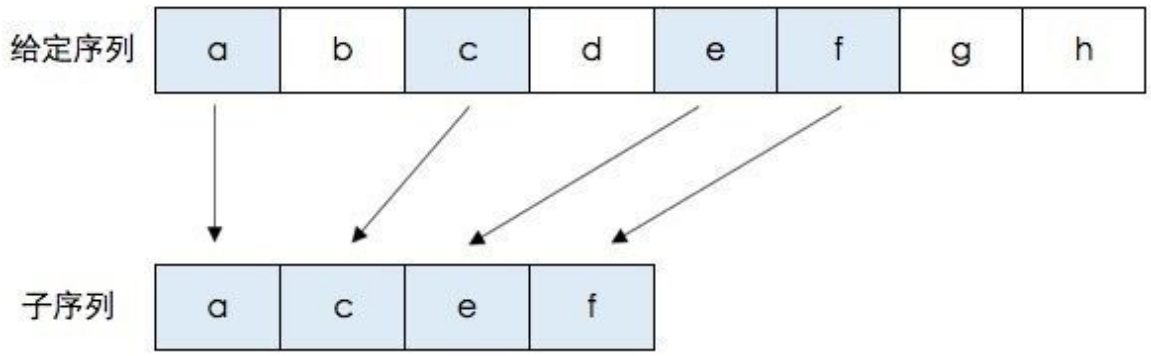
最长公共子序列(LSC)

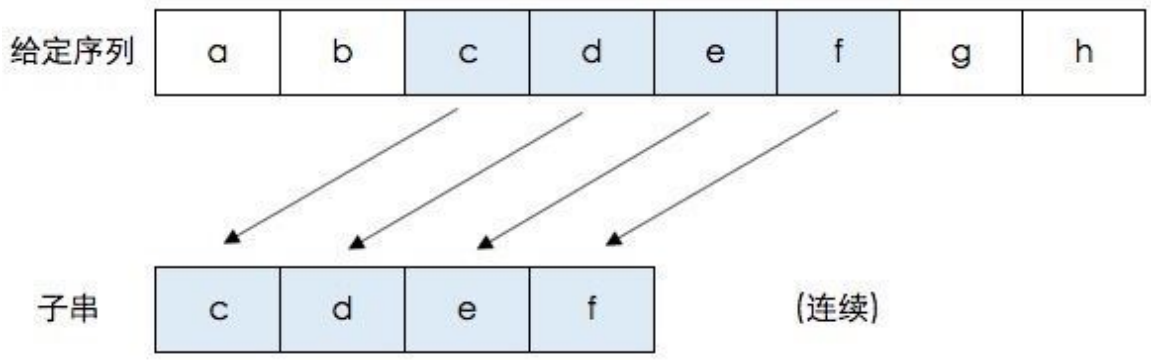
Monday, July 15, 2019 22:26

给定序列s1={1,3,4,5,6,7,7,8},s2={3,5,7,4,8,6,7,8,2}，s1和s2的相同子序列，且该子序列的长度最长，即是LCS。s1和s2的其中一个最长公共子序列是{3,4,6,7,8}

子串和子序列的区别



http://blog.csdn.net/



求出一群序列的LCS ，是NP-hard 问题，没有快速的演算法。
简单的方式是穷举法：穷举s1 的所有子序列，检查s2...sN 是否都有该子序列。

时
间复杂度是O(s1! · (s2 + ... + sN)) 。
求出两个序列的LCS ，是P 问题。接下来将介绍各种算法

子问题划分

现在要找出s1 和s2 的LCS 。写成LCS(s1, s2) 。
拆解s1 和s2 的最后一个元素，叫做e1 和e2 。

```
s1 = sub1 + e1
s2 = sub2 + e2
```

LCS此时可分成四种情况：

- 一、 LCS 包含e1 ，不含e2 ；二、包含e2 ，不含e1 ；三、不含e1 e2 ；四、包含e1 e2 。第一种情况。e2 毫无用处，想找LCS 只需要找sub2 。得到结论LCS(s1, s2) = LCS(s1, sub2) 。
- 第二种情况。LCS(s1, s2) = LCS(sub1, s2) 。
- 第三种情况。LCS(s1, s2) = LCS(sub1, sub2) 。
- 第四种情况。LCS 同时包含e1 e2 ，而且e1 e2 位于尾端。因此LCS 的最后一个元素一定是e1 （ 也是e2 ），而且e1 e2 相等！得到结论LCS(s1, s2) = LCS(sub1, sub2) + e1 。

总合以上分析，可以得到递推公式：

```
LCS(s1, s2) =
 { LCS(sub1, s2) or LCS (s1, sub2) or LCS(sub1, sub2) , when
e1 != e2
 { LCS(sub1, sub2) + e1 , when e1 == e2
```

经过简化：

```
LCS(s1, s2) =
 { max( LCS(sub1, s2), LCS(s1, sub2) ) , when e1 != e2
 { LCS(sub1, sub2) + e1 , when e1 == e2
```

也就得到了：

$$\begin{cases} 0 & \text{若 } i=0 \text{或} j=0 \end{cases}$$

$$C[i,j]=\begin{cases} C[i-1,j-1]+1 & 若\ i,j>0, x_i=y_j \\ \max\{C[i,j-1],C[i-1,j]\} & 若\ i,j>0, x_i\neq y_j \end{cases}$$

1.动态规划方法

二维阵列length[i][j] ，代表**s1 前i 个元素**和 **s2 前j 个元素**的LCS 长度。

图解矩阵的填充过程：

第一步：初始化矩阵，也就是i==0或者j==0时，C[i][j]=0

下标j / 下标i		0	1	2	3	4	5	6	7	8	9
	$S2_j$	3	5	7	4	8	6	7	8	2	
0	$S1_i$	0	0	0	0	0	0	0	0	0	0
1	1	0									
2	3	0									
3	4	0									
4	5	0									
5	6	0									
6	7	0									
7	7	0									
8	8	0									

http://blog.csdn.net/

第二步：按行从上往下填

如果s1[i]==s2[j],则C[i][j]的值来自左上角：C[i][j]=C[i-1][j-1]+1

如果s1[i]!=s2[j],则C[i][j]=MAX(C[i-1][j],C[i][j-1])

例如C[1][2]=C[0][1]+1=1，C[2][2]=MAX(C[1][2],C[2][1])=1

下标i		下标j	0	1	2	3	4	5	6	7	8	9
		$S2_j$	3	5	7	4	8	6	7	8	2	
0	$S1_i$		0	0	0	0	0	0	0	0	0	0
1	1		0	0	0	0	0	0	0	0	0	0
2	3		0	1	1	1	1	1	1	1	1	1
3	4		0									
4	5		0									
5	6		0									
6	7		0									
7	7		0									
8	8		0									

http://blog.csdn.net/

将矩阵填充完，可得到：

下标i		下标j	0	1	2	3	4	5	6	7	8	9
		$S2_j$	3	5	7	4	8	6	7	8	2	
0	$S1_i$		0	0	0	0	0	0	0	0	0	0
1	1		0	0	0	0	0	0	0	0	0	0
2	3		0	1	1	1	1	1	1	1	1	1
3	4		0	1	1	1	2	2	2	2	2	2
4	5		0	1	2	2	2	2	2	2	2	2

http://blog.csdn.net/

5	6	0	1	2	2	2	2	3	3	3	3
6	7	0	1	2	3	3	3	3	4	4	4
7	7	0	1	2	3	3	3	3	4	4	4
8	8	0	1	2	3	3	4	4	4	5	5

最终的C[s1.length()-1][s2.length()-1]就是最长的长度(这里为C[8][9]=5)。LCS的解可能存现多个长度一样的子序列，现在，我们的目标不仅仅得到最长的长度，还需要得到其中一个最长的序列。我们可以根据C[8][9]倒推回去，得到一个LCS序列。

如果遇到S1[i] != S2[j] ，且c[i-1][j] = c[i][j-1] 这种存在分支的情况，可以随便选择一个方向。例如，相等时，我们都选择左边的元素，可以得到LCS = {3,4,6,7,8}这一个结果：

下标i	下标j	0	1	2	3	4	5	6	7	8	9
	$S2_j$	3	5	7	4	8	6	7	8	2	
0	$S1_i$	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	3	0	**1**	**1**	**1**	1	1	1	1	1	1
3	4	0	1	1	1	**2**	2	2	2	2	2
4	5	0	1	2	2	**2**	**2**	2	2	2	2
5	6	0	1	2	2	2	2	**3**	3	3	3
6	7	0	1	2	3	3	3	**3**	4	4	4
7	7	0	1	2	3	3	3	3	**4**	4	4
8	8	0	1	2	3	3	4	4	4	**5**	**5**

http://blog.csdn.net/

选择不同的方向将得到不同的结果。

时间复杂度分析

构建c[i][j]表需要Θ(mn)，输出1个LCS的序列需要Θ(m+n)。

代码

main

```
vector<int> longestCommonSubsequence(vector<int> a, vector<int> b) {

    int m=a.size(),n=b.size();
    vector<vector<int>>dp(m+1,vector<int>(n+1,0));
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(a[i]==b[j])
            {
                dp[i+1][j+1]=dp[i][j]+1;
            }
            else
            {
                dp[i+1][j+1]=max(dp[i][j+1],dp[i+1][j]);
            }
        }
    }
    vector<int>res(dp[m][n],0);
    int idx=dp[m][n]-1;
    int i=m-1,j=n-1;
    while (i>=0&&j>=0)
    {
        if(a[i]==b[j])
        {
            res[idx--]=a[i];
            i--;
```

```
35              j--;
36          }
37          else
38          {
39              if(dp[i][j+1]>=dp[i+1][j])
40              {
41                  i--;
42              }
43              else
44              {
45                  j--;
46              }
47          }
48      }
49      return res;
50  }
```

2.减小空间复杂度(只要求LCS的长度)

如果只想求出LCS 的长度，而不需要求出LCS ，那么有个节省记忆体空间的方法。

填dp数组，只需要上方、左方、左上方的格子。计算顺序设定成由左到右、再由

上到下，那么二维dp数组简化成一维的，然后暂存左上方的格子的值。

3.Hirschberg's Algorithm

http://en.wikipedia.org/wiki/Hirschberg's_algorithm

从中央分割，节省空间。空间复杂度为O(min(X,Y)) 。也有人省略了min 函数，直接写成O(N) 。

4.Hunt-Szymanski Algorithm(转化成LIS问题)

从LCS 问题的DP 表格可以观察到， LCS 问题可以化作类似于LIS 的问题：从两序列中找出对应的相同元素，以(**位置-数)对**表示；这些(位置-数)对可以排出的**最长严格递增序列**，即是两序列的LCS 。

比如现在我们要找A、B的最长公共子序列，其中
A：1 3 9 2 3
B：3 2 1 7 2 3
我们要在A中找到B中的各个字符出现的位置，
3：2 5
2 ：4

1：1
7：没有
2：4
3：2 5

然后依次把右边的数写出来，但是对于每个「：」后面的一组数要逆序输出，因为在B中那个位置只有一个元素，所以要避免在做最长上升子序列的时候在那一个位置选择了多于一个的元素。
这样我们就得到了5 2 4 1 4 5 2，对这个序列求最长(严格)上升子序列即可。

LIS 问题与LCS 问题可以互相转换

LIS 转LCS ：令原序列A 排序后会变成B 。A 和B 的LCS ，就是A 的LIS 。

LCS 转LIS ：将序列A 和B 当中的相同字母配对都找出来，呈现成索引值数对，再以特殊规则排序，最后找LIS ，就是A 和B 的LCS

参考资料

https://blog.csdn.net/hrn1216/article/details/51534607
http://www.csie.ntnu.edu.tw/~u91029/LongestCommonSubsequence.html#7