Algorithms Seminar 1999–2000,
F. Chyzak (ed.), INRIA, (2000), pp. 121–124.

Available online at the URL
`http://algo.inria.fr/seminars/`.

Factor Oracle, Suffix Oracle

Matthieu Raffinot
Institut Gaspard-Monge, Université de Marne-la-Vallée

October 4, 1999

Summary by Alain Denise and Matthieu Raffinot

Abstract

The aim of this work is to design efficient algorithms for string matching. For this purpose, we introduce a new kind of automaton: the *factor oracle*, associated with the string p to be recognized in a text. This leads to simple algorithms which are as efficient in time as already known ones, while using less memory. This is a joint work with Cyril Allauzen and Maxime Crochemore.

1. Introduction

The efficiency of string matching algorithms depends on the underlying automaton which represents the string p to be found in the text. Ideally, this automaton A should satisfy the following properties:

1. A is acyclic;
2. A recognizes at least the factors of p;
3. A has the fewer states as possible;
4. A has a linear number of transitions according to m, the length of p. (Such an automaton has at least $m+1$ states.)

The suffix or factor automaton [3, 5] satisfies 1., 2., and 4. but not 3. whereas the subsequence automaton [2] satisfies 1., 2., and 3. but not 4. We present in Section 2 an intermediate structure called *factor oracle*: an automaton with $m+1$ states that satisfies all the above requirements. Section 3 is devoted to the study of a string matching algorithm based on the factor oracle.

2. Construction of the Factor Oracle

The *factor oracle* of a word $p = p_1p_2\dots p_m$, denoted $\mathrm{Oracle}(p)$, is the automaton built by the algorithm *Build_Oracle* (Figure 1). All the states of the automaton are final. Figure 2 gives the factor oracle of the word $p = abbbaab$. On this example, the reader will notice that the word aba is recognized whereas it is not a factor of p.

Here are some notations which are used in the following. The set of all prefixes (resp. suffixes) of p is denoted by $\mathrm{Pref}(p)$ (resp. $\mathrm{Suff}(p)$). The word $\mathrm{pref}_p(i)$ is the prefix of length i of p for $0 \le i \le m$. For any $u \in \mathrm{Fact}(p)$, we define

$$\mathrm{poccur}(u,p) = \min\{\, |z| \mid z = wu \text{ and } p = wuv \,\},$$

the ending position of the first occurrence of u in p. For any $u \in \mathrm{Fact}(p)$, we define the set

$$\mathrm{endpos}_p(u) = \{\, i \mid p = wup_{i+1}\dots p_m \,\}.$$

Build_Oracle($p = p_1 p_2 \dots p_m$)
    For i from 0 to m
        Create a new state i
    For i from 0 to $m - 1$
        Build a new transition from i to $i + 1$ by p_{i+1}
    For i from 0 to $m - 1$
        Let u be a minimal length word in state i
        For all $\sigma \in \Sigma, \sigma \neq p_{i+1}$
        If $u\sigma \in \mathrm{Fact}(p_{i-|u|+1} \dots p_m)$
            then build a new transition from i to $i + \mathrm{poccur}(u\sigma, p_{i-|u|+1} \dots p_m)$ by σ

FIGURE 1. High-level construction of the Oracle.

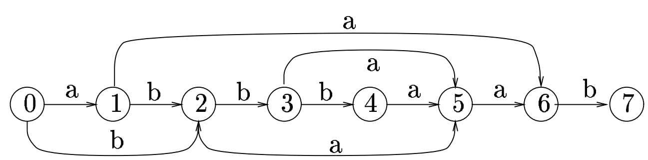


FIGURE 2. Factor oracle of *abbbaab*.

Given two factors u and v of p, we write $u \sim_p v$ if $\mathrm{endpos}_p(u) = \mathrm{endpos}_p(v)$.

The authors prove in [1] the following lemmas.

Lemma 1. *Given a state i of* Oracle(p)*, let $u \in \Sigma^*$ be a minimal length word among the words recognized in i. Then $u \in$* Fact(p) *and $i =$* poccur(u, p).

Corollary 1. *For any state i of* Oracle(p)*, there exists an unique minimal length word among the words recognized in state i.*

We denote min(i) the minimal length word of state i.

Corollary 2. *Let i and j be two states of* Oracle(p) *such that $j < i$. Then* min(i) *cannot be a suffix of* min(j).

Lemma 2. *Let i be a state of* Oracle(p)*. Then* min(i) *is a suffix of any word $c \in \Sigma^*$ which is the label of a path leading from state 0 to state i.*

Lemma 3. *Any word $w \in$* Fact(p) *is recognized by* Oracle(p) *in a state $j \leq$* poccur(w, p).

Corollary 3. *Let $w \in$* Fact(p)*. Every word $v \in$* Suff(w) *is recognized by* Oracle(p) *in a state $j \leq$* poccur(w).

Lemma 4. *Let i be a state of* Oracle(p)*. Any path ending by* min(i) *leads to a state $j \geq i$.*

Lemma 5. *Let $w \in \Sigma^*$ be a word recognized by* Oracle(p) *in i. Any suffix of w is recognized in a state $j \leq i$.*

Lemma 6. *The number $T_{Or}(p)$ of transitions in* Oracle($p = p_1 p_2 \dots p_m$) *satisfies $m \leq T_{Or}(p) \leq 2m - 1$.*

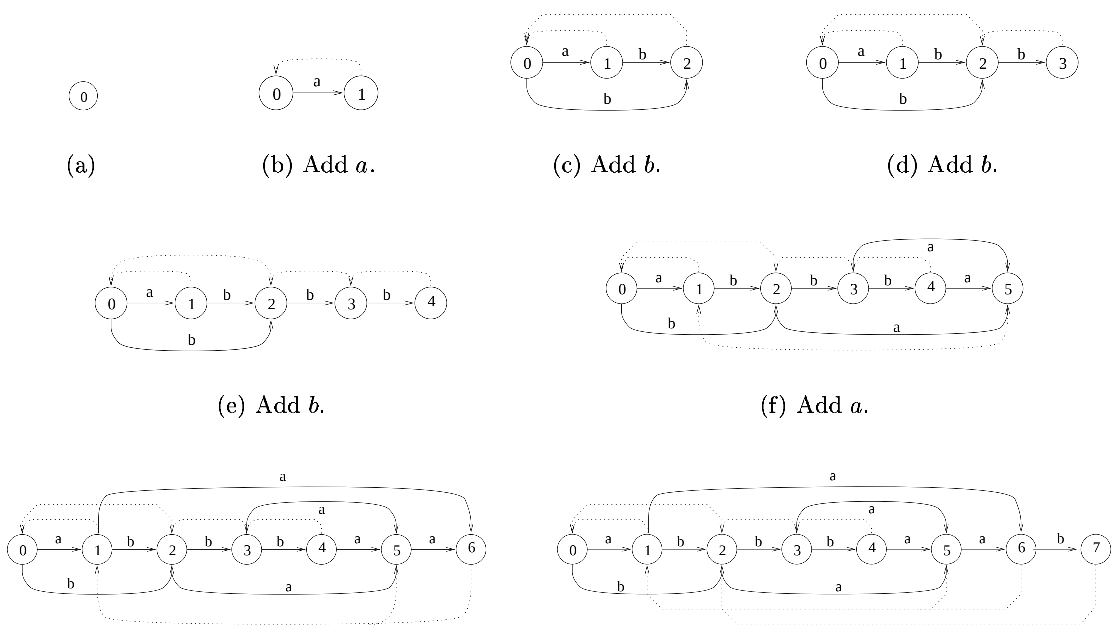
The high-level construction of the factor oracle is equivalent to the on-line algorithm given in Figure 3. An example of this construction is shown in Figure 4.

Exemple. The on-line construction of Oracle(*abbbaab*) is given Figure 4.

Fonction add_letter(Oracle($p = p_1 p_2 \ldots p_m$), σ)
    Create a new state $m+1$
    Create a new transition from m to $m+1$ labeled by σ
    $k \leftarrow S_p(m)$
    While $k > -1$ and there is no transition from k by σ **Do**
        Create a new transition from k to $m+1$ by σ
        $k \leftarrow S_p(k)$
    End While
    If $(k = -1)$ **Then** $s \leftarrow 0$
    Else $s \leftarrow$ where leads the transition from k by σ.
    $S_{p\sigma}(m+1) \leftarrow s$
    Return Oracle($p = p_1 p_2 \ldots p_m \sigma$)

Oracle-on-line($p = p_1 p_2 \ldots p_m$)
    Create Oracle(ϵ) with:
        one single state 0
        $S_\epsilon(0) \leftarrow -1$
    For $i \leftarrow 1$ à m **Do**
      Oracle($p = p_1 p_2 \ldots p_i$) $\leftarrow$ `add_letter`(Oracle($p = p_1 p_2 \ldots p_{i-1}$), p_i)
    End For

FIGURE 3. On-line construction of Oracle($p = p_1 p_2 \ldots p_m$).



(a) (b) Add a. (c) Add b. (d) Add b.

(e) Add b. (f) Add a.

(g) Add a. (h) Add b.

FIGURE 4. On-line construction of Oracle($abbaba$).

3. String Matching

The authors replace the suffix automaton with a factor oracle in the BDM (for *backward dawg matching*) [4, 6], obtaining the BOM (for *backward oracle matching*) algorithm.

The BOM algorithm consists in shifting a window of size m on the text. For each new position of this window, the factor oracle of the mirror image of p is used to search the suffix of the window from right to left. The basic idea is that if this backward search fails on a letter σ after the reading of a word u then σu is not a factor of p and the beginning of the window can be shifted just after σ. The worst-case complexity of BOM is $O(nm)$.

The average complexity of the original BDM is in $O\big(n \log_{|\Sigma|}(m)/m\big)$ under a uniform Bernoulli model. In view of the experimental results (see [1]), the authors claim that their new BOM algorithm is also optimal on average:

Conjecture 1. *Under a model of independence and equiprobability of letters, the BOM algorithm has an average complexity of* $O\big(n \log_{|\Sigma|}(m)/m\big)$.

The authors show in [1] how to obtain a linear (in n) worst case algorithm from the BOM.

Bibliography

[1] Allauzen (Cyril), Crochemore (Maxime), and Raffinot (Mathieu). – *Oracle des facteurs, oracle des suffixes.* – Technical Report n° 99-08, Institut Gaspard-Monge, Université de Marne-la-Vallée, 1999. Available from `http://www-igm.univ-mlv.fr/~raffinot/ftp/IGM99-08.ps.gz`.

[2] Baeza-Yates (Ricardo A.). – Searching subsequences. *Theoretical Computer Science*, vol. 78, n° 2 (Part A), 1991, pp. 363–376.

[3] Blumer (A.), Blumer (J.), Haussler (D.), Ehrenfeucht (A.), Chen (M. T.), and Seiferas (J.). – The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, vol. 40, n° 1, 1985, pp. 31–55. – Special issue: Eleventh international colloquium on automata, languages and programming (Antwerp, 1984).

[4] Crochemore (M.), Lecroq (T.), Czumaj (A.), Gasieniec (L.), Jarominek (S.), Plandowski (W.), and Rytter (W.). – Speeding up two string-matching algorithms. *Algorithmica*, vol. 12, n° 4-5, 1994, pp. 247–267.

[5] Crochemore (Maxime). – Transducers and repetitions. *Theoretical Computer Science*, vol. 45, n° 1, 1986, pp. 63–86.

[6] Crochemore (Maxime) and Rytter (Wojciech). – *Text algorithms.* – The Clarendon Press Oxford University Press, New York, 1994, xiv+412p.