文書列挙問題に対する実用的なデータ構造

定兼　邦彦†　　渡邉　大輔††

† 東北大学大学院情報科学研究科システム情報科学専攻
〒 980–8579 仙台市青葉区荒巻字青葉 09
†† 東北大学工学部情報工学科
E-mail: †{sada,watanabe}@dais.is.tohoku.ac.jp

あらまし　文書列挙問題とは，あるパタンを含む文書を列挙する問題である．これは文書検索において基本的な問題であり，文書検索システムはこの操作を備える必要がある．この問題を解くために転置ファイルがよく用いられているが，任意のパタンに対しては不可能である．任意のパタンに対するアルゴリズムでは接尾辞木などを用いるためデータ構造のサイズが大きかった．これに対し定兼のデータ構造は省スペースであり問い合わせ時間も高速であるが構造が複雑である．本論文ではこのデータ構造を簡略化し容易に実装できるものを提案する．

キーワード　テキスト DB, 転置ファイル, 接尾辞配列

Practical Data Structures for the Document Listing Problem

Kunihiko SADAKANE† and Daisuke WATANABE††

† Graduate School of Information Sciences, Tohoku University
Aramaki Aza Aoba09, Aoba-ku, Sendai 980–8579, Japan
†† Faculty of Engineering, Tohoku University
E-mail: †{sada,watanabe}@dais.is.tohoku.ac.jp

Abstract　The document listing problem is the one to enumerate all documents that contain a given pattern. This is a basic problem in document retrieval and any document retrieval system should support that operation. Though the inverted files are used for the problem, they cannot support the operation for any pattern. While an algorithm which can solve the problem for any pattern has been proposed, the size of the data structure is large. Sadakane has proposed a data structure which supports efficient queries for the problem using small amount of memory. Although it has theoretically the advantages, its data structure is too complex to implement. This paper simplifies the data structure and implements it. Experimental results shows that it is of small size and supports efficient queries.

Key words　text databases, inverted files, suffix arrays

1. はじめに

文書列挙問題は，テキストデータベースでの基本的な問題であり，テキストデータベースシステムではこの問題を解くことが必須である．

［問題 1］（文書列挙問題 [6]）　文書 $d_1, d_2, \ldots, d_k$ の集合が与えられたとき (前処理可)，パタン p に対する文書列挙問い合わせ list(p) とは p を含む全ての文書を出力することである．つまり，出力は $\{j | d_j[i..i+m-1] = p \text{ for some } i\}$ となる (m は p の長さ)．

テキストデータベースのためのデータ構造としては転置ファイルが広く用いられている．これは文書中の全ての単語に対して問い合わせの解を予め求め格納しておくものであり，高速に問い合わせを行うことができる．また，文書列挙問い合わせの解のみでなく，文書中の単語の頻度なども同時に格納しておく場合が多い．これらは文書の重み付け法の 1 つである *tf*idf* 法 [10] で用いられている．

文書列挙問い合わせのためのデータ構造として転置ファイルを用いる際の問題点としては，任意のパタンに対する問い合わせができない (非常に遅い) ということがある．その結果，複合語に対する問い合わせができなくなってしまう．これを解決するために Muthukrishnan [6] は接尾辞木と区間最小値問合わせ (RMQ, Range Minimum Query) のためのデータ構造を用いたアルゴリズムを提案している．これは前処理が $O(n)$ 時間 (n は文書の総長)，問い合わせが $O(|p| + q)$ 時間 (q は出力される文書の数) であり最適である．しかしデータ構造のサイズ

が大きいため実用的ではない．

Sadakane [8] は RMQ のための省スペースなデータ構造を提案し，これと圧縮接尾辞配列 [2] を用いた文書列挙問い合わせのための省スペースなデータ構造を提案した．データ構造のサイズは圧縮された文書の他に $4n + o(n)$ ビット必要なだけであり非常にコンパクトである．問い合わせ時間は $O(|p| + q \log^{\epsilon} n)$ 時間であり (ϵ は 0 から 1 の任意の定数)，最適に近い．欠点としてはデータ構造が複雑であり実装が難しいという点である．

実装が難しい理由の 1 つは，上記のアルゴリズムがビット演算を多用するということである．コンピュータのモデルは，$\log n$ ビットの四則演算やビット演算，サイズ $O(n)$ の配列の 1 要素のアクセスが定数時間で行えるというものである．このモデルの下でデータ構造をビット列で表すことでアクセス時間を落とさずにデータ構造のサイズを削減している．このようなデータ構造を実装する際に問題となるのは，CPU からメモリへのアクセスが 32 ビット単位であることと，扱う問題のサイズが大きくないために漸近的には無視できるデータ構造のサイズが実際には無視できないということである．

扱う問題のサイズを $n = 2^{32}$ とする．すると $\log n = 32$, $\log\log n = 5$ である．データ構造のサイズを削減するためにビット幅が $\log n$ ではなく $\log\log n$ である配列を用いることを考える．このときに 5 ビットの値を格納するために 8 ビットの配列を用いると無駄が生じる．そこで 5 ビットの値 32 個を 32 ビットの配列 5 つに詰め込むとスペースの無駄はなくなるが，アクセス速度が低下してしまう．

あるデータ構造のサイズが $n + o(n)$ ビットであるとする．この場合 n が非常に大きければサイズはほぼ n ビットということができる．しかしこのサイズが実際には $n + \frac{10n}{\log logn}$ ビットであるとすると，$n = 2^{32}$ のときにこれは $n + \frac{10n}{\log logn} = 3n$ ビットとなり，$o(n)$ の部分が無視できないということになる．つまりこの場合にデータ構造のサイズをほぼ n ビットにするにはデータ構造を変更する必要がある．

本論文では文書列挙問い合わせのための上記の 2 つのデータ構造の中間的なものを提案する．つまり，サイズが比較的小さく，かつ容易に実装できるという特徴がある．データ構造のサイズは圧縮された文書の他に $O(n)$ ビットであり，問い合わせ時間は $O((|p| + q)\log n)$ 時間である．実際には約 $13n$ ビットとすることができる．また，問い合わせ時間も十分高速である．

2. 準　　備

2.1 接尾辞木・接尾辞配列

文書を $d_1, d_2, \ldots, d_k$ とし，それを連結した文字列を T で表す．T の長さは n とする．T の s 番目から e 番目までの部分文字列を $T[s..e]$ で表す．T の接尾辞とは $T[j..n]$ $(j = 1, 2, \ldots, n)$ である．T の接尾辞木は T の全ての接尾辞を格納する木で，図 1 のようになる．葉の数字は根からその葉までのパス上の文字列で表される接尾辞 $T[j..n]$ の添え字 j である．この配列は接尾辞配列 ([3] 参照) と呼ばれ，SA で表す．接尾辞木を用いると任意のパタン P の出現個数 occ を $O(|P|)$ 時間で求めることができる．また，P の全ての出現位置を $O(occ)$ 時間で列挙することができる．接尾辞配列のみを用いる場合，出現個数は $O(|P|\log n)$ 時間で求まり，出現位置の列挙は $O(occ)$ 時間となる．パタンの検索結果は接尾辞配列中の区間 $[l, r]$ となる．このとき接尾辞 $T[j..n]$ ($j = SA[i]$, $i \in [l, r]$) はパタン P を接頭辞として持つ．つまり $P = T[j..j + |P| - 1]$ が成り立つ．i をこの接尾辞の辞書順と呼ぶ．P の出現個数は $r - l + 1$ で計算できる．

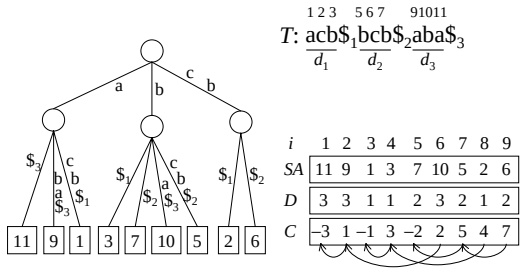

図 1　“acb$\$_1bcb\$_2aba\$_3$” に対する接尾辞木と文書列挙問い合わせのためのデータ構造.

接尾辞配列は長さ n の整数の配列であるため，そのサイズは $n < 2^{32}$ の場合 $4n$ バイトとなる．接尾辞木のサイズは実装にも依存するが最低でも $10n$ バイトである [4]．n の大きさを制限しない場合には接尾辞配列のサイズは $n \log n$ ビット(注1)となる．

2.2 圧縮接尾辞配列

圧縮接尾辞配列 (CSA) [2], [7] は接尾辞配列を圧縮したものである．サイズは $n \log n$ ビットであったものが $O(n \log |\mathcal{A}|)$ ビット ($\mathcal{A}$ はアルファベット) となり，文字列サイズ ($n \log |\mathcal{A}|$ ビット) よりも小さくすることができる．この場合は接尾辞配列の各要素は $O(\log n)$ 時間で復元でき，パタンの出現個数は $O(|P| \log n)$ 時間で求まる．また，出現位置の列挙は $O(occ \cdot \log n)$ 時間となる．

圧縮接尾辞配列では配列 SA の替わりに $\Psi[i] = SA^{-1}[SA[i] + 1]$ で定義される関数 Ψ を格納する．パタンの出現個数および辞書順は Ψ を用いて計算できる．Ψ は $O(n \log |\mathcal{A}|)$ ビットで表現できる．実際には $2.3n + 64n/L$ ビット (L はパラメタ) 程度になる．$SA[i]$ の値を計算するには別のデータ構造が必要である．そのサイズは $O(n)$ ビットである．実際には $32n/D$ ビット (D はパラメタ) である．

2.3 区間最小値問い合わせ

区間最小値問い合わせ (RMQ) とは，次のような問題である．［問題 2］（区間最小値問い合わせ） 配列 $A[1, n]$ が与えられたとき (前処理可)，区間最小値問い合わせ $RMQ_A(i, j)$ とは部分配列 $A[i, j]$ の中の最小値の添え字を求めることである．

既存手法では $O(n)$ 前処理時間，$O(1)$ 問い合わせ時間，$O(n \log n)$ ビット空間のもの [1]，$O(n)$ 前処理時間，$O(1)$ 問い合わせ時間，$O(n)$ ビット空間のもの [8] が存在する．

(注1)：log の底は 2 とする．

3. 文書列挙問題のためのデータ構造

3.1 Muthukrishnan のデータ構造

文書列挙問い合わせ list(p) に対する Muthukrishnan のアルゴリズム [6] は $O(|p|+q)$ 時間で動作し，これは最適である．データ構造としては，接尾辞木，配列 C,D および C での RMQ を求めるデータ構造を用いる．配列 D の要素 $D[i]=j$ は接尾辞木で左から i 番目の葉の表す文字列 (接尾辞と呼ぶ) が文書 d_j に含まれることを示す．つまり，あるパタンに対応する D の範囲が求まった場合，その中の要素を重複せずに出力すれば問題が解ける．

アルゴリズムは以下のようになる．ここで，配列 C の要素 $C[i]$ は $j<i$ かつ $D[j]=D[i]$ となる j のうち最大のものと定義する．そのような j が存在しない場合は $C[i]=-D[i]$ と定義する．

（1） p に対応する接尾辞配列の範囲 $[l,r]$ を計算する．

（2） $C[l,r]$ 中の最小値 $C[x]$ の位置 x を RMQ のデータ構造を用いて計算する．

（3） もし $C[x] \geqq l$ ならば終了

（4） $D[x]$ を出力し，$C[l,x-1]$ と $C[x+1,r]$ に対して Step 2 以下を再帰的に実行する．

このアルゴリズムのポイントは，Step 1 において接尾辞配列中でパタン p に対応する範囲を求めているが，それに対応する D の値を全て列挙して重複を削除するのではなく，配列 C に対する RMQ のデータ構造を用いて出力文書数 q に比例する個数のみ用いるという点である．配列 C は各文書 d_i に対してその中の接尾辞の接尾辞配列中での出現位置のリストを格納している．C に対する RMQ を求めることで各リスト中の先頭の要素のみを求めることができ，p の出現する文書を重複なしに列挙することができる．

問題点はデータ構造のサイズが大きいという点である．文書集合に対する接尾辞木のサイズは $10n$ バイト以上である．また，RMQ のデータ構造も $10n$ バイト以上必要であるため，合計で $20n$ バイト以上になってしまい，現実的ではない．

3.2 Sadakane のデータ構造

Muthukrishnan のデータ構造では配列 D が必要であり，そのサイズは $n\log_2 k$ ビット (1 から k までの数が n 個) である．つまり，文書データベース中の文書数が増えるに従って D の各要素を格納するために必要なビット数も増大するため好ましくない．

Sadakane のデータ構造 [8] では，配列 $D[i]$ の値は $SA[i]$ から計算できるため D は格納しない．また，$SA[i]$ は圧縮接尾辞配列を用いて $O(\log^{\epsilon} n)$ 時間で計算できる．接尾辞配列の範囲 $[l,r]$ は圧縮接尾辞配列を用いて $O(|p|)$ 時間で求まる [9]．アルゴリズムは以下のようになる．

（1） p に対応する接尾辞配列の範囲 $[l,r]$ を計算する．

（2） $C[l,r]$ 中の最小値 $C[x]$ の位置 x を RMQ のデータ構造を用いて計算する．

（3） $SA[x]$ から $D[x]$ を計算する．

（4） もし $D[x]$ と等しい値がすでに出力されていれば終了．

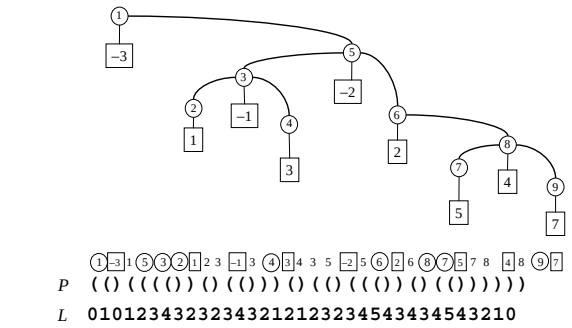

図 2 配列 C の表現方法.

（5） $D[x]$ を出力し，$C[l,x-1]$ と $C[x+1,r]$ に対して Step 2 以下を再帰的に実行する．

配列 C に関しては，値自体は格納せず，値の大小関係を表す木を格納する (図 2)．配列 $C[1,n]$ の最小値が $C[x]$ のとき，木の根に $C[x]$ を格納し，左の部分木に $C[1,x-1]$，右の部分木に $C[x+1,n]$ を再帰的に格納する．このとき，ノードの通りがけ順と接尾辞配列での順序が一致するように，$C[x]$ を左右の子の間に置く．配列の部分区間 $C[l,r]$ での最小値は，この木のノード間の最近共通祖先 (lowest common ancestor) を用いるデータ構造 [9] を用いて定数時間で求まる．データ構造のサイズは $4n+o(n)$ ビットである．

4. RMQ のためのデータ構造

本節では区間最小値問い合わせのための $O(n)$ 前処理時間，$O(\log n)$ 問い合わせ時間，$O(n)$ ビット空間の簡素なデータ構造を提案する．

4.1 Sparse Table 法

まず，提案手法のサブルーチンとして用いる Sparse Table 法 [1] について述べる．配列 $A[1,n]$ の区間 $A[i,i+l-1]$ $(l=1,2,4,\ldots,\lfloor\log_2 n\rfloor)$ に対し，その中の最小値を格納する配列 $M[i,l]$ を作成する．すると $A[i,j]$ の最小値は $M[i,m]$ と $M[j-2^m+1,m]$ $(m=\lfloor\log_2(j-i)\rfloor)$ のうちの小さい方である．最小値の格納されている位置も同様に求まる．データ構造のサイズは $O(n\cdot\log n\cdot\log n)=O(n\log^2 n)$ ビットである．問い合わせ時間は $O(1)$ 時間である．

4.2 提案手法

本論文で提案するデータ構造は次のような性質を持つ．

［補題 1］ 長さ n の配列に対する RMQ のためのデータ構造のサイズを $s(n)$，問い合わせ時間を $t(n)$ とすると，以下の式が成り立つ．

$$t(n) = O(1) + t\left(\frac{4n}{\log n}\right)$$

$$s(n) = 4n + s\left(\frac{4n}{\log n}\right) + o(n)$$

証明: 配列を C とする．図 2 のように C の要素を木の葉に格納する．しかし実際の値は格納せず，木の形のみが括弧列 P で表現されている．括弧列の中では () というパタンが C の要素と 1 対 1 に対応しており，$C[i]$ に対応する括弧は P の中で

左から i 番目の () である．また，木の各ノードは P 中では (...) で表されており，ノードの深さは閉じる括弧に対応する L の値と等しい．C の 2 つの要素の大小関係と，それらに対応する木のノードの深さの大小関係は等しいため，$RMQ_C(i,j)$ は $RMQ_L(i',j')$ に変換できる．L の長さは $4n$ である．L は木のノードを深さ優先でたどり，各ノードの深さを順に配列に格納することで求まる．L は長さ $4n$ の括弧列 P で表現でき，i',j' は $o(n)$ ビットのデータ構造を用いて定数時間で求まる [8].

L での RMQ を行うために，L を長さ $w = \log n$ のブロックに分割し，各ブロックでの最小値を格納する配列 B を新たに作る．B の要素数は $4n/w$ である．すると $RMQ_L(i',j\prime)$ は $l = RMQ_L(i',i''-1), m = RMQ_L(i'',j''-1), r = RMQ_L(j'',j')$ の中の最小値になる (i'' は i' より大きい w の倍数で最小のもの，j'' は j' より小さい w の倍数で最大のもの)．l と r は w ビットの全ての (,) のパタンに対して予め最小値を計算して表に格納しておくことでおくことで定数時間で求まる．表のサイズは $2^w \cdot w^2$ ビット，つまり約 2M バイトである．m は $RMQ_B(i''/w, i''/w-1)$ に等しいため $t(4n/\log n)$ 時間で求まる． □

このデータ構造を再帰的に用いるとデータ構造のサイズは $s(n) = 4n + O(n/\log n) + O(n/\log^2 n) + \cdots$ となる．3 回の再帰で配列 B の要素数が $O(n/\log^3 n)$ になる．これに対し Sparse Table 法を用いると，サイズが $O(n/\log n)$ ビットになるため，全体で $4n + o(n)$ ビットになる．問い合わせ時間は $O(1)$ である．なお，w ビットの全ての (,) のパタンに対する表は再帰の全てのレベルで共有できる．

4.3 実装について

このアルゴリズムで実装が難しい点は，C の添え字 i,j と L の添え字 i',j' の変換である．i' は P の中で左から i 番目の () の位置であり，i は $P[1,i]$ の中の () の個数である．これを $i' = select(i)$，$i = rank(i')$ と表す．$rank$ と $select$ は $o(n)$ ビットのデータ構造を用いて定数時間で求まることが知られているが，その実装は複雑であり，またデータ構造のサイズも $o(n)$ ではあるが小さい n に関しては無視できない．よって，本論文では簡素で実際のサイズが小さいデータ構造を提案する．

$rank$ を求めるためのデータ構造は以下の通りである．$w = 16$ とする．L をサイズ w^4 の大ブロックに分割し，配列 $R_1[i]$ に左から 1 番目から $i-1$ 番目の大ブロック中の $rank$ の最大値を格納する．次に各大ブロックをサイズ w^2 の中ブロックに分割し，配列 $R_2[i+w^2+j]$ に i 番目の大ブロック中の左から 1 番目から $j-1$ 番目の中ブロック中の $rank$ の最大値を格納する．次に各中ブロックをサイズ $w * w'$ の小ブロックに分割し，配列 R_3 に $rank$ の値を同様に格納する．配列 R_1 のサイズは，32 ビットの数を格納するため $32n/w^4 = n/2048$ ビットである．配列 R_2 のサイズは，$\log w^4 = 16$ ビットの数を格納するため $16n/w^2 = n/16$ ビットである．配列 R_3 のサイズは，$\log w^2 = 8$ ビットの数を格納するため $8n/(ww') = n/(2w')$ ビットである．計算時間は $O(w')$ である．

$select(i)$ を求めるには，単純には $rank$ を用いて二分探索を行えばいい．しかしこの場合は計算時間は $O(w' \log n)$ となる．これを $O(w' + \log n)$ 時間にするアルゴリズムを提案する．まず，$select(i)$ を含む大ブロックを求める．これは R_1 を用いた二分探索で $O(\log n)$ 時間で求まる．次に $select(i)$ を含む中ブロック，小ブロックを同様に求める．最後に小ブロックの中を逐次検索で求める．計算時間は $O(\log(n/w^4) + \log w^2 + \log(ww') + w') = O(w' + \log n)$ となる．$w' \leqq \log n$ であるため計算時間は $O(\log n)$ となる．

また，L もそのまま格納するのではなく，括弧列 P で表されている．よって $L[i]$ の値を求めるためにも $rank$ を求めるときに用いた方法とほぼ同じデータ構造を用いる．つまり，L を大中小のブロックに分割して各ブロックの左端の L の値を配列に格納しておく．計算時間は $O(w')$ である．

このようなデータ構造を用いた場合，RMQ の時間は $s(n) = O(\log n) + s(4n/\log n) = O(\log n)$ となる．

5. 文書列挙問題のアルゴリズムの詳細

5.1 概　　要

Sadakane [8] のデータ構造で実装が難しいのは以下の部分である．まず，$O(|p|)$ 時間の問い合わせをサポートする圧縮接尾辞配列の実装は複雑である．また，RMQ のデータ構造では $select$ を定数時間で行う必要があるが，この実装も複雑であった [5].

本論文ではこれらを以下のように変更する．圧縮接尾辞配列を用いたパタンの検索は単純な二分探索を用いる．検索時間は $O(|p|\log n)$ となる．次に，select 操作を定数時間で行うデータ構造は実装せずに，select の反対の操作である rank (集合の中で i よりも小さい値の数を返す操作) を用いた二分探索を行う．

5.2 詳　　細

第 3.2 節のアルゴリズムの各 Step は以下のようになる．

Step 1: 圧縮接尾辞配列を用いると $O(|p|\log n)$ 時間で $[l,r]$ が求まる．

Step 2: 第 4. 節のアルゴリズムを用いると，問い合わせ時間は $O(\log n)$ である．

Step 3: 文字列 T は文書 $d_1, d_2, \ldots, d_k$ を連結したものである．各文書の最後の文字 \$ の T 中での位置を配列 $E[1,k]$ に格納しておく．すると $D[x]$ は $SA[x] \leqq E[i]$ を満たす最大の i となるため，配列 E の上で二分探索を行うことにより $O(\log k)$ 時間で求まる．$SA[x]$ の値は圧縮接尾辞配列を用いると $O(\log n)$ 時間で求まる．

Step 4: $D[x]$ の重複判定は，長さ k の 0,1 のベクトル V を用意し，出力した文書番号に対応するビットを 1 にしていくことで行う．また，長さ k の別の配列 E を用意し，出力した値を格納しておく．全ての値を出力したあとに，E に格納してある値を順に取り出し，それに対応する V の値を 0 に戻す．総計算時間は $O(q)$ である．

なお，Step 3 は以下のようなアルゴリズムにすることもできる．

Step 3': $D[i]$ の値を i がパラメタ m の倍数の場合のみ別の配列 D_2 に格納しておく (つまり $D_2[i] = D[mi]$)．$m = O(\log k)$

表 1　Bender-Farach のデータ構造のサイズと問い合わせ時間

n	size (bytes)	time (s)
10^3	27686	0.361
10^4	317404	0.661
10^5	3474880	1.049
10^6	37749856	1.047
10^7	417499824	1.056

となるようにする．また，各文書の最後の文字 \$ の辞書順が $[t_l, t_r]$ の範囲であるとすると，$i \in [t_l, t_r]$ の場合に $D[i]$ を配列 D_3 に格納する．このとき，$D[i]$ を求めるには，i が m の倍数でなく，かつ，$i \notin [t_l, t_r]$ の間 $i := \Psi[i]$ を計算する．そして配列 D_2 または D_3 に格納してある値を取り出すと $D[i]$ が求まる．計算時間の期待値は $O(m) = O(\log k)$ であり，データ構造のサイズは $n \log k/m + k \log k = O(n)$ ビットである．

5.3　データ構造のサイズ

文書列挙問い合わせのためのデータ構造としては圧縮接尾辞配列と RMQ のためのデータ構造が必要である．圧縮接尾辞配列のサイズは圧縮された文書のサイズにほぼ等しい [7]．RMQ のためのデータ構造のサイズは括弧列 P のための $4n$ ビットの他は $O(n/\log n)$ ビットであるため無視できる．その他の配列は要素数が $O(k)$ であるため無視できる．

6.　実　　験

データ構造のサイズと検索速度に関する実験を行った．実験で用いたマシンは CPU Pentium III 1.4GHz，メモリ 4GB である．OS は Solaris 8 で，コンパイラは gcc 2.95.3 で最適化は -O3 である．

6.1　RMQ のデータ構造

RMQ のデータ構造として，本論文のものと，Bender と Farach のデータ構造 [1] を比較した．用いた入力データは長さが 10^7 万のランダムな整数の値の配列である．そしてデータ構造のサイズと 100 万回のランダムな問い合わせに対する時間を測定した．

本論文の RMQ のデータ構造には 2 つのパラメタ d, w' がある．d は再帰の深さで，$d = 0$ のときは Sparse Table 法と等しく，d が増えるにしたがってデータ構造のサイズは小さくなる．w' は $rank(i)$ や $L[i]$ を計算するための表のサイズに関するパラメタで，w' が大きくなるほどサイズが小さくなり，速度が遅くなる．

Bender と Farach のデータ構造では，配列 L の長さは $2n$ であり，圧縮せずに用いている．また，*rank* や *select* がビットベクトルでは表せないためそれぞれ長さ $2n$ の整数の配列で表現している．表 1 はこのデータ構造のサイズと 100 万回のランダムな問い合わせの合計時間を示している．入力配列のサイズは 10^3 から 10^7 である．データ構造のサイズは約 $2n \log_2 n$ バイトで，1 回の問い合わせ時間は 1μ 秒である．

次に，本論文で提案する RMQ のデータ構造について実験を行った．データ構造には 2 つのパラメタ d, w' がある．これらを $d = 1, 2, \ldots, 7$, $w' = 2, 4, 8, 16$ と変化させ，データ構造のサイズと問い合わせ時間を測定した．n は 10^7 万である．

表 2　RMQ のデータ構造のサイズと問い合わせ時間

	$w' = 2$		$w' = 4$	
d	size (bytes)	time (s)	size (bytes)	time (s)
1	448129894	2.598	446879894	2.857
2	110162372	3.851	108599872	4.298
3	33170500	5.048	31529874	5.486
4	13297570	5.880	11637412	6.613
5	11376200	6.587	9711160	7.432
6	10954466	7.048	9288206	7.769
7	10863688	7.276	9197122	7.931

表 3　RMQ のデータ構造のサイズと問い合わせ時間

	$w' = 8$		$w' = 16$	
d	size (bytes)	time (s)	size (bytes)	time (s)
1	446254894	3.209	445629892	3.702
2	107818622	4.904	107037368	5.668
3	30709562	6.380	29889244	7.417
4	10807334	7.574	9977250	8.920
5	8878640	8.537	8046114	9.965
6	8455076	9.003	7621938	10.486
7	8363840	9.101	7530548	10.461

パラメタ d, w' とデータ構造のサイズ，問い合わせ時間の関係を図 3 に示す．横軸はデータ構造のサイズであり，縦軸は 100 万回の問い合わせ時間の合計である．各パラメタでのサイズと時間をグラフの点で表す．w' を一定にし，d を 4 から 7 まで変化させ対応する点を線で結んでいる．あるパラメタでの点に対し，それの右上の領域に含まれるパラメタは用いる意味がない．つまり，$(d, w') = (4, 4), (4, 8), (4, 16)$ は用いる意味がないことがわかる．その他のパラメタはサイズと時間のトレードオフがある．$(d, w') = (7, 16)$ のときにデータ構造のサイズが最も小さくなる．このときのサイズは $n = 10^7$ のときに Bender-Farach のものの約 1/55 であり，問い合わせ時間は 9.90 倍である．サイズは約 $6.02n$ ビットであり，$4n + o(n)$ で表せるはずのものが $n < 2^{32}$ 程度ではサイズが理論値の 1.5 倍になっていることがわかる．$(d, w') = (4, 2)$ のときはサイズが Bender-Farach の約 1/31 であり，時間は 5.57 倍である．

次に，RMQ をデータ構造を用いずに行ったときの時間を測定した．入力は長さ 200 万の整数の配列で，問い合わせは長さ 10^2 から 10^6 のランダムな区間とした．表 4 は naive 法 (区間内の全要素を比較して最小値を求める)，Bender-Farach のデータ構造，本論文のデータ構造 $(d = 7, w' = 16)$ での 10000 回のランダムな問い合わせの合計時間を表している．naive 法は問い合わせ時間が区間の長さに比例している．Bender-Farach は naive 法よりも常に高速である．本論文の手法でも区間長が 1000 以上では naive 法よりも高速である．よって本論文の手法は実用的であるといえる．

6.2　文書列挙問い合わせのデータ構造

Web ページの集合に対して文書列挙問い合わせのためのデータ構造を作成した．用いたデータは `http://www.linux.or.jp`

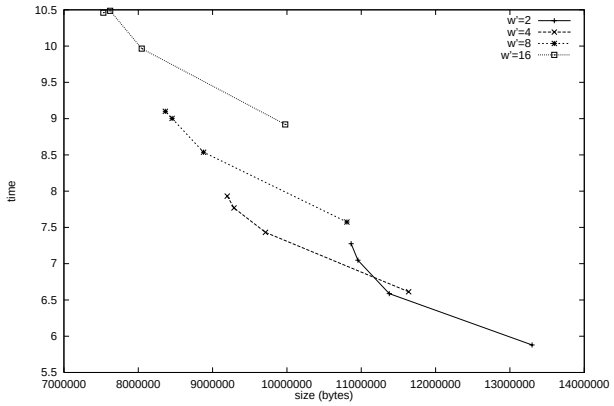

図 3　パラメタとサイズ，問い合わせ時間の関係

表 4　問い合わせ区間の長さと問い合わせ時間

	10000 回の問い合わせ時間 (s)		
区間長	naive	本論文	Bender-Farach
10^2	0.006	0.037	0.001
10^3	0.041	0.039	0.001
10^4	0.399	0.067	0.001
10^5	4.984	0.151	0.001
10^6	39.870	0.156	0.001

以下のページである．ページ数は 13421，文書サイズの合計 (n) は約 77M バイトであるが，Muthukrishnan のデータ構造はサイズが大きいため，このうちの 50M バイトを用いた．ページ数は 5984 である．これらのページを 1 つの文字列に連結し，それに対する圧縮接尾辞配列を作成した．Ψ のサイズは約 15 メガバイトであり，1 文字あたり 2.371 ビットである．なお，圧縮接尾辞配列から元の文書を復元することが可能であるため，文書を別に格納する必要はない．

文書列挙問い合わせのデータ構造としては次の 3 種類を用いた．

（1） (Muthukrishnan のデータ構造) 配列 C, D を圧縮せずに格納．RMQ のデータ構造は Bender-Farach のものを使用．接尾辞木の替わりに圧縮接尾辞配列を使用．

（2） (naive) 配列 C や RMQ のデータ構造は用いない．配列 D を圧縮せずに格納．文書列挙問い合わせはパタンの全ての出現個所に対し D の値を配列から読み，重複を除去する．

（3） (本論文のデータ構造) 配列 C, D は格納しない．D の値は圧縮接尾辞配列から求める．RMQ のデータ構造は本論文のものを用いる ($d = 7, w' = 16$).

D の要素は文書番号であり，文書数は 13421 で 16 ビットに収まるため圧縮しない D のサイズは $2n$ バイトである．

まず，これらのデータ構造に対し，サイズを測定した (表 5)．なお，圧縮接尾辞配列はどのデータ構造でも用いているためそのサイズは含んでいない．表の bpc とは bits per character の意味で，文字 1 文字あたりのデータ構造のサイズを表す．アルファベットサイズは 256 であるため文字は 8 bpc である．本論文のデータ構造は Muthukrishnan のものと比べてサイズが約 0.11 倍，配列 D を格納する手法 (naive) の約 0.38 倍である．なお，Muthukrishnan や naive では配列 D のサイズは $n \log k$ ビット (k は総文書数) であり，データベースのサイズが大きくなるに従い D のサイズも大きくなる．一方，本論文のデータ構造はサイズが $O(n)$ ビットであり，k には依存しない．

表 5　文書列挙問い合わせのデータ構造のサイズ

データ構造	size (bytes)	size (bpc)
Muthukrishnan	354139810	54.03
naive	104857602	16.00
本論文	39567008	6.03

表 6 に各手法での文書列挙問い合わせの時間を示す．時間は同じパタンに対する問い合わせを 100 回行った場合のものである．圧縮接尾辞配列のパラメタは $D = 8, L = 32$ であり，そのサイズは約 52M バイト (8.371 bpc) である．(出現頻度)/(文書数)，つまり 1 つの文書あたりのパタンの出現頻度が小さいほど naive 法は高速である．実験ではどのパタンに対してもこの比が 300 以内であり，この場合は naive 法が RMQ を用いるよりも高速であった．しかし RMQ を用いる Muthukrishnan のデータ構造の方が高速な場合もあった．

本論文のデータ構造と Muthukrishnan のデータ構造では後者の方が約 4 倍高速である．しかしデータ構造のサイズは前者が $6.03 + 8.37 = 14.4$ bpc，後者が接尾辞配列 (32 bpc) を用いたとして $54.03 + 32 = 86.03$ bpc となり，本論文の方のサイズは約 1/6 である．

表 6　文書列挙問い合わせの時間

			問い合わせ時間 (100 回) (s)		
パタン	出現頻度	文書数	Muthu	naive	本論文
<	1756521	5942	1.412	2.520	5.772
>	1759674	5942	4.056	2.544	12.472
e	921696	5943	3.901	1.341	13.374
a	536034	5943	4.300	0.793	15.195
i	626679	5943	4.099	0.923	14.231
p	340467	5910	3.665	0.512	12.656
html	102310	5940	1.531	0.086	6.638
7	40384	4401	1.864	0.042	8.471
linux	30180	3834	1.738	0.031	6.074
the	10543	3019	0.913	0.016	4.529

本論文の手法での問い合わせ時間は CSA のパラメタに大きく依存する．表 7 に CSA のパラメタとサイズ，問い合わせ時間 (100 回の合計) の関係を示す．第 2,3,4 列は各パラメタの CSA を用いたときの文書列挙問い合わせの時間 (100 回の合計) である．下から 2 番目の行は CSA のサイズ (bpc) である．CSA のサイズを 8.371 bpc から 3.871 bpc に圧縮すると，問い合わせの時間が約 6 倍になる．一番下の行は CSA と RMQ のデータ構造のサイズの合計である．

次に，本論文のアルゴリズムの Step 3 を Step 3' と置き換えた場合のデータ構造のサイズ (bpc) と問い合わせ時間 (100 回) を調べた．データ構造にはパラメタ m, L があり，サイズは $6.03 + 2.371 + 64/L + 16/m$ bpc である．パラメタが $(m, L) = (4, 32), (4, 128)$ のときは CSA をそのまま用いるよ

表 7　CSA のサイズと文書列挙問い合わせの時間

	CSA のパラメタ (D, L)		
パタン	$(32, 32)$	$(32, 128)$	$(8, 32)$
<	22.179	29.888	5.772
>	44.059	68.093	12.472
e	46.254	74.474	13.374
a	48.650	79.628	15.195
i	47.792	77.531	14.231
p	43.734	69.454	12.656
html	21.672	28.081	6.638
7	28.274	45.592	8.471
linux	21.150	31.045	6.074
the	15.631	24.122	4.529
CSA のサイズ	5.371	3.871	8.371
全サイズ	11.401	9.901	14.401

りもサイズ，速度共に優れている．

提案手法を従来手法と比較するには，RMQ のデータ構造または配列 D のサイズの他に，パタンに対応する接尾辞配列の区間 $[l, r]$ を求めるためのデータ構造が必要がある．これは圧縮接尾辞配列の Ψ 関数を用いる．その他に $64/L$ ビットが必要である．$L = 128$ とすると $2.371 + 64/128 = 2.871$ bpc である．提案手法で $(m, L) = (4, 128)$ のものを Muthukrishnan の手法と比較すると問い合わせ時間は約 2.5 倍で，サイズは 0.23 倍である．naive 法と比較すると，問い合わせ時間はパタンの出現頻度が多い場合で 1.4 倍，サイズは 0.68 倍である．

なおこのデータ構造を用いた場合はパタンの出現する文書を求めることはできるが出現位置は求まらない．

表 8　文書列挙問い合わせの時間

	パラメタ (m, L)					
パタン	$(8, 16)$	$(8, 32)$	$(4, 16)$	$(4, 32)$	$(4, 128)$	$(16, 32)$
<	5.258	5.788	2.896	3.052	3.520	10.070
>	11.297	12.473	7.203	7.587	9.397	22.696
e	11.884	13.412	7.359	7.875	10.932	24.171
a	13.391	15.221	8.284	8.984	12.813	26.118
i	12.548	14.197	7.801	8.420	11.881	25.159
p	11.258	12.698	7.074	7.602	10.350	23.312
html	6.021	6.594	3.622	3.804	4.299	11.429
7	7.402	8.519	4.210	4.693	7.150	15.043
linux	5.539	6.077	3.320	3.414	3.930	11.352
the	4.007	4.524	1.944	2.083	2.777	8.325
全サイズ	14.401	12.401	16.401	14.401	12.901	11.401

最後に，代表的なデータ構造 4 つを用いて，文書列挙問い合わせでの出力文書数が小さい場合の問い合わせ時間を測定した．表 9 で，N は配列 D を圧縮せずに格納するもの (naive)，O は本論文のデータ構造 $(d = 7, w' = 16, D = 32, L = 128)$，NC は naive 法だが配列 D を Step 3' の手法で圧縮したもの $(m = 4, L = 128)$，OC は本論文のデータ構造で Step 3' の手法で D を圧縮したもの $(m = 4, L = 128)$ を表す．NC はパタンの出現頻度が少ないときは OC より高速であるが，(出現頻度)/(文書数) が 3 を越えると OC の方が高速になる．また，OC の速度は実用に耐えうると思われる．

表 9　文書列挙問い合わせの時間 (1000 回)

			データ構造			
パタン	出現頻度	文書数	N	O	NC	OC
RPM	1304	241	0.015	30.568	7.002	3.117
Debian	1501	329	0.019	46.811	7.288	4.601
RedHat	829	291	0.015	37.361	3.639	3.197
Apache	321	92	0.004	14.328	1.362	0.858
Tokyo	99	27	0.002	2.932	0.589	0.319
tohoku	25	21	0.001	1.740	0.122	0.143
algorithm	25	12	0.000	1.268	0.101	0.095
	全サイズ (bpc)		18.871	9.901	6.871	12.901

6.3 考　　察

区間最小値問い合わせのアルゴリズムとしては，本論文で提案する手法がデータ構造のサイズが小さく，問い合わせも高速であることがわかる．

文書列挙問い合わせの時間は，実験に用いたデータではほとんどのパタンに対してその全ての出現位置に対応する D の値を列挙し，重複を除去するという単純な手法が最も高速であった．しかし，配列 D を圧縮せずに保存するとそのサイズが非常に大きくなる．文書数を k，全文書の合計の長さを n とすると D のサイズは $n \log k$ ビットとなり，大量の文書を格納するデータベースではこの方法はサイズが大きすぎる．

配列 D はそれ自体では圧縮することは難しい．しかし，$D[i]$ を圧縮するかわりに $SA[i]$ の値から計算することができる．ただし $SA[i]$ の計算に圧縮接尾辞配列を用いる場合，そのアクセス速度が遅いために全ての $D[i]$ を列挙することができない．よって RMQ のデータ構造が必須となる．また，D は圧縮接尾辞配列の Ψ 関数を用いると圧縮することができる．D の値を $m = O(\log k)$ 個おきにそのまま格納し，それ以外の値は Ψ から計算することができるため格納しない．総文書数 k は文書サイズの合計 n よりも必ず小さいため，SA を復元するためのデータ構造よりも D を復元するためのデータ構造は小さくすることができる．また，$D[i]$ の復元は $O(\log k)$ 時間であり，$O(\log n)$ 時間かかる $SA[i]$ の復元よりも高速であるため文書列挙問い合わせも高速になる．ただしこのデータ構造では $SA[i]$ の値，つまりパタンの出現位置を求めることはできない (データ構造のサイズを $32n/D$ ビット増やせば可能である)．

いずれのアルゴリズムを用いる場合でもパタンの辞書順を求める必要がある．これは接尾辞木，接尾辞配列，圧縮接尾辞配列を用いて計算することができる．特に，圧縮接尾辞配列はパタンの出現位置でなく辞書順を求めるのみの場合にはサイズを非常に小さくすることができる．

以上まとめると，パタンの出現位置を求める必要がない場合には表 9 の OC のデータ構造，パタンの出現位置まで求める必要がある場合には OC に加えて表 8 の $(D, L) = (32, 128)$ のデータ構造を用いればよい．データ構造のサイズは前者が $6.03 + 2.371 + 64/L + 16/m = 12.901$ bpc，後者が $6.03 + 2.371 + 64/L + 16/m + 32/D = 13.901$ bpc となる．これは文書サイズ (8bpc) の 1.74 倍であるため，実用的であると思われる．

7. ま と め

本論文ではテキストデータベースでは必須である，文書列挙問い合わせに対する簡素なデータ構造を提案した．Muthukrishnan や naive なデータ構造のサイズは $O(n \log n)$ ビットであったが，本論文のものは $O(n)$ ビットであるため文書数の多いデータベースに対しても適用できる．実際には約 $13n$ ビットにできる．

なお，本論文では実装していないが，単語の *tf*idf* スコアも同様の手法で計算することができる [8]．データ構造のサイズは約 2 倍となる．また，文書列挙問い合わせの時間は圧縮接尾辞配列のアクセス時間に大きく依存するため，改良が必要である．これらのデータ構造の効率の良い実装については今後の課題とする．

文　献

[1] M. Bender and M. Farach-Colton. The LCA Problem Revisited. In *Proceedings of LATIN2000*, LNCS 1776, pages 88–94, 2000.

[2] R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. In *32nd ACM Symposium on Theory of Computing*, pages 397–406, 2000.

[3] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[4] S. Kurtz. Reducing the Space Requirement of Suffix Trees. *Software – Practice and Experience*, 29(13):1149–1171, 1999.

[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] S. Muthukrishnan. Efficient Algorithms for Document Retrieval Problems. In *Proc. ACM-SIAM SODA*, pages 657–666, 2002.

[7] K. Sadakane. Compressed Text Databases with Efficient Query Algorithms based on the Compressed Suffix Array. In *Proceedings of ISAAC'00*, number LNCS 1969, pages 410–421, 2000.

[8] K. Sadakane. Space-Efficient Data Structures for Flexible Text Retrieval Systems. In *Proc. ISAAC 2002*, pages 14–24. LNCS 2518, 2002.

[9] K. Sadakane. Succinct Representations of *lcp* Information and Improvements in the Compressed Suffix Arrays. In *Proc. ACM-SIAM SODA 2002*, pages 225–232, 2002.

[10] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.