Dynamic Programming (3)

by music960633

Sprout

課程內容

- 矩陣快速冪優化
- 狀態壓縮
- 資料結構的優化
- 有限背包問題的優化

矩陣快速冪優化

- 已知一N*N的矩陣A，可以用Divide and Conquer的方法在$O(N^3\log(k))$時間求得A^k
- 這跟DP有什麼關係？

矩陣快速冪優化

- 用1*2的骨牌填滿2*n的格子，共有幾種排法？

- f(n)　=f(n-1)+f(n-2)
- f(n-1)=f(n-1)

- $\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix}$

矩陣快速冪優化

- $\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix}$
- $\begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n-2) \\ f(n-3) \end{bmatrix}$
- $\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} f(1) \\ f(0) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1}$可以用矩陣快速冪求出

- 如此可以在O(log(n))時間求出f(n)的值

矩陣快速冪優化

- 將n個排成一列的格子塗上紅、綠、藍三種顏色，且藍綠不可相鄰，問有幾種塗法？

- f(n,0)=f(n-1,0)+f(n-1,1)+f(n-1,2)
- f(n,1)=f(n-1,0)+f(n-1,1)
- f(n,2)=f(n-1,0)+f(n-1,2)
- $$\begin{bmatrix} f(n,0) \\ f(n,1) \\ f(n,2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(n-1,0) \\ f(n-1,1) \\ f(n-1.2) \end{bmatrix}$$

狀態壓縮

- Travelling Salesman Problem(TSP)
- 求一張圖上權重最小的Hamilton Circuit

- 已知此問題為NP-Hard，暴力做法為O(n!)，有沒有快一點的做法呢？

狀態壓縮

- 如果只有5個點，可以怎麼做？
- 定義狀態：dp[n][s0][s1][s2][s3][s4]
  - n表示目前在n號節點上
  - si表示是否走過第i號節點(0/1)
- 狀態轉移：
  - dp[2][0][1][1][1][1]=min(dp[1][0][1][0][1][1]+dis[1][2],
    dp[3][0][1][0][1][1]+dis[3][2],
    dp[4][0][1][0][1][1]+dis[4][2])
- 答案：dp[0][1][1][1][1][1]

狀態壓縮

- 如果點數再多一點，例如15個點呢？
- 狀態：dp[][][][][][][][]......[][]
- 太累了！
- 只要記錄有沒有走訪過(0/1)，因此可以使用位元運算

- 定義狀態：dp[n][s]
  - n表示目前在n號點上
  - s表示目前走過哪些點
  - 狀態數量：$n*2^n$

狀態壓縮

- 狀態轉移：
  - dp[n][s]=min(dp[i][s-(1<<n)]+dis[i][n]),
    for all i such that (s & (1<<i)) != 0
  - 時間複雜度：O(n)

- 答案：dp[0][(1<<N)-1]
- 整體時間複雜度：$O(n^2*2^n)$

- PS: 注意初始值的設定

資料結構的優化

- 回顧之前看過的問題：
- 給定一整數陣列，求取出數字的總合最大，且滿足兩數距離不能小於K

- 狀態：dp[n]: 從前n個數中取數，且有取到arr[n]的最大總合
- 轉移：dp[n]=arr[n]+max(dp[n-K],dp[n-K-1],...,dp[1])
- 答案：max(dp[1],dp[2],...,dp[N])

- 時間複雜度：$O(n^2)$

資料結構的優化

- 再令dpMax[n]=max(dp[1],dp[2],...,dp[n])

- 狀態：dpMax[n]： 從前n個數中取數字的總合最大值
- 轉移：dpMax[n]=max(dpMax[n-1],dp[n])

  =max(dpMax[n-1],arr[n]+dpMax[n-K])
- 答案：dpMax[N]

- 時間複雜度：O(n)

資料結構的優化

- 稍微改一下問題：
- 給定一整數陣列，求取出數字的總合最大，且滿足兩數距離不能大於K

- 狀態：dp[n]: 從前n個數中取數，且有取到arr[n]的最大總合
- 轉移：dp[n]=arr[n]+max(dp[n-1],dp[n-2],...,dp[n-K],0)
- 答案：max(dp[1],dp[2],...,dp[N])

- 時間複雜度：$O(n^2)$

資料結構的優化

- 再令dpMax[n]=max(dp[1],dp[2],...,dp[n])
- 狀態轉移裡面是max(dp[n-1]~dp[n-K])，取這個max沒意義啊

- 囧...好吧，那就令dpMax[n]=max(dp[n],...,dp[n-K+1])
- 發現dpMax[n]和dpMax[n-1]好像沒什麼關係，結果還是得重算

- 有其他辦法加速嗎？

資料結構的優化

- 方法1：
- 我們發現我們要需要取的是一個區間的max
- 線段樹！

- 時間複雜度：O(nlog(n))

- 缺點：線段樹比較複雜，也比較難寫

資料結構的優化

- 方法2：
- 注意到取max的左界是遞增的，也就是說，只要一個位置跑到範圍外面之後，之後就再也不會被取到了
- 使用heap取最大值
  - push：記錄該點的值和位置
  - top ：如果取到的值已經「過期」了則直接丟掉，直到top不是過期的
- 時間複雜度：O(nlog(n))
- 優點：heap好寫多了，甚至還有stl

資料結構的優化

- 方法3
- 注意到如果存在i,j使得**i<j且dp[i]<dp[j]**，則dp[i]就永遠不會被取到了
- 只要記錄「可能成為max的dp[]」就好了

- 實做：
  - 假設有個神祕的容器，每次放進(dp[n],n)
  - push: 將所有「i<n 且 dp[i]<dp[n]」的元素丟掉，再放入(dp[n],n)
  - top : 先將所有「過期」的元素丟掉，接著取出dp值最大的元素

資料結構的優化

- 方法3
- 我們可以發現，對於任意兩個容器內的元素(dp[i],i)和(dp[j],j)，一定滿足：若i<j，則dp[i]>dp[j]
- 先將這些元素依照dp值的順序放入陣列裡，然後觀察push、pop和top做了哪些事：
  - push: 從dp值較小的一端，一直將元素pop出來，直到這端的第一個元素的dp[i]>dp[n]，接著將(dp[n],n)推進陣列
  - top ： 從dp值較大的一端，一直將元素pop出來，直到這端的第一個元素是還沒過期的，接著回傳這端的一個元素的dp值(max)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

()				

```
push (3,1)
```

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)				

push (3,1)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)				

push (-2+3,2)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)	(1,2)			

push (-2+3,2)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)	(1,2)			

push (-1+3,3)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)	(2,3)			

push (-1+3,3)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)	(2,3)			

push (-2+3,4)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)	(2,3)	(1,4)		

push (-2+3,4)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

(3,1)	(2,3)	(1,4)		

Sprout

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

	(2,3)	(1,4)		

push (2+2,5)

資料結構的優化

- ex: arr[] = {3,-2,-1,-2,2}, K=3

	(4,5)			

push (2+2,5)

資料結構的優化

- 方法3
- 可以使用陣列或deque實做

- 時間複雜度
  - 每個元素只會被push一次
  - 每個元素只會被pop一次
  - O(n)!

- 該方法稱作「單調隊列優化」

資料結構的優化

- 再看一個例題
- 円円想在一條筆直的路上開設一些漢堡店，已知他取了N個等間隔的點做了評估，得到在每個點開店可以賺的金額val[i](負的表示賠錢)。另外，如果開設超過一間店，則兩間相鄰的店面不能相距超過K單位，且這兩家店之間需要交通成本，金額為(c*距離)。求円円最多可以賺多少錢？

資料結構的優化

- 狀態：
  - dp[n]表示在第1~n個點開店，且第n個點有開店的最大值
- 轉移：
  - dp[n]=val[n]+max(dp[i]-c*(n-i)), n-K ≦ i ≦ n-1
- 時間複雜度：O(NK)

資料結構的優化

- 優化
- 改寫一下轉移式：
  - dp[n]=val[n]+max(dp[i]-c*n+c*i)), n-K ≦ i ≦ n-1
  - dp[n]=(val[n]-c*n)+max(dp[i]+c*i), n-K ≦ i ≦ n-1
- 令t[i]=dp[i]+c*i
  - dp[n]=(val[n]-c*n)+max(t[i]), n-K ≦ i ≦ n-1
  - 「max(t[i]), n-K ≦ i ≦ n-1」可以使用單調隊列優化！
- 時間複雜度：O(N)

有限背包問題的優化

- 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
- 問題：真的需要分成k[i]那麼多個嗎？

有限背包問題的優化

- 對於同一種物品，不知道取幾個是最佳解，而分成個別的物品一定能找到最佳解
- 想法：如果可以將k[i]個同種物品分成幾堆，且可以經由不同的組合湊出1~k[i]每種組合就可以了！
- ex: k[i]=6，此時把6個相同物品分成{1,2,3}三堆，可以發現
  - 1=1 / 2=2 / 3=3 / 4=1+3 / 5=2+3 / 6=1+2+3
  - 如此只要分成3堆也可以得到最佳解

有限背包問題的優化

- 要怎麼分堆呢？
- 二進位！
  - 將k[i]分成{1,2,4,8,...,2^p,q}，其中p為使2^{p+1}-1不大於k[i]的最大整數
  - ex: 21可以分成{1,2,4,8,6}
- 為什麼這樣分可以湊出所有的組合？
  - 若x<2^{p+1}，則一定可以用{1,2,4,...,2^p}湊出x
  - 若x≥2^{p+1}，則先取q，剩下的x-q<2^{p+1}，就可以用{1,2,4,...,2^p}湊出
- 時間複雜度
  - 最多分出log(k[i])+1堆
  - O(NW*logK)

有限背包問題的優化

- 單調隊列優化
- 觀察轉移式
  - dp[n][m]=max{dp[n-1][m-w[n]*i]+v[n]*i}, 0 ≦ i ≦ k[n]
  - dp[n][m]=max{dp[n-1][j]+v[n]*(m-j)/w[n]},
    j=m,m-w[n],...,m-w[n]*k[n]
- 分別對所有除w[n]餘數相同的索引值做單調隊列優化DP，一共做w[n]次，可以得到O(NW)的演算法