字符串匹配实验报告

软件 71　唐建宇　2017012221

2019 年 5 月 13 日

1．实验环境

编程语言：C++
IDE: Visual Studio 2012
图形界面框架：Qt 5.12
图形界面 IDE: Qt Creator 4.8
操作系统：Windows 10
CPU: Intel Core i5 2.36GHz
内存：8GB

2．算法设计和分析

本次实验比较了字符串匹配的三种算法：Brutal Force 算法，KMP 算法，Boyer-Moore 算法。对于这三种算法课堂上均进行了详细分析，故直接给出其复杂度

Algorithm	Preprocessing	Matching
Brutal Force	0	$O(mn)$
KMP	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m + \|\Sigma\|)$	$O(mn)$

其中，在 Boyer-Moore 算法的实现中，对于 Overlapping-suffix 的求解的伪代码如下见 Algorithm 1:
如此一来，求 suffix[i] 时就可以利用到之前求过的 suffix[1...i-1] 的信息，这会好于朴素方法。

Algorithm 1 get-suffix

```
1: function GET-SUFFIX(p)
2:     Let suffix[0...m] be a new array
3:     π=prefix(p) 先计算模式串的前缀函数
4:     suffix[0]=0
5:     for i = 1 to m − 1 do
6:         suffix[i]=suffix[π[i]]
7:         if suffix[i]=π[i] then
8:             t=i-suffix[i]
9:             while t ≥ 1 and p[t]=p[m-i+t] do
10:                t = t − 1
11:            end while
12:            suffix[i]=i-t
13:        end if
14:    end for
15:    suffix[m]=m
16:    return suffix[1...m]
17: end function
```

3. 实验设计与结果

实验设计

调用 c++11 标准的 <random> 库中随机数生成引擎，利用均匀分布随机生成长度为 5,10,100,500,1000 的模式串和长度为 1000,10000,100000,500000,1000000 的待匹配串，对这些模式串和待匹配串的不同长度进行组合得到用时结果。

测试结果

Brutal Force 算法用时 (ms)

time n / m	1000	10000	100000	500000	1000000
5	0.006	0.06	1	3	7
10	0.006	0.06	0.6	3	6
100	0.006	0.06	0.6	3	6
500	0.006	0.06	0.6	3	6
1000	0.006	0.06	0.6	4	6

KMP 算法用时 (ms)

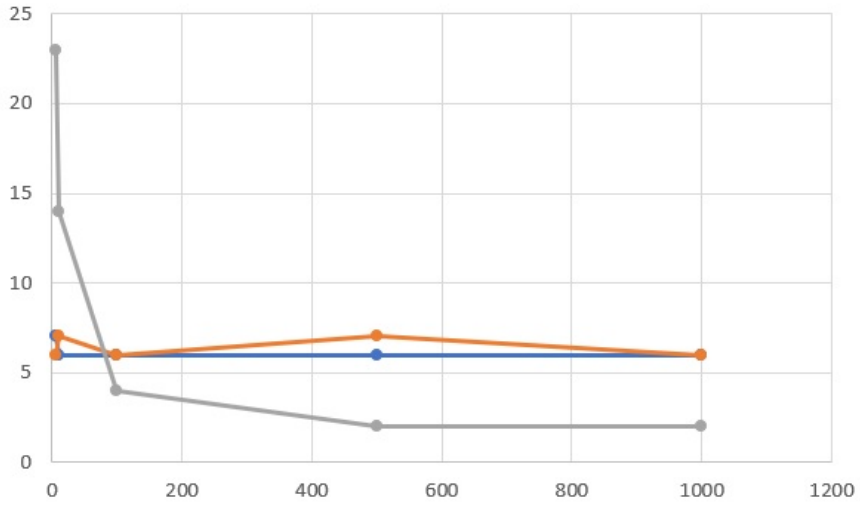
time n / m	1000	10000	100000	500000	1000000
5	0.007	0.07	0	4	6
10	0.007	0.07	0.7	3	7
100	0.008	0.07	0.7	4	6
500	0.011	0.07	0.7	4	7
1000	0.014	0.07	0.7	3	6

Boyer-Moore 算法用时 (ms)

time n / m	1000	10000	100000	500000	1000000
5	0.033	0.3	3	11	23
10	0.025	0.14	1.4	7	14
100	0.057	0.086	0.4	2	4
500	0.15	0.18	0.3	1	2
1000	0.266	0.29	0.4	1	3

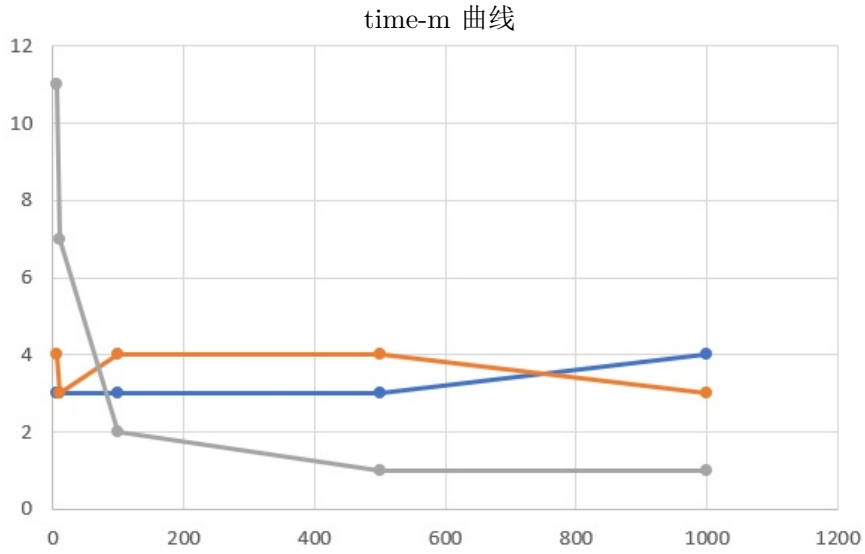
固定 $n = 1,000,000$，改变模式串长度 m 得到三种算法的用时曲线：

time-m 曲线



蓝色-BF 算法　橘色-KMP 算法　灰色-BM 算法 (下同)

固定 $n = 500,000$，改变模式串长度 m 得到三种算法的用时曲线：



4. 结论与分析

结论

如上曲线所示，纵向对比明显发现 Boyer-Moore 算法在待匹配串长度较大时，其用时随着模式串长度的增加而显著减小；对于暴力算法和 KMP 算法而言，其时间主要取决于待匹配串的长度，在待匹配串长度显著长于模式串时，对模式串长度的变化相对不敏感。

横向对比发现，在百万数据量级下，三种算法并没有太大的差距，或者说其实际表现的差距远小于复杂度分析中的差距，具体而言每一种情况下其用时都是处在相同的量级且绝对值都不大。

分析

这三种算法的实际表现接近的原因，主要在于数据的随机性。对 BF 算法而言，虽然最坏复杂度达到 $O(mn)$，但是这是在非常精心设计的情况下才有可能在待匹配串的每一个位置都要检查到模式串的最后一位才能判断出该位置是否匹配，随机的情况往往是根本不需要检查几位就已经发现不匹配

而往下一位走，这就造成了他的用时与线性的 KMP 算法相当甚至个别情况还要更好。

而之所以 BM 算法对于模式串的长度非常敏感，主要还是因为从后向前的比较方式，导致当模式串长度增加时，移动次数可以大大减小，甚至有可能每一次移动都几乎是模式串的长度即 m 位，最终用时趋进于最好情况 $O(n/m)$。