

哈尔滨工业大学

软件安全课程实验报告

实验三

## 学院：计算机学院

## 班级：1403202

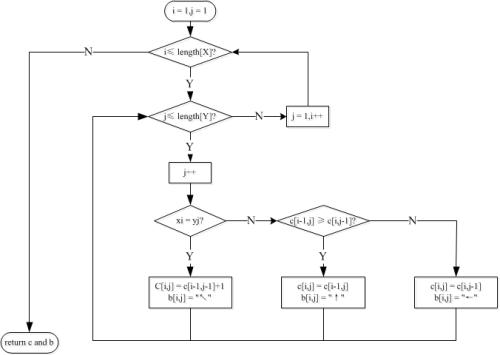
## 学号：1140320206

## 姓名：霍峻杰

# 实验三 恶意代码特征提取

1. 实验内容
2. 理解基于最长公共子序列的协议特征提取方法
3. 掌握网络恶意代码特征的提取流程
4. 学习最长公共子序列的提取
5. 实现字符串最长公共子序列的提取算法
6. 利用动态规划的方法实现字符串最长公共子序列的提取
7. 依据输入的字符串构建L(m,n)数组，利用L(m,n)数组查找两个字符串直接的最长公共子序列
8. 实验要求
9. 实验数据准备。利用ASCII字符集作为输入集，不考虑多字节编码的中文、英文字符集。
10. 程序的输入部分：2个字符串。输出部分：这2个字符串的最长公共子序列，如有多个一同给出。
11. 实验结果和实验数据一起给出。
12. 实验结果

1. 设计的算法流程图

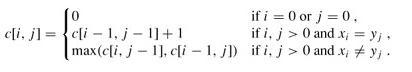


2. 算法说明

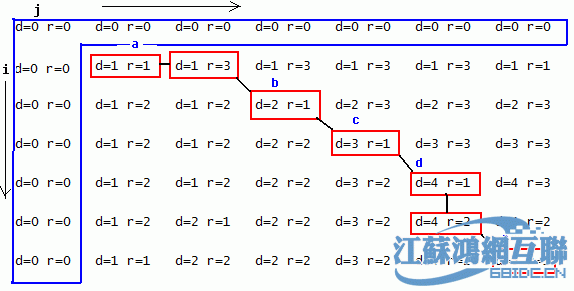
由最长公共子序列问题的最优子结构性质可知，要找出X=<x1, x2, …, xm>和Y=<y1, y2, …, yn>的最长公共子序列，可按以下方式递归地进行：当xm=yn时，找出Xm-1和Yn-1的最长公共子序列，然后在其尾部加上xm(=yn)即可得X和Y的一个最长公共子序列。当xm≠yn时，必须解两个子问题，即找出Xm-1和Y的一个最长公共子序列及X和Yn-1的一个最长公共子序列。这两个公共子序列中较长者即为X和Y的一个最长公共子序列。

实现思路：

1. 用动态规划思想实现



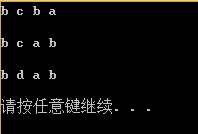
1. 回溯输出所有的LCS。

3. 关键的数据结构



存放回溯二维数组，也就是动态规划的数组。

4. 实验结果



5. 程序源码

# coding=utf-8

class LCS():

def input(self, x, y):

#读入待匹配的两个字符串

if type(x) != str or type(y) != str:

print 'input error'

return None

self.x = x

self.y = y

def Compute\_LCS(self):

xlength = len(self.x)

ylength = len(self.y)

self.direction\_list = [None] \* xlength #这个二维列表存着回溯方向

for i in xrange(xlength):

self.direction\_list[i] = [None] \* ylength

self.lcslength\_list = [None] \* (xlength + 1)

#这个二维列表存着当前最长公共子序列长度

for j in xrange(xlength + 1):

self.lcslength\_list[j] = [None] \* (ylength + 1)

for i in xrange(0, xlength + 1):

self.lcslength\_list[i][0] = 0

for j in xrange(0, ylength + 1):

self.lcslength\_list[0][j] = 0

#下面是进行回溯方向和长度表的赋值

for i in xrange(1, xlength + 1):

for j in xrange(1, ylength + 1):

if self.x[i - 1] == self.y[j - 1]:

self.lcslength\_list[i][j] = self.lcslength\_list[i - 1][j - 1] + 1

self.direction\_list[i - 1][j - 1] = 0 # 左上

elif self.lcslength\_list[i - 1][j] > self.lcslength\_list[i][j - 1]:

self.lcslength\_list[i][j] = self.lcslength\_list[i - 1][j]

self.direction\_list[i - 1][j - 1] = 1 # 上

elif self.lcslength\_list[i - 1][j] < self.lcslength\_list[i][j - 1]:

self.lcslength\_list[i][j] = self.lcslength\_list[i][j - 1]

self.direction\_list[i - 1][j - 1] = -1 # 左

else:

self.lcslength\_list[i][j] = self.lcslength\_list[i - 1][j]

self.direction\_list[i - 1][j - 1] = 2 # 左或上

self.lcslength = self.lcslength\_list[-1][-1]

return self.direction\_list, self.lcslength\_list

def printLCS(self, curlen, i, j, s):

if i == 0 or j == 0:

return None

if self.direction\_list[i - 1][j - 1] == 0:

if curlen == self.lcslength:

s += self.x[i - 1]

for i in range(len(s)-1,-1,-1):

print s[i],

print '\n'

elif curlen < self.lcslength:

s += self.x[i-1]

self.printLCS(curlen + 1, i - 1, j - 1, s)

elif self.direction\_list[i - 1][j - 1] == 1:

self.printLCS(curlen,i - 1, j,s)

elif self.direction\_list[i - 1][j - 1] == -1:

self.printLCS(curlen,i, j - 1,s)

else:

self.printLCS(curlen,i - 1, j,s)

self.printLCS(curlen,i, j - 1,s)

def returnLCS(self):

#回溯的入口

self.printLCS(1,len(self.x), len(self.y),'')

if \_\_name\_\_ == '\_\_main\_\_':

p = LCS()

p.input('abcbdab', 'bdcaba')

p.Compute\_LCS()

p.returnLCS()