Lab 0 实验报告

2021 年 10 月 7 日

1 问题描述

求解最长公共子序列，按字母序输出所有结果。

2 解题思路

用动态规划求解最长公共子序列长度，再根据状态转移方程反推输出子序列, 输出要求字母序且不重复，使用 set 结构存储结果。

3 状态定义

$dpTable[i][j]$ 表示 $x[0:i]$ 与 $y[0:j]$ 的最长公共子序列长度，$x[0]$ 和 $y[0]$ 定义为空字符，意在 $dpTable$ 初始化时为 0。

4 状态转移方程

- 若 $x[i-1]=y[j-1]$，表示 x 和 y 最后一位相等，则 $dpTable[i][j]=dpTable[i-1][j-1]+1$；
- 若 $x[i-1]\neq y[j-1]$，则 $dpTable[i][j]$ 为 $dpTable[i-1][j]$ 与 $dpTable[i][j-1]$ 中的最大值。

于是可得状态转移方程如下：

$$dpTable[i][j]=\begin{cases}0 & i==0\ or\ j==0\\ max(dpTable[i-1][j],dpTable[i][j-1]) & x[i-1]\neq y[j-1]\\ dpTable[i-1][j-1]+1 & x[i-1]==y[j-1]\end{cases}$$

由此编写动态规划部分代码如下：

```
int lcs(string a, string b, int m, int n) {
  dpTable = vector<vector<int>>(m + 1, vector<int>(n + 1));
  for (int i = 0; i < m + 1; ++i) {
    for (int j = 0; j < n + 1; ++j) {
      if (i == 0 || j == 0)
        dpTable[i][j] = 0;
      else if (a[i - 1] == b[j - 1])
        dpTable[i][j] = dpTable[i - 1][j - 1] + 1;
      else
        dpTable[i][j] = max(dpTable[i - 1][j], dpTable[i][j - 1]);
    }
  }
  return 0;
}
```

时间复杂度为 $O(mn)$，for 循环执行 $(m+1)\times(n+1)$ 颗粒时间；
空间复杂度为 $O(mn)$，生成表格大小为 $(m+1)\times(n+1)$；
通讯复杂度为 $O\left(\frac{mn}{B}\right)$，内层循环 $cachemiss$ 为 $\left(\frac{n}{B}+1+\frac{n}{B}\right)$；
程序优化方法：将传统递归解法优化为动态规划打表。

5 结果输出

根据状态转移方程反推可得子序列，结果输出部分代码如下：

```
int getlcs(int i, int j, string str) {
  while (i > 0 && j > 0) {
    if (x[i - 1] == y[j - 1]) {
      str.push_back(x[i - 1]);
      --i;
      --j;
    }
    else if (dpTable[i - 1][j] > dpTable[i][j - 1])
      --i;
    else if (dpTable[i - 1][j] < dpTable[i][j - 1])
      --j;
    else {
      getlcs(i - 1, j, str);
```

```
            getlcs(i, j - 1, str);
            return 0;
        }
    }
    reverse(str.begin(), str.end());
    lcsSet.insert(str);
    return 0;
}
```

此处用尾递归解决路径分叉和降低时间复杂度，再用 set 结构去重以及保持字母升序。
程序正确性：能力范围内的测试均通过，其他的靠自信。