
# Dynamic Programming - 2

上課補充 by howard41436

- 影片看了嗎
- Q&A


# 背包問題

- 今天要講各種背包問題
- 如果你在哪裡看過跟今天講課一樣的編排順序，那可能不是巧合

# 背包問題

- 背包問題的原形有三種，讓我們一一來複習！
- 1．0/1背包問題
- 2．無限背包問題
- 3．有限背包問題


# 1. 0/1背包問題

每個東西只有一個，取或不取


# 1. 0/1背包問題

$f(n,m)=max(f(n-1,m),f(n-1,m-c_i)+w_i)$

可以滾動陣列，或是甚至壓成一維陣列

```
for i = 1...n :
   for j = m...0 :
      f[j]=max(f[j],f(j-c[i])+w[i])
```


# 2. 無限背包問題

每個東西有無限多個，愛取幾個就取幾個

# 2. 無限背包問題

$f(n,m)=max(f(n-1,m),f(n,m-c_i)+w_i)$

可以滾動陣列，或是甚至壓成一維陣列

```
for i = 1...n :
   for j = 0...m :
      f[j]=max(f[j],f(j-c[i])+w[i])
```

# 2. 無限背包問題

這是正確的code嗎？

```
for j = 0...m :
   for i = 1...n :
      f[j]=max(f[j],f(j-c[i])+w[i])
```

# 3. 有限背包問題

每個東西有t[i]個，也就是可以不取或是取最多t[i]個

# 3. 有限背包問題

$f(n,m)=max(f(n-1,m-k*c_i)+k*w_i)$ , for $0<=k<=t[i]$

可以滾動陣列，或是甚至壓成一維陣列

```
for i = 1...n :
   for j = m...0 :
      for k = 0...t[i]:
         f[j]=max(f[j],f(j-k*c[i])+k*w[i])
```

複雜度n*m*max{t[i]}

# 物品拆分

我們可以做點變化：把第i個物品當做t[i]個不同的物品，但有一樣的c[i]和w[i]，然後做一般的背包問題。

這樣複雜度沒有變啊： sum{t[i]}*m = n*max{t[i]}*m

要組合出1~t[i]的數字，需要把t[i]拆成t[i]個1嗎？
用多少個數字可以組合出1~8所有數字呢？
是8個嗎？

# 物品拆分

將t[i]個物品拆成1個，2個，4個，...，2^h個，t[i]-2^h個
其中h是滿足2^h<=t[i]的最大整數。

如此一來t[i]個物品就可以拆成lg(t[i])種不同物品就好！然後新的背包就會有sum{lg(t[i])}個物品，每個物品選或不選，變回0/1背包問題了！

複雜度O(n*lg(maxt)*m)

# 3. 有限背包問題

$f(n,m)=max(f(n-1,m),f(n-1,m-c_i)+w_i)$

可以滾動陣列，或是甚至壓成一維陣列

```
for i = 1...n’ :
   for j = m...0 :
      f[j]=max(f[j],f(j-c[i])+w[i])
```

其實有複雜度更好的做法，下一次的影片會教到

# 4. 混合背包問題

每個東西有三種可能:
(1)只有一個
(2)有無限個
(3)有t[i]個

一樣問最大價值

# 4. 混合背包問題

一樣先對有限個數的做物品拆分(拆分完的就變只有一個)

```
for i = 1...n’ :
   if i 為只有一個的物品:
      for j = m...0 :
         f[j]=max(f[j],f(j-c[i])+w[i])
   if i 為有無限個的物品:
      for j = 0...m :
         f[j]=max(f[j],f(j-c[i])+w[i])
```


# 5. 二維背包問題

每個東西有重量(d[i])，價錢(c[i])，跟價值(w[i])
你除了要滿足c[i]<=A，還要滿足d[i]<=B
一樣問最大價值？

# 5. 二維背包問題

每個東西有重量(d[i])，價錢(c[i])，跟價值(w[i])
你除了要滿足c[i]<=A，還要滿足d[i]<=B
一樣問最大價值？

開狀態f[n][A][B]即可，轉移為
$f(n,A,B)=max(f(n-1,A,B),f(n-1,A-c_i,B-d_i)+w_i)$
實作上一模一樣！


# 6. 分組背包問題

有t組物品，每個物品跟以往一樣有價錢，價值。
但是每組物品裡面最多只能選一個。

請問最大價值？

# 6. 分組背包問題

```
for i = 1...t :
   for j = m...0 :
      for k = 1...size(i):
         f[j]=max(f[j],f(j-c[i][k])+w[i][k])
```

複雜度 sum(size(i))*m

# 6. 分組背包問題

這是正確的code嗎？

```
for i = 1...t :
   for k = 1...size(i):
      for j = j...0 :
         f[j]=max(f[j],f(j-c[i][k])+w[i][k])
```

# 7．依賴背包問題

每個物品跟以往一樣有價錢，價值。

但是每個物品i有可能依賴於另一個物品j，也就是j有選i才能選。

先假設每個物品只依賴於一個物品，且不會有物品依賴於別的物品也同時被某個物品依賴。

# 7．依賴背包問題

形成一群一群，每群有一個老大，那群中所有人都依賴他。

這群中可以都不選，也可以選老大就好，也可以選老大加上任意一個其他物品的子集合。

把這些選法的價錢，價值，當作新的物品！集中這個群中所有這種物品後就能變回分組背包問題，因為這些物品中只能選一個。

不是有2^k種選法嗎？太多了！

# 7. 依賴背包問題

每種選法的價錢都落在sum{c[i]}以內，其實最多只有sum{c[i]}種選法是有意義的。

對這個群組本身做一次背包問題！

做完後就變回一般的分組背包問題了！

# 7. 依賴背包問題 and 8.背包合併

假如去除限制:每個物品不能依賴別人又被依賴呢？

依賴的關係會變成一棵樹，每層要做的事都是分組背包問題。

也有另一個想法:

對一個子樹的root來說，每個child都是一個背包，而我們可以在V^2的時間把兩個背包合併。

# 8.背包合併

也有另一個想法:

對一個子樹的root來說，每個child都是一個背包，而我們可以在m^2的時間把兩個背包合併。

```
for i = 0...m :
   for j = 0...i :
      f[i]=max(h[j],k[i-j])
```

這跟前面的第一種做法複雜度一樣(為什麼？)


# 9．背包問題的變化

(1)求最大價值的方法總數
(2)求最大價值的一組方案
(3)求最大價值的字典序最小的一組方案
(4)求次大價值的解/第K大價值的解

# 9. 背包問題的變化

(1)求最大價值的方法總數

```
for i = 1...n :
   for j = m...0 :
      if f[j]<f[j-c[i]]+w[i] :
         f[j]=f[j-c[i]]+w[i]
         g[j]=g[j-c[i]]
      else if f[j]==f[j-c[i]]+w[i] :
         g[j]+=g[j-c[i]]
```

# 9. 背包問題的變化

(2)求最大價值的一組方案

```
for i = 1...n :
   for j = m...0 :
      if f[i][j]<f[i-1][j-c[i]]+w[i] :
         f[i][j]=f[i-1][j-c[i]]+w[i]
         g[i]=1
      else:
         g[i]=0
```

# 9. 背包問題的變化

(3)求最大價值的字典序最小一組方案

```
for i = 1...n :
   for j = m...0 :
      if f[i][j]<=f[i-1][j-c[i]]+w[i] :
         f[i][j]=f[i-1][j-c[i]]+w[i]
         g[i]=1
      else:
         g[i]=0
```

這樣真的對嗎？

# 9. 背包問題的變化

(3)求最大價值的字典序最小一組方案

```
for i = 1...n :
   for j = m...0 :
      if f[i][j]<f[i-1][j-c[i]]+w[i] :
         f[i][j]=f[i-1][j-c[i]]+w[i]
         g[i]=1
      else:
         g[i]=0
```

這樣真的對嗎？ 要把物品順序倒過來！

# 9. 背包問題的變化

(3)求最大價值的字典序最小一組方案

```
reverse(items)
for i = 1...n :
   for j = m...0 :
      if f[i][j]<=f[i-1][j-c[i]]+w[i] :
         f[i][j]=f[i-1][j-c[i]]+w[i]
         g[i]=1
      else:
         g[i]=0
```

# 9. 背包問題的變化

(4)求第K大價值

```
//f[i][j] becomes a sorted vector
for i = 1...n :
   for j = m...0 :
      for k = 0...K-1:
         vec.push_back(f[i-1][m][k])
         vec.push_back(f[i-1][m-c[i]][k]+w[i])
      sort(vec)
      f[i][j]=vec[0...K-1]
```