**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与实现**

**实验项目名称： 实验4动态规划 代码查重**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 杨烜老师**

**报告人： 李若龙 学号：2018171028 班级： 计科02**

**实验时间： 2020/5/17**

**实验报告提交时间：**

**教务处制**

1. 问题描述：

问题：根据给定重复比率阈值*r*与模板代码文件“*A*.cpp”*，*计算测试代码文件“*B*.cpp”的最多代码重复行数？

1. 设计动态规划算法求解文件“*A*.cpp”在第*i* 行代码与文件“*B*.cpp”在第*j*行代码的最长相同子代码模块问题，并写出求得最长相同子代码模块的递推公式。

2）对于任意给定的 *i*, *j*, 求出*S*(*i*,*j*)。根据给定重复比率阈值*r，*求出*D*(*i*,*j*)。

3）根据求出的D(i,j)，利用动态规划算法求出测试代码文件“B.cpp”的最多代码重复行数？写出求得最多行重复模板代码的递推公式。

4）设计变量名代换的不同程序代码查重（此问题为加分选项）。

1. 求解问题的算法原理描述

**求解两行代码相似度 ：LCS**

LCS即最长公共子序列，描述了两个序列之间公共的部分，LCS问题的求解，需要使用动态规划

**LCS 符号与子问题确定：**

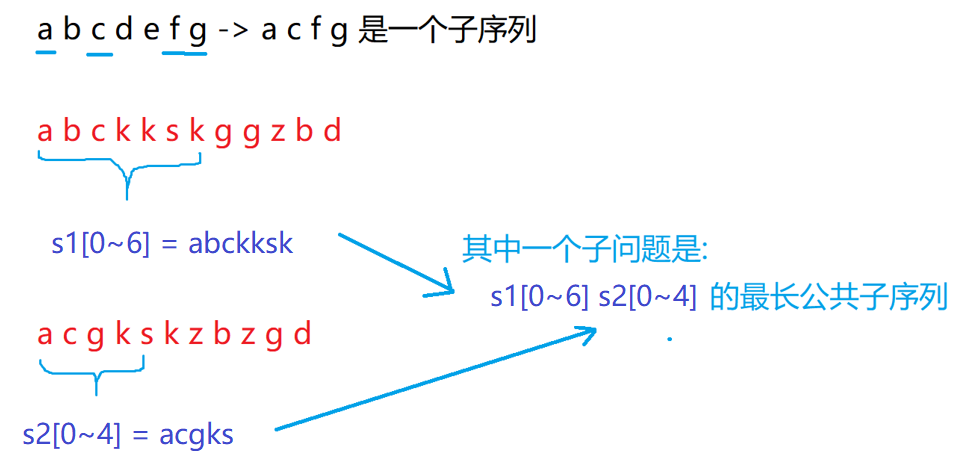
假设有序列 s1[] 和 s2[] ，其中s1长度为len1，s2长度为len2，那么子问题即求解s1 s2的任意不长于len1/len2的子序列（有多组）即是子问题

**如何描述LCS子问题？：**

s1[0~i]表示在s1中截取下标 0-i的子串

s2[0~j]表示在s2中截取下标 0-j的子串

求解 s1[0~i]与s2[0~j]的最长公共子序列，这个子问题的解存储在 dp[i][j]中



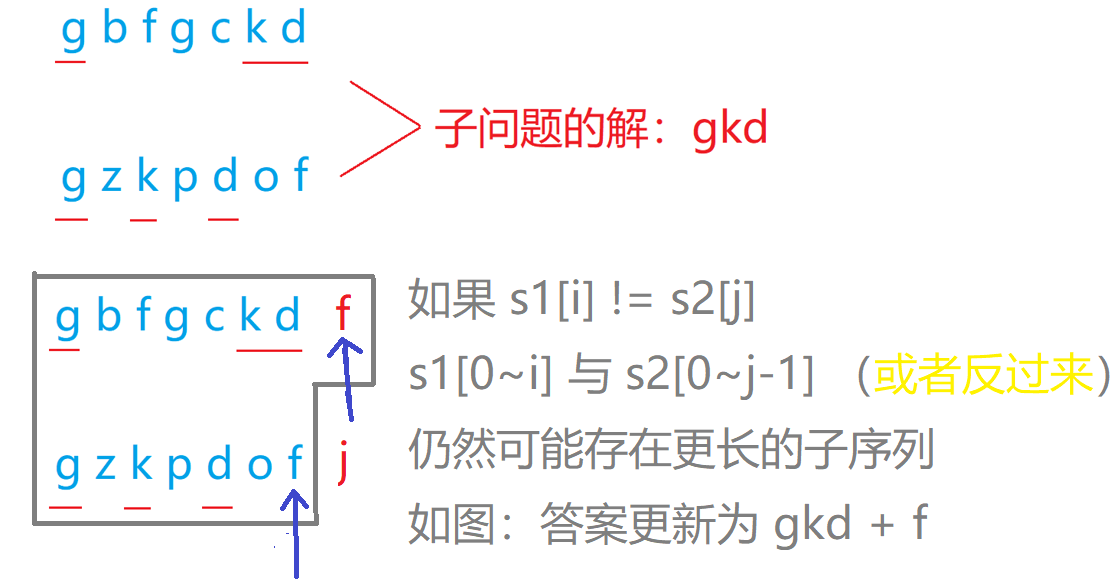
**LCS状态转移方程：**

现在欲求s1[0~i]和s2[0~j]的最长公共子序列长度（原问题），我们很自然想到匹配s1[i]和s2[j]处的字符 字符匹配不外乎两种情况：等/不等

我们考虑相等的情况：假设我们已知s1[0~i-1]和s2[0~j-1]的最长公共子序列为 gkd 如下图，因为当前字符相等，所以可加入末位，得到原问题的答案，即gkda



如果s1[i]和s2[j]不等，但是因为增加了一个字符，使得 s1[0~i]与s2[0~j-1] 或者反过来，s1[0~i-1]与s2[0~j]，存在延长相同子序列的可能



所以我们可以推出：

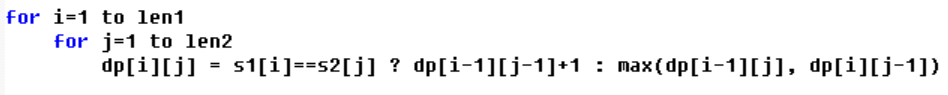
当s1[i] = s2[j]时，显然 dp[i][j] = dp[i-1][j-1]+1

当s1[i] ≠ s2[j]时，答案可能有两种：s1[0~i]与s2[0~j-1] 或者s1[0~i-1]与s2[0~j]的最长相同子序列长度，即 dp[i][j] = dp[i-1][j] 或者 dp[i][j-1]，因为求的是最长相同子序列，我们取最大，即 dp[i][j] = max(dp[i-1][j], dp[i][j-1])

**LCS问题： 最小子问题答案**

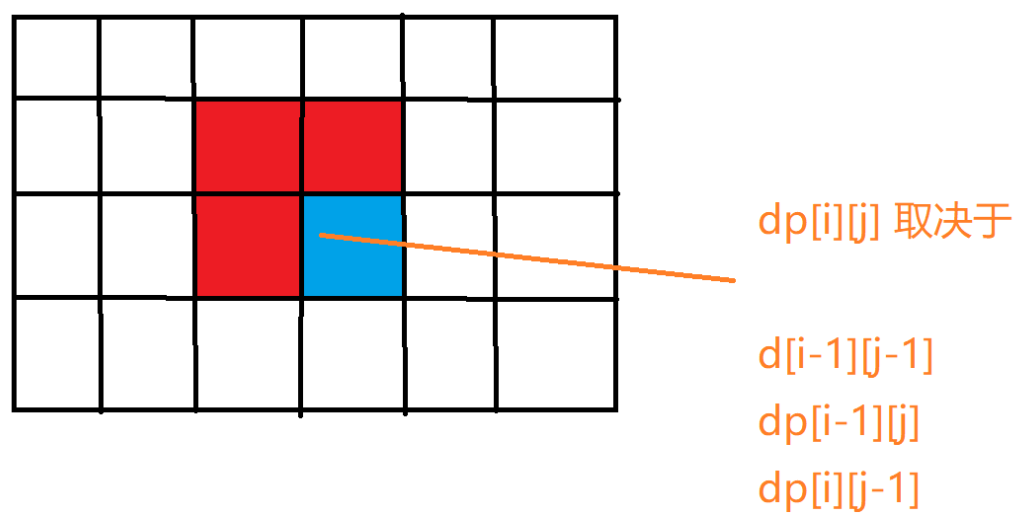
长度为0的序列和任意序列的LCS为0，即 dp[0][0] = dp[0][j] = dp[i][0] = 0

**LCS问题：伪代码**



**LCS问题：空间优化**

因为要填表所以lcs问题的空间复杂度是O(n^2)，但是因为lcs问题的递推式结构，当前问题的答案只来源于上一行和这一行的答案，我们只需要保存上一行的答案即可，即我们可以使用O(n)空间的复杂度完成

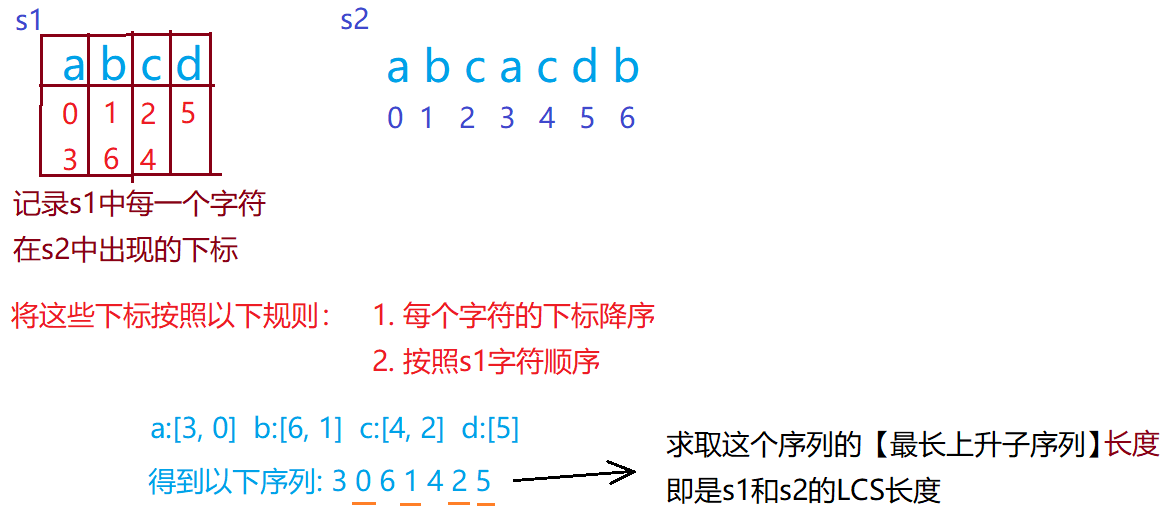


**LCS问题：时间优化**

先将LCS转化为LIS（最长上升子序列）问题

因为LCS问题本质是**在s1和s2中选取一定的下标序列，下标序列上升，即选取上升子序列，然后这些上升子序列对应的字符要一致**即可

那么我们统计s1中所有字符在s2中出现的下标，然后对每一个字符x，按照x出现的下标降序排序（防止一字多选），然后按照s1中的顺序拼接起来，得到下标序列，再对下标序列求LIS，LIS长度即为问题的答案



**LIS问题的求解：**

使用动态规划求解LIS问题，时间复杂度任然是O(n^2)，但是LIS可以使用贪心法求解。

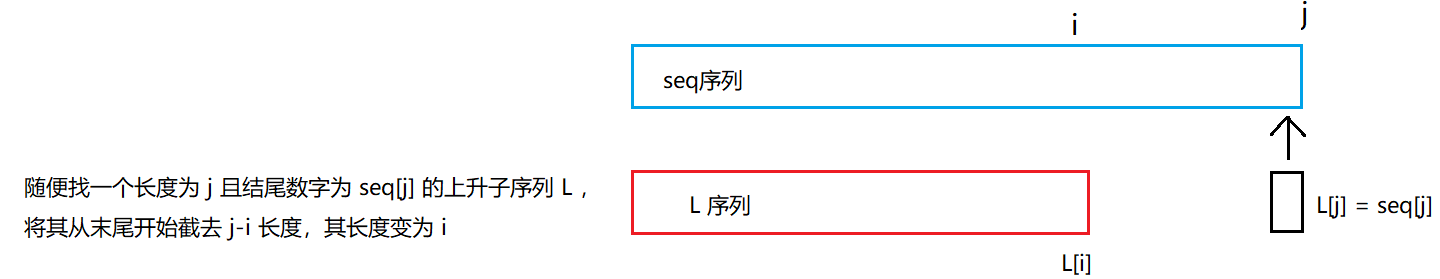
求解nums[]数组（即上文提到的下标序列）的LIS，通过维护序列seq[]

seq[i]表示nums[]数组中，长度为i+1的上升子序列的最小结尾数字

求解后，seq的长度即是LIS的长度 （请牢记这个定义，下文会反复用）

**铺垫：证明seq序列必定有序递增**

使用反证法，假设有 seq[i]>seq[j]且i<j 那么我们随便找一个长度为j+1，结尾数字为seq[j]的上升子序，列L，并且将L从末尾开始，截去j-i长度，这样l就变成i+1长度的上升子序列了



因为L是上升子序列，所有有 L[i] < L[j] = seq[j] < seq[i] ，那么我们就发现了一个长度为i+1的，结尾下标比 seq[i]要小的上升子序列，这和seq[]保存最小的结尾相矛盾

**维护序列seq方法**

1. 按照顺序遍历nums[]中的数字
2. 在seq中找到**第一个大于等于**nums[i]的数字seq[x]
3. 如果找到seq[x]，则用nums[i]替换seq[x]，找不到则将nums[i]加入seq末尾

**维护seq方法可行性证明：**

因为seq[x-1]必定小于nums[i](有序)，即存在长度为x的上升子序列，其结尾小于nums[i]

所以在上面那个长度为x的上升序列后面添加nums[i]，可以得到长度为x+1的上升序列

而这个新序列的结尾是nums[i]，小于原来的seq[x]，所以用nums[i]替换seq[x]

**LIS贪心法时间复杂度分析：**

因为seq序列有序，我们可以使用二分查找快速找到第一个大于nums[i]的seq[x]，遍历nums[]需要O(n)，每次遍历都二分查找seq序列，需要O(log(n))这使得算法的时间复杂度保持在O(nlog(n))

**LCS时间测试**

使用不同规模的随机字符串来测试，随机数字和字母



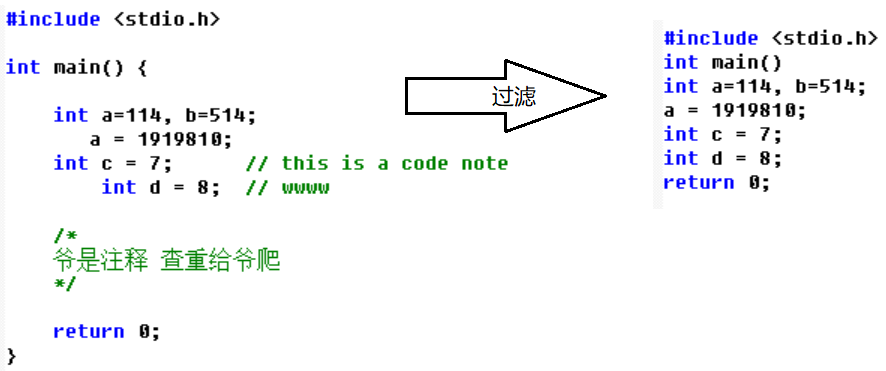
可以看到不同规模下，优化后的LCS速度远超常规LCS，但是因为随机字符的字符范围只有62个（字母+数字+大写字母），所以测试字符串的长度越长，同样的字符出现频率越高，越容易逼近优化LCS的最坏情况（即 s1=aaa s2=aaa 最怕一个字符在s2中出现多次），所以得出结论，LCS时间优化是一种不稳定的优化

它的下界是O(nlong(n))，其中n是s1 s2中短串的长度，而他的上界达到O(n^2 \* log(n^2))，但是在实际测试中，总体优化情况还可以接受

**回到查重问题 -- 行代码预处理：**

读取的代码不能够直接进行行LCS的计算的，因为不同人的编码习惯不同，采取的空格，回车，空行，TAB缩进，括号摆放的策略，注释习惯，都不同，行LCS结果会产生极大的误差，所以需要对输入的每一行代码进行过滤预处理：

去除前后空格，空行，TAB缩进，{}括号，去除注释// 和 /\*



**行代码LCS与查重：**

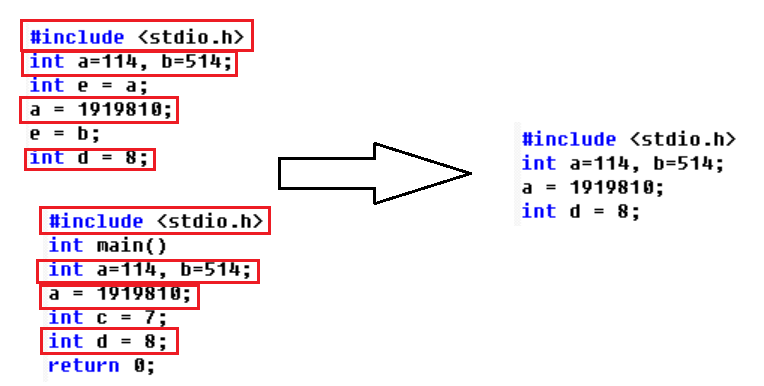
回到LCS查重问题，现在我们已经掌握了求取两行代码相似度的方法，即求解两行代码的LCS，并且得到相似矩阵D[][]，那么如何分析呢？

**每一行暴力匹配：**

将a.cpp每一行，在b.cpp中出现的重复行一一列举，得到所有重复行数，然后对b.cpp中的每一行，求在a.cpp中重复的次数，然后选取重复比例大的一方作为重复率

**以行为单位求LCS：**

将每一行看作一个元素，那么一篇代码任然是一个序列，D[I][J]存储行与行之间是否相等，所以可以求取行LCS，相似的行代码序列的长度是多少

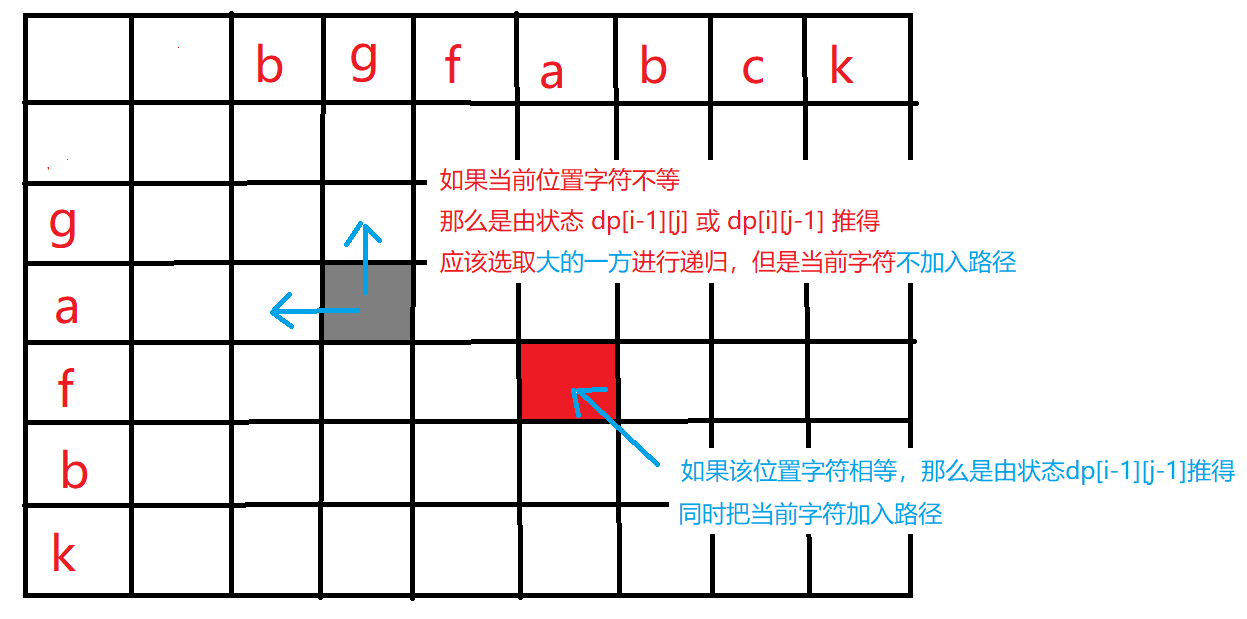


**LCS路径打印：**

根据LCS得到的二维数组dp[][]可以判断出当前状态是从何处推导过来的，从而求得递推路径

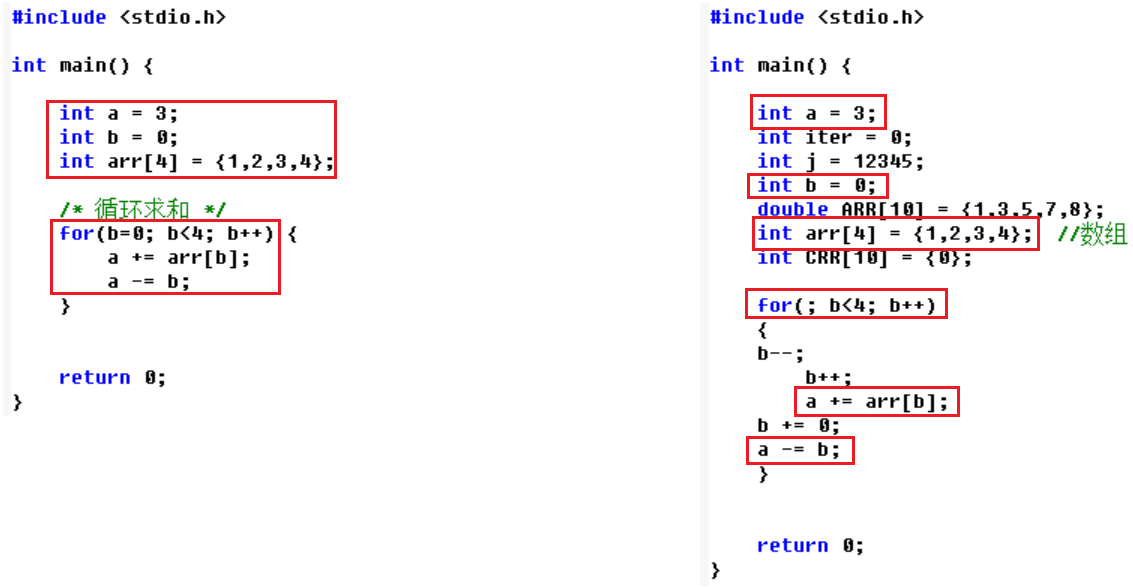
若当前位置字符相等，说明是由 i-1 j-1状态推得，所以将当前字符加入路径并且对 i-1, j-1递归

若当前位置字符不等，说明是由 i-1, j 或者 i, j-1 推得，选取大的一方进行递归打印路径，但是不将当前字符加入路径



1. 算法测试结果

如图 有a.cpp 和 b.cpp的源码，对于b.cpp，是在a.cpp上加入一些无关代码来达到混淆的目的

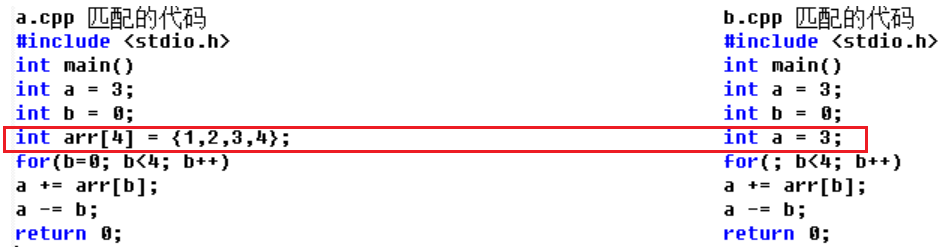


（这里用LCS求解两行代码相似度，然后用行LCS或者行暴力匹配求解重复代码行数）

行LCS：可以看到，插入一些无关语句，以行为单位的LCS可以检测出明显抄袭的部分



暴力：暴力匹配 因为LCS求相似度高于阈值的情况，a.cpp每一行可能和b.cpp中多行代码有较高的匹配度，所以有可能会出现误判的情况，但是还是可以检测出大部分的抄袭语句

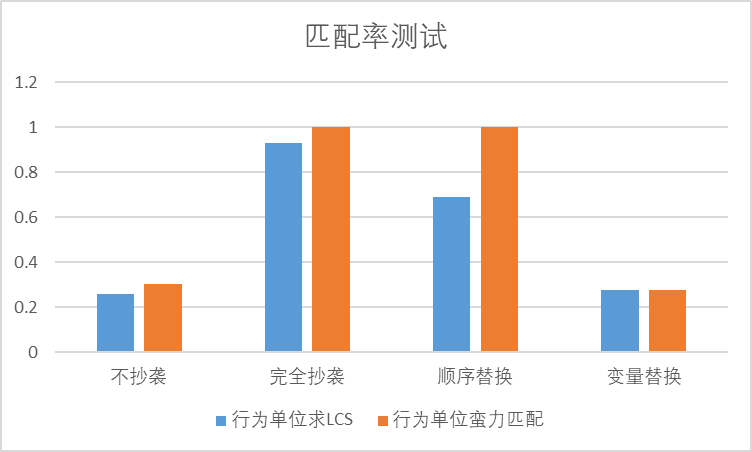


这里用更多的代码样本，追加测试，代码样本分为四个部分，其中

1. 不抄袭的代码是两份功能完全不同的代码
2. 完全抄袭的代码是抄袭+一点点改动
3. 顺序替换的代码是将代码块各个部分的顺序替换
4. 变量替换则是用长且无规律的字符串替换原变量名

（之后的所有测试都是基于这四类代码测试样例进行的）

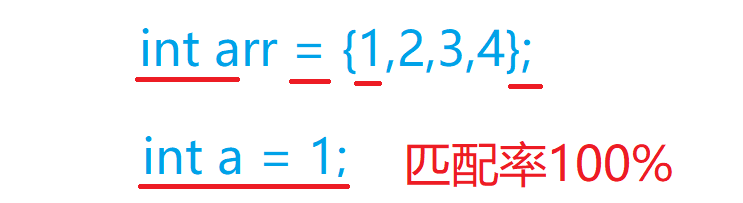




可以看到不论是行LCS还是行暴力匹配，在完全抄袭的情况下。都能有效检查出来，在不抄袭的数据中，能够做到“不杀好人”，在顺序替换的数据下，行暴力匹配效果好于行LCS，因为行暴力匹配不考虑顺序，二者都无法在变量替换的情况下，检查出抄袭

**解决编辑暴力匹配误判问题：引入编辑距离（**两行代码**之间用编辑距离求解相似度）**

如图，因为LCS只考虑顺序，那么会造成误判的情况，一句长的代码可能包含一句短的代码的序列，但是本质上是两句不相干的代码



**编辑距离：介绍**

一个字符串s1通过 增加一个字符 减少一个字符 替换一个字符 的若干次操作，变成字符串s2，这些操作的最小次数，则是他们之间的编辑距离

horse 与ros 的最小编辑距离为3，过程如下

horse → rorse (将 'h' 替换为 'r')

rorse → rose (删除 'r')

rose → ros (删除 'e')

**编辑距离：动态规划求解**

编辑距离的动态规划求解和LCS比较相似，都是通过比较字符来做递推的，不过递推式稍有不同。

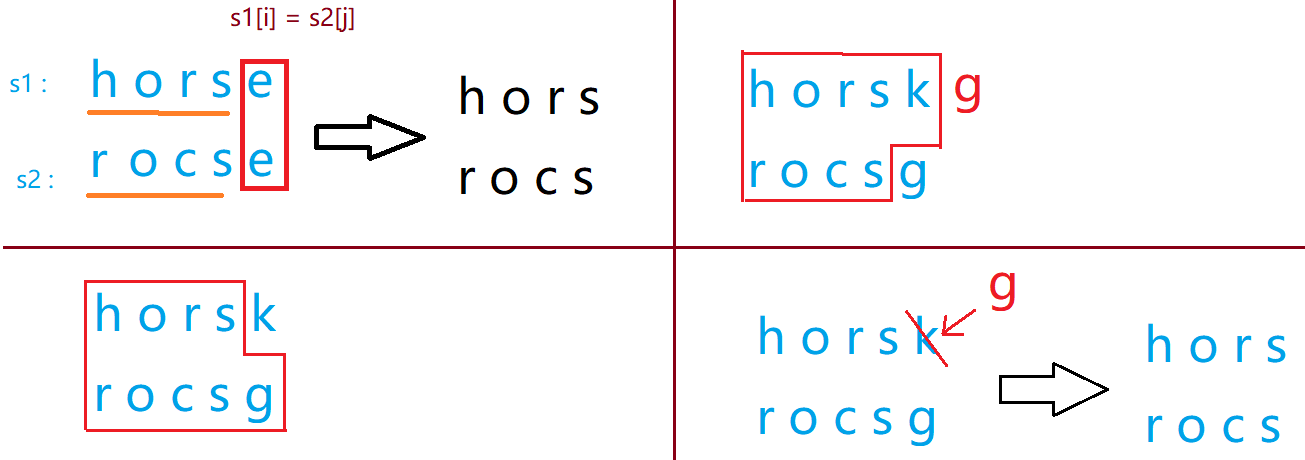
对于s1[i]=s2[j] 的情况，两个相等的字符编辑距离为0，那么他们之间的编辑距离为 s1[0~i-1] 与 s2[0~j-1] 的编辑距离 即 dp[i][j] = dp[i-1][j-1]

对于 s1[i] ≠ s2[j] ，我们有三种方案使得s1向s2转化

1. 将s1的末位删除，那么答案转变为求 s1[0~i-1] 与 s2[0~j] 的编辑距离
2. 将s1的末位替换为s2的末位，答案变为求s1[0~i-1] 与 s2[0~j-1] 的编辑距离
3. 将s1的末位增加一个字符，增加s2的末位字符，那么s1和s2的末位现在相等了，那么答案转变为求 s1[0~i] 与 s2[0~j-1] 的编辑距离

递推式: min(dp[i-1][j-1],dp[i-1][j], dp[i][j-1]) + 1 （+1是因为做了一次操作）

**状态方程推导图解**

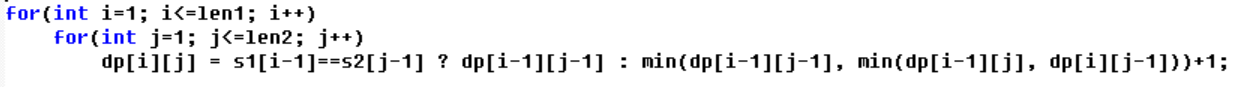


**编辑距离：状态转移方程**

dp[i][j] = dp[i-1][j-1] (s1[i] == s2[j])

dp[i][j] = min(dp[i-1][j-1],dp[i-1][j], dp[i][j-1]) + 1 (s1[i] ≠ s2[j])

**编辑距离：伪代码及时间复杂度分析**



时间复杂度同LCS一样，常规情况下是O(n^2)，这里不探讨编辑距离的时间优化算法了

**编辑距离效果：**

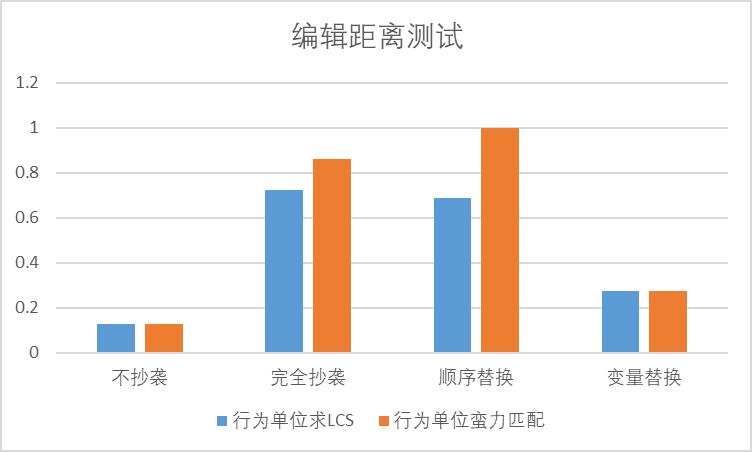
使用 1 – 编辑距离 / max(len1, len2) 得到两行代码之间的相似度（匹配率），再进行代码行匹配，并且输出匹配的代码行



通过上述求得的匹配率，进行暴力匹配，可以发现匹配的效果良好，达到了100%检测出抄袭的代码行，而不会被其他行混淆，因为编辑距离不仅考虑了序列之间的序列，还对序列的长度有要求，两个长度不同的序列，编辑距离得到的匹配率与LCS得到的匹配率相差会大

**编辑距离测试**（两行代码之间相似度用编辑距离求解，然后以行为单位做LCS或暴力匹配）

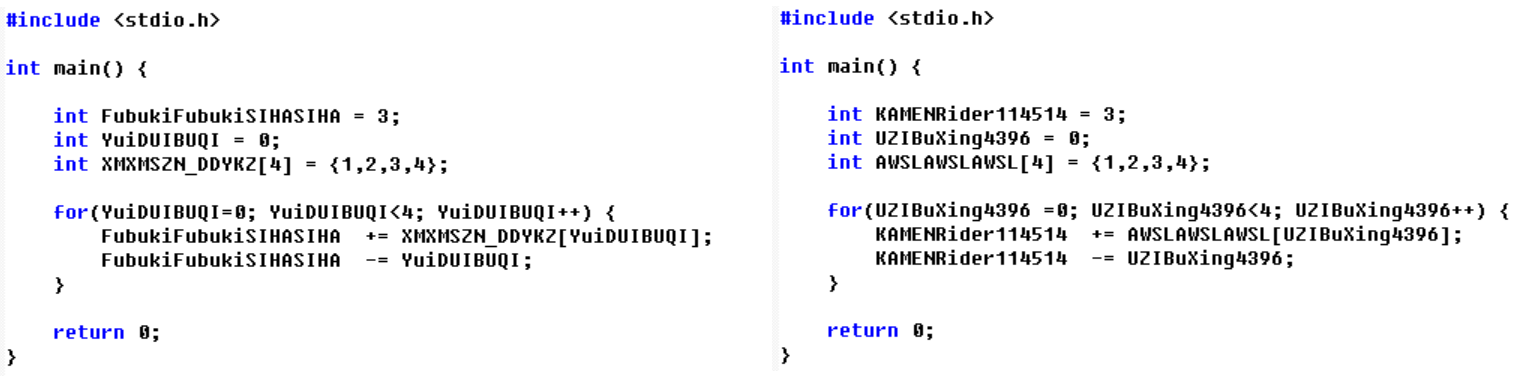




可以看到编辑距离和LCS求解两行代码相似度，效果其实差不多，因为LCS本质是求两行代码相似度，编辑距离本质是求两行代码不相似度，而且因为没有准备特殊样例，（即上文提到的“一行中包含另一行”的误判代码），所以效果和LCS相似

**变量替换的解决方案：**

如下图所示，a.cpp和b.cpp的内容，将所有变量名做代换，并且代换的变量名相当长，应该算是完全的抄袭，可是LCS或者编辑距离计算的相似度并不能检测出来



LCS/编辑距离 匹配的结果，只能匹配到include，main 和return 0 ，需要改进



**改进方法1：将变量换回去**

将变量依次用abcdefg替换，然后再做匹配



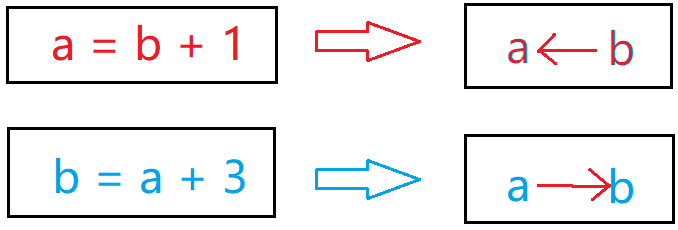
**改进方法2：空变量名**

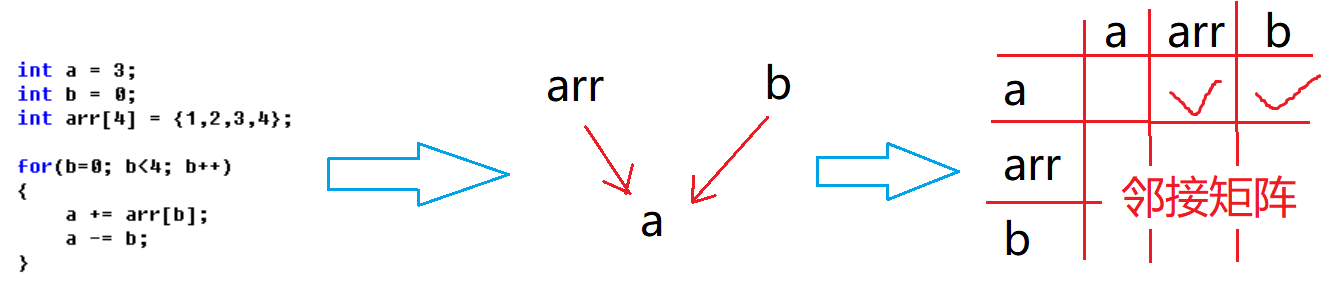
或者选择将变量名换成空字符串，只关注对变量的操作不看变量名，再进行行LCS匹配



**改进方法3：变量依赖关系建图**

变量之间都有一定的依赖关系，比如如图所示，a变量的值，取决于arr变量和b变量，那么根据变量在赋值语句左右的位置可以得出变量的依赖关系并建立有向图

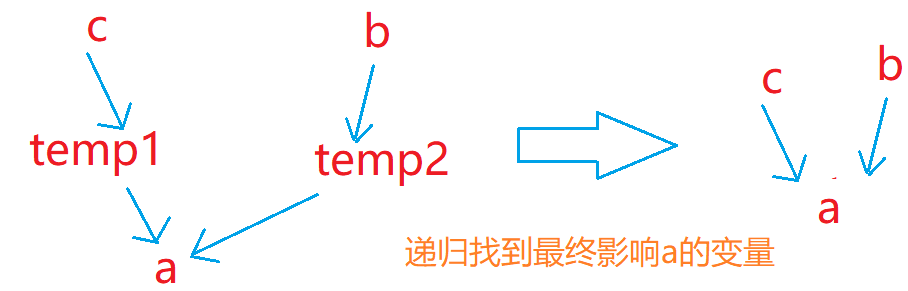




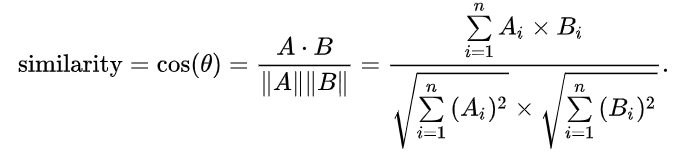
建立有向图，可以通过对邻接矩阵**递归**找到**最终**是哪些变量影响了x的值

通过比较最终的邻接矩阵的行与行之间的“余弦相似度”可以得出是那些语句在抄袭，而绕过变量名的影响

（余弦相似度的比较超过阈值则认为该行是抄袭）

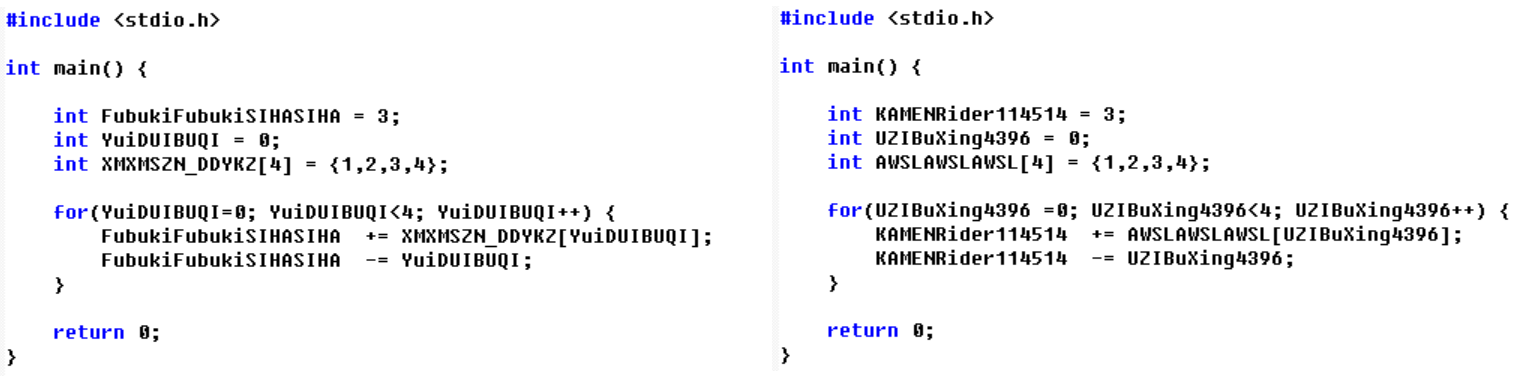


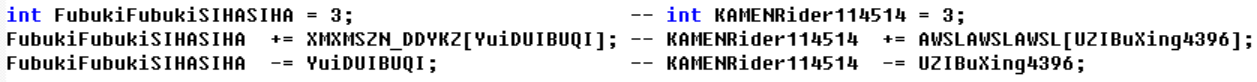
最终的邻接矩阵



余弦相似度（图片来自百度百科）

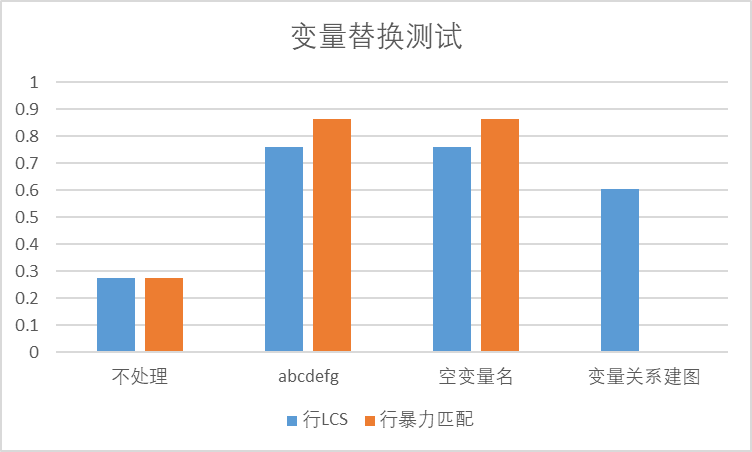
如下图，通过暴力匹配每一行邻接矩阵的相似度，找到抄袭行，注意该方法不能完全匹配所有代码，只能匹配变量赋值语句即变量依赖关系不变的核心代码





**匹配结果：**

|  |  |  |
| --- | --- | --- |
| 解决方案/匹配率 | 行LCS | 行暴力匹配 |
| 不处理 | 0.275862 | 0.275862 |
| abcdefg | 0.758621 | 0.862069 |
| 空变量名 | 0.758621 | 0.862069 |
| 变量关系建图 | 0.603348 | |



结论：可以看到不论是用abcdefg还是空格替换变量名，都能起到很好的效果

而变量关系建图，则是对变量出现的关键语句的匹配，而其他语句则不匹配，所以匹配率偏低，但相比不做处理的方法，仍然有不小的提升，所以适用在于二次人工筛查时对关键语句的检测

1. 对求解这个问题的经验总结

熟练掌握STL中string类的构造，使用以及其常见函数，可以大大加快编程速度，尤其是涉及到大量的字符串的处理的程序

动态规划方程在推导之前一定要明确符号的意义，子问题的定义

（此处写你的过程，比如遇到的错误，以及解决方法，你的所想、所得）

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。