第12次实验

### 1、[求连续子数组的最大和问题](https://www.cnblogs.com/allzy/p/5162815.html)

一个有N个整数元素的一维数组A[0]、A[1]、…、A[N-1]，求其中**连续的**子数组和的最大值？不需要返回子数组的具体位置、数组中包含：正、负、零整数、子数组不能空。

例如：

int A[] = { 1, -1, 2, 3, -4, 4 };

符合条件的子数组为{1, -1, 2, 3}或{2,3}或{2, 3, -4, 4}，即答案为5；

#### 实验1 内容：

1. 实现动态规划、穷举法程序；
2. 对比两种方法处理长度为100000的数组时的运行时间差异；

长度为100000数组每一个元素用rand()%100-50随机产生。

* 动态规划法

尝试寻找递归式的子结构。先考虑序列Xi-1={A[0]、A[1]、…、A[i-1]}当中和最大的连续子数组，以及序列Xi={A[0]、A[1]、…、A[i]}中和最大的连续子数组的关系。**但是**在Xi-1的连续子数组以及在Xi中的连续子数组**很有可能不互相连接**，这样两者之间的关系就很难确定。

这里将上面问题改一下，考虑序列Xi-1={A[0]、A[1]、…、A[i-1]}当中**以A[i-1]为结尾的**和最大的连续子数组（记作zi-1），以及序列Xi={A[0]、A[1]、…、A[i]}中**以A[i]为结尾的**和最大的连续子数组（记作zi）的关系。显然，原问题解的子数组一定在{z0、z1、…、zN-1}当中，而**zi的和**与**zi-1的和**的关系容易确定：

**由于zi以A[i]做结尾，zi-1以A[i-1]做结尾，所以zi要么是zi-1尾部添加元素A[i]，要么是A[i]自身，两者必居其一。**

假设zi-1的最大的连续子数组和为TempMaxSum[i-1]，则zi的最大的连续子数组和 必然是TempMaxSum[i-1]+A[i] 或者 A[i]，两者必居其一。即：

TempMaxSum[i]=max(A[i], TempMaxSum[i-1]+A[i])

**伪代码：**

定义长度足够的TempMaxSum[]；

TempMaxSum[0]初始值为 A[0];

设MaxSum= A[0]； //存储最大值

循环算出每一个TempMaxSum[i]，若发现TempMaxSum[i]大于MaxSum，则将MaxSum设置为TempMaxSum[i]；

* 穷举法

考察每一个子数组A[i]、A[i+1]、…、A[j]的和，这里i从0到N-1，j从i到N-1，用两重循环可以解决问题。

**伪代码：**

设MaxSum= A[0]； //存储最大值

循环 ( i = 0 到N-1)

{

sum = 0;

循环计算A[i]到A[j]的（i<=j<N）和放入sum，

若sum大于MaxSum， 则MaxSum = sum);

}

输出MaxSum；

--------------------------------------------------------------------------------------------------

穷取法最为直接，当然耗时也较多，时间复杂度为O(n2);

### 2、最长公共子序列问题

子序列就是在给定的序列中删除若干元素后得到的序列。给定两个序列X和Y，当另一个序列Z既是X的子序列又是Y的子序列时，就称Z是序列X和序列Y的公共子序列。最长公共子序列，就是指元素个数最多的公共子序列。例如，若X={A,B,C,B,D,A,B}，Y={B,D,C,A,B,A}，则序列{B,C,B,A}是X和Y的最长公共子序列之一，长度为4。

为了解决问题，我们先找一下它的子结构，即能否找到这个问题的递归式，它很可能是问题的状态转换函数。要找出X={x1,x2,……,xm}和Y={y1,y2,……,yn}的最长公共子序列，我们考虑X和Y的最后一个元素：

情形1：当xm=yn时，找出Xm-1和Yn-1的最长公共子序列，然后在其尾部加上xm即可得到X和Y的最长公共子序列。

情形2：当xm≠yn，显然{x1,x2,……,xm}和{y1,y2,……,yn}的最长公共子序列等于**下面两者中最长的序列**：

a) {x1,x2,……,**xm-1** } 和 {y1,y2,……, yn-1, yn} 的最长公共子序列

b) {x1,x2,……,xm-1,xm } 和 {y1,y2,……,**yn-1** } 的最长公共子序列

我们用f[i][j]记录序列Xi和Yj的最长公共子序列的长度。其中Xi={ x1,x2,……,xi}，Yj={ y1,y2,……,yj}。当i=0或者j=0时（即序列为空），Xi和Yj的最长公共子序列为空，所以此时f[i][j]=0。所以我们可以建立如下的递推关系：



矩阵f[][]就是下表的内容，容易看出，为了求**f[2][2]**，需要先知道f[1][1]、f[1][2]、f[2][1]，显然下表的第一行和第一列都为0，因此可以逐行向下、自左向右计算。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Y0 | Y1 | Y2 | …… | Yj | …… |
| X0 | f[0][0] | f[0][1] | f[0][2] |  |  |  |
| X1 | f[1][0] | f[1][1] | f[1][2] |  |  |  |
| X2 | f[2][0] | f[2][1] | **f[2][2]** |  |  |  |
| … |  |  |  |  |  |  |
| Xi |  |  |  |  | f[i][j] |  |
| … |  |  |  |  |  |  |

计算上面矩阵的伪代码如下：

void **LCSLength**(char x[], char y[],int m，int n)

{ /\* 计算最长公共子序列的长度 \*/

int F[m][n], i，j;

将F矩阵第一行设为0;

将F矩阵第一列设为0;

for (i = 1; i <= m; i++)

for (j = 1; j <= n; j++) {

若x[i]==y[j], F[i][j]= ?;

否则，若F[i-1][j]>= F[i][j-1]

…..; //计算F[i][j]

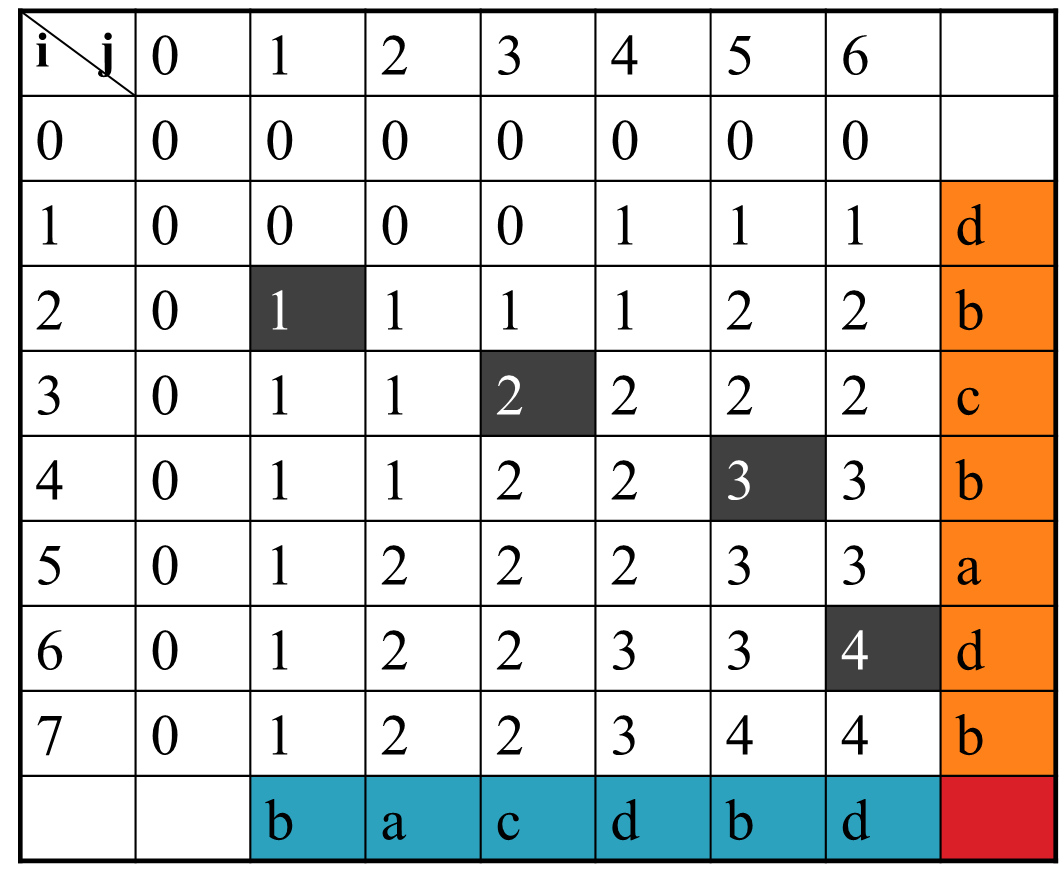
否则， …..;

return F[m][n];

}

另外：如何从上面计算出来的**长度矩阵**中找到最长公共子序列 ：（**注意观察黑色方格）**

**算法：**从矩阵右下方开始，执行：



1）向上搜索数字相同的方格，直到数字相同方格的最后一个；

2）向右搜索数字相同的方格，直到数字相同方格的最后一个；

3）这个位置对应的横坐标、纵坐标都是一样的字符，是最长公共子序列的一个元素，记下这个元素。

重复以上1）2）3）直到最后走到左上角的数值为1的空格

void **LCS**(char a[],int L[][],int m,int n)

{

初始化，并且令 i=m、j=n、k=m;

循环（i>0且j>0）

{

if(上方数字没变) i--; **//向上一格**

else if(左方数字没变) j--; **//向左一格**

else {

c[k]=x[i]或y[j]; **//记录元素**

k--; i--; j--;  **//走到相同数字区域左上角，向左上跳**

}

}

……

}

#### 实验2 内容：

**实现动态规划程序，算出最长公共子序列的长度；**

**利用上面产生的矩阵，求出最长公共子序列的内容**。