# 字符串匹配算法

实现的算法为BM算法和Sunday算法，见BM\_and\_Sunday.cpp

1. Rabin-karp算法

哈希匹配法，即将原先主串和模式串的字符比较转换成对应哈希值比较，这样的要容易得多。具体的哈希转换方法有很多，例如，按位相加法、转换成26进制法。由于主串通常要长于模式串，把整个主串转化成hashcode是没有意义的，只有比较主串当中和模式串等长的子串才有意义，当哈希值相等时，再模仿BF算法，进行两个字符串的逐项比较。

hashcode = hash（string）

1. 哈希法可以通过优化字符串累加方法，即新子串的hash值都用上一次子串进行简单的增量来计算；

（2）通过使用优化方法，最终RK算法的时间复杂度可以优化为O(n);

（3）与BF算法相比，免去了很多无谓的字符比较，时间复杂度上有很大提高；

（4）RK算法的缺点在于哈希冲突，每一次哈希冲突时都要进行子串和模式串的逐个比较，如果冲突过多，RK算法就会退化成BF算法。

1. 有限自动机算法

一个有限自动机 M 是一个 5 元组（Q，q， A， ∑，δ），其中：

（1）Q 是状态的有限集合

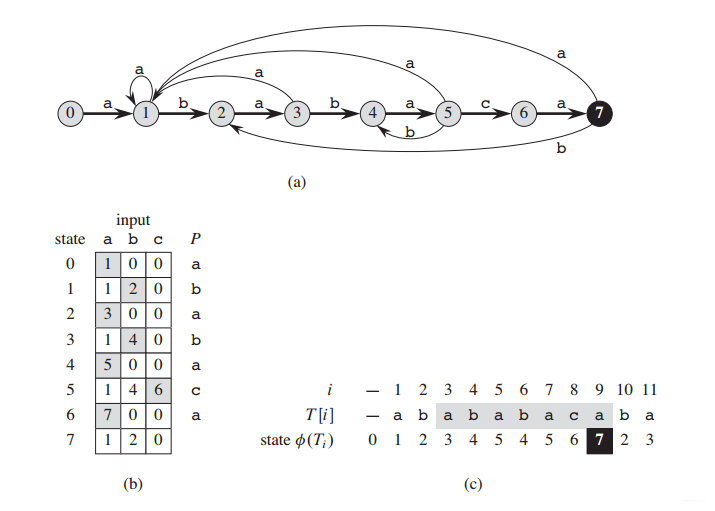
（2）q∈Q 是一个初始状态

（3）A 包含于 Q 是一个特殊的接受状态集合

（4）∑ 是有限输入字母表

（5）δ 是一个从 Q✖∑ 到 Q 的函数，称为 M 的转移函数

下图为有限自动机的状态转移图，模式串为 ababaca 。



转移函数算法伪代码：

其中， Pk ⊇ Pqa 表示 Pk是 Pqa的后缀。

COMPUTE-TRANSITION-FUNCTION(P,∑)

m = P.length

for q = 0 to m

for each charater a∈∑

k = min(m+1,q+2)

repeat

k = k - 1

until Pk ⊇ Pqa

δ(q,a) = k

return δ

有限自动机算法伪代码：

FINITE-AUTOMATON-MATCHER(T,δ,m)

n = T.length

q = 0

for i = 1 to n

q = δ(q,t[i])

if q == m

print "Pattern occurs with shift" i-m

1. Boyer-moore算法

（1）坏字符规则

即是指模式串和子串当中不匹配的字符，当模式串和主串的第一个等长子串比较时，从右侧开始确定坏字符（检测顺序相反，是从字符串最右侧向最左侧检测），如果坏字符在模式串中不存在，则直接把模式串挪到主串坏字符的下一位。

坏字符的位置越靠右，下一轮模式串的挪动跨度就可能越长，节省的比较次数也就越多；

（2）好后缀规则

好后缀就是指模式串和子串当中相匹配的后缀。即当子串和模式串不匹配时，但模式串和子串存在好后缀，且在模式串中可以找到与好后缀相同的片段，这样就可以直接移动模式串中的相同片段与模式串的好后缀对齐，从而实现快速位移；但是当不存在这样的相同片段时，切记不可一次性把模式串移动到好后缀的后面，要判断模式串的前缀是否与好后缀的后缀相匹配，以免移动过多而错过。

何时采用坏字符或者好后缀规则，并没有直接结论，需要分别计算下一轮模式串移动的长度并进行比较，可以使模式串移动更多的规则，就是更好的方法。

1. Simon算法：未能找到。
2. Colussi算法：未能找到。
3. Galil-giancarlo算法：未能找到。
4. Apostolico-crochemore算法

horspool算法将主串中匹配窗口的最后一个字符跟模式串中的最后一个字符比较。如果相等，继续从后向前对主串和模式串进行比较，直到完全相等或者在某个字符处不匹配为止（如下图中的f与d失配） 。如果不匹配，则根据主串匹配窗口中的最后一个字符在模式串中的下一个出现位置将窗口向右移动。（跳跃式匹配）

1. shift-or算法

设模式字符串为P，它主要通过保存一个集合D（D中记录了P中所有与当前已读text的某个后缀相匹配的前缀），每当text有新的字符读入，算法利用位并行机制来更新这个集合D。

设P长度为m，则集合D可表示为D = dm…d1 而用D[j]代表dj

D[j]=1当且仅当p1…pj 是 t1…ti 的某个后缀，当D[m]=1时，就认为P已经于text匹配。

当读入下一个字符 ti+1, 需要计算新的集合 D′. 当且仅当D[j]=1并且 ti+1 等于 pj+1时D'[j+1]=1. 这是因为D[j]=1时有 p1…pj 是 t1…ti 的一个后缀，而当ti+1 等于 pj+1可推出p1…pj +1是 t1…ti+1 的一个后缀.这个集合可通过位运算来更新.

算法首先建立一个数组B, 数组长度为text串所属字符集长度（例如A-Z的话数组B的长度为26.） 如果P的第j为等于c则将B[c] 中第j位置为1.

因为要预处理计算B，如果字符集很大的话，并不划算。如果m很长的话（大于机器字长），也很不方便。所以这种算法适用于字符集较小，模式串小于机器字长的情况。当然对于模式串较长的情况，也是比brute force要快的，只是逻辑上要复杂些。

Shift-Or算法跟Shift-And算法思想是一样的，只是在通过取补，减少了位运算的次数，提高了速度。Shift-Or作的修改是，用零表示一个数在集合里，1表示不在，所以

D = ((D<<1) | 1) & B[s[i]];

修改为D=D<<1 | B[s[i]]; 省了一次位运算，当然B和D的初始化的时候，也要作相应的修改。

1. Sunday算法

给定的字符母串S（长度为n）和子串T（长度为m），从左到右开始匹配，之后出现不匹配的情况后，判断母串S参与匹配的最后一位的下一位字符，如果该字符出现在子串T中，选择子串中最右面出现的位置进行对齐；否则直接跳过该匹配区域。平均时间复杂度是O（n），最坏降低到O（n\*m）