On the All Occurrences of a Word in a Text

O.C. Dogaru
West University of Timișoara
Bd.V. Pârvan, nr. 4, Timișoara, 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 \(O(n \cdot m)\), \(0 < m \leq n\), 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 \(p=a^{n-1}b\) and the string is \(s=a^{n-1}b\), then \(m \cdot n\) 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 \(s_i\), \(i=0,1,\ldots,n-m\). If it exists no match of the \(p_0\) with \(s_i\), \(i=0,1,\ldots,n-m\) 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 \(s_{i+j}\), \(j=0,1,\ldots,m-1\) 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 \( i \geq n - m + j \).

Example.

\[
\begin{align*}
p &= abcd \quad (m=4) \\
s &= abcdabxaycdabcd \quad (n=19)
\end{align*}
\]

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.

```pascal
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

\[
N_{max} = j + 1 + (n - m) \ast m \leq m - 1 + 1 + (n - m)m = m(n - m + 1).
\]

The complexity of the algorithm DO3 is \( O(n.m) \) too.

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

For the example \( p=a^{m-1}b \) and \( s=a^n b \) 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 taken 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

<table>
<thead>
<tr>
<th>(m)</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>OD</td>
<td>0.52</td>
<td>0.20</td>
<td>0.56</td>
<td>0.24</td>
<td>0.30</td>
<td>0.364</td>
</tr>
<tr>
<td>DIR</td>
<td>0.46</td>
<td>0.58</td>
<td>0.24</td>
<td>0.34</td>
<td>0.58</td>
<td>0.432</td>
</tr>
<tr>
<td>BM</td>
<td>0.12</td>
<td>0.22</td>
<td>0.44</td>
<td>0.42</td>
<td>0.22</td>
<td>0.284</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>(m)</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>OD</td>
<td>0.362</td>
<td>0.374</td>
<td>0.396</td>
<td>0.376</td>
<td>0.368</td>
<td>0.372</td>
</tr>
<tr>
<td>DIR</td>
<td>0.408</td>
<td>0.398</td>
<td>0.408</td>
<td>0.414</td>
<td>0.396</td>
<td>0.404</td>
</tr>
<tr>
<td>BM</td>
<td>0.364</td>
<td>0.350</td>
<td>0.308</td>
<td>0.300</td>
<td>0.312</td>
<td>0.326</td>
</tr>
</tbody>
</table>

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;
{\text{pre:input}=(p_0,p_1,\ldots,p_{m-1}) \land (s_0,s_1,\ldots,s_{n-1}) \land n \geq m > 0 \land \forall i \in \{0,1,\ldots,n-1\} : s_i \text{ are characters} \land \forall j \in \{0,1,\ldots,m-1\} : p_j \text{ are characters}}
\text{f:=false; i:=0;}
\text{loop}
\quad j:=0;
while \((j<m)\) and \((p(j)=s(i))\) do
\[
\begin{align*}
\text{inv: } & 
\forall h \in \{0, 1, \ldots, j-1\}: p_h = s_{i} \land 0 \leq j, i \leq m \\
& i := i + 1; j := j + 1 \text{ repeat; }
\end{align*}
\]
\{ \forall h \in \{0, 1, \ldots, j-1\}: p_h = s_h \land (j=m \lor 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 \land j < m \land p_j \neq s_i \}
1: i := i + 1;
\{ 0 < i \leq n-m+j \land p_j \neq s_i \land i > n-m+j \}\}
while \((i < n-m+j)\) and \((p(j) \neq s(i))\) do
\{ \forall p_j \neq s_{i-1} \land i \leq n-m+j \}
i := i + 1 repeat;
\{ \forall k \in \{0, 1, \ldots, m-1\}: p_k = s_{i-j+k} \land 0 \leq k \leq m \}
k := k + 1 repeat;
\{ \forall p_k = s_{i-j+k} \land k = m \land p_k \neq s_{i-j+k} \}\}
(\forall k \in \{0, 1, \ldots, m-1\}: p_k = s_{i-j+k} \land k = m) \lor
(\forall p_k = s_{i-j+k} \land p_k \neq s_{i-j+k} \}\}
if k = m then write(’p is in s’); f := true; i := i - j + m
\{ p_j \neq s_i \}
exit endif;
\{ p_j = s_i \land 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
\{ \forall p_k = s_{i-j+k} \land 0 \leq k \leq m \}
k := k + 1 repeat;
\{ \forall p_k = s_{i-j+k} \land k = m \land p_k \neq s_{i-j+k} \}\}
(\forall k \in \{0, 1, \ldots, m-1\}: p_k = s_{i-j+k} \land k = m) \lor
(\forall p_k = s_{i-j+k} \land p_k \neq s_{i-j+k} \}\}
if k = m then write(’p is in s’); f := true; i := i - j + m
\{ p_j \neq s_i \}
goto 1
endif
until \(i \geq n-m+j\) repeat;
\{ f = false \lor f = true \land 0 \leq j \leq n-m+j \}
if not f then write(’p is not in s’) endif;
\{ post: output = \} \}
end DO3;

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) \} \quad v := e \{ P(v) \}

ii) the conditional rules of inference
a) \{ P \land B \} \quad s \quad \{ Q \}

b) \{ P \land B \} \quad s_1 \quad \{ Q \}
\[ P \land \neg B \Rightarrow Q \]
\[ P \land \neg B \Rightarrow s_2 \{ Q \} \]
\[ \{ P \} \text{ if } B \text{ then } s \{ Q \} \]
\[ \{ P \} \text{ if } B \text{ then } s_1 \text{ else } s_2 \{ Q \} \]

iii) \text{the loop rules of inference}

\[ \{ \text{inv} \land B \} \Rightarrow \{ \text{inv} \} \]
\[ \{ \text{inv} \} \text{ while } B \text{ do } \{ \text{inv} \land B \} \]

\[ \{ \text{inv} \} \text{ repeat } s \text{ until } B \{ \text{inv} \land B \} \]

where \( P, Q \) denote propositions, \( B \)-Boolean expression, \( \text{inv} \)-the invariant of the loop and \( s, s_1, s_2 \) 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


[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

56


