字符串匹配算法实验报告

软件 13 杨楠 2021010711

摘要：比较了几种典型的字符串匹配算法：Brute-Force，KMP，BM。理论分析了这几种算法的复杂度，并通过设计具体程序，测试了算法的运行时间。

1 实验环境

语言：C++
操作系统：Windows
平台：VSCode mingw

2 算法分析

字符串匹配问题如下：给定一个长度为 n 的字符数组，作为文本 T，以及一个长度为 m 的字符数组，作为模式 P，要求算出 P 在 T 中出现的位置。

2.1 Brute-Force 算法

朴素的匹配算法，即从头遍历文本 T，逐个比较模式串 P。如果这 m 个字符都相同，那么输出相应的偏移。否则，模式串向前移动一位，继续进行匹配。
没有对模式串进行预处理。匹配过程的时间复杂度为$\Theta\big((m-n+1)m\big)$。

2.2 KMP 算法

首先对模式 P 进行预处理，计算 P 的前缀函数 π，满足

$$\pi[q]=\max\{k:k<q,P_k \sqsupset P_q\}$$

即$\pi[q]$是P_q的真后缀的最长前缀长度。
在遍历比较时，如果发现文本在$P[q+1]$处不匹配，那么移动到$\pi[q]$继续进行匹配。
分析可知，计算前缀函数的时间为$\Theta(m)$，匹配过程的时间为$\Theta(n)$。

2.3 BM 算法

首先对模式 P 进行预处理，计算“坏字符”与“好后缀”这两类优化的数组。从头遍历文本，但是从模式串的右边开始匹配检查的，通过上述两种优化，尽可能往后跳过。
两个数组的定义为：

$$bmBc[i]=\begin{cases}\min\{i:1\le i\le m-1,P[m-1]=c\}, if\ c\ occurs\ in\ P\\ m, otherwise\end{cases}$$

$$bmGs[i]=\min\{s>0:Cs(i,s)\ and\ Co(i,s)\ hold\}$$

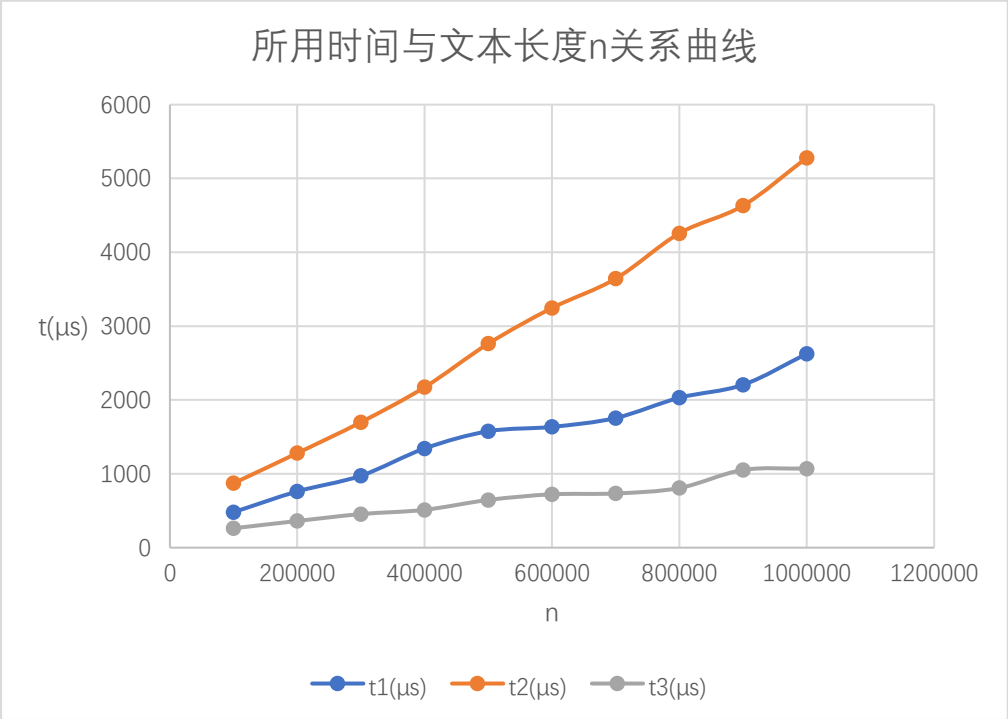
预处理的时间为$\Theta(m+|\Sigma|)$，匹配时间复杂度为$O(mn)$，但最好情况，即复杂度的下界是$\Omega(\frac{n}{m})$。

3 实验设计

设计了能够生成一定数量的文本 txt 和模式串的程序。输入所需的文本串的字符数 n 和所需的模式串的字符数 m，会输出所需模式串，生成一个单行 txt 文本存储文本串。文本为随机生成，选取的字母表包括 26 个英文字母（小写），10 个数字，若干英文标点符号，以及空格符。模式串是选取文本串的随机某个位置的连续 m 个字符，便于测试。
将这 3 个算法分别编写代码，编译生成 3 个可执行文件，第一行输入 txt 文件的路径，第二行输入所需匹配的模式串。程序会逐行进行匹配，输出每一行匹配到的位置，以及算法运行的时间。

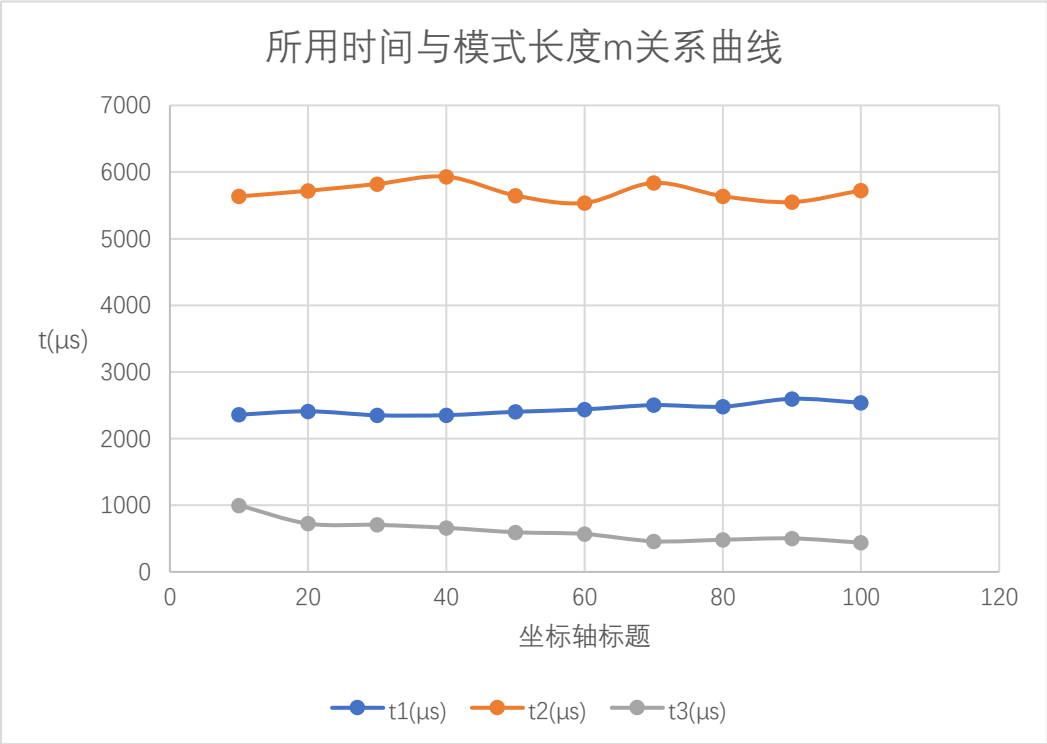
4 结果分析

从匹配结果来看，三种算法的匹配都是正确的，可以通过文本搜索模式串的位置进行验证。固定模式串的长度 m 为 10，设定 n 的大小为 100000 到 1000000，分别进行测试，记录三种算法所需时间，绘制曲线如下。



从图中可以看到，三种算法的时间复杂度与 n 基本是呈线性增长的关系，KMP 算法所需时间是最长的。随着 n 增大，与另外两种算法的时间的差距逐渐增大。在 n 小于 1000000 的情况下，BM 算法的时间基本低于 1ms，增长平缓。Brute-Force 算法虽然不是最快的，但时间也还可以接受，甚至比 KMP 的时间还快些。

固定文本的长度 n 为 1000000，设定 m 的大小,为 10 到 100，分别测得三种算法所用的时间，绘制曲线如下。



可以看到，Brute-Force 算法和 KMP 算法，时间随 m 的变化不是很明显，这是由于 n 远大于 m 的时候，这两种算法的时间复杂度主要取决于 n 的情况，所以时间基本不会有大的变化。而 BM 算法的时间随 m 的增加而下降比较明显，由于算法的时间复杂度的下界是$\Omega(\frac{n}{m})$，当 m 越大，实际匹配时，跳过的距离也会更大些。

5 结论

综合上述分析可得，三种算法在处理文本长度 n 在 1000000 的数量级的情况下，运行时间还是比较短的。其中，Brute-Force 算法虽然理论时间复杂度是最大的，但是实验结果表明，其性能并不算差。而 KMP 算法的理论时间复杂度比较低，但实际应用时的时间甚至比前者还高。BM 算法虽然最坏情况的时间较高，但是时间可达下界较低，实际运行的性能也比较优秀。