Dynamic Programming (2)

by music960633

Sprout

課程內容

- LIS
- LCS
- 空間優化
- 背包問題
- 換零錢問題

LIS

- Longest Increasing Subsequence (最長遞增子序列)
- 子序列：從原本序列中挑幾個數字出來，不改變前後順序而產生的序列
  - ex: arr[] = {1,5,2,4,6,3}，則 {1,6,3}、{5,2,4,3} 皆是arr的子序列
- LIS：所有遞增的子序列中，最長的子序列，以上面的例子來說，{1,5,2,4,6,3} 的LIS為 {1,2,4,6} (可能不唯一)

- 本題只需要求出最長子序列的長度即可

LIS

- 定義狀態
  - f(n)為從arr[1]到arr[n]中取出數字，且以arr[n]做結尾的最大遞增子序列長度
  - ex: arr[] = {1,5,2,4,6,3}， 則f(1~6) = {1,2,2,3,4,3}

- 狀態轉移
  - 若i<n且arr[i]<arr[n]，則可以將以arr[i]為結尾的最大遞增子序列接上arr[n]，形成一個長度為f(i)+1的子序列
  - f(n)為試過所有i之後的最大值
  - f(n)=max(f(i))+1, for all i<n and arr[i]<arr[n]

LIS

- 最後答案
  - max(f(i)), i=1 to N

- 空間複雜度(狀態數)
  - O(n)

- 時間複雜度
  - n個狀態，每個狀態轉移O(n)
  - $O(n^2)$

LIS的優化

- 已知LIS有O(n²)的演算法，是否能做到更快呢？
- f(n)=max(f(i))+1, for all i<n and arr[i]<arr[n]

- 觀察到，對於任一個位置i，若存在j使得arr[j]≦arr[i]且dp[j]≧dp[i]，則dp[i]永遠不會被用到

arr	1	3	2	5	4
dp	1	2	2	3	

LIS優化

- 因此，我們可以另外開一個陣列(tmp)，記錄「有可能被取到的arr[i]及dp[i]」
- 又，因為LIS長度每次只會加1，因此可以再簡化成只記錄arr的值即可。此時，tmp[i]=min(arr[j]), where dp[j]=i

arr	1	3	2	5	4
dp	1	2	2	3	

tmp (arr,dp)	(1,1)	(2,2)	(5,3)	

LIS優化

- 因此，我們可以另外開一個陣列(tmp)，記錄「有可能被取到的arr[i]及dp[i]」
- 又，因為LIS長度每次只會加1，因此可以再簡化成只記錄arr的值即可。此時，tmp[i]=min(arr[j]), where dp[j]=i

arr	1	3	2	5	4
dp	1	2	2	3	

tmp	1	2	5		

- 我們可以發現，tmp內的元素是嚴格遞增的！

LIS優化

- 於是，狀態轉移式可以變成：
- f(n)=i+1,

    where tmp[i] < arr[n] and tmp[i+1] ≧ arr[n]

- 由於tmp為嚴格遞增，可以用二分搜在log(n)時間內得到i值，也就是說，此狀態轉移為O(log(n))

- 於時我們得到了一個O(n*log(n))的LIS演算法！

LCS

- Longest Common Subsequence (最長共同子序列)
- ex: s1[] = “abcabc”, s2[] = “abbcad”, 則LCS為“abca”, “abbc”,...等，長度為4
- 本題只需要求出LCS的長度即可

LCS

- 定義狀態
  - f(n,m)表示s1[0~n]和s2[0~m]的LCS長度
  - 最後答案為f(N-1,M-1)

- 對於f(n,m)，考慮s1[n]和s2[m]，此時分成幾種case
  - s1[n]和s2[m]都是LCS的一部分，則因為這兩個字元都是字串的最後一個字元，s1[n]=s2[m]，剩下的LCS則在s1[0~n-1]和s2[0~m-1]。因此，f(n,m) = f(n-1,m-1) + 1
  - 只有s1[n]是LCS的一部分，則表示s2[m]沒有貢獻，f(n,m)=f(n,m-1)
  - 只有s2[m]是LCS的一部分，則f(n,m)=f(n-1,m)
  - s1[n]和s2[m]都不是LCS的一部分，則f(n,m)=f(n-1,m-1)

LCS

- 定義狀態
  - f(n,m)表示s1[0~n]和s2[0~m]的LCS長度
  - 最後答案為f(N-1,M-1)

- 狀態轉移
  - f(n,m)=max(f(n-1,m-1)+1,f(n-1,m),f(n,m-1),f(n-1,m-1)), if s1[n]=s2[m]
  - f(n,m)=f(n-1,m-1)+1, if s1[n]=s2[m]
  - f(n,m)=max(f(n-1,m),f(n,m-1),f(n-1,m-1)), if s1[n]!=s2[m]
  - f(n,m)=max(f(n-1,m),f(n,m-1)), if s1[n]!=s2[m]

LCS

- 定義狀態
  - f(n,m)表示s1[0~n]和s2[0~m]的LCS長度
  - 最後答案為f(N-1,M-1)

- 狀態轉移
  - f(n,m)=f(n-1,m-1)+1, if s1[n]=s2[m]
  - f(n,m)=max(f(n-1,m),f(n,m-1)), if s1[n]!=s2[m]
  - O(1)轉移

- 時間複雜度
  - 狀態$O(n^2)$*轉移$O(1)=O(n^2)$

Sprout

LCS

- 用 LIS 做 LCS
- ex: s1[] = “abc”, s2[] = “bac”
- 記錄每個字元在s1出現的位置
  - a: 0
  - b: 1
  - c: 2
- 接著將s2的字元換成上一步中對應的數字
  - “bac” => {1,0,2}
- 然後對這個序列做LIS，結果就是原本兩個字串的LCS
  - LIS = {1,2} or {0,2}
  - 對應到 “bc” or “ac”

LCS

- 用 LIS 做 LCS
- 如果一個字元在s1有重複出現呢？
- ex: s1[] = “abcabc”, s2[] = “abbcad”
- 記錄出現位置
  - a: 0,3
  - b: 1,4
  - c: 2,5
- 由大到小排列
  - 避免相同的字元被算兩次
  - {3,0,4,1,4,1,5,2,3,0}
  - LIS: {0,1,2,3}，長度為4

空間優化

- 給一個N*M的矩形，每格內有一個數字。由左上走到右下，且只能往右走和往下走的路徑中，總和最大為多少？

- 定義狀態
  - f(i,j)為走到點(i,j)時，路徑的最大值

- 狀態轉移
  - f(i,j)=max(f(i-1,j),f(i,j-1))+a[i][j]

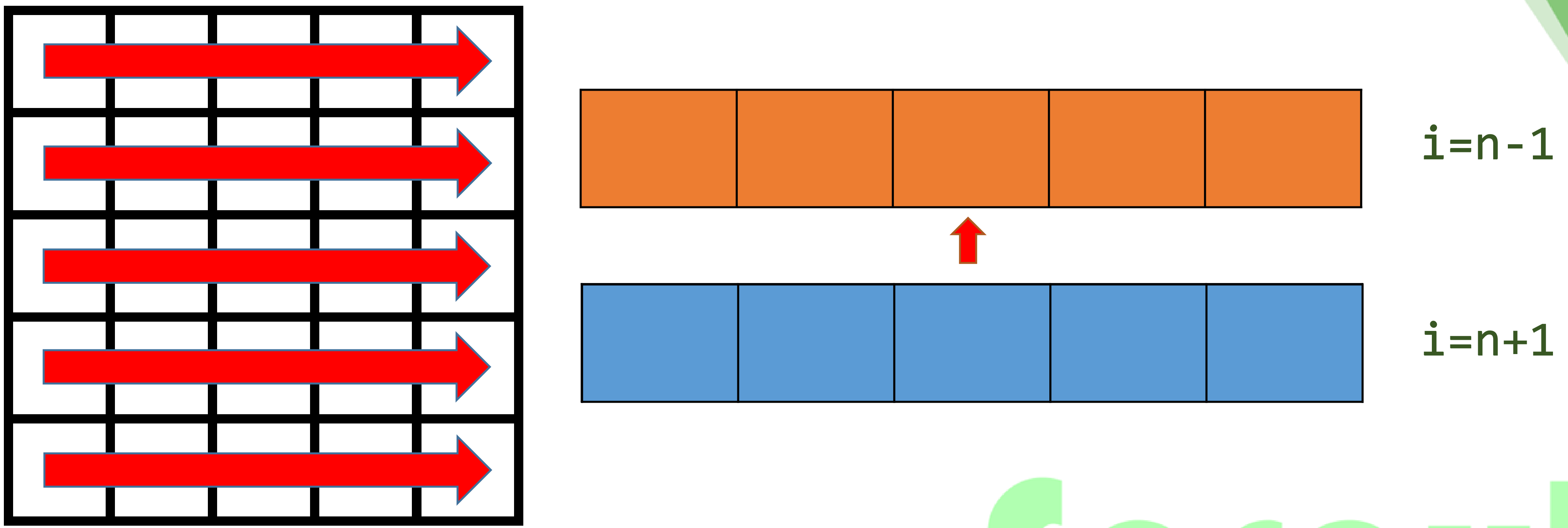
- 最後答案
  - f(N,M)

空間優化

- 如何儲存狀態？
  - 開一個N*M的二維陣列
  - 能不能開少一點？
- 注意到f(n,?)只會用到f(n-1,?)和f(n,?)
  - 不會用到f(n-2,?)、f(n-3,?)等狀態！
  - 滾動數組
  - 壓成1維陣列

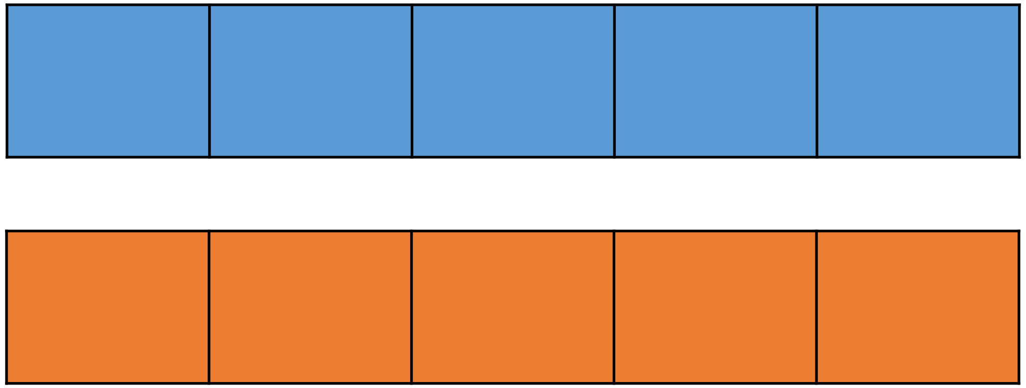
空間優化

- 只開兩個陣列，做完一次之後將結果複製到原本的陣列



空間優化

- 滾動數組：兩個陣列交替使用



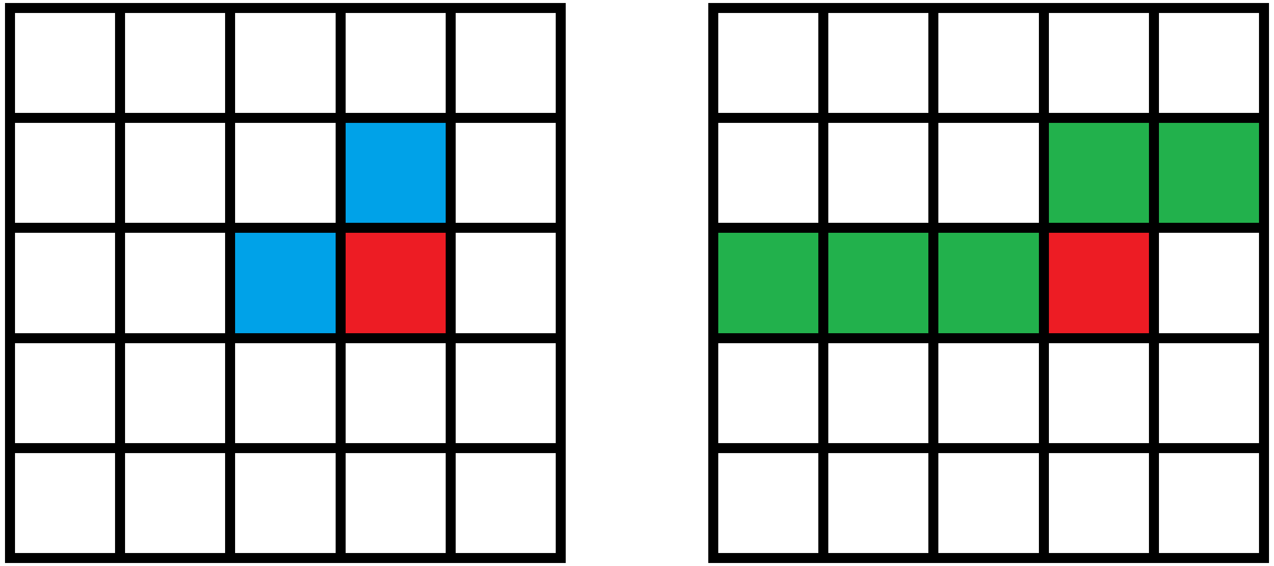
i=n+1

i=n+2

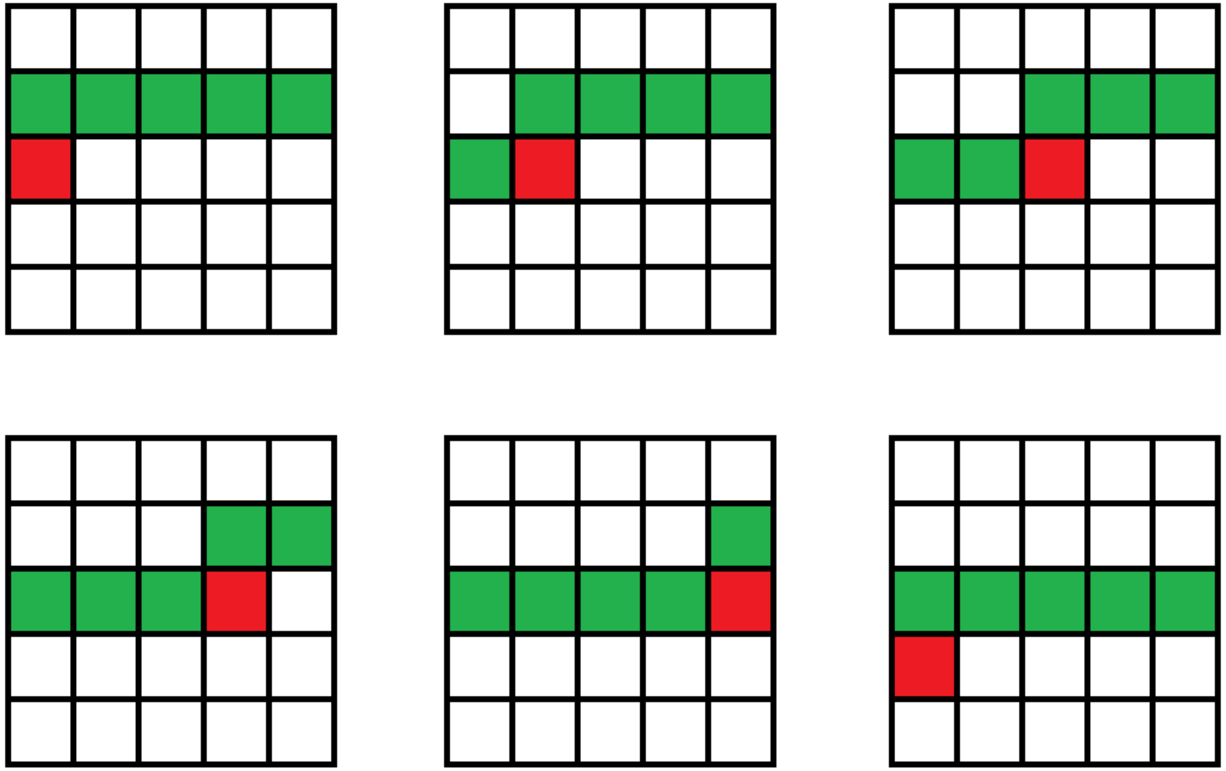
- dp[i][j]=max(dp[i-1][j],dp[i][j-1])+a[i][j]
- dp[i%2][j]=max(dp[(i+1)%2][j],dp[i%2][j])+a[i][j]

空間優化

- 只使用1維陣列
- 注意到，每個狀態(紅色)只會使用到上面一格和左邊一格的狀態(藍色)
- 開一個一維陣列表示右圖中綠色格子的值：dp[j]



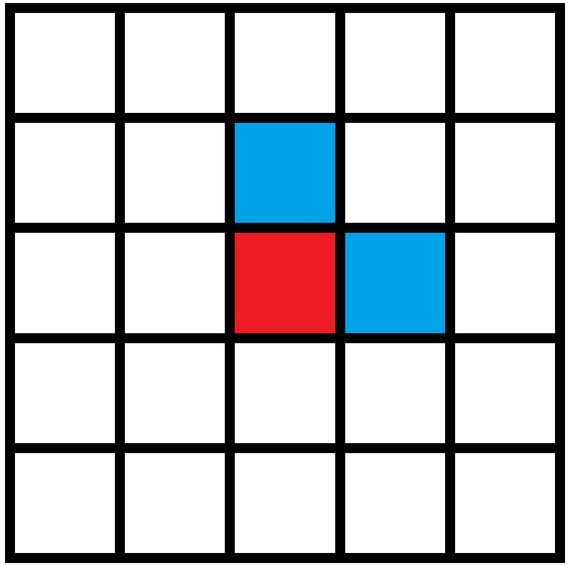
空間優化



- dp[j]=max(dp[j-1],dp[j])+a[i][j]

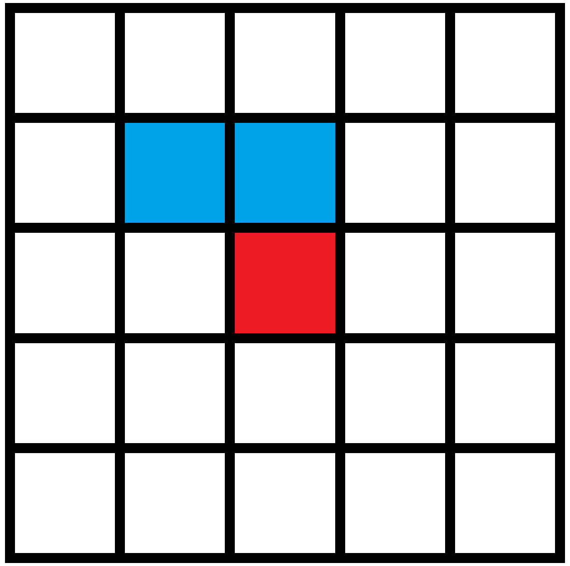
空間優化

- 當轉移式為f(i,j)=min(f(i,j+1),f(i-1,j))+a[i][j]
  - dp[j]=min(dp[j+1],dp[j])+a[i][j]
  - j由M掃到1



空間優化

- 當轉移式為f(i,j)=min(f(i-1,j),f(i-1,j-1))+(i+j)
  - dp[j]=min(dp[j],dp[j-1])+(i+j)
  - j由M掃到1



0/1 背包問題

- 有一個可以耐重W的背包，及N個物品，每個物品有各自的重量w_i和價值v_i，求在不超過重量限制的情況下往背包塞盡量多的東西，總價值最大為多少？
- 如果w_i和v_i都很大，則此問題為一NP問題，但如果範圍較小，則可以用DP的方法解決

- 暴力法：窮舉2^N種可能的取法，找重量小於W的v_i總合最大值

0/1 背包問題

- 定義狀態
  - f(n,m)表示從前n個物品中選出重量總和恰為m的物品時，價值總合的最大值。若不存在一種取法使得重量為m，則f(n,m)=-INF(或是其他數值，如-1)
  - 狀態數：N*W (重量超過W就不需要考慮了)

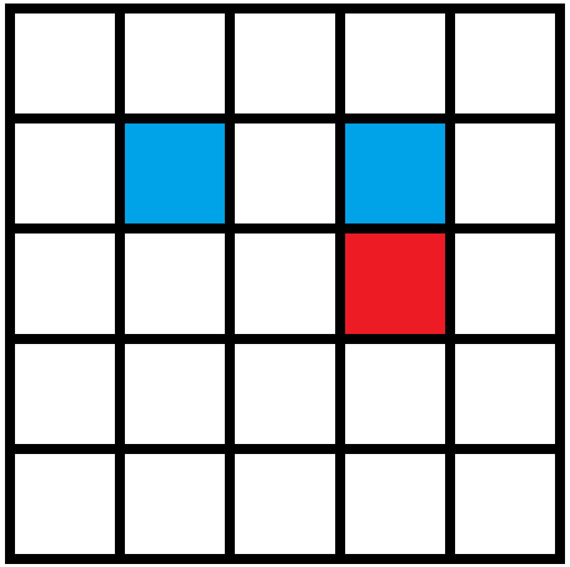
- 狀態轉移
  - 從前n樣物品中選擇物品的最佳方案，一定是「有選到第n樣物品」和「沒選到第n樣物品」其中一個(或者兩者一樣好)
  - 如果最佳方案包含第n樣物品，則此最佳方案必為「選擇第n樣物品」及「從前n-1項物品中取出重量為$m-w_n$的最佳方案」，因此可以得到 $f(n,m) = f(n-1,m-w_n)+v_n$
  - 如果最佳方案不包含第n樣物品，則$f(n,m) = f(n-1,m)$

0/1 背包問題

- 狀態轉移
  - $f(n,m) = \max(f(n-1,m),\ f(n-1,m-w_n)+v_n),\ m \geqq w_n$
  - $f(n,m) = f(n-1,m),\ m < w_n$
- 初始條件
  - $f(0,0) = 0$
  - $f(0,k) = -\mathrm{INF}$, for $k>0$
- 最後答案
  - $\max(f(N,k)),\ 0 \leqq k \leqq W$

0/1 背包問題

- 實做
  - 注意到，f(n,m)只依賴於f(n-1,m)及f(n-1,m-w_n)
- 滾動數組
  - dp[n%2][m]=max(dp[(n+1)%2][m],
    dp[(n+1)%2][m-w[n]]+v[n])
- 開一維陣列
  - m從M跑到0
  - dp[m]=max(dp[m], dp[m-w[n]]+v[n])



- 時間複雜度
  - O(NW)，N為物品個數，W為背包重量上限

0/1 背包問題

- 另解
- 定義狀態
  - f(n,m)表示從前n個物品中選出價值總和恰為m的物品時，重量總合的最小值。若不存在一種取法使得價值為m，則f(n,m)=INF
  - 狀態數：N*V (V為所有物品總合)

- 狀態轉移
  - 如果最佳方案包含第n樣物品，則此最佳方案必為「選擇第n樣物品」及「從前n-1項物品中取出價值為$m-v_n$的最佳方案」，因此可以得到 $f(n,m) = f(n-1,m-v_n)+w_n$
  - 如果最佳方案不包含第n樣物品，則f(n,m) = f(n-1,m)

0/1 背包問題

- 狀態轉移
  - $f(n,m) = \min(f(n-1,m),\ f(n-1,m-v_n)+w_n),\ m \geq v_n$
  - $f(n,m) = f(n-1,m),\ m < v_n$
- 初始條件
  - $f(0,0) = 0$
  - $f(0,k) = \mathrm{INF}$, for $k>0$
- 最後答案
  - $\max(k)$, for all $0 \leq f(N,k) \leq W$
- 時間複雜度
  - O(NV)，N為物品個數，V為物品價值總合

Sprout

0/1 背包問題

- 比較兩種做法

- 用重量做為狀態
  - 空間複雜度：O(W)
  - 時間複雜度：O(NW)
  - 限制：W不能太大

- 用價值做為狀態
  - 空間複雜度：O(V)
  - 時間複雜度：O(NV)
  - 限制：V不能太大

無限背包問題

- 有一個可以耐重W的背包，及N種物品，每種物品有各自的重量w_i和價值v_i，且數量為無限多，求在不超過重量限制的情況下往背包塞盡量多的東西，總價值最大為多少？

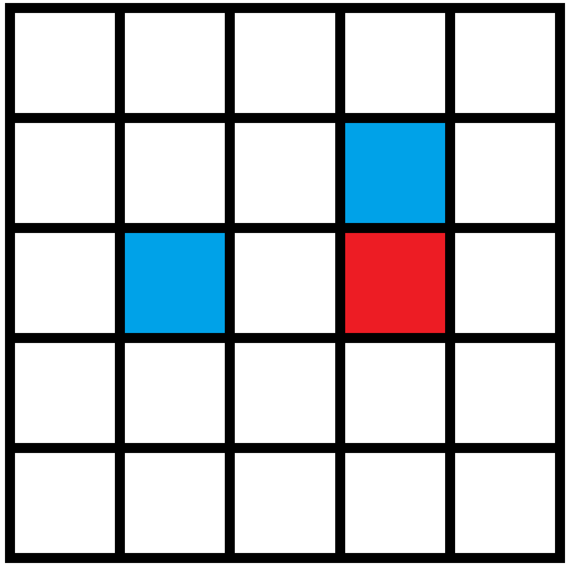
- 定義狀態
  - f(n,m)表示從前n種物品中選出重量總和恰為m的物品時，價值總合的最大值。若不存在一種取法使得重量為m，則f(n,m)=-INF

無限背包問題

- 狀態轉移
  - 從前n樣物品中選擇物品的最佳方案，一定是「第n樣物品取了0個」、「第n樣物品取了1個」...「第n樣物品取了k個」中的最佳方案，其中k為滿足 $w_i*k \leq m$ 的最大可能值
  - $f(n,m)=\max(f(n-1,m),$
    $f(n-1,m-w_n)+v_n,$
    $f(n-1,m-2*w_n)+2*v_n,$
    $...,$
    $f(n-1,m-k*w_n)+k*v_n\ \ )$
  - 整理之後得到 $f(n,m)=\max(f(n-1,m),\ f(n,m-w_n)+v_n)$

無限背包問題

- 實做
  - 注意到，$f(n,m)$只依賴於$f(n-1,m)$及$f(n,m-w_n)$
- 滾動數組
  - dp[n%2][m]=max(dp[(n+1)%2][m],
    dp[n%2][m-w[n]]+v[n])
- 開一維陣列
  - m從0跑到M
  - dp[m]=max(dp[m], dp[m-w[n]]+v[n])



- 時間複雜度
  - O(NW)，N為物品個數，W為背包重量上限

有限背包問題

- 有一個可以耐重W的背包，及N種物品，每種物品有各自的重量w_i和價值v_i，且數量為k_i個，求在不超過重量限制的情況下往背包塞盡量多的東西，總價值最大為多少？
- 做法：將k_i個相同物品視為不同物品，做0/1背包，時間複雜度為O(NWK)，其中K為重複數量的最大值

有限背包問題

- 另一種做法
- 狀態轉移
  - $f(n,m)=max(f(n-1,m), f(n-1,m-w_n)+v_n, f(n-1,m-2*w_n)+2*v_n,...,f(n-1,m-k*w_n)+k*v_n)$, where $k \leq k_i$
- O(K)轉移
- 時間複雜度還是O(NWK)

- 更快的做法將在之後的課程中提到，有限背包問題有時間複雜度O(NW*logK)及O(NW)的做法

換零錢問題

- 有N個不同的銅板，面額分別為c[1~N]，問
  - 1. 能不能湊出恰好M元？
  - 2. 如果可以，共有幾種方法？
  - 3. 如果可以，最少需要用幾個銅板？
  - 4. 能不能分成兩堆，使得兩堆的總和相等？
  - 5. 能不能分成兩堆，使得兩堆的總和為x:y？
  - 6. ......

換零錢問題

- 其實這根本是弱化版的背包問題！
- 可以想成每樣物品有重量c[i]，價值為1

- 1. 能不能湊出恰好M元？
  - 因為只問「能不能湊出」，因此只需要開bool陣列就可以了
  - f(n,m) = f(n-1,m) OR f(n-1,m-c[n])

- 2. 湊出M元的方法數有幾種？
  - f(n,m) = f(n-1,m) + f(n-1,m-c[n])

換零錢問題

- 3. 湊出M元需要最少的銅板數是幾個？
  - 完全是背包問題，只是最大值改成最小值
  - f(n,m) = min(f(n-1,m), f(n-1,m-c[n])+1)
- 4. 能不能分成兩堆，使得兩堆總和相等？
  - 只要看能不能湊出sum(c[i])/2元就可以了
- 5. 能不能分成兩堆，使得兩堆總和為x:y？
  - 只要看能不能湊出sum(c[i])*x/(x+y)元就可以了