# LCS 算法改进

## 江泽群 15电子科学与技术1班 201530371299

改进的最长公共子序列算法在《算法导论》15.4节 LCS-LENGTH和 PRINT-LCS 算法的基础上，找出所有最长公共子序列。代码实现使用 Java 语言。

1. 算法的基本原理

在算法 LCS-LENGTH 中，把 c[i-1,j]=c[i,j-1]的情况并入 c[i-1,j]>c[i,j-1]的情况中，如果将这种情况单独处理，就能够保留必要的信息，以实现查找所有最长公共子序列的目的。具体地，将c[i-1,j]=c[i,j-1]情况下的 表 b 箭头方向设置为←和↑同时存在的第4种情况。根据箭头的指向，我们可以把整个表 b看做是一个有向无环图的邻接矩阵，每个矩阵的横纵坐标确定一个点。

搜索所有LCS 时，我们用递归的方式，最外层调用找到每一个LCS的最后一个字符，第二层调用找到每一个 LCS 的倒数第二个字符……以此类推，直到找到LCS 的首字符。这个递归算法实现在FindAllLCS.java 中 的 findAllLCS 中。

既然在每一层递归中涉及到字符的搜索，就需要实现一个新的算法，对应于FindAllLCS.java 中的 search函数中。其基本思想是：前面提到，表 b 能够当成有向无环图的邻接矩阵，因此，利用深度优先搜索，从一点出发，遍历矩阵，找到目标字符后，在该处的搜索就停止。判断访问的字符是不是目标字符，只要判断它在 b 矩阵的方向是不是指向左上方。根据 b 矩阵的性质，搜索过程中访问到的字符，在 c 矩阵中都对应相同的值。

1. 代码实现

算法实现程序FindAllLCS.java如下：

import java.util.\*;

public class FindAllLCS {

final static int LEFT\_AND\_UP = 0;

final static int LEFT = 1;

final static int UP = 2;

final static int LEFT\_OR\_UP = 3;

public static int solve(String X, String Y) {

int m = X.length(),

n = Y.length();

int[][] C = new int[m + 1][n + 1];

int[][] directions = new int[m + 1][n + 1];

for (int i = 0; i <= m; i++) {

C[i][0] = 0;

}

for (int i = 0; i <= n ; i++) {

C[0][i] = 0;

}

for (int i = 1; i <= m ; i++) {

for (int j = 1; j <= n ; j++) {

if (X.charAt(i - 1) == Y.charAt(j - 1)) {

C[i][j] = C[i - 1][j - 1] + 1;

directions[i][j] = LEFT\_AND\_UP;

}

else if (C[i - 1][j] > C[i][j - 1]) {

C[i][j] = C[i - 1][j];

directions[i][j] = UP;

}

else if (C[i - 1][j] < C[i][j - 1]) {

C[i][j] = C[i][j - 1];

directions[i][j] = LEFT; }

else {

C[i][j] = C[i - 1][j];

directions[i][j] = LEFT\_OR\_UP;

}

}

}

System.out.println("There are " + C[m][n] + " solutions in this case.");

System.out.println();

System.out.println("All LCS without duplicated results are as follows.");

System.out.println(" " + new HashSet<String>(findAllLCS(C, directions, X, m, n)));

System.out.println();

return C[m][n];

}

public static List<String> findAllLCS(int[][] C, int[][] directions,

String X, int i, int j) {

List<String> results = new ArrayList<String>();

if (i > 0 && j > 0) {

List<int[]> end\_points = searchEndPoints(directions, i, j);

List<String> partial\_lcs;

for (int[] end\_pos : end\_points) {

if (C[end\_pos[0]][end\_pos[1]] == 1) {

results.add(X.charAt(end\_pos[0] - 1) + "");

}

else {

partial\_lcs = findAllLCS(C, directions, X,

end\_pos[0] - 1, end\_pos[1] - 1);

for (String s : partial\_lcs) {

s += X.charAt(end\_pos[0] - 1);

results.add(s);

}

}

}

}

return results;

}

static List<int[]> searchEndPoints(int[][] directions, int i, int j) {

List<int[]> ends = new ArrayList<int[]>();

int[][] visited = new int[i + 1][j + 1];

search(directions, i, j, ends, visited);

return ends;

}

static void search(int[][] directions, int i, int j,

List<int[]> ends, int[][] visited) {

if (visited[i][j] == 1) {

return;

}

if (directions[i][j] == LEFT\_AND\_UP) {

visited[i][j] = 1;

ends.add(new int[]{i, j});

}

if (directions[i][j] == LEFT\_OR\_UP || directions[i][j] == LEFT) {

visited[i][j] = 1;

search(directions, i, j - 1, ends, visited);

}

if (directions[i][j] == LEFT\_OR\_UP || directions[i][j] == UP) {

visited[i][j] = 1;

search(directions, i - 1, j, ends, visited);

}

}

public static void main(String[] args) {

String X = args[0];

String Y = args[1];

solve(X, Y);

}

}

1. 测试代码正确性

编写测试样例samples，代码如图3-1所示。

编写测试脚本，代码如图3-2所示。

测试结果如图3-3所示。

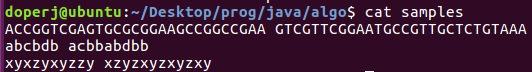


图3-1 测试样例

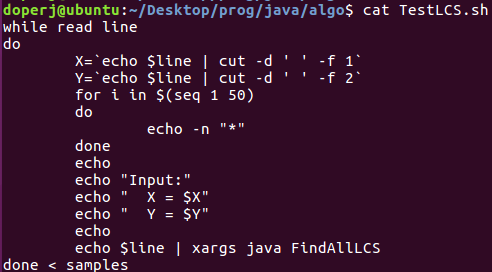


图3-2 测试脚本

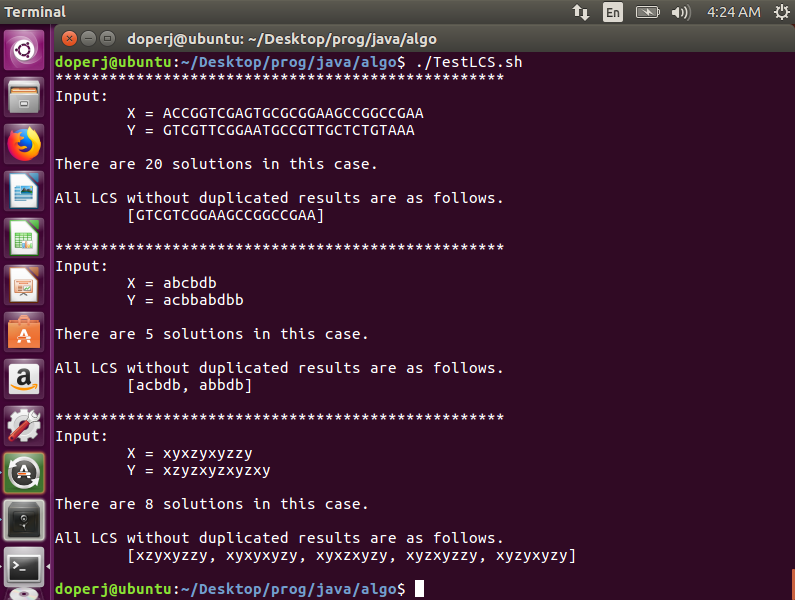


图 3-3 测试结果

1. 进一步改进

在 findAllLCS 中，存在重叠子问题现象。这一问题可以使用备忘录方法解决，即在函数开头判断问题是否求解过，如果是，直接获得答案并返回，否则继续求解并保存求解的结果。改进后的findAllLCS代码如下：

public static List<String> findAllLCS(int[][] C, int[][] directions,

String X, int i, int j, Map<String, List<String>> memo) {

if (memo.get("" + i + j) != null) {

System.out.println("Find overlapped sub-problem!");

return memo.get("" + i + j);

}

List<String> results = new ArrayList<String>();

if (i > 0 && j > 0) {

List<int[]> end\_points = searchEndPoints(directions, i, j);

List<String> partial\_lcs;

for (int[] end\_pos : end\_points) {

if (C[end\_pos[0]][end\_pos[1]] == 1) {

results.add(X.charAt(end\_pos[0] - 1) + "");

}

else {

partial\_lcs = findAllLCS(C, directions, X,

end\_pos[0] - 1, end\_pos[1] - 1, memo);

for (String s : partial\_lcs) {

s += X.charAt(end\_pos[0] - 1);

results.add(s);

}

}

}

}

memo.put("" + i + j, results);

return results;

}

solve 函数对 fingAllLCS 调用的部分改为：

Map<String, List<String>> memo = new HashMap<String, List<String>>();

List<String> results = findAllLCS(C, directions, X, m, n, memo);

测试结果如图4-1、图4-2所示。

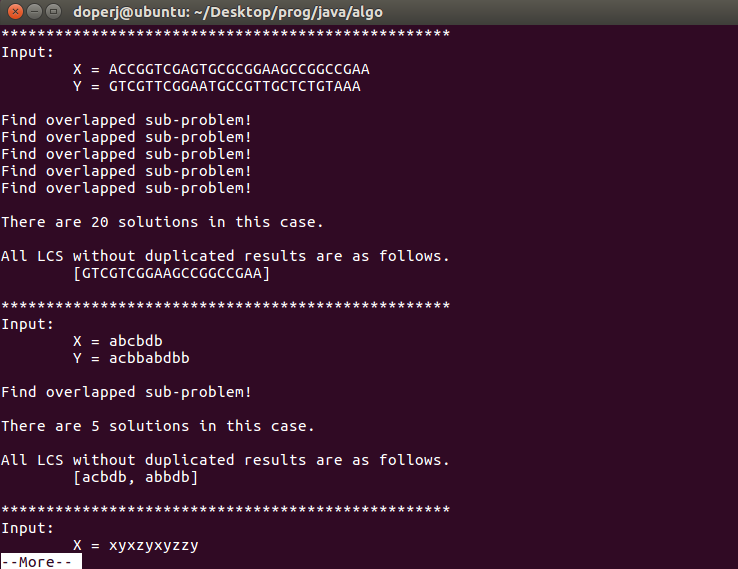


图4-1 部分测试结果

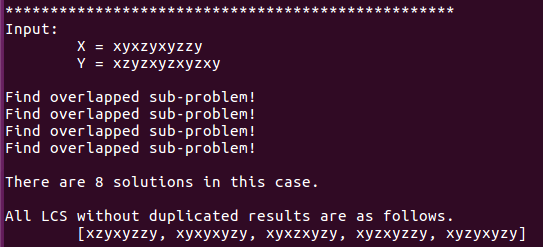


图4-2 部分测试结果

从测试结果可以看出，重叠子问题的现象确实存在，但子问题数目不一，视测试样例而定。