Математички Билтен
21 (XXI)
1997 (121-128)
Скопје, Македонија

ISSN 0351-336X

A METHOD OF STRING SEARCH

Octavian C. Dogaru

Abstract

This paper presents a simple algorithm for searching a *word* or a *pattern* of a lenght m characters in a *text* of n characters. It is a straightforword method which begins the searching with the *word* and the *text* aligned to the left ends. The number of comparisons to determine thet the *word* is or it is not in the *text* in the worst unfavourable case, theoretically is of $m\,(n-m+1)$. This seems to be faster than the direct method presented in [9].

1. Introduction

The problem of searching of a *word* or *pattern* $p\,[0\cdots m-1]$ of m characters in a *string* $s\,[0\cdots n-1]$ of n characters, $0<m\leq n$, is well studied. The methods of searching can be classificated in many ways. For example some of them are direct methods which do not use special tools, others use special tools, as example the methods based on the precompiling the pattern p as Boyer – Moore [BM], Knuth – Morris – Pratt [KMP] algorithms and variants of them. The methods based on the idea of precompiling p are faster than the direct methods but they are more intricated. An example of direct method is that presented by N. Wirth in [9] and of which core, presented in the algorithmic language, slight modified, described in [7] is following

$i := -1;$

loop

$i := i + 1;\ j := 0;$

while $(j < m)$ and $(s(i + j) = p(j))$ do $j := j + 1$ repeat

until$(j = m)$ or $(i = n - m)$ repeat;

where $j = m$ means that 'p is in s' and $i = n - m$ means that 'p is not in s'. It is a straightforword algorithm. In this text we refer to it as the DIRECT algorithm. It is $O(n.m)$ time complexity.

Our contribution is to present a straight string search algorithm named DO and which, in average, seems to be faster than the DIRECT algorithm. Theoretically it is also $O(m.n)$time complexity as the direct method. In the worst unfavourable case the algorithm proposed does $m(n - m + 1)$ comparisons to determine the first occurence of p in s. The DO algorithm searches the *first* occurence of p in s.

2. Main result

The idea of the algorithm proposed is following. Initally the pattern p and the string s are aligned at the left ends and the process of comparison of the characters begins and continues repeatedly. If all characters of p match with the corresponding characters of s then 'p is in s', this being the first occurence of p in s and the algorithm stops. If in this process of searching of p in s there exists a character p_j, $0 \leq j \leq m-1$ of p which is a mismatch with the corresponding s_j character of s then p_j is searched in the rest of the string s that is between the characters $s_{j+1}, s_{j+2}, \ldots, s_{n-m+j}$ that is the process of searching it is not resumed with p_0 as in the DIRECT method but it always continues with that character p_j, which has produced the mismatch. The algorithm is built on this idea. If p_j is not found in the substring $s_{j+1} \cdots s_{n-m+j}$ then 'p is not in s' and the algorithm finishes. If p_j is in this substring then let s_i be the first this occurence of p_j in this substring $(p_j = s_i)$. Then one compares the corresponding left and right neighbours of p_j and s_i. If they all occur then 'p is in s' and the algorithm halts. If $p_j = s_i$ but in the process of comparison of left and right neighbours there exists k, $0 \leq k \leq m - 1$ (exceptind $s_i = p_j$) for which $p_k \neq s_{i-j+k}$ then the process of comparison of the neighbours stops and the process of searching p_j is resumed beginning with the character s_{i+1}, that is, the same p_j is searched between $s_{i+1}\, s_{i+2} \cdots s_{n-m+j}$.

Example. One wishes to see if the pattern $p = abcd$ is or is not in the string $s = abxdyyaycdxabxxcbyyabcd$.

The DO method does the comparisons following

```
abxdyyaycdxabxxcbyyabcd
abc
   a · c
      · c
         abcd
```

where p_j ='c'. In this example $n = 23$ and $m = 4$. The character p_j which produces the mismatch in 'c'. There are necessery 29 comparisons to find p at the and of s.

This algorithm will be written as a procedure DO in the algorithmic language, slight modified, presented in [7]. It is the following

```
procedure DO (p, s, m, n)
//search the word p(0:m − 1) in a string s(0:n − 1), 0 < m ≤ n. //
//The variable i controles the string s, the variable j //
//controles the word p. The Boolean variable f returns true //
//if p is in s, otherwise false. The algorithm search the first //
//occurence of p in s. //
char p(0:m − 1), s(0:n − 1); integer i, j, k; Boolean f
j:0; f:= false;
while (j < m) and (p(j) = s(j)) do j:= j + 1 repeat
if j = m then f:= true; return endif
//the character p(j) ≠ s(j) //
i: j + 1;
loop
while (i ≤ n − m + j) and (p(j) ≠ s(i)) do i:= i + 1 repeat
if i > n − m + j then exit endif;
//exists i such that p(j) = s(i), //
//one tests the neighbours of p(j) and s(i) //
k:= 0;
while (k < m) and (p(k) = s(i − j + k)) do k:= k + 1 repeat
if k = m then f:= true; return endif
//exists p(k) ≠ s(i − j + k) //
i:= i + 1//
until i > n − m + j repeat;
return
endDO
```

The exit command in a loop in this algorithmic language means that the control is transfered to first statement which follows to loop statement that contains it.

The partial correctness of the algorithm is proved using a proof table. In this we shall insert a set of assertion between the statements of the program such that beginning from the preconditions one arrives to the postconditions. The justifications are based on the applications of logical equivalence and rules of inference to the sequence of Pascal statements which are similar with that of the algorithmic language:

1) the assignement rule of inference:

$$\{P(e)\}\, v := e\, \{P(v)\}$$

2) the conditional rule of inference

$$\text{i)} \quad \frac{\begin{array}{c}\{P \wedge B\}\, s\, \{Q\} \\ P \wedge \sim B \rightarrow Q\end{array}}{\{P\} \text{ if } B \text{ then } s\, \{Q\}}$$

$$\text{ii)} \quad \frac{\begin{array}{c}\{P \wedge B\}\, s_1\, \{Q\} \\ \{P \wedge \sim B\} s_2 Q\end{array}}{\{P\} \text{ if } B \text{ then } s_1 \text{ else } s_2 \{Q\}}$$

3) the loop rule of inference

$$\frac{\{\text{inv} \wedge B\}\, s\, \{\text{inv}\}}{\{\text{inv}\} \text{ while B do s } \{\text{inv} \wedge \sim \text{B}\}}$$

where P, Q–are propositions, B–is a Boolean expression, inv–is the invariant of the loop and s_1, s_2, s–are statements.

procedure DO (p, s, m, n)
char $p(0{:}\, m-1)$, $s(0{:}\, n-1)$; integer i, j, k; Boolean f
{pre: input $= p(0 \cdots m-1) \wedge s(0 \cdots n-1) \wedge m > 0 \wedge n > 0 \vee$
$\forall j \in \{0 \cdots m-1\}: p_j \wedge \forall i \in \{0 \cdots n-1\}: s_i$ are characters $\wedge$ output $= \emptyset$}
$j{:}\, 0$; $f{:}=$ false
while $(j < m)$ and $(p(j) = s(j))$ do
{inv: $\forall h \in \{0 \cdots j-1\}: p_h = s_h \wedge 0 \leq j \leq m$}
$j{:}\, j+1$
repeat
{$(\forall h \in \{0 \cdots m-1\}: p_h = s_h \wedge j = m) \vee (\forall h \in \{0 \cdots j-1\}: p_h = s_h \wedge p_j \neq s_j \wedge 0 \leq j < m)$}
if $j = m$ then $f{:}=$ true;
{$j = m \wedge \forall h \in \{0 \cdots m-1\}: p_h = s_h \wedge f =$ true}
return
{output = true}
endif
{$\forall j \in \{0 \cdots m-1\}: p_j \neq s_j \wedge f =$ false}
{$i = j$}
$i{:}=\, j+1$
{$i > j \wedge j < m \wedge i \leq n-m+j \wedge f =$ false}
loop
while $(i \leq n-m+j)$ and $(p(j)\ \ s(i))$ do
{inv: $\forall h \in \{j+1 \cdots i-1\}: p_h \neq s_i \wedge p_j \neq s_i \wedge i \leq n-m+j \wedge j < m$}
$i{:}=\, i+1$
repeat
{$(\exists i \in \{j+1 \cdots n-m+j\}: p_j = s_i \wedge \forall i \in \{j+1 \cdots i-1\}: p_j \neq s_i) \vee (\forall i \in \{j+1 \cdots n-m+j\}: p_j \neq s_i)$}
if $i > n-m+j$ then
{$\forall i \in \{j+1 \cdots n-m+j\}: p_j \neq s_i \wedge f =$ false}
exit
endif
{$\exists i \in \{j+1 \cdots n-m+1\}: p_j = s_i \wedge \forall h \in \{j+1 \cdots i-1\}: p_j \neq s_h$}
$k{:}=\, 0$
while $(k < m)$ and $(p(k) = s(i-j+k))$ do
{inv: $\forall h \in \{0 \cdots k-1\}: p_h = s_{i-j+h} \wedge p_k = s_{i-j+k} \wedge k \leq m$}
$k{:}\, k+1$
repeat
{$(\forall k \in \{0 \cdots m-1\}: p_k = s_{i-j+k}) \vee (\exists k \in \{0 \cdots m-1\}: p_k \neq s_{i-j+k}) \wedge f =$ false}
if $k = m$ then $f{:}=$true
{$k = m \wedge \forall h \in \{0 \cdots m-1\}: p_h = s_{i-j+h} \wedge f =$ true}
return
{output = true}
endif

$\{\exists k \in \{0 \cdots m-1\}: p_k \neq s_{i-j+k} \wedge f = \text{false} \wedge i \leq n-m+j\}$
$i := i+1$
$\{(i \leq n-m+j) \vee (i > n-m+j)\}$
until $i > n-m+j$ repeat
$\{i > n-m+j \wedge f = \text{false}\}$
return
{output = false}
endDO

Finally we do a comparison between the DIRECT, DO and BM [Boyer–Moore] algorithms. The tests have been realized on texts having n: 1 000, 2 000, 4 000, 6 000 characters and words of$m$: 5, 10, 15, 25, 50, 100 characters. The *Table* contains in each column the average times of five tests for the same n and m. The p's have been entered randomly. We used a compatible IBM 386 – PC and the time is expressed in hundredth of seconds.

Table

n	1000	2000	4000	6000	Average
DIRECT	4.93	6.73	11.93	18.13	10.54
DO	3.67	5.43	9.73	12.56	7.85
BM	3.2	4.63	6.97	8.33	5.78

If we notes t_{DIRECT}, t_{DO}, t_{BM} the average times for the DIRECT, DO, BM algorithms respectively then, from the *Table*, the relations between them are

$$t_{\text{DIRECT}} = 1.34\, t_{\text{DO}}; \qquad t_{\text{DO}} = 1.36\, t_{BM}$$

The time of DO algorithm is comparable with the time of BM algorithm for m little ($m = 5, 10$ characters).

References

[1] R.S. Boyer, J.S. Moore, A fast string searching algorithm, *Comm* ACM, 20, 10 (1977), 762-772.

[2] R. Cole, R. Hariharan, Tighter Bounds on The Exact Complexity of String Matching, *Procc. 33rd Symp. on Foundation of Computer Sci.*, (1992), 600-609.

[3] R. Cole, Tight Bounds on Complexity of the Boyer – Moore string matching algorithm, SIAM J. *Computer*,23, 5 (1994), 1075-1091.

[4] R. Cole, R. Hariharan, M. Paterson, U. Zwick, Tighter Lower Bounds on the exact Complexity of String Matching, SIAM J. *Computer*, 34, 1(1995), 30-45

[5] O. Dogaru, Algorithm of Straight String Search, *Romanian Symposium on Computer Science*, ROSYCS, University of Iasi, (1993), 172-177.

[6] Z. Galil, A constatnt – Time Optimal Parallel String Matching Algorithm, J. ACM, 42, 4(1995), 908-919.

[7] E. Horowitz, S. Sahni, Foundamentals of Computer Algorithm, *Computer Science Press*, 1983, 625pp

[8] D. E. Knuth, J. H. Morris, V. R. Pratt, Fast pattern matching in string, SIAM J. *Computer*, 6, 2(1977), 323-449.

[9] N. Wirth, Algorithm and Data Structures, *Prentice – Hall*, N. J. (1986), 288 pp.

ЕДЕН МЕТОД ЗА ПРЕБАРУВАЊЕ НА НИЗА ОД ЗНАЦИ

Октавиан Ц. Догару

Резиме

Во оваа статија се воведува едноставен алгоритам за препознавање на мустра од знаци (стрингови) во даден текст. Даден е доказ на делумната точност на алгоритмот DO (со помош на инваријанти). На крајот направена е компаративна анализа на емпириските резултати од извршувањето на алгоритмот DO, спореден со алгоритмот на Wirth и со референтниот на BM. Покажано е дека DO работи најмалку за 30 % побрзо од директниот алгоритам на Wirth.

West University of Timisoara

Bd. V. Parvan, nr. 4, Timisoara, 1900,

Romania

E-mail: dogaru@info.uvt.ro