****

**数据结构**

**实 验 报 告**

实验名称：实验三 基于字符串模式匹配算法的病毒检测问题

实验学时：2

实验环境：dev511

学生学院：信息科学与技术学院/网络空间安全学院

专业班级：计算机2101

学 号：20211003153

学生姓名：赖永超

指导教师：周咏梅

学 年： 2021-2022第二学期

2022 年 4月 26日

目录

[一、实验目的 3](#_Toc102134154)

[二、实验内容 3](#_Toc102134155)

[1. 病毒感染检测 3](#_Toc102134156)

[2. 三元组顺序表表示的稀疏矩阵加法 13](#_Toc102134157)

[三、实验总结 22](#_Toc102134158)

**实验三 基于字符串模式匹配算法的病毒检测问题**

姓名：赖永超 班级：计算机2101 学号：20211003153

## 一、实验目的

1.掌握字符串的顺序存储方法。

2.掌握字符串模式匹配算法BF算法或KMP算法的实现

## 二、实验内容

### 1. 病毒感染检测

1）实验题目

pta-DS2022-ch4-4-7-3病毒感染检测

2）问题描述

人的DNA和病毒DNA均表示成由一些字母组成的字符串序列。然后检测某种病毒DNA序列是否在患者的DNA序列中出现过，如果出现过，则此人感染了该病毒，否则没有感染。例如，假设病毒的DNA序列为baa，患者1的DNA序列为aaabbba，则感染，患者2的DNA序列为babbba，则未感染。（注意，人的DNA序列是线性的，而病毒的DNA序列是环状的)

输入格式:

输入第一行中给出1个整数i(1≤i≤11)，表示待检测的病毒DNA和患者DNA的对数。

输入i行串序列，每行由两个字符串组成（字符串中不含不可见字符），两字符串之间用一个空格分隔，分别代表病毒的DNA序列和患者的DNA序列，病毒的DNA序列和患者的DNA序列长度不超过500。

输出格式:

依次逐行输出每对检测样本的结果，感染输出：YES，未感染输出：NO。

输入样例1:

1

baa bbaabbba

输出样例1:

YES

输入样例2:

2

cced cdccdcce

bcd aabccdxdxbxa

输出样例2:

YES

NO

3）算法分析

①数据结构

使用两个string类型的s和x，存储患者DNA和病毒DNA。定义和输入代码见图1-1。



图1-1 s和x字符串的定义和输入

②功能描述

首先输入数字n，得知有n对患者DNA和病毒DNA。然后逐个判断患者DNA中是否有病毒DNA，从而判断患者是否被感染。

③算法描述

A.输入

根据题意，先输入基因对总数n，然后输入n个基因对。代码见图1-2。

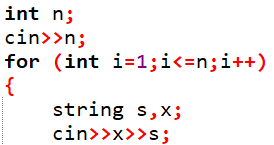


图1-2 输入基因对

B.“扩增”病毒DNA，获取病毒的其它DNA序列

获取病毒DNA长度lx，然后将病毒扩增一倍，接着从DNA字符串的第i位（0 < i < lx-1）往后截取lx个字符串，得到该病毒的其它DNA序列。代码见图1-3。

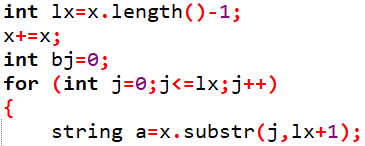


图1-3 扩增病毒DNA，获取病毒其它DNA序列

C.判断患者是否感染该病毒

对截取的每一种DNA序列，都判断是否在患者DNA中出现过。若出现过则表示患者感染病毒，否则没有感染病毒。具体代码见图1-4。

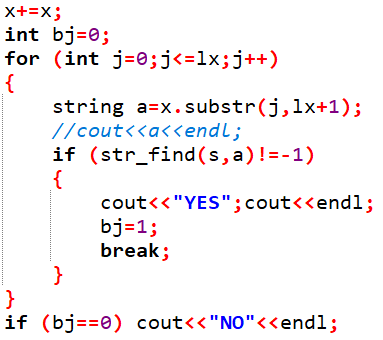


图1-4 判断患者是否感染

D．病毒DNA与患者DNA匹配（BF算法）

本版本代码中，使用模式匹配算法（Brute-Force算法）

算法步骤：

1. 分别利用计数指针i和j指示主串s和p当前正待比较的字符位置。

②i从0循环到s.length()-1，在i循环下j从0到p.length()循环。

③当s[i]!=p[j]时，退出本次j循环，i++，进入下一个位置的匹配。如果当前i循环中的j循环完成后，未有s[i]!=p[j]的情况出现，说明在此位置s与p匹配。

具体代码见图1-5。

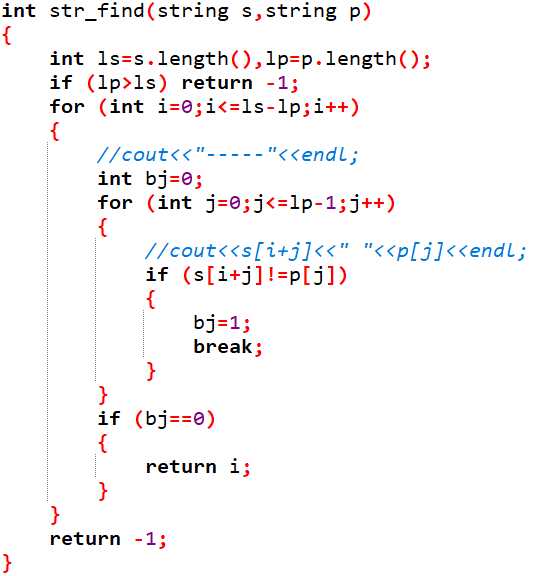


图1-5 BF算法

4）具体代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/DS/DS-3%20%E5%9F%BA%E4%BA%8E%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%A8%A1%E5%BC%8F%E5%8C%B9%E9%85%8D%E7%AE%97%E6%B3%95%E7%9A%84%E7%97%85%E6%AF%92%E6%A3%80%E6%B5%8B%E9%97%AE%E9%A2%98/sy3_1_1_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy3\_1\_1\_2101\_20211003153\_赖永超（文件编码：ANSI，推荐使用Dev C++打开）

5）运行结果

①PTA提交结果，图1-6。



图1-6 PTA 病毒感染检测提交结果

②Dev C++运行截图

具体运行截图见图1-7，图1-8和图1-9。



图1-7 病毒感染检测 运行截图①

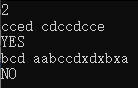


图1-8 病毒感染检测 运行截图②

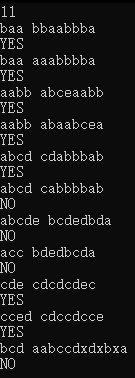


图1-9 病毒感染检测 运行截图③

6）程序运行结果分析

程序运行正常，耗时正常，内存占用低。

7）功能扩展

①使用KMP算法来完成病毒感染检测

A.功能描述

使用KMP算法来完成病毒感染检测

B.算法描述

使用KMP算法来加快字符串的匹配。每当一趟匹配过程中出现字符比较不等时，不需回溯i指针，而是利用已经得到的“部分匹配”的结果将模式向右“滑动”尽可能远的一段距离后，继续进行比较。

C.代码实现

核心代码见图1-10

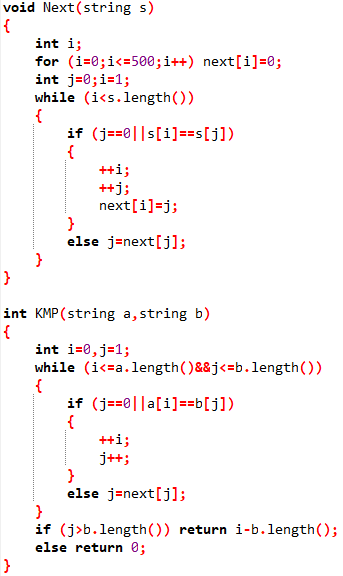


图1-10 KMP算法

详细代码

D.GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/DS/DS-3%20%E5%9F%BA%E4%BA%8E%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%A8%A1%E5%BC%8F%E5%8C%B9%E9%85%8D%E7%AE%97%E6%B3%95%E7%9A%84%E7%97%85%E6%AF%92%E6%A3%80%E6%B5%8B%E9%97%AE%E9%A2%98/sy3_1_2_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy3\_1\_2\_2101\_20211003153\_赖永超.cpp（文件编码：ANSI，推荐使用Dev C++打开）

E.运行结果，见图1-11，图1-12和图1-13。



图1-11 病毒感染检测KMP版本 运行截图①

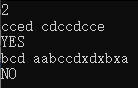


图1-12 病毒感染检测KMP版本 运行截图②

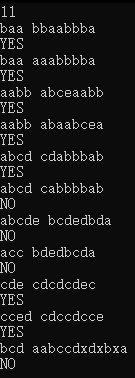


图1-13 病毒感染检测KMP版本 运行截图③

F.BF算法与KMP算法运算时间的比较

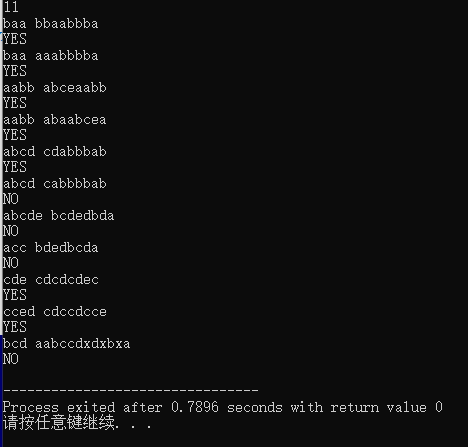


图1-14 BF算法的运行结果

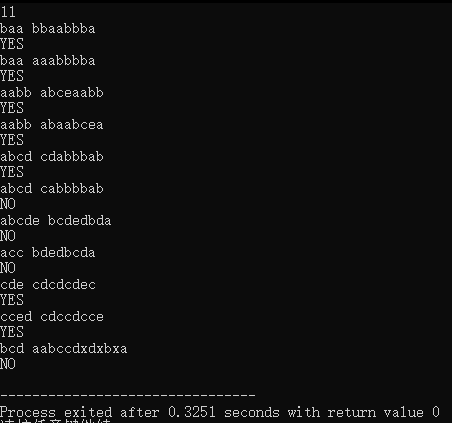


图1-15 KMP算法的运行结果

在相同的输入下，使用KMP算法的运算时间比BF算法短。

### 2. 三元组顺序表表示的稀疏矩阵加法

1）实验题目

pta-2020DS-ch4 练习题-4-7-4 三元组顺序表表示的稀疏矩阵加法

2）问题描述

三元组顺序表表示的稀疏矩阵加法。

输入格式:

输入第1行为两个同型矩阵的行数m、列数n，矩阵A的非零元素个数t1，矩阵B的非零元素个数t2。

按行优先顺序依次输入矩阵A三元组数据，共t1行，每行3个数，分别表示非零元素的行标、列标和值。

按行优先顺序依次输入矩阵B三元组数据，共t2行，每行3个数，分别表示非零元素的行标、列标和值。

输出格式:

输出第1行为相加后矩阵行数m、列数n及非零元素个数t。

输出t行相加后的三元组顺序表结果，每行输出非零元素的行标、列标和值，每行数据之间用空格分隔。

输入样例1:

4 4 3 4

0 1 -5

1 3 1

2 2 1

0 1 3

1 3 -1

3 0 5

3 3 7

输出样例1:

4 4 4

0 1 -2

2 2 1

3 0 5

3 3 7

3）算法分析

①数据结构

使用一个二维数组储存最后的结果矩阵。代码见图2-1。



图2-1 最终结果矩阵的定义

②功能描述

首先输入数字m与n，得知有矩阵大小为m×n。接着输入t1和t2告知非零元的个数，下列的t1+t2依次输入矩阵某位置的数字，最终结果矩阵对应的位置加上这个数。最后得到矩阵加法的结果。

③算法描述

A.输入

输入数字m,n,t1,t2，然后t1+t2行非零数的位置和数字，代码见图2-2。

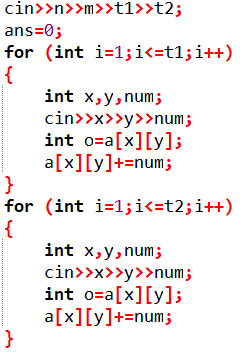


图2-2 矩阵加法的综合输入

B.计算非零个数

遍历矩阵每个位置，若不是0，非0个数加1。代码见图2-3。

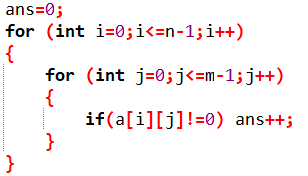


图2-3 计算矩阵内非零个数

C.输出

先矩阵大小和非零个数，然后遍历矩阵每个位置，若不是0，则输出其位置和值。具体代码见图2-4。

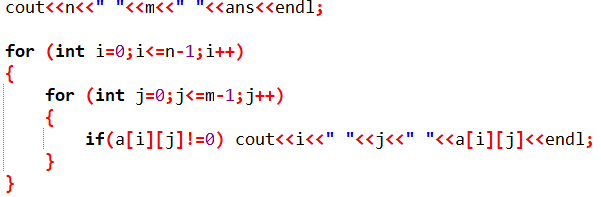


图2-4 最终结果矩阵的综合输出

4）具体代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/DS/DS-3%20%E5%9F%BA%E4%BA%8E%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%A8%A1%E5%BC%8F%E5%8C%B9%E9%85%8D%E7%AE%97%E6%B3%95%E7%9A%84%E7%97%85%E6%AF%92%E6%A3%80%E6%B5%8B%E9%97%AE%E9%A2%98/sy3_2_1_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp)

文件：sy3\_2\_1\_2101\_20211003153\_赖永超（文件编码：ANSI，推荐使用Dev C++打开）

5）运行结果

①PTA提交结果，图2-5。



图2-5 PTA 三元组顺序表表示的稀疏矩阵加法提交结果

②Dev C++运行截图

具体运行截图见图2-6。

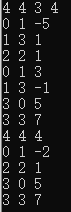


图2-6 三元组顺序表表示的稀疏矩阵加法 运行截图

6）程序运行结果分析

程序运行正常，耗时正常，内存占用低。

## 三、实验总结

在本次实验中，主要使用了研究字符串的存储和寻找子串在主串出现的位置的问题。在子串匹配问题上，研究了BF算法和KMP算法。BF算法简单易懂，容易实现，面对数据量少的题目时，可优先使用BF算法。当数据量变得庞大时，需要速度更加快的KMP算法。KMP算法的时间复杂度比BF算法要小得多，使用KMP算法节省了很多时间。我在使用KMP算法解决病毒检测问题时，计算next的算法研究很久。最后根据书上给的参考加以改进后得出了现在的计算next算法。计算出子串的next值后，KMP算法就可以顺利的运行了。最后经过比较，对于同一个数据点，KMP算法的运行时间比BF算法短。