Vol.2012-AL-142 No.9
2012/11/2

疎な接尾辞木構築の Word RAM 上の高速化

高木 拓也[1,a)]　上村 卓史[2]　有村 博紀[3,b)]

概要：長さ N のテキストの $K \leq N$ 個の索引点に対する接尾辞木を疎接尾辞木 (sparse suffix tree) といい，$O(K)$ 語の領域しか使用しないため，さまざまな応用に用いられている．上村と有村 (Proc. CPM2011, LNCS 6661, 2011) は，長さ N ビットのハフマン圧縮テキストが入力として与えられたとき，その疎接尾辞木を $O(\delta)$ 前処理時間と $O(K+\delta)$ 語の領域を用いて，オンライン構築するアルゴリズムを与えている．本稿では，ビット並列計算と簡潔トライ構造からなる詰め込み文字列 (packed string) 技法を用いて，長さ $O(N)$ ビットのハフマン圧縮テキストに対して，疎接尾辞木を $O(\delta)$ 前処理時間と $O(K+\delta)$ 語の領域を用いて，$O(\lceil \frac{N}{w} \rceil \sqrt{w} + K\sqrt{w})$ 時間でオンライン構築するアルゴリズムを与える．ここに，w は計算機のレジスタ長 (ビット) であり，δ は符号中の符号語長さの総和である．これは，疎接尾辞木のオンライン構築で初めて，$O(N)$ 時間より少ない計算時間を達成したアルゴリズムである．提案手法は，$K \leq O(\frac{N}{\sqrt{w}})$ のときに従来手法より高速である．また，一般の有限接頭符号上や，単語アルファベット上の符号化テキストに対しても拡張可能である．

キーワード：疎接尾辞木，詰めこみ文字列技法，接頭符号，簡潔データ構造，q-fast trie

Faster Sparse Suffix Tree Construction on Word RAM

Takuya Takagi[1,a)]　Takashi Uemura[2]　Hiroki Arimura[3,b)]

Abstract: We present an efficient algorithm on Word RAM for constructing a sparse suffix tree on an encoded text over a regular prefix-code in $O(\lceil \frac{N}{w} \rceil \sqrt{w} + K\sqrt{w})$ time using $O(\delta)$ preprocessing and $O(K+\delta)$ word space, where N is the length of the text in base letters, K is the length of the text in code words, σ is the size of a base alphabet Σ, δ is the total size of a code alphabet on Σ, and w is a bit-length of a register of Word RAM.

Keywords: sparce suffix tree, packed string, word RAM, prefix code, Q-fast trie,

1. 序論

1.1 背景

長さ N のテキストの $K \leq N$ 個の索引点に対する接尾辞木を，疎接尾辞木 (sparse suffix tree．以後，SST と略す) [9] とよぶ．疎接尾辞木は，$O(K)$ 語の領域しか使用しないため，自然言語テキスト索引 [1], [8] や，圧縮テキストの索引 [13]，簡潔データ構造 [11] などのさまざまな応用に用いられている．一般のテキストに対する疎接尾辞木を $O(K)$ 領域で $O(N)$ 時間で構築できるかは未解決の問題であるが，その部分ケースである入力テキストが等間隔の索引点を持つ場合 [9](等間隔索引付け接尾辞木) や，空白区切りをもつ単語列の場合 [1], [8](単語接尾辞木) には，オンライン線形時間構築が可能である．

最近，上村と有村ら [13] は，接頭符号上の長さ N 文字かつ K 符号の符号化テキストが入力として与えられたとき，その疎接尾辞木を $O(\delta)$ 前処理時間と $O(K+\delta)$ 語の領域を用いて，$O(N)$ 時間でオンライン構築するアルゴリズムを与えた．ここに，δ は文字数で計った接頭符号の総サイズである．これは，先行研究 [1], [8], [9] の一般化であ

1 北海道大学 工学部 Hokkaido University, Fac. Eng.
2 株式会社調和技研 Chowa Giken Co.
3 北海道大学大学院情報科学研究科 Hokkaido University, IST.
a) tkg@ec.hokudai.ac.jp
b) arim@ist.hokudai.ac.jp

り，一般の正則な（有限オートマトンで受理可能な）接頭符号上の符号化テキストに対しても適応可能である．

1.2 主結果

本稿では，上村と有村ら [13] の手法を，ビット並列計算と簡潔トライ構造を用いて Word RAM 上で高速化し，入力の N ビットかつ K 符号語のハフマン符号化テキストに対して，疎接尾辞木を $O(\delta)$ 前処理時間と $O(K+\delta)$ 語の領域を用いて，$O(\lceil \frac{N}{w} \rceil \sqrt{w} + K\sqrt{w})$ 時間でオンライン構築するアルゴリズムを与える（定理 1）．ここに，w は計算機のレジスタ長 (ビット) であり，δ はハフマン符号の符号語の総ビット数である．

一般の有限な接頭辞符号に対しては，長さ N 文字で，K 個の符号語からなるテキストの疎接尾辞木を $O(\delta)$ 前処理時間と $O(K+\delta)$ 語の領域を用いて，$O(\lceil \frac{N \log \sigma}{w} \rceil \sqrt{w} + K\sqrt{w} + K \log \sigma)$ 時間で構築可能である（定理 2）．ここに，$\sigma = |\Sigma|$ は基本アルファベット Σ のサイズであり，δ は Δ の符号語の総文字数である．最後に，無限接頭符号である単語アルファベットについても，同様の拡張ができることを示す（定理 3）．

1.3 本研究の貢献と関連研究

本研究の一般の有限な接頭辞符号に関する結果は [13] に対して，単語アルファベットに関する結果は [8] に対して，ビット並列および詰めこみ文字列技法を用いた高速化になっている．圧縮テキストからそのまま構築するので，圧縮パターン照合の索引構築版とも見なせる．

提案手法は，$K \leq O(\frac{N}{\sqrt{w}})$ のときに従来手法より高速であり，接尾辞木系索引のオンライン構築で初めて，$O(N)$ 時間より高速な構築を達成した．もし符号の平均符号長または索引区切りが $w/\log\sigma$ 文字程度なら，定理 2 は $O(N \log \sigma)$ ビット領域索引の $O(N \log \sigma/\sqrt{w})$ 時間構築法を与える．

パターン照合の高速化においては，ビット並列化手法 [10] と圧縮パターン照合手法 [5] が広く研究されてきた．最近，入力をまとめ読みすることで高速化を図る**詰めこみ文字列手法**（packed string matching）による $o(N)$ 時間への高速化の研究が盛んである（例 [2]）．索引構築について，詰めこみ文字列手法による研究はほとんどなく，接尾辞構築に関しては本結果が最初の結果の一つだと思われる．

2. 準備

基本的な定義を [4], [6], [14] に従って与える．読者が，接尾辞木と Ukkonen によるそのオンライン構築アルゴリズム [14] の基礎的な知識を持っていると仮定する．

2.1 基本的な定義

文字列を構成するアルファベットを Σ とする．また長さ 0 の文字列を**空文字列**といい ε で表す．Σ^* と Σ^+ はそれぞれ Σ 上の**空文字列を含む有限文字列の集合**と**非空の有限文字列の集合**を表す．Σ 上の文字列 $T = a_1 \cdots a_N \in \Sigma^*$ の長さは $|T|_\Sigma = |T| = N$ であり，すべての $i = 1, \ldots, N$ に対し T の i **番目の文字**を $T[i] = a_i \in \Sigma$ と表す．任意の $1 \leq i \leq j \leq N$ に対し，T の i **番目から** j **番目の部分文字列**を $T[i..j] = a_i \cdots a_j$ と表す．このとき，$i > j$ ならば $T[i..j] = \varepsilon$ である．また，i と j をそれぞれ**開始位置**および**終了位置**と呼ぶ．

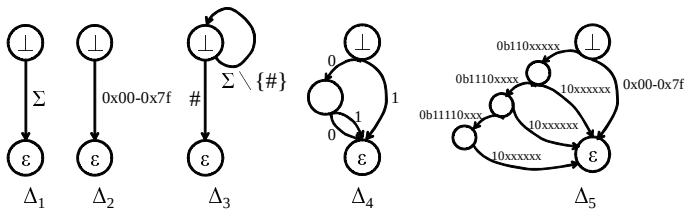

図 1 例 1 の接頭符号 Δ_1 と Δ_2，Δ_3，Δ_4，Δ_5 に対するコードオートマトンである．ここで，$\perp$ と ε はそれぞれ初期状態と最終状態である．

ある文字列 $T \in \Sigma^*$ に対して，$T = xyz$ となる $x, y, z \in \Sigma^*$ が存在するとき，x, y, z をそれぞれ T の**接頭辞**，**部分文字列**，**接尾辞**という．文字列の集合 $\mathcal{S} \subseteq \Sigma^*$ の**接頭辞の集合**と**真の接頭辞** *(Proper Prefix)* **の集合**をそれぞれ $\mathsf{Pre}(\mathcal{S})$ と $\mathsf{PropPre}(\mathcal{S})$ で表す．文字列 x, y の**最大の共通接頭辞** *(longest common prefix)* を $LCP(x, y)$ で表す．

2.2 Word RAM モデル

計算機の**レジスタ長（ワード長）**を w とする．1 つのレジスタに格納された整数 x を**ワード** (語) と呼ぶ．レジスタ長 w のワード x が取りうる値の範囲は $0 \leq x \leq 2^w - 1$ となる．このとき，x をビット列 $x = x_{w-1} \cdots x_0$ と表現するとき $x = \Sigma_{i=0}^{w-1} x_i 2^i$ となる．Word RAM モデルでは 2 つのワード x と y に対する**加算**と**減算**，論理演算 *AND*, *OR*, *NOT*，**左シフト**, **右シフト**，連続する w ビットに対する入出力をすべて定数時間で実行可能とみなす．本論文のアルゴリズムでは，入力長 N に対して $\log N = O(w)$ が成立すると仮定する[*1]．乗算は使用しない．

2.3 接頭符号

基本文字のアルファベットを Σ とおく．**符号**は，Σ 上の空でない文字列の集合 $\Delta \subseteq \Sigma^+$ である．各要素 $W \in \Delta$ を**符号語**と呼ぶ．符号 Δ は無限もしくは有限集合である．任意の符号語が他の符号語の接頭辞になってないとき，符号 Δ は *prefix-free* であるといい，そのような Δ を**接頭符号**と呼ぶ．符号が**正則**（regular）であるとは，それを受理する決定性有限オートマトン（DFA）が存在することをい

[*1] 代表的な Word RAM 上のアルゴリズム技法である表引き法（4 人のロシア人技法）では，より強い条件 $\log n = \Theta(w)$ を仮定するが，われわれの手法ではこれは仮定しない．

う．有限な符号はすべて正則であるが，逆は真ではない．

例 1 Σ を文字アルファベット，$\mathbb{B} = \{1,0\}$ を 2 進アルファベットとする．文字集合 $\Delta_1 = \Sigma$ と *ASCII* **コード** $\Delta_2 = [0x00 - 0x7f] \subseteq \mathbb{B}^8$ は固定長の接頭符号である．区切り文字 $\sharp \notin \Sigma$ を用いた**単語アルファベット** $\Delta_3 = \Sigma^+\sharp$ ([1], [8]) と，出現確率 $p(A) = 1/4, p(B) = 1/4, p(C) = 1/2$ をもつ記号集合 $\{A, B, C\}$ の**ハフマン符号** $\Delta_4 = \{00, 01, 1\}$([4])，**3 バイトに制限した** *UTF-8* 符号 $\Delta_5 = (10\mathbb{B}^1) \cup (110\mathbb{B}^2) \cup (1110\mathbb{B}^3)$ ([7]) は，可変長の接頭符号の例である．無限符号 Δ_3 を除く，その他の符号 $\Delta_1, \Delta_2, \Delta_4, \Delta_5$ は有限符号である．図 1 に示すように，上記の五つの符号はすべて正規である

2.4 コードオートマトン

図 1 で示したように Σ 上の有限接頭符号 Δ は決定性有限オートマトン (DFA) で受理することができる．この DFA を，**コードオートマトン**といい，CFA^Δ で表す．その初期状態と最終状態をそれぞれ $[\bot]$ と $root = [\varepsilon]$ で表す．CFA^Δ の状態数は，Δ の符号語の総文字数 $\delta = ||\Delta|| = \sum_{W \in \Delta} |x|$ で抑えられる．この論文で扱うコードオートマトンは最小である必要はない．コードオートマトン上のノードと，疎な接尾辞木上のノードを区別するために，前者を**コードノード**とよび，後者を**ツリーノード**と呼ぶ．コードノードの全体とツリーノードの全体を，それぞれ，$\mathsf{dom}(\mathrm{code})$ と $\mathsf{dom}(\mathrm{tree})$ で表す．

2.5 入力となる符号文字列

文字列 $T = W_1 \cdots W_K = t_1 \cdots t_N \in \Delta^*$ を Δ 上の**符号文字列** (もしくは Δ **文字列**) と呼ぶ．ここに，各 $W_j \in \Delta$ $(1 \le j \le K)$ は j 番目の符号語であり，各 $T[i] = t_i \in \Sigma$ $(1 \le i \le N)$ は i 番目の基本文字である．本稿では，詰め込み文字列手法を用いるにあたって，次の仮定をおく．

定義 1 (オンライン入力の仮定) 入力の符号文字列 T は，先頭から順に w ビットずつレジスターに詰められて，アルゴリズムに与えられる．

任意のインデックス $j = 1, \ldots, K$ に対して，文字列 T の j 番目の Δ **接尾辞** $\mathsf{suf}_j^\Delta(T)$ を j 番目の符号の境界から始まる T の接尾辞と定義する．すなわち $\mathsf{suf}_j^\Delta(T) = W_j \cdots W_{K-1} W_K$ となる．もし T がある Δ 文字列の真の接頭辞ならば，末尾の W_K はある符号語の真の接頭辞になる．

2.6 接頭符号上の疎接尾辞木

上村ら [13] に従って，接頭符号上の疎接尾辞木を導入する．テキスト T とその接尾辞の集合 Suf が与えられたとき，Suf に対する T の**疎接尾辞木**（sparse suffix tree）とは，Suf のすべての接尾辞を格納するパス圧縮トライである [9]．Δ を任意の接頭符号とし，T をその上の K 個の符号からなる N 文字の符号化文字列とする．T に対する**符号上の疎接尾辞木**（または**符号接尾辞木**）は，Δ 接尾辞の集合 $Suf^\Delta(T) = \{\, \mathsf{suf}_j^\Delta(T) \,|\, j = 1, \ldots, K+1 \,\}$ に対する疎接尾辞木として定義される [13].

ここに，疎接尾辞木 CST^Δ の各有向辺 $e = (u, v)$ は，T の部分文字列 $Label(v) \in \Sigma^+$ を辺ラベルとしてもつ．各文字 $a \in \Sigma$ に対して各内部ノード v は，ラベルが a からはじまる出辺をたかだか 1 つだけもつ．a から始まる辺によってつながれた子を a-*child* とよび，$u = child(v, a)$ とかく．根を除く内部ノードは，すべて**分岐**している．すなわち少なくとも 2 つの子をもつ．各ノード v に対し，$L(v)$ を根から v までのパス上のすべてのラベルを連結して得られる文字列とする．もし T の末尾 $T[N]$ が末尾以外に現れない一意な記号なら，T のすべての Δ 接尾辞は $\mathsf{CST}^\Delta(T)$ の葉で表現される．

明らかに $\mathsf{CST}^\Delta(T)$ は最大 K 個の葉と $K-1$ 個の内部ノードをもつ [9]．領域の節約のため $Label(v)$ を $T[j..i] = Label(v)$ となるような T におけるラベルの開始位置と終了位置のペア $\langle j, i\rangle \in \mathbb{N}^2$ で表現する．空語 ε は任意の i に対して $\langle i+1, i\rangle$ で表現する．これにより，$\mathsf{CST}^\Delta(T)$ は $O(K)$ 語の領域で表せる．

ワード境界上から始まる T の任意の部分文字列 α が $\alpha = L(v) \cdot T[j..i]$ で表現されるとき，その位置を**ポインタ** (もしくは**参照**) と呼ぶ三つ組 $p = \langle v, j, i\rangle \in V \times \mathbb{N}^2$ で表現する．$T[j..i]$ を最も短く表現しているときポインタ $\langle v, j, i\rangle$ は**正規形**であるという．部分文字列 α の**場所** *(locus)* は α の正則なポインタであり，$loc(\alpha)$ とかく．もし $j \le i$ すなわち $T[j..i] \neq \varepsilon$ のとき p を**仮想ノード**とよび，$j > i$，すなわち $T[j..i] = \varepsilon$ のとき p を**実ノード**と呼ぶ．どちらの場合も，ノード p のラベル文字列を $L(p) = L(u) \cdot T[j \cdots i]$ と定義する．実ノード p は接尾辞リンク (suffix link) と呼ばれる実ノードへのポインタを持つ．また，p には CST^Δ の根からの文字深さ $d(p) = |L(p)| \ge 0$ を格納する．

3. 基本的な構築アルゴリズム

本節では，提案アルゴリズムの基本となる正則接頭符号化テキストに対する疎接尾辞木の構築アルゴリズム ConstrutCST1 を導入する．以下では，Σ をサイズ σ の基本アルファベットとし，$\Delta \subseteq \Sigma^+$ を総文字数 $\delta = ||\Delta||$ の符号語をもつ有限な接頭辞符号とする．長さ N ビットかつ K 個の符号語からなる入力符号化テキスト T を仮定する．

本節と次節では簡便のため，アルファベット $\Sigma = \{0, 1\}$ に対するバイナリハフマン符号化テキストの場合を説明する．一般の正則接頭符号の場合も，木の分岐数と文字のビット長さ $b = \log \sigma$ の違い以外は，アルゴリズムは本質的に同じである．

この基本アルゴリズムは，上村ら [13] の ConstrutCST と

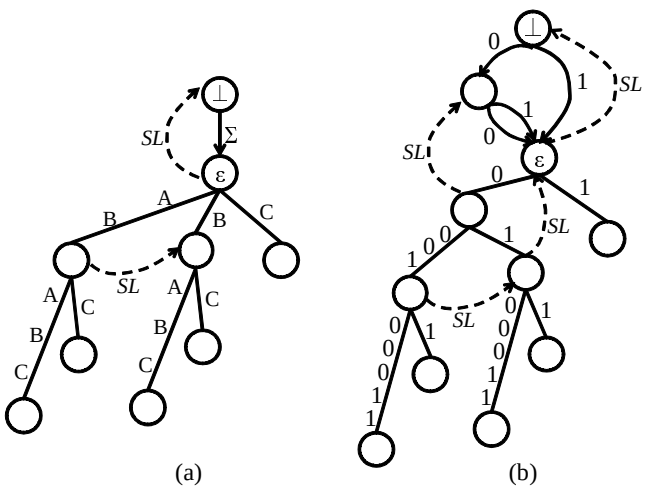

図 2 (a) アルファベット $\{A, B, C\}$ 上の文字列 $S = ABABC$ に対する通常の接尾辞木と (b) 接頭符号 $\Delta = \{A/00, B/01, C/1\}$ 上の符号文字列 $T = 000100011$ の符号に対する疎な接尾辞木の例.

基本的には同一であるが，手続き MatchNode を導入することで，枝上の複数回の文字遷移を一括して行うように修正されている．これに，次節の簡潔データ構造による高速化を組み合わせることで，高速化を図る．

3.1 接尾辞リンク

初めに数学的な準備として，文字列でない特別な要素 $\perp \notin \Delta$ (ボトム) を導入する．これは，任意の符号語 $W \in \Delta$ に対して $\perp W = \varepsilon$ が成立する特別な要素と定義する．この $\perp$ は，いわば符号語の**逆元**であり，文字列に左から作用し，先頭の符号語を消去する演算を表す．

次に文字列 $\alpha \in \Sigma^* \cup \perp\Sigma^*$ に対して，木 CST^Δ とコードオートマトン CFA^Δ 内の**場所** $[\alpha]$ を，次のように対応付ける．(i) もし $U \in \Sigma^*$ がふつうの文字列ならば，$[\alpha]$ は木 CST^Δ の根から文字列 α で到達可能な一意なノードを表す．(ii) もし $U \in \Delta\Sigma^*$ が符号語 W と文字列 α に対して，$U = W\alpha$ の形ならば，$[\perp U]$ は $\perp U$ を簡約して得られる文字列 $\perp W\alpha = \alpha$ の CST^Δ 内の場所である．これは接尾辞リンクに対応する．(iii) もし $U \in \mathsf{PropPre}(\Delta)$ がある符号語の真の接頭辞の形ならば（接頭符号なので U は符号語になり得ない），場所 $[\perp U]$ はコードオートマトン CFA^Δ の内部状態を表し，初期状態 $[\perp]$ から文字列 α で到達可能なノードである．

以上の準備の元で，CST^Δ の接尾辞リンクが定義できる．木 CST^Δ のツリーノード v は，そのラベルの先頭から符号語を一つ除いた文字列をラベルに持つノードへの接尾辞リンクをもつ．すなわち，$SL^\Delta(v) = loc(\perp L(v))$ が成立する．ツリーノード v のラベルが先頭に完全な符号語を含むならば $SL^\Delta(v)$ は再びツリーノードである．一方で，ツリーノード v のラベルが符号語の真の接頭辞ならば $SL^\Delta(v)$ はコードノードである．コードオートマトン CFA^Δ のコードノードは接尾辞リンクをもたない．

Algorithm ConstructCST1:
入力: 接頭符号 $\Delta \subseteq \Sigma$ 上の文字列 $T = w_1 \cdots w_K \in \mathsf{Pre}(\Delta^*)$
出力: Δ に関する T の疎な接尾辞木 $\mathsf{CST}^\Delta(T)$;

1: root node $root = [\varepsilon]$ のみを持つ空の木 CST^Δ を作成する;
2: source $\hat{\perp} = [\perp]$ と sink $root = [\varepsilon]$ である Δ に対するコードオートマトンを作成する.
3: $SL^\Delta(root) = \hat{\perp}$;
4: $\phi \leftarrow \langle root, 1, 0\rangle$; // $\phi = \langle s, m, l\rangle$;
5: $i \leftarrow 1$;
6: **while** $i \le N$ **do**
7: $\quad \phi \leftarrow \mathsf{MatchNode}(\phi, i)$;
8: $\quad \langle s, k, j\rangle \leftarrow \phi$;
9: $\quad$ **if** $k \le j$ **then** $p \leftarrow \mathsf{Split}(\phi)$;
10: $\quad$ **else** $p \leftarrow s$;
11: $\quad$ 新しい葉 q を作成;
12: $\quad child(p, T[j+1]) \leftarrow q$;
13: $\quad L(q) \leftarrow \langle j+1, \infty\rangle$;
14: $\quad \phi \leftarrow \mathsf{Canonize}(SL(s), k, j)$;
15: $\quad i \leftarrow j + 1$;
16: **end while**
17: **return** CST^Δ;

図 3 接頭符号 $\Delta \subseteq \Sigma^+$ 上の文字列 T に対する code suffix tree $\mathsf{CST}^\Delta(T)$ を作成する手続き.

Procedure Canonize:$(\phi = \langle s, k, j\rangle, i)$

1: **while** $j \le i$ **do begin**
2: $\quad u \leftarrow child(s, T[j])$;
3: $\quad \langle q, p\rangle \leftarrow label(u)$;
4: $\quad$ **if** $p - q > i - j$ **then**
5: $\quad\quad$ **break**;
6: $\quad j \leftarrow j + (p - q + 1)$;
7: $\quad s \leftarrow u$;
8: **end**
9: **return** $\langle s, j, i\rangle$; {End of Canonize}

図 4 参照ポインタを正規形にする手続き Canonize

3.2 主アルゴリズム

図 3 に，アルゴリズムの本体 ConstructCST1 を示す．また図 4 にサブルーチン Canonize を示す．このアルゴリズムの構成は，Ukkonen によるオリジナルの接頭辞アルゴリズム (Ukkonen と呼ぶ) とほぼ同じであり，上村らでは根にコードオートマトンを結合することで，符号上の疎接尾辞木構築に対応している．アルゴリズムは入力文字列 $T = T[1] \cdots T[N] = W_1 \cdots W_K W_{K+1} \in \mathsf{Pre}(\Delta^*)$ に対して，T 上の文字を先頭から順に読みながら，CST^Δ とコードオートマトン上のノードへの参照である**アクティブポイント** ϕ を更新し，$\mathsf{CST}^\Delta(T)$ をオンラインに構築する．

3.2.1 初期化

まずはじめに，根 $root = [\varepsilon]$ とコードオートマトン $\mathsf{DFA}(\Delta)$ のみをもつ CST を作成する．いま参照しているアクティブポイントをポインタ ϕ で表す．最初は根を参照するので，$\phi = \langle root, 1, 0\rangle$ である．

3.2.2 入力のまとめ読み

CST の更新は，CST$^\Delta$ に表現されていない部分文字列を必要なだけ追加することで行われる．Ukkonen によるアルゴリズムでは，与えられた入力 T を先頭から 1 文字ずつ読み込み処理を行う．それに対し提案手法では，T を不一致が起きるまでまとめて読み込み，アクティブポイントの参照 ϕ を移動させながら処理を行う．

これを行う手続きが MatchNode である．MatchNode は，仮想ノード ϕ から下方に延びるすべての枝を考え，そのラベル文字列の集合から，現在の入力 $T[i \cdots N]$ の接頭辞と最長一致するものを選択し，不一致箇所の仮想ノードへの参照を返す．（実装は 4.4 節で与える）素朴な実装は，読み進める文字数 N に対して，$O(N)$ 時間を要するが，次節では，ビット並列計算と簡潔トライ構造を用いて，Word RAM 上の高速なアルゴリズムを与える．

3.2.3 接尾辞拡張のタイプ

手続き MatchNode が不一致箇所の参照 ϕ を返したあとの処理は，以下の 3 つのタイプに分類することができる．これらの処理は，アクティブポイント ϕ が CST$^\Delta$ と CFA$^\Delta$ のどちらにあるかに関わらず，共通である．

- **タイプ 1**： ϕ が葉のとき LCP 長の計算結果は必ず 0 である．Ukkonen [14] は，葉ノードの辺ラベルを $\langle j, \infty \rangle$ と表現し，∞ は常に現在の入力 i を表すものとすることで自動的にエッジの延長を行う．ここに，定義 1 の仮定を利用して，挿入時の葉 ℓ の文字深さ $d(\ell)$ が常に w の整数倍になるよう待って，葉を挿入すると定める．ただし，これは，次節のマイクロ木分解で必要である．
- **タイプ 2**： Ukkonen では，現在の入力 $T[i]$ と ϕ から延びる辺ラベルの先頭文字に不一致が発生したときタイプ 2 と呼ぶ．このとき，MatchNode は不一致個所への参照ポインタ $\phi = \langle s, k, j \rangle$ を返し，ϕ が仮想ノードならば，手続き Split を用いて仮想ノード ϕ を実ノード p に変換し，ϕ が以前に指したノードから p へ新しく接尾辞リンクを張る．次に参照ポインタの位置に，新たに作成した葉ノードを連結する．このときの葉のラベルはタイプ 1 で述べたように $\langle j+1, \infty \rangle$ とする．タイプ 2 が発生した場合は接尾辞リンクを用いて ϕ を更新して，引き続き，MatchNode の実行と，その後の接尾辞拡張を行う．
- **タイプ 3**： Ukkonen では，現在の入力 $T[i]$ と ϕ から延びる辺ラベルの先頭文字が一致したときタイプ 3 とよび，参照 ϕ を更新することで処理を終了し，次の入力を読み込む．提案手法では，手続き MatchNode を用いるため，常に不一致を検出するので，このタイプの拡張は実際にはタイプ 2 に統合される．

提案手法のアルゴリズムは，以上の過程を繰り返すことで CST$^\Delta$ を構築する．このアルゴリズムは，タイプ 3 で入力をまとめて読み込む以外は上村ら [13] のアルゴリズムと本質的に同じ動作をする．よって，[13] と同様に $O(N \log \sigma)$ 時間を要する．この解決に，次節では簡潔データ構造を用いた高速化を行う．

4. Word RAM 上の高速化

本節では，有限接頭符号に対する符号化テキストの疎接尾辞木に対して，簡潔トライ QFT をノードアクセスに対する索引として用いて，ノード巡回の高速化を行う手法を提案する．これにより，前節の基本アルゴリズムの時間計算量を改善し，ハフマン符号化テキストの場合に Word RAM 上で，$O(\lceil \frac{N}{w} \rceil \sqrt{w} + K\sqrt{w})$ 時間を達成できる．

以下の説明では，入力はアルファベットサイズ $\sigma = 2$ のハフマンテキストの場合を考える．一般の有限接頭符号の場合も，CST$^\Delta$ が σ 分木であり，高速化率として $\rho = w/\log \sigma$ を用いる他は同様である．CST$^\Delta$ の総ノード数を K とする．

4.1 簡潔トライ QFT（q-fast trie）

簡潔トライである q-fast trie (以後，QFT と呼ぶ) は，Willard [16] によって提案された動的データ構造であり，最大値が $M \geq 0$ 以下である N 個の非負整数の集合 S を $O(N)$ 語で格納し，次の演算を一演算あたり $O(\sqrt{\log M})$ 時間で実現する．

- MEMBER(x): キー x の S への所属の有無を返す．
- PREDECESSOR(x): キー x を超えない S のキーの最大値を返す．
- SUCCESSOR(x): キー x 以上となる S のキーの最小値を返す．
- INSERT(x): 新しいキー x を S に挿入する．
- DELETE(x): 既存のキー x を S から削除する．

キーはすべて長さ w ビット以下の二進数と仮定する．以下では，PREDECESSOR(x) と，SUCCESSOR(x)，INSERT(x) を用いる．より高速な $O(\log \log M)$ 演算を許す Willard [15] の y-fast trie は挿入削除ができないのに対して，QFT はそれが可能である点に特徴がある．

4.2 木のマイクロ木分割

今，Word RAM のレジスター長 w に対して，ブロック長を w とおく．初めに，疎接尾辞木 CST$^\Delta$ において，文字深さ，すなわち，根からの距離 $d(v)$ が w の整数倍，つまり $0, 1w, 2w, \ldots, iw, \ldots$ の頂点 v（仮想または実頂点）を**索引候補点**（w 区切り点）と定める．整数 i を**レベル**とよぶ．索引候補点自体は，$O(N^2)$ 個存在することに注意．

次にレベル i の索引候補点 v に対して，v を根とし，高さがちょうど w 文字の CST$^\Delta$ の部分木 $T(v)$ を考えて，根 v をもつレベル i の**マイクロ木**と呼ぶ．マイクロ木 $T(v)$ において，根 v は深さ $d(v) = i$ をもち，すべての内部ノード u は深さ $d(u) \in [iw, (i+1)w]$ をもつ．3.2.3 節のタイプ 1

Procedure BNA(α, c)
データ構造: q-fast trie QFT;
入力: w ビット文字列 α, 整数 $c \geq 0$;
出力: マイクロ木 $T(v)$ 中の $\alpha[1..c]$ が指す場所の直上の分岐ノード;
1: $\beta_L \leftarrow \alpha[1..c]0^{w-c}$; $\beta_R \leftarrow \alpha[1..c]1^{w-c}$;
2: $\ell_L \leftarrow QFT$.PREDECESSOR(β_L);
3: $\ell_R \leftarrow QFT$.SUCCESSOR(β_R);
4: $p_L \leftarrow NCA(\ell_L, \ell_{L+1}) =$ NCA(ℓ_L);
5: $p_R \leftarrow NCA(\ell_{R-1}, \ell_R) =$ NCA(ℓ_{R-1});
6: **if** $d(p_L) \leq d(p_R)$ **then** $p \leftarrow p_L$ **else** $p \leftarrow p_R$;
7: **return** p;

図 5 マイクロ木 $T(v)$ 中の $\alpha[1..c]$ が指す場所の直上の分岐ノードを見つける手続き

Procedure MatchNode(QFT, i)
1: $\alpha = T[i..i+w-1]$; //現在の入力文字列
2: $\ell_\alpha \leftarrow QFT$.PREDECESSOR($\alpha$); //先行する葉
3: $len \leftarrow LCP(\alpha, \mathsf{Lab}(\ell_\alpha))$; //不一致点の深さ.
4: $p \leftarrow QFA$.BNA(α, len); //性質：$d(\phi) = d(v) + len$
5: $k \leftarrow i + d(p) - d(v)$; $j \leftarrow i + len$;
6: **return** $\phi = (p, k, j)$;

図 6 マイクロ木 $T(v)$ 中の文字列 $\alpha = T[i..i+w-1]$ の不一致頂点 ϕ を見つける手続き

の仮定より，すべての葉 ℓ は深さ $d(\ell) = (i+1)w$ をもつ．ただし，葉には仮想と実ノードの両方があり得る．

明らかに，マイクロ木どうしは根と葉以外は共有せず，その実頂点の総数は $O(K)$ である．

4.3 マイクロ木 $T(v)$ の索引化

レベル i のマイクロ木 $T(v)$ を考える．$T(v)$ 中のすべての葉を $\ell_1, \ldots, \ell_m$ とする．根 v から葉 ℓ_i への枝のラベル文字列を $\mathsf{Lab}(\ell_i)$ で表す．これらの葉は，$\mathsf{Lab}(\ell_1) <_{\mathrm{lex}} \cdots <_{\mathrm{lex}} \mathsf{Lab}(\ell_m)$ と辞書順で昇順に整列されていると仮定する．各 $i \in [1, m-1]$ に対して，葉 ℓ_i と ℓ_{i+1} の**最近共通先祖**（NCA, nearest common ancestor）とは，ℓ_i と ℓ_{i+1} の共通の先祖で，最も下方にあるものをいい，$NCA(\ell_i, \ell_{i+1})$ で表す．NCA は常に分岐ノードである．3.2.3 節のタイプ 1 の仮定より，$\mathsf{Lab}(\ell_i)$ の長さは常に w である．

4.3.1 索引化の手順

最初に，CST$^\Delta$ のすべてのノード p に対して，索引フラグを $indexed(p) = 0$ と設定しておく．このとき，CST$^\Delta$ の各マイクロ木 $T(v)$ を次のように分類する．(1) $T(v)$ が分岐頂点を含まないときは，何もしない．(2) $T(v)$ が 1 個以上の分岐頂点を含むとき，$T(v)$ を以下の手順で索引化する：

(1) マイクロ木 $T(v)$ の葉を，それが仮想ノードであれば実ノード化する．
(2) $QFT(v)$ を新しく生成した q-fast trie とし，INSERT 演算でラベル $\mathsf{Lab}(\ell_i)$ をキーとして，$T(v)$ の葉すべてを挿入する．
(3) 各 $i \in [1, m-1]$ に対して，葉 ℓ_i の欄 NCA(ℓ_i) として，最近共通先祖 $NCA(\ell_i, \ell_{i+1})$ を対応付ける．

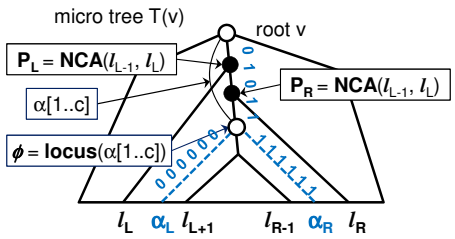

図 7

CST$^\Delta$ のオンライン構築では，新しい実頂点 p の導入毎に，キー $\mathsf{Lab}(\ell)$ による葉 ℓ の挿入を行い，p から関連データ構造へのポインタをもたせてオンライン化する．

図 5 に，マイクロ木内でキー文字列の最長一致（前方一致）検索を高速に行うための鍵となる手続き BNA を示す．これは，簡潔トライ QFT によるビット検索と，マイクロ木の葉に格納した NCA 情報を組み合わせて，木内部の分岐数に依存しない高速な検索を実現する．

補題 1 手続き BNA は，$T(n)$ 中の仮想ノードの参照 ϕ から最近分岐先祖を $O(\sqrt{w})$ 時間で見つける．

証明: 図 7 を参照のこと．一般に，根付木では並んだ三つの葉 ℓ_1, ℓ_2, ℓ_3 の二組の隣接対の NCA が木の中に存在するなら，それらは同じ枝に含まれ，比較可能である．$T(v)$ の葉 $\ell_L, \ell_{L+1}, \ell_{R-1}, \ell_R$ に対してもこれが成立する．QFT での前者と後者の存在と，ϕ の存在から，結果が示される．■

図 6 の手続き MatchNode0 は，BNA をサブルーチンに用いて，入力テキスト $\alpha = T[i..i+w-1]$ を最大 w ビット読み進めたときに，$T(v)$ 中の文字列で不一致が起こる場所の参照を $O(\sqrt{w})$ 時間で計算する．

仮想ノード ϕ での不一致の際は，タイプ 2 拡張により ϕ を実ノード p に変換し，続けて，新しい葉 ℓ の生成と，キー $\mathsf{Lab}(\ell) = \alpha$ の QFT への挿入，葉の NCA 情報の更新，フラグ $indexed(p) = 1$ の設定等のデータ構造の管理を行う．

4.4 索引を用いた木の巡回の高速化

以下の手続き MatchNode は，前節の図 6 の手続き MatchNode0 を繰り返し用いることで，入力テキスト $\alpha = T[i..N]$ を制限なしに読み進めたときに，CST$^\Delta$ 中の文字列で不一致が起こる場所の参照を計算する．

手続き MatchNode(ϕ, i)：

- **STEP 1:** 現在のノード $\phi = (p, k, j)$ において，フラグ $indexed(p)$ をテストし，次のいずれかに行く．
- **STEP 2:** もしフラグ $indexed(p)$ が 0（非分岐）ならば，v から出る唯一の枝の文字列 $Y = T[k..j]$ に対して，$l = LCP(X, Y) \leq w$ を計算する．もし $l = w$ ならば，上記の過程を繰り返す．もし $l < w$ のとき，不一致点 ϕ が見つかったら出力して停止する．それ以外は，到達した枝の終端 e を v とおき，**STEP 1** へ行く．
- **STEP 3:** もしフラグ $indexed(p)$ が 1（分岐）ならば，

対応する索引 $QFT(p)$ と現在の入力位置 i から，手続き MatchNode0 を用いて，マイクロ木中の不一致 ϕ を見つける．もし $l < w$ のとき，必ず不一致点 ϕ が見つかるので，ϕ を出力して停止する．もし $l = w$ ならば，y 到達した $T(v)$ の葉を v とおき，**STEP 1** へ行く．

以上の議論をまとめると，次のことが言える．

補題 2 手続き MatchNode が N' 文字を読み，K' ノードを訪問したと仮定する．このとき，手続きは $O(\lceil \frac{N'}{w} \rceil \sqrt{w} + K'\sqrt{w})$ 時間を要する．$\diamond$

4.5 コードオートマトンの索引化と高速化

有限接頭符号 $\Delta \subseteq \Sigma^+$ に対して，Δ の符号語に対する非圧縮トライ C を構築し，4.2 節のマイクロ木分割と 4.3 節の QFT を適用して索引化する．コードトライのノード数は $O(\delta)$ である．トライ内の巡回には，前節で導入した手続き MatchNode を用いる．コードトライにおいては，タイプ 3 拡張しかないので，実行が STEP3 に限定されている以外は，CST^Δ の巡回とまったく同じあり，補題 2 が CFA^Δ においても成立する．

5. 計算量のみつもり

提案手法の領域計算量と時間計算量を解析する．テキストの総ビット数を N とし，その符号語の数を K とする．Δ の符号語の総文字数を δ とおく．このとき，CST^Δ と CFA^Δ の実ノード数は，それぞれ，$O(K)$ と $O(\delta)$ である．

5.1 領域計算量解析

まず各マイクロ木 (QFT) の領域計算量を求める．まず，CST^Δ のノード数は高々 $2K - 1 = O(K)$ 個である．すべてのマイクロ木の頂点は高々 2 回しか重複しないので，マイクロ木中の実ノード数の総和 $O(K)$ である．このとき，マイクロ木の葉の挿入によって生成された実ノード数の増加は，高々 $O(K)$ である．なぜならば，葉が生成される度に唯一の子孫にチャージしてならし解析することで，新たに作られる実頂点数は $O(K)$ で抑えられるからである．よって，CST^Δ とコードトライの総ノード数は，それぞれ $O(K)$ 個と $O(\delta)$ 個である．一方，簡潔トライ QFT の領域は格納する w ビット超のキー数に線形であるので，アルゴリズムの使用領域は $O(K + \delta)$ 語となる．

5.2 時間計算量解析

前処理の CFA^Δ の構築は $O(\delta)$ 時間でできる．これまでの議論から，明らかに，(a.1) CST^Δ の非分岐ノードから長さ w 以上の枝をたどる回数と，(a.2) CST^Δ の分岐ノードで MatchNode0 を計算する回数，(a.3) CFA^Δ で MatchNode0 を計算し読み進める回数は，合計で $O(\lceil \frac{N}{w} \rceil)$ 回となる．

一方，(b.1) 非分岐ノードから長さ w 以下の枝をたどる回数と，(b.2) 不一致で新ノードを追加する回数，(b.3) 接尾辞リンクをたぐるときに実ノードを訪問する回数，(b.4) CFA^Δ で符号語の残りを読みとばす回数は合計で $O(K)$ 回でおさえられる．ここで，(b.1) と (b.4) は自明である．(b.2) は，不一致をするたびに新しい実ノードが追加され，さらに，生成されるノード数の総和は K 以下であることから容易に示される．(b.3) は，Ukkonen [14] の議論から，ならし解析によって示される．

それぞれに対して，MatchNode 手続きによる LCP の計算に $O(\sqrt{w})$ 時間がかかる．よって，本稿の主結果が示される．

定理 1 (ビットハフマン符号化テキスト) 図 3 のアルゴリズム ConstructCST1 は，長さ N ビットかつ K 個の符号語からなるハフマン圧縮テキストが入力として与えられたとき，その疎接尾辞木を Word RAM 上で $O(\delta)$ 前処理時間と $O(K + \delta)$ 語の領域を用いて，$O(\lceil \frac{N}{w} \rceil \sqrt{w} + K\sqrt{w})$ 時間でオンライン構築する．ここに，w は計算機のレジスタ長 (ビット) であり，δ は符号語の総ビット数である．$\diamond$

本結果は，次のように一般の有限な接頭辞符号に対しても，拡張可能である．ここに，δ を Δ の符号語の総サイズとする．

定理 2 (一般の有限な接頭符号化テキスト) アルファベット Σ 上の任意の有限な接頭符号 $\Delta \subseteq \Sigma^+$ に対して，長さ N 文字かつ K 個の符号語からなるテキストの疎接尾辞木を Word RAM 上で $O(\delta)$ 前処理時間と $O(K + \delta)$ 語の領域を用いて，$O(\lceil \frac{N \log \sigma}{w} \rceil \sqrt{w} + K\sqrt{w} + K \log \sigma)$ 時間でオンライン構築可能である．ここに，$\sigma = |\Sigma|$ は基本アルファベットのサイズであり，δ は Δ の符号語の総文字数である．$\diamond$

提案手法は，$K \leq O(\frac{N}{\sqrt{w}})$ のときに，従来の $O(N)$ 時間アルゴリズム [13] を改善している．上記の二つの結果では，Word RAM の加算を用いるが，乗算は用いない．

6. 無限の接頭符号：単語アルファベットの場合

本節では，無限の正則接頭符号である単語アルファベットに対する疎接尾辞木構築の高速化を考察する．単語アルファベットは，無限接頭符号 $\Delta_3 = \Sigma^+\#$ ($\# \notin \Sigma$) であるので，簡潔トライを用いる前節の手法は使えない．$\Sigma^+\#$ を受理するコードオートマトン [8] (図 1) は，本質的には，単語の途中で CST^Δ の巡回に失敗したときに，可変個の未読文字を，単語の終わりの空白まで読み飛ばす演算を行う．

図 8 に，ビット並列手法を用いて，コードオートマトンを模倣する手続き SkipToSpace を示す．説明の準備として，ワード長 w と文字幅 b はある整数 $m \geq 1$ に対して $w = mb$ を満たすと仮定する．これにより，w ビットのマスク $X \in \{0,1\}^w$ を，文字ブロック $X_i \in \{0,1\}^b$ のベクトル $X = X_1 \cdots X_m \in (\{0,1\}^b)^m$ と見なす．CST^Δ 構築アルゴリズムが，$i \in [1, N - w]$ 番目の文字まで読み，タイプ

Procedure SkipToSpace($i \in [1, N]$, w, $T[1..N]$)
1: $X \leftarrow T[i..i+w-1]$;
2: $Z \leftarrow X \oplus (\sim \sharp^m)$;
3: $Z0 \leftarrow (Z \gg m-1)$;
4: $Z \leftarrow Z \,\&\, (\sim HIGH) + Z0$;
5: $Z \leftarrow (Z \,\&\, HIGH) \gg m-1$;
6: $P \leftarrow MSB(Z)$;
7: $i \leftarrow i + \lceil P/m \rceil + 1$;

図 8 単語オートマトンを模倣し，単語の終わりの空白まで未読文字を読み飛ばす手続き

2 の接尾辞拡張により，アクティブポイント ϕ が CFA^{Δ} のコードノードを指したと仮定する．

手続き SkipToSpace を説明する．まず 1 行目で，まず現在のテキスト w ビットをレジスタ X に格納する（1）．次に 2 行目で，空白記号 “♯” のビット反転を並べたマスク $A = \sim \sharp^m$ を用いて，空白記号 ♯ が出現したブロックを 1^m で埋める．次の 3 行目と 4 行目で，加算によるキャリー伝搬を使い，1^m で埋まったブロックを検出し，その最上位ビットに 1 を立てる．5 行目と 6 行目では，検出したビットの位置を合わせて，MSB を $O(\log w)$ 時間で計算して読み飛ばすビット数 P を得て，ポインタを進める．

以上の計算は，加算をもつ Word RAM 上で，$O(\log w)$ 時間で実現可能であるので，前節の結果と組み合わせて次が示せる．

系 3 (単語接尾辞木) Σ 上の単語アルファベット $\Delta = \Sigma^+\#$ に対して，長さ N 文字で，K 個の符号語からなるテキストの疎接尾辞木を Word RAM 上で $O(1)$ 前処理時間と $O(K)$ 語の領域を用いて, $O(\lceil \frac{N \log \sigma}{w} \rceil \sqrt{w} + K\sqrt{w} + K \log \sigma)$ 時間でオンライン構築可能である．ここに，$\sigma = |\Sigma|$ は基本アルファベットのサイズである．$\diamond$

この結果は，[1], [8] の単語接尾辞木のオンライン線形時間構築に対する詰めこみ文字列技法を用いた高速化を与える．

7. 結論

本稿では, 符号化上のテキストに対する疎接尾辞木の高速化手法を与えた．結果として, ビット並列計算と簡潔トライ構造を用いて, σ 進文字上の総サイズ δ の接頭符号において, 長さ N 文字かつ K 個の符号語からなる符号化テキストに対する疎接尾辞木を，語長 w ビットの Word RAM 上で $O(\delta)$ 前処理と，$O(K+\delta)$ 語領域, $O(\lceil \frac{N}{w} \rceil \sqrt{w} + K\sqrt{w} + K \log \sigma)$ で構築するアルゴリズムを与えた．さらに，単語接尾辞木等に対する一般化についても議論した．

最近, Kolpakov ら [11] は, Chien ら [3] らの簡潔索引のアイディアを利用し, $O(K) = O(N/\log_2 N)$ 語 $= O(N)$ ビットの領域量をもつ疎接尾辞木に基づく簡潔索引を提案している．この索引において, 文字列検索はビット並列手法を用いて $O(M\lceil \log N \log \sigma / w \rceil + (\log N / \log\log N)(\log N + occ))$ と効率化されているが，索引構築は高速化の工夫があまりされておらず，依然として $O(N)$ 時間を要する．そこで，本稿の手法を用いて，領域 $O(N)$ ビットで $o(N)$ 時間の索引構築が可能かどうかは興味深い問題である．

6 節の単語アルファベット上の疎接尾辞木構築が，一般の正則接頭符号に拡張可能などうかは課題である．

謝辞

本研究を進めるにあたり，議論とコメントをいただいた竹田正幸教授と篠原歩教授，稲永俊介准教授，平田耕一教授，喜田拓也准教授に感謝いたします．本研究は文部科学省基盤研究 (A), 24240021, FY2012–2015 の支援による．

参考文献

[1] A. Andersson, N. J. Larsson, and K. Swanson, Suffix trees on words, *Algorithmica*, 23(3), 246–260, 1999.
[2] O. Ben-Kiki, P. Bille, D. Breslauer, L. Gasieniec, R. Grossi, O. Weimann, Optimal Packed String Matching, *Proc. FSTTCS 2011*, LNCS, 423–432, 2011.
[3] Y. F. Chien, W.-K. Hon, R. Shah, J. S. Vitter. Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing, *Proc. 2008 Data Compression Conference (DCC'08)*, 252 - 261, 2008.
[4] M. Crochemore and W. Rytter, *Jewels of Stringology: Text Algorithms*, 2002.
[5] M. Farach and M. Thorup, String matching in Lempel-Ziv compressed strings, *Proc. STOC'95*, 703-712, 1995.
[6] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, – Computer science and computational biology, Cambridge, 1997.
[7] IETF, UTF-8, RFC 3629, 2003. `http://tools.ietf.org/html/rfc3629`
[8] S. Inenaga and M. Takeda, On-line linear-time construction of word suffix trees, *Proc. CPM'06*, LNCS, 60–71, 2006.
[9] J. Kärkkäinen and E. Ukkonen, Sparse suffix trees, *Proc. COCOON'96*, LNCS, Springer, 219–230, 1996.
[10] Y. Kaneta, H. Arimura, and R. Raman Faster bit-parallel algorithms for unordered pseudo-tree matching and tree homeomorphism, Journal of Discrete Algorithms, 14, 119 - 135, 2012.
[11] R. Kolpakov, G. Kucherov, T. Starikovskaya, Pattern matching on sparse suffix trees, *Proc. CCP'11*, 92-97, 2011.
[12] E. M. McCreight, A space-economical suffix tree construction algorithm, *J. ACM*, 23, 262–272, 1976.
[13] T. Uemura, H. Arimura, Sparse and truncated suffix trees on variable-length codes, *Proc. CPM'11*, LNCS 6661, Springer, 246-260, 2011.
[14] E. Ukkonen, On-line construction of suffix-trees, *Algorithmica*, 14(3), 249-260, 1995.
[15] D. E. Willard, Log-logarithmic worst-case range queries are possible in space Θ(N), *Information Processing Letters*, 17, 81-84, 1983.
[16] D. E. Willard, New Trie Data Sturucture Which Support Very Fast Search Operations, *Journal of Computer And System Sciences*, 28, 379-394, 1984.