Vol. 39 No. 2 Transactions of Information Processing Society of Japan Feb. 1998

Technical Note

Fast On-line String Searching

ZHIBIN LIU,[†] XIAOYONG DU[†] and NAOHIRO ISHII[†]

The on-line string searching algorithm searches the text while the pattern is being read, and is widely used in text editor software. In this paper, we give an implementation of the on-line BM algorithm which is much better than other on-line algorithms both in the number of character comparisons and in running time.

1. Introduction

The string searching problem is to find all occurrences of a pattern in a text. Two famous algorithms are the KMP algorithm and the BM algorithm, both are "off-line" algorithms in the sense that they do not begin to search before the last character of the pattern is read. In text editor software, the "on-line" versions of string searching algorithms are used, which can search the text while the pattern is being read. The advantages of an on-line algorithm are not only its quick response time, but also helping the user to find what he wants easily.

The on-line version of the naive string searching algorithm can be implemented easily. Barth[1)] and Takaoka[4)] both present the on-line KMP algorithm by computing the auxiliary array *next* incrementally. Recently, Takaoka[5)] points out that it is possible to implement an on-line BM algorithm by computing the auxiliary arrays from left to right. However he does not implement an efficient on-line algorithm.

In this paper, we give an implementation of the on-line BM algorithm which is much better than other on-line algorithms both in the number of the character comparisons and in the running time. We also improve the algorithm to meet the requirements of on-line searching. Our algorithm can be easily embedded in a text editor software.

2. On-line BM Algorithm

Implementing an on-line BM algorithm requires two modifications to the original one: the right-to-left comparison and the computation of the skip function arrays *d* and *dd*.

The right-to-left comparison in the BM algorithm requires to compare the last character first in the pattern. It does obviously not suit on-line searching. Although some people believe those algorithms with right-to-left comparison have better performance, that is a misunderstanding. The right-to-left comparison order will result in more comparisons than the normal left-to-right order, in fact it has no contribution to the performance of the algorithms[2),3)].

The right-to-left comparison can be easily replaced by the left-to-right one. Furthermore, this replacement does not lose efficiency.

The other modification is the computation of the skip function *d* and *dd*. In this paper, we only use the *d* as the skip function since the *dd* has little contribution when the algorithm is used to search normal texts. The on-line BM algorithm using the *dd* can be found in Ref. 5).

The BM algorithm uses the pattern length to compute the *d* before the search starts. In the on-line searching case, the algorithm does not know the pattern length until the terminating symbol is read. In this paper, we first initiate all elements of the *d* by zero before the search starts, then update the *d* as a search proceeds. When a character of the pattern is read, the element corresponding to that character in the *d* is updated by the position of that character in the pattern.

After a mismatch, the character next to the rightmost character of the current substring in the text is chosen to determine the skip. The skip distance is equal to one plus the difference between the current length of the pattern and the value of corresponding element in the *d*, i.e. *pattern_length - d[text[i0+1]]+1*.

In principle, our algorithm can be viewed as the loop by the following three steps: (1) reading a character from input, and appending it to the pattern; (2) updating the *d*, and increasing the pattern length by one; (3) searching the read part of pattern in the text in the same

† Department of Intelligence and Computer Science, Nagoya Institute of Technology

way as the off-line algorithm does since the *d* is ready and the pattern length is known.

Our on-line BM algorithm written in C language is presented in following:

```
int on_line_BM(text,n)
char text[]; /*begins from text[0]*/
int n; /*length of text*/
{
char pat[MAX_PAT_LEN]; /*begins from pat[0]*/
int d[ALPHABET_SIZE];/*skip function*/
int patlen; /*current length of pattern*/
int j0, i0, i, j, k;
/*initiates skip function*/
for(k=0;k<ALPHABET_SIZE;k++)d[k]=0;
patlen=0; j0=-1; i0=-1;/*assigns initial values*/
while(i0<n){ /*main searching loop*/
/*reads character from input*/
    if(!get_next_char(&pat[patlen]))break;
/*updates corresponding element in d*/
    d[pat[patlen]]=patlen+1;
    patlen++; /*pattern's length increases 1*/
/*j0 points to new read character in pattern*/
/*i0 points to corresponding one in text*/
    j0++; i0++;
/*following loop searches pattern, of length*/
/*patlen, in text as off-line algorithms do,*/
/*begins from i0-patlen +1, ends at n*/
    while(i0<n){
/*if two characters equal, compares the others;*/
/*otherwise, i0 skips to next substring in text*/
        if(pat[j0]==text[i0]){
/*j points to first character in pattern*/
/*i points to corresponding one in text*/
            j=0; i=i0-patlen+1;
/*if all pairs equal, current pattern matches*/
/*this substring; otherwise a mismatch occurs.*/
            while(i<i0){
             if(pat[j++]!=text[i++]) goto mismatch;
            };
/*current pattern matched*/
            goto read_next_char;
        };
mismatch: /*skips to next substring*/
        i0 += patlen + 1 - d[text[i0+1]];
    };
read_next_char: ;
};
if(i0<n)return(i0-patlen+1);/*find complete match*/
else return(-1);/*not found*/
}/* end of algorithm*/
```

3. Some practical considerations

When we implement an on-line string searching algorithm, we should also consider other "on-line" requirements: (1) the algorithm should immediately report the searching result as soon as a new character of the pattern is read; (2) the algorithm should permit the user to update the pattern by erasing the last character as the search proceeds; (3) the algorithm should be able to find the previous matching of the pattern in the text.

The first requirement means that it is necessary to call our algorithm many times to find out a match of the "whole" pattern. So our algorithm must remember the information of the current search until the next search begins. This problem is solved by defining the following as global variables: the current pattern; the current length of the pattern; the skip function array and the return value of the last call.

When the last character is erased from the pattern, the *d* must be restored to the state before that character is input. In our implementation, we use a stack to resolve the problem. When a character is input, we push the element corresponding to that character in the *d* into the stack before updating it. So, after that character is deleted from the pattern, the *d* can be easily restored by popping out the corresponding element from the stack.

Finding the previous matching of the pattern needs to search backwardly in the text. Although the pattern is the same, the *d* cannot be used in backward search because the computation of the *d* has direction. In other words, the *d* records the rightmost occurrences in the pattern for each character in the alphabet. The backward search requires the *d* recording the first (leftmost) occurrence in the pattern. In our implementation, we use two arrays to replace the *d*: *fwdd* for normal search, and *bwdd* for the backward search, respectively.

The computation of the *bwdd* is similar to that of the *d*: initiating to zero before the search starts, then updating as the (normal) search proceeds. After a mismatch in the backward search the *bwdd* is used as the skip function. When the last character in the pattern is erased, the *bwdd* can be easily restored without using a stack.

The improved algorithm has been implemented in C language. Full program can be fetched through WWW:

http://eyes.ics.nitech.ac.jp/~duyong/publication/onlineBM.c

Table 1 The percentage improvement of number of comparisons.

Pat Len	Ours /Naive	Ours /Barth	Ours /Taka86	Ours /Taka96
1	50.3	50.3	50.3	50.3
2	36.4	36.6	36.6	36.7
3	30.6	30.6	30.6	30.9
4	27.6	27.4	27.4	28.1
5	25.7	25.4	25.4	26.4
6	24.3	24.1	24.1	25.3
7	23.4	23.2	23.2	24.5
8	22.7	22.5	22.5	23.9
9	22.1	21.9	21.9	23.4
10	21.8	21.5	21.5	23.1

Table 2 The percentage improvement of running time.

Pat Len	Ours /Naive	Ours /Barth	Ours /Taka86	Ours /Taka96
1	200.0	40.0	66.7	28.6
2	81.2	20.3	22.4	21.0
3	68.1	16.5	19.4	17.7
4	58.1	14.9	17.5	16.3
5	54.8	13.9	16.3	15.4
6	53.3	13.2	15.5	14.8
7	54.5	13.6	15.9	15.1
8	50.5	12.4	14.6	14.0
9	47.1	12.0	14.0	13.7
10	44.0	11.0	12.9	12.6

4. Experiment

We test our on-line BM algorithm with other on-line algorithms (the on-line naive algorithm, Barth's on-line KMP[1], Takaoka's on-line KMP[4] and Takaoka's on-line BM algorithm[5]). The number of character comparisons and the running time are two measures to evaluate the algorithms. All algorithms read the pattern from a procedure simulating the user's input. The running time is counted from reading the first character of a pattern to finding out the first complete match in the text.

The sample text comes from a technical report written in English. The length of text is 50000 characters. 1000 patterns of each length from 1 to 10 characters are chosen at random from the same report. For the number of character comparisons, we first count the number of characters actually compared for a pattern, then compute the average value over all 1000 patterns of each length. **Table 1** shows the percentage improvement of our algorithm over the others in the number of character comparisons. **Table 2** is that of the running time.

5. Summary

The on-line string searching algorithm can help the user to find what he wants efficiently and easily. Although the naive on-line algorithm is widely used in text editor software, the research on the on-line searching algorithms has not presented an efficient algorithm until now. For example, the on-line KMP algorithms[1),4)] run slower than the naive on-line algorithm. In this paper we implement an on-line BM algorithm which runs much faster than other on-line algorithms. Furthermore our algorithm meets the requirement of on-line searching, for example, doing backward search and erasing the last character of the pattern while the pattern is being searched. Our algorithm can be easily embedded in a text editor software.

Acknowledgments We wish to acknowledge the helpful suggestions of the anonymous referees. The work is supported in part by The Hori Information Science Promotion Foundation.

References

1) Barth, G.: An alternative for the implementation of the Knuth-Morris-Pratt algorithm, *Inf. Process. Lett.*, Vol.13, No.4, 5, pp.134–137 (1981).

2) Liu, Z., Du, X. and Ishii, N.: Right-to-left or left-to-right: Which is better comparison order in string searching?, 平成 9 年度電気関係学会東海支部連合大会講演論文集, 講演番号 579, p.290 (1997).

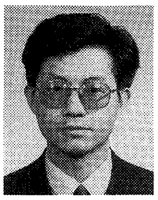
3) Sunday, D.: A very fast substring search algorithm, *Comm. ACM*, Vol.33, No.8, pp.132–142 (1990).

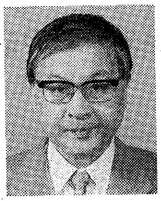
4) Takaoka, T.: An on-line pattern matching algorithm, *Inf. Process. Lett.*, Vol.22, No.6, pp.329–330 (1986).

5) Takaoka, T.: Left-to-right preprocessing computation for the Boyer-Moore string matching algorithm, *Comput. J.*, Vol.39, No.5, pp.413–416 (1996).

(Received August 25, 1997)
(Accepted November 5, 1997)

Zhibin Liu received the B.E. degree in computer science and technology from Tianjin University, Tianjin, China in 1985, and the M.E. degree in information and computer science from the People's University of China, Beijing, China in 1988. From 1990 to 1992 he was a lecturer of the Department of Computer Science and Technology at Tianjin University. From 1992 to 1995, he was a lecturer in the Department of Computer Science and Technology at Beijing Polytechnic University. He is now pursuing the Doctor of Engineering degree at the Nagoya Institute of Technology, Nagoya, Japan. His current research interests include algorithm design and analysis, databases and artificial intelligence.


Xiaoyong Du received the B.S. degree in computational mathematics from Hangzhou University, Zhejiang, China in 1983, and the M.E. degree in information and computer science from the People's University of China, Beijing, in 1988, and the Doctor of Engineering degree at the Nagoya Institute of Technology, Japan, in 1997. From 1989 to 1992, he was a lecturer in the Institute of Data and Knowledge Engineering at the People's University of China. He is now a research associate in the Department of Intelligence and Computer Science at the Nagoya Institute of Technology, Nagoya, Japan. His current research interests include databases and artificial intelligence. He is a member of the IPSJ and IEICE.


Naohiro Ishii received the B.E. and M.E., and Doctor of Engineering degrees in electrical and communication engineering from Tohoku University, Sendai, in 1963, 1965, and 1968, respectively. From 1968 to 1974 he was at the School of Medicine in Tohoku University, where he worked on information processing in the central nervous system. Since 1975 he has been with the Nagoya Institute of Technology, where he is a professor in the Department of Electrical and Computer Engineering. His current research interests include databases, software engineering, algorithm design and analysis, nonlinear analysis of neural network and artificial intelligence. He is a member of the IPSJ, IEICE, ACM, and IEEE.