On the All Occurrences of a Word in a Text

O.C. Dogaru

West University of Timişoara
Bd.V.Pârvan,nr.4,Timişoara,1900,Romania

e-mail: dogaru@info.uvt.ro

Abstract. In this paper a simple straight string search algorithm is presented. For a string s that consists of n characters and a pattern p that consists of m characters the order of comparisons is $O(n.m)$, $0 < m \leq n$, in the worst case, but the average time complexity is good. The algorithm presented finds all occurrences of p in s. It do not use a precompiling of the pattern p.

1991 Mathematical Subject Classifications: 68P10 [Searching and Sorting]

Key words: direct, string, pattern, search

1 Introduction

The string matching problem is following. Given an array $s[0..n-1]$ of n characters and an array $p[0..m-1]$ of m characters where $0 < m \leq n$, the task is to find all occurrences of p in s. The string s is regarded as a *text* and the string p as a *word(pattern)*. Generally, s and p are item.

In [W86] it is presented a direct method to determine the first occurrence of p in s. In the same book it is presented the fact that the algorithm proposed is very inefficient, for example, if the pattern is p=$a^{m-1}b$ and the string is s=$a^{n-1}b$, then $m * n$ comparisons are necessary to determine that p is in s.

In this direct method the pattern and the text are aligned at the left ends. The searching begins with p_0 and s_0. If a mismatch appears then a new searching begins always with p_0, the first character of the pattern.

2 The algorithm

The algorithm proposed by us begins with p and s aligned at the left ends too but in the case that a mismatch occurs in the process of comparisons of p and s ($p_j \neq s_j$) then the searching continues with the character of p which produced the mismatch, that is p_j, which is searched between s_{j+1} and s_{n-m+j}. On this idea the algorithm is built. It will contain the followings.

1. One compares successively p_0 with s_i, i=0,1,...,$n-m$. If it exists no match of the p_0 with s_i, i=0,1,...,$n-m$ then 'p is not in s' and the process is terminated.

2. If s_i is the first match of p_0 then one compares successively p_1 with s_{i+1}, p_2 with s_{i+2} etc. If all p_j match with s_{i+j}, j=0,1,...,m-1 then this is the first occurrence of p in s. A new searching is resumed beginning with p_0 and s_{i+m}.

3. If in the process of searching a mismatch occurs between p_j and s_{i+j} ($p_j \neq s_{i+j}$) then p_j is searched in the rest of string s between s_{i+j+1} and s_{n-m+j}. If p_j is not in this rest then the searching is ended.

4. If in the substring $s_{i+j+1}, ..., s_{n-m+j}$ there exists a character which match with p_j, one renames this character s_i. Therefore $p_j = s_i$. In this case one compares the left and right neighbours of p_j and s_i that is $p_0, p_1, ..., p_j, ...,$
p_{m-1} with correspondings $s_{i-j}, ..., s_i, ..., s_{i-j+m-1}$. If all occur then this is an occurrence of p in s and the process of searching is resumed. If in the time of verification the neighbours of p_j and s_i a mismatch occurs then a new searching of p_j begins with the character s_{i+1}.

5. The algorithm stops if $i >= n - m + j$.

Example.

```
p=abcd  (m=4)
s=xabcdxabxxaycdxabcd    (n=19)
  a
   abcd
       a
        abc
           c
            c
             c
              c
            a?c
               c
                c
                 c
                  c
                 abcd
```

In this example there are 23 comparisons to find two occurrences of p in s.

The complete algorithm, presented as a procedure named DO3(written in a Pascal-like language described in [HS83]), is the following.

```
 procedure DO3(s,p,n,m)
//find all occurrences of the word p(0:m-1)//
//in the string s(0:n-1) if this exists. If yes//
        //then procedure writes 'p is in s' else it//
// write 'p is not in s'. 0<m<=n//
     char p(0:m-1),s(0:n-1); integer i,j,m,n,k; boolean f;
     i:=0; f:=false;
     loop
       j:=0;
       while (j<m) and (p(j)=s(i)) do i:=i+1;j:=j+1 repeat;
       if (j=m) then write('p is in s');f:=true;cycle endif
         // the character p(j) is a mismatch:p(j)<>s(j) //
       1:i:=i+1;
       while (i<=n-m+j)and(p(j)<>s(i)) do i:=i+1 repeat
```

```
        if i>n-m+j and not f then exit endif;
          // it exists i thus p(j)=s(i),one verifies the //
          //left and right neighbours of p(j) and s(i)//
        k:=0;
        while(k<=m-1) and (p(k)=s(i-j+k) do k:=k+1 repeat;
        if k=m then write('p is in s'); f:=true; i:=i-j+m
               else  goto 1  endif
      until i>=n-m+j repeat;
      if not f then write('p is not in s') endif
    endDO3;
```

3 Number of comparisons

The maximum number of comparisons to determine that 'p is or it is not in s', theoretically, it is obtained when, after $p_k = s_k, k = 0, 1, ..., j-1$ match, it appears $p_j \neq s_j$, but $p_j = s_i, i = j+1, ..., n-m+j$ and all the left neighbours of p_j match with the corresponding neighbours of s_i and the right neighbours of p_j, that is, $p_{j+1}, p_{j+2}, ..., p_{m-2}$ match with the right corresponding neighbours of s_i excepting p_{m-1}. For $i = n-m+j, p_{m-1}$ may or it may not match with his corresponding in s. Therefore for:

$i = j+1$, $p_0 = s_1, ..., p_j = s_i, ..., p_{m-2} = s_{m-1}; p_{m-1} \neq s_m$ there are m comparisons;
$i = j+2$, $p_0 = s_2, ..., p_j = s_i, ..., p_{m-2} = s_m; p_{m-1} \neq s_{m+1}$ there are m comparisons;
....
$i = n-m+j$, $p_0 = s_{n-m+j}, ..., p_j = s_i, ..., p_{m-2} = s_{n-2}$ and $p_{m-1} = s_{n-1}$ or $p_{m-1} \neq s_{n-1}$, there are m comparisons. Therefore in all it exists $j+1$ comparisons p_k with $s_k, k = 0, 1, ..., j$; between $j+1$ and $n-m+j$ there exists $(n-m+j)$-$(j+1)+1 = n-m$ cases for which p_j may match with $s_i, i = j+1, j+2, ..., n-m+j$ and the neighbours of p_j, that is $p_0, p_1, ..., p_{m-2}$ match with the corresponding neighbours of s_i, but $p_{m-1} \neq s_{m+k}, k = -1, 0, 1, ..., n-m-1$. Possibly, $p_{m-1} = s_{n-1}$. Every case gives m comparisons. Hence the maximum number of comparisons is

$$N_{max} = j + 1 + (n-m) * m \leq m - 1 + 1 + (n-m)m = m(n-m+1).$$

The complexity of the algorithm DO3 is $\mathcal{O}(n.m)$ too.

But in the most unfavourable cases the algorithm DO3 reduces the maximum number of comparisons from $m * n$ as in algorithm presented by N.Wirth in [W86] to $m(n-m+1)$.

For the example p=$a^{m-1}b$ and s=$a^{n-1}b$ presented in Section 1, the algorithm DO3 carries out $n+m-1$ comparisons.

4 Profiling

The variant of this algorithm(OD) written to determine the first occurrence of p in s [D98] has been compared with a direct method(DIR) presented in [W86] and the Boyer-Moore algorithm(BM) [BM77].The tests have been realized for different values of $p(m$=5, 10, 20, 50, 100) and $s(n$=1000, 2000, 3000, 4000, 5000). The p and s have been generated randomly. One generated sequences of m and n decimal integer random numbers between 32-127 and one has tacken the ASCII corresponding characters for p respectively for s. For the same m and n the three methods have been executed 10 times. The average time for an m and five values for n(=1000, 2000, 3000, 4000, 5000) are written down in the following table

m=	5	10	20	50	100	Average
OD	0.52	0.20	0.56	0.24	0.30	0.364
DIR	0.46	0.58	0.24	0.34	0.58	0.432
BM	0.12	0.22	0.44	0.42	0.22	0.284

Between the average times of three methods there are the relations

$$t_{OD} = 1.28t_{BM}; \quad t_{DIR} = 1.18t_{OD}.$$

But if the three methods are executed 100 times then the values are the following

m=	5	10	20	50	100	Average
OD	0.362	0.374	0.396	0.376	0.368	0.372
DIR	0.408	0.398	0.408	0.414	0.396	0.404
BM	0.364	0.350	0.308	0.300	0.312	0.326

In this case the relations are

$$t_{OD} = 1.14t_{BM}; \quad t_{DIR} = 1.08t_{OD}.$$

5 Correctness of the algorithm

Theorem. *The algorithm DO3 works correctly.*
Proof. To proof the correctness of the algorithm we use a proof table [TBCG92]

procedure DO3(s,p,n,m)
char p(0:m-1),s(0:n-1); integer i,j,m,n,k; boolean f;
{pre:input=$(p_0, p_1, ..., p_{m-1})\wedge(s_0, s_1, ..., s_{n-1})\wedge$
$n \geq m > 0\wedge\forall$i$\in${0,1,...,n-1}:$s_i$ are characters$\wedge$
$\forall$j$\in${0,1,...,m-1}:p_j are characters}
f:=false; i:=0;
loop
j:=0;

while (j<m) and (p(j)=s(i)) do
{inv:$\forall$ h$\in$\{0,1,...,j-1\}:$p_h = s_h \wedge 0\leq$j,i$\leq$m}
i:=i+1;j:=j+1 repeat;
{$\forall$ h$\in$\{0,1,...,j-1\}:$p_h = s_h \wedge$(j=m$\vee p_j \neq s_i$}
if (j=m) then write('p is in s');f:=true;
{output f=true}
cycle endif
// the character p(j) is a mismatch:p(j)$\neq$ s(j) //
{f=false$\wedge$j<m$\wedge p_j \neq s_i$}
1:i:=i+1;
{0<i$\leq$n-m+j$\wedge p_j \neq s_i$)$\vee$i>n-m+j)}
while (i<=n-m+j)and(p(j)$\neq$s(i)) do
{inv:$p_j \neq s_{i-1} \wedge$ i$\leq$ n-m+j}
i:=i+1 repeat;
{($p_j \neq s_{i-1} \wedge \neg$(i<=n-m+j$\wedge$ $p_j \neq s_i$)$\equiv$
{($p_j \neq s_{i-1}$ $\wedge$ i>n-m+j)$\vee$($s_i = p_j \wedge$ i<=n-m+j)}
if i>n-m+j and not f then
{$p_j \neq s_i$}
exit endif;
{$p_j = s_i \wedge$ i<=n-m+j}
// it exists i thus p(j)=s(i) //
//one verifies the left and right neighbours of p(j) and s(i)//
k:=0;
while(k<=m-1) and (p(k)=s(i-j+k) do
{inv:$\forall$h$\in$\{0,1,...,k-1\}:$p_h = s_{i-j+h} \wedge$ 0$\leq k \leq$ m}
k:=k+1 repeat;
{($\forall$h$\in$\{0,1,...,k-1\}:$p_h = s_{i-j+h}$)$\wedge$ $\neg$(k$\leq$ m-1$\wedge$ $p_k = s_{i-j+k}$)}
$\equiv$($\forall$ k$\in$\{0,1,...,m-1\}:$p_k = s_{i-j+k} \wedge$ k=m)$\vee$
($\forall$h$\in$\{0,1,...,k-1\}:$p_h = s_{i-j+h} \wedge$ $p_k \neq s_{i-j+k}$})
if k=m then write('p is in s'); f:=true; i:=i-j+m
{$\forall$k$\in$\{0,1,...,m-1\}:$p_k = s_{i-j+k}$}
else
{$\exists$k$\in$\{0,1,...,m-1\}:$p_k \neq s_{i-j+k}$}
goto 1
endif
until i>=n-m+j repeat;
{f=false $\vee$ f=true $\wedge$ 0<=j<=m$\wedge$ i>n-m+j}
if not f then write('p is not in s') endif;
{post:output=$\emptyset$}
endDO3;

The justifications are based on the application of logical equivalences and the rules of inference to the sequence of Pascal statements. These are:

i)*the assignment rule of inference*
{P(e)} v:=e {P(v)}

ii)*the conditional rules of inference*
a){P $\wedge$ B} s {Q}   b){P $\wedge$ B } s1 {Q}

$$\frac{P \wedge \neg B \Rightarrow Q}{\{P\} \text{ if B then s } \{Q\}} \qquad \frac{P \wedge \neg B\} \text{ s2 } \{Q\}}{\{P\} \text{ if B then s1 else s2 } \{Q\}}$$

iii)*the loop rules of inference*

$$\text{a)}\frac{\{inv \wedge B\} \text{ s } \{inv\}}{\{inv\} \text{ while B do s } \{inv \wedge \neg B\}} \qquad \text{b)}\frac{\{inv \wedge B\} \text{ s } \{inv\}}{\{inv\} \text{ repeat s until B } \{inv \wedge B\}}$$

where P,Q denote propositions, B-Boolean expression, inv-the invariant of the loop and s, s1, s2 are statements.

Conclusions

1)Algorithm OD is faster than algorithm DIR in average time;

2)There are pairs of p and s where algorithms OD or DIR are faster than algorithm BM;

3)At limit, the average times of the three methods tend to approach;

4)Possibly, for other p and s, the relations between the average times of the three methods can be slight different.

References

[BM77] R.S. Boyer, J.S. Moore, *A fast string searching algorithm*, Comm. ACM, **20**, 10(1977), pp.762-772

[CH92] R.Cole, R.Hariharan, *Tighter bounds on the exact complexity of string matching*, Proc. 33rd IEEE Symp. on Foundations of Computer Science,(1992), pp.600-609

[C94] R.Cole, *Tight bounds on the complexity of the Boyer-Moore string matching algorithm*, SIAM J.Comput, **23**, 5(1994), pp.1075-1091

[CHPZ95] R.Cole, R.Hariharan, M.Paterson, U.Zwick, *Tighter lower bounds on the exact complexity of string matching*, SIAM J. Comput., **24**, 1(1995), pp.30-45

[CH97] R.Cole, R.Hariharan, *Tighter upper bounds on the exact complexity of string matching*, SIAM J.Comput., **26**, 3(1997), pp.803-856

[CCGJLPR94] M.Crochemore, A.Czumaj, L.Gasiniec, S.Jarominek, T.Lecroq, W.Plandowski, W.Ritter, *Speeding up two string-matching algorithms*, Algoritmica, 5(1994), pp.247-267

[D93] O.Dogaru, *Algorithm of straight string search*, Proceedings of the 9th ROmanian SYmposium on Computer Science (ROSYCS), University of Iasi,(1993), pp.172-177

[D98] O.Dogaru, *On the first occurence of a pattern in a text*, Proceedings of MOSIS'98(Modelling and Simulation of Systems), International Conference, Volume 2, pp.45-50, May 5-7, 1998, Sv.Hostyn-Bistrice pod Hostynem, Czech Republic

[GG93] Z.Galil, R.Giancarlo, *On the exact complexity of string matching:Upper bounds*, SIAM J. Comput., **3**(1993), pp.407-437

[HS83] E.Horowitz, S.Sahni, *Fundamentals of Computer Algorithm*, Computer Science Press(1983)

[KMP77] D.E.Knuth, J.H.Morris, V.R.Pratt,*Fast pattern matching in string*, SIAM J.Comput. **6**, 2(1977), pp.323-349

[TBCG92] A.B.Tucker, W.J.Bradley, R.D.Cupper, D.K.Garnick, *Fundamentals of Computing* I, McGraw-Hill,Inc, (1992)

[W86] N.Wirth, Algorithm and Data Structures, Prentice Hall, N.J.(1986)