**字符串匹配算法综述**

吴浩泽1

（1.同济大学 软件学院，上海 ）

**摘要：**字符串匹配算法一直是研究的热点，本文介绍了几种经典字符串匹配算法以及它们的基本思想，同时也介绍了近年来的高效算法以及改进算法。其中对Sunday算法进行介绍后还进行了算法的改进主要是进行了一个条件判断语句减少了无意义的匹配次数，提高了算法的执行速度。

**关键词：**软件工程；模式匹配；字符串；算法

**中图分类号：**TP301.6 **文献标志码：**A

**String matching algorithm optimization**

WU Haoze1

(1.Tongji University School of Software, Shanghai )

**Abstract:** String matching algorithms have always been a hot research topic. This article introduces several classic string matching algorithms and their basic ideas. It also introduces efficient algorithms and improved algorithms in recent years. Among them, after introducing the Sunday algorithm, the algorithm is improved. The main reason is to carry out a conditional judgment sentence to reduce the number of meaningless matches and improve the execution speed of the algorithm.

**Keywords:** software engineering; pattern matching; string; algorithm

字符串的匹配问题就是对于给定的一个字符串和—个待匹配的字符串，如何能在最短的时间内确定待匹配的字符串在原字符串中是否出现过，如果出现过就返回其出现的位置。计算机网络的出现串匹配算法发展了几十年，即使是现在，人们仍在也对字符串匹配算法进行研究。

在现实生活中串匹配算法有着非常重要的应用，主要包括信息检索以及过滤、病毒检测、入侵检测、等领域。在不断寻找时间复杂度最低的算法过程中，学者们想出许多高效的匹配算法。但在字符匹配算法中存在几个定理:一个是“任何的字符串匹配算法在最糟的情况下需要检查至少n-m+1个text中的字符”，这说明算法的平均性能可以得到提高，但最糟情况下的结果是一个严格的限制。另一个定理是“任何字符匹配算法至少检查n/m个字符”，这个是显而易见的。基于以上的结论，接下来研究的主要方向是设法提高算法的平均性能和减少资源耗费从而做到减少匹配所耗费的时间。

一个人尽皆知的事实是“算法的思想越简单，实际应用的效果越好”。在字符串匹配研究领域中，在众多算法中最著名的当属KMP算法，而本文主要介绍了几种经典的字符串匹配算法：比如BF、KMP、BM、Sunday这些经典算法[1]，同时本文也介绍了对Sunday算法的改进算法Rosunday算法，提高该算法的搜索效率。

**1 常见字符串匹配算法介绍**

以下将会按照BF、KMP、BM、Sunday以及Rosunday的顺序进行算法介绍[2]。在介绍之前根据需要做出如下假设：原字符串为T,有N个字符；待匹配字符串（即模式串）为P，有M个字符。

**1.1 BF算法**

BF（即Brute Force）是一种暴力检索算法，暴力检索法是最好想到的算法，也是最好实现的算法，但是它比较原始和低效，在情况简单的情况下可以直接使用。

其核心思想是：首先将原字符串和子串左端对齐，逐一比较；如果字符不能匹配，则子串向后移动一位继续比较；如果字符匹配，则继续比较后续字符，直至全部匹配。首先比较 T[1]和P[1]，若相等，则再比较T和P的下一位，一直到P[M]也比较相等为止，则匹配成功。若T[X]和P[X]不相等，则P向右移动一个字符的位置，再依次进行比较。如图1所示为BF算法的核心代码：

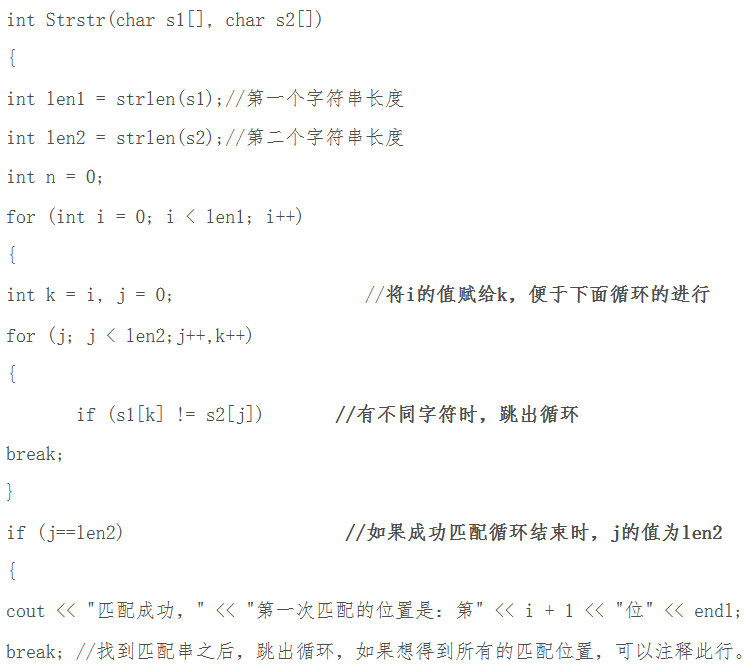


图1 BF算法核心代码

Fig.1 BF algorithm core code

虽然BF算法思想简单，但其处理效率确实低下，该算法最坏的情况会比较M\*（N-M+1）次，该算法的时间复杂度为：O（MN）。故该算法并不能得到广泛接受。而接下来介绍的KMP算法虽然思路比BF算法复杂，但其性能优越，被广泛使用以及传播。

**1.2 KMP算法**

KMP算法（即Knuth-Morris-Pratt Algorithm）核心思想是：在发生失配时，利用已经得到的部分匹配的结果将模式串右移尽可能远的距离，继续进行比较。与BF算法不同的是，待匹配串不一定向右移动一个字符的位置，右移也不一定必须从待匹配串起点处重新试匹配，即待匹配串一次可以右移多个字符的位置，右移后可以从待匹配串起点后的某处开始试匹配。

首先建立next数组：找出前缀before，设为before0~m]；再找出后缀post，设为post[0~n]。从前缀before里，先以最大长度的s[0~m]为子串，即设k初始值为m，跟post[n-m+1～n]进行比较，如果相同，则before[0~m]则为最大重复子串，长度为m，则k=m；如果不相同，则k=k-1；缩小前缀的子串一个字符，再和后缀的子串按照尾巴对齐，进行比较是否相同。如此下去，直到找到重复子串，或者k没找到。

其次进行比较操作：规定i是主串S的下标，j是模式T的下标。现在假设现在主串S匹配到 i 位置，模式串T匹配到 j 位置。

如果j = -1，则i++、j++，继续匹配下一个字符；

如果S[i] = T[j]，则i++、j++，继续匹配下一个字符；

如果j != -1，且S[i] != P[j]，则i不变，j = next[j]。此举意味着失配时，接下来模式串T要相对于主串S向右移动j - next [j] 位。如图2所示为KMP算法的核心代码：

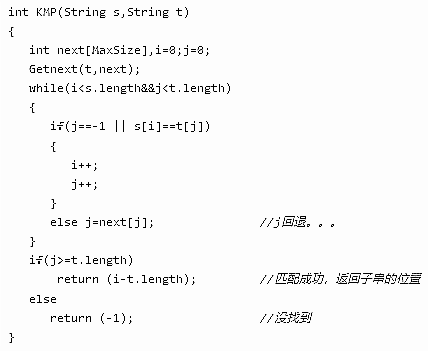


图2 KMP算法核心代码

Fig.2 KMP algorithm core code

整个算法最坏的情况是i移动到终点，仍然模式不匹配原字符串。所以花费时间是线性的O(n)。再加上预处理待匹配串的时间O(m)，该算法时间复杂度是O(n+m)。虽然该算法效率不低，但是这个算法的不易理解也成为一个令人头疼的问题。而接下来介绍的算法不仅相对容易理解，其运行效率也比KMP还要略高。

**1.3 BM算法[3]**

BM算法（即Boyer-Moore）法是目前相当有效并且也容易理解的一种，就一般情况而言，它比KMP 算法快大约3倍左右。在移动待匹配串的时候是从左到右，而进行比较的时候是从待匹配字符串的右端开始。一般来说BM算法的核心是理解坏字符以及好后缀规则。这两种规则的最终目的就是让待匹配串每次向右移动尽可能大的距离。再用下面一个例子进行详细说明：

假定原字符串为"HERE IS A SIMPLE EXAMPLE"，待搜索字符串为"EXAMPLE"。

首先，将两者头部对齐，从尾部开始比较，如果尾部字符不匹配，那么只要一次比较，就可以知道该位置肯定不是要找的结果。我们可以看到"S"与"E"不匹配，这时，"S"就被称为"坏字符"。我们还发现，"S"不包含在搜索词"EXAMPLE"之中，这意味着可以把搜索词直接移到"S"的后一位。

依然从尾部开始比较，发现"P"与"E"不匹配，所以"P"是"坏字符"。但是，"P"包含在搜索词"EXAMPLE"之中。所以，将搜索词后移两位，两个"P"对齐。我们由此总结出"坏字符规则"：

后移位数 = 坏字符的位置 - 搜索词中的上一次出现位置（如果"坏字符"不包含在搜索词之中，则上一次出现位置为 -1。）

依然从尾部开始比较，可以得到"MPLE"与"MPLE"匹配。我们把这种情况称为"好后缀"（good suffix），即所有尾部匹配的字符串。（注："MPLE"、"PLE"、"LE"、"E"都是好后缀）

比较前一位，发现"I"与"A"不匹配。所以，"I"是"坏字符"。根据"坏字符规则"，此时搜索词应该后移 2 -（-1）= 3位。但此时也可以采用"好后缀规则"：

后移位数= 好后缀的位置 - 搜索词中的上一次出现位置。

此时，所有的"好后缀"（MPLE、PLE、LE、E）之中，只有"E"在"EXAMPLE"还出现在头部，所以后移 6 - 0 = 6位。可以看到，"坏字符规则"只能移3位，"好后缀规则"可以移6位。所以，Boyer-Moore算法的基本思想是，每次后移这两个规则之中的较大值。而且这两个规则的移动位数，只与搜索词有关，与原字符串无关。

接着继续从尾部开始比较，"P"与"E"不匹配，因此"P"是"坏字符"。根据"坏字符规则"，后移 6 - 4 = 2位。从尾部开始逐位比较，发现全部匹配，于是搜索结束。如果还要继续查找（即找出全部匹配），则根据"好后缀规则"，后移 6 - 0 = 6位，即头部的"E"移到尾部的"E"的位置继续进行。

BM算法核心代码如图3所示：

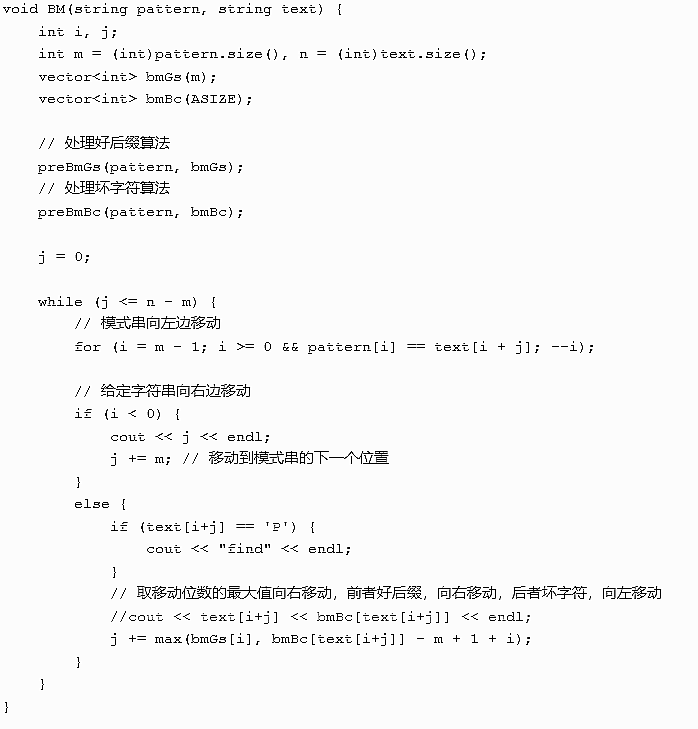


图3 BM 算法核心代码

Fig.3 BM algorithm core code

以上描述了BM算法的一般思路，思想并不复杂，其匹配的时间复杂度最差是O(m\*n)，而最好的情况是O（n/m）。经过实践的证明虽然其最差时间复杂度比KMP算法高，但其平均性能还是优于KMP算法。

**1.4 Sunday算法**

在经典算法的不断改进中出现了一种更优的算法，性能高于KMP以及BM算法，并且也容易理解，就是这个Sunday算法。其核心思想[4]是：在匹配过程中，模式串发现不匹配时，算法能跳过尽可能多的字符以进行下一步的匹配，从而提高了匹配效率。

与BM算法一样我们先假定原字符串为"THIS IS A SIMPLE EXAMPLE"，待搜索字符串序列为"EXAMPLE"。

首先原字符串和子串左端对齐，发现"T"与"E"不匹配之后，检测原字符串中待匹配字符串长度加一位的字符（在这个例子中是"IS"后面的那个空格）是否在子串中出现，如果该字符没有出现在待检测的字符串中则直接将子串头移动到该字符的下一个位置。（这里空格没有在子串中出现，移动子串到空格的下一个位置"A"） 。如果该字符出现在待匹配字符中则移动子串将两者对齐（之后会看到）。

移到下一位置之后发现"A"与"E"不匹配，但是原字符串中下一个字符"E"在子串中出现了。且第一个字符和最后一个字符都有出现，那么首先移动子串靠后的字符与原字符串对齐：发现空格和"E"不匹配，原字符串中下一个字符“空格”也没有在子串中出现，所以直接移动子串到空格的下一个字符"E"。这样从头开始逐个匹配，最终匹配成功。如下图4为Sunday算法的伪代码[5]实现：

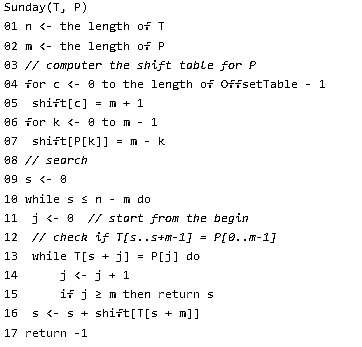


图4 Sunday 算法伪代码

Fig.4 Sunday algorithm pseudo code

Sunday算法在所有的经典模式匹配算法中执行效率最快。最坏情况下时间复杂度为：O(n\*m)，平均时间复杂度：O(n)。

**2 基于Sunday算法的当前最新改进[6]**

RoSunday算法和Sunday算法最大的区别就是while循环的开始加了一个if条件语句，是用来判断模式串的最后一个字符位置在主串中对应的位置上的字符有没有在模式串中出现过。

**2.1 Rosunday算法[7]**

Sunday算法每次都是从首字符开始依次匹配，直到发生不匹配时才进行跳转，进入下一轮匹配。为了提高算法的执行效率，Rosunday算法对Sunday算法进行改进的核心思想是让字符先转移再匹配，以减少匹配次数。

这就是Sunday算法的优化。算法在匹配前先检验模式串的最后一个字符位置在文本串中对应的位置的字符是否在模式串中出现，如果没有则从该字符的下一位进行下一轮循环，移动步长=模式串长度，这样就用一次判断避免了多于的匹配操作，提高了执行速度。

这样的说法可能不利于理解，以下两个图对比增强对改进部分的理解，如下图5（Sunday算法基本匹配过程）以及图6（RoSunday算法基本匹配过程）。

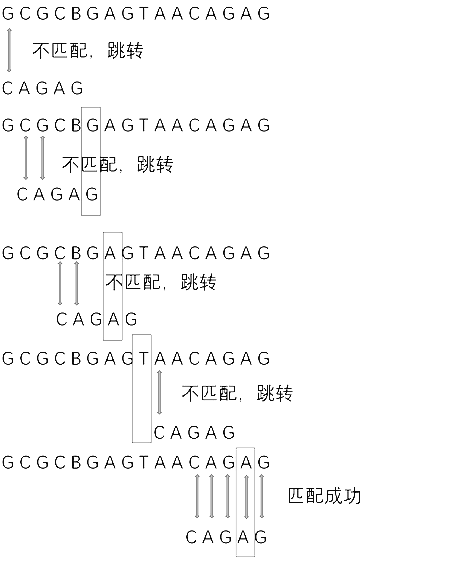


图5 Sunday算法基本匹配过程

Fig.5 The basic matching process of Sunday algorithm

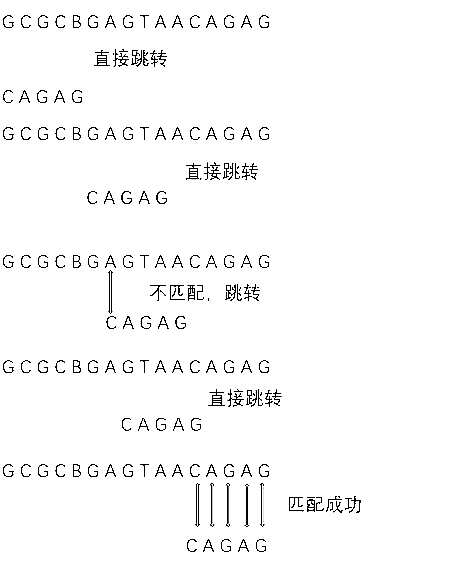


图6 RoSunday算法基本匹配过程

Fig.6 The basic matching process of RoSunday algorithm

由以上可以看出上文中看到使用 Sunday算法进行模式匹配经历了10次匹配待匹配串才匹配成功，而这里使用改进后的RoSunday算法只需要6次匹配就匹配成功了，极大地大提高了匹配执行的效率。

RoSunday算法是在Sunday算法基础上改进而成，思想简单易懂，经查阅该算法已经用于自主研发的网络监测与审计系统中，系统性能得到进一步优化。如下图7为RoSunday算法的核心代码：

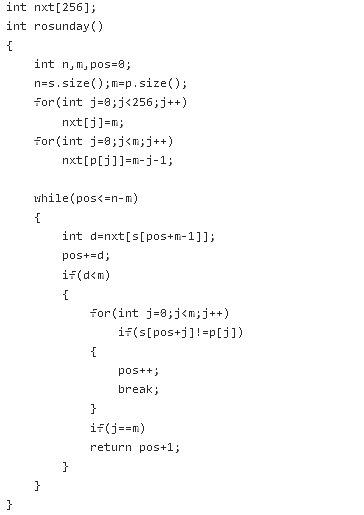


图7 RoSunday 算法核心代码

Fig.7 RoSunday algorithm core code

自Sunday算法出现以来在模式匹配算法中一直占据着重要位置,很多研究者都对该算法进行了改进，但是绝大部分的改进算法都是对Sunday进行预处理。而这种对Sunday算法本身的算法思想做了修改，简单的说就是把字符先匹配后转移的思想改成了先转移后匹配，具有较高的创新性。

**3 结束语**

本文第一部分介绍了几个较为著名的基本算法，它们根据列出的顺序是依次优化的过程，逐步减小运行的时间消耗。

而第二部分主要比较了经典Sunday算法以及新的改进算法的比较，此新改进的算法对原有的Sunday算法进行了思想上的改进，给出RoSunday算法的实现过程。

**参考文献：**

1. Mr.chen.字符串匹配算法.2019-10-13[2020-12-24].https://www.cnblogs.com/chentianwei/p/11665656.html

1. [arthur.dy.lee](https://blog.csdn.net/paincupid" \t "https://blog.csdn.net/paincupid/article/details/_blank).字符串匹配算法综述.2018-7-22[2020-12-24].https://blog.csdn.net/paincupid/article/details/81159320
2. 阮一峰.字符串匹配的Boyer-Moore算法.2013-5-3[2020-12-24].http://www.ruanyifeng.com/blog
3. Switchvov.字符串匹配——Sunday算法.2016-07-08[2020-12-24].https://blog.csdn.net/q547550831/article/details/51860017
4. Sunday D M.A very fast sub string search algorithm 【J】Communication of ACM，1990，33(8)：132—142
5. 万晓檎，杨渡，樊自甫.改进的Sunday模式匹配算法【J】．计算机工程，2009，35(7)：125—129．
6. 徐珊，袁小坊，王东.Sunday字符串匹配算法的效率改进【A】.计算机工程与应用，2011，47(2;p99)：96—98．