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

Made by 培訓團隊群

Dynamic Programming

Outline

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

What is DP ?

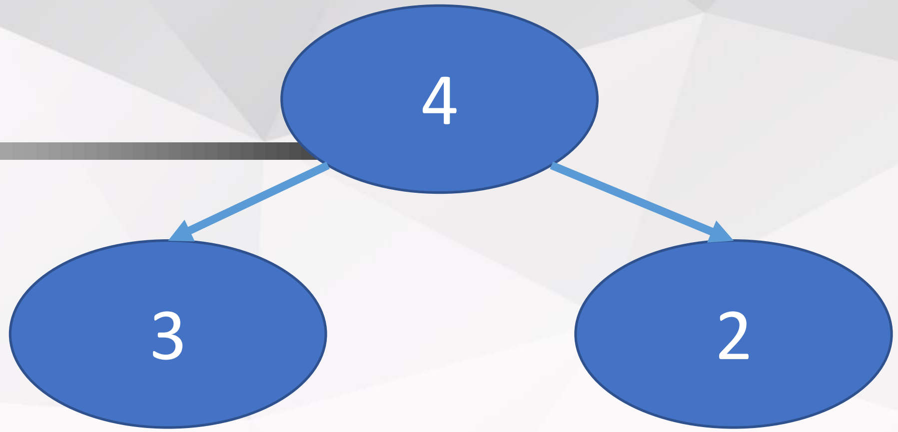
DP 是啥能吃嗎?

Intro to DP

- 計算費伯納契數列

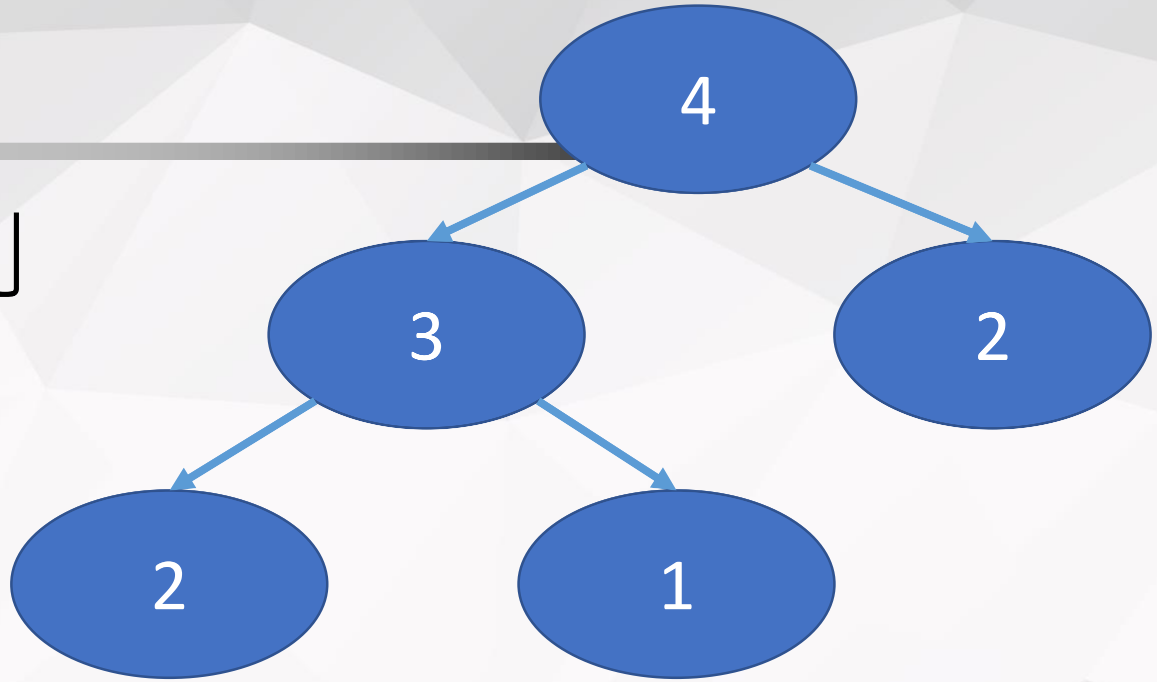
Intro to DP

- 計算費伯納契數列



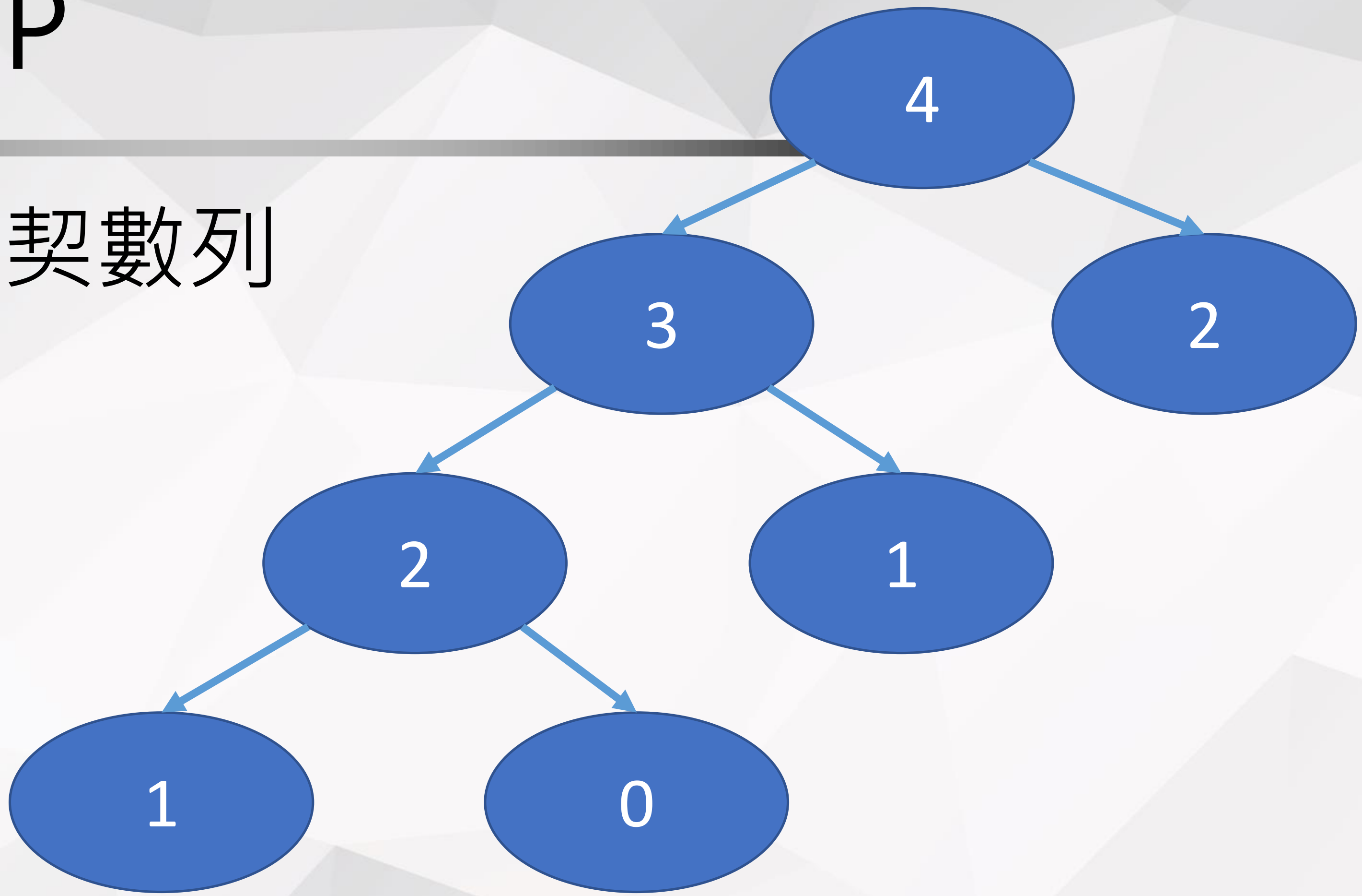
Intro to DP

- 計算費伯納契數列



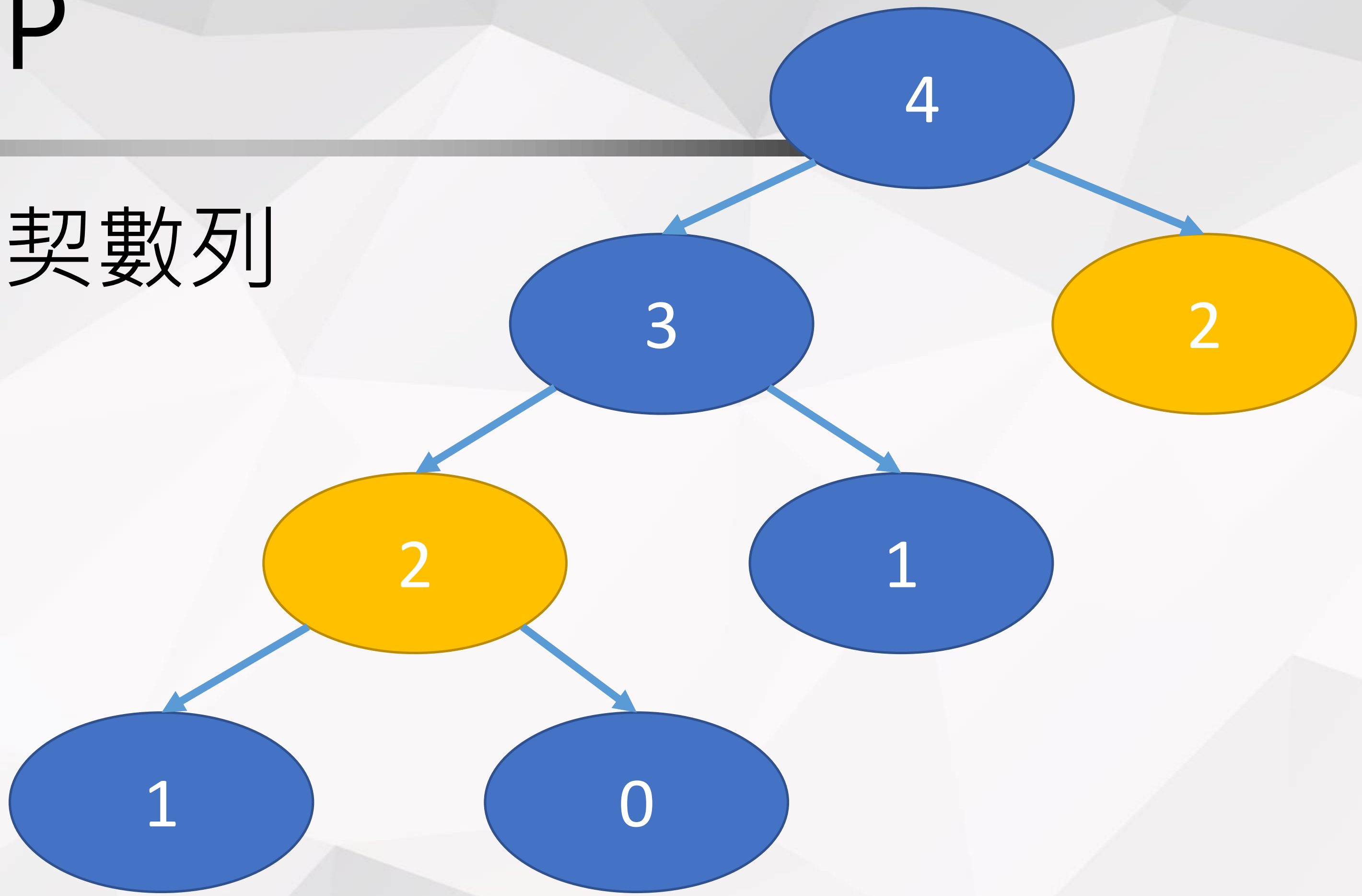
Intro to DP

- 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
  - 後無效性

最優子結構

- 問題的最優解，是子問題最優解的合併解。其子問題也具有同樣的特性

重複子問題

- 有很多子問題可歸為同樣的問題
- -> 引入記憶化

後無效性

- 確定的子問題解，並不會受到其他決策影響

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

Knapsack problem

- 經典背包問題
  - 疊積木
  - 疊積木
  - 疊積木

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

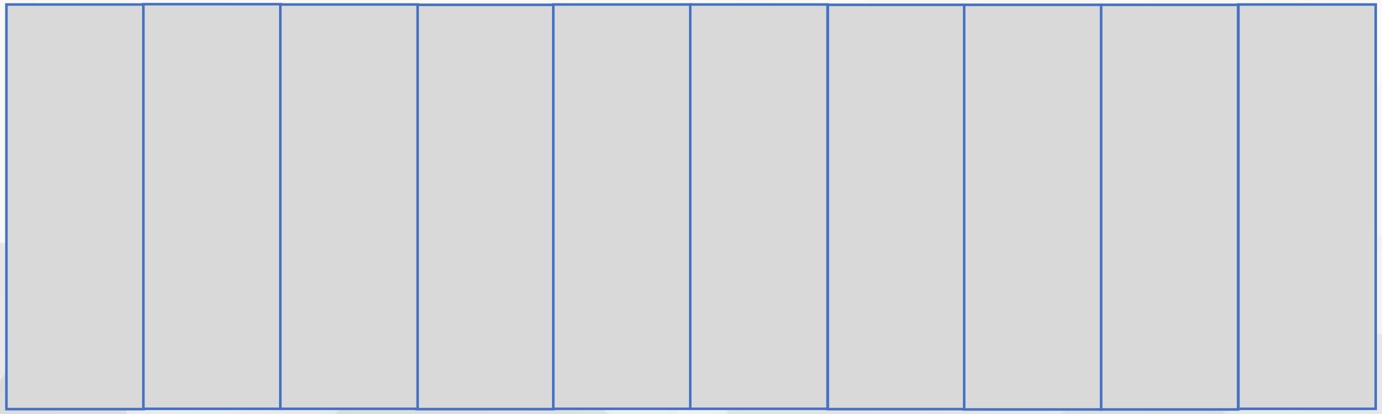
- 對於每一**種**物品，其個數是無限多個
- 定義 S[n] : 當背包的容量只有 n 時，問題的最佳解
- 定義 P[i] : 第 i 個物品的價值
- 定義 V[i] : 第 i 個物品的體積

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
- $S[n] = S[n-V[i]] + P[i]$?

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
- $S[n] = \max(S[n], S[n-V[i]] + P[i])$

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

0	0	10	0	0	0	0	0	0	0

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	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	10	10	20	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	10	10	20	20	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

0	0	10	10	20	20	30	0	0	0

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

0	0	10	10	20	20	30	30	0	0

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])

0	0	10	10	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	~~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	90	~~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	20	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	20	90	160	~~30~~ 100	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	90	160	100	~~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	20	90	160	100	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
- $S[n] = \max(S[n], S[n-V[i]] + P[i])$

0	0	10	80	~~20~~ 110	90	160	100	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	~~90~~ 110	160	100	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	100	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

0	0	10	80	110	110	160	~~100~~ 190	170	240

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
- $S[n] = \max(S[n], S[n-V[i]] + P[i])$

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	200	210	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 < C; ++i) {
  for (int j = V[i]; j <= N; ++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
- $S[n] = \max(S[n], S[n-V[i]] + P[i])$

0	0	0	0	0	0	0	0	0	10

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	0	0	0	0	0	0	10	10

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	0	0	0	0	0	10	10	10

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	0	0	0	0	10	10	10	10

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	0	0	0	10	10	10	10	10

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	0	0	10	10	10	10	10	10

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	0	10	10	10	10	10	10	10

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	10	10	10	10	10	10	10

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	10	10	10	10	10	10	~~10~~ 90

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	10	10	10	10	10	~~10~~ 90	90

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	10	10	10	10	~~10~~ 90	90	90

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	10	10	10	~~10~~ 90	90	90	90

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	10	10	~~10~~ 90	90	90	90	90

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	10	~~10~~ 80	90	90	90	90	90

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	~~10~~ 80	80	90	90	90	90	90

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	90	90	90	90	~~90~~ 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	80	90	90	90	~~90~~ 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	80	90	90	~~90~~ 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	80	90	~~90~~ 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	80	~~90~~ 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	~~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 < C; ++i) {
  for (int j = N; 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, ..., n 個

多重背包問題

- 對於每一**種**物品，其個數是**有限**多個
- 轉為 01 背包問題
- 若第 i 種物品可選 n 個，則換成 n 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 n 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
- $r = \{ 1, 2, 4, ..., 2^{k-1}, n - (2^k - 1) \}$，
  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

Competitive Programming

Made by 培訓團隊群

CCNS.

Longest Increasing Subsequence

Competitive Programming

Made by 培訓團隊群 CCNS

定義問題

- 在一數值序列中
- 找到一個子序列
  - 使得子序列元素的數值依次遞增
  - 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 –遞增子序列

- 原序列：
  - 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
- 遞增子序列：
  - 元素的數值依次遞增

    Ex:

    合法 0, 6, 14, 15

    不合法 8, 4, 12, 11, 7, 15 (雖然是子序列 但是沒有遞增)

Note: 我們先從嚴格遞增討論起。

補充說明 –最長遞增子序列

- 原序列：
  - 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
- 最長遞增子序列
  - *有沒有人要回答*

對每個元素分析找出 LIS

Longest Increasing Subsequence

LIS 要素

- 時間軸
  - 對過去紀錄
  - 對現在進行整理
  - 對未來充分準備

時間軸

- T = 1
- 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

- T = 4
- 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

直到 T = 16 結束

Note: 過去, 現在, 未來

如何記錄過去？

思考一個問題： 至少要大於 x 才可以接在 k 個數字之後。

0, 8, 4, [Now]
對於 一個數字 比 x=4 大
就可以接在 k=2個數字之後

Anime 1

影片網址：
https://youtu.be/1V881saODNs

思考一個問題： 至少要大於 x 才可以接在 k 個數字之後。

```
[0]
i =  1 ->
想要接在幾個數字之後需要多大?
Before:
After : [ 1]:  0 _
```

動畫說明

- 第一行 [x] 代表現在正在處理的數字。

```
42 6 35 50 15 39 49 7 19 29 31 [25]
i = 12 ->
想要接在幾個數字之後需要多大?
Before: [ 1]:  6 [ 2]:  7 [ 3]: 19 [ 4]: 29 [ 5]: 31
After : [ 1]:  6 [ 2]:  7 [ 3]: 19 [ 4]: 25 [ 5]: 31
```

- 在 After 裡面 [4]:25 的意思是：
  - 對於未來至少要大於25 才可以接再4個數字之後。

如何整理現在？

同樣可以讓未來接續在 k 個數字之後，那麼越小越優

說明

```
42 6 35 50 15 39 49 7 19 29 31 [25]
i = 12 ->
想要接在幾個數字之後需要多大?
Before: [ 1]:  6 [ 2]:  7 [ 3]: 19 [ 4]: 29 [ 5]: 31
After : [ 1]:  6 [ 2]:  7 [ 3]: 19 [ 4]: 25 [ 5]: 31
```

- 在處理「現在」之前
  - 「未來」至少需要大於 29 才可以接續在4個數字之後。
  - Ex: 6 15 19 29 [未來]
- 在處理「現在」之後
  - 「未來」至少需要大於 25 就可以接續在4個數字之後。
  - Ex: 6 15 19 25 [未來]

Anime 2

影片網址：
https://youtu.be/RsG37c3z4ds

```
[42]
i =  1 ->
想要接在幾個數字之後需要多大?
Before:
After : [ 1]: 42 _
```

如何對未來充滿希望？

確定未來是未來：未來是不能夠影響現在與過去

說明

- 想想看？
- 整理過去的時候需不需要未來的資料？

```
42 6 35 50 15 39 49 7 19 29 31 [25]
i = 12 ->
想要接在幾個數字之後需要多大?
Before: [ 1]:  6 [ 2]:  7 [ 3]: 19 [ 4]: 29 [ 5]: 31
After : [ 1]:  6 [ 2]:  7 [ 3]: 19 [ 4]: 25 [ 5]: 31
```

我們已經可以找出LIS的長度了！

至於往前回溯找出任意一組 LIS 那就是各位想一想的啦！

Hint! 整理「現在」要做紀錄。

練習時間 a009 All the Way North

https://judge.cp.ccns.io/problem/a009

片段程式碼

```
for(int i = 0; i < n; i++) {
      int j = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
      if(j == v.size()) v.push_back(a[i]);
      else v[j] = a[i];
  }
```

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 要素

其實思路上與 LIS 差不多，
只是這裡就直接使用術語，
而不是「過去、現在」。

- 切割小問題(記錄過去)
  - 序列A的前i個元素與序列B的前j個元素的 LCS 長度。
- 轉換狀態(處理現在)
  - 在已知「序列A的前i個元素與序列B的前j個元素的 LCS 長度」的時候
  - 在已知「序列A的前i+1個元素與序列B的前j個元素的 LCS 長度」的時候
  - 在已知「序列A的前i個元素與序列B的前j+1個元素的 LCS 長度」的時候
  - 計算「序列A的前i+1個元素與序列B的前j+1個元素的 LCS 長度」

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

Question?