基于后缀搜索的多模式匹配算法——Wu-Manber 算法

https://memorycn.wordpress.com/2011/11/05/matching_algorithm_-_wu-manber_algorithm_based_on_the_the_suffix_search_of_multi-mode/

Wu-Manber 算法采用跳跃不可能匹配字符和 hash 散列的方法，加速匹配的进行。该方法需要对所有模式进行预处理，构建 SHIFT，HASH 和 PREFIX 这 3 个表。SHIFT 表用来存储字符集中所有块字符在文本中出现时的转移距离；HASH 表用来存储匹配窗口内尾块字符散列值相同的模式串；PREFIX 表用来存储匹配窗口内首块字符散列值相同的模式串。在对模式串进行匹配的时候就是利用这三个表完成文本的扫描和寻找匹配的过程。

首先介绍预处理过程：

1、计算模式集合 P 中最短的模式长度 m。后续讨论仅考虑每个模式串的前 m 个字符，设这些长度为 m 的模式串组成新集合 P'。

2、对 P'中每个模式串进行分块，以 B 个字符为块长度，每次比较长度为 B 的块。推荐取 $B=\log_{|\Sigma|}(2*m*k)$，其中 k 是模式串的个数。

3、构建一个 SHIFT 表，该表用于在扫描文本串时，根据读入的块决定移动的距离。对于|Σ|大小的字符集，长度为 B 的块的组合方式有$|\Sigma|^B$种可能，因此表的大小为$|\Sigma|^B$。

4、将每个长度为 B 的块用哈希函数计算出一个整数值 h，将 h 为 SHIFT 表的索引值。

5、穷举 P'中所有长度为 B 的块，对于每个块 Bl，计算其相应的 SHIFT 表值。计算规则如下：找出 Bl 在 P'的每个模式串中出现的最右位置，设这些位置中的最大值为 j（以末尾位置为准），则 SHIFT[h]=m-j。[注 1]

6、其余所有不在 P'中的块，SHIFT 表值为 m-B+1。[注 2] [注 3]

7、构建 HASH 表：设 HASH[h]=p，p 指向两个单独的表：PAT_POINT 和 PREFIX[注 4]。有一个排序过的（根据模式串末 B 位的哈希值排序）指针链表 PAT_POINT，存储着指向所有模式串的指针。p 指向 PAT_POINT 中末 B 位哈希值为 h 的第一个节点。

8、构建 PREFIX 表：将每个模式串前 B'位（B'的推荐值为 2）的哈希值存入 PREFIX 表[注 5]，用于检查前缀是否匹配，可以进一步减少需要朴素匹配的模式串个数。

接下来介绍匹配扫描过程：

1、计算文本串当前匹配窗口中 m 个字符的末 B 位的哈希值 h。

2、检查 SHIFT[h]的值，如果 SHIFT[h]>0，就将窗口向右移动 SHIFT[h]位，并返回第 2 步；否则 SHIFT[h]=0 时，进入第 4 步。

3、计算文本串当前匹配窗口中的前 B'位的哈希值，记为 prefix_hash。

4、对符合 HASH[h]<= ptr <HASH[h+ 1]的每一个 ptr 值（以机器的指针长度为步进），检查是否存在 PREFIX[ptr]=prefix_hash。如果有，就对文本串和模式串进行朴素匹配 。

Wu-Manber 算法的时间复杂度平均情况是 O(BN/m)。该算法对 m 敏感，SHIFT 函数的最大值受 m 的限制，如果最 m 很小，则移位的值不可能很大，因此对匹配过程的加速有限。

最后放一个来自《Wu-Manber 算法性能分析及其改进》的例子。懒得复制排版了，直接截图好了。

例如: 在文本串“All of the students are very cool in this school.”中匹配模式串 student、crude 和 school。假设 B= 2。

(1) 计算 m= 5, 即匹配窗口大小为 5, 如图 1 所示。

(2) 计算 SHIFT 表

考虑每一个模式串的前 5 个字符, 计算每个块字符与匹配窗口内模式串串尾的距离如图 2 所示, 此即当该块字符在文本中出现时的转移距离。

student:

st	tu	ud	de
3	2	1	0

crude:

cr	ru	ud	de
3	2	1	0

school:

sc	ch	ho	oo
3	2	1	0

图 2 块字符转移距离示意图

合并后的 SHIFT 表如图 3 所示, 其它未出现在匹配窗口内的块字符的 SHIFT 表值均为 $m-B+1$= 4:

st	tu	ud	de	cr	ru	sc	ch	ho	oo	…
3	2	1	0	3	2	3	2	1	0	4

图 3 SHIFT 表示意图

All of the students are ve

图 1 文本匹配

(3) 计算 HASH 表, 如图 4 所示

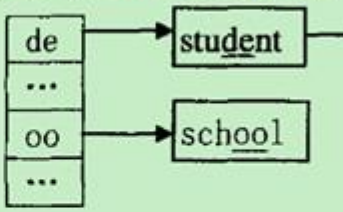


图 4 HASH 表

(4) 计算 PREFIX 表, 如图 5 所

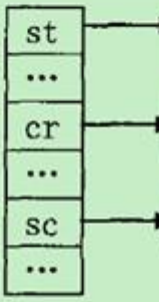


图 5 PREFIX 表

(5) 匹配过程。

All of the students are very cool in this school.

图 6 匹配过程

如图 6 所示: 从右向左扫描前 5 个字符, o 在 SHIFT 表中值为 4, 可以将考察的位置向后移动 4 个字符的距离。th 在 SHIFT 表中值也为 4, 所以也将考察的位置向后移动 4 个字符的距离。st 在 SHIFT 表中值为 3, 所以将考察的位置向后移动 3 个字符的距离。de 在 SHIFT 值为 0, 转入 HASH 表, 在 HASH 表中对应的模式串有

算当前文本窗口 stu de 的 text

入 PREFIX 表对应的模式串有 stu

剩余文本匹配过程类似。

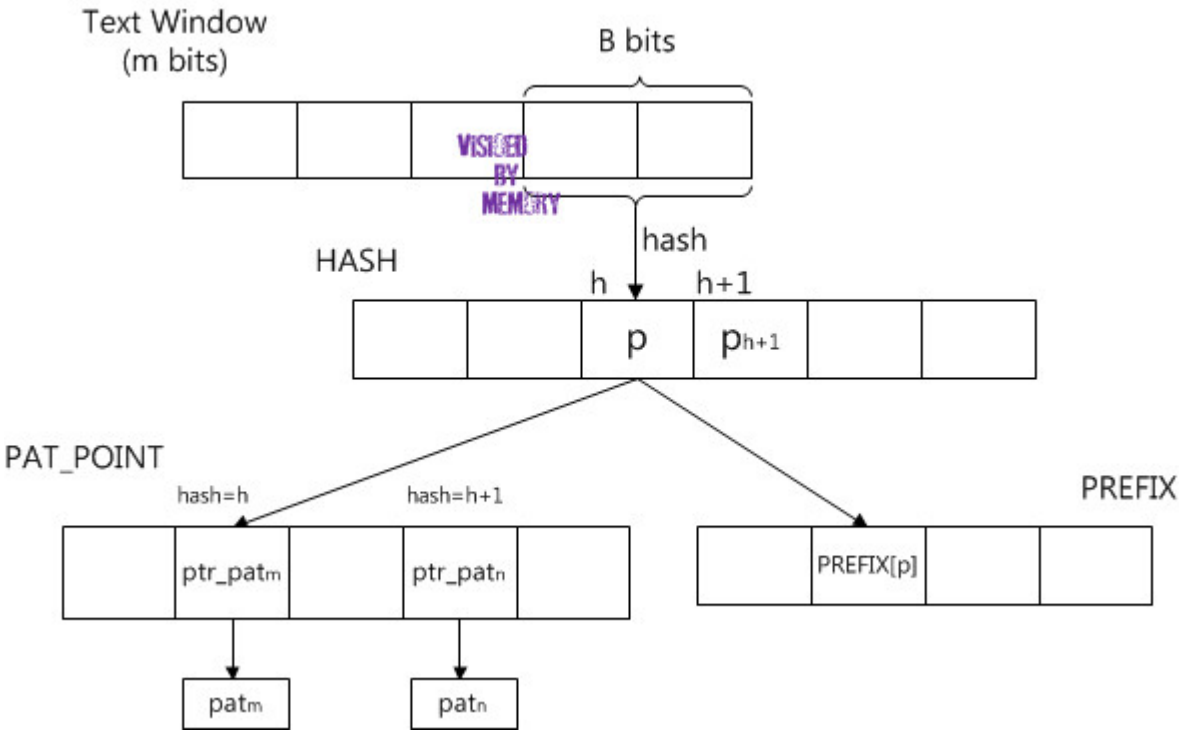
[注 1] 等同于 BM 算法的“坏字符规则”

[注 2] SHIFT 表值为 m-B+1 的原因：这 B 个字符不出现在 P'中，说明最多可能有(B-1)个字符出现在 P'中，因此 SHIFT 表值为 m-(B-1)=m-B+1。

[注 3] 预处理的 5、6 步与原文表述的顺序不同，但效果是一样的。原文的做法是先将 SHIFT 表所有位置初始化为 m-B+1，再根据本文的第 5 步更新 SHIFT 表值。参见原文第 4 页第 1 自然段。

[注 4] 此处的表述是按照自己的理解总结得出的，可能不正确。原文第 4~5 页表述如下："Let h be the hash value of the current suffix in the text and assume that SHIFT[h] = 0. The value of HASH[h] is a pointer p that points into two separate tables at the same time: We keep a list of pointers to the patterns, PAT_POINT, sorted by the hash values of the last B characters of each pattern. The pointer p points to the beginning of the list of patterns whose hash value is h. To find the end of this list, we keep incrementing this pointer until it is equal to the value in HASH[h+1] (because the whole list is sorted according to the hash values). So, for example, if SHIFT [h] 0, then HASH [h] = HASH [h+1] (because no pattern has a suffix that hash to h). In addition,
we keep a table called PREFIX, which will be described shortly."

[注 5] 预处理过程 SHIFT、HASH 和 PREFIX 表的内存位置关系图（根据个人对原文的理解画出，可能不正确）：



参考文献

[1] 《A Fast Algorithm for Multi-Pattern Searching》

[2] 《Wu-Manber 经典多模式匹配算法》

[3] 《多模式匹配算法及硬件实现》

[4] 《Wu-Manber 算法性能分析及其改进》