A-023

接尾辞木に対する二分木化と簡潔データ構造による圧縮
Compressing a Suffix Tree by Converting it to a Binary Tree and Using Succinct Data Structures.

馬場 雅大† 小野廣隆‡ 定兼邦彦§ 山下雅史‡
Masahiro Baba Hirotaka Ono Kunihiko Sadakane Masafumi Yamashita

1. はじめに

接尾辞木 (ST) は文字列検索をはじめとした多くの操作を実現するが，一方でサイズの大きさに課題を抱えている．そのため圧縮接尾辞配列 (CSA) や順序木の簡潔データ構造などを用いた圧縮接尾辞木 (CST) が考えられている．本稿では順序木の中でも全二分木に対する簡潔データ構造を示し，それを用いて CST のサイズを小さくする手法を示す．また、この手法では単純な存在問合せの計算量は CST と変わらない．

CST [5] は本来ポインタ表現などで明示的に持っている木構造・葉の情報・枝長などを CSA や簡潔データ構造を用いて表現したものである．ここで、検索対象のテキスト T の長さを n とすると木構造には最大 $4n+o(n)$ ビット必要である．この他枝長に $2n+o(n)$ ビット必要である．以下にサイズの比較を示す．

表 1:ST のサイズ (ビット)

ナイーブな ST	CST	提案表現
$O(n \lg n)$	$\|CSA\|+6n+o(n)$	$\|CSA\|+4n+o(n)$

まず 2 章では，簡潔データ構造について取り上げる．3 章で全二分木表現について説明し，4 章ではそれを活用して，CST での操作を模倣することを示す．

2. 簡潔データ構造

まず，本論文で用いる演算について説明する．計算モデルとしては語長 $\Theta(\lg n)$ ビットの word RAM を用いる．word RAM モデルは，$\Theta(\lg n)$ ビットの数に関する任意の算術演算や，連続する $\Theta(\lg n)$ ビットのメモリに対する入出力が定数時間で行える．

2.1.rank/select 操作と簡潔データ構造

$|\mathcal{A}| = \sigma$ となるアルファベット $\mathcal{A}$ 上の文字列 $S[1..n]$ に対する *rank/select* を以下のように定義する．

- $rank_c(S,i)$: $S[1..i]$ 中の c の出現回数
- $select_c(S,i)$: S 中で先頭から i 番目の c の位置

ビット配列 ($\sigma = 2$) の場合，これらを定数時間で計算する $n+o(n)$ ビットのデータ構造があり，さらに 1 の数が m 個であるビット配列に対して各操作を定数時間で計算する $\lg \binom{n}{m} + O(n \lg\lg n/\lg n) = m \lg \frac{n}{m} + \Theta(m) + O(n \lg\lg n/\lg n)$ ビットのデータ構造が存在する (Raman ら [4])．このデータ構造を完全索引付辞書 (fully indexable dictionary, FID) と呼ぶ．$m = O(n/\lg n)$ ビットであれば，このデータ構造のサイズは $O(n \lg\lg n/\lg n)$ となり $o(n)$ で収まる．

2.2. 順序木の括弧列表現と用いられるデータ構造

順序木の簡潔データ構造は BP [3] や DFUDS [1] などいくつか提案されているが，いずれもノード数 n の木を $2n+o(n)$ ビットで模倣する．

BP(Balanced Parentheses Encoding) 表現とは，木を前置順に巡回し，ノードの行きがけに開き括弧 '(' ，帰りがけに閉じ括弧 ')' を配置していくもの．括弧は全体でバランスし，括弧列 P の長さは $2n$ である．

n ノードの根付き順序木 T に対して，括弧列を用いて表現する BP では木の巡回を行うために，以下の操作を $o(n)$ ビットで定数時間で行う補助データ構造が提案されている [3]．対象となる括弧列を P とすると，以下の通りである．

- $findopen(P,x)$: 閉括弧 $P[x]$ に対応する開括弧の位置

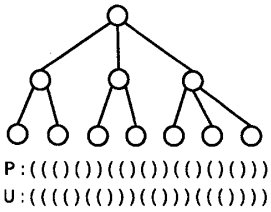

図 1: BP 表現 P と DFUDS 表現 U

- $findclose(P,x)$: 閉括弧 $P[x]$ に対応する開括弧の位置
- $enclose(P,x)$: 括弧 $P[x]$ とそれに対応する括弧を囲う最小の括弧対の開括弧の位置

前述の $rank$ などを用いて可能となる操作例を以下に示す．

- $isleaf(x)$: x は葉であるか．
- $parent(x), firstchild(x), sibling(x)$: x の親，長男，弟
- $degree(x)$: x の子の数 (次数)
- $child(x,i)$: x の左から i 番目の子
- $childrank(x)$: x がその親の左から何番目の子か
- $lca(x,y)$: x と y の最近共通祖先
- $LA(x,d)$: x の祖先で深さが d のもの

3. 全二分木の表現と操作プロセス

全二分木とは任意の内部ノードが子ノードを必ず 2 個持つような二分木のことを指す．全二分木のノード数を n としたとき，この情報理論的下限は $L_2 = n\text{-}\Theta(\lg n)$ ビットであり，順序木の $L_1 = 2n\text{-}\Theta(\lg n)$ よりも小さい．BP などは L_1 に近いサイズで木を表現しているわけだが本稿では全二分木に適用できる L_2 に近いサイズの括弧列表現を示す．

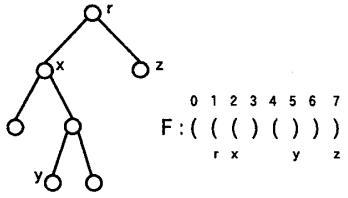

図 2: 全二分木表現の例

3.1. 全二分木表現の括弧列表現 [6]

全二分木に対して，前置順に内部ノードであれば開括弧 '(' ，葉ノードであれば閉括弧 ')' を配置していく．この場合，閉括弧が 1 つ多くなるので先頭に開括弧を追加する．この全二分木表現を F とする．F の長さは $n+1$ である．

ノード 1 個につき順番に 1 つずつ括弧を並べているので，$F[0,..n]$ 中，前置順で x 番目のノードを示している括弧は $F[x]$ である．

3.2. 全二分木表現における操作の方法

3 章で挙げた操作のうち，$findclose$ などを用いてできる操作の一部を示す．多くは DFUDS に関するもの [2] と似た方法をとっている．なお，lca は括弧列に対する超過数列の区間最小値問題 (RMQ) に帰着させることで $o(n)$ ビットの領域を用いて $O(1)$ 時間で得ることができる．

†九州大学大学院システム情報科学府
‡九州大学大学院システム情報科学研究院
§国立情報学研究所情報学プリンシプル研究系

•$isleaf(x)$: $F[x] =)$ ならば yes. それ以外 no.
•$parent(x)$: $F[x\text{-}1]=($ ならば x-1, それ以外 $findopen(x\text{-}1)$
•$firstchild(x) : x + 1$
•$lastchild(x) : findclose(x) + 1$
•$sibling(x)$: $F[x-1] =($ ならば $lastchild\ (parent\ (x))$, それ以外の場合は存在しない.

4.ST の変形と圧縮

CST [5] では ST の木構造に BP を用いる. テキスト S の長さを n とすると, 木のノード数は高々$2n$ 個なので, 最大 $4n + o(n)$ ビット必要である. ここでは木構造を全二分木に変更して, 前章の表現を適用して $2n + o(n)$ ビットにする.

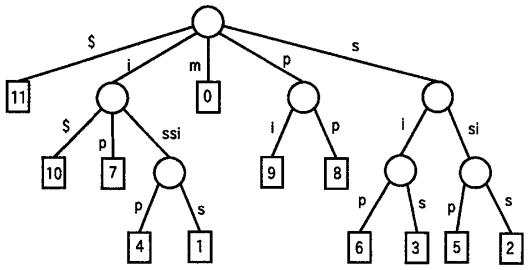

図 3: mississippi$ の接尾辞木

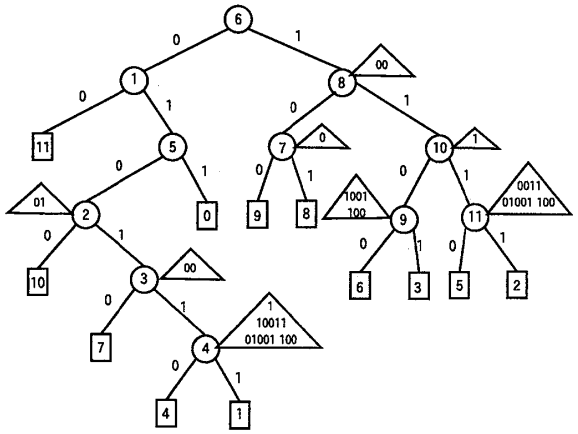

図 4: 図 3 の 5 ビット符号化での全二分木

4.1.CSA と CST

ST は n 個の接尾辞 $S[i:n]$ を葉におくものである. 存在問合せを n に依らない計算量で返す他, 接尾辞リンクを用いて様々な操作に対応する. しかし, ST を単純に実装すると $10n$ バイト以上の領域を消費する.

接尾辞配列 (SA) は各内部ノードから出る枝のラベルを正しくアルファベット順に並べた時, 接尾辞木の葉ノードと左から順番に対応するものである. 木や接尾辞リンクなどの情報を持たないためサイズは $4n$ バイトである. CSA はさらに使用領域を削減したものである. である. この CSA における演算の時間を以下のように定義する.
•t_{SA} :$SA[i]$ を返すのにかかる時間
•t_Ψ :$\Psi[i] = SA^{-1}[SA[i] + 1]$ を返すのにかかる時間
なお SA^{-1} は接尾辞配列の値と添字に関する逆関数である. CST はこの CSA を葉ノードの置いた上で, 木構造を BP 表現などの簡潔表現で表す. 本稿では CST 中の以下の操作を模倣することを考える.
•$CLD(v, c)$: v の子のうち間の枝ラベルが c で始まるもの
•$SL(v)$: ノード v の接尾辞リンク先
接尾辞リンクとはあるノードから他のノードを指すポインタである. 具体的にはあるノード v の根からの枝ラベルつなげたものをテキストの部分文字列 $S[i:j]$ としたとき, それが $S[i+1:j]$ となるノードを指すものである.

4.2. 接尾辞木の変形

各アルファベットを符号化した上で, 接尾辞木を構築する. 図 4 は '$' をテキストの末尾を示す終端記号としたとき, テキスト mississippi$ に対して $\{'\$' = 00000, 'i' = 01001, 'm' = 01101, 'p' = 10000, 's' = 10011\}$ と符号化して作ったものである. 三角形の内部は省略されたビットである. 0 と 1 の 2 値から構成されるこの木は全二分木となる. この形状はパトリシアトライであるが, 葉ノードは接尾辞を示し, 接尾辞リンクが存在する点などが異なる.

4.3. 接尾辞木巡回の模倣

操作 CLD, SL の全二分木化 CST での模倣を説明する.
(1)$CLD(v, c)$

CST では内部ノード v に対して子ノードとの間の枝ラベルが, c かどうか判定するために各子の子孫の葉からテキストの実際の文字を参照して, これらから c を二分探索にかける. アルファベットサイズを σ としたとき, v の子の数は高々 σ 個なので, $CLD(v, X)$ の計算量は $O(t_{SA} \lg \sigma)$ である.

全二分木化 CST でも基本的な方法は同じである. v の左ノード v_l の最右子孫葉を参照することで左右どちらに下るかを判断する. 終了条件は枝長の変化で判断する. 枝長 HGT は内部ノード v_l の中置順番号を v_{lj} としたとき, $SA[v_{lj}]$ と $2n + o(n)$ ビットの索引から定数時間で求めることができる. 下る回数は高々$O(\lg \sigma)$ なので, 計算量は CST のときと同様で $O(t_{SA} \lg \sigma)$ である.
(2)$SL(v)$

CST では v の最左子孫葉と最左子孫葉の葉番号をそれぞれ u_{ml}, u_{mr} とすると, $\Psi(u_{ml})$ と $\Psi(u_{mr})$ を求めた上で, これらに対応する葉の lca が $SL(v)$ となる. lca の計算が RMQ を用いて定数時間とすると $SL(v)$ の計算量は $O(t_\Psi)$ である.

全二分木化 CST で同様の方法をとる場合追加処理が必要である. これは元来の木構造で 3 個以上子を持っていたノードを分割しているためである. これに全二分木を上ることで対処する. 上る回数も枝長によって判断する. すなわちそもそも同一であったノードは枝長も変わらないので, その間は木を上っていくのである. 目標とするノードとの全二分木上での深さの差は $\lg \sigma$ 以内なのでこの回数は $O(\lg \sigma)$ 回である. 木を上るにつれて枝長も小さくなるので, LA を用いた二分探索により $O(\lg \lg \sigma)$ 回に削減できる. 以上より全二分木化 CST における $SL(v)$ は $O(t_\Psi + t_{SA} \lg \sigma \lg \sigma)$ となる.

5. おわりに

全二分木の簡潔データ構造を示すと共に, 木構造を全二分木に変換することで CST をさらに圧縮できることを示した. ただ $SL(v)$ の計算量に t_{SA} が入るのが課題点である.

参考文献

[1] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, S. S. Rao : "Representing trees of higher degree." *Algorithmica 43(4)*, 275–292, 2005.

[2] J. Jansson, K. Sadakane, and W.-K. Sung. : "Ultra-succinct representation of ordered trees."In Proc. ACM-SIAM SODA, pages 575-584, 2007.

[3] J. I. Munro, V. Raman : "Succint Representation of Balanced Parentheses and Static Trees." *SIAM Journal on Computing*, 31(3):762–776, 2001.

[4] R. Raman, V. Raman, S. S. Rao : "Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees and Multisets." In *Proc. ACM-SIAM SODA*, pages 233–242, 2002.

[5] K. Sadakane. : "Compressed Suffix Trees with Full Functionality." Theory of Computing Systems, 41(4):589-607, 2007.

[6] 馬場雅大, 小野廣隆, 定兼邦彦, 山下雅史, : "全二分木の簡潔な表現", 2009 年度冬の LA シンポジウム