IAPR Workshop on CV - Special Hardware and Industrial Applications OCT.12-14, 1988, Tokyo

THE EXTENSION OF THE AHO-CORASICK ALGORITHM TO MULTIPLE RECTANGULAR PATTERNS OF DIFFERENT SIZES AND N-DIMENSIONAL PATTERNS AND TEXT

Rui Feng ZHU Masayuki NAKAJIMA Takeshi AGUI

Imaging Science and Engineering Laboratory,
Tokyo Institute of Technology.
4259 Nagatsuta, Midori-ku, Yokohama 227, JAPAN

Abstract

The pattern matching problem is to find all occurrences of a pattern in a text, or to decide that no such pattern exists in the text. An efficient algorithm was proposed by Boyer and Moore [1]. Unlike the Knuth-Morris-Pratt algorithm [2] and the 'brute-force' algorithm, the Boyer-Moore algorithm compares the pattern with the text from the right end of the pattern. The authors have presented a method [3] to improve the avebrage performance of the Boyer-Moore algorithm. As the length of pattern gets longer, it decreases the number of characters inspected and the running time rapidly. Another efficient algorithm for matching multiple patterns was proposed by Aho and Corasick [4]. We call it AC algorithm. In these methods, both the pattern and text are one-dimensional arrays. A natural extension of the pattern matching problem is the case in which both the patterns and text are represented in computer in the form of rectangular arrays. In the present paper, we first show that multiple rectangular pattern arrays of various sizes can be efficiently recognized by extending the idea proposed in the AC algorithm, then demonstrate such method permits extension to arrays of arbitrarily many dimensions. Both the running time and preprocessing time of our algorithms are linear to the size of the text.

1. Introduction

The pattern matching problem is to find the first or all occurrences of a pattern in a text, or to decide that no such pattern exists in the text. The techniques for finding the first occurrence of the pattern can be easily modified to find all the occurrences of the pattern. we will concentrate on finding the first occurrence of the pattern. The authors have presented a method [3] to improve the average performance of the Boyer-Moore algorithm. The basic idea is to utilize two characters for a precomputed table instead of one character. Computer experiments have shown that as the length of pattern gets longer, it decreases the number of characters inspected and the running time sharply. An efficient algorithm for matching multiple patterns was proposed by Aho and Corasick [4]. We call it AC algorithm. Let $K = \{ PT_i \mid i=1,2, \ldots k \}$ be a finite set of keywords PT_i and text T an arbitrary string. The theme is to find all keywords included in text T. A natural extension of the pattern matching problem is the case in which both the patterns and text are rectangular arrays, that is $K = \{ PT_i[P_{i,1},P_{i,2}] \mid i=1,2,\ldots k \}$, where $P_{i,1}$ is the column length and $P_{i,2}$ is the row length.

2. The AC algorithm

The AC algorithm first makes a finite state pattern matching machine from the set of keywords K, then apply the text T as input to the machine. We discribe the behavior of the pattern matching machine by three functions: a goto function g, a

failure function f, and a output function p. The pattern matching machine consists of a set of states, and represent each state by a number. The machine processes the text T by reading the characters in text T, making state transitions and occasionally emitting output. The goto function g maps a pair consisting of a state and an input character into a state or the message fail. The failure function f maps a state into a state. The failure function is referred whenever the goto function reports fail. We also assign some states as output states which means that a set of keywords has been found. The details of algorithms for construction the three functions are given in [4].

3. The Algorithm

Based on the AC algorithm, we first present an efficient algorithm to find all occurrences of multiple patterns of various sizes in the text T, then show such idea permits extension to patterns of any dimensions. The general scheme of the algorithm is composed of two distinct steps, a row-matching step and a column-matching step. The purpose of the row-matching step is to determine which row of the patterns matches a terminal substring of text T. We realize this by using the AC algorithm in which each row of patterns is considered to be a keyword. We assume that $P_{1,2} \geq P_{2,2} \geq P_{3,2} \geq \ldots \geq P_{k,2}$, and make a partition according to the following condition:

$$PT_i \text{ and } PT_{i+1} \text{ in a same set iff } P_{i,2}=P_{i+1,2}. \quad (i=1,2,\ldots k-1) \qquad (1)$$

the size of such a partition is assumed to be k1($k1 \leq k$). We define an array PAR[1..k] of 1..k1 to partition patterns according to their row length. The component rows of patterns within the same partition are all of the same length. It follows that no two component rows may be proper suffixes in a same partition. Therefore, for each position in a row of the text, at most one distinct row of given length of patterns may match with text T in that location.

We identify the distinct rows of patterns and assign each a unique index. Let the distinct rows be $X_1, X_2, \ldots X_q$. Thus the subscript of pattern PT_i can be represented by the following column in $\{1,2, \ldots q\}$.

$$(p(i,1),p(i,2), \ldots p(i,P_{i,1}))^t \text{ in} \qquad (2)$$

So, we can represent the pattern PT_i in the following form:

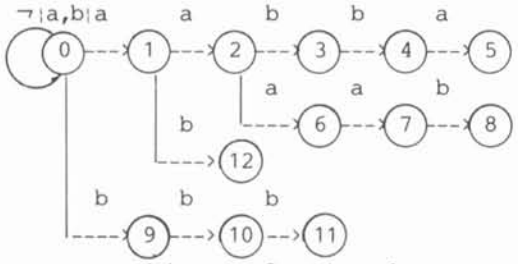
$$PT_i = X_{p(i,1)} \, X_{p(i,2)} \, \cdot \, \cdot \, \cdot \, X_{p(i,Pi,1)} \qquad (3)$$

We put such a unique index in output function p.

Example 1. k=5, k1=4

	PAR	index
PT_1:		
a a b b a=X_1	1	p(1,1)=1
a a a a b=X_2	1	p(1,2)=2
PT_2:		
a a a=X_3	2	p(2,1)=3
b b b=X_4	2	p(2,2)=4
a a a=X_3	2	p(2,3)=3
PT_3:		
a a a=X_3	2	p(3,1)=3
PT_4:		
a b=X_5	3	p(4,1)=5
a a=X_6	3	p(4,2)=6
PT_5:		
a=X_7	4	p(5,1)=7

Also we add to the output function p another value to retain the partition value of the row found. The goto function g1, failure function f1, output function p1 for example 1 are shown in Fig.1.



(a) goto function g1

i	1	2	3	4	5	6	7	8	9	10	11	12
f1	0	1	12	10	1	2	6	3	0	9	10	9

(b) failure function f1

state	(output11,output12)
1	[(7,4)]
2	[(6,3),(7,4)]
3	[(5,3)]
5	[(1,1),(7,4)]
6	[(7,4),(6,3),(3,2)]
7	[(7,4),(6,3),(3,2)]
8	[(2,1),(5,3)]
11	[(4,2)]
12	[(5,3)]

(c) output function p1

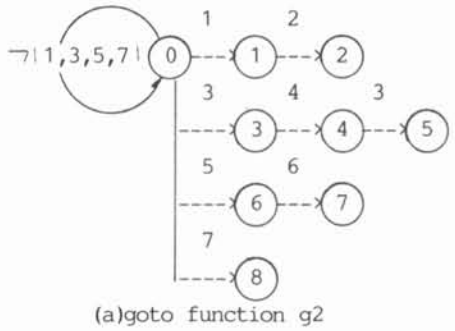
Fig.1

After finding that the i-th row of a pattern occurs in a given place in the text, we must examine whether or not the rows from the 1st row until the i-1th row of the pattern occur immediately above the i-th row. This can be done by making another pattern matching machine in the column-matching step.

We construct another pattern matching machine M2 with Goto function g2, failure function f2, and output function p2. The input keywords are:

$$KK = \{\, p(i,1)\ p(i,2)\ \ldots\ p(i,P_{i,1}) \mid i=1,2,\ldots,k \,\} \qquad (4)$$

The goto function g2, failure function f2, and output function p2 for example 1 are shown in Fig.2.



(a)goto function g2

i	f2(i)
1	0
2	0
3	0
4	0
5	3
6	0
7	0
8	0

(b)failure function f2

state	output2
2	PT_1
3	PT_3
5	PT_2,PT_3
7	PT_4
8	PT_5

(c)output function p2

Fig.2

We assume string $X_1X_2\ldots X_u$ represents state s of pattern matching machine M2 if the shortest path in the goto function of M2 from the start state to state s spells out $X_1X_2\ldots X_u$. Also We maintain a rectangular array a[1..k1,1..N2] of states, such that for each position (row,col) the fact that a[k,col]=s means just that $X_1X_2\ldots X_u$ of pattern PT_i with partition value k have been discovered to match the text in positions as shown in the following:

T[row-u+1,col-$P_{i,2}$+1]...T[row-u+1,col]		X_1
T[row-u+2,col-$P_{i,2}$+1]...T[row-u+2, col]		X_2
.	.	.
.	.	.
.	.	.
T[row,col-$P_{i,2}$+1]	... T[row,col]	X_u

If we have p2(s)≠empty that means a complete pattern PT_i has been found as a subarray of the text at (row,col). The

number of p1(1,state1) might be more than 1, but always less than k1. For it to work correctly at the start of the algorithm, we initialize a[t,col]=0 $1 \leq t \leq k1$, and $1 \leq col \leq N2$, that is each element of array a begins from start state. The Algorithm is shown below:

Algorithm.

Input. A text T[1..N1,1..N2] and pattern matching machines M1 and M2 with Goto function g1 and g2, failure function f1 and f2, output function p1 and p2.

Output. locations(row,col) at which PT_i occurs in text T.

Method.

```
begin
 state1:=0;
 for row:=1 until N1 do
 for col:=1 until N2 do
 begin
 while g1(state1,T[row,col])=fail
  do state1:=f1(state1);
 state1:=g1(state1,T[row,col]);
 if p1(1,state1)≠empty
  then for all p1(1,state1)≠empty do
   begin
    state2:=a(p1(2,state1),col);
    c:=p1(1,state1);
    while g2(state2,c)=fail do
     state2:=f2(state2);
    state2:=g2(state2,c);
    a(p1(2,state),col):=state2;
    if p2(state2)≠empty then
     begin
      print (row,col)
      print (p2(state2))
     end
   end
 end
end.
```

The running time depends on how often the rows of the patterns occur in the text. In average case it is linear to the size of text T.

4. The extension of our algorithm.

If $P_{1,2}=P_{2,2} \cdots =P_{k,2}$, and $P_{1,1}=P_{2,1}= \cdots =P_{k,1}$, we consider the set of patterns as a three dimensional array, PT[P1,P2,P3], and also text T is a three dimensional array, T[N1,N2,N3]. The theme becomes to find all occurrences of the pattern PT as embedded subarraies in the text T. The algorithm is composed of three matching steps. Both the time and space requirments of the algorithm are linear to the size of text T. Naturelly, we can also extend our algorithm to higher dimensions.

5.Conclusion

In the paper, we have demonstrated that multiple rectangular pattern arrays of various sizes and N-dimensional arrays can be efficiently recognized by using the method proposed in AC algorithm [4]. The algorithms described here have the following noteworthy properties:

(1) Both its running time and space requirement is linearly proportional to the size of text, which is clearly optimal since the text have to be read and this needs time and space equal to the size of the text.

(2) its on-line nature, that is the input is scanned only once, and after scanning the characters at any position of the input, before scanning further, it is possible to answer yes or no to whether any of the patterns match at that position.

References:

[1] R. S. Boyer, and J. S. Moore, A Fast String Matching Algorithm, Comm. ACM 20(10), pp.762-772 (1977).

[2] D. E. Knuth, J. H. Morris and V. R. Pratt, Fast Pattern Matching in Strings, SIAM J. Comput. 6(2), pp.323-350, (1977).

[3] R. Zhu. and T. Takaoka, On Improving the Average Case of the Boyer-Moore String Matching Algorithm, Journal of Information Processing, Japan, 10(3), (1987).

[4] A.V.Aho and M.J.Corasick, Efficient string matching: An aid to bibliographic search, Comm. ACM 18(6) pp.333-340 (1975).