TELKOMNIKA, Vol.11, No.3, March 2013, pp. 1497 ~ 1505
ISSN: 2302-4046

Multi-pattern Matching Methods Based on Numerical Computation

Lu Jun*[1,2], Zhang Zhuo[1,2], Mo Juan[1,2], Lv Xingfeng[1]
[1]College of Computer Science and Technology, Heilongjiang University, Harbin, China
[2]Key Laboratory of Database and Parallel Computing of Heilongjiang Province, Harbin, China
*Corresponding author, e-mail: lujun111@yahoo.com.cn

Abstract

Multi-pattern matching methods based on numerical computation are advanced in this paper. Firstly it advanced the multiple patterns matching algorithm based on added information. In the process of accumulating of information, the select method of byte-accumulate operation will affect the collision odds , which means that the methods or bytes involved in the different matching steps should have greater differences as much as possible. In addition, it can use balanced binary tree to manage index to reduce the average searching times, and use the characteristics of a given pattern set by setting the collision field to eliminate collision further. In order to reduce the collision odds in the initial step, the information splicing method is advanced, which has greater value space than added information method, thus greatly reducing the initial collision odds. Multiple patterns matching methods based on numerical computation fits for large multi-pattern matching.

Keywords*: Single pattern matching, Multi-pattern matching, Collision*

1. Introduction

Pattern matching is an essential technology in computer knowledge. It is widely applied in text editing, literature searching, natural language identifying, biology, image manipulation, information security, network etc. There are two types of pattern matching: single pattern matching and multiple patterns matching. In simple single pattern matching algorithm, a pattern matching procedure is looked as key word (pattern string) searching. It divides the text with the length n into n-m+1 sub-string with the length m and detects whether each sub-string matches to pattern string with length m or not. Classical single pattern matching algorithm includes KMP [1] algorithm and BM [2] algorithm. Other algorithm is mostly improved on these classical algorithms [3-5]. KMP algorithm analyzes the distribution characteristic of pattern string and makes use of this in pattern matching to advance matching efficiency. In BM algorithm, pattern string moves from left to right, while character is compared from right to left. When it loses matching, the predefined excursion function adopts the max value to decide right shift value. So it realizes jumping as far as possible and reduces the times of matching.

Related work. Multiple pattern matching is an extension of single pattern matching, including AC [6] algorithm based on automata, Wu-Manber [7] algorithm, multiple pattern matching algorithm based on sequential binary tree [8], fast matching algorithm based on cross data link table[9], etc. Moreover, there are other improved algorithms [10,11] derived from these algorithms. A searching tree is constructed based on pattern set in AC algorithm. The size of the searching tree is related to the size of character set, so the biggish memory is needed. Wu-Manber algorithm is multiple pattern matching algorithm and it extends BM algorithm in single pattern matching. This algorithm adopts bad character shifting mechanism of BM algorithm and makes use of block character to extend efficiency of bad character shifting. It also uses hash Table to select pattern string matched in matching stage and reduce more computing. The average capability of Wu-Manber algorithm is the best in application.

2. Limitation of Existent Pattern Matching Algorithms

Existent pattern matching algorithms restrict a pattern as “character string”. The whole pattern set is not been disposed as a body in matching procedure. It takes at most out part of common prefixes or postfixes to reduce the times of matching.

Received October 23, 2012; Revised January 18, 2013; Accepted January 31, 2013

Some pattern matching algorithms have requirements for language. These algorithms are fit for English and unfit for Chinese. They are aimed at English. In the case of thousands of patterns, the advance of efficiency of these algorithms is notable. But there are some questions in dealing with multiple patterns matching in Chinese.

The efficiency of BM algorithm is preferably in single pattern matching. Some improved algorithms derive from BM. These algorithms adopt compared manner from back to front and generate jumping Table based on current matched characters. All these reduce matching times. When we anatomize these algorithms based on character jumping, we will find that matching efficiency is not essentially advanced by analyzing character of pattern strings to realize jumping, because it needs time to judge jumping. For example, Horspool advanced Boyer-Moore-Horspool algorithm. This algorithm is an improved algorithm of BM. We call it BMH [12] algorithm. In this algorithm, the character pointed by text pointer is firstly compared with the last character of pattern. If it is equality, then compare the rest m-1 characters. No matter which character in the text results in failed matching, pattern will slide to the right according to the last character in the text that corresponds to pattern. Figure 1 shows the matching process in BMH algorithm.

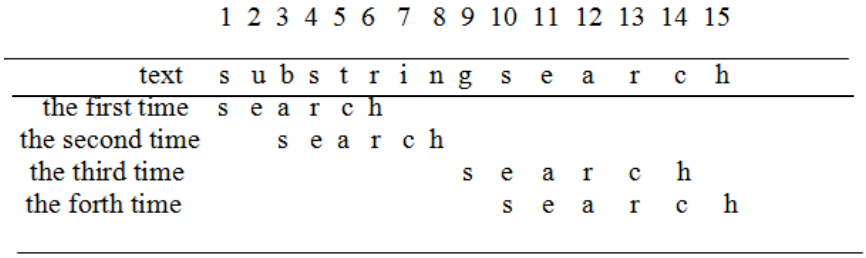

Figure 1. Matching process in BMH algorithm

Figure 1 shows that if pattern string “search” can jump matching, the text T[6] should be searched in pattern string at first matching. It is obvious that d[r]=2, so the pattern slides right 2 characters to the right. In fact, efficiency is not advanced, because it needs 2 times to compare when character ‘r’ is searched in pattern. It is not better to compare the last character ‘h’ in the pattern and each character in the text step by step. It only saves one time in the last successful pattern matching. Apparently, if jumping algorithm is used in parallel multiple patterns matching, the process will be much more complex. It is not favorable to realize parallel operation.

Representative multiple pattern algorithm is Wu-Manber. Wu-Manber algorithm is based on BM, but it uses character block with B length (2 or 3) to reduce the possibility of part matching. Three tables are needed in this algorithm: SHIFT table, HASH Table and PREFIX table. SHIFT[h] can store the number of characters jumped in security, HASH[h] can store pattern chain Table with h computed by the last B characters’ hash value. PREFIX Table filtrates the pattern that has the same hash value but different prefix with the text, so the number of compared patterns will decrease further. But when pattern set becomes bigger, jumping character number will be closer to 1. In this case, correlative judging algorithm will lose efficiency and algorithm becomes more complex. Further more, Wu-Manber algorithm does not make use of more useful information in pattern set.

3. Multi-Pattern Matching Based on Added Information

The whole ideal of the algorithm advanced in this paper is that each pattern in pattern set is not been looked as string. We look each byte in the pattern as a number from 0 to 255. So it overcomes the limitation on language. Then pattern matching is transformed into number disposing. Pattern set is sorted based on the sum of the shortest length of pattern and the index is built for each pattern. To increase searching speed, it adopts AVL tree to store index. Because the maintenance of AVL tree is relatively independent and does not take matching time, it can be ignored. It is important that we firstly judge whether the contents of the shortest continuous patterns from text entrance are present in pattern set. It involves colligate

information about multi-bytes, so matching collision probability is reduced. Only when the sum of the shortest pattern length is the same and collision is eliminated will accurate matching be further achieved. Consequently, matching times are reduced greatly. In the pattern set relate to this algorithm, it is obvious that the bigger the length of the shortest pattern is, the better matching efficiency is.

3.1. Pretreatment of Pattern Set

At first, pattern set needs to be pretreated, that is, obtaining the shortest pattern length (minlen) in the pattern set. To each pattern in this pattern set, the sum of minlen length bytes is computed as $m_1 + m_2 + ... + m_{\min lin}$, then the pattern set is sorted based on the sum. To some patterns, if the sum is the same, they can be sorted based on the value of the bytes again. It is shown as Table 1.

Table 1. The sorted pattern set based on the length of the shortest pattern

Pattern	The sum of the foregoing 4 bytes	1	2	3	4	5	6	7	8	9	10	11	12	13
acaapzy	390	a	c	a	a	p	z	y						
abcdekg	394	a	b	c	d	e	k	g						
cbdaefjlaa	394	c	b	d	a	e	f	j	l	a	a			
dacbfjoire	394	d	a	c	b	f	j	o	i	r	e			
ldifjglk	415	l	d	i	f	j	g	l	k					
jdhkjwe	416	j	d	h	k	j	w	e						
slkdjgioejgr	430	s	l	k	d	j	g	i	o	e	j	g	r	
klkrfhkdfksdl	436	k	l	k	r	f	h	k	d	f	k	s	d	l
ayzxabk	460	a	y	z	x	a	b	k						
xyza	460	x	y	z	a									

In the pattern set of Table 1, the shortest pattern length minlen is 4 (xyza) and the longest pattern length maxlen is 13(klkrfhkdfksdl). The pattern set is sorted based on the sum of the shortest pattern. When the pattern is matched, the sum of the minlen length bytes is computed from the text matching entrance and compares with the sum in the sorted pattern set. The pattern set is filtrated. This collision (e.g.394) odds（1/（256*minlen））is less than the collision odds（1/256）based on the comparison with single byte. So matching efficiency is promoted. It is apparent that the bigger of the shortest pattern length is, the more the pattern number is, and the more obvious the advantage of this algorithm is.

To compare easily, the index should be built on the sorted pattern. It is shown in figure 2.

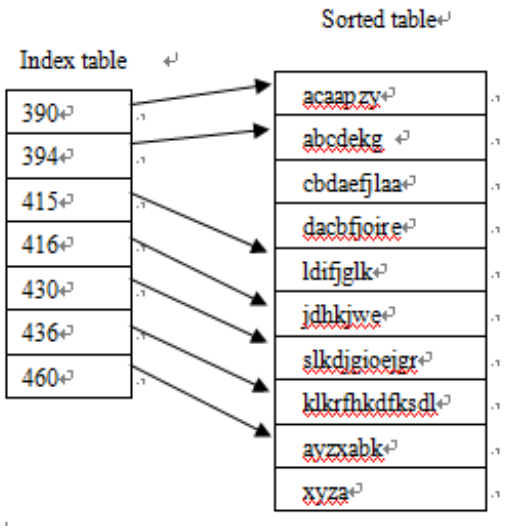

Figure 2. The index on the sorted pattern set

3.2. Built AVL Tree For The Indexes

To quickly locate in matching based on the index, the index Table in Figure 2 should be transformed to AVL tree, so average searching times become the least. Figure 3 shows the AVL tree based on the index table.

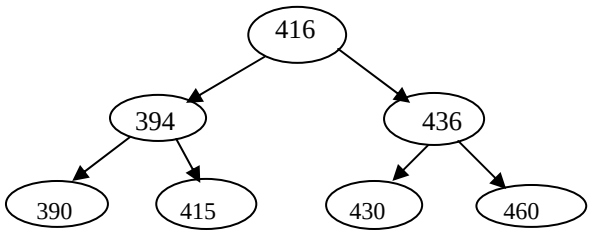

Figure 3. AVL tree

The node's data structure in AVL tree is as follows:

```
struct node
{  int data;
   struct node *llink;
   struct node *rlink;
   struct sortnode *slink;
   int number;
}
```

In the node's data structure, data is the value of the node in AVL tree, such as 394. Llink points to the left node and rlink points to the right node. Slink points to the first node in the pattern sub-set that the sum of the foregoing minlen length bytes in sorted pattern Table is equal to the data value. Number is the number of nodes in the pattern sub-set. It is 1 at least, while the number of nodes in the pattern sub-set with the sum 394 is 3.

3.3. Collision Eliminating

Those patterns with the same foregoing minlen length bytes will collide. To reduce the comparison times of the collision patterns in matching, the pattern set needs to be disposed further. Of course, the disposition does not occupy matching time, because it is completed in pretreatment. The main idea is to search the byte positions with the most difference in the collision patterns and let the text compares directly with the corresponding position of the pattern. Thus comparing times are reduced. In the previous example, the pattern set with index 394 has three patterns. If these patterns are stored in array (suffix is 0~maxlen-1), we search the position with the biggest difference degree. Difference degree is the number of different bytes with the same suffix in pattern set. It is apparent that the biggest difference degree is less than or equal to the pattern number (3) of collision pattern sub-set. It is shown in figure 4.

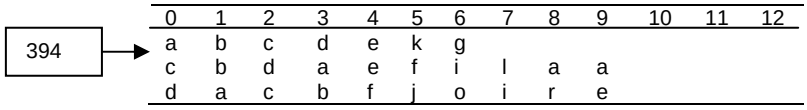

Figure 4. Collide sketch map

In Figure 4, the suffix with the biggest difference degree is 0,3,5,6. We select the smallest 0 and write it down. So we need to add a new field for pattern set—collision field, whose initial value is max length. The first collision field with the max length is modified as this suffix in the collision pattern set. It is shown in figure 5.

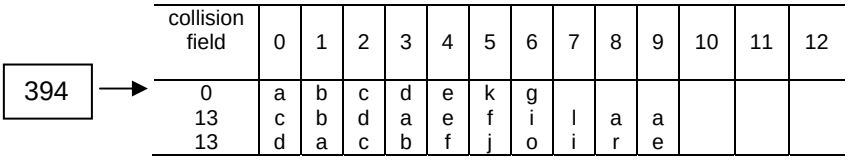

Figure 5. Modify collision field

To advance matching efficiency in the future, each pattern needs an additional field. This field is used to store byte number of this pattern. So the data structure of the pattern node is as follows:

```
struct string
{  int num;
 int collision;
   char *s;
}
```

In multi-pattern matching, the continue maxlen length bytes from the text entrance is adopted as matching window. The sum of minlen length bytes of the matched text is computed firstly, such as 394. Then we search the pattern sub-set by AVL tree. When the value 0 in the first pattern collision field is found, the byte at 0 position in each pattern is checked in turn. If corresponding byte is matched in certain pattern, precision matching can be done with this pattern. If corresponding byte is not matched in each pattern, matching is lost at this text entrance, and the window slides to the next byte.

It is possible that multiple collisions happen. If the index of another pattern (abcdfkil) is also 394, its position is added to the second of the pattern sub-set. Here, the biggest difference degree of the collision pattern sub-set is still 3, less than the number (4) of patterns. After the first collision checking, it is found that the first pattern still collides with the second pattern, because the context with 0 suffix is ‘a’. The next collision point needs to be found. This position is suffix 4. These two patterns can be distinguished by storing this suffix to the second pattern’s collision field. It is shown in figure 6.

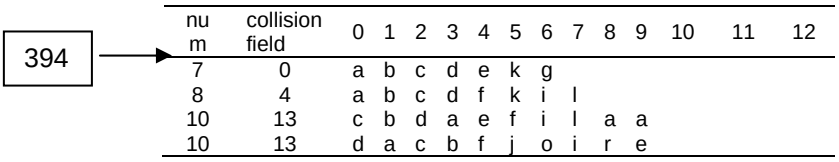

Figure 6. Multiple colliding

In mult-pattern matching, if the corresponding byte in the text is matched with the third or the fourth pattern by 0 suffix and the number of the matched pattern is 1, precision matching with this pattern happens directly. If the byte with 0 suffix in the text is also ‘a’, there are two matching patterns, then collision happens. Search for the next pattern (other than the pattern collision whose field value 0) whose collision field with value 4, namely, the fourth byte content in the text’s current byte string compares with the fourth byte content in these two patterns. If the corresponding byte in a certain pattern is the same as the byte in the text, precision matching with this pattern happens. If it is different from the corresponding byte in each pattern, matching is lost. It can also work with further collisions.

4. The Multi-Pattern Matching Method Based on Stitched Information

The first few bytes of each pattern are accumulated in the multiple patterns matching algorithm based on added information. To reduce the collision probability and then to reduce the times of pattern matching, these accumulated value are compared. When the added values collide, this method will eliminate the collision further by increasing collision field. After further analysis of the method, it is found that this method can be improved.

In applications, the pattern which needs to match may have the same head or tail of information. For example, it needs to search for the names of books in the library or the names of papers in the paper database, "based on ..." or "... methods" are the common forms. This means that the information accumulation method against the head or tail information still has higher collision frequency, although the collision odds smaller than the traditional method of non-numerical matching.

In order to avoid focusing on the head or tail of the same or similar information, it can use the method that extract some bytes of each pattern to obtain the average information and then to reduce the collision odds. For example, it can be extracted every other k bytes. In Table 2, if the length of the shortest pattern in the pattern set is 8 (minlen = 8), then the character data in odd position of each pattern are accumulated, this means that it is extracted every other byte(k=1). This extracting method reflects the even information of each pattern and the characteristics of different patterns better, so the collision odds will reduce further.

Table 2. The sorted pattern set based on the sum of even information

Pattern	The sum of the foregoing 4 odd bytes	0	**1**	2	**3**	4	**5**	6	**7**	8	9	10	11	12
cbdaefjmaa	406	c	**b**	d	**a**	e	**f**	j	**m**	a	a			
dacbfjoire	406	d	**a**	c	**b**	f	**j**	o	**i**	r	e			
abcdekggh	408	a	**b**	c	**d**	e	**k**	g	**g**	h				
ldifjglk	412	l	**d**	i	**f**	j	**g**	l	**k**					
xyzabcde	418	x	**y**	z	**a**	b	**c**	d	**e**					
slkdjgioejgr	422	s	**l**	k	**d**	j	**g**	i	**o**	e	j	g	r	
acaapzyijr	423	a	**c**	a	**a**	p	**z**	y	**i**	j	r			
klkrfhkdfksdl	426	k	**l**	k	**r**	f	**h**	k	**d**	f	k	s	d	l
jdhkjwelm	434	j	**d**	h	**k**	j	**w**	e	**l**	m				
ayzxabkn	449	a	**y**	z	**x**	a	**b**	k	**n**					

In Table 2, only two patterns with the sum of the first four odd bytes (406) are same. These two patterns can be distinguished by the collision field further.

Furthermore, it is significant to reduce the collision odds of the initial part of the calculation. If the character stitched method is used instead of using added method, it will avoid the collision in which the added characters are different but the added value is the same, even distinguish the situation that the characters are same but just the location is changed. For example, to the character string “cbdaefjmaa” in the Table 2, whether using the method of adding the first few continuous characters or the method of adding the first few odd characters, it will both be collided by other strings. It needs to take an ulterior step to solve the problem by using collision field. According to the method of spliced character, it could expand calculation numerical space, and reduce the collision odds. To the string “cbdaefjmaa” , the process of splicing from the first four odd characters (bafm) is shown in figure 7.

b	a	f	m
98	97	102	109
01100010	01100001	01100110	01101101
01100010011000010110011001101101			
1650550381			

Figure 7. Spliced information

The value obtained from the even position character splicing method can distinguish the patterns that the characters are same but their positions are changed. It can not be achieved by

the added method. To the first four odd characters (bafm) in the foregoing example, if only the position is changed into “mabf”, the result of accumulation will be still 406, but its splicing value is different. It is shown in figure 8.

f	b	m	a
102	98	109	97
01100010	01100001	01100110	01101101
01100110011000010110011001101101			
1717659245			

Figure 8. Spliced information after the position is changed

To the strings in the Table 2, the splicing value of each string is calculated out. Then, Table 3 is a sorted Table based on the splicing value.

Table 3. The sorted pattern set based on the splicing value of even information

pattern	The splicing value of the foregoing 4 odd bytes	0	**1**	2	**3**	4	**5**	6	**7**	8	9	10	11	12
dacbfjoire	1633839721	d	**a**	c	**b**	f	**j**	o	**i**	r	e			
cbdaefjmaa	1650550381	c	**b**	d	**a**	e	**f**	j	**m**	a	a			
abcdekggh	1650748263	a	**b**	c	**d**	e	**k**	g	**g**	h				
acaapzyijr	1667332713	a	**c**	a	**a**	p	**z**	y	**i**	j	r			
ldifjglk	1684432747	l	**d**	i	**f**	j	**g**	l	**k**					
jdhkjwelm	1684764524	j	**d**	h	**k**	j	**w**	e	**l**	m				
slkdjgioejgr	1818519407	s	**l**	k	**d**	j	**g**	i	**o**	e	j	g	r	
klkrfhkdfksdl	1819437156	k	**l**	k	**r**	f	**h**	k	**d**	f	k	s	d	l
xyzabcde	2036425573	x	**y**	z	**a**	b	**c**	d	**e**					
ayzxabkn	2037932654	a	**y**	z	**x**	a	**b**	k	**n**					

It can be seen from Table 3, the compared value is stitched by the first four odd characters, which contains combinatorial information of related character positions. The collision odds can be up to $1/(2^{32})$, which is far less than the collision odds 1 / (256 * minlen) based on information accumulation. Hence, the multi-pattern matching speed is advanced. In this example, the collision is not occurred by using the splicing value method. It is not necessary to reduce collisions by collision field.

In summary, the Multi-pattern matching methods based on the spliced information have lower collision odds than the Multi-pattern matching methods based on the added information. Besides, the collision odds of splicing information reduce rapidly with the increasing number of bytes. It supposes that the length of simple data type which system can handled each time is not more than 8 bytes, then the collision odds between added information method and spliced information method are compared in Table 4.

Table 4. The comparison of collision odds between added information method and spliced information method

method	added information method		spliced information method	
byte number	value space	collision odds	value space	collision odds
1 byte	2^8	$1/2^8$	2^8	$1/2^8$
2 bytes	$2*2^8$	$1/(2*2^8)$	2^{16}	$1/2^{16}$
3 bytes	$3*2^8$	$1/(3*2^8)$	2^{24}	$1/2^{24}$
4 bytes	$4*2^8$	$1/(4*2^8)$	2^{32}	$1/2^{32}$
5 bytes	$5*2^8$	$1/(5*2^8)$	2^{40}	$1/2^{40}$
6 bytes	$6*2^8$	$1/(6*2^8)$	2^{48}	$1/2^{48}$
7 bytes	$7*2^8$	$1/(7*2^8)$	2^{56}	$1/2^{56}$
8 bytes	$8*2^8$	$1/(8*2^8)$	2^{64}	$1/2^{64}$

Figure 9 indicates the comparison between two methods based on the numerical space. The difference between the two methods reaches the exponential level, so it is easier to draw figure by using Log calculation to process the numerical space of the two methods, which can evaluate the value of the exponent space.

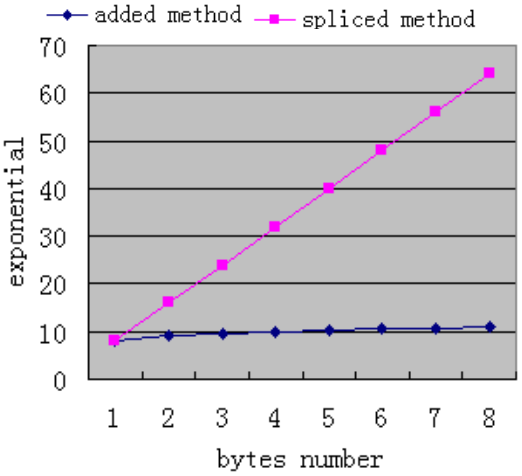

Figure 9. The comparison of value space between added information and spliced information

It can be seen from Figure 9 that the related numerical space of the spliced information method are greater than that of the added information method by increasing the inspecting bytes, which can greatly reduce collision odds and increase the successful probability of initial matching.

In summary, multi-pattern matching should follow these rules:

(1) The collision probability in the previous steps should keep as low as possible

(2) The sorting rules taken by different steps should keep as different as possible

(3) The bytes data involved in the operation should reflect the characteristics of each pattern.

5. Conclusion

Two multiple patterns matching methods based on numerical computation are advanced in this paper: one is based on the added information method, and the other is spliced information method. Multiple patterns matching methods based on numerical computation can overcome the language problems of pattern matching algorithm. To reduce collision, the information of multiple bytes in matched text is integrated. To eliminate collision further, the collision fields can be selected based on the characteristics of given pattern set. This paper summarizes the rules followed by different steps in multi-pattern matching process. It can be analyzed that the multi-pattern matching methods based on the spliced information have lower collision odds than the multi-pattern matching methods based on the added information, thereby multi-pattern matching method based on the spliced information increases the successful possibility of the initial matching.

Acknowledgment

This paper is supported by the Natural Science Foundation of Heilongjiang Province of china under Grant No. F201138, the Scientific Research Fund of Heilongjiang Provincial Education Department under Grant No. 12521395 and the Youth Science Fund of Heilongjiang University under Grant No.QL201044. We are grateful to all those who made constructive comments.

References

[1] Knuth DE, Morris H, Pratt VR. Fast Pattern Matching in Strings. *SIAM J Comp*. 1977; 6: 323-350.
[2] Boyer RS, Moore JS. A Fast String Searching Algorithm. *Communications of the ACM*. 1977; 20: 762-772.
[3] ZHANG Na, HOU Zheng-feng. *A Fast Improved BM Algorithm for Pattern Matching in Strings*. Journal of Hefei University of Technology. 2006; 29(7): 834 - 838
[4] YANG Wei-wei, LIAO Xiang. *Improved Pattern Matching Algorithm of BM.* Computer Applications. 2006; 26(2): 318 -319
[5] PANG Shan-chen, WANG Shu-dong. JIANG Chang-jun, *Improved BM-altorithm for Pattern Matching in String*. Computer Application. 2004; 24(12): 11-13
[6] Aho V, Corasick MJ. *Efficient String Matching: An Aid to Bibliographic Search.* Communications of the ACM.1975; 18: 333-340.
[7] WU Sun, Manber U. *A Fast Algorithm for Multi-Pattern Searching*. Technical Report TR. University of Arizona at Tuscon. 1994; 94-17
[8] HU Pei-hua, WAHG Yong-cheng, LIU Gong-shen. *A Multiple Pattern Matching Algorithm Based on Sequential Binary Tree*. Computer Science. 2002; 29(11): 65-68.
[9] LI Wei, GUAN Xiao-hong, TAHG Wen-rong. *A Fast Matching Algorithm Based on a Special Cross Data Link Table*. *Journal of china institute of communication*. 2004; 25(10): 38-44
[10] YANG Donghong, Xu ke, CUI Yong. *Improved Wu-Manber Multiple Patterns Matching Algorithm*. J Tsinghua Univ. 2006; 46(4): 555-558
[11] SUN Xiao-shan, WANG Qiang, GUAN Yi, WANG Xiao-long. An Improved Wu-Manber Multiple-pattern Matching Algorithm and Its Application. *Journal of Chinese Information Processing*. 2006; 20(2): 47-52
[12] HORSPOOL RN. *Practical Fast Searching in Strings*. Software-Practice and Experience. 1980; 10(6): 501-506.