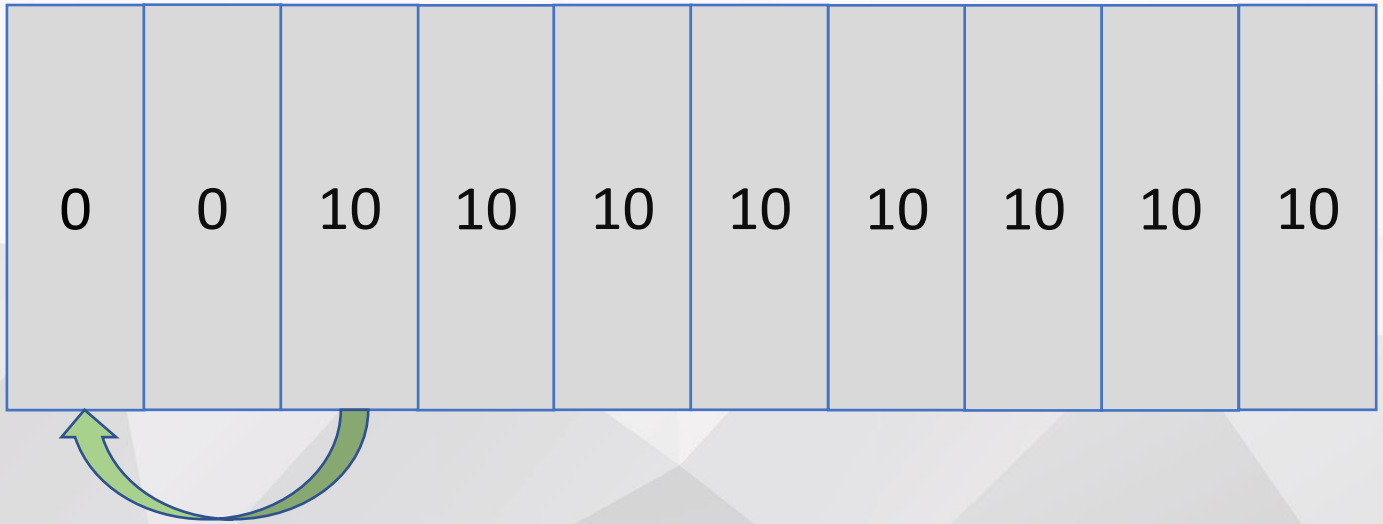
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

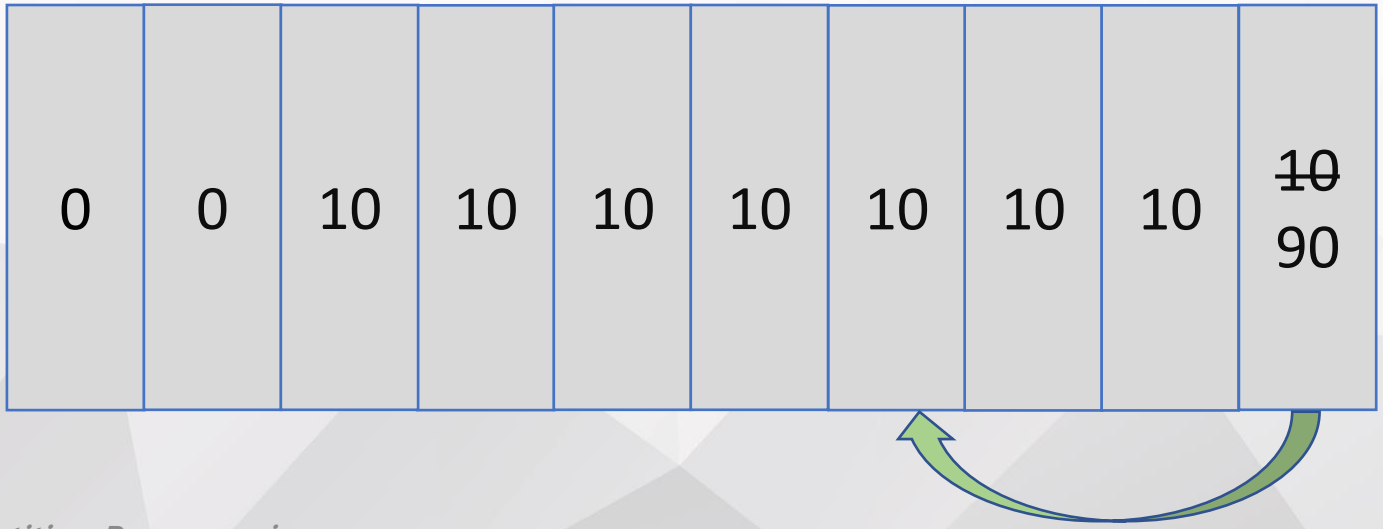
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

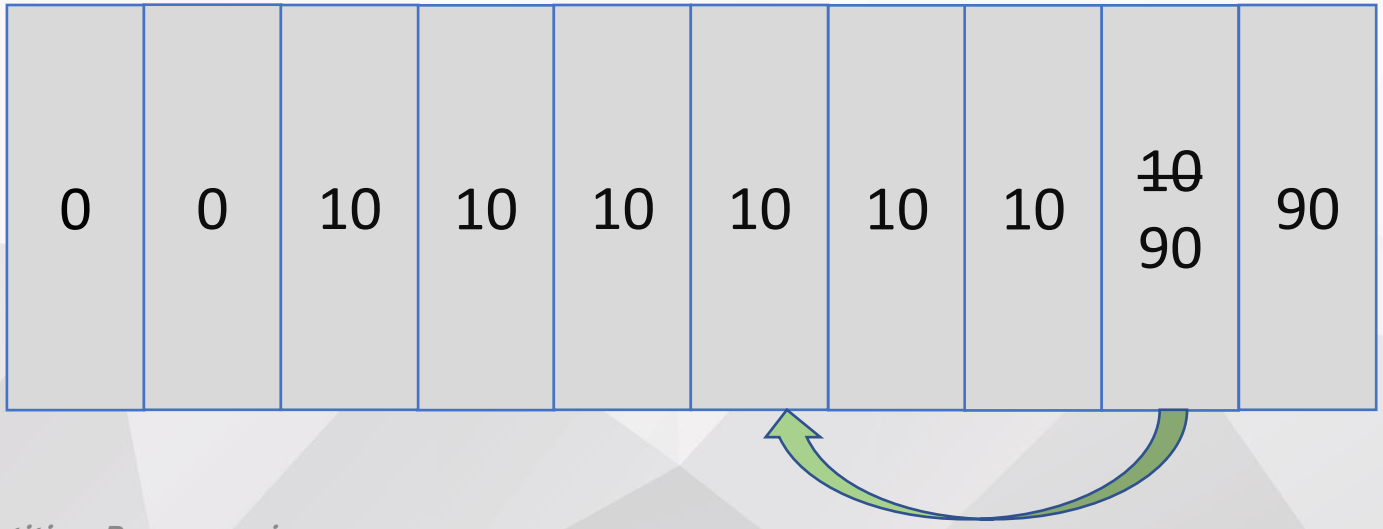
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

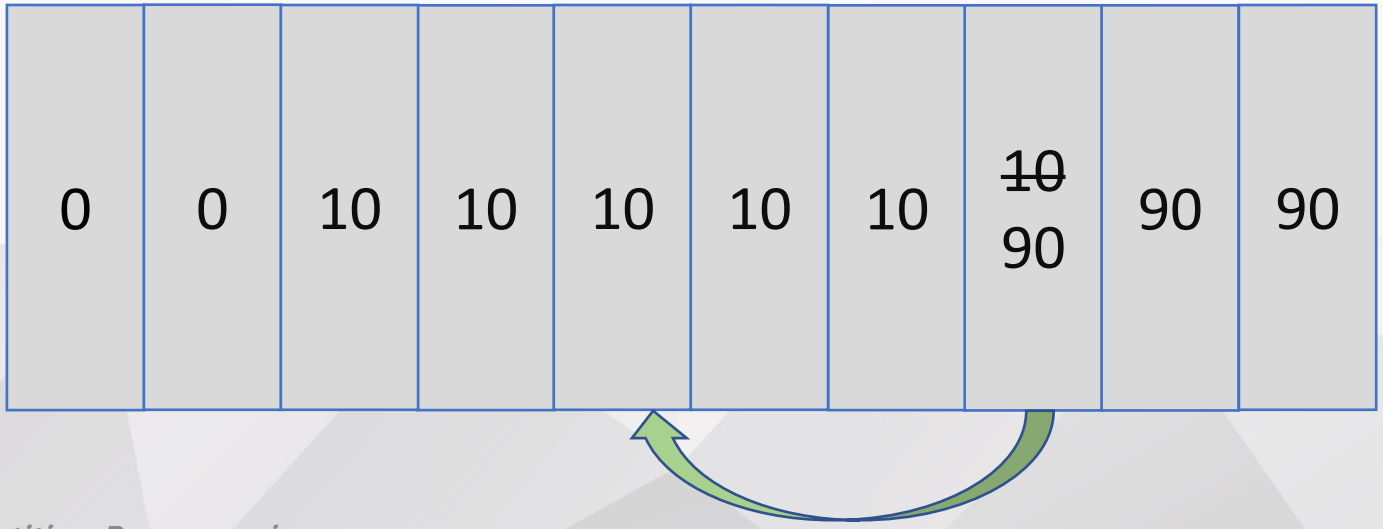
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

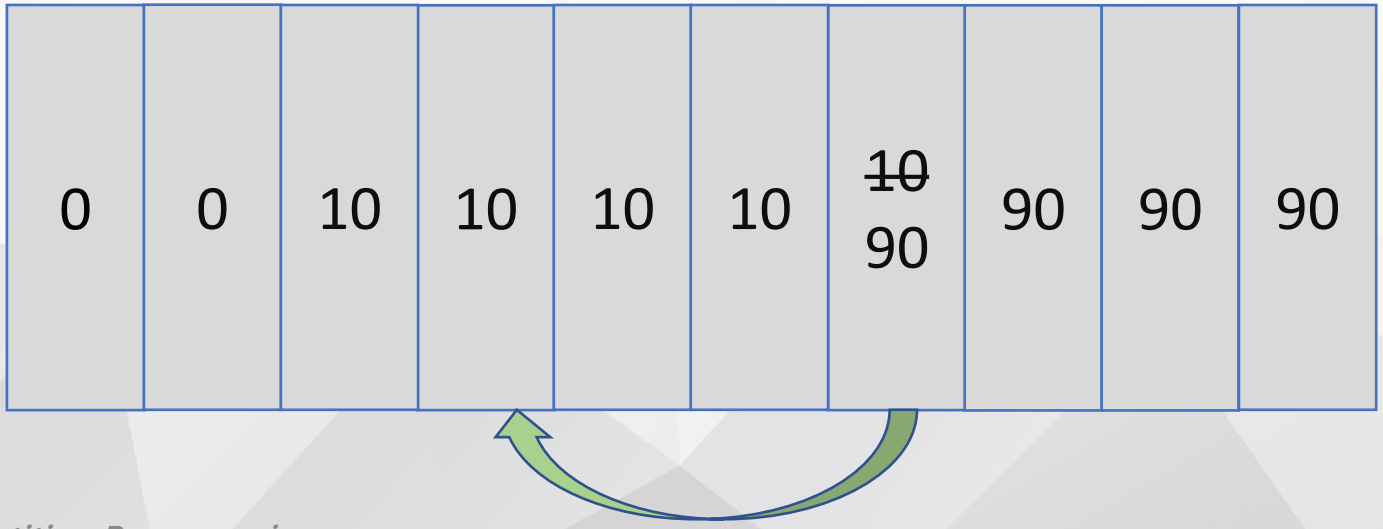
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

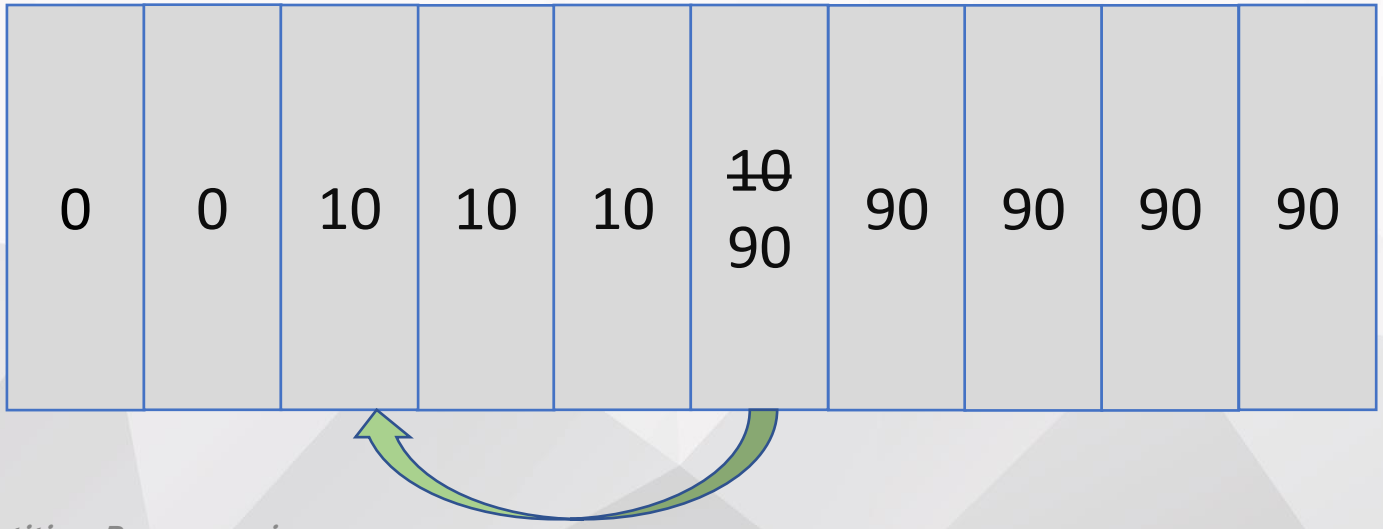
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

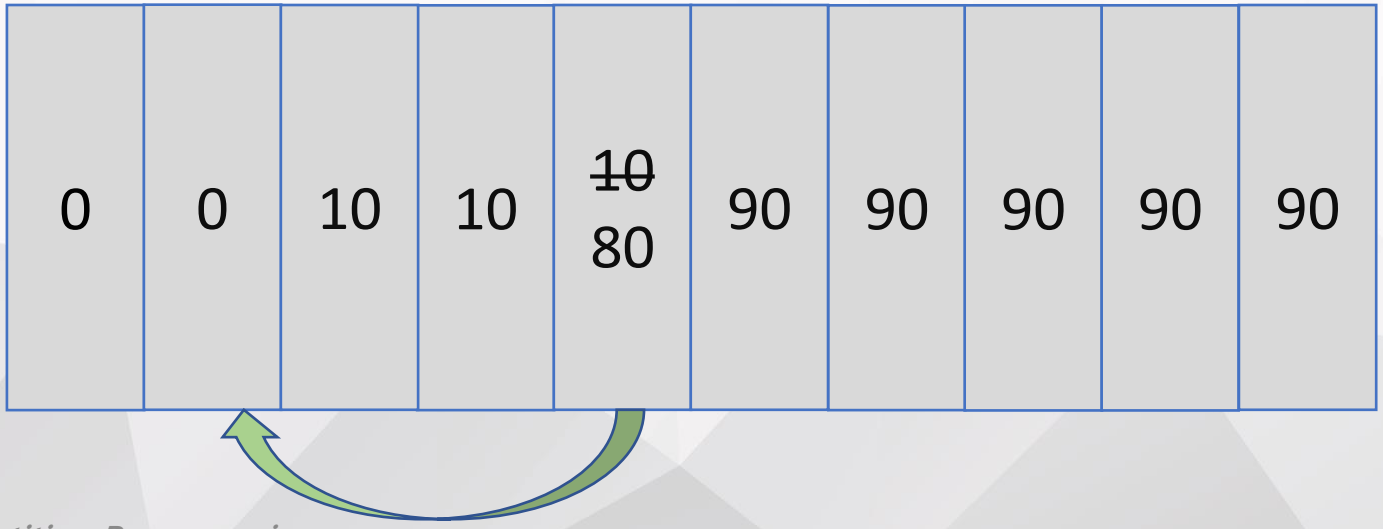
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

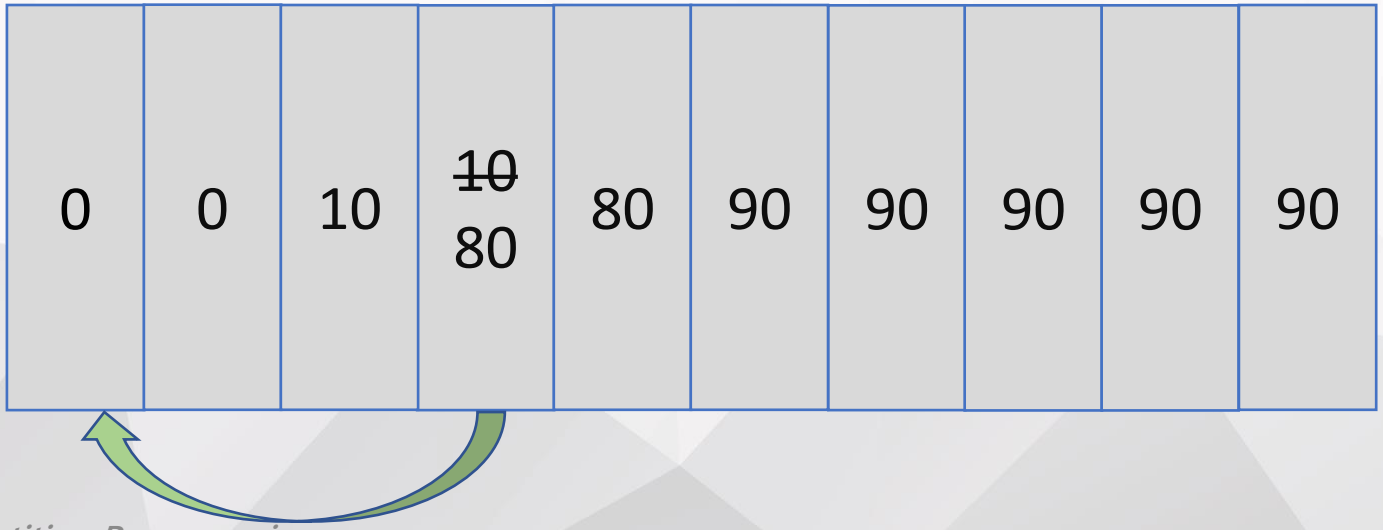
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

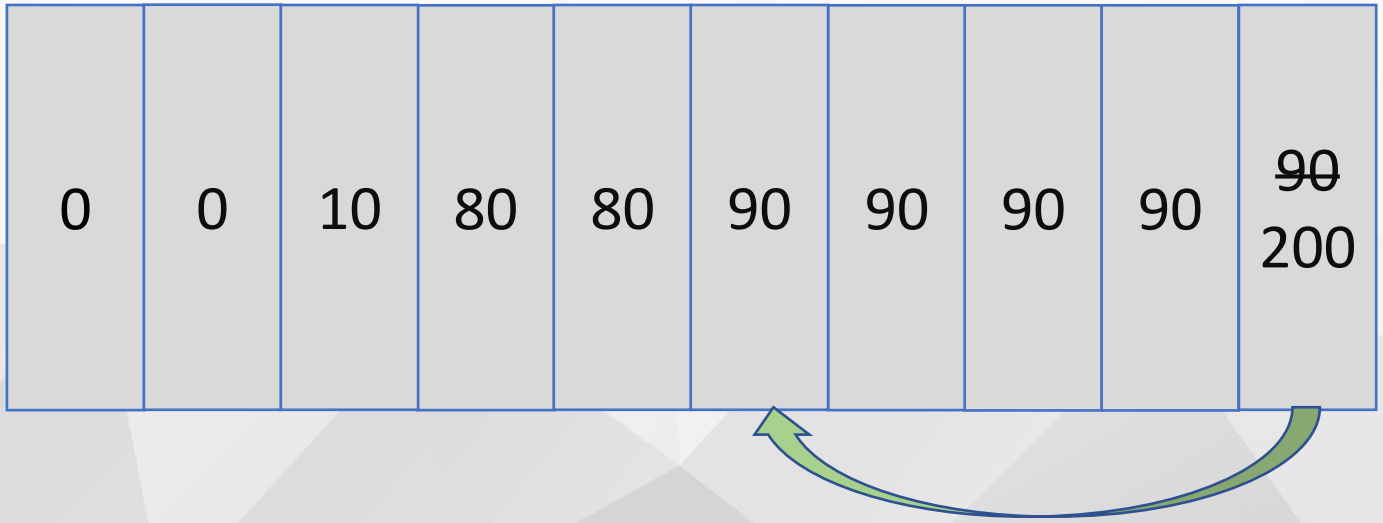
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

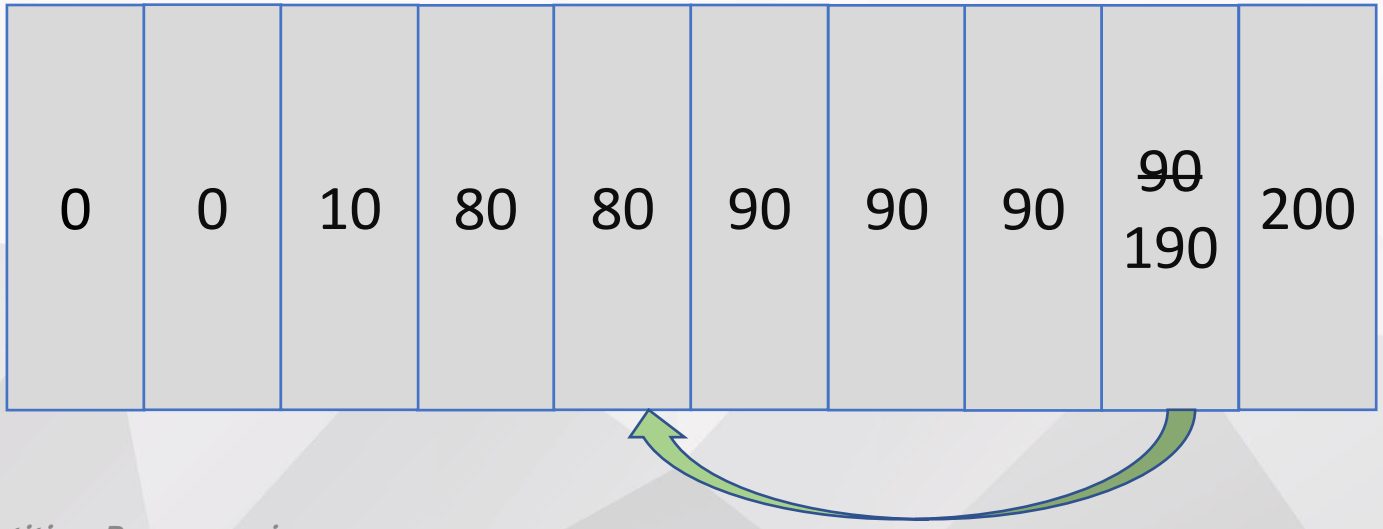
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

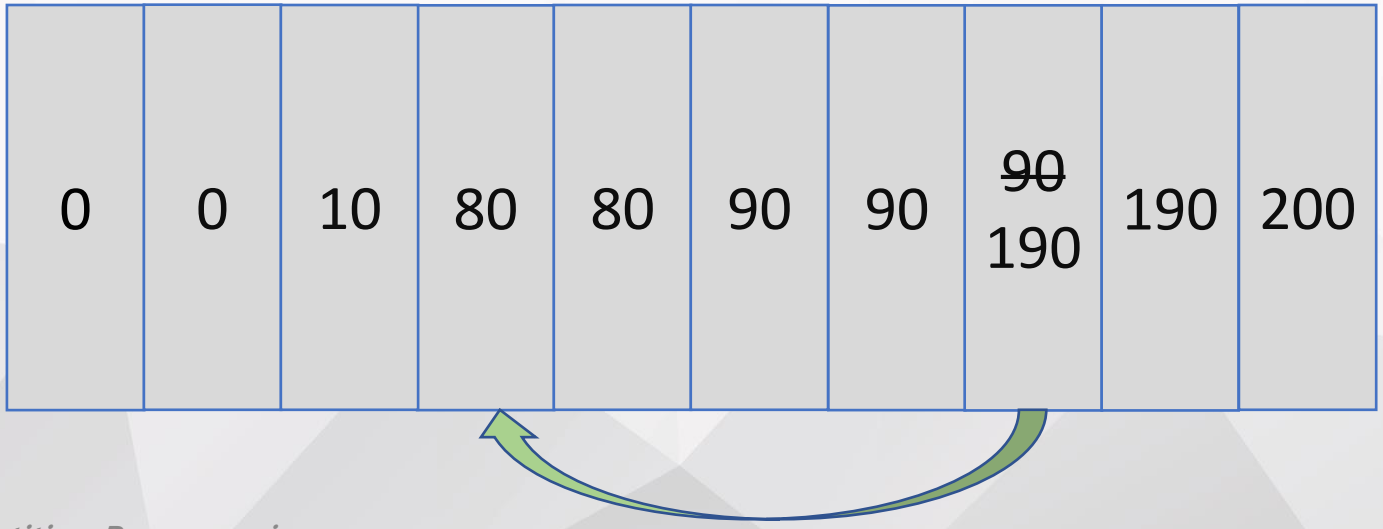
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

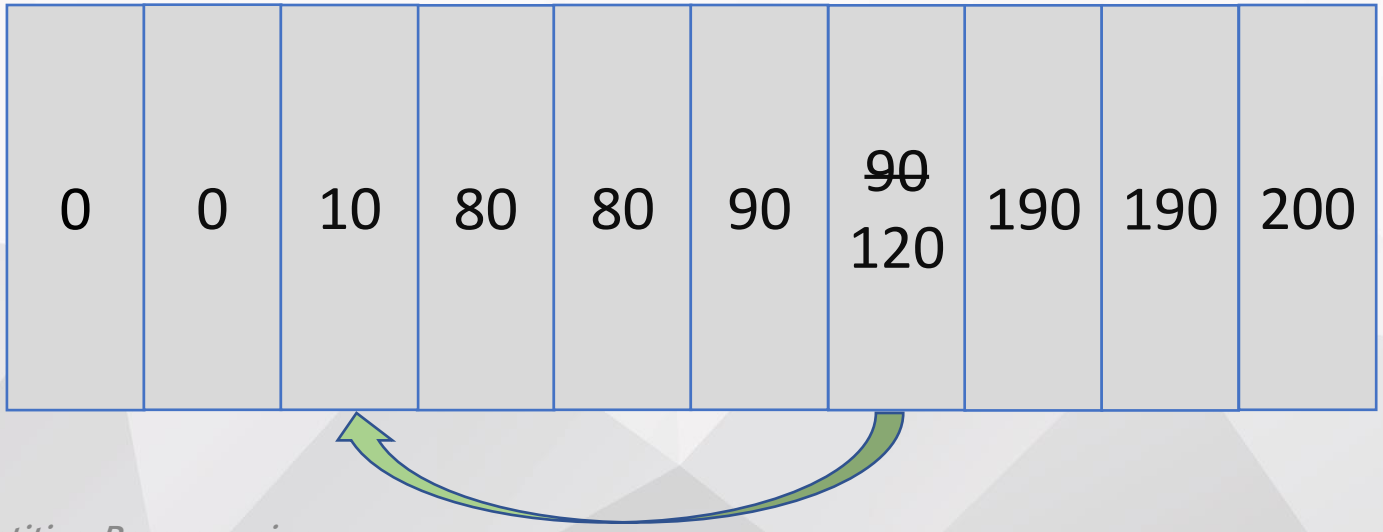
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

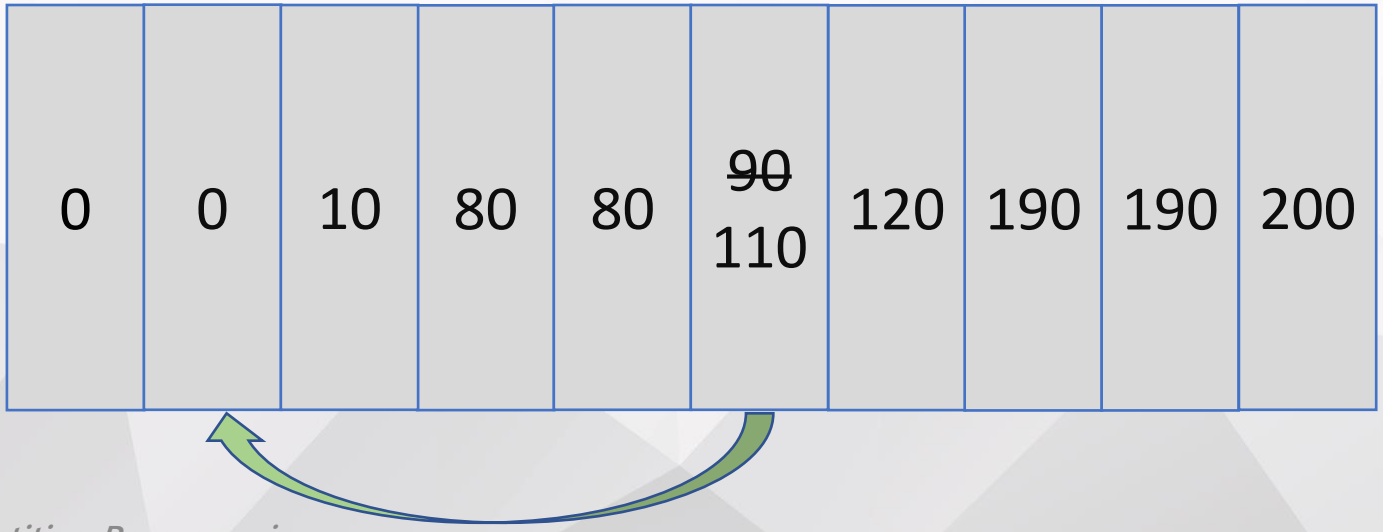
- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])



價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	~~80~~ 110	110	120	190	190	200

價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	150	150	190	230	260

價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

- 對於每一**種**物品，至多拿一個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- S[n] = max(S[n], S[n-V[i]] + P[i])

0	0	10	80	110	150	200	200	230	280

價值	體積
10	2
80	3
110	4
150	5
200	6

01 背包問題

```
for (int i = 0; i < 物品數量; ++i) {
  for (int j = 背包容量; j >= V[i]; --j) {
    S[j] = max(S[j], S[j - V[i]] + P[i]);
  }
}
```

01 背包問題

- UVa OJ 624 CD
- UVa OJ 10819 Trouble of 13-Dots

多重背包問題

- 對於每一**種**物品，其個數是**有限**多個

多重背包問題

- 對於每一**種**物品，其個數是**有限**多個
- 對該種物品，我們可以選擇 1, 2, 3, 4, ..., m 個

多重背包問題

- 轉為 01 背包問題
- 若第 i 種物品可選 m 個，則換成 m 個第 i 種物品

多重背包問題

- 利用 binary 技巧優化
- 若第 i 種物品可選 m 個，則將其換為多件物品，物品的大小與價值皆為 r 倍的原物品大小與價值
- r = { 1, 2, 4, ..., 2^{k-1}, n - (2^k -1) /*剩下的*/ } ,
  k 為滿足 n - (2^k -1) > 0 的最大整數

多重背包問題

- 舉個例子，當物品可選 13 個
- 則 k = 3, r = { 1, 2, 4, 6 }
- =>造出 4 件物品，個別包含 1, 2, 4, 6 個原物品

Questions?

DP 經典問題 LIS and LCS

Longest Increasing Subsequence

Longest Increasing Subsequence: Robinson-Schensted-Knuth Algorithm

LIS 做法很多種，可以去看之前的講義或是

http://www.csie.ntnu.edu.tw/~u91029/LongestIncreasingSubsequence.html#3

練習時間

Longest Common Subsequence

定義問題

- 在兩個序列(A, B)中
- 找到一個共同子序列
  - 並且使子序列的長度儘可能地大。

補充說明 – 子序列

- 原序列：
  - 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
- 子序列：
  - 元素前後順序性不更動
  - 可不連續元素

  Ex:

  0, 6, 14, 15

  8, 4, 12, 11, 7, 15

補充說明 – 共同子序列

- 原序列：
  - A: ATCGCCTC
  - B: TCGCATCA

- 共同子序列
  - 合法: CTC
  - 不合法: TCA (不是A的子序列)

LCS

- 定義 DP 表
- dp[i][j] 定義為
  - str A 的前 i 個字元以及
  - str B 的前 j 個字元
  - 的 LCS 長度
- dp[0][0] 也就是兩個空字串的 LCS = 0

LCS

- 例如
  - A: ATCGCCTC
  - B: TCGCATCA
  - `dp[3][2] = 2 // TC 相同`

LCS

- `if A[i] != B[j] // 字串從一開始`
- `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`

- `if A[i] == B[j]`
- `dp[i][j] = dp[i-1][j-1] + 1`

`// 建議從 i=j=1 開始做 注意邊界`

狀態轉換示意圖

		A	C	B	D	E	A	A
	0	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1	1
B	0	1	1	2	2	2	2	2
C	0	1	2	2	2	2	2	2
D	0	1	2	2	3	3	3	3
A	0	1	2	2	3	3	4	4
E	0	1	2	2	3	4	4	4

回溯示意圖

		A	C	B	D	E	A	A
	0	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1	1
B	0	1	1	2	2	2	2	2
C	0	1	2	2	2	2	2	2
D	0	1	2	2	3	3	3	3
A	0	1	2	2	3	3	4	4
E	0	1	2	2	3	4	4	4

DP 很難

- DP 應用的範圍很廣，而且很難
- 曾經有位大神說他寫了一千題才大概知道 DP 在幹嘛
- 多看 DP 的題目可以幫助你發現他是 DP 題
- https://zerojudge.tw/Problems
- 在裡面搜尋 DP 可以取得寫不完的練習題
- 挑幾題寫看看吧