Technical Reports on Mathematical and Computing Sciences: TR-C179
title: Factor-Oracle を索引に用いた全文検索アルゴリズム
author: 加藤 亮一 [1]
affiliation:
1. Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
Meguro-ku Ookayama, Tokyo 152-8552, Japan
(Ryoichi.Kato@is.titech.ac.jp)

Abstract:

Factor Oracle[ACR01] を索引に用いたパターンの長さに対して線形時間の全文検索アルゴリズムを提案する.

Factor Oracle は任意の有限アルファベット文字列から生成される効率的なオートマトンで少なくとも, 元の文字列の部分列 (Factor) を全て受理する. 文字列 p から生成された Factor Oracle, *Oracle*(p) がパターン w を受理しなかった場合 w は確実に p 中に部分列として存在しない. しかしながら *Oracle*(p) は p の部分列ではない文字列を受理する可能性がある.

そのため Factor Oracle をそのまま索引構造に使うことはできなかった. 本論文では Factor Oracle に若干の修正を加えることによって, ある文字列 w が *Oracle*(p) に受理されたとき, w が p の部分列であるか否かを $|w|$ に比例した時間で判別するアルゴリズムを提案する.

また, このアルゴリズムでは w の p 中でのすべての出現箇所を効率的に特定することも可能である.

1 始めに

文字列検索においては, 検索対象の文字列 p とパターン w のどちらに注目して効率化を図るかによって検索アルゴリズムが大きく二つに大別される. w に注目した場合, Boyer-Moore 法, Backword-Dawg-Match 法のように w を前処理して, 最適な場合でおおよそ $\mathcal{O}(|p|/|w|)$ 時間でパターンの出現を判別できる. 一方 p が始めに与えられる場合 p を前処理していわゆる索引を生成することによって, おおよそ $\mathcal{O}(|w|)$ 時間でパターンの出現を発見できる. 後者の手法で索引として用いられるデータ構造としては Suffix Array[MM90, Sad98], Suffix Tree[Apo85, Ukk95], compact-Dawg[BBE$^+$85, IHS$^+$01, CV97] などがあるが, 本論文では Factor Oracle[ACR01] を索引に用いて後者の文字列検索を行うことを考える.

Factor Oracle は任意の有限アルファベット文字列から生成される非常に単純で空間効率的な非循環オートマトンであり, 線型時間で構成可能である. 同様に線型時間で構成可能な類似のデータ構造: Suffix Tree, compact-DAWG などと比較して, Factor Oracle は構成アルゴリズムが非常に単純で実装が用意である. また構成に実用上必要なメモリもこれらのデータ構造と比較して少ない.

しかしながら, Suffix Tree や compact-DAWG が, 正確に元となる文字列の部分列のみを受理するのに対して, Factor Oracle では元の文字列の部分列 以外 の文字列も受理する可能性がある. ただし, 文字列 p から生成された Factor Oracle, *Oracle*(p) がパターン w を受理しなかった場合

は w は確実に p 中に部分列として存在しない.

元の文字列の部分列以外の文字列をも受理してしまうという性質のために Factor Oracle の場合は Suffix Tree, compact-DAWG のようにそのまま直接索引として用いることは出来ない. 本論文では Factor Oracle のデータ構造に若干の修正を加えることで, あるパターン w が検索対象の文字列 p からなる Factor Oracle, $\mathit{Oracle}(p)$ に受理されたとき w が p の部分列であるか否かを判別するアルゴリズムを提案する. このアルゴリズムの時間計算量は $\mathit{Oracle}(p)$ 上の各状態 $i \in \mathit{Oracle}(p)$ に対して定まる非負整数 $|\mathit{SP}^{-1}(i)|$ を用いて $\mathcal{O}(|w| \times |\mathit{SP}^{-1}(\overline{w})|)$ と表わせる. ただし $\overline{w} \in \mathit{Oracle}(p)$ は w の $\mathit{Oracle}(p)$ における受理状態である. また,

$$\sum_{i \in \mathit{Oracle}(p)} |\mathit{SP}^{-1}(i)| = |p|$$

が成り立っており, $\mathit{Oracle}(p)$ に含まれる状態数は正確に $|p|+1$ 個であるから, 単純な $|\mathit{SP}^{-1}(i)|$ の平均を考えるとおおよそ 1 である.

2 Factor Oracle

この節では, いくつかの準備に続いて次の節で必要となる factor oracle の性質について示す. 大半は [ACR01] で示されているものである.

まず, 必要となる記号類を次のように定義する.

- $p = p[1 \ldots m]$ を Σ 上の長さ $|p|$ の文字列とする. $(p \in \Sigma^*, |p| = m)$
- ϵ を空文字列とする $(|\epsilon| = 0)$
- 関係 $a \overset{\mathrm{Suf}}{\prec} b$ (同様に $\overset{\mathrm{Pre}}{\prec}$) は a が b の接尾辞 (接頭辞) であることを表わす. すなわち適当な $u \in \Sigma^*$ に対して b は $b = ua$ $(b = au)$ と表わせる. また $u \neq \epsilon$ の場合, a は b の真の接尾辞 (接頭辞) であるという.
- 関係 $a \overset{\mathrm{Fac}}{\prec} b$ は a が b の部分列であることを表わす. すなわち, 適当な $u, v \in \Sigma^*$ に対して b は $b = uav$ と表わせる. また $u, v, a \neq \epsilon$ の場合, a は b の 真の部分列であるという.
- 任意のオートマトンに対して $\overline{w}$ でオートマトンが文字列 w の読み込みを終了した時点の状態を表わすとする.

続いて Factor Oracle の基本的性質について示す.

定義 2.1 ([ACR01]). $\mathit{Oracle}(p)$ は文字列 p から生成され, 状態 $0, 1, \ldots, |p| = m$ の $m+1$ 個の状態から成る p の全ての部分列を受理する非循環オートマトンである. すべての状態は終了状態であり, 丁度 $m+1$ 個の状態と m 以上 $2m+1$ 以下の遷移を持つ.

$\mathit{Oracle}(p)$ は, 順番に並んだ $m+1$ 個の状態 $0, 1, \ldots, |p| = m$ を遷移 $p[1], p[2], p[3], \ldots, p[m]$ (m 個) で継いだものに, 更にいくつかの遷移 ($0 \sim m+1$ 個) を加えた形をしている. 前者を内部遷移, 後者を外部遷移と呼ぶことにする. $\mathit{Oracle}(\text{“abbbaab”})$ を図 2 に示す.

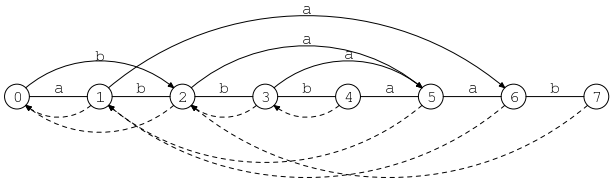

図 1: *Oracle*(“abbbaab”)

直線で表わされているのが内部遷移, 上側の左から右への実線の弧が外部遷移,
下側の右から左への破線の弧が suffix link である.

定義 2.2 ([ACR01]). $\mathit{endpos}_p(w)$ $w \overset{\mathrm{Fac}}{\prec} p$ のとき, 文字列 p の中での w の出現に対してそれぞれの出現の末尾の位置の集合. すなわち適当な $u \in \Sigma*$ に対して $\mathit{endpos}_p(w) = \{i | p = uwp[i] \dots p[m]\}$

定義 2.3 ([ACR01]). $\mathit{poccur}(w, p)$ $w \overset{\mathrm{Fac}}{\prec} p$ のとき, 文字列 p の中での w の最初の出現に対してその末尾の位置. すなわち $\mathit{poccur}(w, p) = \min\{\mathit{endpos}_p(w)\}$

定義 2.4 ([ACR01]). $\mathit{min}(i)$ で状態 $i \in \mathit{Oracle}(p)$ に対して, 初期状態から i へ到達する経路の中で最短のものを表わす.

オートマトンにおいて, ある経路とその経路を辿る文字列は一対一に対応するから, これらを区別せずに扱うことにする. すなわち文脈によっては $\mathit{min}(i)$ を文字列だと考えることがある.

定義 2.5 ([ACR01]). $\mathit{repet}_p(i)$ p の接頭辞 $p[0 \dots i]$ の接尾辞の集合 $\{p[j \dots i] | 0 \le j \le i\}$ 中で $p[0 \dots i]$ に二回以上出現するもののうち, 最長のもの.

次に [ACR01] で示されている Factor Oracle の基本的性質を示し, まず Factor Oracle を構成するときに使われる補助的構造: *suffix link* を説明する.

その後最少到達経路 $\mathit{min}(i)$ と suffix link の関係を明らかにし, そこから導かれるいくつかの性質を用いて, 本論文の主題に必要となる補題を示す.

補題 ([ACR01] Lemma 1). $u \in \Sigma^*$ を $i \in \mathit{Oracle}(p)$ で受理される文字列の中で最短のものとする (i.e., $\mathit{min}(i) = u$) このとき $u \overset{\mathrm{Fac}}{\prec} p$ かつ $i = \mathit{poccur}(u, p)$ である.

補題 ([ACR01] Lemma 2). i を $\mathit{Oracle}(p)$ 上の状態とすると i に受理される任意の文字列 $v \in \Sigma^*, \overline{v} = i$ に対して $u \overset{\mathrm{Suf}}{\prec} v$ である.

補題 ([ACR01] Lemma 3). $w \overset{\mathrm{Fac}}{\prec} p$ とすると $\overline{w} \le \mathit{poccur}(w, p)$ である.

補題 ([ACR01] Lemma 4). i を $\mathit{Oracle}(p)$ 上の状態とし, $u = \mathit{min}(i)$ とする. どんな u を末尾にもつ文字列 v ($u \overset{\mathrm{Suf}}{\prec} v$) も $\mathit{Oracle}(p)$ 受理されるならば, その受理状態 j は $j \ge i$ である.

定義 2.6 ([ACR01]). 状態 $i \in \mathit{Oracle}()$ の *suffix link* $S(i)$ は状態 $\overline{\mathit{repet}_p(i)}$ に等しい. また 状態 0 に対しては $S(0) = -1$ と定義しておくものとする.

Suffix link は Factor Oracle 構成に際して補助的な構造として使用され, 本論文の主題においても重要な役割を果す. 図 2 では点線で示される右から左への矢印が suffix link である.

補題 ([**ACR01**] **Lemma 7**). $s > 0$ を $S(s) \neq 0$ であるような $\mathit{Oracle}(p)$ 上の状態とすると, $\mathit{repet}_p(S(s)) \overset{\mathrm{Suf}}{\prec} \mathit{repet}_p(s)$.

[ACR01] Lemma 3 と Suffix Link の定義より, 任意の状態 $i \in \mathit{Oracle}(p)$ に対し常に $S(i) < i$ が成り立つから, ある状態 k_0 から始まって順に suffix link を辿って状態 0 までさかのぼるような一連の状態 $\{k_0 = i, k_1, k_2, \ldots, k_t = 0\}$ $(k_{j+1} = S(k_j))$ を考えることができる.

定義 **2.7** ([**ACR01**]). $SP(i)$ は状態 i から suffix link で辿れる一連の状態の集合と定義する. すなわち $SP(i) = \{i, S(i), S(S(i)), \ldots, 0\}$ であると定義する.

定義 **2.8.** $S^{-1}(i)$ は $S(j) = i$ であるようなすべての $\mathit{Oracle}(p)$ 上の状態 j の集合であるとする.

直観的には $S^{-1}(i)$ は 通常の suffix link を「逆向きに」辿ることを表わす.

系 **2.1.** 任意の状態 $i \in \mathit{Oracle}(p)$ は全てだた一つの suffix link $S(i)$ を持つから, i に対して $S(i)$ を親と考えれば, suffix link は全体で木構造を成すと見ることができる. 状態 0 はこの木構造の root であり, 任意の状態 i は子として $S^{-1}(i)$ (空集合の場合もある) をもつ節点であるとみなせる.

このような木構造を suffix-link-tree, また任意の状態 i を頂点とする部分木を i の suffix-link-subtree と呼ぶことにする.

定義 **2.9.** $i \in \mathit{Oracle}(p)$ に対して $SP^{-1}(i)$ は i の suffix-link-subtree に含まれる全ての節点 (状態) の集合だとする. すなわち, $SP^{-1}(i) = \{j | i \in SP(j)\}$

補題 ([**ACR01**] **Lemma 13**). どんな p の接尾辞 $u \overset{\mathrm{Suf}}{\prec} p$ に対しても $\mathit{Oracle}(p)$ 上の u の受理状態 $\overline{u}$ は $\overline{u} \in SP(p)$ を満す.

これらの補題をもとに次の関係が導ける.

補題 **2.1.** $i > 1$ を $\mathit{Oracle}(p)$ 上の状態とすると, 常に $|\mathit{min}(S(i))| < |\mathit{min}(i)|$ が成り立つ

Proof. $i = 1$ の場合 $|\mathit{min}(i) = 1|$, $|\mathit{min}(S(i))| = |\mathit{min}(0)| = 0$ より成り立つ.

$S(i) = 0$ の場合 $\mathit{min}(0) = 0$, $|\mathit{min}(i)| \geq 1$ より明らか.

$S(i) > 0$ の場合

すべての状態 s $(0 \leq s \leq i)$ に対し $|\mathit{min}(S(s))| < |\mathit{min}(s)|$ が成り立っていると仮定する. また, このとき $j = S(i)$ とおき, $k \in SP(i-1)$ を $j = \delta(k, \sigma)$ であるような状態とする. $(j \neq 0$ よりそのような k は必ず存在する).

ここで, 丁度 1 回の遷移で i に至るすべての状態の集合 $L = \{l' | \delta(l', \sigma) = i\}$ を考えると次の関係が成り立つ

$$|\mathit{min}(i)| = \min\left\{|\mathit{min}(l')| \,\middle|\, l' \in L\right\} + 1$$

更に構成アルゴリズムの定義より $L \subset SP(i-1)$ が成り立つ, すなわち任意の $l', l'' \in L,\ l' < l''$ に対して常に $l' \in SP(l'')$ が成り立つことに注意すれば, 帰納法の仮定より L 中で状態番号最小の要素を l とおくと, この l が L の要素中で最も短いの最少経路長を持つから $|min(i)| = |min(l)|+1$ が成り立つ.

加えて構成アルゴリズムの定義より $S(l) = k$ であることに注意すると, 次の関係が導かれる. (図 2 参照)

$$
\begin{aligned}
|min(i)| &= |min(l)| + 1 \\
&\geq |min(k)| + 2 && \text{— (1) } (S(l) = k \text{ と仮定より}) \\
|min(k)| + 1 &\geq |min(j)| && \text{— (2) } (\delta(k, \sigma) = j \text{ より})
\end{aligned}
$$

$$
\begin{aligned}
|min(i)| &\geq |min(k)| + 2 && \text{— (1) より} \\
&> |min(j)| && \text{— (2) より}
\end{aligned}
$$

□

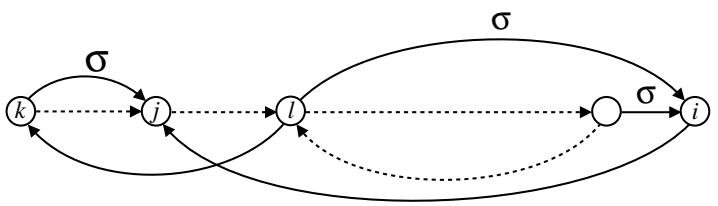

図 2: i, j, m, $m+1$ の関係

右から左への矢印は遷移, 左から右への矢印は suffix link. 実線で一つ, 破線で複数の遷移または suffix link を表わす.

補題 2.2. i を $Oracle(p)$ 上の状態, $j = S(i)$ とすると, $min(j) \overset{\text{Suf}}{\prec} min(i)$ が成り立つ[*1]

Proof. アルゴリズムの定義より $\overline{repet_p(i)} = S(i) = j$ なので [ACR01]–Lemma 2, Lemma7 より $min(j) \overset{\text{Suf}}{\preceq} repet_p(i)$ が成り立つ. また $repet_p(i)$ の定義より, $poccur(repet_p(i), p) < min(i)$ であるから

仮に $min(i) \overset{\text{Suf}}{\preceq} repet_p(i)$ と仮定すると [ACR01]–Lemma 3 より $poccur(min(i), p) \leq poccur(repet_p(i), p)$ が成り立つが, これは [ACR01]–Lemma 1 の要請, $poccur(min(i), p) = i$ に矛盾する. つまり $repet_p(i) \overset{\text{Suf}}{\prec} min(i)$ であるから, これと前半とを併せると $min(j) \overset{\text{Suf}}{\prec} min(i)$ が成り立つ. □

[ACR01] で示された AddLetter を少し変更すれば, 補題 2.1 の証明に用いた性質を使って次のように全ての状態 i に対して最短経路長 $|min(i)|$ を求めることができる.

[*1] 補題:2.1 はこの補題から実は明らかである.

Algorithm 1 NEWADDLETER($\mathit{Oracle}(p)$, σ)

```
 1. Create a new state m + 1
 2. Create a new transition from m to m + 1 by σ
 3. minlen[m+1] ← minlen[m]+1        /* Added */
 4. k ← S(m)
 5. while ( k > −1 ∧ there is no transition from k by σ)
 6.       Create a new transition from k to m by σ
 7.       minlen[m + 1] ← minlen[k]+1   /* Added */
 8.       k ← S(k)
 9. if (k = −1)
10.       S(m + 1) ← 0
11. else
12.       S(m + 1) ← δ(k, σ)
```

補題 2.3. アルゴリズム NEWADDLETEER を [ACR01] に示される BUILD_ORACLE に対して ADDLETTER の代りに用いれば, 全体の計算量 ($\mathcal{O}(n)$) に影響せずに $\mathit{Oracle}(p)$ の全ての状態 i に対して $|\mathit{min}(i)|$ を求めることができる.

Proof. 3 および 7 行だけが元のアルゴリズムに新たに加えられていてそれぞれ定数時間で実行可能である. (元のアルゴリズムが $\mathcal{O}(n)$ 時間で実行可能な証明は [ACR01]-Theorem 2 を参照のこと.) □

3 パターンの出現の有無, および出現位置の特定

この節では任意の文字列 w が $\mathit{Oracle}(p)$ に受理された時, w が $\mathit{Oracle}(p)$ の部分列であるか否かを判別する方法について考察する.

部分列 $p\big[(\overline{w} - |w| + 1) \ldots \overline{w}\big]$ と w を比較して一致すれば w が本当の p の部分列である ($\overline{w} = \mathit{poccur}(w, p)$ の場合) と判明するが, 実は $\overline{w} \neq \mathit{poccur}(w, p)$ の場合でも $\overline{w}$ から suffix link を逆向き (左から右) に一世代だけ辿ることで任意の文字列 w が p の部分列であるかを判別できることを示す. これは $\mathcal{O}(|w| \times S^{-1}(\overline{w}))$ で実行可能である.

またこのとき w が確かに p の部分列であれば, w の p 中での全ての出現位置 ($\mathit{endpos}_p(w)$) を $\mathcal{O}(|\mathit{endpos}_p(w)|)$ の計算時間の追加で特定できることも示す.

3.1 Suffix-Link-Tree と部分列の出現

まず最初に suffix-link-subtree と部分列の出現との関係を考える.

補題 3.1. 任意の部分列 w に対して $w \overset{\text{Fac}}{\prec} p$ ならば, $\mathit{endpos}_p(w) \subset SP^{-1}(\overline{w})$ である.

Proof. 任意の $i \in \mathit{endpos}_p(w)$ に対し [ACR01]–Lemma 13 より $\overline{w} \in SP(i)$ が成り立つ

さらに [ACR01]–Lemma 3 より $\forall i \in \mathit{endpos}_p(w), \overline{w} \leq i$ が言えるから suffix-link-tree に

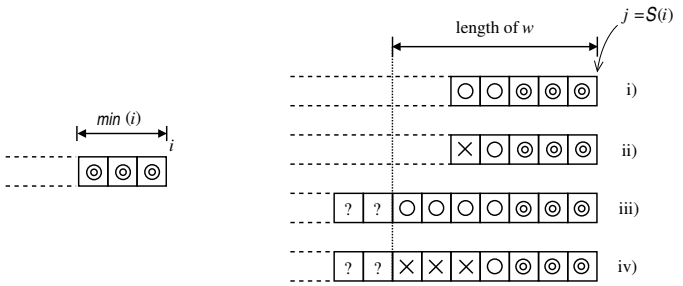

図 3: i, j と $\mathit{min}(i)$ の関係

四角い枠が一つの内部遷移 (=一つの文字に対応) を, 枠と枠の境界が状態を表わす. 明示的に示されている四角は $\mathit{min}(i)$(左) および $\mathit{min}(j)$(右) を表わす. ○ は w と一致した文字, × は一致しなかった文字を表わす. ◎ は [ACR01]–Lemma 2, および補題 2.2 より必ず w と一致することが保証されている文字を表わす.
なお i) の状況は起り得ない. (補題 3.2)

おいてどんな $i \in \mathit{endpos}_p(w)$ も $\overline{w}$ の子孫のノードになっている. つまり $\overline{w}$ を頂点とする suffix-link-subtree に $\mathit{endpos}_p(w)$ の全ての要素が含まれている. □

すなわち, $\mathit{endpos}_p(w) \subseteq \mathit{SP}^{-1}(\overline{w})$ が成り立っているので任意の部分列 w が $\mathit{Oracle}(p)$ に受理されたとき, 受理状態 $\overline{w}$ を頂点とする suffix-link-subtree の中にすべての w の出現 (の末尾の位置) が (もしあれば) 含まれる.

また, 補題 2.2 より次の性質が導かれる.

系 3.1. 任意の状態 i に対して $i \in \mathit{endpos}_p(w)$ のとき, $\mathit{min}(i) \overset{\mathrm{Suf}}{\preceq} w$ ならば $\mathit{SP}^{-1}(i) \subset \mathit{endpos}_p(w)$.

また逆に $i \notin \mathit{endpos}_p(w)$ のとき, $\mathit{min}(i) \overset{\mathrm{Suf}}{\prec} w$ と $w \overset{\mathrm{Suf}}{\prec} \mathit{min}(i)$ のどちらでもないならば[*2] $\mathit{SP}^{-1}(i) \cap \mathit{endpos}_p(w) = \emptyset$.

以上を踏まえて文字列 w が $\mathit{Oracle}(p)$ に状態 $i = \overline{w}$ で受理された場合を考える.

補題 3.2. 文字列 w が $\mathit{Oracle}(p)$ に状態 $i = \overline{w}$ で受理されたとする.

$\mathit{S}^{-1}(i) \neq \emptyset$ のとき $j \in \mathit{S}^{-1}(i)$ とすると, j と文字列 w の関係について次の 4 つの場合に分けられるが, どんな $j \in \mathit{S}(i)$ についても ii), iv) の場合は $\mathit{SP}^{-1}(j) \cap \mathit{endpos}_p(w) = \emptyset$, iii) の場合は $\mathit{SP}^{-1}(j) \subset \mathit{endpos}_p(w)$ が常に成り立つ. また i) は起りえない.

i) $|\mathit{min}(j)| \lneq |w|$, $\mathit{min}(j) \overset{\mathrm{Suf}}{\preceq} w$
ii) $|\mathit{min}(j)| \lneq |w|$, $\mathit{min}(j) \overset{\mathrm{Suf}}{\not\preceq} w$
iii) $|w| \leq |\mathit{min}(j)|$, $w \overset{\mathrm{Suf}}{\prec} \mathit{min}(j)$
iv) $|w| \leq |\mathit{min}(j)|$, $w \overset{\mathrm{Suf}}{\not\prec} \mathit{min}(j)$

Proof. ii), iv) の場合 $j \notin \mathit{endpos}_p(w)$, iii) の場合 $j \in \mathit{endpos}_p(w)$ であるから, ii), iv) の場合は $\mathit{SP}^{-1}(j) \cap \mathit{endpos}_p(w) = \emptyset$, iii) の場合は $\mathit{SP}^{-1}(j) \subset \mathit{endpos}_p(w)$ が系 3.1 より直ちに言える.

[*2] すなわち $\mathit{min}(i)$ と w の短い方が長い方の接尾辞になっていない場合.

i) の場合を考えると [ACR01]-Lemma 4 によれば $\mathit{min}(j) \overset{\mathrm{Suf}}{\prec} w$ の場合 $\overline{w} \geq j$ でなければならず, $\overline{w} < j$ という前提と矛盾が生じる. すなわち i) は起り得ない. □

すなわち, 任意の文字列 w が $\mathit{Oracle}(p)$ に受理されたとき w が本当に $\mathit{Oracle}(\mathrm{p})$ の部分列であるかは知るには, まず $\overline{w}$ が $\mathit{poccur}(w,p)$ に等しいか[*3]を調べ, 等しくない場合, すべての状態 $\in S^{-1}(\overline{w})$ それぞれについて補題 3.2 の ii), iii), iv) のうちどの場合に相当するかを調べれば良いことが分かった.

$\overline{w}$ が $\mathit{poccur}(w,p)$ に等しいか, または条件 iii) を満すような状態 $i \in S^{-1}(\overline{w})$ がひとつ以上存在すれば, w が p の部分列であることが言える. さらに補題 3.1 より $\mathit{endpos}_p(w) \subset SP^{-1}(\overline{w})$ を併せて考えれば, w が p の部分列であるのは以上の場合に限られる.

つまり全ての w の出現位置の集合 $\mathit{endpos}_p(w)$ は上記の条件 iii) を満す状態 $i \in S^{-1}(\overline{w})$ を頂点とする Suffix-Link-Tree に含まれる状態の集合に等しい. ($\overline{w} = \mathit{poccur}(w,p)$ の場合はそれに加えて $\{\overline{w}\}$ も含まれる.)

これをまとめると次の定理として表わせる.

定理 **1.** p 中の全ての w の出現 (の末尾位置) $\mathit{endpos}_p(w)$ は,

・$\overline{w} = \mathit{poccur}(w,p)$ の場合

$$\mathit{endpos}_p(w) = \{\overline{w}\} \cup \{j \in SP^{-1}(i) \mid i \in S^{-1}(\overline{w}) \text{ は iii) の条件を満すもの全て }\}$$

・それ以外の場合

$$\mathit{endpos}_p(w) = \{j \in SP^{-1}(i) \mid i \in S^{-1}(\overline{w}) \text{ は iii) の条件を満すもの全て }\}$$

である.

3.2 検索アルゴリズム

以上より, 部分列かどうかの判定をするアルゴリズムを擬似コードとして表わすと次のようになる.

Algorithm 2 IsSubstring?($\mathit{Oracle}(p)$, w)

```
1.  if w recognized in Oracle(p)
2.      i ← w̄ of Oracle(p)
3.  else
4.      return FALSE
5.  if p[(i − |w| + 1) ... i] = w
6.      return TRUE                    /* w̄=poccur(w,p) */
7.  else
8.      foreach j ∈ S^{-1}(i)
```

[*3] 部分列 $p[\overline{w} - |w| + 1 \ldots \overline{w}]$ と w が一致していれば等しい.

```
 9.        if p[(j − |w| + 1) ... j] = w
10.            return TRUE          /* case iii) in Lemma3.2 */
11.        else
12.            return FALSE         /* case ii) or iv) in Lemma3.2 */
```

定理 2. IsSubstring? は $\mathcal{O}(2|w| + |w| \times |S^{-1}(i)|)$ 時間で w が p の部分列であるか判別可能である.

Proof. 1 行目で w が $Oracle(p)$ に受理されるか否かは $\mathcal{O}(|w|)$ で判定できる. 5 行目の比較は $\mathcal{O}(|w|)$ である. 8 行目から始まる foreach 文は最大 $|S^{-1}(i)|$ 回実行されるから, 9 行目の比較は $|S^{-1}(i)| \times |w|$ 回実行されるである. □

また, IsSubstring? を少し変更して, もし $w \overset{\text{Fac}}{\prec} p$ ならば w の p 中でのすべての出現位置 $endpos_p(w)$ を求めることもできる.

Algorithm 3 FindAllOccurence($Oracle(p)$, w)

```
 1.  if w recognized in Oracle(p) at state w̄
 2.      i ← w̄ of Oracle(p)
 3.  else
 4.      return ∅                          /* empty set */
 5.  if p[(i − |w| + 1) ... i] = w
 6.      endpos_p(w) ← i
 7.  else
 8.      foreach j ∈ S^{-1}(i)
 9.          if p[(j − |w| + 1) ... j] = w
10.              endpos_p(w) ← endpos_p(w) ∪ SP^{-1}(j)
11.          else
12.              next
13.  return endpos_p(w)
```

定理 3. FindAllOccurence の実行には IsSubstring? の実行時間に加えて $|endpos_p(w)|$, すなわち p 中の w の出現回数に比例した時間だけ余計に必要である.

Proof. 考慮する必要があるのは 10 行目だけであるが, ここでは補題 3.2 の条件 iii) を満すような全ての $j \in S^{-1}(i)$ に対して $SP^{-1}(j)$ を頂点とする Suffix-Link-Tree の全探索を行う必要がある. これは明らかにこれらの Suffix-Link-Tree 全ての大きさに比例した時間が必要であり, 補題よりそれは $|endpos_p(w)|$ ($\overline{w} \notin endpos_p(w)$ の場合) または $|endpos_p(w)| - 1$ ($\overline{w} \in endpos_p(w)$ の場合) に等しい. □

なお, IsSubString? の 2, 6 行目 FindAllOccurence の 5, 9 行目 でそれぞれ $p\big[(i - |w| + 1) \dots w\big]$ および $p\big[(j - |w| + 1) \dots j\big]$ と文字列 w を比較していたが, $Oracle(p)$ に w が受理され

たとすれば

$$
\begin{aligned}
w\big[(|w| - |\mathit{min}(i)| + 1) \ldots |w|\big] &= \mathit{min}(i) && \text{—[ACR01]–Lemma 2 より} \\
&= p\big[(j - |\mathit{min}(i)| + 1) \ldots j\big] && \text{—補題 2.2 より}
\end{aligned}
$$

が成り立っているので, 実はそれぞれ $p\big[(i - |w| + 1) \ldots (i - |\mathit{min}(w)|)\big]$ および $p\big[(j - |w| + 1) \ldots (j - |\mathit{min}(w)|)\big]$ との比較十分であるので, NewAddLetter を用いて予め全ての i に対して $|\mathit{min}(i)|$ を計算しておけば, 比較する文字の個数を最大 $|\mathit{min}(i)| \times |S^{-1}(i)|$ だけ減らすことができる.

4 結論

オートマトンの一種である Factor Oracle を索引に利用した新しい全文検索アルゴリズムを示した. 文字列 p の索引の生成には $|p|$ に比例した時間がかかり, パターン w が文字列 p の中に出現するか否かを調べるのに必要な時間は, パターン w の長さ $|w|$ に比例する. w が p の部分列であるときは, その最初の出現位置も同時に特定可能である. またそれに加えて w の出現回数に比例した時間で p 中の全ての w の出現位置を特定することができる.

アルゴリズムの実行に必要な時間は正確にはパターン w に対して $\mathcal{O}(|w| \times |SP^{-1}(\overline{w})|)$, すなわち w の受理状態 $\overline{w}$ に入ってくる suffix link の本数にも比例する. suffix link の本数は $\mathit{Oracle}(p)$ 全体で丁度 $|p|$ 本あるので, 与えられたパターンに対して特定の状態が受理状態になる確率がどの状態についても等しいと仮定して単純な平均を考えるならば, この値はほぼ定数とみなせる.

今後の課題として, 現実の様々データ, 例えば DNA やタンパク質のアミノ基の配列などについて $|SP^{-1}(\overline{w})|$ がどのような振舞いを見せるか, またそれに関連してアルゴリズムの実際の実行速度がどの程度になるか等について実験や考察が必要だと考えられる.

参考文献

[ACR01] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Efficient experimental string matching by weak factor recognition. *Lecture Notes in Computer Science*, 2089:51–??, 2001.

[Apo85] A. Apostolico. The myriad virtues of suffix trees., 1985.

[BBE+85] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen, and J. Selferas. The smallest automaton recognizing the subwords of a text. In *Theoretical Computer Science*, volume 40, pages 31–55, 1985.

[CV97] M. Crochemore and R. Vérin. Direct construction of compact directed acyclic word graphs. In A. Apostolico and J. Hein, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, number 1264, pages 116–129, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.

[IHS+01] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. On-line construction of compact directed acyclic word graphs. In *Combinatorial Pattern Matching*, pages 169–180, 2001.

[MM90] U. Manber and G. Myers. A new method for on-line string searches. In *1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327, 1990.

[Sad98] K. Sadakane. A fast algorithm for making suffix arrays and for burrows-wheeler transformation. In *Proceedings of IEEE Data Compression Conference (DCC'98)*, pages 129–138, March 1998.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 3(14):249–260, 1995.