Experiment—LCS problem

PB18010496 杨乐园

Introduction

编程实现最长公共子序列（LCS）算法，并理解其核心思想，编写相关算法分别符合如下有关空间复杂度以及时间复杂度的要求：

1. 时间复杂度$O(mn)$，空间复杂度 $O(mn)$，求出LCS及其长度。
2. 时间复杂度$O(mn)$，空间复杂度$O(2*min(m,n))$，求出LCS的长度。
3. 时间复杂度$O(mn)$，空间复杂度$O(min(m,n))$，求出LCS的长度 。

Purpose

实验目的：熟悉并掌握动态规划的算法设计思想，并进一步实现相关空间复杂度的优化。

Idea

在求解$X=<x_1,...,x_m>$和$Y=<y_1,...,y_n>$的一个LCS时，考虑分两种情况：如果$x_m=y_n$，则我们求解X_{m-1}和Y_{n-1}的LCS，并将x_m追加到末尾即可；若$x_m\neq y_n$，则考虑求解两个子问题，即X_{m-1}和Y的一个LCS与X和Y_{n-1}的LCS，其中二者较长者即为原问题的LCS。从而若定义$c[i,j]$为X_i与Y_j的LCS的长度，则由其最优子结构性质，可得如下公式：

$$c[i,j]=\begin{cases}0 & if \quad i=0 \quad or \quad j=0\\ c[i-1,j-1]+1 & if \quad i,j>0 \quad and \quad x_i=y_j\\ max(c[i,j-1],c[i-1,j]) & if \quad i,j>0 \quad and \quad x_i\neq y_j\end{cases}$$

从而依据如上最优子结构，我们可以结合动态规划给出相应的递归算法。

设$c[0,...,m][0,...,n]$用来存放最优解值，计算时行优先；设$b[1,...,m][1,...,n]$为解矩阵，用来存放构造最优解信息。其中$b[i,j]$的计算公式为：

$$b[i,j]=\begin{cases}\nwarrow & \text{若}c[i,j]\text{由}c[i-1,j-1]\text{决定}\\ \uparrow & \text{若}c[i,j]\text{由}c[i-1,j]\text{决定}\\ \leftarrow & \text{若}c[i,j]\text{由}c[i,j-1]\text{决定}\end{cases}$$

从而当构造解时，从$b[m,n]$出发，上溯至$i=0$或$j=0$为止，上溯过程中，当$b[i,j]$包含"$\nwarrow$"时打印出对应的x_i或y_j即可构造出对应LCS。此时时间复杂度为$O(mn)$，空间复杂度为$O(mn)$。

对于优化存储空间，可以看到，计算$c[i][j]$时只与$c[][]$的第i行与第j行有关，从而本质上可以只用$O(2min(m,n))$储存当下正在计算的该行与上一行这两行信息即可，从而将空间复杂度直接降到了$O(2min(m,n))$。

再者更进一步，我们可以注意到，计算$c[i][j]$时只与$c[i-1][j]$或$c[i][j-1]$有关，我们进行比较时将三种情况依次考虑，从$c[i][j]$只由$c[i-1][j-1]$决定，到由$c[i-1][j]$决定，最后考虑由$c[i][j-1]$决定，这样在给$c[i][j]$更新值时，前面所需要的信息都未曾改变，无需新的额外的存储空间，这样空间复杂度就降到了$O(min(m,n))$。

Algorithm

首先我们采取最普遍的动态规划，即空间复杂度为$O(mn)$，并做到同时计算出LCS的长度与构造：

```
//求最大公共子序列；时间复杂度为O(mn)，空间复杂度为O(mn)
void LCS_mn(string x, string y, vector<vector<int>>& b, vector<vector<int>>& c)
{
    for (int i = 1; i <= size(x); i++)
    {
        for (int j = 1; j <= size(y); j++)
        {
            if (x[i - 1] == y[j - 1])
                c[i][j] = c[i - 1][j - 1] + 1, b[i][j] = upleft;
            else if (c[i - 1][j] >= c[i][j - 1])
                c[i][j] = c[i - 1][j], b[i][j] = up;
            else
                c[i][j] = c[i][j - 1], b[i][j] = left;
        }
    }
}

//输出最大公共子序列；
char printLCS(vector<vector<int>> b, string x, int i, int j)
{
    if (i == 0 || j == 0)
        return ' ';
    if (b[i][j] == upleft)
        printLCS(b, x, i - 1, j - 1), cout << x[i - 1];
    else if (b[i][j] == up)
        printLCS(b, x, i - 1, j);
    else
        printLCS(b, x, i, j - 1);

}
```

其次我们实现第一步优化，将空间复杂度降为$O(2*min(m,n))$：

```
//求最大公共子序列长度；时间复杂度为O(mn)，空间复杂度为O(2min{m,n})
int LCS_2n(string x, string y)
{
    if (size(x) < size(y))//取最小的字符串放在y里面；
    {
        string temp = y;
        y = x;
        x = temp;
    }
    vector<vector<int>> c(2, vector<int>(size(y) + 1, 0));
    for (int i = 1; i <= size(x); i++)
    {
        for (int j = 1; j <= size(y); j++)
        {
            if (x[i - 1] == y[j - 1])
                c[i % 2][j] = c[(i + 1) % 2][j - 1] + 1;
            else if (c[(i + 1) % 2][j] >= c[i % 2][j - 1])
                c[i % 2][j] = c[(i + 1) % 2][j];
```

```
            else
                c[i % 2][j] = c[i % 2][j - 1];
        }
    }
    return c[size(x) % 2][size(y)];
}
```

最后，我们再进一步优化，将空间复杂度降为$O(min(m,n))$：

```
//求最大公共子序列长度；时间复杂度为O(mn)，空间复杂度为O(min{m,n})
int LCS_n(string x, string y)
{
    if (size(x) < size(y))//取最小的字符串放在y里面；
    {
        string temp = y;
        y = x;
        x = temp;
    }
    vector<int> c(size(y) + 1, 0);
    for (int i = 1; i < size(x) + 1; i++)
    {
        int temp = c[0];
        for (int j = 1; j < size(y) + 1; j++)
        {
            if (x[i - 1] == y[j - 1])
            {
                int tempp = c[j];
                c[j] = temp + 1;
                temp = tempp;
            }
            else if (c[j] >= c[j - 1])
                temp = c[j];
            else
                temp = c[j], c[j] = c[j - 1];
        }
    }
    return c[size(y)];
}
```

Results

通过运行程序与测试数据，我们有如下输出结果：

```
请输入第一个字符串：This is a sentence. (###***9032821)
请输入第二个字符串：This is an ambiguous sentence. (@kdwk324%kds^kdx9)

空间复杂度为O(mn)的算法，返回LCS的长度与序列：
LCS: This is a sentence. (32)
长度：24

空间复杂度为O(2*min{m,n})的算法，返回LCS的长度与序列：
长度：24

空间复杂度为O(min{m,n})的算法，返回LCS的长度与序列：
长度：24



是否继续查找LCS，若是输入任何非0数，若否输入0结束：0
```

我们可以看到，输出结果正确。

Code

具体完整代码，参看附件文件LCS。