プロパティ接尾辞木のオフライン線形時間構築アルゴリズム

上村　卓史[†a)]　　喜田　拓也[†]　　有村　博紀[†]

A Linear-Time Off-Line Construction of Property Suffix Trees

Takashi UEMURA[†a)], Takuya KIDA[†], and Hiroki ARIMURA[†]

あらまし　プロパティ付きテキストとは，長さ n のテキストに，補助情報としてテキスト上の互いにオーバラップを許した区間の集合（プロパティという）が付加された構造化文書の一種であり，アノテーション付きの系列データの形式的なモデルとなっている．このプロパティ付きテキストへの全文テキスト索引として，Amir ら（CPM2006）は，プロパティ接尾辞木を提案した．これは，プロパティの各区間に含まれるすべての部分文字列を格納する索引構造であり，遺伝子情報や，ビデオストリーム，メタデータ付き時系列データなどへの応用がある．また，高度な検索問題である重み付きパターン照合にも用いられる．Amir らは，定数サイズのアルファベット上で，プロパティ接尾辞木を $O(n \log\log n)$ 時間でオフライン構築するアルゴリズムを与えたが，その線形時間構築アルゴリズムは，現在まで未解決の問題であった．本論文では，定数アルファベット上で，プロパティ接尾辞木を線形時間で構築するオフラインアルゴリズムを与え，この問題を肯定的に解決する．提案アルゴリズムは，接尾辞リンクの巡回を用いた簡潔な手法であり，理論的に効率良いだけでなく，実際のデータに対しても高速に動作する．更に，人工データ上の計算機実験を行い，実際の性能を評価する．

キーワード　テキスト索引，情報検索，プロパティ付き文字列，接尾辞木，パターン照合

1. まえがき

1.1 背　　景

文字列照合問題とは，テキスト T とパターン P と呼ばれる二つの文字列が与えられた際に，T 中での P の出現を求める問題である．この問題は計算機科学の黎明期から知られていると同時に，現在においても情報検索の重要な基盤技術の一つとしての役割を担っている．しかし，照合のたびにテキストを読み込む手法では，少なくともテキスト長に比例した計算時間が必要である．

大規模な検索システムにおいては，同一のテキストに対して，いくつもの異なるパターンの照合を高速に行いたいという要求がある．これに対し，与えられたテキストに対して前処理を施すことで，高速な照合を可能とする索引構造を構築する手法がある．接尾辞木 [12], [15] はこの要求を満たすよく知られた索引構造であり，長さ n のテキスト T のすべての部分文字列を $O(n)$ 領域で保持することができる．接尾辞木を用いることで，テキスト長 n に依存せず，長さ m のパターン P に対する文字列照合問題を $O(m \log|\Sigma| + occ)$ 時間で解くことができる．ここで，$|\Sigma|$ はアルファベットのサイズであり，occ は T 中での P の出現回数である．また，接尾辞木は $O(n \log|\Sigma|)$ 時間で構築可能である．

1.2 プロパティ接尾辞木

遺伝子情報処理やストリーム処理等において，テキストの特定の条件を満たす部分だけを対象として文字列照合を行う場合がある．このような特定の条件を満たすテキストの部分をプロパティと呼ぶ．

形式的には，アルファベット Σ 上の長さ $n \geq 0$ の入力テキスト T に対して，T のプロパティ（*property*）を整数の順序対の集合 $\pi = \{(s_1, f_1), \ldots, (s_k, f_k)\}$ と定義する．ここに，各順序対 $(s, f) \in \pi$ $(1 \leq s \leq f \leq n)$ は，位置集合 $\{1, \ldots, n\}$ 上の空でない閉区間であり，指定された性質を満たすような T の部分文字列 $T[s \ldots f]$ を表す．ただし，各区間は互いに重複していてもよい．プロパティ付きテキスト（*text with property*）は，テキスト T とそのプロパティ π の組 $I = (T, \pi)$ である．図 1 にプロパティ付きテキストの例を示す．

† 北海道大学大学院情報科学研究科，札幌市
Graduate School of Information Science and Technology, Hokkaido University, N14 W9, Sapporo-shi, 060–0814 Japan
a) E-mail: tue@ist.hokudai.ac.jp

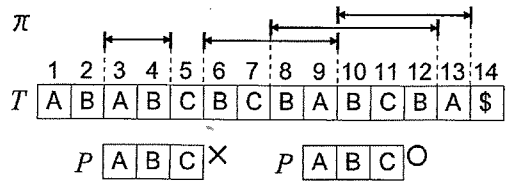

図 1 プロパティ付きテキストとパターンの例
Fig. 1 Examples of a text with property and the occurences of a pattern.

このとき，プロパティ付き文字列照合問題（*property matching problem, PMP*）は，プロパティ付きテキスト $I=(T,\pi)$ に対して，与えられたパターン $P\in\Sigma^*$ の T 中の出現位置のうちで，π の区間に含まれるものすべてを求める問題である．このような出現位置を，P の π のもとでの T 中の出現という．このプロパティ付き文字列照合問題は，KMP 法 [7], [13] 等の既存の文字列照合アルゴリズムの簡単な拡張によって，$O(n+m+k)$ 時間で解くことができる．

これに対して，プロパティ付き文字列索引構築問題（*property indexing problem, PIP*）とは，静的に与えられたプロパティ付きテキストに対して，プロパティ付き文字列照合問題を繰り返し効率良く解くための索引構造を構築する問題である．PMP の場合と異なり，PIP の効率良い解法は自明ではない．

この PIP に対して，近年，Amir ら [1] は接尾辞木をもとに，プロパティ付きテキストに対する索引構造であるプロパティ接尾辞木（*property suffix trees, PST*）を提案した．長さ n のテキストに対し，そのプロパティ接尾辞木は，$O(n)$ 領域のデータ構造であり，これを用いて，長さ m のパターン P に対し，$O(m\log|\Sigma|+tocc)$ 時間で π のもとでの P の T 中の出現すべてを求めることができる．ここに，$|\Sigma|$ はアルファベットのサイズであり，$tocc$ は P の π のもとでの出現回数である．

Amir らは，プロパティ付きテキストから，プロパティ接尾辞木を $O(n\log|\Sigma|+n\log\log n)$ 時間で構築するアルゴリズムを与えている．しかし，これが $O(n\log|\Sigma|)$ 時間で構築可能であるかどうかは未解決の問題である．本論文の目標は，プロパティ接尾辞木を $O(n\log|\Sigma|)$ 時間で構築する最適なアルゴリズムを示すことである．

1.3 主 結 果

本論文では，Amir らのアルゴリズムを改良して，プロパティ接尾辞木のオフライン線形時間構築アルゴリズムを与える．

Amir らの構築アルゴリズムでは，プロパティ接尾辞木は，接尾辞木をいったん構築した後に不要な枝を刈り込むことで作られる．枝を刈り込む際には，どこまでを削除するかの境界を決める必要がある．文字列 $T=t_1\cdots t_n$ のすべての位置 $1\le i\le n$ について，i 番目から始まる文字列がどこまでプロパティの区間内に含まれるかを表す位置 $end(i)$ を考える．境界は，根から部分文字列 $t_i\cdots t_{end(i)}$ を葉の方向にたどって到達する頂点である境界ノード $bd(i)$ によって定められる．各 $bd(i)$ を根からたどって見つけると一つ当り $O(n)$ 時間かかり，全体として構築に $O(n^2)$ 時間かかる．Amir ら [1] は，重み付き先祖質問 [5] を実現する特殊なデータ構造を用いて，全部で n 個ある境界ノードを一つ当り $O(\log\log n)$ 時間で求める方法を与えた．彼らはこれを用いて，境界より下にある枝を刈る手法により，プロパティ接尾辞木を構築する $O(n\log|\Sigma|+n\log\log n)$ 時間アルゴリズムを与えた．ただし，$|\Sigma|$ はアルファベットのサイズである．

これに対して，提案アルゴリズムの基本的なアイディアは，接尾辞リンク [12], [14] を利用することである．この接尾辞リンクを用いて，すべての境界ノードを一つ当り $O(\log|\Sigma|)$ ならし時間で効率良く見つけることで，提案アルゴリズムは，定数サイズのアルファベット上で，長さ n のプロパティ付きテキスト $I=(T,\pi)$ のプロパティ接尾辞木を $O(n\log|\Sigma|)$ 時間で構築する．これは，定数アルファベットに対して，プロパティ接尾辞木のオフライン線形時間構築アルゴリズムを与え，PIP の線形時間解法を与える．

Amir ら [1] のアルゴリズムと比較して，我々の構築アルゴリズムは，[1] のような複雑なデータ構造を用いることなく，接尾辞木の構築時に副産物として得られる接尾辞リンクによって高速なアルゴリズムを実現している点に特色がある．

これにより，アルゴリズムは単純で実装が容易である．また，理論的に高速なだけでなく，計算機実験が示すように，実際にも高速である．

1.4 プロパティ接尾辞木の応用

プロパティ付きテキストは，MPEG7 [11] のようなアノテーション付きの時系列データを表すための形式的なモデルとなっている．このようなデータが与えられたとき，プロパティ接尾辞木を用いることで，データ中の必要な部分のみから索引構造を構築することができる．これにより，アノテーション付き動画像に対

する高速な検索エンジンへの応用が可能となる．

また，Amir らはプロパティ接尾辞木を重み付き系列データの照合に応用している [1]．これは，各文字に重みの付いたテキストに対する文字列照合の問題を，プロパティ付きテキストに対する問題へ還元したものであり，ゲノムデータ等への適用が可能である．

1.5 関連研究

Weiner は接尾辞木の線形時間オフライン構築アルゴリズム [15] を最初に示した．McCreight はより領域効率の良い線形時間オフライン構築アルゴリズム [12] を示した．比較的近年になり，Ukkonen はオンライン線形時間構築アルゴリズム [14] を示した．

本論文で扱うプロパティのほかにも，接尾辞木に含める部分文字列を限定する手法が多数提案されている．Larsson はスライド窓の範囲の文字列のみに対する接尾辞木のオンライン線形時間構築アルゴリズム [10] を示した．Andersson らは単語の先頭から始まる文字列のみを扱う単語接尾辞木 [3] を提案した．Inenaga らは単語接尾辞木のより簡潔なオンライン線形時間構築アルゴリズム [9] を与えた．Na らは索引に含める部分文字列の長さを，指定された文字数に制限した切り落とし接尾辞木 [4] を提案した．上村らはこれを指定された単語数に制限した単語幅制約付き接尾辞木 [16] を提案し，オンライン線形時間構築アルゴリズムを与えた．

最近，Iliopoulos ら [8] は，本論文の結果と独立に，PIP に対する線形時間解法を与えている．彼らのプロパティ付き文字列索引 IDS_PIP は，枝刈りしない接尾辞木と区間最大値問題を組み合わせたデータ構造であり，長さ n のテキストに対して，$O(n)$ 領域を用い，プロパティ付き文字列照合問題を $O(m \log|\Sigma| + tocc)$ 時間で解くことができる．文献 [8] の主結果として，彼らは，一般のアルファベット Σ に対して，IDS_PIP を $O(n \log|\Sigma|)$ 時間でオフライン構築可能であることを示している．更に，彼らは $|\Sigma|$ が n の多項式サイズのとき，Farach [6] の線形時間接尾辞木構築アルゴリズムを用いて，IDS_PIP の構築を $O(n)$ 時間に改善している．

1.6 本論文の構成

本論文の構成は以下のとおりである．**2.** では基本的な定義を与える．**3.** ではプロパティ接尾辞木を定義する．**4.** では本論文で提案するプロパティ接尾辞木構築アルゴリズムを与え，その正当性と計算時間を示す．**5.** では計算機実験を行った結果について述べる．**6.** では本論文をまとめ，今後の課題を与える．

なお，本論文は，論文 [17], [18] の拡充版である．論文 [17], [18] では本論文の **4.** で記述されている手続き Find-border を提案し，計算機実験によりその性能を評価した．本論文では，Find-border が線形時間計算量をもつことを証明し，これを用いてプロパティ接尾辞木が定数アルファベット上で線形時間構築可能であることを示している．

2. 準　　備

2.1 基本的な定義

アルファベットを Σ とする．Σ 上の文字列全体の集合を Σ^* で表す．文字列 $x \in \Sigma^*$ について，x の長さを $|x|$ と表す．特に長さが 0 の文字列を空語（*empty word*）といい，ε で表す．文字列 $x_1, x_2 \in \Sigma^*$ の連結を $x_1 \cdot x_2$ で表す．

ある文字列 $w \in \Sigma^*$ に対して，$w = xyz$ となる $x, y, z \in \Sigma^*$ が存在するとき，x, y, z をそれぞれ w の接頭辞（*prefix*），部分文字列（*substring*），接尾辞（*suffix*）と呼ぶ．w の i 番目の文字を $w[i]$ と表し，w の i 番目から j 番目までの部分文字列を $w[i \ldots j]$ で表す．$i > j$ のとき，便宜的に $w[i \ldots j] = \varepsilon$ と定義する．文字列 $w \in \Sigma^*$ の部分文字列全体の集合を $Fac(w) = \{w[i \ldots j] | 1 \leq i, j \leq |w|\}$ で表す．また同様に，接頭辞全体の集合，接尾辞全体の集合をそれぞれ，$Pre(w)$，$Suf(w)$ と表す．

2.2 プロパティ付きテキスト

プロパティ付きテキスト（*string with property*）とは，長さ n の文字列 T とプロパティ π の組 $I = (T, \pi)$ である．ここで，T のプロパティ（*property*）π とは，集合 $\pi = \{(s_1, f_1), \ldots, (s_m, f_m)\}$ であり，任意の $1 \leq i \leq m$ について，区間 $(s_i, f_i) \in \pi$ は $s_i, f_i \in \{1, \ldots, n\}$ かつ $s_i \leq f_i$ を満たす．このとき，s_i と f_i を開始位置及び終了位置と呼ぶ．$I = (T, \pi)$ の長さは，文字列 T の長さ $n = |T|$ と定義する．以降では，長さが n のプロパティ付きテキスト $I = (T, \pi)$ を仮定する．

文字列 S が π の区間に含まれる T の部分文字列であるとは，ある区間 $(s, f) \in \pi$ と，ある位置の組 $1 \leq i, j \leq n$ に対して，(i) $j = i + |S| - 1$ (ii) $S = T[i \ldots j]$ かつ (iii) $s \leq i$ 及び $j \leq f$ が成立することをいう．このとき，S は π のもとで T 中に位置 i で出現するという．π の区間内に含まれるすべての T の部分文字列からなる集合を

$Fac(T,\pi) = \bigcup_{(s,f)\in\pi} Fac(T[s\ldots f]) \subseteq \Sigma^*$ と表す．

[例 1] 図 1 に，プロパティ付きテキスト $I = (T,\pi)$ を示す．ここに，T = ABABCBCABCBA\$ はテキストであり，$\pi = \{(3,4),(6,9),(8,12),(10,13)\}$ はプロパティである．この図から，パターン P = ABC に対して，部分文字列 $T[9\ldots11]$=ABC が区間 $(8,12)$ に含まれているので，位置 9 は π における P の出現位置である．一方，区間 $(3,9)$ はどの区間にも含まれていないので，位置 3 は π における P の出現位置ではない．

T のプロパティ $\pi = \{(s_1,f_1),\ldots,(s_m,f_m)\}$ は，以下の性質をもつとき標準形（*standard form*）であるという．

（1） 任意の $1 \le i \le n$ について，$s_k = i$ となるような区間 $(s_k,f_k) \in \pi$ $(1 \le k \le m)$ はたかだか一つしか存在しない．

（2） $s_1 < s_2 < \cdots < s_m$.

つまり，π が標準形であるならば，すべての区間が開始位置の昇順で与えられ，それらの開始位置はすべて異なる．また，このとき $m \le n$ となることに注意する．議論の簡単のため，以降では標準形のプロパティのみを扱う．

2.3 接尾辞木

文字列集合 S を表すトライをパス圧縮して得られる木をパス圧縮トライ（*path-compacted trie*）と呼び，$STree(S)$ と表す．すなわち，$STree(S)$ の各辺は異なる文字で始まる文字列でラベル付けされ，$STree(S)$ 上に S の各要素 $s \in S$ を表すノードが存在する．

長さ $n \ge 1$ のテキスト T を，$T[1\ldots n-1]$ に現れない特別な記号 \$ で終わる文字列と仮定する．T の接尾辞木（*suffix tree*）は $ST(T) = STree(Suf(T)) = (V \cup \{\perp\}, root, E, suf)$ である．ここで，V はノードの集合である．E は辺の集合で，$E \subseteq V^2$ である．ノード $s,t \in V$ について，$(s,t) \in E$ のとき，s を t の親（*parent*），t を s の子（*child*）と呼ぶ．また，子をもたないノードを葉（*leaf node*）と呼ぶ．葉全体は接尾辞全体の集合 $Suf(T)$ と 1 対 1 で対応する．*root* は木 $ST(T)$ の根である．$\perp$ は $\perp \notin V$ であり，*root* の親となる特別なノードとする．*root* からノード $s \in V$ へのパス上にあるノード $t \in V$ を s の祖先（*ancestor*），s を t の子孫（*decendant*）と呼ぶ．s は s 自身の祖先かつ子孫であることに注意する．任意の辺 $e \in E$ には T の空でない部分文字列 $T[j\ldots k]$ $(1 \le j \le k \le |T|)$ がラベル付けされる．ノード $t \in V$ へ向かう辺のラベル文字列を $label(t)$ と書くことにする．便宜上，$label(root) = \varepsilon$ と定義する．更に，任意のノード $s \in V$ について，*root* から s への祖先の列 $root, a_1, a_2, \ldots, s$ の各ラベル文字列を連結した文字列 $label(root) \cdot label(a_1) \cdot label(a_2) \cdots label(s)$ をノード s が表す文字列（*string represented by the node* s）と呼び，$\overline{s}$ と書く．*suf* は V 上の関数 $suf : V \to V$ であり，ノード $s,t \in V$ と文字 $a \in \Sigma$ に対して，$\overline{s} = a \cdot \overline{t}$ であるとき，$suf(s) = t$ である．ここで，任意の実ノード $s \in V$ について，$\overline{s} = a \cdot \overline{t}$ となる実ノード $t \in V$ が存在する [14]．この関数は s から t への辺 (s,t) とみなすこともできるので，これを接尾辞リンク（*suffix link*）と呼ぶ．特に $suf(root) = \perp$ とし，$suf(\perp)$ は未定義とする．

集合 $Suf(T)$ に対するトライからパス圧縮によって削除され，接尾辞木 $ST(T)$ 上においては存在しないノードを $ST(T)$ の仮想ノード（*virtual node*）と呼ぶ．これと区別するため，以降は V の要素を $ST(T)$ の実ノード（*real node*）と呼ぶ．T の部分文字列 w に対応する仮想ノードを $[w]$ で表す．

仮想ノード v が表現する T の部分文字列を $\overline{v} = T[i\ldots j]$ とする．このとき，v の祖先でかつ実ノードである $s \in V \cup \{root\}$ が少なくとも一つ存在するので，$\overline{v} = T[i\ldots j] = \overline{s} \cdot T[k\ldots j]$ となる整数 $i \le k \le j$ が存在する．したがって，仮想ノード v は，参照ポインタと呼ばれるノードと区間の組 $\langle s,(k,i)\rangle$ で指し示すことができる．一方，任意の $s \in V$ についても，参照ポインタ $\langle s,(i+1,i)\rangle$ はノード s 自身を指し示すとみなせる．よって以降では，実ノード全体の集合 V と仮想ノード全体の集合 U に対し，任意のノード $w \in V \cup U$ を $w = \langle s,(k,i)\rangle$ と表現し，混乱が生じない限り仮想ノードも実ノードと同等に扱う．

参照ポインタ $w = \langle s,(k,j)\rangle$ は，s が w の最も近い実ノードの祖先であるとき，正規形（*canonical form*）であるという．任意の参照ポインタ $w = \langle s,(k,j)\rangle$ が与えられたとき，これを正規形に変換する手続き **Canonize** を図 2 に示す．

3. プロパティ接尾辞木

本章では，Amir ら [1] に従ってプロパティ接尾辞木を導入する．長さ n のプロパティ付きテキスト $I = (T,\pi)$ に対するプロパティ接尾辞木（*property suffix tree*）$\mathcal{P} = PST(T,\pi)$ は，T の接尾辞木 $\mathcal{S} = ST(T) = (V \cup \{\perp\}, root, E, suf)$ から，以下の

```
手続き Canonize(⟨s,(k,i)⟩);
入力 : 参照ポインタ v = ⟨s,(k,i)⟩;
出力 : v̄ を表す正規形の参照ポインタ;
1  if (k > i)
2    return ⟨s,(k,i)⟩;
3  s'=T[k] で始まるラベル文字列をもつ s の子;
4  while (|label(s')| ≤ i − k + 1)
5    ⟨s,(k,i)⟩ = ⟨s', (k + |label(s')|, i)⟩;
6    if (k ≤ i)
7      s'=T[k] で始まるラベル文字列をもつ s の子;
8    end while
9  return ⟨s,(k,i)⟩;
```

図 2 参照ポインタを正規形に変換する手続き
Fig. 2 A procedure for canonizing a reference pointer.

ように，辺の除去と併合，及び補助情報の付加を行って得られる根付き木 $\mathcal{P} = (V' \cup \{\perp\}, root, E', end, pos)$ である．ここで，$V' \subseteq V$ は実ノードの集合であり，$E' \subseteq V'^2$ は辺の集合である．終了位置関数 end は，各添字 $1 \leq i \leq n$ に，終了位置と呼ばれる添字を割り当てる．添字集合関数 pos は，各実ノード $s \in V'$ に対して，添字集合 $pos(s) \subseteq \{1, \ldots, n\}$ を割り当てる．

3.1 終了位置関数

はじめに，終了位置関数 end を与える．T 上の任意の位置 $1 \leq i \leq n$ に対して，i に対応する終了位置 (*the end position for* i) を $end(i)$ と書き，次のように定義する：$end(i)$ は $\sigma = (s, f) \in \pi$ かつ $s \leq i \leq f$ を満たす最大の終了位置 f である．上の条件を満たす σ が存在しない場合は，$end(i) = i - 1$ と定義する．更に，$end(0) = -1$ と定義する．

次の補題は，$end(i)$ に関する重要な性質であり，**4.** で示す構築手続きの正当性に関して重要である．

[補題 1] 長さ n の任意のプロパティ付きテキスト $I = (T, \pi)$ について，$end(0), \ldots, end(n)$ は単調非減少な列をなす．すなわち，$end(0) \leq \cdots \leq end(n)$ が成立する．

(証明) ある位置 $1 \leq i \leq n$ で $end(i-1) > end(i)$ であると仮定すると，$T_\pi^{i-1} = T[i-1 \ldots end(i-1)]$ は $T_\pi^i = T[i \ldots end(i)]$ を完全に含む．$end(i)$ の定義から $end(i) = end(i-1)$ となり，仮定と矛盾する．よってすべての位置で $end(i-1) \leq end(i)$ が成立する． □

位置 i から始まる T のプロパティ接尾辞を，$T_\pi^i = T[i \ldots end(i)]$ と定義する．$I = (T, \pi)$ のすべてのプロパティ接尾辞の集合を $Suf(T, \pi) = \{T_\pi^i | 1 \leq i \leq n\}$ で表す．

T = ABABCBCABCBA\$
π = {(3, 4), (6, 9), (8, 12), (10, 13)}

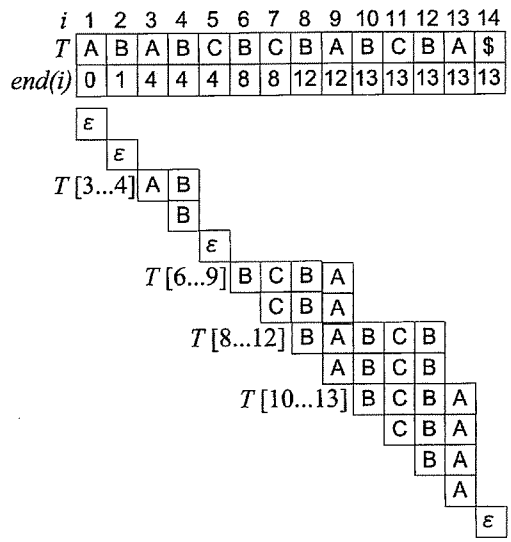

図 3 終了位置の配列 $end(\cdot)$ と $Suf(T, \pi)$ の関係
Fig. 3 The relation of $end(\cdot)$ and $Suf(T, \pi)$.

[例 2] 図 3 に，例 1 のプロパティ付きテキスト $I = (T, \pi)$ に対する，終了位置の配列 $end(\cdot)$ とプロパティ接尾辞の集合 $Suf(T, \pi)$ を示す．

次の補題は，プロパティ付き文字列照合に関して本質的である．

[補題 2] 任意のパターン P と任意の位置 $1 \leq i \leq n$ に対して，P が π のもとで T 中の位置 i に出現することは，P が T_π^i の接頭辞であることと同値である．

上の補題より，プロパティ付き文字列照合問題を解くための索引構造として，プロパティ接尾辞の集合 $Suf(T, \pi)$ を効率良く格納するデータ構造を実現すればよいことが分かる．

3.2 下方集合キュー

任意の全順序集合を $(A, \leq)$ とする．ここに，$A = \{a_1, \ldots, a_n\}$ は要素の集合であり，$\leq$ は A 上の全順序である．任意の集合 A の要素数を $|A|$ と書く．任意の要素 $x, y \in A$ に対して，もし $x \leq y$ ならば，y は x より下方にあるという．

[定義 1] (下方集合キュー) $(A, \leq)$ に関する下方集合キュー (*lowerset queue*) とは，次の操作をもつような集合 $S \subseteq A$ を格納する抽象データ型 $LQ(S)$ である．

- 構築手続き **Create** (S)：与えられた集合 S に対する下方集合キュー $LQ(S)$ を返す．

```
手続き Create(S);
入力 : 集合 S = {a_1, ..., a_n}(n ≥ 0);
出力 : 下方集合キュー LQ(S);
1  LQ = LQ(∅);
2  S' = S;
3  while S' ≠ ∅ do
4    (LH, UH) =Split(S');
5    foreach a ∈ UH
6      LQ の先頭に要素 a を追加する;
7    S' = LH;
8  end while
9  return LQ;
```

図 4 下方集合キューの構築手続き
Fig. 4 A procedure for constructing the lowerset queue.

```
手続き Lower(LQ(S), x);
入力 : 下方集合キュー LQ(S), x ∈ A;
出力 : 下方集合 L;
1  L = ∅;
2  i = 1;
3  flag =true;
4  while flag == true do
5    for j = 1, ..., 2^{i-1} do
6      LQ(S) の先頭から要素 a を取り出す;
7      if x ≤ a then
8        L = L ∪ {a};
9      else
10       flag =false;
11   end for
12   i = i + 1;
13 end while
14 return L;
```

図 5 下方集合を求める手続き
Fig. 5 A procedure for lowerset query.

• 下方集合問合せ手続き $\mathbf{Lower}(LQ(S), x)$：任意の要素 $x \in A$ に対し，x より下方にあるすべての S の要素の集合，すなわち S における x の下方集合 $Lower(S, x) = \{y \in S \mid x \leq y\}$ を返す．

[定理 1]（Amir *et al.* [1]） $(A, \leq)$ を $O(1)$ 時間で比較可能な全順序 $\leq$ をもつ全順序集合と仮定する．このとき，任意の集合 $S \subseteq A$ と任意の要素 $x \in A$ に対して，$LQ(S) = \mathbf{Create}(S)$ を $O(|S|)$ 時間で，$\mathbf{Lower}(LQ(S), x)$ を $O(|Lower(S, x)|)$ 時間で計算するように，下方集合キュー $LQ(S)$ を実装可能である．

図 4 と図 5 に，定理 1 の手続き **Create** と **Lower** を示す．ここに，$\mathbf{Split}(S)$ は S の中央値より下方にある部分の集合 LH とそれ以外の部分の集合 UH に線形時間で分割する手続きである．上述の定理 1 の条件を満たすような下方集合キューを，効率良い（*efficient*）下方集合キューという．

3.3 添字集合関数

次に，添字集合関数 pos を定義する．T に対する接尾辞木を $\mathcal{S} = ST(T)$ とおき，位置の集合を $Pos = \{1, \ldots, n\}$ とおく．接尾辞木 $\mathcal{S}$ では，各接尾辞 $T^i = T[i \ldots n]$ に対応する葉 x に添字 i を格納した．これに対し，プロパティ接尾辞木 $\mathcal{P}$ では，各プロパティ接尾辞 $T^i_\pi = T[i \ldots end(i)]$ に対応する仮想ノードに添字 i を格納したいが，実際には $\mathcal{P}$ は実ノードしかもたないので次のように実現する．以後，T の部分文字列 w と，$\mathcal{S}$ の根から w の文字で枝をたどって到達する仮想ノード $[w]$ を同一視する．

はじめに，各仮想ノード（及び実ノードも含む）$x \in Fac(T, \pi)$ に対して，位置の集合 $pos_1(x) = \{\, i \mid T^i_\pi = x \,\}$ を関連づける．これは，仮想ノード x が表すプロパティ接尾辞の添字の全体である．次に，任意の実ノード $s \in V$ に対して，文字列の集合 $X(s) = Pre(\overline{s}) \setminus Pre(\overline{t})$ を定義する．ここに，$t \in V$ は s の親となる実ノードである．定義より，$X(s)$ は s を下端とする辺上にのっている仮想ノードで，t と異なるもの全体に対応する．最後に，任意の実ノード $s \in V'$ について，$pos(s) = \bigcup_{x \in X(s)} pos_1(x) \subseteq Pos$ と定義する．各要素 $i \in pos(s)$ に対して，その深さを $dep(i) = |T^i_\pi|$ と定義する．

このとき，$pos(s)$ の要素間の全順序 $\leq_{dep}$ を，次のように定義する：任意の $i, i' \in pos(s)$ に対して，$i \leq_{dep} i'$ であるのは，$dep(i) \leq dep(i')$ であるとき，そのときのみである．以上の準備の下で，$pos(s)$ を全順序集合 $(Pos, \leq_{dep})$ に関する効率良い下方集合キューとして表す．

3.4 実ノード集合と辺集合

最後に，実ノードの集合 V' と辺の集合 E' を定義する．与えられた接尾辞木 $\mathcal{S}$ において，関数 pos が計算されていると仮定する．このとき，$pos(s)$ が空でない $\mathcal{S}$ の実ノード s を境界ノードと呼ぶ．**4.** で境界ノードの等価な定義を与える．V', E' は以下のようにして求められる．まず，境界ノードとそれらのすべての先祖に印を付ける．次に，印が付いていないすべての実ノードと，それに接続するすべての辺を $\mathcal{S}$ から除去する．最後に，すべての鎖ノード（子を一つしかもたない実ノード）を除去し，それに接続する辺を併合する．このとき，対応する添字集合 pos も同時に併合する．手続きの詳細は **4.** の手続き **Prune** と手続き **Adjust** を参照されたい．上の手続きで得られた木

の実ノードの集合と辺の集合を，V' と E' とおく．得られた木の任意の葉 $s \in V'$ について，$pos(s)$ の要素 $i \in pos(s)$ のうち最大の $dep(i)$ をもつ i に対し，もし $|\bar{s}| > |T^i_\pi|$ ならば，$\bar{s} = \tilde{T}^i_\pi$ となるように $label(s)$ を変更する．以上によって，$I = (T, \pi)$ のプロパティ接尾辞木 $\mathcal{P} = PST(T, \pi)$ が定義された．

図 6 に，図 1 の例のプロパティ付きテキストに対するプロパティ接尾辞木 $PST(T, \pi)$ を示す．

3.5 プロパティ付き問合せ

長さ n のプロパティ付きテキスト $I = (T, \pi)$ が与えられたとき，プロパティ付き問合せとは，長さ m のパターン P に対して，P の π の下での T 中のすべての出現位置 $i_1, \ldots, i_{tocc} \subseteq \{1, \ldots, n\}$ を求める問題である．

プロパティ接尾辞木 $\mathcal{P} = PST(T, \pi)$ が与えられたとき，長さ $m \geq 0$ のパターン P に対するプロパティ付き問合せ問題は，$\mathcal{P}$ を用いて次のようにして解くことができる：はじめに，$\mathcal{P}$ の根から葉に向かって P の各文字で辺をたどり，$P \in X(s)$ である実ノード $s \in V'$ を見つける．次に，s を根とする $\mathcal{P}$ の部分木を深さ優先探索で巡回し，s の子孫 t $(t \neq s)$ すべてについて，$pos(t)$ のすべての要素を出力する．最後に，下方集合キュー $pos(s)$ を用いて，$dep(i) \geq m$ であるすべての位置 $i \in pos(s)$，すなわち下方集合 $Lower(pos(s), m)$ のすべての要素を出力する．これらが P の π のもとでの T 中のすべての出現位置である．

Amir ら [1] はプロパティ付き問合せ問題に対し，次の補題を示した．

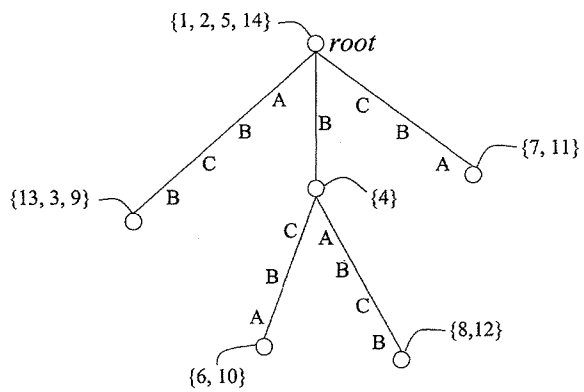

図 6 例 1 のプロパティ付きテキストに対するプロパティ接尾辞木
Fig. 6 The property suffix tree for the text with property in Example 1.

[補題 3] プロパティ付きテキスト $I = (T, \pi)$ と長さ m のパターン P が与えられたとき，プロパティ接尾辞木 $PST(T, \pi)$ によって，プロパティつき問合せ問題は $O(m \log |\Sigma| + tocc)$ 時間で解くことができる．ここに，$|\Sigma|$ はアルファベットのサイズであり，$tocc$ は P の π のもとでの T 中のすべての出現位置の個数である．

4. プロパティ接尾辞木の構築

4.1 構築アルゴリズムの概要

プロパティ付きテキスト $I = (T, \pi)$ に対するプロパティ接尾辞木 $PST(T, \pi)$ の構築手続き **Construct-PST** を図 7 に示す．この手続きでは，まず接尾辞木 $ST(T)$ を構築する．次に，$1 \leq i \leq n$ について $end(i)$ を計算する．そして，$ST(T)$ から必要なノードと不要なノードを決定するための境界となる実ノードを計算する．更に，不要な実ノードを削除する．最後に，葉に接続している辺のラベル文字列を修正する．

4.2 境界を計算する手続き

長さ n のプロパティ付きテキスト $I = (T, \pi)$ が与えられたとき，任意の位置 $1 \leq i \leq n$ に対する接尾辞木 $ST(T)$ 上の境界ノード (*border node*) $bd(i)$ とは，$T^i_\pi \in X(s)$ である実ノード $s \in V$ である．各 i について，$bd(i)$ は，*root* から接尾辞 $T[i \ldots n]$ を表す葉 t へのパス上にただ一つ存在することに注意する．$ST(T)$ からすべての位置 $i = 1, \ldots, n$ に対する境界ノード $bd(i)$ を計算する手続き **Find-border** を図 8 に示す．この手続きでは，T^i_π を表すノード $w = \langle s, (k, end(i)) \rangle$ を保持し，前回得られた境界ノードを用いて次の境界ノードを計算する．実ノード $s \in V'$ に対する位置 i の集合 $pos(s)$ にはリストを用いる．

終了位置の列 $end(0), \ldots, end(n)$ の単調非減少

手続き **Construct-PST** (T, π);
入力 ：長さ n のプロパティ付き文字列 $I = (T, \pi)$;
出力 ：$PST(T, \pi)$;
1 $PST = ST(T)$;
2 **for** $i = 1$ **to** n
3 $end(i) = \max(\{f | (s, f) \in \pi, s \leq i \leq f\} \cup \{i - 1\})$;
4 **Find-border**(PST, end);
5 **Prune**(PST);
6 **Adjust**(PST);
7 **return** PST;

図 7 プロパティ付き接尾辞木の構築手続き
Fig. 7 A procedure for constructing a property suffix tree.

```
手続き Find-border(ST(T), {end(1), ..., end(n)});
入力: 接尾辞木 ST(T), 終了位置 {end(1),...,end(n)}
1  w = ⟨s, (k, j)⟩ = ⟨root, (1, 0)⟩;
2  for i = 1 to |T| do
3    w = ⟨s, (k, j)⟩ =Canonize(⟨suf(s), (k, end(i))⟩);
4    if k > j then
5      border(i) = s;
6    else
7      border(i) =w に最も近い実ノードの子孫;
8    pos(border(i)) = pos(border(i)) ∪ {i};
9    t = border(i);
10   while t ≠ root and t は印が付けられていない do
11     t に印をつける;
12     t = t の親;
13   end do
14 end do
```

図 8 すべての境界ノードを計算する手続き
Fig. 8 A procedure for finding all border nodes.

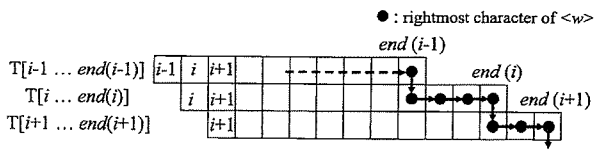

図 9 境界ノードの計算における参照ポインタ w の移動
Fig. 9 Moving the reference pointer w in border node computation.

```
手続き Prune (I = (T, π), ST(T));
入力 : 境界ノードとその祖先に印が付けられた接尾辞木 ST(T);
1  foreach {s ∈ V' | 印の付いていない実ノード s} do
2    s を根とする ST(T) の部分木を削除する;
3    if s の親 t がただ一つの子 s' しかもたない then
4      pos(s') = pos(s') ∪ pos(t);
5      t を削除し, t に接続している辺を併合する;
6    end if
7  end do
8  foreach s ∈ V'
9    下方集合キュー pos(s) を構築する;
```

図 10 プロパティ付き接尾辞木の刈り込み手続き
Fig. 10 A procedure for pruning property suffix trees.

```
手続き Adjust(PST*(T, π));
入力 : Construct-PST によって構築された木 PST*(T, π);
1  foreach {s ∈ V' | PST(T, π) の葉 s} do
2    i = max({dep(i) | i ∈ pos(s)});
3    if |s̄| > dep(i) then
4      s̄ = T_π^i となるように label(s) を変更する;
5  end do
6  return PST(T, π);
```

図 11 葉に対するラベルの修正
Fig. 11 A procedure for adjusting the label strings.

性に関する補題 1 から，T_π^{i-1} と T_π^i の文字列としての差異は，先頭の 1 文字 $T[i-1]$ と，末尾の $T[end(i-1)+1 \ldots end(i)]$ であると分かる．ここで，$\overline{bd(i-1)} = T_\pi^{i-1}$ ならば，$T[i \ldots end(i-1)]$ を表す実ノードへの接尾辞リンク $suf(bd(i-1))$ が存在するので，そのノードから末尾の差分の文字列 $T[end(i-1)+1 \ldots end(i)]$ をたどることで $bd(i)$ が求められる．$|\overline{bd(i-1)}| \neq T_\pi^{i-1}$ の場合も，図 12 のように，最も近い実ノードの祖先 s の接尾辞リンクを使うことで同様に $suf(s)$ をたどることができる．

手続き **Find-border** における w が表す文字列の変化の例を図 9 に示す．接尾辞リンク先への遷移には下への移動が，文字列 $T[end(i-1)+1 \ldots end(i)]$ で進む分には右への移動が対応している．$w = \langle s, (k, j)\rangle$ から接尾辞リンク先のノード $w' = \langle s', (k', j)\rangle$ へ移動する際，図 12 で表したように，手続き **Canonize** は辺のラベル文字列の長さによらず $suf(s)$ から実ノードのみをたどる．

4.3 接尾辞木から不要なノードを削除する手続き

すべての境界ノードを計算したら，図 10 の手続き **Prune** によって，境界ノードより下の部分を削除する．このときパス圧縮された状態を維持するが，ただ一つの子 s しかもたない内部ノード t を削除するとき，$X(s)$ は辺の連結により $X(s) \cup X(t)$ となるため，$pos(s)$ もこれに合わせ $pos(s) \cup pos(t)$ とする．次に，残ったすべての実ノードに対し下方集合キューを構築する．

最後に，図 11 の手続き **Adjust** を行うことで，すべての葉 $s \in V$ について $\overline{s} \in Fac(T, \pi)$ が成り立つように枝を刈る．すなわち，$PST(T, \pi)$ のすべての葉 $s \in V$ について，

$$|\overline{s}| = \max(\{end(i) - i \mid i \in pos(s)\})$$

となるようにラベル文字列の終端を変更する．

4.4 構築アルゴリズムの正当性

まず，境界ノード $bd(1), \ldots, bd(n)$ が終了位置のリスト $end(1), \ldots, end(n)$ から手続き **Find-border** によって計算できることを示す．この手続きの正当性に関して，次の補題が本質的である．

[補題 4] 長さ n のプロパティ付きテキスト $I = (T, \pi)$ と T の接尾辞木 $ST(T)$ が与えられたとする．任意の時点 $i = 1, \ldots, n$ において，あるノード $s \in V$ と値 k に対し，$bd(i-1) = \langle s, (k, end(i-1))\rangle$ であるとき，$bd(i) = \langle suf(s), (k, end(i))\rangle$.

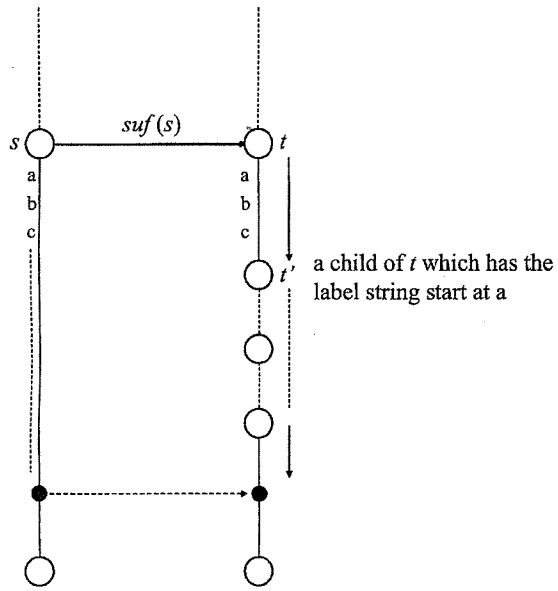

図 12　仮想ノードに対する接尾辞リンクのたどり方
Fig. 12　Following the suffix link for a virtual node.

(証明)　定義より，$\overline{bd(i-1)} = T_\pi^{i-1}$，$\overline{bd(i)} = T_\pi^i$ が成り立つ．また $\overline{bd(i-1)} = \overline{s} \cdot T[k \ldots end(i-1)]$ であるから，$\overline{s} = T[i-1 \ldots k-1]$ である．更に，$\overline{suf(s)} = T[i \ldots k-1]$ である．これらより，$\overline{\langle suf(s), (k, end(i))\rangle} = T[i \ldots k-1] \cdot T[k \ldots end(i)] = T[i \ldots end(i)] = \overline{bd(i)}$ となる．よって $\langle suf(s), (k, end(i))\rangle = bd(i)$ が成り立つ．□

この補題 4 から，帰納法により直ちに次の補題を導くことができる．

[補題 5]　手続き **Find-border** は，接尾辞木 $ST(T)$ と，終了位置のリスト $end(1), \ldots, end(n)$ を入力として受け取り，すべての境界ノード $bd(1), \ldots, bd(n)$ を計算する．

[定理 2]　プロパティ付きテキスト T, π が与えられたとき，手続き **Construct-PST** はプロパティ接尾辞木 $PST(T, \pi)$ を構築する．

(証明)　まず，実ノード $s \in V$ と任意の位置 $1 \le i \le n$ に対して，$s = bd(i) \Leftrightarrow i \in pos(s)$ である．これより，$pos(s)$ が空でないことと，s が境界ノードであるということは同値である．境界ノードがすべて計算されたあとは，明らかに手続き **Prune** と手続き **Adjust** は **3.** の定義どおりにプロパティ接尾辞木を構築する．ここで，補題 5 より手続き **Find-border** の正当性が示されるので，手続き **Construct-PST** の正当性は明らかである．よって示された．□

4.5 計算時間

まず手続き **Find-border** の計算時間を示す．接尾辞木 $ST(T)$ のノード v について，$root$ から v までの実ノードの道 $root, s_1, \ldots, s_k$ を $path(v)$ と書く．ただし，v が仮想ノードの場合は v に最も近い実ノードの祖先までの道とする．$path(v)$ に含まれる実ノードの個数を $depth(v)$ で表す．$suf(root) = \perp$ であるが，$root$ からの道の長さを考えるときは $\perp$ は数えない．ここで，次の補題が成り立つ．

[補題 6]　文字列 T の接尾辞木 $ST(T)$ 上の任意の実ノード $s \in V$ に対し，$depth(suf(s)) \ge depth(s) - 1$.

(証明)　接尾辞木の定義より，$path(s)$=$root$, $s_1, \ldots, s_k$ 上のすべての実ノードに対して，接尾辞リンク先の実ノードが存在する．したがって，$path(suf(s))$ は，$suf(root) = \perp$ を除くノード $suf(s_1)$, $suf(s_2), \ldots, suf(s_k)$ を含むため，命題が成り立つ．□

仮想ノード v についても，$depth(v)$ は v に最も近い実ノードの祖先 s までの道の長さと等しいから，補題 6 と同様の議論で次の補題が成り立つ．

[補題 7]　文字列 T の接尾辞木 $ST(T)$ 上の任意のノード v と，$\overline{v}$ の 1 文字短い接尾辞を表すノード w に対し，$depth(w) \ge depth(v) - 1$.

特に，$depth(bd(i))$ を $d(i)$ と書くことにする．また，手続き **Construct-PST** において，時点 $i = 1, \ldots, n$ で **Canonize** により実ノードの子をたどる個数を $\Delta(i)$ と書く．このとき，次の補題が成り立つ．

[補題 8]　長さ n のプロパティ付きテキスト T_π と接尾辞木 $ST(T)$ が与えられたとき，$\sum_{i=1}^{n} \Delta(i) \le n+1$ が成り立つ．

(証明)　$0 \le end(i) \le n$ より $|\overline{bd(i)}| \le n$ であり，また辺のラベル文字列は空ではないため，$d(i) \le |\overline{bd(i)}|$ より $1 \le d(i) \le n$ である．補題 7 より，$bd(i-1)$ から 1 文字短い接尾辞を表すノード v に移動するとき，$depth(v) \ge d(i-1) - 1$ が成り立つ．また，$\Delta(i)$ の定義より明らかに $d(i) = depth(v) + \Delta(i)$ である．よって，$bd(i-1)$ から $bd(i)$ への移動では $d(i) = depth(v) + \Delta(i) \ge d(i-1) - 1 + \Delta(i)$ が成り立つ．また，$end(0) = -1$ より $T[0 \ldots -1] = \varepsilon$ であるから $d(0) = 1$，$end(n) \le n$ より $d(n) \le 2$ である．ここで，$i = 1, \ldots, n$ について $d(i)$ の総和をとると，

$$\sum_{i=1}^{n} d(i) \geq \sum_{i=1}^{n} \{d(i-1) - 1 + \Delta(i)\}$$

$$= \sum_{i=1}^{n} d(i-1) - n + \sum_{i=1}^{n} \Delta(i),$$

$$\therefore \quad \sum_{i=1}^{n} \Delta(i) \leq \sum_{i=1}^{n} d(i) - \sum_{i=1}^{n} d(i-1) + n$$

$$= d(n) - d(0) + n \leq n + 1.$$

よって $\sum_{i=1}^{n} \Delta(i) \leq n+1$ が示された. □

[補題 9] 長さ n のプロパティ付きテキストが与えられたとき，図 8 の手続き **Find-border** の計算時間は，$O(n \log |\Sigma|)$ 時間である.

(証明) 図 8 の 1 行目は初期値の設定であり定数時間で行える．**Canonize** の実行時間は，子をたどって s, k を更新する図 2 の *while* ループの反復回数に比例する．補題 8 より，**Canonize** は手続き全体でたかだか $n+1$ 回だけ実ノードの子をたどる．実ノードの子をたどるのは $O(\log |\Sigma|)$ 時間で行えるので，図 8 の 3 行目の **Canonize** の計算時間は手続き全体で $O(n \log |\Sigma|)$ 時間である．また，接尾辞リンクは接尾辞木の構築の際に定数時間でたどることができる．4 行目の判定は定数時間で行える．5 行目ならば定数時間で実行でき，7 行目ならば子を一つたどればよいから，$O(\log |\Sigma|)$ 時間で実行可能である．したがって 4 行目から 7 行目の計算は $O(\log |\Sigma|)$ 時間で行える．以上より，10 行目から 13 行目を除き，2 行目の **for** ループによる計算時間は $O(n \log |\Sigma|)$ 時間である．ここで，10 行目から 13 行目で印が付けられる実ノードの個数を考える．この手続きでは祖先をたどり印が付けられた実ノードが見つかると **while** を終了するから，一つの実ノードが印を付けられる回数はたかだか 1 回である．よって，手続き全体で印を付けられる実ノードの個数は木全体から *root* を除き，たかだか $2n-2$ 個である．したがって 10 行目から 13 行目は全体で $O(n)$ 時間で計算可能である．よって，手続き **Find-border** の計算時間は $O(n \log |\Sigma|)$ 時間である. □

ここに，本論文の主結果を示す.

[定理 3] 与えられた長さ n のプロパティ付きテキスト T_π に対し，手続き **Construct-PST(T)** は，$O(n \log |\Sigma|)$ 時間でプロパティ付き接尾辞木 $PST(T_\pi)$ を構築する.

(証明) 1 行目の接尾辞木の構築は，Ukkonen [14] などの手法によって $O(n \log |\Sigma|)$ 時間で可能である．2 行目と 3 行目の終了位置の計算は，π が標準形で与えられると仮定すると，$end(i)$ は $i = 1, \ldots, n$ についてたかだか 1 個現れる新たな区間 (i, f_i) の終端 f_i と，$end(i-1) \neq i-2$ であるならば前回の結果 $end(i-1)$，そして $i-1$ のたかだか三つを比較することで，1 回当り $O(1)$ 時間で計算可能であり，$end(1), \ldots, end(n)$ は $O(n)$ 時間で計算できる．補題 9 より 4 行目の手続き **Find-border** は $O(n \log |\Sigma|)$ 時間で実行できる.

次に 5 行目で行われる図 10 の手続き **Prune** について考える．2 行目の **foreach** ついて，$ST(T)$ の印の付いていない実ノード $s \in V$ を根とする部分木が削除されるとき，部分木の実ノードの個数を $num(s)$ とおく．部分木の削除は $O(num(s))$ 時間で実行可能である．3 行目の子を一つしかもたないかどうかの判定は定数時間で行える．4 行目の操作を行うとき，位置 $i = 1, \ldots, n$ について境界ノード $s \in V'$ はただ一つしか存在しないから，任意の実ノード $s, t \in V'$ について $pos(s) \cap pos(t) = \emptyset$ が成り立つ．したがって，$pos(s) \cup pos(t)$ はリストの単純な連結により実現されるため，4 行目の操作は定数時間で行える．5 行目の辺の併合は定数時間で行うことができる．よって 3 行目から 6 行目は定数時間で実行可能であり，2 行目から 7 行目までの計算時間は $O(num(s))$ 時間である．印が付いていない $0 \leq k \leq n$ 個の実ノード $s_1, \ldots, s_k$ に対し 2 行目から 6 行目の操作を実行したとき，$\sum_{i=1}^{k}(num(s)+1) = \sum_{i=1}^{k} num(s) + k$ に比例した時間を要する．ここで，$1 \leq i \leq k$ 個目の s_i について $num(s)$ は $i-1$ 回目までに削除されていない実ノードである．$ST(T)$ はたかだか $2n-1$ 個の実ノードしかもたないから，$\sum_{i=1}^{k} num(s) \leq 2n$ が成り立つ．よって 1 行目から 7 行目は $O(n)$ 時間で実行可能である．また，定理 1 より，実ノード $s \in V'$ について下方集合キュー $pos(s)$ の構築に $O(|pos(s)|)$ 時間必要である．V' 全体の $pos(s)$ に含まれる位置はたかだか n 個しか存在しないので，$O(\sum_{s \in V'} |pos(s)|) = O(n)$ 時間ですべての下方集合キューを構築可能である．したがって手続き **Prune** は $O(n)$ 時間で実行可能である.

最後に 6 行目で行われる図 11 の手続き **Adjust** について考える．任意の葉 $s \in V'$ に対し，$i \in pos(s)$ から最大の $dep(i)$ をもつ位置 i を求めるための計算は

$O(pos(s))$ 時間必要であるが，ならし評価により，pos に含まれる位置はすべての実ノードの分を合計してもたかだか n 個しか存在しないため，すべての葉に対する処理は合計で $O(\sum_{s\in V'}|pos(s)|) = O(n)$ 時間で処理を行うことができる．また，葉に対するラベル文字列の修正は定数時間で行えるから，手続き **Adjust** は $O(n)$ 時間で実行可能である．

以上より手続き **Construct-PST** は $O(n\log|\Sigma|)$ 時間で実行可能である． □

5. 実　　験

前章で与えたプロパティ接尾辞木構築アルゴリズムを実装し，人工的に作成したデータを用いて計算機実験を行った．

5.1 デ ー タ

テストデータとして，$\Sigma = \{A, G, C, T\}$ をアルファベットとする人工データを用いた．テキストの各文字はランダムに発生させた．ただし，実験によって各文字の出現確率を変化させている．また，プロパティは実験ごとに与える．

5.2 方　　法

この実験では，接尾辞木を構築した後の，すべての境界ノードを計算する部分のみの実行時間を測定する．境界を計算するアルゴリズムとして，提案手法のほか，比較のために，各境界ノードを毎回根から探索する手法，及び毎回葉から探索する手法を実装した．以下では，それらの手法をそれぞれ，*suf*，*root*，*leaf* と呼ぶことにする．

実験 1: すべての文字が等確率に出現するテキストについて，テキスト長 n を変化させて実行時間を測定した．ここで，プロパティ π は，$1 \le i \le n$ について，$end(i) = i + 10$ とした．

実験 2: 各文字の出現確率を $P(A) = P(T) = 0.4$, $P(G) = P(C) = 0.1$ とし，それ以外は実験 1 と同様に実行時間を測定した．

実験 3: $P(A) = 1$，つまり $T = \mathrm{AAA}\cdots\mathrm{A\$}$ としたときの実行時間を測定した．ここで，プロパティ π は，$1 \le i \le n$ について，$end(i) = min(i + \lfloor (n-i)/2 \rfloor, n)$ とした．このとき，各境界ノードは根と葉の中間に位置する．

5.3 結　　果

結果 1: 実行結果を図 13 に示す．テキスト長が短いときは *leaf* が最も高速であるが，長いときには *suf* の方が速くなっている．

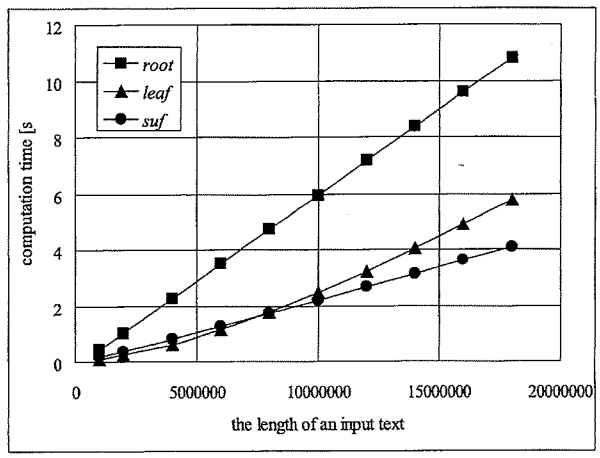

図 13　実験結果 1：すべての文字の出現確率が等確率である場合の実行時間の測定

Fig. 13　Experimental result 1: Computation time for random texts generated with probability $\{P(A) = P(G) = P(C) = P(T) = 0.25\}$.

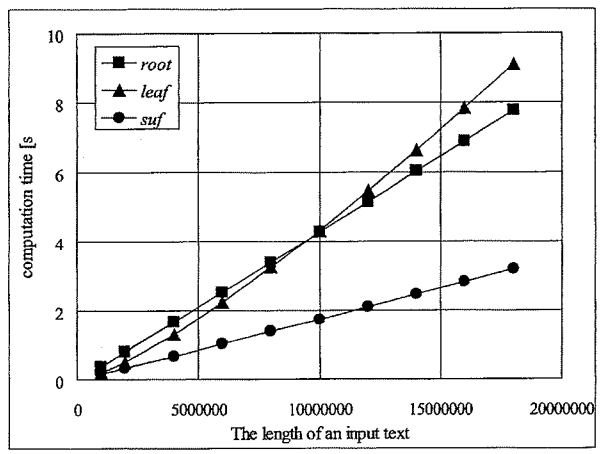

図 14　実験結果 2：文字の出現確率を $\{P(\mathrm{A}) = P(\mathrm{T}) = 0.4, P(\mathrm{C}) = P(\mathrm{T})0.1\}$ としたときの実行時間の測定

Fig. 14　Experimental result 2: Computation time for random texts generated with probability $\{P(A) = P(T) = 0.4, P(C) = P(T) = 0.1\}$.

結果 2: 実行結果を図 14 に示す．*root*，*suf* は実験 1 とほぼ同じ速度で動作したが，*leaf* は遅くなっている．

結果 3: 実行結果を図 15 に示す．*root* と *leaf* では実行時間が増えたのに対し，*suf* ではほぼ同じ速度で実行された．これは，実ノードの深さが最大で n となるため，根からも葉からも毎回 $O(n)$ 個の実ノードをたどって境界ノードを見つける必要があるためであると考えられる．

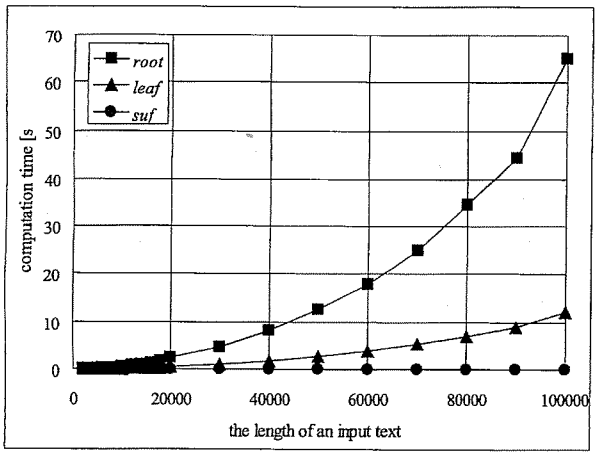

図 15 実験結果 3：T =AAA···A$ に対する実行時間の測定

Fig. 15 Experimental result 3: Computation time for a text T=AAA···A$.

6. むすび

本論文では，プロパティ接尾辞木の構築において，接尾辞木から不要なノードを削除する際，その境界となるすべての位置を $O(n \log |\Sigma|)$ 時間で計算する手法を提案し，その結果プロパティ接尾辞木の $O(n \log |\Sigma|)$ 時間オフライン構築が可能であることを示した．

今後の課題として，接尾辞木を作らずに，与えられたプロパティ付き文字列からプロパティ付き接尾辞木をオンライン構築する手法が望まれる．また，複数のプロパティが付随した文字列に対する索引の構築も応用面から興味深い．

謝辞 九州大学の竹田正幸先生，東北大学の石野明先生，北海道大学の湊真一先生，トーマス・ツォイクマン先生には，本研究に関して貴重な助言を頂きました．また，2 名の匿名査読者には有益なコメントを頂きました．ここに記して感謝致します．

文　献

[1] A. Amir, E. Chencinski, C. Iliopoulos, T. Kopelowitz, and H. Zhang, "Property matching and weighted matching," Proc. 17th Annual Symposium on Combinatorial Pattern Matching, LNCS4009, pp.188–199, Barcelona, Spain, July 2006.

[2] A. Amir, D. Keselman, G.M. Landau, M. Lewenstein, N. Lewenstein, and M. Rodeh, "Text indexing and dictionary matching with one error," J. Algorithms, vol.37, no.2, pp.309–325, 2000.

[3] A. Andersson, N.J. Larsson, and K. Swanson, "Suffix trees on words," 7th Annual Symposium on Combinatorial Pattern Matching, pp.102–115, Laguna Beach, USA, June 1996.

[4] J.C. Na, A. Apostolico, C.S. Iliopoulos, and K. Park, "Truncated suffix trees and their application to data compression," Theor. Comput. Sci., vol.304, pp.87–101, July 2003.

[5] M. Farach and S. Muthukrishnan, "Perfect hashing for strings: Formalization and algorithms," Proc. 7th Annual Symposium on Combinatorial Pattern Matching, LNCS 1075, pp.130–140, March 1996.

[6] M. Farach, "Optimal suffix tree construction with large alphabets," Proc. FOCS'97, pp.137–143, 1997.

[7] D. Gusfield, Algorithms on strings, trees, and sequences — Computer science and computational biology, Cambridge University Press, 1997.

[8] C.S. Iliopoulos and M.S. Rahman, "Faster index for property matching," Inf. Process. Lett., doi:10.1016/j.ipl.2007.09.004, 2007. (to appear)

[9] S. Inenaga and M. Takeda, "On-line linear-time construction of word suffix trees," Proc. 17th Ann. Symp. on Combinatorial Pattern Matching, LNCS4009, pp.60–71, 2006.

[10] N.J. Larsson, "Extended application of suffix trees to data compression," Proc. Conference on Data Compression, pp.190–199, 1996.

[11] B.S. Manjunath, P. Salembier, and T. Sikora (eds.), Introduction to MPEG-7: Multimedia Content Description Interface, John Wiley & Sons, 2002.

[12] E.M. McCreight, "A space-economical suffix tree construction algorithm," J. ACM, vol.23, pp.262–272, 1976.

[13] G. Navarro and M. Raffinot, Flexible pattern matching in strings, Cambridge University Press, 2002.

[14] E. Ukkonen, "On-line construction of suffix trees," Algorithmica, vol.14, no.3, 249–260, 1995.

[15] P. Weiner, "Linear pattern matching algorithms," Proc. IEEE 14th Annual Symposium on Switching and Automata Theory, pp.1–11, 1973.

[16] 上村卓史，喜田拓也，有村博紀，"単語幅を制約した接尾辞木の効率のよい構築アルゴリズム，" 情報科学技術レターズ，vol.5, pp.5–8, 2006.

[17] 上村卓史，喜田拓也，有村博紀，"プロパティ付き接尾辞木の構築における境界発見アルゴリズム，" 第 18 回データ工学ワークショップ (DEWS2007)，電子情報通信学会，March 2007.

[18] 上村卓史，喜田拓也，有村博紀，"プロパティ接尾辞木：メタデータ付き系列データのための効率よい索引構造，" 第 5 回情報科学技術フォーラム講演論文集 (FIT2007)，vol.5, pp.45–46, Sept. 2007.

(平成 19 年 5 月 29 日受付，9 月 28 日再受付)

上村　卓史　（学生員）

2006 北大・工・情報工学卒．同年同大大学院情報科学研究科修士課程入学，現在に至る．2006 年度下期 IPA 未踏ソフトウェア創造事業（未踏ユース）「テキストマイニング技術を融合したウェブブラウザの開発」開発代表者．現在，情報検索と文字列処理アルゴリズムの研究に従事．

喜田　拓也　（正員）

2001 九州大学大学院システム情報科学研究科情報理学専攻博士後期課程了．同年，九州大学附属図書館研究開発室専任講師を経て，2004 北海道大学大学院情報科学研究科助教授，現在に至る．2000～2001 日本学術振興会特別研究員．2001 情報処理学会創立 40 周年記念論文賞を受賞．現在，文字列処理アルゴリズム，データ圧縮，電子図書館等の研究に従事．情報処理学会．博士（情報科学）．

有村　博紀

1990 九州大学大学院総合理工学研究科修士課程了．同年，九州工業大学情報工学部助手，同助教授を経て，1996 九州大学大学院システム情報科学研究科助教授，2004 北海道大学大学院情報科学研究科教授，現在に至る．1996 ヘルシンキ大学訪問研究員．2005 リヨン第一大学訪問研究員．1999～2002 科学技術振興事業団「さきがけ研究 21」研究員．現在，データマイニングと，計算学習理論，情報検索の研究に従事．1999 人工知能学会 2000 年度優秀論文賞，PAKDD 2000 Paper with Merit Award，本会 DEWS2002，2003，2004 優秀論文賞，2005 日本データベース学会上林記念研究奨励賞等を受賞．ACM，情報処理学会，人工知能学会会員．博士（理学）．