问题描述:

> 求两个序列中最长的公共子序列

算法思想:

O(n^2)

使用动态规划 , dp[i][j] 代表 s1以i为结尾,s2以j为结尾,得到的LCS的最长长度

我们需要找出子状态 , 根据dp[i][j]的定义,那么它的子状态只有

- dp[i-1][j]
- dp[i][j-1]
- dp[i-1][j-1]

所以我们可以得到状态转移方程

> $$dp[i][j] = \begin{cases} max(dp[i-1][j], dp[i][j-1]), & s1[i] != s2[j] \\ dp[i-1][j-1] + 1, & s1[i] = s2[j] \end{cases}$$

根据定义 , dp[i][j]初始化为0 , 然后双重循环.从小到大遍历

输出路径: 在状态转移时做标记 , 然后通过回溯法带路 , 返回的时候输出路径

O(nlogn)

利用LIS的nlogn算法(二分查找)

假设有两个序列 s1[1~6] = { a, b, c , a, d, c }, s2[1~7] = { c, a, b, e, d, a, b }。

记录s1中每个元素在s2中出现的位置, 再将位置按降序排列,

则上面的例子可表示为： loc(a) = { 6, 2 }, loc(b) = { 7, 3 }, loc(c) = { 1 }, loc(d) = { 5 }。

将s1中每个元素的位置按s1中元素的顺序排列成一个序列s3 = { 6, 2, 7, 3, 1, 6, 2, 5, 1 }。

在对s3求LIS得到的值即为求LCS的答案。

求解过程:

```
#include<bits/stdc++.h>
using namespace std;
char a[1005],b[1005];
int dp[1005][1005],mark[1005][1005];

void print_lcs( int i , int j )   //输出路径
{
    if(!i&&!j)      //出口(返回条件)
        return;
    else
    {
        if(mark[i][j]==0)
        {
            print_lcs(i-1,j-1);
            printf("%c",a[i]);
        }
        else if(mark[i][j]==1)
        {
            print_lcs(i-1,j);
        }
        else
        {
            print_lcs(i,j-1);
        }
    }
}

int main()
{
    while(~scanf("%s%s",a+1,b+1))
    {
        int i,j;
        int lena= strlen(a+1) , lenb= strlen(b+1);
        //cout<<lena<<" "<<lenb<<endl;
        for(i=0 ;i<=lena ; i++)
            mark[i][0] = 1 ;//不能进dp[i][j-1]的口子, 不然j<0
        for(i=0 ;i<=lenb ;i++)
            mark[0][i] = -1;

        for(i=1 ;i<=lena ; i++)
        {
            for(j=1 ;j<=lenb ;j++)
            {
                if(a[i]==b[j])
                {
                    dp[i][j] = dp[i-1][j-1]+1;
                    mark[i][j] = 0;
                }
                // 由最优的子状态来更新自己
                else if(dp[i][j-1]>dp[i-1][j])
                {    dp[i][j] = dp[i][j-1];
                    mark[i][j] = -1;
                }
                else
                {    dp[i][j] = dp[i-1][j];
```

```
                    mark[i][j] = 1;
                }
            }
        }
        print_lcs(lena,lenb);
        printf("\n");

    }
    return 0;
}

/*  O(nlogn)

    for(int i = 0; i < n; i ++)
    {
        if(len == 0 ||a[i] > lis[len - 1])
        {
            lis[len] = a[i];
            len ++;
        }
        else
        {
            p = lower_bound(lis,lis +len,a[i]) - lis;
            lis[p] = a[i];  //二分查找属于a[i]的位置
        }
    }


*/
```