ファクターオラクルを用いたコンパクトな全文検索索引の構築.

2021年1月31日

佐藤 明幸

第1章　序論

1.1　研究の背景

全文検索問題とは、文字列集合 $D = T_0..T_n$ から検索したいパターン p の (テキスト番号, 維持番号) の組からなる位置情報をすべて出力する問題である。簡潔に述べると、複数のテキストファイルから特定のパターンの出現位置を探すということである。

情報処理の多くの分野において全文検索問題を含むパターンマッチングの問題は重要な問題であり、データ処理、文書編集、字句解析、情報検索の中でよく使われる。ウイルス対策ソフトがウイルスを発見するための最も一般的な方式として用いられる。また、多くのテキストエディタやプログラミング言語にはパターンを照合する機能が含まれている。この問題は情報工学において中心的な問題として幅広く研究されたものの一つである。

全文検索問題へのアプローチとしては大きく分けて逐次型と索引型に分類される。逐次型は検索対象となる複数のファイルをそれぞれ逐次に探索し、検索パターンを全て探し出すため、検索にテキスト長に比例した時間がかかってしまう問題がある。これに対し索引型では、事前にテキストから作成した索引を検索に用いることで検索を高速化することができる。ただし索引の作成には時間がかかり、検索を行うためには索引を保持するためのメモリ空間が必要となることに留意しなければならない。一般的な索引型の手法としては Suffix array 等のデータ構造を用いたものがある。他には DAWG(Directed Acyclic Word Graph)[4] というオートマトンを使用した手法があり、こちらはビット列で表現することで Suffix array を使用した場合よりも少サイズとなることが [5] で述べられていた。

Allauzen 他は、文字列に対し、ファクターオラクルと呼ばれる新たなデータ構造を考案し、文字列パターン照合に応用した。文字列 w に対するファクターオラクル [1] は、少なくとも w のすべての部分文字列を受理するオートマトンである。状態数は $|w|+1$、遷移数は $2|w|-1$ である。

ファクターオラクルを索引に用いる難しさは、あるテキスト w から構成されたファクターオラクル $Oracle(w)$ があるパターン p を受理したとき、もとのテキスト w に p が出現しているかどうかわからない擬陽性をもつという点にある。

ファクターオラクルを用いることでほぼ線形時間でパターンマッチングを行うアルゴリズム [2] が知られている。

ファクターオラクルが単一の文字列から構成されるのに対し、先行研究 [3] では、複数文字列に対するファクターオラクルを定義することで、全文検索索引へ活用していた。また、DAWG を用いた場合よりも状態数、遷移数共に少サイズで表現できることが実験的に述べられていた。

1.2 研究の目的

単一のテキストに対する効率的な検索手法は数多く提案されてきたが、複数の文字列集合の対する全文検索の手法は多くはない。

そこで先行研究では、文字列集合からトライ木を構成し、そのトライ木を入力としてファクターオラクルと部分集合木を構成し全文検索索引としていた。しかしこの方法では構成、検索の共に線形時間で実行することは不可能であった。

本研究では、先行研究の問題を解決し、ファクターオラクルを用いた線形時間で構築と検索が可能な全文検索索引を提案することを目的とする。

第2章　諸定義

2.1　オートマトン

一般的にオートマトンとは、入力によって内部状態を遷移させ、処理や入力を行う計算機モデルである。図にあるような状態遷移図として表される。

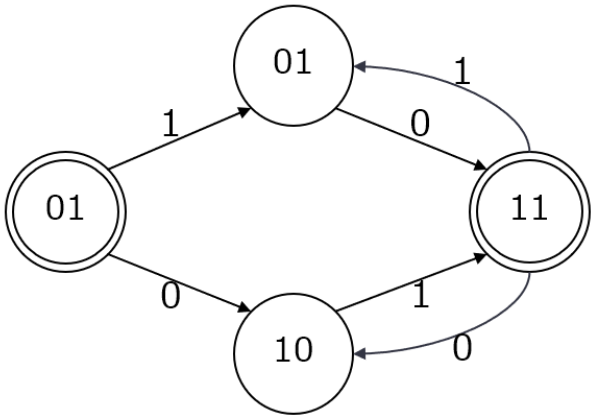


図 2.1: オートマトン例図

図中の丸が状態を表しており、矢印が外部からの入力によってどの状態に映るかを表している。ここで、初期状態は状態 00 であり、1 を入力とした場合には状態 01 へ、0 を入力とした場合には状態 10 へ状態を遷移することになる。さらに、二重丸で表された状態のことを特に受理状態という。全ての入力を受け、遷移し終わった後に、受理状態に到達した場合、オートマトンが入力を受理したことになり、受理と出力する。そうでない場合は非受理となる。状態と入力によって次に遷移する状態が一位に定まり、状態数が有限このものを決定性有限オートマトン (DFA) という。図の例では空気号列か、松木が 10 または 01 で終わる文字列を受理する決定性有限オートマトンである。

決定性有限オートマトン M は以下の 5 字組で定義される。

$$M = (Q, \Sigma, \delta, q_0, F)$$

ここで、
Q : 状態 $(state)$ の有限集合
Σ : 入力記号 $(inputsymbol)$ の有限集合
δ : 遷移関数 $(movefunction)(\delta : Q \times \Sigma \to Q)$

q_0 : 初期状態
F : 受理状態の有限集合
である。

2.2 全文検索問題

任意の文字列 $w,x,y,z \in \Sigma^*$ に対し、$w = xyz$ と表すことができるとき、x を w の**接頭辞**、z を w の**接尾辞**, y を w の**部分文字列**と呼ぶ。このとき、$FACT(w)$ を

$$FACT(w) = \{y \mid y \text{ は } w \text{ の部分文字列 }\}$$

と定義する。

検索対象となる文字列を**テキスト**といい、T と表す。また、検索する文字列を**パターン**といい、P と表す。テキストには先頭の文字から 1 文字ずつ順に番号を付ける。これを**位置番号**と呼ぶ。n 文字あるテキスト T は $T = t_1t_2...t_n$ と表す。テキストが N 個ある場合、文字列集合を D とおき、$D = T_1, T_2, ..., T_N$ と表す。議論の簡単化のため、任意の T_iはいかなる T_j の接頭辞ではないとする。各テキストにつけられた番号を**文章番号**と呼ぶ。本研究における全文検索問題とは、P が与えられたとき、$P \in FACT(D)$ を満たす文章を見つけ、その文章番号と位置番号の組を全て出力する問題である。

$L(M)$ を $DFAM$ によって受理される言語とすると、

$$FACT(D) = \{y \mid \exists w \in L(M), y \text{ は } w \text{ の部分文字列 }\}$$

と定義する。このことより、本研究では $FACT(D)$ に含まれる全ての部分文字列を受理できるオートマトンを構成し、そのオートマトンに入力として文字列 P を与え、受理可否受理であるかを見ることで全文検索を実現する。

第3章　ファクターオラクル

3.1　ファクターオラクルとは

ファクターオラクル [1] は Allauzen 他によって 1999 年に発表されたオートマトンである。p に対するファクターオラクル $Oracle(p)$ は、以下の性質を持つ。

1. 少なくとも p のすべての部分文字列を受理する。
2. 全ての状態が受理状態である。
3. 状態数が $|p|+1$ 個、遷移数が $2|p|-1$ 個以下
4. 受理した文字列が入力のテキスト p に含まれていない可能性がある。すなわち $Oracle(p)$ は**擬陽性**を持つ。

3.2　Allauzen の構成アルゴリズム

本章では、ファクターオラクル $Oracle(p)$ の構成法を示す。ここで、アルゴリズムで使用する supply function S を以下のように定義する。

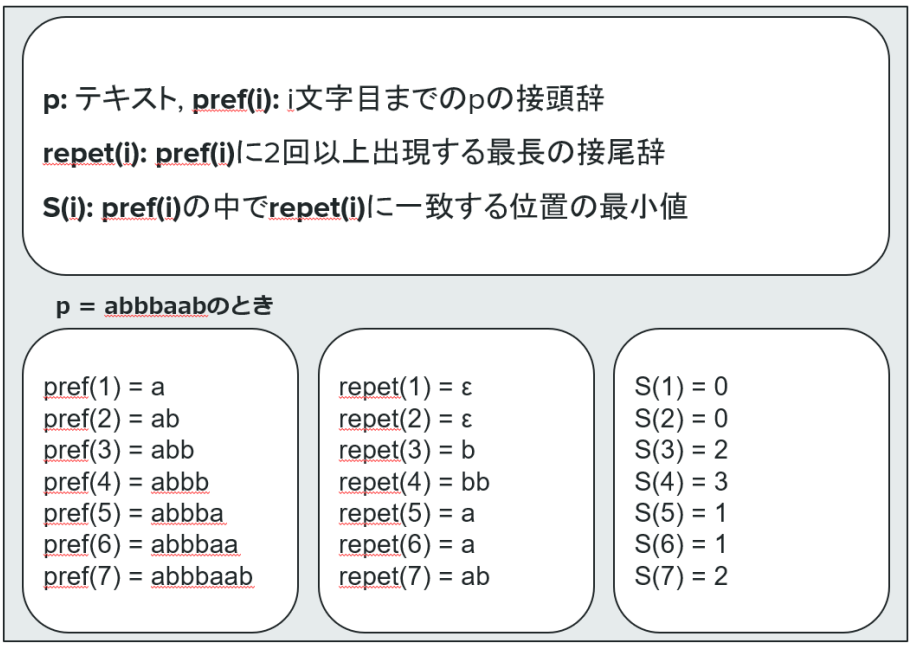


図 3.1: supply function S の定義

ファクターオラクルは以下のあるテキスト p を引数としてオンラインアルゴリズム create_oracle で構成される。

Algorithm 1 create_oracle$(p = p_1p_2...p_m)$

1: *Create Oracle*(ε) *with* :
2: *one single state* 0
3: $S_\varepsilon(0) \leftarrow -1$
4: **for** $i \leftarrow 1..m$ **do**
5: $Oracle(p_1p_2...p_i) \leftarrow add\_letter(Oracle(p = p_1p_2...p_{i-1}), p_i)$

create_oracle はオンラインアルゴリズムであり、テキスト p を 1 文字ずつ読み込み内部で関数 add_letter を呼び出している。関数 add_letter の定義は以下のようになっている。

Algorithm 2 add_letter$(Oracle(p = p_1p_2...p_m), \sigma)$

1: *Create a new state* $m+1$
2: *Create a new transition from* m *to* $m+1$ *labbeled by* σ
3: $k \leftarrow S_p(m)$
4: **while** $k > -1$ *and there is no transition from* k *by* σ **do**
5: *Create a new transition from* k *to* $m+1$ *by* σ
6: $k \leftarrow S_p(k)$
7: **if** $k = -1$ **then**
8: $s \leftarrow 0$
9: **else**
10: $s \leftarrow$ *where leads the transition from* k *by* σ.
11: $S_{p\sigma}(m+1) \leftarrow s$ **return** $Oracle(p = p_1p_2...p_{m\sigma})$

具体例として、p ='abbbaab' を create_oracle の実行例を以下に示す。

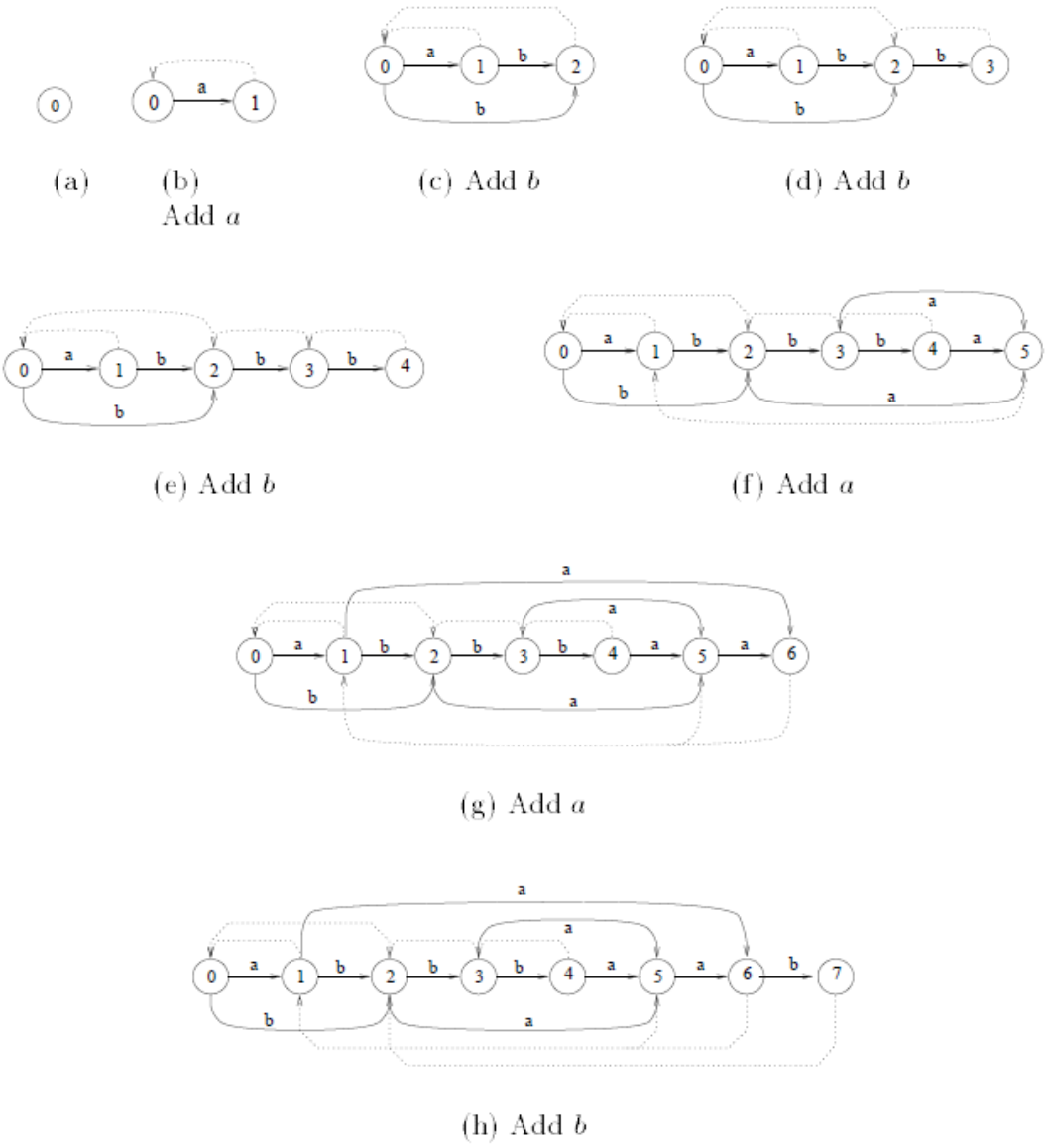


図 3.2: create_oracle による Oracle('abbbaab') の構成例 点線は supply function を示す [1] より引用

3.2.1 内部遷移

アルゴリズム 2 の 2 行目で作られた遷移を内部遷移と定義する。

内部遷移をたどることで入力のテキストを保持することなく参照することができ、索引が使用するメモリの量を削減することができる。

3.3 サフィックスリンクツリー

前節で述べたように、ファクターオラクルは状態数と遷移数が少なく、効率的なオートマトンであるが、擬陽性を持つ。Oracle('abbbaab') では、入力のテキスト'abbbaab' の部分文字列ではないパターン'aba' を誤って受理してしまうことが確認できる。

ファクターオラクルを部分文字列検索の検索索引として使用するためにはこの擬陽性を解消する必要がある。[2] では、擬陽性を解消し、更に効率的に検索を行うためにサフィックスリンクツリーという木構造が定義されていた。

図 3.2 で点線で各状態をつないでいた supply function の値を全てつなげたものがサフィックスリンクツリーである。Oracle('abbbaab') に対するサフィックスリンクツリーを以下に示す。

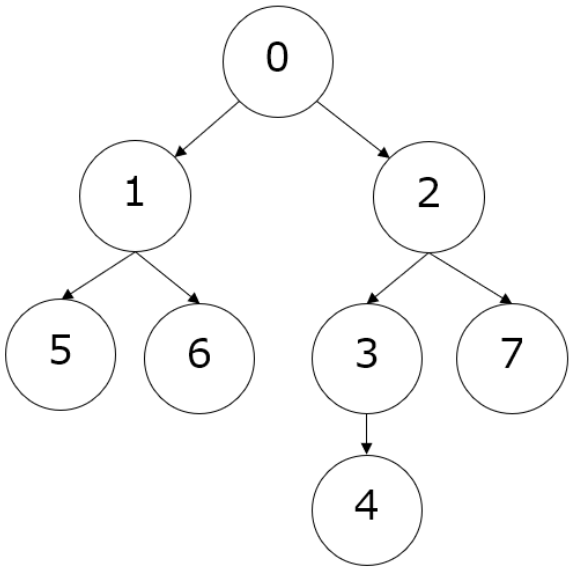


図 3.3: Oracle('abbbaab') に対するサフィックスリンクツリー

3.4 検索アルゴリズム

本節では、ファクターオラクルとサフィックスリンクツリーを用いた部分文字列検索アルゴリズムを紹介する。

ここで、以下を定義する。

1. $S^{-1}(i)$ は $Oracle(p)$ のサフィックスリンクツリーのノード i の全子ノードの集合
2. $SP^{-1}(i)$ は $Oracle(p)$ のサフィックスリンクツリーのノード i の全子孫ノードの集合

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

1: **if** w *recognized by* $Oracle(p)$ *atstate* $\overline{w}$ **then**
2: $\quad i \leftarrow \overline{w}\ of\ Oracle(p)$
3: **elsereturn** $\emptyset$ $\quad /* \ empty\ set \ */$
4: **if** $p[(i-|w|+1)...i] = w$ **then**
5: $\quad output \leftarrow i$
6: **for** $j \in S^{-1}(i)$ **do**
7: $\quad$ **if** $p[(j-|w|+1)...j] = w$ **then**
8: $\quad\quad output \leftarrow output \cup SP^{-1}(j)$
9: $\quad$ **else** *continue*
return *output*

検索アルゴリズムは、

1. まず最初に入力のパターン w で $Oracle(p)$ を遷移させ、受理した状態番号 $\overline{w}$ を得る。受理しなければ空集合を返す。

2. その状態番号に対応するテキスト p の位置に w が出現していれば出現位置の集合 $output$ に $\overline{w}$ を追加する。

3. $Oracle(p)$ があるパターン w を受理した場合の出現位置の候補は $\overline{w}$ だけではなく、$SP(\overline{w})$ の全てに及ぶ。よって、Algorithm 3 の 9 行目からは、テキスト p の各 $j \in S^{-1}(\overline{w})$ の値に対して w が出現しているか判定している。もし出現していれば、$output$ に $SP^{-1}(j)$ を追加し、そうでなければ追加しない。

このアルゴリズムは、$\mathcal{O}(|S^{-1}(\overline{w}) \times w)$ の実行時間を要す。$\mathcal{O}(|S^{-1}(i))|$ は最悪で $\mathcal{O}(p)$ になる。しかし、$Oracle(p)$ のサフィックスリンクツリーのノード数が p であるため、ならし計算量では $\mathcal{O}(1)$ となることがわかる。

よってアルゴリズム全体でのならし計算量は $\mathcal{O}(w)$ である。

第4章　文字列集合に対するファクターオラクル

前章では、単一のテキストに対して効率的に検索を可能にするデータ構造を紹介した。我々の研究では複数のテキストを含むような文字列集合に対して、効率的に検索を行いたい。本章では、先行研究 [3] で提案されていた文字列集合に対するファクターオラクルを用いた全文検索索引について解説する。

4.1　文字列集合に対するファクターオラクルとは

文字列集合に対するファクターオラクルは、ファクターオラクルを全文検索に応用するために [3] で定義されている。

4.2　構成アルゴリズム

文字列集合に対するファクターオラクルは、

1. 入力の文字列集合からトライ木を構成する
2. トライ木に部分集合構成法を適用する

以上の手順で構成される。

4.2.1　トライ木の構成

Algorithm 4 BuildDFA(D)

Input $D = T_1, T_2, ..., T_N$
$s \leftarrow 0, Q \leftarrow \{s\}, statenum \leftarrow 0$
for $i = 1...N$ **do**
    for $j = 1...N$ **do**
        if $\delta(s, t_j = s')$ **then**
            $s \leftarrow s'$
        else
            $statenum \leftarrow statenum + 1$
            $\delta(s, t_j) = statenum, Q \leftarrow Q \cup \{statenum\}$
            $s \leftarrow statenum$
    $F \leftarrow F \cup \{s\}$
return $M = (Q, \Sigma, \delta, q_0, F)$

BuildDFA を $D = \{$"*abbac*","*aaaac*"$\}$ を入力として実行した例を以下に示す。

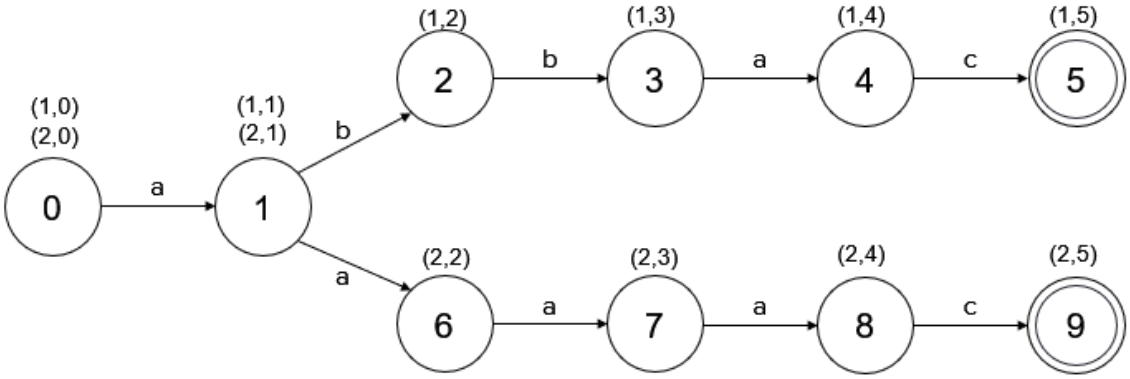


図 4.1: $D = \{$"*abbac*","*aaaac*"$\}$ を入力としたときのトライ木

4.2.2 文字列集合に対するファクターオラクルの構成

本説では文字列集合に対するファクターオラクルの構成法を示す。本アルゴリズムは前節で示したアルゴリズムで構成したトライ木の全ての状態を初期状態と置き、非決定性有限オートマトン (NFA) から DFA を構成する**部分集合構成法**を、応用したアルゴリズムである。また、$s = \{i_1, i_2, ..., i_j\}(i_1 < i_2 < ... < i_j)$ をトライ木Mの状態の部分集合としたとき、$min(s) = i_1$ と定義する。

BuildOracle を $D = \{$"*abcba*","*abbac*"$\}$ から構築したトライ木を入力として実行した例を以下に示す。

文字列集合に対するファクターオラクルは少なくとも $FACT(D)$ の全ての文字列を受理するが、擬陽性を持つ。

Algorithm 5 BuildOracle(M)

Input $M = (Q, \Sigma, \delta, q_0, F)$
$s_0 \leftarrow Q, m \leftarrow |Q|, Q' \leftarrow s_0$
for $i = 0...m$ **do**
    s を $min(s) = i$ になる Q'の状態とする。もしこのような状態がなければ *next*
    for all $\sigma \in \Sigma$ **do**
        $s' = \delta(s, \sigma)$ を計算する
        if $minn(s') = min(t)$ となる t が存在する **then**
            $\delta'(s, \sigma) = t$
        else
            $\delta'(s, \sigma) = s', Q' \leftarrow Q' \cup \{s'\}$
$F' = Q'$
return $M_{FO} = (Q', \Sigma, \delta', q_0, F')$

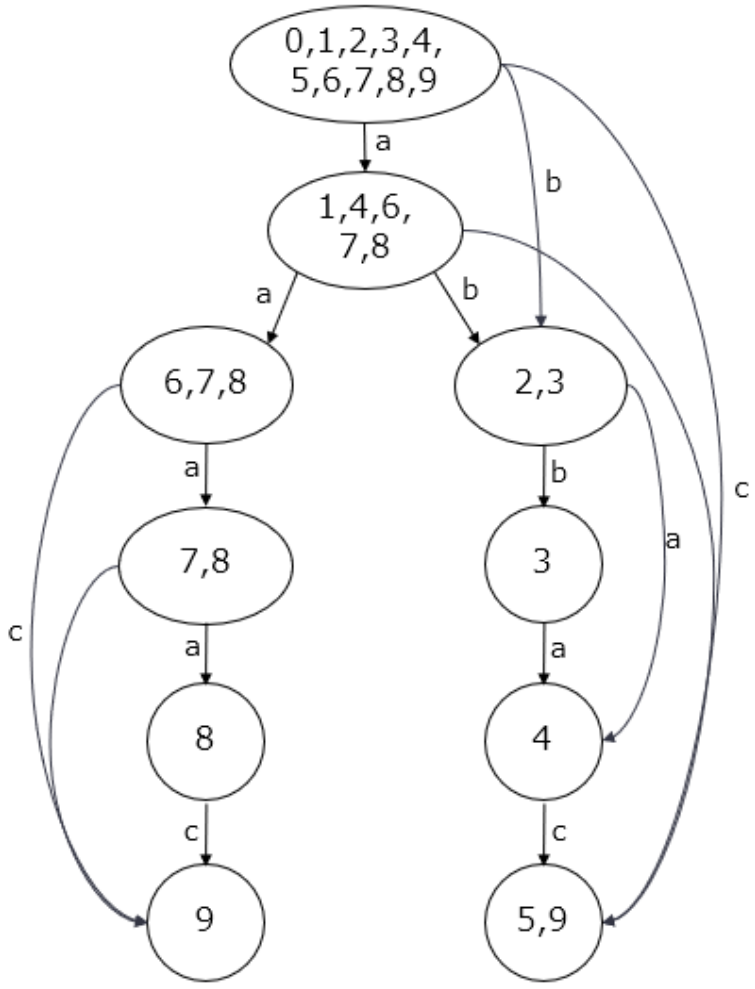


図 4.2: $D = \{$"*abcba*","*abbac*"$\}$ から構築したトライ木を入力としたときの文字列集合に対するファクターオラクル

4.3 全文検索アルゴリズム

本節では文字列集合に対するファクターオラクルを使用した全文検索アルゴリズムを示す。

文字列集合に対するファクターオラクルがあるパターン p を状態 x で受理した時、出現位置の候補は x に対応する位置だけではなく、$\bar{y} \subseteq \bar{x}$ となる全て状態が候補となる。候補を素早く得たり、コンパクトに表現するために Allauzen のファクターオラクルではサフィックスリンクツリーを使用したように、文字列集合に対するファクターオラクルでは部分集合木という木構造を使用する。

4.3.1 部分集合木

索引のサイズ索引のため、**部分集合木**という木構造を使用する。部分集合木 T_W を次のように定義する。

1. T_W の頂点は、ファクターオラクル M_{FO} の状態からなる。
2. T_W の根は q_0、つまり M_{FO} の初期状態である。
3. 任意の頂点に含まれる番号集合を X、その子である頂点に含まれる番号集合 Y としたとき、$Y \subset X$ である。

部分集合木の各頂点には、(文章番号, 位置番号) の組からなる位置情報を付与する。検索時には、この位置情報を参照することで位置情報の候補を絞る。ファクターオラクルの各状態に位置情報を付与する手もあるが、この場合は異なる状態に同じ位置情報が複数つけられてしまう。部分集合木に位置情報を付与することで、結果としてよりコンパクトな索引となる。

4.3.2 検索アルゴリズム

ここまでで、文字列集合に対するファクターオラクルを使用した索引は、文字列集合に対するファクターオラクル M_{FO} と部分集合木 T_W の 2 つのデータ構造で構成されることが分かった。本節ではこの索引を使用した検索アルゴリズムを述べる。

Algorithm 6 Search(P)

Input P
M_{FO} に P を入力
$f = \delta(s_0, P)$ とする
if $f \in F'$ **then**
    $f \in V$ を根とする部分木にある位置情報 (i, j) を全て取り出す
各位置情報について T_i の j 文字目から先頭方向に P と比較し、一致したら R に位置情報を加える **return** R

この検索アルゴリズムの実行時間は $\mathcal{O}(p \times |M_{FO}|)$ である。また構築にも $|M_{FO}|^2$ の時間がかかる。

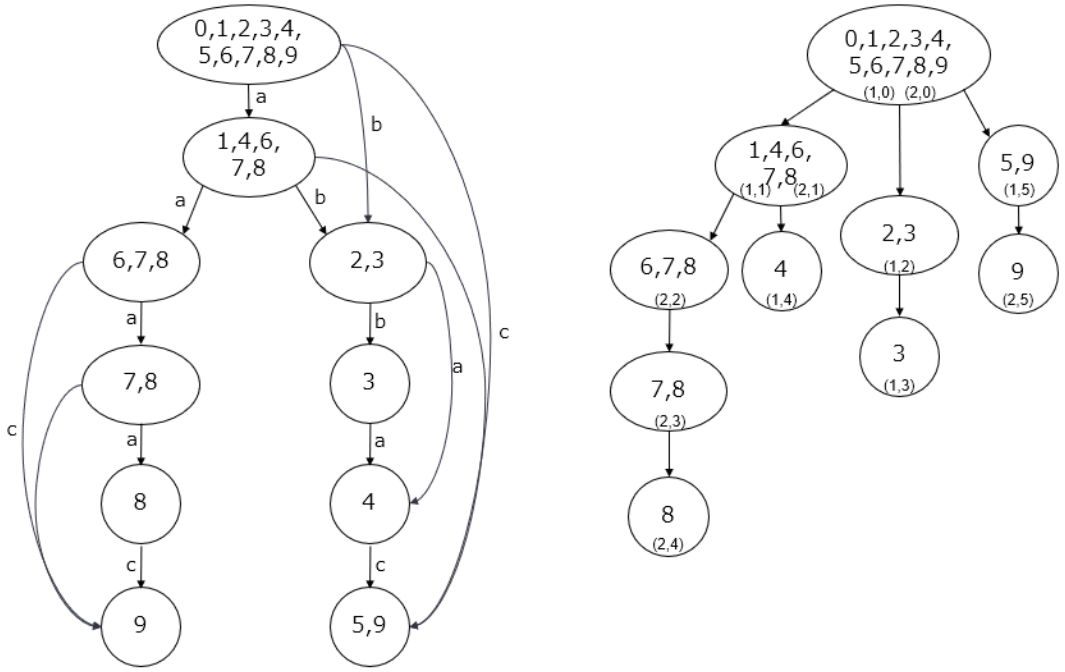


図 4.3: $D = \{$"*abcba*","*abbac*"$\}$ から構築した索引 (左:ファクターオラクル、右:部分集合木)

第5章　提案手法

本章では、本研究の提案手法を解説する。

5.1　先行研究の問題点と改善案

文字列集合に対するファクターオラクルを用いた索引は、文字列集合から構築したトライ木の状態数を n、検索パターン長を p としたとき、構築に $\mathcal{O}(n^2)$、検索に $\mathcal{O}(p \times n)$ の時間がかかることが分かった。

本研究では、構築を $\mathcal{O}(n)$、検索を $\mathcal{O}(p$ の出現回数$)$ に改善する。

方針としては、3 章で述べた単一テキストに対して線形時間で構築と検索を行うことができる Allauzen の構築アルゴリズムと Kato の検索アルゴリズムを複数文字列に拡張することを考える。さらに、DFA 最小化アルゴリズムを適用することでさらなる少サイズ化を試みる。

提案アルゴリズムは 3 つの手順で構成される。

1. 入力の文字列集合からトライ木を構成する
2. トライ木に拡張した Allauzen のアルゴリズムを適用し、文字列集合に対するファクターオラクルを構築する
3. 文字列集合に対するファクターオラクルを最小化する

5.2　Allauzen のアルゴリズムの拡張

先述の通り、Allauzen のアルゴリズムは入力として単一テキスト T を入力とすることで T に対するファクターオラクルを構築しているため、入力を文字列集合に拡張する必要がある。そこで、前章の先行研究と同様に文字列集合から構築したトライ木を入力とすることでこれを解決する。

拡張した Allauzen のアルゴリズムを以下に示す。

関数 add_state の定義は以下のようになっている。

具体例として、p ='abbbaab' を create_oracle の実行例を以下に示す。

Algorithm 7 create_oracle(M)

Create Oracle(ε) *with* :
    one single state 0
    $S_\varepsilon(0) \leftarrow -1$
for all $q \in Q$ **do**
    for all $\sigma \in \Sigma$ **do**
        $i \leftarrow \delta(q, \sigma)$
        $Oracle \leftarrow add\_state(Oracle, q, \sigma, i)$
return *Oracle*

Algorithm 8 add_state$(Oracle, q, \sigma, i)$

Create a new state i
Create a new transition from q to i labbeled by σ
$k \leftarrow S(q)$
while $k > -1$ *and there is no transition from k by* σ **do**
    Create a new transition from k to i by σ
    $k \leftarrow S(k)$
if $k = -1$ **then**
    $s \leftarrow 0$
else
    $s \leftarrow$ *where leads the transition from k by* σ.
$S(i) \leftarrow s$
return *Oracle*

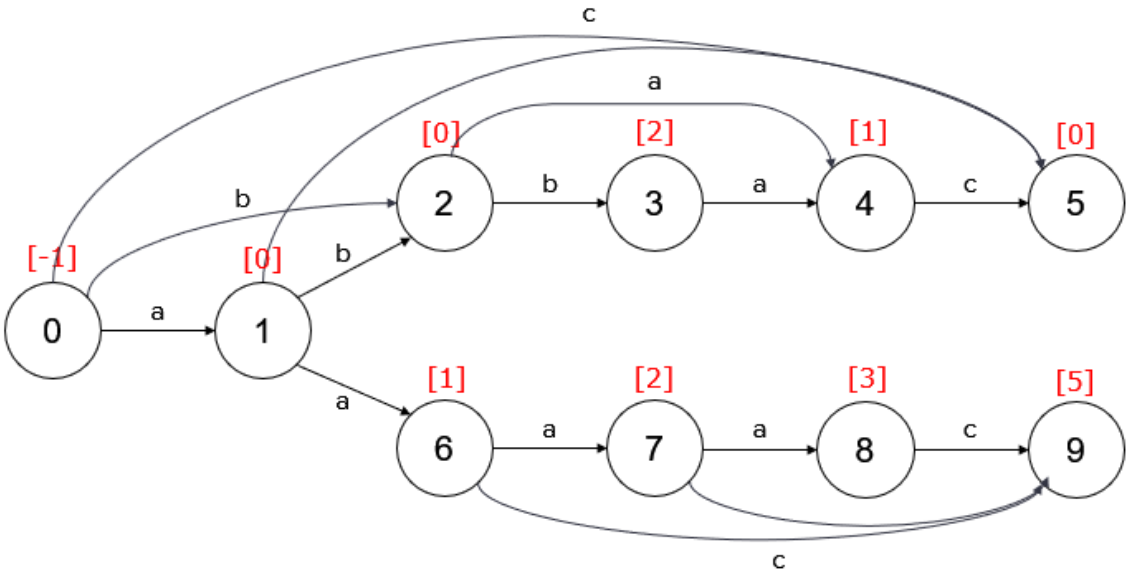


図 5.1: create_oracle による Oracle('abbbaab') の構成例 各状態上の赤字は supply function を示す [1] より引用

5.3 ファクターオラクルの最小化

本節では、ファクターオラクルの最小化について述べる。 ファクターオラクルを最小化することによって、より索引の省サイズ化が見込める。

オートマトンの状態数を n と置く。DFA の最小化には、一般的には $\mathcal{O}(n^2)$ を要するが、非巡回型 DFA から構成したファクターオラクルは常に非巡回型 DFA であるため、ここでは $\mathcal{O}(n)$ で動作する非巡回型 DFA に対する最小化アルゴリズムを用いる。

また、ファクターオラクルの最小化をより検索に支障が出ない範囲の内で効率的に行うため、最小化の際の等価性の検証には内部遷移のみを使用する。

以下に非巡回型 DFA を入力とする最小化アルゴリズムを示す。

1. 入力の DFA の状態のうち、遷移を持たない状態を統合する。
2. 統合された状態に、同じラベルでの遷移を持つ状態を統合する。
3. 統合が続く、もしくは初期状態に到達するまで 2 を繰り返す。

また、統合された状態の情報は検索アルゴリズムで使用するため、保持しておく必要がある。

5.4 検索アルゴリズム

本章で解説した文字列集合に対するファクターオラクルの構築アルゴリズムは 3 章で解説したファクターオラクルのものと非常に似ており、検索アルゴリズムも同様である。

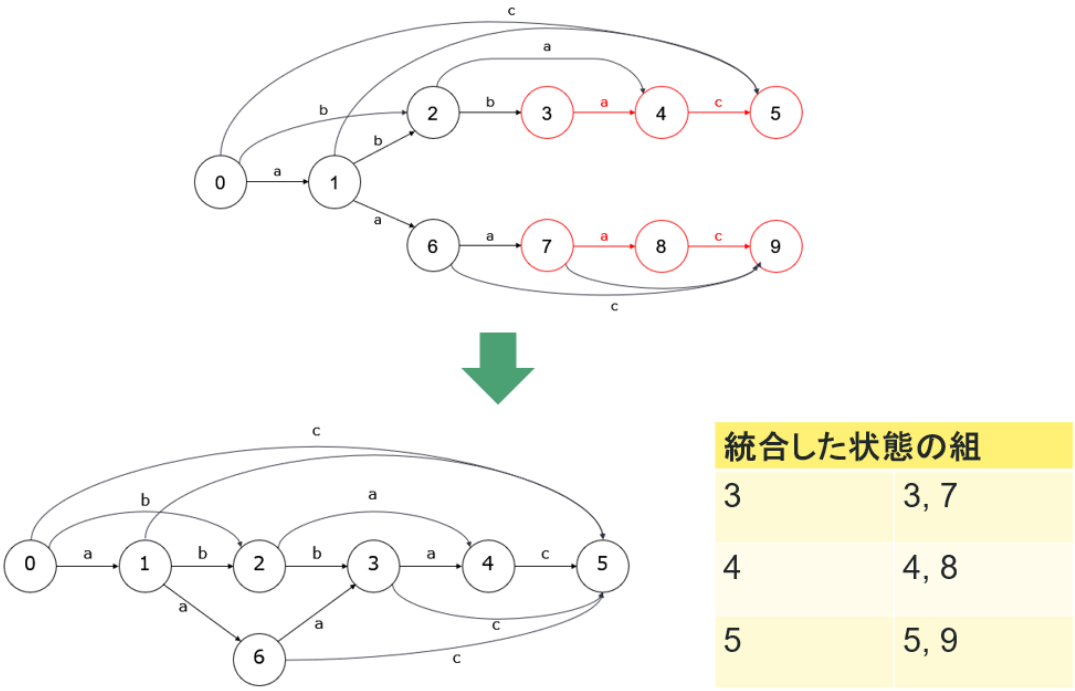


図 5.2: ファクターオラクル最小化の例

変更点としては、最初の判定で最小化によって統合された状態全てに対して検証する必要がある点と、複数テキストを入力としているために、状態番号がそのまま出現位置を示さなくなった点がある。そのため本手法では、最小化によって統合された状態番号を保持する *pair* と各状態に対応する位置情報を保持する *endpos* という配列を定義している。

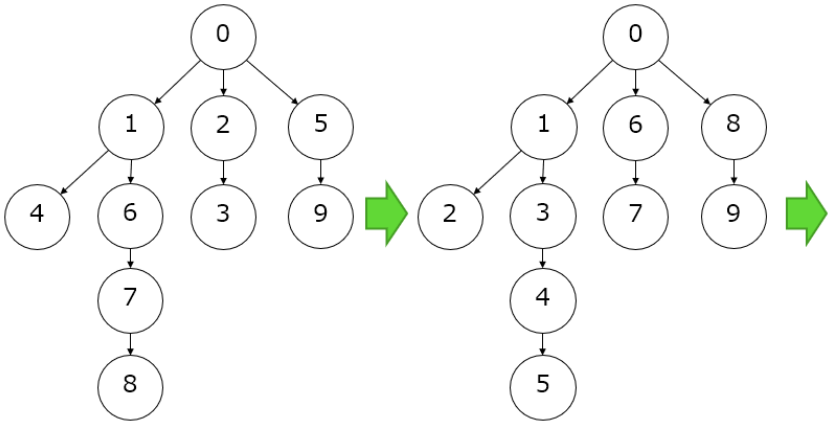
本節では、アルゴリズムで使用する SP_{-1}、S_{-1}, *endpos* の実装方法について解説した後、検索アルゴリズムを示す。

5.4.1 検索アルゴリズムで使用するデータの表現

SP_{-1} を素直に実装すると、最悪の場合ファクターオラクルの状態数 n の二乗の空間を使用する。

これを解消するため、SP_{-1} をサフィックスリンクツリーのあるノードの子孫ノードの集合を保持するのではなく、子孫ノード集合の番号の最小値と最大値の組のみを保持するように変更することで、要素数 n のタプルの配列として実装できる。(使用空間: $2 \times n$)

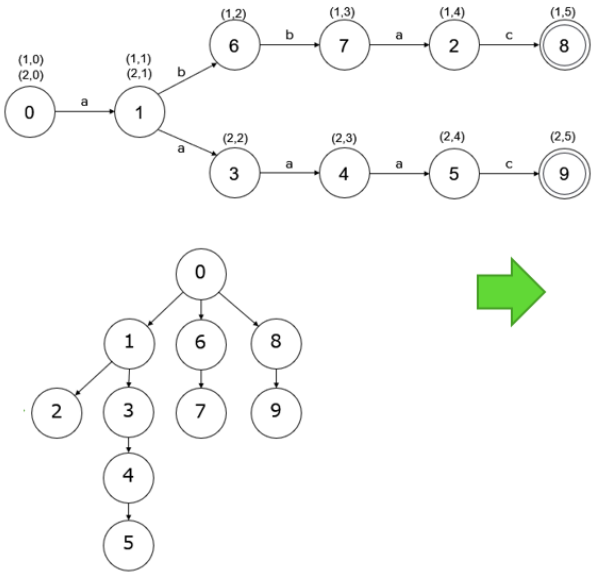
しかし、子孫ノード集合を最小値と最大値をによって表現するためには、任意の集合が常に連番であるようなノードの集合である必要がある。Allauzen のアルゴリズムで構成されたサフィックスリンクツリーは常にそうなっているわけではないため、サフィックスリンクツリーを深さ優先探索で辿り、番号付けし直すことでこれを解決する。



状態番号	$SP^{-1}(i)$
0	(0, 9)
1	(1, 5)
2	(2, 2)
3	(3, 5)
4	(4, 5)
5	(5, 5)
6	(6, 7)
7	(7, 7)
8	(8, 9)
9	(9, 9)

図 5.3: SP_{-1}の配列での実装

また、S_{-1}、$endpos$ の実装は以下のようになっている。



状態番号	endpos(i)	$S^{-1}(i)$
0	(1,0), (2,0)	1, 6, 8
1	(1,1), (2,1)	2, 3
2	(1,4)	
3	(2,2)	4
4	(2,3)	5
5	(2,4)	
6	(1,2)	7
7	(1,3)	
8	(1,5)	9
9	(2,5)	

図 5.4: S_{-1}と $endpos$ の配列での実装

5.4.2 提案アルゴリズム

ここでは、提案手法の検索アルゴリズムを示す。
検索アルゴリズムは、

1. まず最初に入力のパターン w で $Oracle(p)$ を遷移させ、受理した状態番号 $\overline{w}$ を得る。受理しなければ空集合を返す。

2. その状態番号と統合された状態に対応するテキスト p の位置に w が出現していれば出現位置の集合 $output$ に $\overline{w}$の $endpos$ を追加する。

3. $Oracle(D)$ があるパターン w を受理した場合の出現位置の候補は $\overline{w}$ だけではなく、$SP(\overline{w})$ の全てに及ぶ。よって、Algorithm 3 の 9 行目からは、テキスト集合 D の各 $j \in S^{-1}(\overline{w})$ の値に対して w が出現しているか判定している。もし出現していれば、$output$ に $SP^{-1}(j)$ の全ての $endpos$ を追加し、そうでなければ追加しない。

Algorithm 9 search$(Oracle(D), w)$

```
 1: if w recognized by Oracle(D) atstate w̄ then
 2:     x ← w̄ of Oracle(D)
 3: else return ∅                    /* empty set */
 4: for all i ∈ pair(x)     /* outer loop */ do
 5:     for all (s,t) ∈ endpos(i) do
 6:         if D[s][(t − |w| + 1)...t] = w then
 7:             output(w) ← (s,t), break outer loop
 8: for j ∈ S^{-1}(i) do
 9:     for all (s,t) ∈ endpos(j) do
10:         if D[s][(t − |w| + 1)...t] = w then
11:             for k ∈ SP_{-1}(j) do
12:                 output(w) ← output(w) ∪ endpos(j)
   return output(w)
```

このアルゴリズムは、3 章の Kato の検索アルゴリズムと同様に $\mathcal{O}(|S^{-1}(\overline{w}) \times w)$ の実行時間を要す。$\mathcal{O}(|S^{-1}(i))|$ は最悪で $\mathcal{O}(n)$ になる。しかし、$Oracle(D)$ のサフィックスリンクツリーのノード数が n であるため、ならし計算量では $\mathcal{O}(1)$ となることがわかる。
よってアルゴリズム全体でのならし計算量は $\mathcal{O}(max(w, output))$ である。

第6章　実験的評価

本章では、提案手法と部分集合構成法を用いた手法での構築と検索に要した時間を比較する。

6.1　実験結果

図 6.1: 実験結果

実験結果から、提案手法を用いた索引のほうが構築、検索の両方において効率的に行うことができるということが分かった。これは理論上も実行時間のオーダーから明らかであるため、納得の行く結果となった。

6.2　実験環境

実験に使用した環境を示す。

OS　CentOS 6.10

メモリ　64GB

CPU　Intel Core i7-6850k (6 core 3.6GHz)

テキスト　Enron mail dataset

第7章　まとめ

本稿では文字列集合に対するファクターオラクルを用いたコンパクトな全文検索索引を提案した。

コンパクトな全文検索索引を効率的に構築するため、Allauzen のアルゴリズムを拡張することで $\mathcal{O}(n)$ の実行時間を得た。

また、提案手法では内部遷移のみに注目した DFA の最小化を実行しているため、文字列集合に対して最も小さいオートマトンを構成できたと考えられる。

今回の手法は、[5] と組み合わせることでより効率的に実装できることが考えられる。、

関連図書

[1] C. Allauzen, M. Crochemore, M. Raffinot, ”Factot Oracle: a new structure for pattern matching” Proc. Of SOSFSEM’99, LNCS, Vol.1725, p.295-310, 1999.

[2] Ryoichi Kato, Osamu Watanabe, ”Substring search and repeat search using factor oracles”, Information Processing Letters, Vol.93, p269-274, 2005.

[3] 大井恒平, ”ファクターオラクルの拡張とその応用に関する研究”, 2019.

[4] A. Blumer and J. Blumer and D. Haussler, ”The smallest automaton recognizing the subwords of a text.”,Theoretical Computer Science, Vol.40, p31-55, 1985.

[5] Miroslav Balik, ”DAWG versus Suffix Array”, CIAA’02: Proceedings of the 7th international conference on Implementation and application of automata, p.233-238, 2002.

謝辞

本研究を行うにあたって、指導教員である山本博章先生には熱心なご指導ご鞭撻を賜り、ありがとうございました。また、藤原洋志先生には、ゼミ等で数々のご指摘やご助言を賜りました。ここに感謝の意を表します。