最长公共子序列问题及其变体

原创 万文俊 LOA算法学习笔记 2021-01-30 19:45

01 问题描述

给定两个序列，求最长公共子序列（Longest Common Subsequence）的长度

注：子序列不同于子串，因为前者不必与原始序列连续。例如，对于“ABCD”和“AEFC”，最长的子序列为“AC”，长度为2。

02 问题分析

1）暴力搜索：如果只匹配两个序列的话，可以用暴力搜索来解决，假设A串的长度是m，B串的长度是n，那么A的子序列个数有 2^m 个，B的子序列个数有 2^n 个，若对每一种情况进行匹配，时间复杂度为 $O(2^m \cdot 2^n)$ ，显然随字符串长度指数式增长不是我们想要的。

2）动态规划：

分析：假设用i表示字符串A中元素的下标，用j表示字符串B中元素的下标，当第i，j位置的字符相匹配时，则当前的LCS是之前的LCS长度加一；而当不匹配时，LCS长度不会变化，此时面临两种选择，一是A串第i位置，B串第j-1位置的LCS，二是A串第i-1位置，B串第j位置的LCS，取二者之间较大的那个。

定义状态：OPT[i][j]表示的是A串第i位置，B串第j位置以前的两个序列的最大的LCS长度，这其中OPT[0][0]=0，OPT[m][n]则是我们要求的解。

状态转移方程：

$$OPT[i][j]=\begin{cases}\max(OPT[i-1][j],OPT[i][j-1]) & i,j>0,S[i]\neq T[j]\\ OPT[i-1][j-1]+1 & i,j>0,S[i]=T[j]\\ 0 & i=0,j=0\end{cases}$$

时间复杂度为 $O(mn)$ ，随字符串长度线性增长。

		a	a	b	a	a	c	a	a
	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1
a	0	1	2	2	2	2	2	2	2
c	0	1	2	2	2	2	3	3	3
a	0	1	2	2	3	3	3	4	4

03 算法设计

```cpp
#include<iostream>
using namespace std;
int main(void){
    string s1,s2;
    cin>>s1>>s2;

    int opt[s2.size()+1][s1.size()+1];//定义一个（A串长度+1）*（B串长度+1）大小的整型数组

    for(int i=0;i<=s2.size();i++)opt[i][0]=0;//对第一列初始化为0
    for(int j=0;j<=s1.size();j++)opt[0][j]=0;//对第一行初始化为0
    for(int i=1;i<=s2.size();i++){
        for(int j=1;j<=s1.size();j++){
            if(s2[i-1]==s1[j-1])opt[i][j]=opt[i-1][j-1]+1;//若相等，则对于对角线值加一
            else opt[i][j]=max(opt[i-1][j],opt[i][j-1]);
        }
    }

    cout<<opt[s1.size()][s2.size()]<<endl;
    return 0;
}
```

04 空间优化

```cpp
#include<iostream>
using namespace std;
int main(void){
    string s1,s2;
    cin>>s1>>s2;

    int opt[s2.size()+1];//定义一个（B串长度+1）大小的整型数组

    for(int i=0;i<=s2.size();i++)opt[i]=0;//初始化为0

    for(int i=1;i<=s1.size();i++){
        int old = opt[0];// 替换了原先的opt[i-1][0]
        for(int j=1;j<=s2.size();j++){

            if(s1[i-1]==s2[j-1])opt[j]=old+1;
            else opt[j]=max(opt[j-1],opt[j]);
            //即原先的dp[i][j]
            old = opt[j];//即原先dp[i-1][j]的值
        }
    }

    cout<<opt[s2.size()]<<endl;
    return 0;
}
```

05 拓展问题

1) 最长公共子串（Longest Common Substring）

最长公共子串需要字符之间位置是连续的。

分析：先分析不匹配的情况，LCS长度不会变化，但由于要求字符间位置连续，所以当前的字符不能利用之前的LCS，只能为0；然后分析匹配的情况，如果匹配则看前一个字符的LCS长度是多少，在其基础上加一。

定义状态：OPT[i][j]表示的是A串第i位置，B串第j位置以前的两个序列的最大的LCS，这其中OPT[0][0]=0，与最长公共子序列不同的是在求得OPT[m][n]后需对所有的OPT[i][j]求最大。

状态转移方程：

$$OPT[i][j]=\begin{cases} OPT[i-1][j-1]+1 & i,j>0, S[i]=T[j] \\ 0 & i,j>0, S[i]\neq T[j] \\ 0 & i=0, j=0 \end{cases}$$

2) 拆分回文串

回文串是指字符串正着读和反着读一样，例如“refer”，拆分回文串就是指给定一个字符串，切割成一些子串，使每个子串都是回文串。

问：最少的切割次数。例：S=“aabaacaa”，答案是2，因为可以拆成[“aabaa”,“c”,“aa”]或者[“aabaa”,“c”,“aa”]

分析：乍一看好像跟上述两种LCS问题都没有什么关系。

不妨从最简单的case入手：

a）S=“aab”，显然只要切割一次,如果把s倒过来，也就是S′=“baa”，如果要形成回文串，说明该字符串倒过来和原来是一样的，既然是找连续且相同的，那么就回到了找最长公共子串的问题上，那么再找个稍微难一点的case验证一下。

b）S=“aabaacaa”,初始化切割次数为0，将原字符串倒过来之后就是S′=“aacaabaa”，按照最长公共子串的方法，先找到最长的公共子串“aabaa”，切割次数加1，在原字符串中删除“aabaa”后得到新的S=“caa”，颠倒后得到S′=“aac”，找到最长公共子串“aa”，切割次数加1，最后只剩下一个字符，不用切割，所以总次数是2。

由以上两个case可以看出拆分回文串可以用最长公共子串的方式来求解，不妨画图来表示。

先按照正常的求解最长公共子串方式来，回溯找到子串“aabaa”

		a	a	b	a	a	c	a	a
	0	0	0	0	0	0	0	0	0
a	0	1	1	0	1	1	0	1	1
a	0	1	2	0	1	1	0	1	2
c	0	0	0	0	0	0	1	0	0
a	0	**1**	1	0	1	1	0	2	1
a	0	1	**2**	0	1	2	0	1	3
b	0	0	0	**3**	0	0	0	0	0
a	0	1	1	1	**4**	1	0	1	1
a	0	1	2	0	2	**5**	0	1	1

此时输出该LCS，原字符串中删除该LCS后，此时字符串只剩下“caa”，利用同样的方法即可求解剩下的LCS。

		a	a	b	a	a	c	a	a
	0	0	0	0	0	0	0	0	0
a	0	1	1	0	1	1	0	**1**	1
a	0	1	2	0	1	1	0	1	**2**
c	0	0	0	0	0	0	1	0	0
a	0	**1**	1	0	1	1	0	2	1
a	0	1	**2**	0	1	2	0	1	3
b	0	0	0	**3**	0	0	0	0	0
a	0	1	1	1	**4**	1	0	1	1
a	0	1	2	0	2	**5**	0	1	1

当然上述方法稍显复杂，因为笔者想和LCS问题找通解，求解该问题应该还有更好的方法，希望读者朋友可以在评论区不吝赐教。

06 总结

LCS问题也是动态规划中比较经典的问题，其应用较为广泛，比如碱基对匹配、近义词识别等等。求解时可以在给出状态转移方程后可以画出表格，再根据表格的规律可以尝试空间上的优化。