社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コンパクト非巡回語グラフに基づく連長圧縮 Burrows-Wheeler 変換の効率良い構築

須江　瑞樹† 　小林　靖明†† 　有村　博紀†† 　中島　祐人††† 稲永　俊介†††

† 北海道大学 大学院情報科学院 情報科学専攻
†† 北海道大学 大学院情報科学研究院
††† 九州大学 大学院システム情報科学研究院
E-mail: †xxx@elms.hokudai.ac.jp, ††{koba,arim}@ist.hokudai.ac.jp, †††nakashima.yuto.003@m.kyushu-u.ac.jp,
††††inenaga@inf.kyushu-u.ac.jp

あらまし　本稿では，反復テキストに対する代表的な圧縮索引であるコンパクト非巡回有向語グラフ（CDAWG）と連長圧縮 Burrows-Wheeler 変換（Run-length BWT, 連長圧縮 BWT）の間の効率良い変換について考察する．主結果として，入力として与えられた長さ n の読み取り専用テキストとサイズ e の CDAWG から，サイズ r の連長圧縮 BWT を $O(e)$ 語の作業領域と時間で構築する効率良いアルゴリズムを与える．これは，入出力の圧縮索引長の和を超えないメモリと時間を使って，CDAWG を連長圧縮 BWT に変換する初めてのアルゴリズムである．

キーワード　データ構造，テキスト索引，接尾辞木，CDAWG, 連長圧縮 BWT

1. はじめに

1.1 研究の背景

文字列アルゴリズム分野において 2000 年代半ばから，反復テキストに対する圧縮テキスト索引の研究が盛んである．ここに，**圧縮テキスト索引** [14], [15] とは, 圧縮テキストのサイズに比例した領域で格納可能で, 通常のテキスト索引と同等以上の計算時間で検索ができる索引構造である．一方，**反復テキスト** (repetitive text) [13] とは，多くの繰り返しを含むような文字列をいい, 反復性を考慮することで, そのテキストがもつ情報エントロピーよりも小さなサイズ（これを反復パラメータ[1] という）で圧縮可能である．反復テキストに対する代表的な圧縮索引として，辞書式圧縮である LZ 圧縮索引 [9] や，有限状態オートマトンを用いたコンパクト非巡回有向語グラフ (CDAWG) [3]，BWT 後の文字列を連長圧縮して得られる連長圧縮 Burrows-Wheeler 変換 (Run-length BWT, 連長圧縮 BWT) [7], [11] 等が提案されている．

1.2 研究目的

本稿では，圧縮テキスト索引間の効率良い変換について考察する．とくに入力として CDAWG を受け取り，それを解凍することなく，出力として連長圧縮 BWT に直接に変換する効率良いアルゴリズムを考察する．とくに効率性の尺度として，入力と出力の索引サイズの和に比例した時間と領域で計算する**圧縮変換** (compressed transformation) の実現を目標とする．

既存手法として，アルファベット $\Sigma = [1, \sigma]$ 上の長さ n のテキストから RL-BWT を構築するには，最初にテキストから通常の BWT を線形時間で求め [12]，さらに連長圧縮を行う直接的な手法が考えられる．ただし，これは $O(n)$ 時間と $O(n \log \sigma)$ ビット領域を要する．入力である圧縮表現のサイズに比例した時間と作業領域で実現可能かどうかは自明ではない問題である．

1.3 主結果

本稿では，入力として与えられた長さ n のテキストとサイズ e の CDAWG から，サイズ $r \leq e$ の連長圧縮 BWT を $O(e + r) = O(e)$ 語作業領域と時間で構築する効率良いアルゴリズムを提案する．我々の知る限り，これは，入出力の圧縮索引長の和を超えない作業領域と時間を使って，CDAWG を連長圧縮 BWT に変換する初めてのアルゴリズムである．

1.4 結果の詳細

次の二つのポイントに関して，いくつかの技術を開発することで提案アルゴリズムを実現している．第一の技術ポイントは，**接尾辞木上の動的計画法の模倣**である．CDAWG の特徴づけの一つに，接尾辞木の同型な部分グラフを繰り返し併合することで CDAWG が得られることが知られている．一方，後で示すように，T の連長圧縮 BWT はノード数 n をもつ T の接尾辞木上の動的計画法を用いて，$O(n)$ 語領域と時間で計算可能である (節 3.)．そこで T の接尾辞木上の動的計画法を CDAWG 上で模倣する戦略をとる．ただし，接尾辞木は $O(n)$ 個のノードと辺をもち，CDAWG は $O(\mu)$ 個のノードと $O(e)$ 本の辺しかもたないので,[2] そのままでは所望の時間と領域を達成できない．そ

(注1)：本稿で考察する CDAWG のサイズ e と連長圧縮 BWT のサイズ r の他, LZ 圧縮のサイズ z，最小文法サイズ g，最小双方向マクロスキームのサイズ b, 最小文字列 アトラクタのサイズ γ 等の反復パラメータが知られている [13].

(注2)：ここに，μ と e は，それぞれ，T の**極大反復数** (maximal repeat)，および，**左拡張と右拡張の総数**である．正確な定義は，節 2.4 を参照されたい．

こで，次の技術を用いる．

第二の技術ポイントは，**第二種辺がもつ Weiner リンク情報の利用**である．CDAWG においては根から各ノードへの最長パス全体が最長パス木と呼ばれる根付き木 LPT をなし，これより，CDAWG の有向辺を，LPT に属する第一種辺とそれ以外の第二種辺に分類できることに着目する．探索する CDAWG の部分空間として，LPT の葉に高々深さ 1 の第二種辺を追加して拡張した**拡張最長パス木** LPT_+ を導入し（これは実際には DAG として表現される），この上で有向辺を辿りながら，動的計画法を行う．

この際に，元々含まれる第一種辺のみからなるパス（これは最長パスに他ならない）に対しては，接尾辞木の場合と同様に再帰を用いた動的計画法を行う．一方，末端の第二種辺 f については，各ノード v の**部分木の葉数** $NL(v)$ と第二種辺の **Weiner 文字**と呼ばれる文字 $WC(f)$ による f に関連するパスがもつ連長圧縮 BWT の特徴づけを与える (節 4.5)．この特徴づけに基づき，あらかじめ $NL(v)$ の前処理 (節 4.3) と $WC(f)$ の前処理 (節 4.4) を行い，それらの情報を合成することで，第二種辺の解を**子への再帰を行わずに**直ちに計算することで，提案アルゴリズムは所望の領域量と時間量を達成している．

1.5 関 連 研 究

テキスト索引 (text index) とは，入力テキスト T を前処理して，構築後に任意に与えられるパターン照合演算を効率よく実現するデータ構造である．以下では，文字数 σ のアルファベット Σ と，長さ n のテキスト，長さ m のパターンを仮定する．

圧縮索引の一つである CDAWG [3] は，高々 e 個の辺をもつ DAG 状のテキスト索引である (節 2.4)．CDAWG 自身は，$O(e)$ 語領域で格納可能であり，$O(n)$ 時間でオフラインまたはオンラインで構築できるが [3], [8]，検索時に $n \log \sigma$ ビット領域のテキストを必要とする．

自己索引 (self-index) [13] とは，テキスト索引や圧縮索引と異なり，検索時にテキストを必要とせず，T よりも小さな領域で格納できるテキスト索引をいう．テキスト T の **Burrows-Wheeler 変換** (BWT) [4] は，T の巡回の辞書順で文字を並べ替えて得られる長さ n の文字列である (節 2.5)．BWT は，接尾辞配列と密接な関係をもち，BWT から元の文字列を線形時間で復元できる．FM-index [5], [6] は，BWT に基づく自己索引であり，長さ n のテキストを $nH_k + o(n)$ ビット領域で格納し，$\sigma = O(\mathrm{polylog}(n))$ のとき，長さ m のパターンの存在と出現回数を $O(m)$ 時間で行える．ここで，H_k は k 次経験エントロピーである．

反復テキストに適応した自己索引 [14] として，次の連長圧縮 BWT [4], [7], [11] と線形領域 CDAWG [18] 等がある.[3] **連長圧縮 BWT**(run-length BWT, RL-BWT) [7], [11] は，多くのくり返しを含むテキストに適応した自己索引である (節 2.5)．これは，文字列である BWT に連長圧縮符号化 (run-length encoding) を適用して得られ，反復文字列に対して領域を圧縮できる．

(注3)：繰り返しテキストに適応した圧縮手法として，解説 [13], [14] には，CDAWG と BWT の他にも Lenpel-Ziv 分解，block tree，string attractor, macro scheme，grammer などについて，それぞれの圧縮尺度の関係と索引が提案されている．

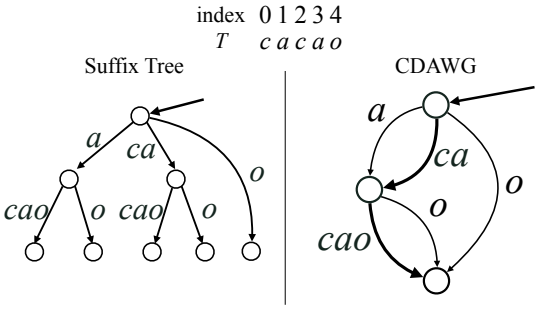

図 1: テキスト $T = cacao$ に対する接尾辞木 (左) と CDAWG(右).

RL-BWT は，BWT 中の**連の数** r に対して，$r(\log n + \log \sigma)$ ビット領域で表せる．関連したテキスト索引である RLFM-index [7], [11] は，長さ n のテキストと長さ m のパターンの occ 個の出現回数と出現位置を，$O(r)$ 語の領域を用いて，$O(m \log \log(\sigma + n/r) + occ \log \log(n/r))$ 時間で実現する．

圧縮索引間の相互変換として，パラメータと索引間の変換に関する既存結果と本稿の結果について表 1 にまとめた．これより，LZ77 から連長圧縮 BWT [9] と SLP から LZ78 [1] の変換に関しては圧縮サイズでの領域と時間で変換可能である．一方で，RL-BWT と CDAWG の間の相互変換について，我々の知る限り，どちらの変換方向についても $o(n)$ 領域または $o(n)$ 時間に関する変換は報告されていない．

2. 準　　備

非負整数全体を $\mathbb{N} = \{0, 1, \dots\}$ で，正整数全体を $\mathbb{N}^+ = \mathbb{N} \backslash \{0\}$ で表す．$\mathbb{N}$ の閉区間を $[i, j] = \{i, i+1, \dots, j\}$ で，半開区間を $[i, j) := [i, j] \setminus \{j\}$ で表す．未定義値を $\perp$ で表す．

2.1 文　字　列

全順序 $\leq_\Sigma$ をもつ文字数 σ の**アルファベット**を $\Sigma = \{1, \dots, \sigma\}$ とする．Σ 上の長さ n の任意の**文字列** (string) を $s = s[0, n) = s[0] \cdots s[n-1]$ とし，その長さを $|s| = n$ で，任意の $0 \leq i \leq j < n$ に対して，s の第 i 番目の文字を $s[i]$，第 i 番目から j 番目までの連続部分列を $s[i, j]$ で表す．長さ 0 の空文字列を ε で表す．Σ 上の長さ 0 以上と長さ 1 以上の文字列すべての集合を，それぞれ，Σ^* と $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ で表す．任意の文字列 s に対して，ある $x, y, z \in \Sigma^*$ に対して $s = xyz$ ならば，それぞれ x, y, z を s の**接頭辞** (prefix), **部分文字列** (substring), **接尾辞** (suffix) と呼ぶ．また，$\mathit{Substr}(s), \mathit{Suffix}(s) \subseteq \Sigma^*$ で，それぞれ文字列 s の**部分文字列全体**と**接尾辞全体**の集合を表す．文字列 s が文字列 t の**接尾辞である関係**を，$t \sqsupseteq^{\mathrm{suf}} s$ で表す．

Σ 上の長さ n の**テキスト** (text) T は，任意の文字列 $T = T[0, n) = T[0] \cdots T[n-1] \in \Sigma^n$ である．ここに，T は，その中に一度しか出現しない記号 $T[n-1] \in \Sigma$（**終端文字**と呼ぶ）で終端すると仮定する.[4] テキストの各添字 $i \in [0, n]$ に対して，T の**第 i 番目の接尾辞**を $T_i := T[i, n)$ と定める．ここに，$T_0 = T$ と $T_n = \varepsilon$ である．以降の節では，T **の接尾辞全ての集合** $\mathit{Suffix}(T) = \{T_n, \dots, T_0\}$ を格納する索引構造を導入する．

(注4)：節 2.5 では，任意の文字に対して条件 $T[n-1] \leq_\Sigma c$ を仮定する．

表 1: 圧縮変換の計算量に関する先行研究と本研究のまとめ. 主なパラメータは, 次のとおり: n はテキスト長, σ はアルファベットサイズ, r は BWT の連長, z は LZ 分解のサイズ, g は文法サイズ, γ は文字列アトラクタのサイズ, e は CDAWG のサイズを表す. パラメータ間に次の関係が知られている: $b < c < z < g$; $b < r < e$. 入力と出力の欄の記号は次のとおり: RL-BWT は連長圧縮 BWT, LZ77 と LZ78 は LZ 分解を, SLP は文法圧縮, SLP_{cs} はコラージュシステム, SLP_{rl} は連長文法圧縮, BM は双方向マクロスキーム, String attractor は文字列アトラクタ, CDAWG はコンパクト非巡回有向語グラフを表す. 圧縮計算?の欄の記号の意味は, 次のとおり: Yes は圧縮パラメータのみの多項式, Yes^- は圧縮パラメータと $\log n$ の多項式, No はそれ以外を表す. *表の本体 4 行目の $g \log n$ の $\log n$ は近似率で文法サイズを表す. 本体 7, 8 行目の String attractor の時間計算量の欄は, [10] の証明から得た粗い上限である.

入力	出力	作業領域		時間		文献
		領域計算量 (words)	圧縮計算?	時間計算量	圧縮計算?	
RL-BWT	LZ77	$O(r)$	Yes	$O(n \log r)$	No	Policriti and Prezza [16]
LZ77	RL-BWT	$O(r+z)$	Yes	$O(n(\log r + \log z))$	No	Policriti and Prezza [16]
LZ77	RL-BWT	$O(z \operatorname{polylog} n)$	Yes	$O(z \operatorname{polylog} n)$	Yes	Kempa and Kociumaka [9]
LZ77	SLP	$O(z + g \log n)$*	Yes	$O(n \log \sigma)$	No	Rytter [17]
SLP	LZ78	$O(g+z)$	Yes	$O((g+z)\log z)$	Yes	Bannai et al. [1]
String attractor	BM, SLP_{cs}	$O(\gamma \log(n/\gamma))$	Yes^-	$poly(O(\gamma \log(n/\gamma)))$	Yes^-	Kempa and Prezza [10]
String attractor	LZ77, SLP, SLP_{rl}	$O(\gamma \log^2(n/\gamma))$	Yes^-	$poly(O(\gamma \log^2(n/\gamma)))$	Yes^-	Kempa and Prezza [10]
CDAWG	**RL-BWT**	$O(e)$	**Yes**	$O(e)$	**Yes**	**This paper (Thm. 2)**

索引構造 $\mathcal{I}$ において，文脈から明らかな時に，ノード v や有向辺 e 等が $\mathcal{I}$ に含まれることを，$v \in \mathcal{I}$ や $e \in \mathcal{I}$ と書く．

2.2 テキストの部分文字列のなす同値関係

テキスト $T = T[0,n)$ の部分文字列全体 $Substr(T) \subseteq \Sigma^*$ の上の二項関係 $\equiv_L$（または $\equiv_R$) を次のように定める．T の部分文字列 $x \in Substr(T)$ の出現の開始位置（終了位置）全体の集合を x の**左出現**（または**右出現**）と呼ぶ．任意の $x, y \in \Sigma^*$ に対して, $x \equiv_L y$ (または $x \equiv_R y$) とは，T 中で x と y の左出現 (または右出現) が同じであるとき，そのときだけと定める．明らかに $\equiv_L$（または $\equiv_R$) は**同値関係** (equivalence relation) なので，任意の部分文字列 x に対して，$\equiv_L$ (または $\equiv_R$) に関する**同値類** (equivalence class) $[x]_L$ (または $[x]_R$) を定める．任意の $x \in Substr(T)$ に対して，同値類 $[x]_L$(または $[x]_R$) の**最長文字列**を，$\overrightarrow{x}$ (または $\overleftarrow{x}$) で表す．文字列 x を左右に拡張して得られる文字列を $\overleftrightarrow{x} := (\overleftarrow{\overrightarrow{x}}) = (\overrightarrow{\overleftarrow{x}})$ で定め，関連する**同値関係** $\equiv$ を $x \equiv y \overset{\text{def}}{\iff} \overleftrightarrow{x} = \overleftrightarrow{y}$ で定める．同値関係 $\equiv$ に関する**同値類**を $[x]$ で表す．ここに，$[x]$ 中の最長文字列は $\overleftrightarrow{x}$ である．以降，同値類 $\mathcal{E}$ の最長文字列を $value(\mathcal{E})$ で表す．

2.3 接 尾 辞 木

入力テキスト $T = T[0,n)$ の接尾辞全ての集合 $Suffix(T)$ を格納するコンパクトトライを，T の**接尾辞木** (suffix tree) [19] という．形式的には, T の接尾辞木は, 次の条件を満たす根付き木 $STree(T) = (V, E, root, label, suf)$ である．(i) 組 $(V, E, root)$ は**ノード集合** V と，**有向辺集合** $E \subseteq V^2$，**根** $root \in V$ をもつ根付き木をなす. (ii) 任意のノード v は, 子供の列 $children(v) \in V^*$ をもつ. (iii) 各有向辺 $e = (u, v) \in E$ の**辺ラベル**は，テキスト T の部分文字列 $label(e) = T[p, p+len)$ であり，対 $(p, len) \in \mathbb{N}^2$ を用いて $O(1)$ 語領域で表される．(iv) 任意のノード v の**出辺の先頭文字は互いに異なり，先頭文字の昇順で整列**されている．[5] (v) 任意のノード v は，根から v への一意なパス π 上の辺ラベルを順に連結して得られる**ラベル文字列** $str(v) \in Substr(T)$ を表す．(vi) $STree(T)$ は，T **の接尾辞全体に対応する** n **個の葉**をもつ．全ての葉を $\ell_0, \ldots, \ell_{n-1}$ と左から右に並べたとき，ℓ_k を**順位** k **の葉**と呼ぶ．**葉** ℓ_k **の順位**を $rank(\ell_k) = k$ で表わし，**対応する接尾辞** T_p **の開始位置**を $spos(\ell_k) = p \in [0,n)$ で表す.

(注5)：この仮定は，節 3. のアルゴリズムの正当性に本質的である.

2.4 CDAWG

提案アルゴリズム (節 4.) の入力となる入力テキスト $T[0,n)$ の**コンパクト非巡回語グラフ** (compact directed acyclic word graph, CDAWG) [2], [8] は，T の接尾辞全体の集合 $Suffix(T)$ を表現する最小かつコンパクトなオートマトンである．正確には，$T[0,n)$ の CDAWG は，次の条件を満たす辺ラベル付きの非巡回有向グラフ $CDAWG(T) = (V, E, root, label, suf)$ である.[6] (i) $V = \{[\overrightarrow{x}]_R |\ x \in Substr(T)\}$ は**頂点集合**である．(ii) $E = \{([\overrightarrow{x}]_R, \alpha, [\overrightarrow{x}\alpha]_R) |\ \alpha \in \Sigma^+,\ \overrightarrow{x} \neq \overrightarrow{x}\alpha\}$ は，辺ラベル付き**有向辺集合**である．各**有向辺** $e \in E$ は，辺の始点ノードと終点ノードの情報 $src(e)$ と $dst(e) \in V$ をそれぞれもつ．(iii) 有向辺 $e = (u, v) \in E$ の**ラベル**は，T 中の部分文字列 $label(e) = T[p, p+\ell) \in \Sigma^+$ であり，開始位置 $p = spos(e) \in [0,n)$ と長さ $\ell = slen(e) \in [1,n]$ の対を用いて $O(1)$ 語領域で符号化される．(iv) 任意のノード v に対して, v の**入辺全体**を $In(v)$ で表し, **出辺全体**を $Out(v)$ で表す．ノード v のすべての出辺は，異なる先頭のラベル文字を持ち，v **の子は入辺の先頭文字の文字順** $\leq_\Sigma$ **の昇順で整列済み**と仮定する.[7] (v) 根からノード v への任意の**パス** (path) $\pi = (e_1, \ldots, e_k)$ は，π 上の辺ラベルを順に連結して得られる**ラベル文字列** $str(\pi) := label(e_1) \cdots label(e_k) \in Substr(T)$ を表す．任意のノード $v = [\overrightarrow{x}]_R \in V$ は，**根から** v **への全てのパスのラベル文字列の集合**は，v 自身である右出現に関する部分文字列の同値

(注6)：部分文字列の右拡張演 $\rightarrow$ は非分岐辺の縮約（コンパクト化）に，同値類 $[\cdot]_R$ は接尾辞木 $STree(T)$ の同型部分木の共有に対応する．これらの定義については，節 2.2 を参照のこと.

(注7)：この仮定は，節 4. のアルゴリズムの正当性に本質的である.

類 $[\overrightarrow{x}]_R \subseteq Substr(T)$ と一致するので，これを $\mathcal{S}[\![v]\!] := [\overrightarrow{x}]$ で表す．(vi) 任意のノード $[\overrightarrow{x}]_R$ に対して，y が $[\overrightarrow{x}]_R \neq [\overrightarrow{y}]_R$ を満たす $value([\overrightarrow{x}]_R)$ の最長の接尾辞としたとき，ノード $[\overrightarrow{x}]_R$ の**接尾辞リンク** (suffix link) を $suf([\overrightarrow{x}]_R) = [\overrightarrow{y}]_R$ と定める．

以下で定義される **CDAWG の辺の分類**は，節 4. の提案アルゴリズムで主要な役割を果たす．CDAWG の任意のノード $v \in V$ に対して，同値類 $\mathcal{S}[\![v]\!]$ は一意な最長文字列 $value(v) = s_1$ を含むので（正確には補題 1），それが通る入辺 $e \in In(v)$ を**第一種辺** (primary edge) と定める．さらに，それ以外の辺を**第二種辺** (secondary edge) と定める．以降，E_1 と E_2 $(E = E_1 \uplus E_2)$ で，それぞれ，$CDAWG(T)$ の第一種辺と第二種辺の集合を表す．次の補題が成立する．

［補題 1］ 任意のノード v に対して，そのラベル文字列の集合 $\mathcal{S}[\![v]\!]$ が含む文字列は，$s_1 \sqsupseteq^{\mathrm{suf}} \cdots \sqsupseteq^{\mathrm{suf}} s_k$ と接尾辞関係 $\sqsupseteq^{\mathrm{suf}}$ で整列される．ここに，$k = |\mathcal{S}[\![v]\!]| \geq 1$ かつ $s_1 = value(\mathcal{S}[\![v]\!])$ である．さらに，任意の $2 \leq i \leq k$ に対し，一意な文字 $c_i \in \Sigma$ が存在して $s_{i-1} = c_i \cdot s_i$ となる．

[証明] $\mathcal{S}[\![v]\!]$ は右出現について同値な部分文字列の集合なので，主張の前半部が成立する．さらに任意の $x, y, z \in \Sigma^*$ に対してもし $x \sqsupseteq^{\mathrm{suf}} y \sqsupseteq^{\mathrm{suf}} z$ かつ $x, z \in \mathcal{S}[\![v]\!]$ ならば $y \in \mathcal{S}[\![v]\!]$ なので，$\mathcal{S}[\![v]\!]$ の文字列は長さが 1 ずつ異なり，後半部が成立する． ◇

CDAWG のサイズ: テキスト T の**極大反復** (maximal repeat) とは，左と右のどちらかに拡張すると，出現集合が真に変化するような T の部分文字列 $x = \overleftrightarrow{x} \in Substr(T)$ である [3], [8]. **左拡張** (left-extension)（または**右拡張** (right-extension)）は，極大反復 $\overleftrightarrow{x}$ と文字 $c \in \Sigma$ に対して $c\overleftrightarrow{x}$ の形（または $\overleftrightarrow{x}c$ の形）に書ける T の部分文字列をいい，両者を合わせて単に**拡張** (extension) と呼ぶ．以降，T 中の**極大反復と，左拡張，右拡張，拡張の総数**を，それぞれ，μ と，$e^l, e^r, e := e^l + e^r$ で表す．CDAWG は高々 μ 個のノードと，e^r 個の辺，e^l 個の接尾辞リンクをもち，テキスト部分を除いて $O(e)$ 語領域で格納できる [3].

2.5 連長圧縮 Burrows-Wheeler 変換 (RL-BWT)

提案アルゴリズム (節 3. と節 4.) の出力となる連長圧縮 Burrows-Wheeler 変換 [4], [7], [11] を導入する．

接尾辞配列を定義する．長さ n の入力テキストを $T[0, n) = T[0] \cdots T[n-1] \in \Sigma^{n-1}\$$ とおく．はじめに，T の**接尾辞配列** (suffix array) とは，テキスト T の全ての接尾辞 $T_0, \ldots, T_{n-1} \in Suffix(T)$ を辞書式順序の昇順でソートして得られた長さ n の整数配列 $SA[0, n) = SA[0] \cdots SA[n-1] \in [0, n)^n$ であり，$SA(T)$ で表す．$SA(T)$ は，任意の順位 $0 \leq k < n$ に対して，k 番目のマス目 $SA[k]$ に，$STree(T)$ の左から k 番目の葉 ℓ_k が表す接尾辞の先頭位置 $spos(\ell_k) \in [0, n)$ を格納した配列に等しい．

Burrow-Wheeler 変換 [4] を導入する．接尾辞配列 $SA(T)$ から，各 $0 \leq k < n$ に対して，辞書順が k 番目の接尾辞の開始位置 $pos := SA[k]$ の直前の文字 $T[pos-1]$ をとり，これらを左から右に並べることで得られる文字列 $BWT(T) := B[0] \cdots B[n-1] \in (\Sigma \cup \{\$\})^n$ を T の **Burrows-Wheeler 変換** (BWT) という．このとき，最長の接尾辞 T_0 では，$T[n-1] = \$$ を直前の文字として取り出す．すなわち，BWT では，$0 \leq k < n$ に対して第 k 番目の文字が $B[k] = T[(SA[k] - 1) \bmod n]$ と定義される．

連長圧縮 Burrow-Wheeler 変換 [7], [11] を定義する．任意の $r \geq 0$ と，r 個の重複を許した文字の並び $a_1, \ldots, a_r \in \Sigma$，$r$ 個の正整数 $\ell_1, \ldots, \ell_r \geq 1$ に対して，文字列 T が $T = \overbrace{a_1 \cdots a_1}^{\ell_1} \cdots \overbrace{a_r \cdots a_r}^{\ell_r}$ の形をもつと仮定する．ただし $(a_i \neq a_{i+1}, 1 \leq \forall i < r)$ とする．このとき，同じ文字の連続繰り返し $(a_i)^{\ell_i}$ を T の**連** (run) と呼ぶ．このとき，文字列 T の**連長圧縮符号化**とは，表現 $rle(T) = (a_1, \ell_1), \ldots, (a_r, \ell_r) \in (\Sigma \times \mathbb{N}^+)^r$ であり，長さ $\ell = \ell_1 + \cdots + \ell_r$ の文字列 $T = (a_1)^{\ell_1} \cdots (a_r)^{\ell_r} \in \Sigma^*$ を表す．文脈から明らかな場合は，以降では連長圧縮符号化を $rle(T) = (a_1)^{\ell_1} \cdots (a_r)^{\ell_r} \in \Sigma^*$ のように書く．連長圧縮符号化 $rle(T)$ の**連の個数**は，$|rle(T)| := r$ 個の連をもち，$r(\log \sigma + \log n)$ ビットで格納可能である．

以上の準備の下で，テキスト T に対する**連長圧縮 Burrow-Wheeler 変換** (Run-Length BWT, 連長圧縮 BWT) とは，$BWT(T)$ に対して連長圧縮符号化を行って得られる文字列 $RLBWT(T) := rle(BWT(T)) = (a_1)^{\ell_1} \cdots (a_r)^{\ell_r}$ であり，r は $BWT(T)$ 中の連数 r に関して，$r(\log \sigma + \log n)$ ビット領域，または $O(r)$ 語領域で格納可能である．

2.6 本稿で考察する問題

本稿では次の問題を考察する．

［定義 1］(CDAWG からの RL-BWT 構築問題） 入力として読み取り専用テキスト T と，T に対する CDAWG を受け取り，T に対する RL-BWT を出力せよ．

長さ n の任意のテキスト T に対して，その CDAWG のサイズ e と RL-BWT のサイズ r は，関係 $r \leq e \leq n$ を満たす [13]. 本稿では，上記の問題を入力サイズ e に比例した時間と領域で解くアルゴリズムを考察する．

3. 接尾辞木からの RLBWT 構築アルゴリズム

本節では，提案アルゴリズムの説明の前段階として，テキスト $T[0, n)$ とノード数 $O(n)$ の T の接尾辞木を入力として，$O(n)$ 時間と $O(n)$ 語領域で，T の RL-BWT を構築する基本アルゴリズムを与える．次節では，このアルゴリズムを CDAWG 入力に拡張する．入力接尾辞木に関する仮定は節 2.3 を参照されたい．

3.1 基本アイディア

本節のアルゴリズムは，入力の接尾辞木 $STree(T) = (V, E, root, label, suf)$ を用いて，**後順巡回** (post-order traversal) で葉から根に向かって探索を行い，各ノード v に対応する連長圧縮 BWT の部分配列 $RLBWT(v)$ を動的計画法を用いて計算する．深さ優先探索の終了時に，根に対応する部分配列 $RLBWT(root)$ が所望の解 $RLBWT(T)$ を与える．以下では，説明の都合上はじめに T の BWT の計算方法を述べ，次にどのように連長圧縮 BWT へ拡張するか述べる．

初めに，$STree(T)$ のノードと，$RLBWT(T)$ の部分区間の対応づけを与える．節 2.5 と節 2.3 の定義から，$STree(T)$ の葉を左から右に並べた列 $\ell_0, \ldots, \ell_{n-1}$ は，接尾辞配列 $SA(T) = SA[0] \cdots SA[n-1] \in [0, n)^n$ と 1 対 1 で対応す

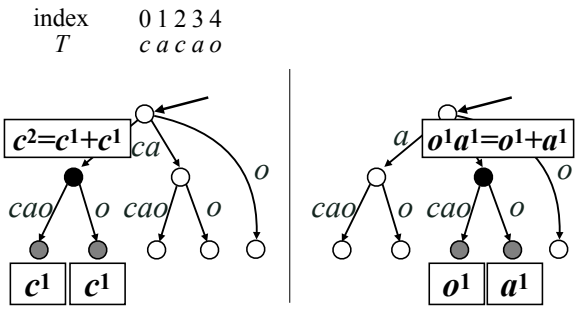

図 2: 接尾辞木の内部ノードにおける連長圧縮配列 $RLBWT(v)$ の再帰的計算の例 (補題 3). 左と右は，それぞれ，隣接部分の文字が同じ場合と異なる場合の連結操作を示す.

る．そこで，$STree(T)$ の任意のノード v に対して，その部分木の葉がもつ**最小順位**と**最大順位**を $L(v), R(v)$ $(0 \le L \le R < n)$ と，v のもつ葉列を $(\ell_L(v), \ldots, \ell_R(v)) \in V^*$ とおいて，ノード v の**順位区間** (rank interval) を $Int(v) := [L(v), R(v)] \subseteq [0, n)$ とする．以上の記法を用いて，v に対応する $BWT(T)$ **の部分配列**を，$BWT(v) := BWT[L(v), R(v)] \in \Sigma^*$ とし，それを連長符号化した $RLBWT(T)$ **の部分配列**を $RLBWT(v) := rle(BWT(v)) = RLBWT[L(v), R(v)] \in (\Sigma \times \mathbb{N})^*$ で定義する.

次に $BWT(v)$ の計算を考える．配列の連結演算 $\circ$ を，$(a_1, \ldots, a_p) \circ (b_1, \ldots, b_q) := (a_1, \ldots, a_p, b_1, \ldots, b_q)$ で定める.

[補題 2](BWT の漸化式) 任意のノード v に対して，部分配列 $BWT(v)$ は，次の帰納条件を満たす:

(1) ノード v が葉であり，その順位が $0 \le k < n$ ならば，$BWT(v) = c \in \Sigma$ が成立する．ここに，$c := T[(spos(v) - 1) \bmod n]$ である.

(2) ノード v が内部ノードであり，m 個の子 $children(v) = (v_i)_{i=1}^m$ をもつならば，$BWT(v) = BWT(v_1) \circ \cdots \circ BWT(v_m)$ が成立する.

[証明] (1) このとき，$v = \ell_k$ は開始位置 $p := spos(v) = SA[k]$ をもつ接尾辞 $T_p \in Suffix(T)$ に対応することから示される．(2) ノード v の部分木の葉列は，子ノード $v_1, \ldots, v_m$ の部分木の葉列を左から右に連結したものに等しいので示される． ◇

さらに連長圧縮 BWT の部分配列の計算を考える．BWT の場合と異なり，単純な連結演算 $\circ$ ではうまくいかない．この問題を解決するために，**二つの連長圧縮符号化 α と β の連結** $\alpha \oplus \beta$ を次のように定義する: 任意の連長圧縮符号化を $\alpha = \alpha' \circ (c)^\ell$ と $\beta = (d)^m \circ \beta'$ $(\forall c, d \in \Sigma, \forall \ell, m \in \mathbb{N}^+)$ とする：

(1) 隣接する文字が異なるときは $(c \ne d)$，通常の配列の連結演算 $\circ$ で連結する：$\alpha \oplus \beta := \alpha \circ \beta$.

(2) 隣接する要素の文字が同一のときは $(c = d)$，一つに併合した上で，残りを連結する：$\alpha \oplus \beta := \alpha' \circ (c)^{\ell+m} \circ \beta'$.

上記の議論から，任意の文字配列 $x, y \in \Sigma^+$ に対して，$rle(x \circ y) = rle(x) \oplus rle(y)$ が成立することが言える．さらに，連長符号化を両方向連結リストで実装し，両端の要素のポインタを保持することで，連結 $\alpha \oplus \beta$ は連長符号化の長さによらず，定数時間で実行できる.

Algorithm 1 接尾辞木から RL-BWT を構築するアルゴリズム

Require: 接尾辞木 $STree$ の任意のノード v, 入力文字列 $T[0, n)$
Ensure: 連長圧縮 BWT 配列 $RLBWT$

```
1: function RecStreeToRLBWT(v, T)
2:     if v = leaf then
3:         c ← T[(spos(v) − 1) mod n]
4:         α ← (c)^1
5:     else
6:         α ← ε                                   ▷ α = RLBWT(v)
7:         for each w ∈ children(v) do
8:             β ← RecStreeToRLBWT(w, T)
9:             α ← α ⊕ β                           ▷ 連長圧縮 BWT の連結
10:    return α
```

以上の議論と補題 2 から，直ちに次の補題が成立する.

[補題 3]($RLBWT$ の漸化式) 任意のノード v に対して，部分配列 $RLBWT(v)$ は，次の帰納条件を満たす:

(1) ノード v が葉であり，その順位が $0 \le k < n$ ならば，$RLBWT(v) = (c)^1 \in (\Sigma \times \mathbb{N})$ が成立する．ここに，$c := T[(spos(v) - 1) \bmod n]$ である.

(2) ノード v が内部ノードであり，m 個の子 $children(v) = (v_i)_{i=1}^m$ をもつならば，$RLBWT(v) = RLBWT(v_1) \oplus \cdots \oplus RLBWT(v_m)$ が成立する.

図 2 に接尾辞木における連長圧縮 BWT 配列の再帰的計算のようすを示す.

アルゴリズム 1 に，補題 3 に基づいて，T と $STree(T)$ から $RLBWT(T)$ を $O(n)$ 時間で計算する再帰手続き RecStreeToRLBWT を示す．この再帰手続きは，接尾辞木 $STree(T)$ の根 $root$ からはじめて $STree(T)$ の**後順巡回** (post-order traversal) を用いて探索を行う．この際，補題 3 に基づいて，葉から根に向かってすべてのノード v を訪問し，動的計画法により $RLBWT(v)$ を計算する．探索終了時は，根がもつ配列 $RLBWT(root) = RLBWT(T)$ を解として出力する.

[定理 1] アルファベット Σ 上の長さ n のテキスト T と，T の接尾辞木 $STree$ が入力として与えられたとき, アルゴリズム 1 は，T の連長圧縮 BWT 配列 $RLBWT$ を，$O(n)$ 時間と $O(n)$ 語領域で計算する.

[証明] 正当性は，補題 3 と本節の議論から直ちに示される．時間計算量について，再帰手続き RecStreeToRLBWT の繰り返しにおいて，ノード v が葉の場合の計算は定数時間で実行できる．一方，v が分岐ノードの場合は，子 w の連結演算は定数時間で実行でき，これを子に課金して償却すると，総計算時間は $O(n)$ となる．メモリに関して，全ノードの部分配列サイズの総和は最悪時に $O(n^2)$ だが，$STree(T)$ のノード v に子へのポインタで表現した $RLBWT(v)$ の差分表現を保持することで，メモリ量を $O(n)$ 語に抑えられる．この際，探索終了時の解 $RLBWT(root)$ の出力は，高々 $O(n)$ 本のポインタをたどり，$O(n)$ 時間で可能である．以上から，定理が示された． ◇

4. CDAWG から RL-BWT を構築する提案アルゴリズム

本節では，前節の接尾辞木に対する結果を，CDAWG を入力とする場合に拡張する．主結果として，長さ n のテキスト T と，サイズ $O(e)$ の $CDAWG(T)$ から，サイズ r の $RLBWT(T)$ を $O(e)$ 時間と $O(e)$ 語領域で構築するアルゴリズムを与える．

4.1 基本アイディア

基本的な方針として，提案アルゴリズムは，接尾辞木のノードの巡回を用いた前節のアルゴリズムの実行を，$CDAWG(T)$ がもつ $O(e)$ 個のパス上の巡回を用いて模倣する．しかし，$CDAWG(T)$ のパスは $STree(T)$ のノードに 1 対 1 対応するので，展開すると $O(n)$ 本の異なるパスが生じてしまう．そこで，探索対象となるパスの適切な部分集合 $LPT_+ \subseteq CDAWG(T)$ を定める．以降では混同を避けるため，$STree(T)$ のノード等を w^{st} のように上添字 $\cdot^{\mathrm{st}}$ をつけて表記する．

$CDAWG(T)$ の**最長パス** (solid path) とは，第一種辺だけからなるパスであり，あるノード v に対して $\pi = value(v)$ と書けるパスをいう．次に $CDAWG(T)$ 上の動的計画法を実行するために，上記の最長パスに加えて，最長パスの末尾に第二種辺を連結して得られる任意のパス π である**境界パス** (border path) を導入する. 以降，最長パスと境界パスを合わせて，**探索可能パス** (searchable path) と呼ぶ．$LPT(T) \subseteq Path(T)$ で**最長パス全ての集合**を表し，$LPT_+(T) = LPT(T) \cup \{s \cdot e \in Path(T) \mid s \in LPT(T), e \in E_2\}$ で**探索可能パス全ての集合**を表す．$LPT(T)$ と $LPT_+(T)$ は，それが含むパスの接頭辞について閉じている．以降，文脈から明らかならば，$LPT(T)$ と $LPT_+(T)$ でそれぞれ誘導する $STree(T)$ の部分木を表わし，**最長パス木** (longest path tree, LPT) と**拡張最長パス木** (extended LPT) と呼ぶ.

今，$CDAWG(T)$ の任意のパス π に対して，$STree(T)$ **の根からラベル文字列** $str(\pi)$ **で到達するノード**を $w_\pi^{\mathrm{st}} \in STree(T)$ とし，節 3. で導入した $STree(T)$ のノードに対応する順位区間の記法を用いて，その部分木の左端の順位 $L(\pi) := L(w_\pi^{\mathrm{st}})$ と，右端の順位 $R(\pi) := R(w_\pi^{\mathrm{st}})$，**パス** π **に対応する順位区間** $Int(\pi) := [L(\pi), R(\pi)]$ を定義する．

任意のノード v に，その最長パス $\pi_v := value(v)$ に対応する順位区間 $Int(v)$ を割り当てる．v **の LPT 順位区間**を $Int(v) := Int(\pi_v) = [L(\pi_v), R(\pi_v)]$ と定め，**ノード** v **に対応する** $BWT(T)$ **の部分配列**を $BWT(v) := BWT[Int(v)]$ と定め．さらに v **に対応する** $RLBWT(T)$ **の部分配列**を $RLBWT(v) := rle(BWT(v)) = RLBWT[Int(v)]$ と定める．任意の探索可能パス $\pi \in LPT_+(T)$ の末尾の辺 e に対して，パス $value(e)$ を用いて $RLBWT(e) := RLBWT(value(e))$ と定める．とくに e が第一種辺ならば $RLBWT(e) := RLBWT(dst(e))$ である．その他の辺 $g \notin LPT_+(T)$ に対して $RLBWT(e) := \perp$ と定める．

4.2 アルゴリズムの概要

$RLBWT(T)$ の計算には，動的計画法による $RLBWT(v)$ の計算に先立ち，後述する Weiner 文字 $WC(e)$ と葉数 $NL(v)$ と呼ばれる補助情報を計算する．そこで，提案アルゴリズム CDawg-ToRLBWT は，入力の $CDAWG(T)$ を受け取り，次のステップを順に実行する

Algorithm 2 入力として $CDAWG$ の任意のノード v が与えられたとき, v の葉数 $NL(v)$ を計算する．配列 $NL(\cdot)$ と $VISITED(\cdot)$ は実行前に，それぞれ，0 と $false$ で初期化されると仮定する．

```
1:  function ComputeNL(v)
2:      if v = sink then
3:          return 1
4:      else
5:          NL(v) ← 0
6:          for each e ∈ Out(v) do
7:              if VISITED(dst(e)) then
8:                  NL(v) ← NL(v) + NL(dst(e))
9:              else
10:                 ComputeNL(dst(e))
11:         VISITED(v) ← true
12:         return NL(v)
```

（1）Step 1: $CDAWG$ 上で，根から開始して，ノードの葉数の配列 $NL(\cdot)$ を計算する (アルゴリズム 2).

（2）Step 2: $CDAWG$ 上で，根から開始して，辺の Weiner 文字の配列 $WC(\cdot)$ を計算する (アルゴリズム 3).

（3）Step 3: $CDAWG$ 上で，根から開始して，ノードの連長圧縮 BWT 文字列の配列 $RLBWT(\cdot)$ を計算する (アルゴリズム 4)

（4）Step 4: 出力として $RLBWT := RLBWT(\varepsilon)$ を返す.

以下の小節では，必要な用語と各ステップの手続きを順に説明する．また，入力の CDAWG に関して，次の仮定をおく．入力の $CDAWG = (V, E, root, sink, label, suf)$ は, Σ 上の長さ n のテキスト T から，Inenaga ら [8] のアルゴリズムで構築され，T は読み取り専用と，各ノードの出辺はラベル文字列の先頭文字の昇順で整列されると仮定する．ここに，$r \leq e$ である．

4.3 葉数の計算

はじめに，$CDAWG(T)$ のパスがもつ順位区間について，次の補題を示す.

［補題 4］ $CDAWG(T)$ の一つのノードで表されるパス全てについて，対応する $STree(T)$ のノードがもつ部分木の順位区間は互いに等しい.

補題 4 から，$CDAWG(T)$ のあるノード v へのパス全てについて，対応する $STree(T)$ のノードの葉数は等しく，その葉数を知るには右同値なパスのどれを展開しても良い．そこで，$value(v)$ を代表に用いて，葉数 $NL(v)$ を次のように定義する．CDAWG の任意のノード v の**葉数**を，v がもつ最長パス $s = value(v)$ に対応する $STree(T)$ のノード w_π^{st} がもつ部分木の葉の数 $NL(v) := Int^{\mathrm{st}}(w_\pi^{\mathrm{st}})$ と定める．$CDAWG(T)$ の出口ノード $sink$ は，$STree(T)$ の任意の葉に対応し，その葉数は $NL(sink) = 1$ となる．

アルゴリズム 2 に, CDAWG のノードの葉数 $NL(\cdot)$ を計算する手続きを示す．この手続きは, 根から CDAWG を深さ優先探索し, 各ノード v の葉数 $NL(v)$ を再帰的に計算する．補題 4 と上記の議論から，手続きがノードの出辺をどの順番で訪問して

Algorithm 3 入力の $CDAWG$ の任意のノード v に対して, v の出辺 $e \in Out(v)$ の Weiner 文字 $WC(e)$ を計算する．トップレベルで ComputeWC$(root, 0, T)$ として呼び出す．

```
1: function ComputeWC(v, ℓ, T)
2:     if v = sink then return
3:     else
4:         for each e ∈ Out(v) do
5:             if e = primary edge then
6:                 ComputeWC(dst(e), ℓ + slen(e), T)
7:             else
8:                 WC(e) ← T[spos(e) − ℓ − 1]
9:     return
```

も，正しく配列 $NL(\cdot)$ を計算することが示せる．

4.4 第二種辺が表すパスの特徴づけ

ここで，第二種辺が表すパスの特徴づけを行う．今，$CDAWG(T)$ のノード v の全ての入辺 $e_1, \ldots, e_m \in In(v)$ $(m \geq 1)$ によって，右同値なパスの集合 $\mathcal{S}[\![v]\!]$ を互いに素な部分集合の族 $\mathcal{S}[\![v]\!] = \mathcal{S}[\![e_1]\!] \uplus \cdots \uplus \mathcal{S}[\![e_m]\!]$ に分割する．ここに，$\forall i \in [1, m]$ に対して，$\mathcal{S}[\![e_i]\!]$ は第 i 辺 e_i を通る $\mathcal{S}[\![v]\!]$ のパスの集合である．さらに，任意の入辺 $e := e_i \in In(v)$ の**最長文字列と最短文字列**を，それぞれ，集合 $\mathcal{S}[\![e]\!]$ の最長文字列 $value(e) := \arg\max_{s \in \mathcal{S}[\![e]\!]} |s|$ と最短文字列 $short(e) := \arg\min_{s \in \mathcal{S}[\![e]\!]} |s|$ で定義する．このとき，$value(e)$ は，もし e が第一種辺ならば最長パスであり，第二種辺ならば境界パスである．このとき，次の補題が成立する．

[補題 5]（部分文字列の直前の文字） $CDAWG(T)$ において，任意のノードを v とし，任意の第二種辺を $e \in In(v)$ とする．このとき，辺 e の最長文字列 $s := value(e)$ にのみ依存するある文字 $c_s \in \Sigma$ が存在して，次の条件 (1)–(3) が成立する:

(1) ある辺 $f \in In(v)$ に対して，$short(f) = c_s \cdot value(e)$.

(2) s の T 中の任意の左出現位置 $p \in [0, n)$ に対して，その直前の文字 $T[p-1]$ は c_s に一致する，すなわち，$T[p-1] = c_s$ が成立する．

(3) $c_s = T[spos(e) - |value(src(e))| - 1]$ が成立する

[証明] 接尾辞関係で $value(e_{i_1}) \sqsupseteq^{\mathrm{suf}} \cdots \sqsupseteq^{\mathrm{suf}} value(e_{i_m})$ と入辺を整列すると，補題 1 から $\mathcal{S}[\![e_{i_1}]\!], \ldots, \mathcal{S}[\![e_{i_m}]\!]$ の順と整合的に $\mathcal{S}[\![v]\!]$ のパスを長さの降順と $\sqsupseteq^{\mathrm{suf}}$ の順で整列できる．(1) 辺 e は第二種辺なので，ある $j \geq 2$ に対して $e = e_{i_j}$ かつ，先行する入辺 $f = e_{i_{j-1}}$ に対して，ある文字 $c := WC(e)$ に対して，$short(f) = cs$ かつ $value(e) = s$ となる．(2) ここで $cs, s \in \mathcal{S}[\![v]\!]$ なので，同じ右出現をもつ．したがって，(i) T 中で s と cs は同じ右出現位置 q をもち，(ii) $T[p, p+|s|) = s$ なので，s と cs の左出現位置は，それぞれ，$p = q - |s|$ と $p-1$ となり，$T[p-1] = c$ が言える．(3) 入力の $CDAWG(T)$ は [8] のオンライン手続きで構築された仮定から，$T[spos(e), spos(e)+slen(e))$ は $s = value(e)$ のある出現 $T[p, p+|s|)$ の接尾辞と仮定して良い．ここで s は境界パスなので，$t = value(src(e))$ をその接頭辞としてもち，$s = t \cdot label(e)$ となり，$p + |t| = spos(e)$ となるので，これを $p-1$ について求めると主張が示される． ◇

任意の入辺 e の **Weiner 文字** (Weiner character) を，任意の第二種辺 e に対して補題 5 で存在が保証される文字 $WC(e) := c_s$ と定める．辺 e が第一種の入辺のときは $WC(e) := \perp$ と定める．アルゴリズム 3 に，補題 5 に基づいて，CDAWG の第二種辺の Weiner 文字を $O(e)$ 時間と領域で計算するアルゴリズムを示す．この手続きは，$CDAWG(T)$ を深さ優先探索し，$WC(e)$ を再帰的に計算する．

Algorithm 4 $CDAWG(T)$ から $RLBWT(T)$ を構築するアルゴリズム．トップレベルで，RecCdawgToRLBWT$(root)$ として呼び出す．

Require: $CDAWG$ の任意のノード v
Ensure: 連長圧縮 BWT 配列 $RLBWT(T)$

```
1: function RecCdawgToRLBWT(v)
2:     if v = sink then
3:         α ← $^1                          ▷ テキスト末尾の文字 $ = T[n − 1]
4:     else
5:         α ← ε                            ▷ α = RLBWT(v)
6:         for each e ∈ Out(v) do
7:             if e = primary edge then
8:                 β ← RecCdawgToRLBWT(dst(e))
9:             else
10:                ℓ ← NL(dst(e)) and a ← WC(e)
11:                β ← a^ℓ (∈ Σ × ℕ)
12:            α ← α ⊕ β                    ▷ RLBWT 配列の結合 (節 3.1)
13:    return α
```

4.5 アルゴリズム全体と計算量解析

提案アルゴリズムは，$CDAWG(T)$ 上を根から深さ優先探索しながら，最長パスと境界パスからなる探索パス上でボトムアップに $RLBWT(e)$ を計算する．次の補題は，提案アルゴリズムの正当性に関して重要である．

[補題 6]（$CDAWG(T)$ 上の $RLBWT$ 配列の漸化式） 任意の辺 e と，その終端のノード $v = dst(e)$ に対して，部分配列 $RLBWT(e)$ は，次の帰納条件を満たす:

(1) 辺 e が第一種辺かつ，ノード v が $sink$ ならば，$RLBWT(e) = (\$)^1 \in (\Sigma \times \mathbb{N})$．ここに，$\$:= T[n-1]$ はテキストの末尾文字である．

(2) 辺 e が第二種辺ならば，$RLBWT(e) = (c)^\ell$．ここに，$c := WC(e)$ は第二種辺 e の Weiner 文字であり，$\ell := NL(v)$ はノード v の葉数である．

(3) 辺 e が第一種辺かつ，ノード v が m 本の出辺 $Out(v) = (e_i)_{i=1}^m$ をもつ内部ノードならば，$RLBWT(e) = RLBWT(e_1) \oplus \cdots \oplus RLBWT(e_m)$.

[証明] まず (1)–(3) のすべてのケースにおいて，e は $LPT_+(T)$ のパスの末尾の辺である．次に，構造に関する帰納法を用いて示す．(1) 第一の基底ケース：$RLBWT(e) = (c)^\ell$ とおくと，補題 2 の (1) から，$\ell = 1$ であり，$sink$ は最長文字列として $T_0 = T[0, n)$ を表すので，$c = T[(0-1) \mod n] = T[n-1]$ となる．(2) 第二の基底ケース：辺 e を通る T の接尾辞全てを $Suffix(T, e) \subseteq Suffix(T)$ とおくと，これは $s = value(e)$ の後に $v = Int(dst(e))$ を根とする部分グラフの葉までのパスを連結して得られるパス全体に等しい．一方，補題 5 から，e

4.5 アルゴリズム全体と計算量解析

提案アルゴリズムは，$CDAWG(T)$ 上を根から深さ優先探索しながら，最長パスと境界パスからなる探索パス上でボトムアップに $RLBWT(e)$ を計算する．次の補題は，提案アルゴリズムの正当性に関して重要である．

[補題 6]（$CDAWG(T)$ 上の $RLBWT$ 配列の漸化式） 任意の辺 e と，その終端のノード $v = dst(e)$ に対して，部分配列 $RLBWT(e)$ は，次の帰納条件を満たす:

(1) 辺 e が第一種辺かつ，ノード v が $sink$ ならば，$RLBWT(e) = (\$)^1 \in (\Sigma \times \mathbb{N})$．ここに，$\$:= T[n-1]$ はテキストの末尾文字である．

(2) 辺 e が第二種辺ならば，$RLBWT(e) = (c)^\ell$．ここに，$c := WC(e)$ は第二種辺 e の Weiner 文字であり，$\ell := NL(v)$ はノード v の葉数である．

(3) 辺 e が第一種辺かつ，ノード v が m 本の出辺 $Out(v) = (e_i)_{i=1}^m$ をもつ内部ノードならば，$RLBWT(e) = RLBWT(e_1) \oplus \cdots \oplus RLBWT(e_m)$.

[証明] まず (1)–(3) のすべてのケースにおいて，e は $LPT_+(T)$ のパスの末尾の辺である．次に，構造に関する帰納法を用いて示す．(1) 第一の基底ケース：$RLBWT(e) = (c)^\ell$ とおくと，補題 2 の (1) から，$\ell = 1$ であり，$sink$ は最長文字列として $T_0 = T[0, n)$ を表すので，$c = T[(0-1) \mod n] = T[n-1]$ となる．(2) 第二の基底ケース：辺 e を通る T の接尾辞全てを $Suffix(T, e) \subseteq Suffix(T)$ とおくと，これは $s = value(e)$ の後に $v = Int(dst(e))$ を根とする部分グラフの葉までのパスを連結して得られるパス全体に等しい．一方，補題 5 から，e

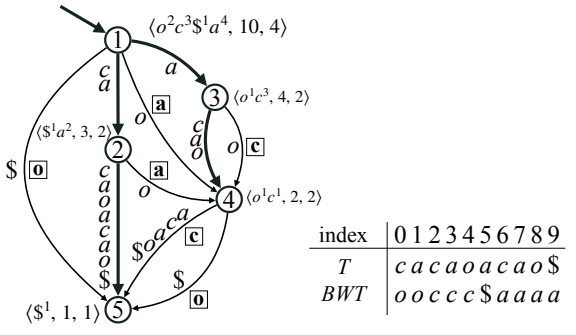

図 3: $CDAWG(T)$ 上の $RLBWT(T)$ の計算の例 (アルゴリズム 4). 図中に, 円はノードを表し, 円の中の数字はノード番号を表す. 太い矢印が第一種辺を, 細い矢印が第二種辺を表し, 第二種辺 e の横の四角で囲われた文字は Weiner 文字 $WC(e)$ を表す. ノード v の横の $\langle\cdot\rangle$ で囲われた三つ組はそれぞれ, v の $RLBWT(v)$, 葉数 $NL(v)$, $RLBWT(v)$ の連長を表す.

にのみ依存する文字 $c_s \in \Sigma$ が存在し，$value(e)$ の全ての左出現位置の直前の文字となるので，$Suffix(T, e)$ の全てのパスの T 中の左出現位置の直前の文字は c_s となるので，示された．(3) パス $value(e) \in LPT(T)$ が言えるので，その出辺 e_i に対して $value(e_i) \in LPT_+(T)$ であることが示せる．よって $RLBWT(e)$ は定義され，帰納法の仮定から示される． ◇

アルゴリズム 4 に，補題 6 に基づいて，入力の $CDAWG(T)$ とテキスト T から，出力の $RLBWT(T)$ を $O(e)$ 語領域と時間で計算する再帰手続き RecCdawgToRLBWT を示す. この手続きは，根から開始して CDAWG を深さ優先探索し，各ノード v に対して $RLBWT(v)$ を再帰的に計算する. 図 3 に，テキスト $T = cacaoacao\$$ の $CDAWG$ に対するアルゴリズム 4 の実行例を示す．以上をまとめると，節 4.2 に示した提案アルゴリズム CdawgToRLBWT は，入力 $CDAWG(T)$ を受け取り，アルゴリズム 2 と，アルゴリズム 3，アルゴリズム 4 を順に実行し，$RLBWT(T)$ を計算する．

［**定理 2**］ 節 4.2 の CdawgToRLBWT は，入力として Σ 上の長さ n のテキスト T と, $CDAWG(T)$ を受け取り，出力として連長圧縮 BWT 配列 $RLBWT(T)$ を, $O(e)$ 時間と領域で計算する.

[証明] 正当性に関して，補題 4 と補題 5 から，前処理でアルゴリズム 2 の手続き ComputeNL とアルゴリズム 3 の手続き ComputeWC は $NL(\cdot)$ と $WC(\cdot)$ の表を正しく計算し，補題 6 の漸化式から，アルゴリズム 4 の再帰手続き RecCdawgToRLBWT は，$RLBWT(\cdot)$ を全てのノードに対して正しく計算する．計算時間は繰り返しにおいて各行が定数時間しか要しないので示された．領域量については，$NL(v)$ と $WC(e)$ については，それぞれ一つあたり定数サイズであり，全体として $O(\mu)+O(e) = O(e)$ 語領域である．RL-BWT を保持するメモリ量は，再帰手続きのバックトラックの際にメモリを解放するので，作業変数 α には RL-BWT が先頭から左詰めに追加されることと合わせて，メモリ量は $O(r)$ である．よって，全体で $O(e+r) = O(e)$ 語領域で抑えられる． ◇

5. おわりに

本稿では, 入力として与えられた長さ n の読み取り専用テキストとサイズ e の CDAWG から，サイズ r の RL-BWT を $O(e)$ 語作業領域と時間で構築する効率良いアルゴリズムを提案した.

文　献

[1] H. Bannai, P. Gawrychowski, S. Inenaga, and M. Takeda. Converting SLP to LZ78 in almost linear time. In *Annual Symposium on Combinatorial Pattern Matching*, pp. 38–49. Springer, 2013.

[2] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M.-T. Chen, and J. Seiferas. The smallest automation recognizing the subwords of a text. *TCS*, 40:31–55, 1985.

[3] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *JACM*, 34(3):578–595, 1987.

[4] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer, 1994.

[5] P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM (JACM)*, 52(4):552–581, 2005.

[6] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20, 2007.

[7] T. Gagie, G. Navarro, and N. Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM (JACM)*, 67(1):1–54, 2020.

[8] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi. On-line construction of compact directed acyclic word graphs. *Discrete Appl. Math.*, 146(2):156–179, 2005.

[9] D. Kempa and T. Kociumaka. Resolution of the Burrows-Wheeler Transform Conjecture. In *FOCS 2020*, pp. 1002–1013. IEEE, 2020.

[10] D. Kempa and N. Prezza. At the roots of dictionary compression: string attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 827–840, 2018.

[11] V. Mäkinen and G. Navarro. Succinct Suffix Arrays based on Run-Length Encoding. *Nord. J. Comput.*, 12(1):40–66, 2005.

[12] J. I. Munro, G. Navarro, and Y. Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Proc. SODA 2017*, pp. 408–424. SIAM, 2017.

[13] G. Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Comput. Surveys*, 54(2):1–31, 2021.

[14] G. Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Comput. Surveys*, 54(2):1–32, 2021.

[15] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2–es, 2007.

[16] A. Policriti and N. Prezza. From LZ77 to the Run-Length Encoded Burrows-Wheeler Transform, and back. In *CPM 2017*, Vol. 78 of *LIPIcs*, pp. 17:1–17:10, 2017.

[17] W. Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *TCS*, 302(1-3):211–222, 2003.

[18] T. Takagi, K. Goto, Y. Fujishige, S. Inenaga, and H. Arimura. Linear-size CDAWG: New repetition-aware indexing and grammar compression. In *SPIRE 2017*, pp. 304–316. Springer, 2017.

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