Dynamic Programming (2) 上

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$O(n^2)$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$O(1)=O(n^2)$

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