Dynamic Programming (2) 下

by music960633

Sprout

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)

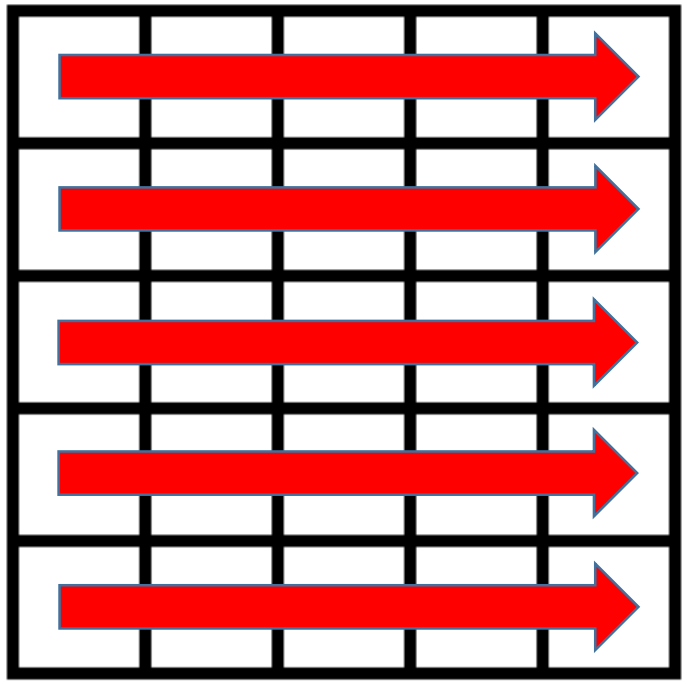
空間優化

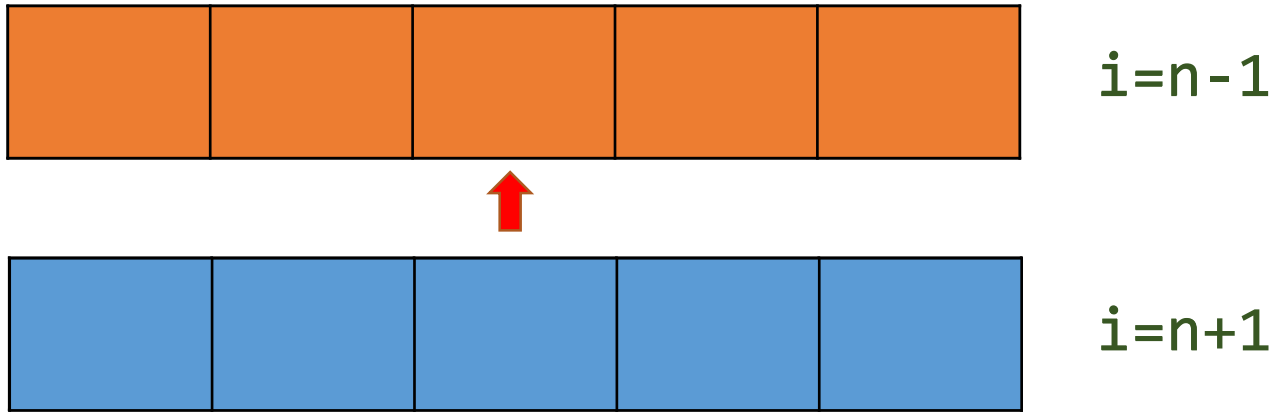
- 如何儲存狀態？
  - 開一個N*M的二維陣列
  - 能不能開少一點？

- 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

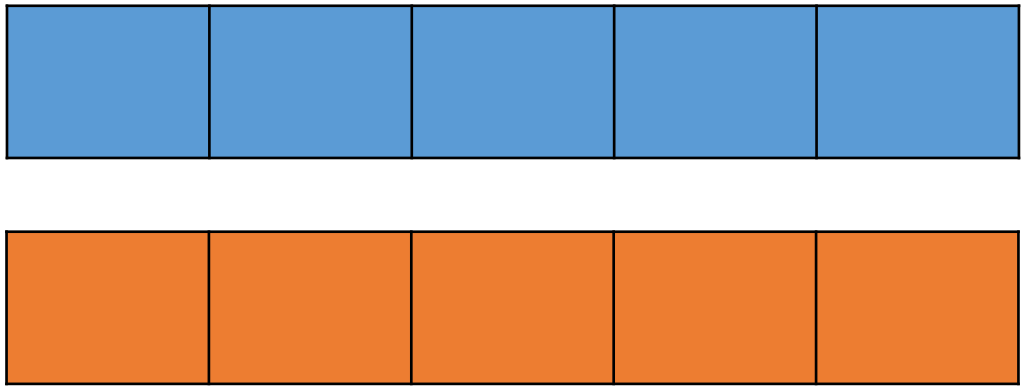
- 只開兩個陣列，做完一次之後將結果複製到原本的陣列





空間優化

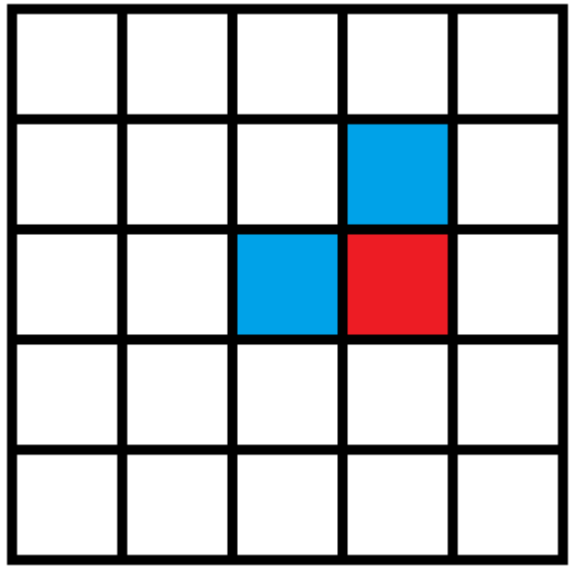
- 滾動數組：兩個陣列交替使用

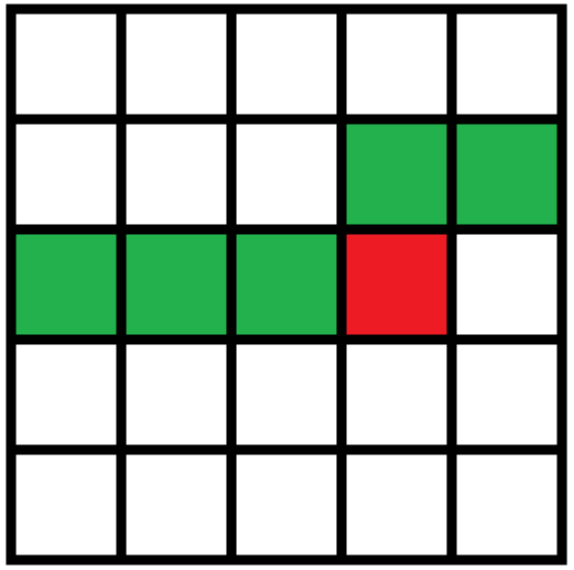


- dp[i][j]=max(dp[i-1][j],dp[i][j-1])+a[i][j]
- dp[i%2][j]=max(dp[(i+1)%2][j],dp[i%2][j])+a[i][j]

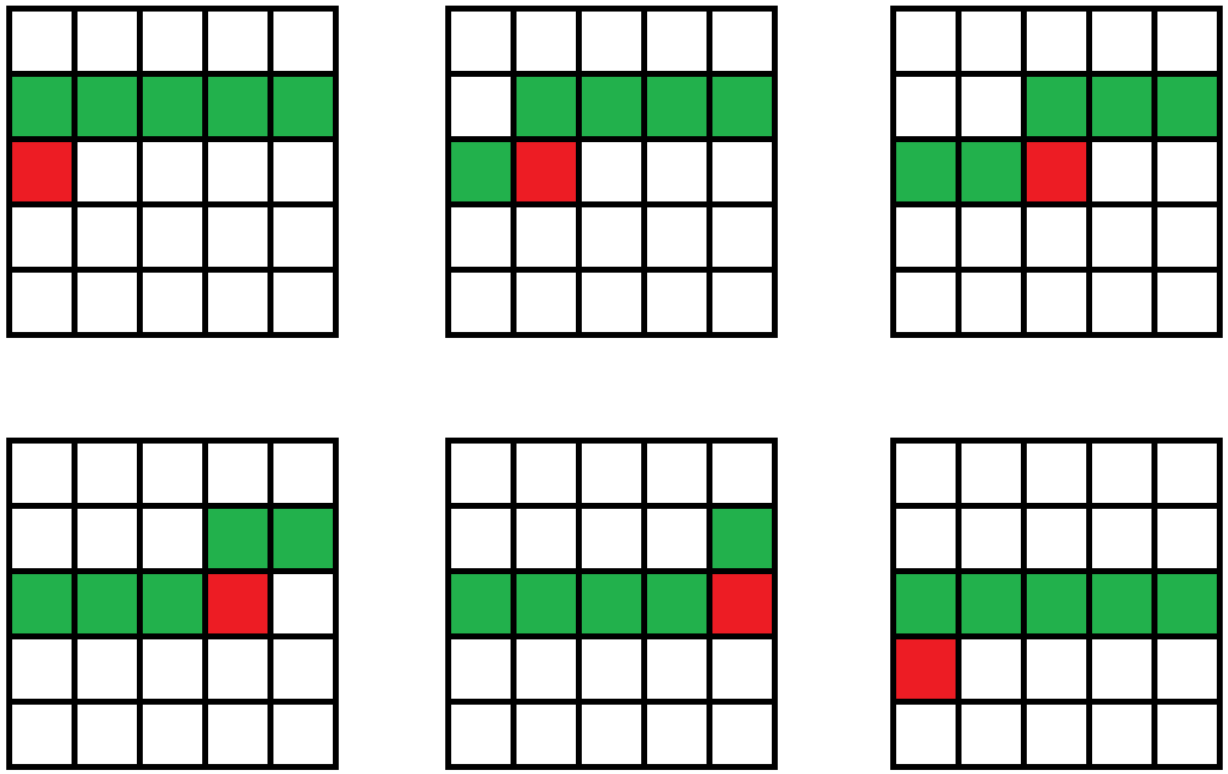
空間優化

- 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]





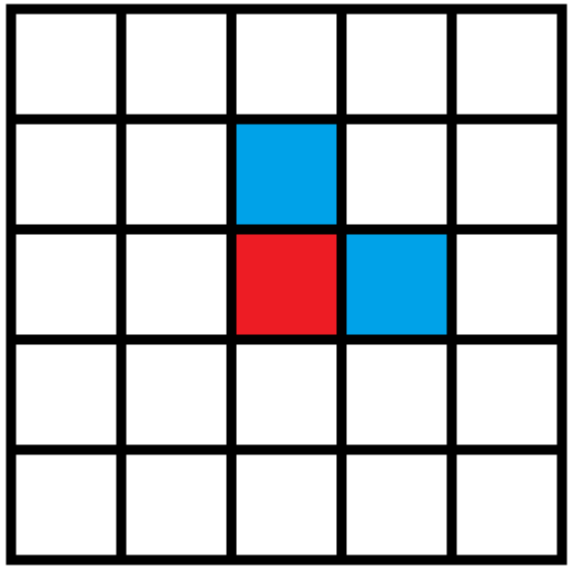
空間優化



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

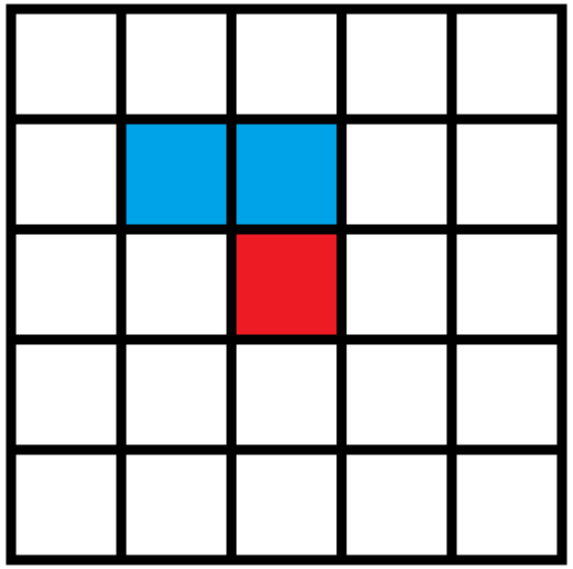
空間優化

- 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)

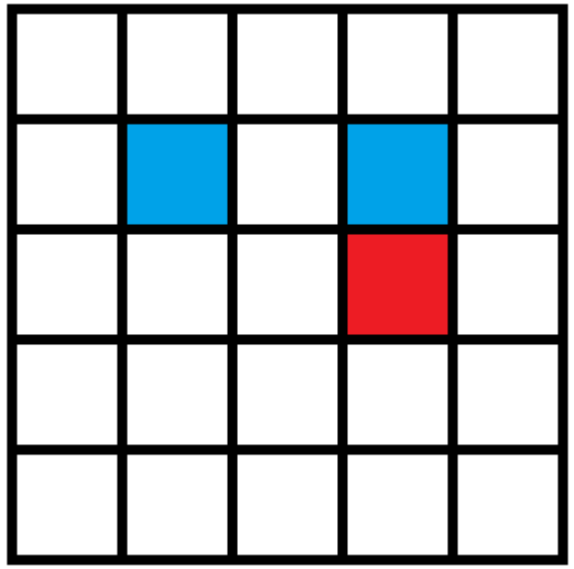
- 狀態轉移
  - 從前n樣物品中選擇物品的最佳方案，一定是「有選到第n樣物品」和「沒選到第n樣物品」其中一個(或者兩者一樣好)
  - 如果最佳方案包含第n樣物品，則此最佳方案必為「選擇第n樣物品」及「從前n-1項物品中取出重量為$m-w_n$的最佳方案」，因此可以得到 $f(n,m) = f(n-1,m-w_n)+v_n$
  - 如果最佳方案不包含第n樣物品，則f(n,m) = f(n-1,m)

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

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

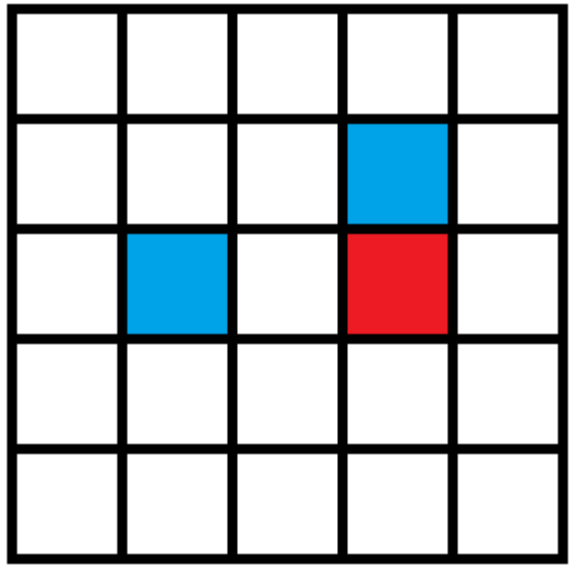
- 狀態轉移
  - 從前n樣物品中選擇物品的最佳方案，一定是「第n樣物品取了0個」、「第n樣物品取了1個」...「第n樣物品取了k個」中的最佳方案，其中k為滿足 $w_i*k \leq m$ 的最大可能值

  - $f(n,m)=\max(f(n-1,m),$
    $f(n-1,m-w_n)+v_n,$
    $f(n-1,m-2*w_n)+2*v_n,$
    $...,$
    $f(n-1,m-k*w_n)+k*v_n \ )$

  - 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 $k \leqq k_i$
- 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