Vol.2019-AL-171 No.2
2019/1/29

右から左に構築するパラメタ化接尾辞木

藤里 法輝[1]　中島 祐人[1]　稲永 俊介[1]　坂内 英夫[1]　竹田 正幸[1]

概要：文字列の「構造」に着目したパターン照合の一種である，パラメタ化パターン照合問題を考える．本論文では，パラメタ化パターン照合問題のための索引構造の 1 つであるパラメタ化接尾辞木を，右から左にオンラインに構築するアルゴリズムを提案する．提案アルゴリズムは，$O(n(|\Pi| + \log |\Sigma|))$ 時間，$O(n)$ 領域で動作する．ただし，n は入力テキスト長，Π と Σ はパラメタ化文字列を構成する互いに素な文字集合である．

キーワード：文字列アルゴリズム，パラメタ化パターン照合，索引データ構造

1. 序論

パラメタ化パターン照合は文字列の構造に焦点を当てたパターン照合であり，Baker [1] によって導入された．Σ と Π を互いに素なアルファベットとする．$(\Sigma \cup \Pi)^*$ の要素をパラメタ化文字列，もしくは簡単のために p-文字列とよぶ．$\Sigma \cup \Pi$ 上の全単射 f を用いて，p-文字列 X を p-文字列 Y に変換できるとき，X と Y は p-**マッチ**するという．ただし，任意の $a \in \Sigma$ について，$f(a) = a$ とする．パラメタ化パターン照合問題とは，テキスト p-文字列 T とパターン p-文字列 P が与えられ，P と p-マッチする T の部分文字列の開始位置をすべて求めることである．パラメタ化パターン照合はソフトウェアのメンテナンス [1] や剽窃の検出などに応用されうる．

通常の文字列の照合問題と同様に，パラメタ化パターン照合問題を効率的に解くための索引構造が提案されている．例えば，パラメタ化ポジションヒープ [4]，構造的接尾辞木 [8]，パラメタ化接尾辞配列 [3], [5]，パラメタ化接尾辞木 [2] などが知られている．本論文では，パラメタ化接尾辞木 [2] に着目する．パラメタ化接尾辞木を用いることにより，パラメタ化パターン照合を $O(m(\log |\Pi| + \log |\Sigma|) + occ)$ 時間で行うことができる．ここで，m はパターン P の長さ，occ はパターンの出現回数である．また，テキスト長を n としたとき，パラメタ化接尾辞木の領域計算量は $O(n)$ であることが知られている．

Baker [2] は，パラメタ化接尾辞木を $O(n(|\Pi| + \log |\Sigma|))$ 時間で構築するオフラインアルゴリズムを提案した．さらに，Kosaruju [6] は，$O(n(\log |\Pi| + \log |\Sigma|))$ 時間でオフラインに構築する手法を与えた．これらのアルゴリズムは，McCreight [7] の接尾辞木構築アルゴリズムに基づいている．Shibuya [8] は，入力テキストを左から右に走査しながら，パラメタ化接尾辞木を $O(n(\log(|\Pi| + \log |\Sigma|)$ 時間でオンラインに構築するアルゴリズムを提案した．Shibuya のアルゴリズムは，Ukkonen [9] の接尾辞木構築アルゴリズムに基づいている．これらのアルゴリズムの領域計算量は，いずれも $O(n)$ である．

本論文では，入力テキストを右から左に走査しながら，パラメタ化接尾辞木を $O(n(|\Pi| + \log |\Sigma|))$ 時間，$O(n)$ 領域でオンラインに構築するアルゴリズムを提案する．この提案アルゴリズムは，Weiner [10] の接尾辞木構築アルゴリズムに基づいている．

2. 表記

文字列 t の長さを $|t|$ と表記する．文字列 $t = xyz$ に対し，x, y, z をそれぞれ t の**接頭辞**，**部分文字列**，**接尾辞**と呼ぶ．$0 \le i, j \le n$ に対し，$t[i:], t[:i], t[i:j]$ はそれぞれ位置 i から始まる t の接尾辞と位置 i で終わる t の接頭辞と位置 i から始まり位置 j で終わる t の部分文字列である．ただし，$i > j$ のとき $t[i:j] = \epsilon$ とする．文字列 t に対し，逆文字列 t^R とはとは $1 \le i \le |t|$ を満たす任意の i について $t[i] = t^R[n+1-i]$ が成り立つ文字列のことである．Σ, Π を互いに素な文字集合とする．$\Sigma \cup \Pi$ 上の文字列を p-文字列と呼ぶ．(p-) 文字列 t の反転文字列を t^R と表す．

p-マッチとは以下のように定義される関係である．

定義 1 ([1])．同じ長さの文字列 w_1 と w_2 が p-マッチするとは，ある全単射 $f : \Sigma \cup \Pi \to \Sigma \cup \Pi$ が存在し，以下の条件を満たすことである．

[1] 九州大学
Kyushu University

$$\forall c \in \Sigma, c = f(c), w_2[i] = f(w_1[i])\ (\forall i, 1 \leq i \leq |w_1|)$$

つまり，f は Σ に含まれる文字については恒等写像であるが，Π に含まれる文字についてはその限りではない．p-文字列 x, y が p-マッチするとき，を $x \approx y$ と表記する．例えば，$\Sigma = \{A, B\}, \Pi = \{x, y, z\}$ とすると，p-文字列 $X = xyzAxxxByzz$ と $Y = zxyAzzzBxyy$ について $X \approx Y$ である．なぜならば，X 上の x を z に，y を x に，z を y に変えれば 2 つの文字列は合致するためである．

次に，パラメタ化パターン照合問題を以下のように定義する．

定義 2 ([1])．パラメタ化パターン照合とは，p-文字列テキスト t と p-文字列パターン p が与えられたとき，p と p-マッチする t の部分文字列の開始位置をすべて出力することである．

例えば，$\Sigma = A, \Pi = x, y, z$ について，テキスト $T = xyzAxxxAyyzAzx$ とパターン $p = yAzz$ を入力とすると，出力するテキスト中の位置の集合は $\{3, 7\}$ である．

p-マッチを行うために，p-文字列の直前符号 $prev$ を用いる．

定義 3 ([1])．p-文字列 t の直前符号 $prev(t)$ は以下のように定義される文字列である．ただし，$1 \leq i \leq |t|$ とする．

$$prev(t)[i] = \begin{cases} t[i] & (t[i] \in \Sigma) \\ 0 & (t[i] \in \Pi \text{ and} \\ & t[j] \neq t[i] \text{ for any } 1 \leq j \leq i) \\ i - j & \text{otherwise} \end{cases}$$

任意の p-文字列 t_1 と t_2 に対して，$prev(t_1) = prev(t_2)$ が成り立つことと，t_1 と t_2 が p-マッチすることは同値である．例えば，$\Sigma = \{A, B\}, \Pi = \{x, y, z\}$ とし，$X = xyzAxxxByzz$，$Y = zxyAzzzBxyy$ とすると，$prev(X) = 000A411B771 = prev(Y)$ である．よって，今後は p-マッチの確認を直前符号化が一致するかどうかで確かめる．また，直前符号化は文字列の長さに線形時間で容易に行うことができる．

3. パラメタ化接尾辞木

この章では，パラメタ化接尾辞木の定義とその性質を示していく．

定義 4 ([2])．$\$ \in \Sigma$ を文字列の末尾のみに現れる特別な文字とする．p-文字列 t に対するパラメタ化接尾辞木 $PST(t)$ とは，$t\$$ の接尾辞を直前符号化した文字列集合を表すコンパクトトライである．

パラメタ化接尾辞木の例を図 1 に示す．

任意の $1 \leq i \leq n$ について，$t[n+1-i:]\$$ のパラメタ化接尾辞木，すなわち $\$, \ldots, prev(t[n+1-i:]\$)$ を表現するコンパクトトライを $PST^i(t)$ と表記する．

a を文字，b を文字列とする．通常の文字列に対する接尾辞木では，文字列 ab を表現する頂点が存在するならば，

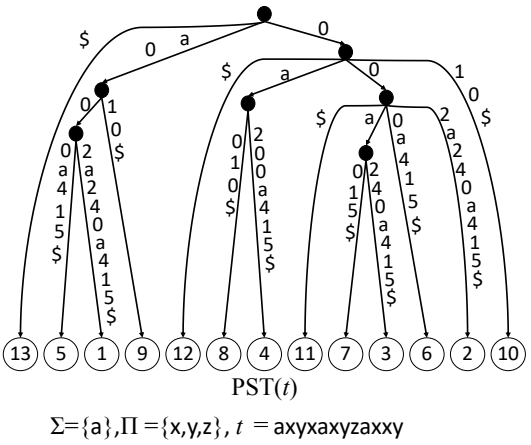

図 1 パラメタ化接尾辞木

必ずその接尾辞である b を表現する頂点が存在する．しかし，パラメタ化接尾辞木においては，$a \in \Pi$，b を p-文字列としたとき，$prev(ab)$ を表現する頂点があったとしても，$prev(b)$ を表現する頂点がが存在するとは限らない．次の補題では，このような頂点の特徴を示す．

補題 1. $a \in \Pi \cup \Sigma$, b を p-文字列とする．$PST(t)$ について考える．$PST(t)$ において $prev(ab)$ を表現する頂点が存在するが $prev(b)$ は頂点として $PST(t)$ 上に存在しないとする．このとき $PST(t)$ 上の頂点 $prev(ab)$ は $a \in \Pi$ であり，子を 2 つ持ち，子はそれぞれ $prev(ab) \cdot 0$，$prev(ab) \cdot |ab|$ を接頭辞にもつ文字列を表す．

証明. $c_1, c_2 \in (\Sigma \cup \mathcal{N})^*$ とする．$j \in \Sigma \cup \mathcal{N}$ とする．まず，$prev(ab)$ を表現する頂点は存在するが $prev(b)$ は頂点として $PST(t)$ 上に存在しないとする．そのとき $a \in \Pi$ であることを示す．背理法を用いる．$a \in \Sigma$ のとき $prev(ab) \cdot c_1$，$prev(ab) \cdot c_2$ を表す葉がそれぞれ存在する．ただし，$c_1[1] \neq c_2[2]$．このとき $prev(b) \cdot c_1$，$prev(b) \cdot c_2$ を表す葉がそれぞれ存在する．よって，仮定に矛盾する．よって，$a \in \Pi$ である．

次に，子が 2 つであることを示す．背理法を用いる．子が 3 つ以上あるとする．ハトの巣原理より $prev(ab) \cdot 0$，$prev(ab) \cdot |ab|$ を接頭辞に持たない頂点が少なくとも一つは存在する．その頂点は $prev(ab) \cdot j$ を接頭辞に持っているとする．ただし，$j \neq 0, j \neq |ab|$．残りの頂点は $prev(ab) \cdot k$ を接頭辞に持つ文字列を表しているとする．ただし，$k \neq j$．このとき，$prev(b) \cdot j$ を接頭辞に持つ頂点と $prev(b) \cdot 0$ もしくは $prev(b) \cdot k$ を表す頂点が存在する．いずれの場合も，$prev(b)$ を表す頂点が存在し，矛盾．よって，子は 2 つである．つぎに，それらの子が $prev(ab) \cdot 0$，$prev(ab) \cdot |ab|$ を接頭辞にもつ文字列であることを示す．背理法を用いて示す．少なくとも片方の頂点が $prev(ab) \cdot 0$，$prev(ab) \cdot |ab|$ を接頭辞に持たないとすると，$prev(ab) \cdot j$ を接頭辞に持っている．ただし，$j \neq 0, j \neq |ab|$．そのとき，上の同様の議

論を用いると $prev(b)$ が必ず存在し，矛盾．よって，上の補題は成り立つ．□

例えば，$t = axyxaxyzaxxy\$$ のとき $PST(t)$ は図 1 のようになる．このときに $0a$ の頂点は確かに存在するが a の頂点は存在しない．このとき 0 と 2 で枝分かれをしている．後述するパラメタ化接尾辞木の構築アルゴリズムでは，この性質に留意しながら木を構築していく．

4. パラメタ化接尾辞木の右-左オンライン構築

この章では，テキストを右から左に走査しながら，パラメタ化接尾辞木をオンラインに構築するアルゴリズムを与える．提案アルゴリズムは，基本的に Weiner の接尾辞木の構築アルゴリズムを模倣したものであるが，p-文字列と一般の文字列の差によりいくつか変更点がある．

Weiner のアルゴリズムは逆接尾辞リンクを利用することで接尾辞木を構築するが，本論文ではそれを p-文字列に拡張した p-逆接尾辞リンクを利用する．p-逆接尾辞リンクは二種類存在し，それぞれ明示的リンク，非明示的リンクと呼ばれる．それぞれを以下のように定義する．

定義 5. $PST(t) = (V, E)$ とする．v を p-文字列とする．任意の $prev(v) \in V$ と $a \in \Sigma \cup \mathcal{N}$ に対し，明示的リンク $exl(a, prev(v))$ は次のように定義される頂点である．

$$exl(a, prev(v)) = \begin{cases} a \cdot prev(v) & (\text{if } a \in \Sigma \cup \{0\} \\ & \text{and it is represented by} \\ & PST(t)) \\ prev(v[a] \cdot v) & (\text{if } a \in \mathcal{N} \\ & \text{and } prev(v)[a] = 0 \\ & \text{and it is represented by} \\ & PST(t)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

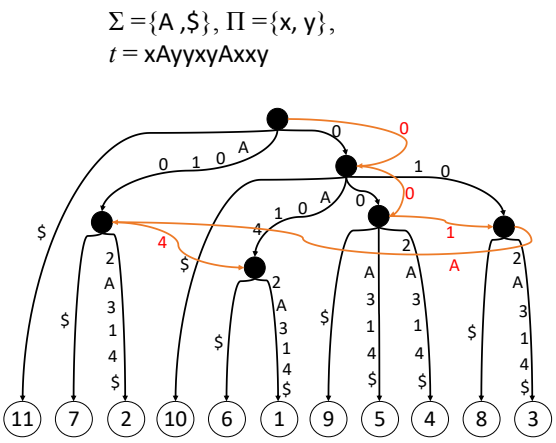

図 2　明示的リンク

図 2 に明示的リンクを示す．$\Sigma = \{A, \$\}, \Pi = \{x, y\}, t = xAyyxyAxxy$ とすると，$PST(t)$ は図 2 のようになる．例えば，$prev(Axxy) = A010$ かつ $prev(yAxxy) = 0A014$ であるため，これらの頂点間に明示的リンクが存在し，4 でラベル付けされている．

定義 6. $PST(t) = (V, E)$ とする．v を p-文字列とする．任意の頂点 $prev(v) \in V$ と $a \in \Sigma \cup \mathcal{N}$ に対し，非明示的リンク $imp(a, prev(v))$ は，$m(a, prev(v))$ を表す根からの道が $PST(t)$ 上には存在するが，頂点には表現されていないときのみ定義され，このとき $m(a, prev(v))$ の接頭辞を表す最深の頂点である．ただし，$x \in \Sigma \cup \mathcal{N}$，$y$ を p-文字列としたとき $m(x, prev(y))$ は以下のように定義される文字列である．

$$m(x, prev(y)) = \begin{cases} x \cdot prev(y) & (\text{if } a \in \Sigma \cup \{0\}) \\ prev(v[x] \cdot y) & (\text{if } a \in \mathcal{N} \\ & \text{and } prev(v)[a] = 0) \\ \text{undefined} & \text{otherwise} \end{cases}$$

また，$imp(a, prev(v))$ が表現している文字列は $m(a, prev(v))$ とする．

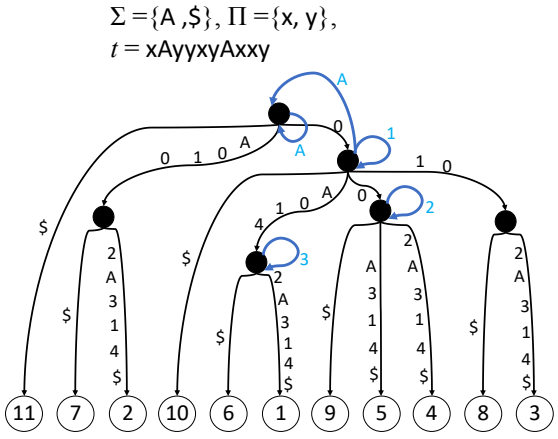

図 3　非明示的リンク

図 3 に非明示的リンクを示す．$\Sigma = A, \$, \Pi = x, y, t = xAyyxyAxxy$ とすると $PST(t)$ は図 3 のようになる．例えば，頂点 0 に対して $A0$ 頂点でないが辺上には存在するため，0 と根をラベル A で繋ぐ．

$PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞（つまり，$prev(t[n+1-i:]\$)$ を挿入した際に $PST^{i}(t)$ 上で親となる頂点）を $prev(t[n+1-i]s)$ とする．ただし，s は $t[n+2-i:]\$$ の接頭辞である．本論文の構築アルゴリズムの基本的な流れは，まず $PST^{i-1}(t)$ 上の祖先の頂点を訪問していき p-逆接尾辞リンクを見ていくことで $prev(t[n+1-i:]\$)$ の最長の接頭辞を表す頂点 $prev(t[n+1-i]s)$ を見つける．次に，$prev(t[n+1-i]s)$

が頂点であればその子に $prev(t[n+1-i:]\$)$ を挿入し，頂点でないならば新たな頂点を作り，その子として葉 $prev(t[n+1-i:]\$)$ を挿入する．

$prev(t[n+1-i]s)$ をどのようにして見つけるに際し，先頭文字の性質によりいくつかの場合分けが存在する．$t[n+1-i] \in \Sigma$ のときは以下の補題と定理を用いる．

補題 2. $t[n+1-i] \in \Sigma$ とする．$PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞を $prev(t[n+1-i]s)$（$s \in (\Sigma \cup \mathcal{N})^*$）とする．このとき，$PST^{i-1}(t)$ 上では必ず $prev(s)$ が頂点である．

証明. $u, w \in (\Sigma \cup \mathcal{N})^*$ とする．$PST^i(t)$ について考える．$PST^i(t)$ 上には仮定より必ず $prev(t[n+1-i]s)u\$$, $prev(t[n+1-i]s)w\$$ が存在する．よって，$prev(s)u\$$, $prev(s)w\$$ が存在する．当然 $PST^{i-1}(t)$ 上でも $prev(s)u\$$, $prev(s)w\$$ が存在する．よって $prev(s)$ は $PST^{i-1}(t)$ 上で必ず頂点である． □

以上の補題より $t[n+1-i] \in \Sigma$ のときは必ず $prev(s)$ が頂点に存在する．仮定より p-逆接尾辞リンクが必ず存在する．つまり，$t[n+1-i] \in \Sigma$ のときにはアルゴリズムは以下のようになる．補題 2 より，最初に見つけたラベル $t[n+1-i]$ の p-逆接尾辞リンクを持つ頂点が $prev(s)$ である．そのとき見つけた p-逆接尾辞リンクをたどり，明示的リンクならばその子として $prev(t[n+1-i:]\$)$ を挿入する．非明示的リンクならば，その子に新たな頂点を作り新たに作られた頂点の子孫として $prev(t[n+1-i:]\$)$ を挿入する．そして，$prev(s)$ を表現する頂点とその子孫のラベル $t[n+1-i]$ の p-逆接尾辞リンクを削除し，$prev(t[n+1-i:]\$)$ の親とつなぎ直す．

上では，$t[n+1-i] \in \Sigma$ のとき $prev(t[n+1-i:]\$)$ を表す葉を PST^{i-1} にどう挿入するかを述べた．続いて，$t[n+1-i] \in \Pi$ の際にはどのように葉を挿入するかを述べていく．

$t[n+1-i] \in \Pi$ かつ $t[n+1-i]$ が s 中に存在するときは，以下の補題を利用する．

補題 3. $t[n+1-i] \in \Pi$ とする．$PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞を $prev(t[n+1-i]s)$（$s \in (\Sigma \cup \mathcal{N})^*$）とする．$t[n+1-i]$ が s 中に存在するとする．このとき，$PST^{i-1}(t)$ 上では必ず $prev(s)$ が頂点である．

証明. $u_1, w_1, u_2, w_2 \in (\Sigma \cup \mathcal{N})^*$ とする．$t[n+1-i]$ と同じ文字は直近で k 文字後ろに出ているものとする．$prev(t[n+1-i]s) = 0 \cdot u_1 \cdot k \cdot u_2$ とする．$PST^i(t)$ について考える．$PST^i(t)$ 上には仮定より必ず $prev(t[n+1-i]s)w_1\$ = 0 \cdot u_1 \cdot k \cdot u_2 \cdot w_1\$$, $prev(t[n+1-i]s)w_2\$ = 0 \cdot u_1 \cdot k \cdot u_2 \cdot w_2\$$ が存在する．$prev(s) = 0 \cdot u_1[2:] \cdot 0 \cdot u_2$ であるため，$prev(s)w_1\$ = 0 \cdot u_1[2:] \cdot 0 \cdot u_2 \cdot w_1\$$, $prev(s)w_2\$ = 0 \cdot u_1[2:] \cdot 0 \cdot u_2 \cdot w_2\$$ が存在する．当然 $PST^{i-1}(t)$ 上でも $prev(s)w_1\$ = 0 \cdot u_1[2:] \cdot 0 \cdot u_2 \cdot w_1\$$, $prev(s)w_2\$ = 0 \cdot u_1[2:] \cdot 0 \cdot u_2 \cdot w_2\$$ が存在する．よって $0 \cdot u_1[2:] \cdot 0 \cdot u_2$ つまり $prev(s)$ は $PST^{i-1}(t)$ 上で必ず頂点である． □

つまり，$t[n+1-i] \in \Sigma$ のときと同様に，$prev(s)$ は必ず頂点である．よって，先ほどと同様の手法によって $prev(t[n+1-i:]\$)$ を挿入することができる．$t[n+1-i]$ と同じ文字が k 文字前に出ているとする．$prev(t[n+2-i:]\$)$ の頂点の祖先を深い順に走査していく．補題 3 より，最初に見つけたラベル k の p-逆接尾辞リンクを持つ頂点は $prev(s)$ である．そのリンクを辿り，明示的リンクならその子孫に $prev(t[n+1-i:]\$)$ を挿入する．非明示的リンクならばその子として頂点を作り，さらにその子として $prev(t[n+1-i:]\$)$ を表す葉を挿入すれば良い．そして，$prev(s)$ を表現する頂点とその子孫のラベル k の p-逆接尾辞リンクを削除し，$prev(t[n+1-i:]\$)$ の親とつなぎ直す．

次に，$t[n+1-i] \in \Pi$ かつ $t[n+1-i]$ が s 中に存在しないときについて考える．このとき，必ず $prev(s)$ が頂点であるとは限らない．$prev(s)$ が頂点でない場合について，以下の補題で議論する．

補題 4. $PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞を $prev(t[n+1-i]s)$（$s \in (\Sigma \cup \mathcal{N})^*$）とする．$PST^i$ 上のダミー頂点の個数が PST^{i-1} より増えるときは次の 2 つの場合のみである．

(1) $prev(t[n+1-i]s)$ が PST^{i-1} の上で頂点ではなく，$prev(t[n+1-i]s) \cdot 0$ を表現する頂点が存在し，$prev(t[n+1-i:]\$)[|t[n+1-i]s|] = |s|$ である．

(2) $prev(t[n+1-i]s)$ が PST^{i-1} の上で頂点ではなく，$prev(t[n+1-i]s) \cdot |s|$ を表現する頂点が存在し，$prev(t[n+1-i:]\$)[|t[n+1-i]s|] = 0$ である．

証明. 補題 1 より自明． □

$k = |0 \cdot prev(s)|$ と定義する．上の補題より，次の二つの場合では $prev(s)$ は頂点ではない．両方の場合を通して $0 \cdot prev(s)$ は $PST^{i-1}(t)$ 上で表現されていない．一つめの場合では，$prev(t[n+1-i:]\$) = 0 \cdot prev(s) \cdot k \cdot prev(t[n+1-i:]\$)[k+1:]$ であり，$PST^{i-1}(t)$ 上に $0 \cdot prev(s) \cdot 0 \cdot r$（ただし，$r \in (\Sigma \cup \mathcal{N})^*$）が存在するときである．これを場合 1 と呼ぶ．二つめの場合では $prev(t[n+1-i:]\$) = 0 \cdot prev(s) \cdot 0 \cdot prev(t)[k+1:]$ であり，$PST^{i-1}(t)$ 上に $0 \cdot prev(s) \cdot k \cdot u$（ただし，$u \in (\Sigma \cup \mathcal{N})^*$）が存在するときである．これを場合 2 と呼ぶ．場合 1，場合 2 の構築アルゴリズムについてそれぞれ述べていく．

場合 1 の際，次の補題が成り立つ．

補題 5. s は p-文字列とする．$PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞を $prev(t[n+1-i]s)$ とする．$prev(t[n+1-i]s)$ が PST^{i-1} の上で頂点ではな

く，なおかつ $prev(t[n+1-i]s)\cdot 0$ を接頭辞に持つ文字列を表現する頂点が存在するとする．そのとき，$prev(t[n-i:]\$)$ を表現する頂点の祖先のうち，文字列深さ $t[n+1-i]s$ 以上の頂点が必ず存在し，その中で一番深さの浅い頂点には 0 のラベルの p-逆接尾辞リンクをもつ頂点が必ず存在する．

証明． $w, z_1, z_2 \in \Sigma \cup \mathcal{N}$ とする．ただし，$z_1[1] \neq z_2[1]$ とする．$PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞を $prev(t[n+1-i]s)$ とする．$prev(t[n+1-i]s)$ が PST^{i-1} の上で頂点ではなく，なおかつ $prev(t[n+1-i]s)\cdot 0$ を接頭辞に持つ文字列を表現する頂点が存在するとする．その後，$prev(n+1-i:)$ を表現する葉を挿入し $PST^i(t)$ を作るとする．仮定より $prev(t[n+1-i]s)$ は二つの枝分かれを持ちそれぞれ 0 と $|s|$ で枝分かれしており，なおかつ $prev(s)$ を表現する頂点が存在しないため．かならず $prev(t[n+1-i]s)\cdot|s|\cdot w\cdot z_1, prev(t[n+1-i]s)\cdot 0\cdot w\cdot z_1$ を表現する葉がそれぞれ存在する．よって，$prev(t[n+1-i]s)\cdot 0\cdot w$ を表現する頂点が必ず存在し，その親は仮定より文字列深さ $t[n+1-i]s$ 未満である．また，その頂点はラベル 0 のリンクを持ち $PST^{i-1}(t)$ 上でも存在する． □

つまり，アルゴリズムは以下のようになる．$t[n+1-i]$ と同じ文字が直近で k 文字前に出現しているとする．$i+1$ 番目の葉から根へと走査していく．文字列深さ k 以下の頂点では k のラベルを持つ p-逆接尾辞リンクを探す．文字列深さ k 未満の頂点で初めてラベルが 0 の p-逆接尾辞リンクを見つけたとき，その下の頂点にもラベルが 0 の p-逆接尾辞リンクが存在するかを探す．もし，存在するならば，最初に見つけたラベル 0 の p-逆接尾辞リンクを辿りその子に文字列深さが k となる頂点を作る．そして，新たに作った頂点の子として $prev(t[n+1-i:]\$)$ を表現する頂点を挿入する．もし，存在しないならば，最初に見つけたラベル 0 の p-逆接尾辞リンクを辿りその子として $prev(t[n+1-i:]\$)$ を表現する頂点を挿入する．

場合 2 の際，次の補題が成り立つ．

補題 6. s は p-文字列とする．$PST^{i-1}(t)$ 上で表現されている $prev(t[n+1-i:]\$)$ の最長の接頭辞を $prev(t[n+1-i]s)$ とする．$prev(t[n+1-i]s)$ が PST^{i-1} の上で頂点ではなく，なおかつ $prev(t[n+1-i]s)\cdot|s|$ を接頭辞に持つ文字列を表現する頂点が存在するとする．そのとき，$prev(t[n-i:]\$)$ を表現する頂点の祖先のうち，文字列深さ $t[n+1-i]s$ 以下の頂点が必ず存在し，その中で一番深さの浅い頂点が持っている値が最大の p-逆接尾辞リンクのラベルは $|s|$ である．

証明． 補題 5 と同様の証明を用いれば $prev(t[n-i:]\$)$ を表現する頂点の祖先のうち，文字列深さ $t[n+1-i]s$ 以下の頂点が必ず存在し，その中で一番深さの浅い頂点はラベルは $|s|$ の p-逆接尾辞リンクを持っていることは証明できる．ラベル $|s|$ がその中で最大であることを示す．u を 1 文字以上の p-文字列とする．背理法を用いて示す $|s| < |su_1u_2|$ となるリンクが存在するとき，$prev(t[n+1-i]su)\cdot|su|$ が $PST^{i-1}(t)$ 上のどこかのパスもしくは頂点で表現されている．また，仮定より $t[n+1-i]$ と同じ文字は $|s|$ 文字以上前に出ているため，$prev(t[n+1-i:]\$) = prev(t[n+1-i]su[1]u[2:j])$ を接頭辞に持つ．ただし，j は $2 \leq j \leq |u|$ を満たす．よって，$prev(t[n+1-i]su[1]pr(u[2:j]))$ で枝分かれを起こし矛盾． □

つまり，アルゴリズムは以下のようになる．$t[n+1-i]$ が初出現の文字であるとする．$i+1$ 番目の葉から根へと走査していく．0 のラベルを持つ p-逆接尾辞リンクを探す．そうして初めてラベル 0 の p-逆接尾辞リンクを見つけた頂点を v_1 その一歩前で走査した頂点を v_2 とする．頂点 v_2 のラベル最大の p-逆接尾辞リンクを確認する．そのラベルを l とする．$|v_1| \leq l$ ならば v_1 のラベル 0 の p-逆接尾辞リンクを辿った先の頂点の子どもに文字列深さ l の頂点を作りその子として $prev(t[n+1-i:]\$)$ を表現する頂点を挿入する．

以上より，p-逆接尾辞リンクは全ての頂点において計算されている際に，$prev(t[n+1-i:]\$)$ を $PST^{i-1}(t)$ にどう挿入するかを述べた．続いて，新たに作られた頂点，辺を指し示す p-逆接尾辞リンクと新たに作られた頂点自身が指し示す p-逆接尾辞リンクを引かなければならない．p-逆接尾辞リンクの挿入について述べる．次の補題を用いてリンクを新たに導入する．

補題 7. $k \in \Sigma \cup \mathcal{N}$ $PST(t) = (V, E)$ とする．$prev(v_1v_2), prev(v_1) \in V$ とする．$prev(v_1v_2)$ がラベル k の p-逆接尾辞リンクを持つならば $prev(v_1)$ はラベル $f(|v_1|, k)$ の p-逆接尾辞リンクをもつ．ただし，$a \in \Sigma \cup \mathcal{N}$，$b \in \mathcal{N}$ としたとき関数 $f(a, b)$ は以下のように定義される．

$$f(a,b) = \begin{cases} a & (\text{if } b \in \mathcal{N}, b \leq a \text{ or } b \in \Sigma) \\ 0 & (\text{otherwise}) \end{cases}$$

証明． k について場合分けする．$k \in \Sigma$ のとき $prev(v_1v_2)$ がラベル k の p-逆接尾辞リンクを持っているならば，定義より $k\cdot prev(v_1v_2)$ が $PST(t)$ 上で表現されている．よって，$k\cdot prev(v_1)$ が $PST(t)$ 上で表現されている．よって，このとき $prev(v_1)$ は p-逆接尾辞リンク $f(|v_1|, k)$ をもつ．$k \in \mathcal{N}, k \leq |v|$ のとき，$prev(v_1v_2)$ がラベル k の p-逆接尾辞リンクを持っているならば，定義より $0\cdot prev(v_1v_2)[1:k]\cdot k\cdot prev(v_1v_2)[k+2:]$ が $PST(t)$ 上で表現されている．$k \leq |v|$ より，$0\cdot prev(v_1)[1:k]\cdot k\cdot prev(v_1)[k+2:]$ が $PST(t)$ 上で表現されている．よって，このとき $prev(v_1)$ は p-逆接尾辞リンク $f(|v_1|, k)$ をもつ．$k \in \mathcal{N}, k > |v|$ のとき，$prev(v_1v_2)$ がラベル k の p-逆接尾辞リンクを持っているならば，定義より $0\cdot prev(v_1v_2)[1:k]\cdot k\cdot prev(v_1v_2)[k+2:]$ が $PST(t)$ 上で表現されている．$k > |v|$ より，$0\cdot prev(v_1)$ が $PST(t)$ 上で表現されている．よって，このとき $prev(v_1)$

は p-逆接尾辞リンク $f(|v_1|, k)$ をもつ．以上より題意は成り立つ． □

補題 8. $PST(t) = (V, E)$ とする．$prev(v) \in V$ とする．このとき，$prev(v)$ の子孫の葉が持っている p-逆接尾辞リンクのラベルの集合を L とする．とする．このとき，$prev(v)$ のもつ p-逆接尾辞リンクの任意のラベル $k_1 \in \Sigma \cup \mathcal{N}$ に対し．$k_1 = f(|v|, k_2)$ を満たすようなラベル k_2 の p-逆接尾辞リンクを持つ葉が必ず存在する．

ただし，$a \in \Sigma \cup \mathcal{N}$，$b \in \mathcal{N}$ としたとき関数 $f(a, b)$ は以下のように定義される．

$$f(a,b) = \begin{cases} a & (\text{if } b \in \mathcal{N}, b \leq a \text{ or } b \in \Sigma) \\ 0 & (\text{otherwise}) \end{cases}$$

証明． w_1 を p-文字列とする．$PST(t)$ 上の任意の頂点 $prev(v)$ のもつ任意の p-逆接尾辞リンクのラベル $k \in \Sigma \cup \mathcal{N}$ について考える．k の文字の種類で場合分けする．$k \in \Sigma \cup 0$ のとき，定義より $k \cdot prev(v)$ が必ず $PST(t)$ 上に表現されている．よって，$prev(k \cdot v \cdot w_1)$ を表現する葉が必ず存在する．よって，$prev(v \cdot w_1)$ を表現する葉が存在する．よって，$prev(v \cdot w_1)$ は必ず，ラベル k の p-逆接尾辞リンクを持つ．$k \in \mathcal{N}$ かつ $k \neq 0$ のとき，定義より $prev(v[k] \cdot v)$ が必ず $PST(t)$ 上に表現されている．よって，$prev(v[k] \cdot v \cdot w_1)$ を表現する葉が必ず存在する．よって，$prev(v \cdot w_1)$ を表現する葉が存在する．よって，$prev(v \cdot w_1)$ は必ず，ラベル k の p-逆接尾辞リンクを持つ． □

よって，新たに作られた頂点自身が指し示すリンクは次のように引かれる．自身の子を探しその子孫の持つ p-逆接尾辞リンクを自身の頂点の文字列深さより小さいラベルの p-逆接尾辞リンクを作る．文字列深さよりラベルが大きいものがあった場合 0 を作る．このとき素朴に各ラベルのリンク先の頂点を探すと，1 つのラベルあたり $O(|\Pi|)$ 時間かかってしまう．このことを次に示す．

補題 9. $PST(t)$ 上の頂点 $prev(v_2)$ とその親 $prev(v_1)$ にがそれぞれ $PST(t)$ 上の頂点 $prev(v_2')$ とその親 $prev(v_1')$ と p-逆接尾辞リンクでで結ばれているとする．ただし，それぞれのラベルは k と $f(k, v_1)$ とする．そのとき，$prev(v_2')$ と $prev(v_1')$ の間には最大 $|\Pi|$ 個の頂点がありうる．ただし，$a \in \Sigma \cup \mathcal{N}$，$b \in \mathcal{N}$ としたとき関数 $f(a, b)$ は以下のように定義される．

$$f(a,b) = \begin{cases} a & (\text{if } b \in \mathcal{N}, b \leq a \text{ or } b \in \Sigma) \\ 0 & (\text{otherwise}) \end{cases}$$

証明． 補題 1 より自明． □

つまり，ラベル $a \in \Sigma \cup \mathcal{N}$ のリンクを作ろうとすれば，その子と親でそれぞれラベル $a, f(a, b)$ のリンクを辿りその中で文字列深さが最適な頂点を選ばなければならない．新たな p-逆接尾辞リンクを挿入するごとにその操作が必要である．

以上が構築アルゴリズムである．以降，本論文のアルゴリズムの構築時間について述べる．構築時間を解析するために以下の補題を示す．

補題 10. $1 \leq i \leq j \leq n$ とする．s_0 を p-文字列，s_1 を長さ 1 以上の p-文字列とする．また $PST^i(t)$ 上で $prev(t[n+1-i:]\$)$ を表現する葉の親を $prev(v_i)$ とする．このとき次の不等式が成り立つ．

$$|v_{i+1}| \leq |v_i| + 1$$

証明． この命題が成り立たないと仮定する．$prev(v_{i+1})$ は仮定より，v_i より 2 文字以上長い．このとき必ず $prev(t[n+1-i:]\$)$ の親で深さ $|v_i| + 2$ 以上の場所に頂点ができてしまうため，矛盾． □

補題 11. 本アルゴリズムでの $prev(t[n+2-i:]\$)$ を挿入する際に走査した頂点の数を c_i とする．$PST^i(t)$ 上で $prev(t[n-i+1:])$ を表す葉の辺のうち 0 となっている場所でなおかつ直前に頂点となっていない場所の総数を m_i とおく．$PST^i(t)$ 上で $prev(t[n-i+1:])$ を表す葉の頂点深さを d_{i+1} とおく．このとき，ある定数 k について，次式が成り立つ．

$$d_{i+1} + m_{i+1} \leq d_i + m_i - c_i + k$$

証明． 補題 10 より，明らか． □

上の補題より，葉を挿入する際に走査した頂点の数は $O(n)$ となる．

つぎに，非明示的リンク の総数について述べる．

補題 12. 非明示的リンクの総数は高々 $3n - 2$ である．

証明． $PST(t)$ 上で頂点が表現してる文字列を直前符号化する前の文字列の集合を M，非明示的リンクが表現している文字列の直前符号化する前の文字列集合を Imp とする．$t\$^R$ の接尾辞を直前符号化した文字列の集合を表現するトライ T を考える．T 上で $prev(y^R)$ をプロットする．ただし，$y \in Imp$ を満たす．このとき T 上で次のことが成り立つ．$prev(y^R)$ の親の表現する文字列を $prev(p)$ とすると $p^R \in M$ である．以上より次のことが考えられる．任意の p-文字列集合 X に対し，$X^R = \{x^R \mid x \in X\}$，$prev(X) = \{prev(x) \mid x \in X\}$ とそれぞれ定義すると $prev(M^R)$ の要素をすべて，T 上の枝分かれする頂点に配置したとすると $|Imp| = 2n - 2$ になり，要素をすべて，T 上の枝分かれしない頂点に配置したとすると $|Imp| = n$ になる．つまり，両方の値を合計した $3n - 2$ より $|Imp|$ は小さい． □

以上より次のことが言える．走査するノードの数，挿入するノードと辺の数の合計はそれぞれ線形であり，また，それぞれの操作には $O(\log(|\Pi| + |\Sigma|))$ 時間かかる．挿入す

る p-逆接尾辞リンクの総数は入力文字列長に線形である．そのそれぞれを木に挿入するのに $O(\log(|\Pi| + |\Sigma|))$ 時間かかる．よって，次の定理が成り立つ．

定理 1. 長さ n の p-文字列 t を右から左に走査しながら，パラメタ化接尾辞木 $PST(t)$ をオンラインに $O(n(|\Pi| + \log|\Sigma|))$ 時間，$O(n)$ 領域で構築できる．

5. まとめと今後の課題

本論文ではパラメタ化パターン照合を高速に行うパラメタ化接尾辞木を構築する新たなアルゴリズムを提案した．提案手法は，入力テキストを右から左に走査し，$O(n(|\Pi| + \log|\Sigma|))$ 時間，$O(n)$ 領域で動作する．

Shibuya のアルゴリズム [8] は，入力テキストを左から右に走査し，パラメタ化接尾辞木を $O(n\log(|\Pi| + |\Sigma|))$ 時間，$O(n)$ 領域でオンラインに構築する．本論文で提案したアルゴリズムを改良し，入力テキストを右から左に走査しながら $O(n\log(|\Pi| + |\Sigma|))$ 時間でパラメタ化接尾辞木を構築することが今後の課題である．

参考文献

[1] B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *STOC 1993*, pages 71–80, 1993.
[2] B. S. Baker. Parameterized pattern matching: Algorithms and applications. *J. Comput. Syst. Sci.*, 52(1):28–42, 1996.
[3] S. Deguchi, F. Higashijima, H. Bannai, S. Inenaga, and M. Takeda. Parameterized suffix arrays for binary strings. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2008*, pages 84–94, Czech Technical University in Prague, Czech Republic, 2008.
[4] Diptarama, T. Katsura, Y. Otomo, K. Narisawa, and A. Shinohara. Position heaps for parameterized strings. In *Proc. CPM 2017*, pages 8:1–8:13, 2017.
[5] T. I, S. Deguchi, H. Bannai, S. Inenaga, and M. Takeda. Lightweight parameterized suffix array construction. In *Proc. IWOCA 2009*, pages 312–323, 06 2009.
[6] S. R. Kosaraju. Faster algorithms for the construction of parameterized suffix trees (preliminary version). In *FOCS 1995*, pages 631–637, 1995.
[7] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of ACM*, 23(2):262–272, 1976.
[8] T. Shibuya. Generalization of a suffix tree for RNA structural pattern matching. *Algorithmica*, 39(1):1–19, 2004.
[9] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
[10] P. Weiner. Linear pattern-matching algorithms. In *Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory*, pages 1–11, 1973.