情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-AL-129 No.1
2010/3/5

全二分木の簡潔な表現

馬場雅大[†1]　小野廣隆[†1]
定兼邦彦[†2]　山下雅史[†1]

大規模なデータを効率よく扱うために圧縮されたデータ上で高速な操作を行える簡潔データ構造が提案されてきた．本稿ではパトリシアトライなどに使われる全二分木に焦点を当てる．提案する全二分木の簡潔表現のサイズは $n+o(n)$ ビットであり，右の子へ巡回するといった操作を定数時間で行うことができる．また接尾辞木 (ST) を全二分木に変換することで圧縮可能であることを示す．

A succinct representation of a full binary tree

Masahiro Baba,[†1] Hirotaka Ono,[†1]
Kunihiko Sadakane[†2] and Masafumi Yamashita[†1]

To take large-scale data efficiently, succinct data structures that support high-speed operations on compressed data have been developed. In this paper, we focus on full binary trees which are used as patricia tries. We propose for a succinct data structure whose size is $n+o(n)$ bits. This representation enables constant time operations such as traversing from a internal node to its right child node. Then, we show that a suffix tree (ST) is compressible by converting it to a full binary tree.

1. はじめに

1.1 簡潔データ構造の概要

大規模なデータに対しても小さなサイズで高速な操作を実現するのが簡潔データ構造である．簡潔データ構造はあるデータに対する簡潔表現とそれに対して基本的な操作を語長 $\Theta(\lg n)$ ビットの word RAM モデル上で行うことを可能にする簡潔索引とで構成される．簡潔表現のサイズは情報理論的下限にほぼ一致する．情報理論的下限とは，位数 L の集合の中のある 1 パターンを格納する場合のデータ構造サイズの下限であり，$\lg L$ ビットである．例えば長さ n のビット列であれば，$L=2^n$ 通り考えられるので情報理論的下限は $\lg L = n$ ビットといった具合である．一般に簡潔表現のサイズはおおよそ $\lg L$ ビットであり，簡潔索引は $o(\lg L)$ ビットである．簡潔データ構造の適用例としてはビット列・文字列の他，木やグラフなどがあり，それぞれで異なる表現が考え出されている．その中でも本稿が取り上げるのは根付き順序木に対するものである．根付き順序木は多くのデータ格納に用いられているデータ表現の１つである．具体例として決定木や符号木，XML DOM が挙げられる．

順序木を実装する典型的な方法はポインタでノードを指し示すものである．しかし，１つのポインタのサイズが４バイトであったとき，あるノードの親ノードを求めるためには，$4n$ バイト必要になる．そこで順序木に対しても簡潔データ構造が考えられてきた．ノード数を n としたとき順序木のパターン数はカタラン数を用いて $C_n = \binom{2n-1}{n-1}/(2n-1) = 2^{2n}/\Theta(n^{\frac{3}{2}})$ 通りなので情報理論的下限は $2n-\Theta(\lg n)$ ビットである．順序木の簡潔データ構造としては括弧列を用いる BP・DFUDS や Tree Covering が考えられており，いずれもサイズは $2n+o(n)$ ビットである．

1.2 本研究

BP 表現などはメモリ使用領域の改善になるが，通常の順序木よりも情報理論的下限が小さい全二分木に対して適用しても $2n+o(n)$ ビットである．そこで本稿では DFUDS を土台に $n+1+o(n)$ ビットの括弧列で全二分木を表現し，かつ右の子へ巡回するといった基本的な操作を定数時間で行う簡潔データ構造を提案する．

表 1.1:全二分木の表現サイズ (ビット)

ポインタ表現	簡潔表現	提案表現
$O(n\lg n)$	$2n+o(n)$	$n+1+o(n)$

2. 準備

まず，本論文で用いるビット配列の扱い方や順序木の括弧列表現に用いられ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

†1 九州大学 Kyushu University
†2 国立情報学研究所 National Indtitute for Informatics

⋆1 lg は底 2 の対数を表すとする

リに対する入出力が定数時間で行える．

2.1 rank/select を実行するための簡潔データ構造

$|\mathcal{A}| = \sigma$ となるアルファベット $\mathcal{A}$ 上の文字列 $S[1..n]$ を考える．ここで S に対する $rank$ と $select$ 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

最も基本的なものはビット列 ($\sigma = 2$) の場合であり，定数時間で rank/select を計算する $n+o(n)$ ビットのデータ構造がある[5]．また 1 の数が m 個であるビット列に対して rank/select を定数時間で計算する $\lg \binom{n}{m} + O(n \lg\lg n / \lg n) = m \lg \frac{n}{m} + \Theta(n) + O(n \lg\lg n / \lg n)$ ビットのデータ構造が存在する (Raman ら[7])．このデータ構造は完全索引付辞書 (FID) と呼ばれている．FID の特徴は 0 と 1 の数に隔たりがある場合，例えば $m = O(n/\lg n)$ ビットのとき，サイズは $O(n \lg\lg n / \lg n)$ となり $o(n)$ で収まる．

また rank/select などで用いられる表引きは括弧列でも重要である．表引きとは小さいサイズの問題に対して，そのサイズで考えられるすべてのパターンの入力に対してあらかじめ答えを求めておき，表に格納するものである．任意の長さ $\frac{1}{2}\lg n$ のビット列は $2^{\frac{1}{2}\lg n} = \sqrt{n}$ 通り存在するので，それに対して 1 つ 1 つの答えが定数 C を用いて $O(\lg^C n)$ ビットであれば，表のサイズは ($\sqrt{n}$ 通り) $\cdot$ (長さ $\frac{1}{2}\lg n$) $\cdot$ (答え $O(\lg^C n)$) $= o(n)$ ビットとなる．なお括弧列は開・閉括弧のビット列とみなすことができ，表引きを同じように適用できる．

2.2 括弧列を用いた順序木の簡潔表現

2.2.1 BP(Balanced Parenthesis Encoding) 表現[6]

BP 表現とは，木を前置順で巡回し，ノードの行きがけに開き括弧 '(' を配置し，帰りがけに閉じ括弧 ')' を配置していくものである．どの開き括弧もそれに対応する閉じ括弧が後方に作られるので，括弧は全体でバランスする．1 つのノードに 2 つの括弧対を配置することになるのでこの括弧列 P の長さは $2n$ である．

2.2.2 DFUDS(Depth-First Unary Degree Sequence) 表現[1]

DFUDS 表現とは，木を前置順で巡回しノードの行きがけにそのノードの次数分だけ開き括弧を並べた後に閉じ括弧を 1 個おくものである．各ノードで次数分 (子の数) の開き括弧のあと閉じ括弧が出る構図になり，全体的には枝の数だけ開き括弧とノード数の数だけ閉じ括弧とが存在する．ここで括弧列の先頭に開き括弧を 1 つだけ追加することで，BP と同じように括弧は全体でバランスする．この括弧列を U とすると，U の長さは $2n$ である．

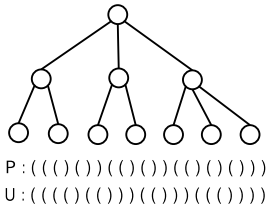

図 1 順序木に対する BP，DFUDS の例
BP 表現を P，DFUDS 表現を U としている．

2.2.3 Tree Covering(TC)[3]

TC では順序木を一定範囲のサイズのグループ (部分木) に分割して，それぞれを DFUDS などの簡潔表現で表すといった手法が採られる．この点は BP，DFUDS とは異なる (図 2)．

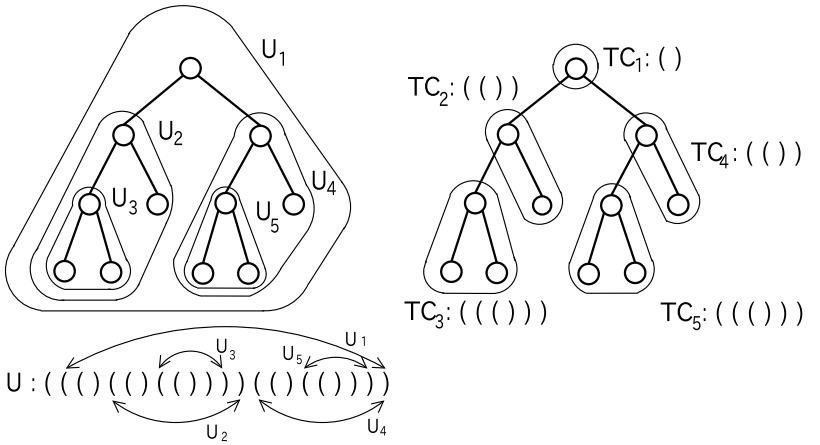

図 2 DFUDS と TC の違いの簡単な図
DFUDS 表現 U に対してはいくつかの部分が括弧列のどの範囲なのかを示している．図のように TC では分割された各グループごとに DFUDS で表すことになる．

2.2.4 BP，DFUDS で共通して用いられるデータ構造と基本的な操作

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，DFUDS では木の巡

回を行うために，以下の操作を定数時間で行う補助データ構造が提案されている[6]．対象となる括弧列を P とすると以下の通りである．

- $findopen(P,x)$: 開き括弧 $P[x]$ に対応する閉じ括弧の位置.
- $findclose(P,x)$: 閉じ括弧 $P[x]$ に対応する開き括弧の位置.
- $enclose(P,x)$: 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き括弧の位置.

上記の 3 演算と rank/select などを用いて以下の操作が可能である．

- $isleaf(x)$: x は葉であるか.
- $parent(x), firstchild(x), sibling(x)$: x の親，長男，弟
- $preorder\_rank(x)$: x の前置順
- $preorder\_select(i)$: 前置順が i であるノードの簡潔表現での位置
- $desc(x)$: x の子孫の数
- $degree(x)$: x の子の数 (次数)
- $child(x,i)$: x の左から i 番目の子
- $child(x,i)$: x がその親の左から何番目の子であるか
- $lca(x,y)$: x と y の最近共通祖先
- $depth(x)$: x の深さ
- $LA(x,d)$: x の祖先で深さが d のもの

なお $findopen$ などのみでは，BP は $child$，$degree$ を定数時間で返すことはできない．逆に DFUDS は BP では分かりやすい $depth$ などに関する操作が苦手である．

2.3 区間最小値問合せとそれを用いた最近共通祖先の求め方

2.3.1 区間最小値問合せ (RMQ)

区間最小値問合せ (RMQ) とは，以下のような問題である．

問題 1 配列 $E[1..n]$ に対する区間最小値問合せ $RMQ_E(i,j)$ は部分配列 $E[i,j]$ の中の最小値の添字を求める．最小値が複数存在する場合，その中で一番小さい添字を求める．

2.3.2 最近共通祖先 (LCA)

DFUDS において LCA を RMQ から求める方法[4] の概略は以下である．

U の超過数列 E を $E[i] = (U[0..i]$ 中の (の数) $-$ ($U[0..i]$ 中の) の数) と定義する．この E は長さ $2n$ で先程述べたような ± 1 配列であり，括弧は全体でバランスしているので値はすべて 0 以上である．この E に対する RMQ は $o(n)$ ビットのデータ構造を用いて定数時間である[8]．またここから LCA についても求めることができる．

補題 1 順序木上で 2 つのノード x と y の最近共通祖先は DFUDS と $o(n)$ ビットの補助データ構造を用いれば定数時間で求めることができる．

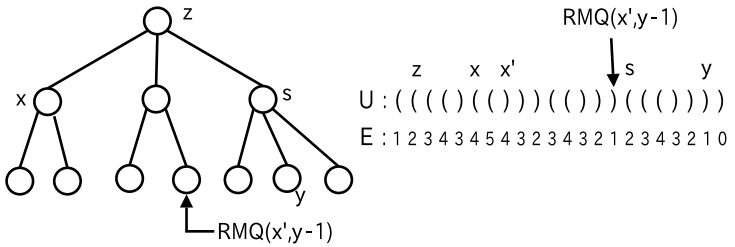

図 3 最近共通祖先の例
順序木に対する DFUDS 表現 U とその超過数列 E

x, y, s, z は各ノードの U での位置と考える (図 3 参照)．$x = y$ のときは $x = y = z$ と自明．$z = lca(x,y)$ $(x < y)$ を求める手順は以下の通りである．
もし x が葉ノードであれば $x' := x$，そうでなければ $x' := firstchild(x) - 1$ とする．ここで x' はノード x を表す括弧列の末端の U での位置である．
$s := RMQ_E(x, y-1) + 1$
$z := parent(s)$
s は求める z の子のうち y を子孫に持つようなノードであり，s の親が $z = lca(x,y)$ になる．

3. 全二分木の表現と操作のプロセス

3.1 全二分木とは

全二分木とは二分木の一つで，すべてのノードの次数が 0 または 2 となるようなものである．n ノードの全二分木はカタラン数を用いて $C_{(n-1)/2}$ 通り存在することから，情報理論的下限は $n - \Theta(\lg n)$ ビットである．なお n は必ず奇数である．

3.2 全二分木の括弧列表現

3.2.1 全二分木を表す括弧列

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

括弧列を前置順に表現していくほか，括弧列の先頭に仮想根として開き括弧を追加するのは DFUDS と同じである．しかし全二分木の場合，次数は 0 か 2 のどちらかである．そこ

Vol.2010-AL-129 No.1
2010/3/5

で DFUDS の次数を表す部分を削減し，次数が 2 すなわち内部ノードであれば (を配置し，次数 0 すなわち葉ノードであれば 0 を配置するようにしたものである．また順番に 1 つずつ括弧を並べているので，$F[0,\ldots,n]$ の中で前置順で x 番目のノードを示している括弧は $F[x]$ である．

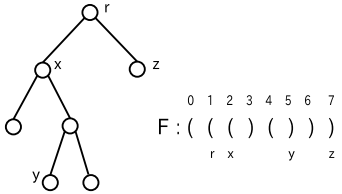

図 4 $n = 7$ のときの全二分木と全二分木表現 F の例
ノードの横でついている添字 r，x，y，z での F での位置はそれぞれの前置順と一致する．

3.2.2 全二分木表現における操作の方法

$findopen$ などを用いてできるものを示す．$lastchild(x)$ は右の子である．

- $isleaf(x) : F[x] = \underline{)}$ であれば yes．そうでなければ no.
- $parent(x) : F[x-1] = \underline{(}$ であれば $x-1$，そうでなければ $findopen(x-1)$
- $firstchild(x) : x+1$
- $lastchild(x) : findclose(x)+1$
- $sibling(x) : F[x-1] = \underline{(}$ であれば $lastchild(parent(x))$.
- $degree : F[x] = \underline{)}$ であれば 0．そうでなければ 2．
- $desc(x) : F[x] = \underline{)}$ であれば 0．そうでなければ $findclose(enclose(x)) - x + 1$
- $childrank(x) : F[x-1] = \underline{(}$ ならば 1．そうでなければ 2．
  また，lca についても RMQ を用いればよい．2.3 節のときと同様に $z = lca(x,y)$ とする．$x < y$ のとき $s = RMQ_E[x, y-1] + 1$ で z の子で y を子孫にもつノード s を見つけ出すことができるので，この $parent(s)$ が z になる．

ここで lca の他 $depth$，LA の 2 つの操作はついては $findopen$ などだけでは求めることはできないのは DFUDS と同じである．

4. 木の分割と深さを求める手法

TC では木の分割を用いてあるパラメータの範囲の大きさの部分木に分割する．本稿では Farzan,Munro の分割アルゴリズム[2] を用いる．

4.1 Farzan，Munro の木分割アルゴリズム

Farzan,Munro のアルゴリズムはあるパラメータ L に対して，ノード数 n の木 T をノード数 $2L$ 以下の部分木 $O(n/L)$ 個に分割する．

定義 1 L を一定のパラメータとして以下のようなものをそれぞれ定義する．

定義した名前	条件
$heavy-node$:	そのノード自身も含めて子孫の数が L 個以上であるノード
$heavy-subtree$:	$heavy-nodes$ とそれらをつなぐ枝のみで構成された T 上の部分木
$petiolar-node$:	$heavy-node$ を子に持たない $heavy-node$
$stalk-node$:	$heavy-node$ を 1 つだけ子に持つ $heavy-node$
$branching-node$:	少なくとも 2 つの $heavy-node$ を子に持つノード
$branching-edge$:	$branching-node$ とその子供である $heavy-node$ をつなぐ枝

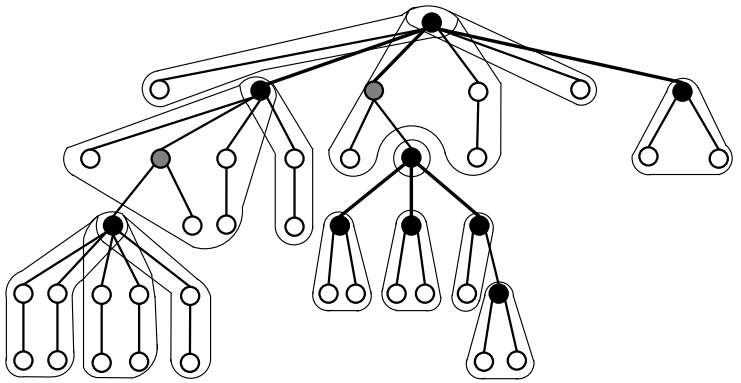

図 5 $L = 3$ で分割した例
ノードを囲んでいる実線は分割アルゴリズム[2] を $L = 3$ で実行した後のグループを示している．また色つきノードは $heavy-node$ であることを示しており，黒丸はそのうちグループの根となったものを示している．

T' を T の $heavy-subtree$ とする．定義から以下の補題が得られる．

補題 2 順序木 T のパラメータ L に対する $branching-node$ と $branching-edge$ の数は，それぞれ高々 n/L 個，$2n/L$ 個である．

またこれにより以下の定理が得られる．

定理 1 Farzan，Munro の分割アルゴリズムにより順序木 T は，以下のような性質を持つ部分木のグループに分割できる．

(1) T は大きさが高々$2L$ のグループに分割でき，その際グループの数は $O(n/L)$ 個とすることができる．
(2) これらのグループはそのグループの根の部分を除けば，互いに素である．
(3) 各グループにおいてその根ノードから出る枝を除けば，グループ内のノードから他のグループにつながるような枝は高々1 本である．

4.2 グループの根ノードを求める手法

Farzan，Munro の分割アルゴリズム[2] により，あるノード x はいずれかのグループに属しているとする．このときのグループの根を $R[x]$ として以下のような問題を考える．

問題 2 x,$R[x]$ をノードの前置順としたときノード x を含むグループの根 $R[x]$ を返す．

4.2.1 RMQ を利用した $R[x]$ を求める手法の提案

提案 1 根であることを表す長さ $2n$ のビット配列 $B_r[1,\ldots,2n]$ と DFUDS 表現 U における LCA を用いて，U 上の x の位置から $R[x]$ の位置を求める．

U 上で p,q の位置にあるノードの LCA の位置を $lca(p,q)$ とする．またグループの根となっているあるノードの U での位置を c としたとき，$B_r[c]=1$ となるような長さ $2n$ のビット配列をつくる．グループの根の数は $O(n/L)$ なので $L=\Theta(\lg n)$ であれば，FID を用いた B_r のサイズは $o(n)$ である．

木を分割した後の状態では，前置順で x の直前にあるグループの根が $R[x]$ になるとは限らない．しかし，定理 1 より各グループはその根を除いて，他のグループに伸びている枝は高々 1 本である．そのため前置順で巡回したとき，x と $R[x]$ の間では以下の 4 パターンが考えられる．分かりやすくするため x，$R[x]$，s などはそれぞれ U での位置とする．

パターン

(1) $R[x]$ と x の間に他にグループの根は存在しない．
(2) $R[x]$ と x の間に他のグループの根が存在する．それは $R[x]$ の子のうち x を子孫に持つノード s を根とする部分木には存在せず，$R[x]$ と s の間に 1 個以上存在している．
(3) $R[x]$ と x の間に他のグループの根が存在する．それは s を根とする部分木に含まれている．つまり s と x の間には 1 個以上存在しているが $R[x]$ と s の間には存在しない．
(4) $R[x]$ と x の間に他のグループの根が存在する．それは s と x の間，$R[x]$ と s の間にそれぞれ 1 個以上存在している．

ここで最も単純なのはパターン 1 の場合であり，直前にある根を示せばよいので B_r に対する rank/select を用いて $select_1(B_r,(rank_1(B_r,x)))$ で求めることができる．しかし，パターン 2 〜 4 のような場合も考えられる．特にパターン 4 の場合，$R[x]$ と x の間にある他のグループの根は少なくとも 2 個存在する．そこで LCA を利用する．

x から $R[x]$ を求めるアルゴリズム

1. $x' := select_1(B_r, rank_1(B_r,x))$
2. $y := lca(x,x')$
3. $y' := select_1(B_r, rank_1(B_r,y))$
4. $w := lca(y,y')$

この結果得られるノード w が $R[x]$ となる．

4.3 Farzan の分割を用いて括弧列から深さを求める方法の提案

全二分木の表現 F は DFUDS と似通っており，DFUDS で深さを求める Jansson らの方法[4] を流用できる．この方法は括弧列に上位・下位パイオニアという概念を用いる．少数の上位パイオニアに深さを格納し，その他のノードについてはパイオニアとの深さの関係をポインタや表を用いるというものである．また TC では大きな部分木とその内部の小さな部分木という 2 層のデータ構造を用いる．本稿では TC の考え方を応用しながら，深さを求めるのに必要な括弧列のみを取り出す方針をとる．

$depth(x)$ を求めるため，まず表引きによって x と $R[x]$ の差を求めるという点では，TC を用いる方法と同じである．しかし TC のように各グループそれぞれで括弧列を記憶しているわけではないので，木全体を表している括弧列から必要な括弧列のみを取り出す．その上で，$R[x]$ の深さについてはさらに大きなグループの根に再帰せずに求める．分かりやすくするため以下のように，

$depth(x)=$(① x と $R[x]$ の深さの差 $depth(\delta)$) + (② $R[x]$ の深さ $depth(R[x])$).

としてこの 2 つの求め方を示す．

4.3.1 全二分木表現の場合

[3] に対して本論文では木全体を 1 つの DFUDS で表現しておき，その中から必要な括弧列のみを定数回の操作で取り出して表引きすることを考える．

①x と $R[x]$ の深さの差 $depth(\delta)$ の求め方

$x=R[x]$ のときは自明なので考えない．まず $L=\frac{1}{4}\lg n$ として木を分割した上で，x から 4.2.1 節の方法で $R[x]$ を求める．その上で $R[x]$ と x の関係を 4 つに場合分けする．

(1) $R[x]$ と x との間に他のグループの根ノードが存在しない.
(2) $R[x]$ と x との間には 1 個以上他のグループの根ノードが存在する.
  (a) x は $R[x]$ の左分木に存在.
  (b) x は $R[x]$ の右分木に存在.
    (i) $R[x]$ の右分木内では $R[x]$ と x の間に他のグループの根が存在しない.
    (ii) $R[x]$ の右分木内で $R[x]$ と x の間に他のグループの根が存在する.

4.2.1 節のアルゴリズムより 1. と 2. では $x' = R[x]$ となっている場合では 1. であり，そうでない場合は 2. となる．また (a) と (b) は $R[x]$ の左分木が $R[x]+1 \sim findclose(R[x])$ の範囲に収まることから x がその中に収まるか収まらないかで判断できる．さらに i. と ii. では $lastchild(x) \leq x' < x$ であれば ii であり，そうでなければ i である．

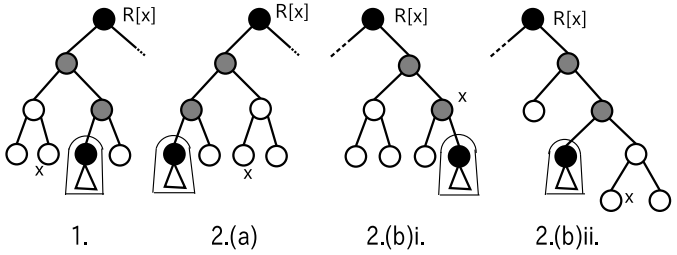

図 6 x と $R[x]$ の位置などで場合分けした例
x は $R[x]$ を根としたグループに存在する．x と $R[x]$ が共に含まれないグループは実線で囲んでいる部分.

各場合で取り出す括弧列を $Q[1,\ldots,k_x]$ とする．以下が取り出す Q のパターンである．

- *1.* の場合は
  $Q[1,\ldots,x-R[x]+1] := F[R[x],\ldots,x]$
- *2(a)* の場合は
  $Q[1,\ldots,y-R[x]+1] := F[R[x],\ldots,y]$
  $Q[y-R[x]+2,\ldots,y-R[x]+2+x-lastchild(y)] := F[lastchild(y),\ldots,x]$
- *2(b)i* の場合は
  $Q[1] := ($
  $Q[2,\ldots,x-lastchild(R[x])+2] := F[lastchild(R[x]),\ldots,x]$
- *2(b)ii* の場合は
  $Q[1] := ($
  $Q[2,\ldots,y-lastchild(R[x])+2] := F[lastchild(y),\ldots,y]$
  $Q[y-lastchild(R[x])+3,\ldots,y-lastchild(R[x])+3+x-lastchild(y)]$
  $:= F[lastchild(y),\ldots x]$

ここで k_x は取り出したい括弧列の長さであるので，*1.* の場合なら $k_x = x - R[x] + 1$，*2(b)ii* の場合ならば $k_x = y - lastchild(R[x]) + 3 + x - lastchild(y)$ である．*2(b)ii* のような場合に，括弧列を取り出してみた例が図 *8* である．もともとの全二分木の形とはまったく違うが x を示している部分の $depth(\delta)$ は変わらないことが分かる．

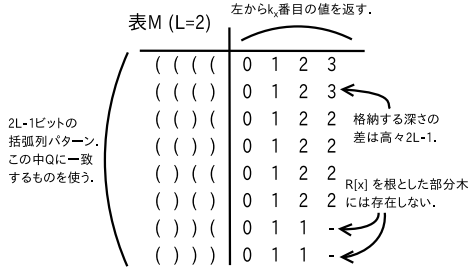

図 7 L=2 での表 M の例
$L = 2$ としたとき，取り出した括弧列は最大で $2L = 4$ まで考えられるので 4 ビットのビット配列に対する表をつくることになる.

F 上で $R[x]$ を示していた括弧から x を示していた括弧までで表引きを行う．表引きに必要なのは Q 内で $R[x]$ を示していた括弧から x を示していた括弧にいたるまでの部分である．図 *7* は $L = 2$ のとき Q と k_x から $depth(\delta)$ 表 M の例である．深さの差は $M[Q][k_x]$ のところを見ることになる．$L = \frac{1}{4}\lg n$ としているので表引き可能である．表のサイズは $2^{2L-1} \cdot (2L-1) \cdot \lg(2L-1) = O(\sqrt{n}\lg n \lg\lg n) = o(n)$ ビットである．

② $R[x]$ の深さ $depth(R[x])$ の求め方

本稿では $R[x]$ の深さを前後の深さの差を格納した 1 進数配列から求める．各 $R[x]$ の深さの差を順番に格納した際，増加分と減少分の合計はそれぞれ高々n である *(*図 *9* 参照*)*.

グループの根となっているノードにのみ深さを格納する．根の数は $O(n/L)$ である．これらのノードについて前置順では単調増加ではない．そこで以下の手法をとる．グループの根となっているノードが前置順で k 番目にくるとき r_k と表すことにする．$1 \leq k \leq$(グルー

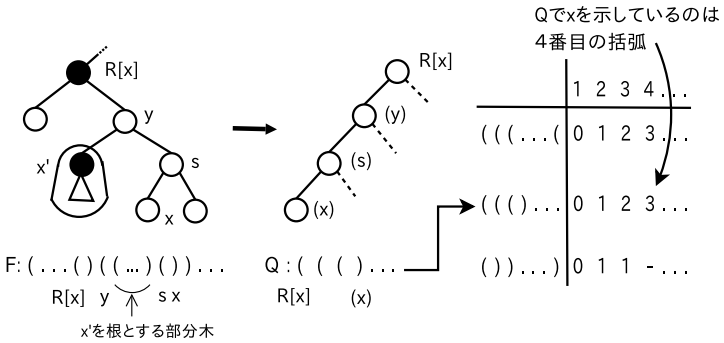

図 8 取り出す括弧列と表引きで $depth(\delta)$ を求める表の例
全二分木を表している F から長さ $k_x = 4$ の括弧列を取り出してそこから全二分木を描いたものと，表引きで $depth(\delta) = 3$ を求める図．$Q[k_x + 1, \ldots]$ は何を入れてもよい．

プの根の数*)* である．ここで $D_r[k] = depth(r_k)$ とする．この配列を基に以下のようなビット配列と 1 進数配列の *FID* を作成する．なお木 T の根 r_1 の深さは 0 とする．

A： $depth[r_{k-1}] < depth[r_k]$ ならば 0，それ以外は 1 となるビット配列．ただし $A[1] = 1$ とする．

D_1： $depth[r_{k-1}] < depth[r_k]$ ならば 0 を $depth[r_k] - depth[r_{k-1}]$ 個並べてその次に 1 をおく 1 進数配列．

D_2： $depth[r_{k-1}] \geq depth[r_k]$ ならば 0 を $depth[r_{k-1}] - depth[r_k]$ 個並べてその次に 1 をおく 1 進数配列．

A は前の配列との増加減少をビット配列で表している．前のノードと変わらない場合は減少に含む．D_1 と D_2 はそれぞれ増加・減少した場合の度合いを 1 進数で表現している．

サイズは配列 A は根ノードの数だけ存在するので $O(n/L)$ である．また D_1 は根ノードから各葉ノードへつながる経路の深さの差分を 1 つ 1 つとっていった場合，その合計が高々 n となることから D_1 の 0 の数は高々n 個であり，1 の数は根ノードの数と同じ高々$O(n/L)$ 個である．これは *FID* を用いて $o(n)$ ビットで表すことができる．D_2 も同様である．

$R[x]$ の深さは以下のように求める．まず $R[x]$ がグループの根としては i, $(1 \leq i \leq k)$ 番目にくるとしたとき，$i = rank_1(B_r, R[x])$ である．木そのものの根から $R[x]$ にいたるまでに i_1 回増加して i_2 回減少してきたとすると，$i_1 = rank_1(A, i)$, $i_2 = i - i_1$ である．これらを用いて $R[x]$ の深さ $depth(R[x])$ は以下のようになる．

配列 A，D_1，D_2 を用いて深さを求める手順

$$\begin{aligned} depth(R[x]) = \; & depth(r_i) = D_r[i] \\ = \; & (D_1 \text{ で } i_1 \text{ 番目の 1 が出るまでの 0 の数}) \\ & -(D_2 \text{ で } i_2 \text{ 番目の 1 が出るまでの 0 の数}) \\ = \; & \{(D_1 \text{ での } i_1 \text{ 番目の 1 の位置 } m_1) - (D_1[1, \ldots, m_1] \text{ の 1 の数})\} \\ & -\{(D_2 \text{ での } i_2 \text{ 番目の 1 の位置 } m_2) - (D_2[1, \ldots, m_2] \text{ の 1 の数})\} \\ = \; & \{select_1(D_1, i_1) - i_1\} - \{select_1(D_2, i_2) - i_2\} \end{aligned}$$

このため新たに必要となるのは，i_1 と i_2 を求めるための A の *rank* 索引とそれにより深さの増加・減少分をそれぞれ導き出すための D_1・D_2 の *select* 索引である．

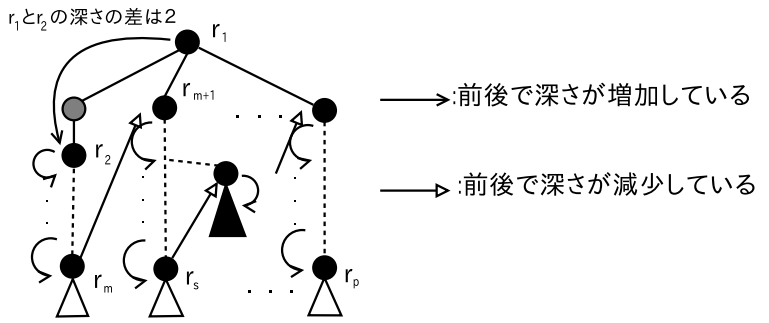

図 9 各 r_i の前後の深さの差についての概略
この図の $depth(r_2)$ は $depth(r_1)$ から 2 増加しているので，$A[2] = 0$，$D_1[0, \ldots, 2] = 001$ とする．

4.3.2 DFUDS の場合

一般の順序木では全二分木のような左分木があれば必ず右分木があるという規則性を持たないため，単純にこちら側だけを省くといった操作を行うことができない．これは場合分けを変更することで解決できる．また表引きを $o(n)$ ビットで行うため $L = \frac{1}{8} \lg n$ とする．

5. 接尾辞木の圧縮

接尾辞木とはテキスト $S[1..n]$ の n 個の接尾辞 $S[i..n]$ を木構造で表すデータ構造である．木の巡回によって多くの操作に適用することができるが，実装上サイズが大きい．そのため圧縮接尾辞木 *(CST)* が提案されている．

Vol.2010-AL-129 No.1
2010/3/5

5.1 CST の概要

CST では木構造を *BP* 表現などを用いて表す．また木のボトムアップ巡回に必要な隣接する接尾辞間の最長共通接頭辞長の配列 $Hgt[i] = lcp(SA[i], SA[i+1])$ を接尾辞配列 $SA[i]$ を参照することで求めることができる．上記の 2 つのために $6n + o(n)$ ビット必要である．また $SA[i]$ を得るのにかかる計算量がそのまま $Hgt[i]$ を求める計算量である．この他，枝ラベルはテキスト $S[SA[i]]$ を参照する．

5.2 基本的な操作

では *BP* を用いた *CST* で以下を仮定している[9]．

- t_{SA} :$SA[i]$ を復元するのにかかる時間
- t_{Ψ} :$\Psi[i] = SA^{-1}[SA[i]+1]$ を求めるのにかかる時間

また以下の操作を導入している．

- $child(v, c)$: 内部ノード v の子で間のラベルが c のノード
- $sl(v)$: 内部ノード v の接尾辞リンク先のノード

接尾辞リンクとは根からのパスラベルが aw(*a* は文字，w は文字列) の内部ノードがパスラベル w の内部ノードを指すものである．

枝ラベルは $O(t_{SA})$ で求まる．$child(v, c)$ は最大 σ 個の子ノードを二分探索して $O(t_{SA} \lg \sigma)$ である．接尾辞リンクは $\Psi[i]$ と *LCA* を用いて求めることができ，$O(t_{\Psi})$ である．

5.3 全二分木への変換

接尾辞木の内部ノードは常に子を *2* つ以上持つ．そこで内部ノードを次数 *2* となるように分割して全二分木にする．これによって必要な追加サイズを $6n$ から $4n$ にすることができる．$child(v, c)$ は *CST* と同様に枝ラベルを一つ見て左右どちらかに進むかを判断する．計算量は同様に $O(t_{SA} \lg \sigma)$ である．また接尾辞リンクは $O(t_{\Psi} + t_{SA} \lg \lg \sigma)$ である．$t_{SA} \lg \lg \sigma$ の項が新たにつくのは，分割されたノードのうち最も浅いものを返すためである．

6. ま と め

本稿では全二分木に対する簡潔な括弧列のデータ構造を示した．このデータ構造が苦手とする *depth* に対して *TC* を応用する方法を示した．この方法は一般の順序木の *DFUDS* にも適用できる．この他，*CST* を全二分木上でシミュレートする概略を示したが，こちらは計算量が若干大きくなるのが課題である．

⋆2 σ はアルファベット数

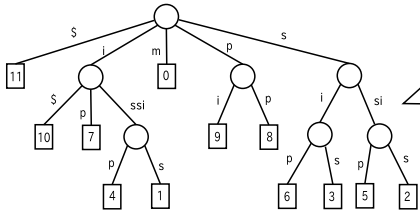

図 10 mississippi$の接尾辞木

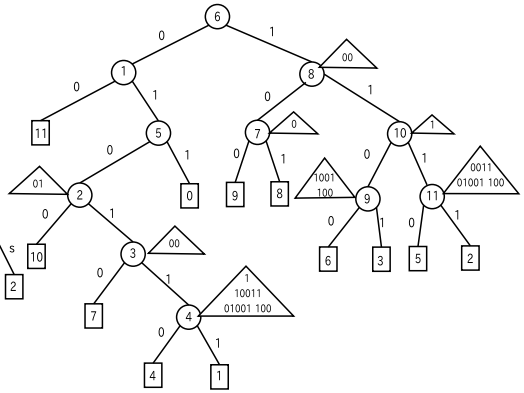

図 11 mississippi$の接尾辞木の全二分木化

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) *A. Farzan, J. I. Munro : "A Uniform Approach Towards Succint Representation of Trees."* Algorithm Theory - SWAT 2008*, pages. 173–184, 2008.*
3) *Richard F. Geary, R. Raman, V. Raman : "Succinct ordinal trees with level- ancestor queries."* ACM Transactions on Algorithms 2*, 4 (2006) 510–534.*
4) *J. Jansson, K. Sadakane, Sung, W.-K. : "Ultra-succinct representation of ordered trees." In* SODA (2007), N. Bansal, K. Pruhs, and C. Stein, Eds., SIAM*, pages. 575–584.*
5) *J. I. Munro : "Tables" In* Proceedings of the 16th Conference on Foundations of Software Technology and Computer Science (FSTTCS '96)*, LNCS 1180, pages 37–42, 1996.*
6) *J. I. Munro, V. Raman : "Succint Representation of Balanced Parentheses and Static Trees."* SIAM Journal on Computing*, 31(3):762–776, 2001.*
7) *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.*
8) *K. Sadakane : "Space-Efficient Data Structures for Flexible Text Retrieval Systems." In* Proc. ISAAC2002*, pages. 14–24. LNCS 2518, 2002.*
9) *K. Sadakane: "Compressed Suffix Trees with Full Functionality." Theory of. Computing Systems, 2007.*