C++方向编程题答案

第七周

day45

> 题目ID：25946 字符串计数
>
> 链接：https://www.nowcoder.com/questionTerminal/f72adfe389b84da7a4986bde2a886ec3
>
> **【题目解析】**
>
> 题目意思：按照字典序列：找到s1和s2之间长度在len1和len2范围内的字符串个数。直接做不好处理，此处需要转化思路，找到一个合适的模型：因为从'a'~'z'，刚好有26个字母，因此可以将s1和s2看成是26进制数据，题目就变得简单了，将其转化为：从s1和s2之间有多少个不同数字，最后求解出长度不同的数组的个数即可。
>
> **【解题思路】**
>
> 1. 循环接受收入，保证所有测试用例可以验证到
> 2. 将s1和s2补齐到26位，因为在字典序列中s1比s2靠前，因此s1后序所有位补'a'，s2后补'z'+1
> 3. 确认s1和s2两个字符串每个字符位置上的差值
> 4. 确认len1和len2之间不同字符的个数

```
/*
补齐字符串，按照26进制进行计算
*/
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#define N 1000007
#include<math.h>
using namespace std;

int main(){
    //根据题中给出的例子，这个字符串只包含小写字母，不然答案就不应该是56了
    string s1,s2;
    int len1,len2;
    while(cin>>s1>>s2>>len1>>len2){
        //只包含小写字母的字符串可以看成26进制的数制
        //将s1和s2补长到len2长度
        s1.append(len2-s1.size(),'a');
        s2.append(len2-s2.size(),(char)('z'+1));

        // 确认s1和s2的两个字符串每个位置上的差值
        vector<int> array;
        for(int i=0;i<len2;i++){
            array.push_back(s2[i]-s1[i]);
        }

        // 确认len1和len2之间可组成的不同字符串的个数
```

```
        int result = 0;
        for(int i=len1;i<=len2;i++){
            for(int k=0;k<i;k++){
                result += array[k]*pow(26,i-1-k);
            }
        }
        //所有字符串最后都不包含是s2自身，所以最后要减1;
        cout<<result-1<<endl;
    }
    return 0;
}
```

806 最长公共子序列

链接：https://www.nowcoder.com/questionTerminal/9ae56e5bdf4f480387df781671db5172

【题目解析】

题目要求比较简单：**获取两个字符串的最长公共的子序列**，注意：**子序列即两个字符串中公共的字符，但不一定连续。**

【解题思路】

动态规划求解最长公共子序列(LCS): 对于**母串X=<x1,x2,⋯,xm>, Y=<y1,y2,⋯,yn>**，求LCS，假设：假设Z=**<z1,z2,⋯,zk>是X与Y的LCS**， 我们观察到：

1. **如果xm=yn，则zk=xm=yn，有Zk−1是Xm−1与Yn−1的LCS**
2. **如果xm≠yn，则Zk是Xm与Yn−1的LCS，或者是Xm−1与Yn的LCS**

因此，求解LCS的问题则变成递归求解的两个子问题。但是，上述的递归求解的办法中，重复的子问题多，效率低下。改进的办法——用空间换时间，用数组保存中间状态，方便后面的计算。这就是动态规划（DP)的核心思想了。

用二维数组dp[i,j]记录串x1x2⋯xi与y1y2⋯yj的LCS长度，则可得到状态转移方程：

1. dp[i,j]=0i=0 or j=0
2. dp[i,j]= dp[i−1,j−1]+1 i,j>0 and xi=yj
3. dp[i,j]=max(dp[i,j−1], dp[i−1,j]) i,j>0 and xi≠yj

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    string A, B;
    while(cin >> A >> B) {
        int aLength = A.length();
        int bLength = B.length();


        // 用来保存状态转移方程中间结果的矩阵
```

```
        vector<vector<int> > dp(aLength, vector<int>(bLength, 0));

        // 初始化dp矩阵边界：边界字符要么在，要么不在
        dp[0][0] = (A[0] == B[0])?1:0;
        for(int i=1; i<aLength; i++) {
            dp[i][0] = (A[i] == B[0]) ? 1 : 0;
            dp[i][0] = max(dp[i-1][0], dp[i][0]);
        }

        for(int j=1; j<bLength; j++) {
            dp[0][j] = (A[0] == B[j]) ? 1 : 0;
            dp[0][j] = max(dp[0][j-1], dp[0][j]);
        }

        // 根据状态转移方程进行计算
        for(int i=1; i<aLength; i++) {
            for(int j=1; j<bLength; j++) {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
                if(A[i] == B[j]) {
                    dp[i][j] = max(dp[i][j], dp[i-1][j-1]+1);
                }
            }
        }
        cout << dp[aLength-1][bLength-1] << endl;
    }

    return 0;
}
```