算法基础上机实验四

PB17000285

雷洋

一．实验内容

1. 实现串匹配算法。其中，文本串 T 的长度为 n，模式串 P 的长度为 m，字符串中的字符均是随机生成的取自字符集[0-9A-Za-z]（共 62 个不同字符）的字符。(n, m)共有 5 组取值，分别为: (2^5， 4), (2^8， 8), (2^{11},16), (2^{14},32), (2^{17},64)。

2. 采用算法：朴素字符串匹配算法；Rabin-Karp 算法；KMP 算法； Boyer-Moore-Horspool 算法。

二．实验要求

1. 实验文件严格按照要求创建。

三．每种算法建立子文件夹，output 建立四个子文件夹，对应四种算法，结果存放在各个文件夹的 result.txt。

四．实验设备及环境

实验设备和环境为 windows10 个人计算机操作系统下的 devC++ IDE，TDM-GCC 4.9.2 64-bit。

五．实验方法

1. 通过文件读入将数据保存在全局数组中。

2.在不同的算法中传入整数 index 代表第几组数据，将首次匹配成功的位置和执行时间写入 result 文件。

3. 通过 windows 库里的函数获取 cpu 频率，以及开始和结束

的周期数差，来计算运行时间（单位秒）。

4.分析得到的数据，绘图观察分析得出结论。

五．实验步骤

1. 创建头文件 head.h,写入必要的宏定义，声明和定义所用到全局变量和函数。

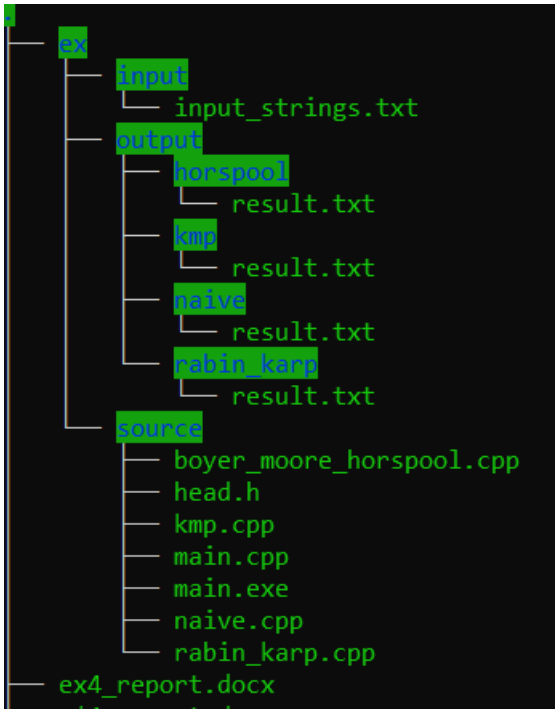
2. 分别创建四种算法的.cpp 文件。实现各个算法所依赖的函数和算法。

3. 创建 main.cpp，对不同算法不同规模的排序时间进行统计和输出。

六．实验结果与分析

1. 本次实验结果截图：

文件结构如下：



朴素字符串匹配算法结果截图：

```
32 4 28 0.0000008000
256 8 -1 0.0000013000
2048 16 1435 0.0000065000
16384 32 -1 0.0000706000
131072 64 -1 0.0005640000
```

Rabin_karp 算法结果截图:

```
32 4 28 0.0000005000
256 8 -1 0.0000059000
2048 16 1435 0.0002569000
16384 32 -1 0.0022684000
131072 64 -1 0.0232644000
```

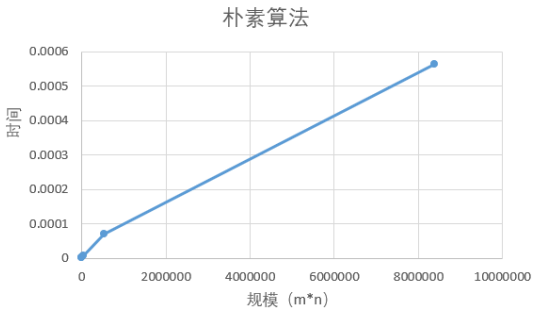
Kmp 算法结果截图:

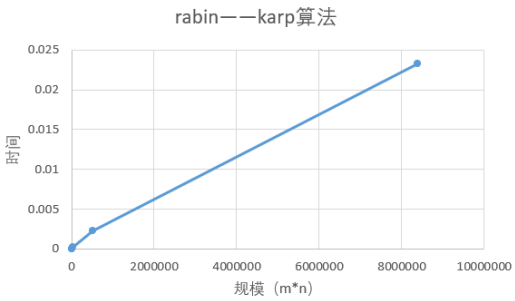
```
32 4 28 0.0000006000
256 8 -1 0.0000010000
2048 16 1435 0.0000050000
16384 32 -1 0.0000538000
131072 64 -1 0.0004237000
```

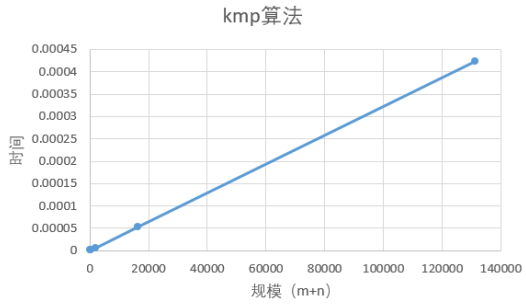
Boyer-moore-horspool 算法结果截图:

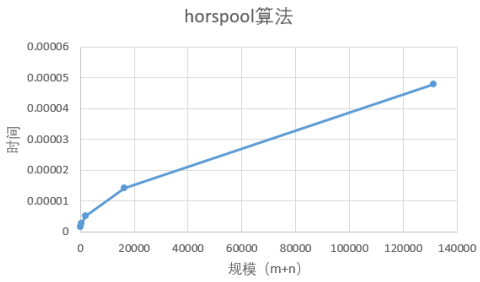
```
32 4 28 0.0000017000
256 8 -1 0.0000027000
2048 16 1435 0.0000051000
16384 32 -1 0.0000142000
131072 64 -1 0.0000479000
```

时间趋势图









2.结果分析

对于朴素算法和 rabin——karp 算法，时间复杂度为 O(m*n)；对于 kmp 和 horspool 算法，时间复杂度为 O（m+n），可以看到，实验结果在复杂度上较为符合预期。

但是通过观察可以发现：rabin——karp 算法的效率并不比朴素算法高，其原因可能是：基数和素数选取不是非常合适；出现较多伪命中点。

3.结论

可以看到，如果输入规模很大时，用 horspool 算法的优势很大，运行时间相较其他算法小 1-3 个数量级。