动态规划2

赵宗昌

2018.5.31

二维模型f[i][j]

一. 最长公共子序列模型LCS

- 最长公共子序列（Longest Common Subsequence，LCS）
- 最长公共子串（Longest Common Substirng）
- 区别为：子串是串的一个连续的部分；子序列则是从改变序列的顺序，而从序列中去掉任意的元素而获得新的序列；也就是说，子串中字符的位置必须是连续的，子序列则可以不必连续。

如：

abcbdab

bdcaba

2 1 4 5 6 7

3 2 5 4 7 6 8

1.1265 最长公共子序列（LCS）（字符串 或 整数序列）

【问题描述】

给定两个字符序列:$X=\{x_1, x_2, \cdots, x_n\}$ ； $Y=\{y_1, y_2, \cdots, y_m\}$

求X和Y的一个最长公共子序列长度。

举例：

X={a, b, c, b, d, a, b}　Y={b, d, c, a, b, a}

其中一个最长公共子序列为：LCS={b, c, b, a}， 长度是4。LCS可能不止一个。

【输入】

第一行：字符串X；

第二行：字符串Y。

【输出】

最长公共子串的长度。

【样例输入】

abcbdab

bdcaba

【样例输出】

4

$X=\{x_1,\ldots,x_{i-1}，x_i\}$

$Y=\{y_1,\ldots,y_{j-1}，y_j\}$

设f[i][j]表示x[1..i]与y[1..j]的最长公共子序列的长度。

确定状态转移方程和边界条件：

分两种情况来考虑：

当x[i]=y[j]:x[i]与y[j]在公共子序列中,该情况下，f[i][j]=f[i-1][j-1]+1。

当x[i]≠y[j]:

x[i]不在公共子序列中：该情况下f[i][j]=f[i-1][j]；

y[j]不在公共子序列中：该情况下f[i][j]=f[i][j-1]；

f[i][j]取上述三种情况的最大值。综上：

状态转移方程：f[i][j]=max{ f[i-1][j-1]+1； x[i]=y[j]

f[i-1][j]；

f[i][j-1],

}

边界条件：f[0][j]=0,f[i][0]=0。

目标：f[n][m];

或者：

- 考虑：
- $X=\{x_1, \cdots, x_{i-1}, x_i\}$
- $Y=\{y_1, \cdots, y_{j-1}, y_j\}$
- 定义f[i,j]为X的前i个字符和Y的前j个字符中最大公共子序列的长度。
- 当$x_i=y_j$时： f[i,j]=f[i-1,j-1]+1；
- 当$x_i<>y_j$时：f[i,j]=max{f[i,j-1]；f[i-1,j]}。

注意字符串的下标和i与j的关系，字符下标从0开始。

1297多组数据LCS

2.1276：编辑距离

【题目描述】

设A和B是两个字符串。我们要用最少的字符操作次数，将字符串A转换为字符串B。
这里所说的字符操作共有三种（对A而言）：
1、删除一个字符；
2、插入一个字符；
3、将一个字符改为另一个字符。
对任意的两个字符串A和B，计算出将字符串A变换为字符串B所用的最少字符操作次数。
【输入】
第一行为字符串A；第二行为字符串B；字符串A和B的长度均小于2000。
【输出】
只有一个正整数，为最少字符操作次数。
【输入样例】
sfdqxbw
gfdgw
【输出样例】
4

$X=\{x_1,\ldots,x_{i-1}，x_i\}$

$Y=\{y_1,\ldots,y_{j-1}，y_j\}$

设f[i][j]表示把x[1..i]变为y[1..j]需要的最少操作次数。

状态转移方程：

f[i][j]=min{

当x[i]=y[j]：f[i-1][j-1]；

当x[i]≠y[j]：f[i-1][j]+1； 删除x[i]

f[i][j-1] +1; 在x[i]后插入一个字符，那么一定是=y[j]，否则无意义

f[i-1][j-1]+1;将x[i]变为y[j]

}

边界条件：

f[0][i]=i 全部插入

f[i][0]=i 全部删除

目标：f[n][m];

3.1298：计算字符串距离

- 对于两个不同的字符串，我们有一套操作方法来把他们变得相同，具体方法为：
- 修改一个字符（如把“a”替换为“b”）；
- 删除一个字符（如把“traveling”变为“travelng”）。
- 比如对于“abcdefg”和“abcdef”两个字符串来说，我们认为可以通过增加/减少一个“g”的方式来达到目的。无论增加还是减少“g”，我们都仅仅需要一次操作。我们把这个操作所需要的次数定义为两个字符串的距离。
- 给定任意两个字符串，写出一个算法来计算出他们的距离。

1276和1298的不同点：

- 1276只对X操作，Y不变，让X变为Y
- 1298 X和Y都可以变，变化后相同

$X=\{x_1,\ldots,x_{i-1}, x_i\}$

$Y=\{y_1,\ldots,y_{j-1}, y_j\}$

设f[i][j]表示把x[1..i]变为y[1..j]需要的最少操作次数。

状态转移方程：

f[i][j]=min{

当x[i]=y[j]：f[i-1][j-1]；

当x[i]≠y[j]：f[i-1][j]+1； 删除x[i]

f[i][j-1]+1; 删除y[j]

f[i-1][j-1]+1; 将x[i]变为y[j]或将y[j] 变为x[i]

}

边界条件：

f[0][i]=i Y全删

f[i][0]=i X全除

目标：f[n][m];

4.1306：最长公共子上升序列（LCIS）

【输入样例】

5

1 4 2 5 -12

4

-12 1 2 4

【输出样例】

2

1 4

LIS与LCS的结合：

- a[1..n]
- b[1..m]
- f[i][j]:是a[1..i]与b[1..j]的LCIS，同时必须是以b[j]为结尾。
- a[i]!=b[j]时：f[i][j]=f[i-1][j];
- a[i]==b[j]时：
  f[i][j]=max{f[i][k],(1<=k<j&&b[k]<b[j])}+1;
- 目标：max{f[n][i]} (1<=i<=m)

$O(n^3)$优化为$O(n^2)$？

二. 资源分配问题

1.1266：机器分配

高效设备M台，准备分给下属的N个分公司.

如：

n=7,m=6

7 6

1 2 3 4 5 6

6 5 4 3 2 1

6 4 8 2 50 194

100 200 300 10 24 72

200 300 400 200 100 50

10 20 30 40 50 60

1 1 1 1 1 1

- f[i][j]： 前i个公司（1..i）分配j台设备最大获利。
- 考察前i-1个公司分配机器台数k,第i个公司分配j-k台
- f[i][j]=max{f[i-1][k]+a[i][j-k]} (0<=k<=j)

2.1279：橱窗布置(flower)

- 【输入样例】
- 3 5
- 7 23 －5 －24 16
- 5 21 -4 10 23
- -21 5 -4 -20 20

- 【输出样例】
- 53
- 2 4 5

- f[i][j]:前i束花插入到前j个花瓶获得的最大值。
- 考察第i束花插在哪个花瓶k上更好
- f[i][j]=max{f[i-1][j-1]+a[i][k],(i<=k<=j)}
- 目标：f[n][m];
- 注意有**负值**的最大值问题。

3.1275 乘积最大 ？

- 长度为n的数串加m个乘号，成绩最大。

- 【输入样例】
- 4 2
- 1231

- 【输出样例】
- 62

方法1：

f[i][j]:i个乘号插入到前j个数字中的最大乘积。

9 4

321044105

		3	2	1	0	4	4	1	0	5
	0	1	2	3	4	5	6	7	8	9
0		3	32	321	3210	…	…	…	…	…
1			?	?	?	?	?	?	?	?
2				?	?	?	?	?	?	?
3					?	?	?	?	?	?
4						?	?	?	?	?

f[i][j]:i个乘号插入到前j个数字中的最大乘积。
考察第i个乘号的位置：第k和k+1个数字之间。(1,k)*(k+1,j)
f[i][j]=max{f[i-1][k]*data(k+1,j) (i<=k<j)}
初始值：f[0][i]=data(1,i)
目标：f[m][n]
行优先求

适合逐行求:

```
for(int i=1;i<=n;i++)f[0][i]=data(1,i);
for(int i=1;i<=m;i++)
    for(int j=i+1;j<=n;j++)
        for(int k=i;k<j;k++)
            f[i][j]=max(f[i][j],f[i-1][k]*data(k+1,j));
cout<<f[m][n]<<endl;
```

方法2：

$f[i][j]$:前i个数字中插入j个乘号到的最大乘积。

$f[i][j]=\max\{f[k][j-1]*data(k+1,i)\ \ (j<=k<i)\}$

初始值：$f[i][0]=data(1,i)$

目标：$f[n][m]$

		0	1	2	3	4
	0					
3	1	3				
2	2	32	?			
1	3	321	?	?		
0	4	3210	?	?	?	
4	5	…	?	?	?	?
4	6	…	?	?	?	?
1	7	…	?	?	?	?
0	8	…	?	?	?	?
5	9	…	?	?	?	?

列优先：

```
for(int i=1;i<=n;i++)f[i][0]=data(1,i);
for(int j=1;j<=m;j++)//列优先求
    for(int i=j+1;i<=n;i++)
        for(int k=j;k<i;k++)
            f[i][j]=max(f[i][j],f[k][j-1]*data(k+1,i));
cout<<f[n][m]<<endl;
```

行优先：

```
for(int i=1;i<=n;i++)f[i][0]=data(1,i);
for(int i=2;i<=n;i++)// 行优先
    for(int j=1;j<=min(m,i-1);j++)
        for(int k=j;k<i;k++)
            f[i][j]=max(f[i][j],f[k][j-1]*data(k+1,i));
cout<<f[n][m]<<endl;
```

4. 1278:复制书稿(book)

复制时间最短。复制时间为抄写页数最多的人用去的时间。最大值最小问题。

- 【输入样例1】
- 9 3
- 1 2 3 4 5 6 7 8 9
- 【输出样例2】
- 1 5
- 6 7
- 8 9

- 样例2输入：
- 4 3
- 3 1 4 5
- 样例2输出：
- 1 1
- 2 3
- 4 4

先求出最小值

- f[i][j]:前i人抄写前j本书的最小值（抄的最多的人）。
- 考察第i个人抄的书从第K+1本到第j本,即前i-1个人抄前k本书。
- i-1<=k<=j-1:保证每人至少一本。
- 方程:
- f[i][j]=min(f[i][j],max(f[i-1][k],s[k+1][j]));
- s[i][j]:第i本书到第j本书的页数。预先求出。

求出f[m][n]，然后方案

- 从后先前分，尽量后面的人多抄，但还要保证前面每人至少保证一本。