文書列挙問題に対する実用的なデータ構造

Practical Data Structures for the Document Listing Problem

定兼 邦彦♦　渡邉 大輔♠

Kunihiko SADAKANE　Daisuke WATANABE

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．

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 that can solve the problem for any pattern has been proposed, the size of the data structure is large. Sadakane has proposed a data structure that 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 show that it is of small size and supports efficient queries.

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[1])
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$\{j \mid d_j[i..i+m-1]=p \text{ for some } i\}$ 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 [2]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 [1]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 [3]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 [4]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 $O(|p|+q \log^{\varepsilon} n)$ 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 $n + 10n / \log\log n$ ビットであるとすると，$n = 2^{32}$ のときにこれは $n + 10n / \log\log n = 3n$ 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

♦ 正会員　九州大学大学院システム情報科学研究院
sada@csce.kyushu-u.ac.jp
♠ 富士通東日本コミュニケーションシステムズ㈱

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．

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．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[1]となる．

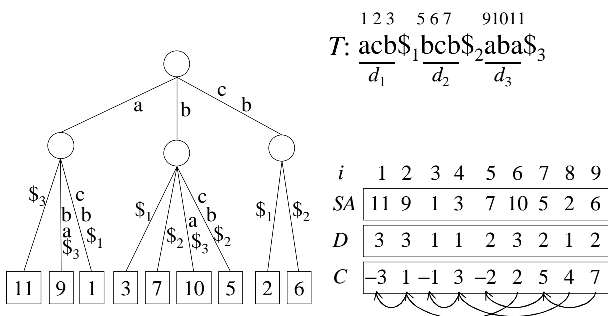

図 1 "acb$_1$bcb$_2$aba$_3$" に対する接尾辞木と文書列挙問い合わせのためのデータ構造.

2.2 圧縮接尾辞配列

圧縮接尾辞配列(CSA) [4,5]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 $O(n \log|A|)$ ビット(A はアルファベット) となり，文字列サイズ ($n \log|A|$ 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 $O(occ \log n)$ 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 $O(n \log|A|)$ 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 [6]，$O(n)$ 前処理時間，$O(1)$ 問い合わせ時間，$O(n)$ ビット空間のもの[3]が存在する．

2.4 文書列挙問い合わせ

文書列挙問い合わせのための Muthukrishnan のアルゴリズム[1]では，図 1 の配列 C, D を用いる．$D[i]$は接尾辞 $SA[i]$ が属する文書の番号を表す．$C[i]$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．アルゴリズムの詳細は第 4 節に記述する．

3. 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．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 [3]．

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 $RMQ_L(i',j')$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 $2^w w^2$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．　□

[1] log の底は 2 とする．

P (()(((())()(())))()(()((())()(()))))))
L 010123432323432121232345434345432 10

図 2 配列 C の表現方法.

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 [6]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.再帰の深さは 7 とする.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. 文書列挙問題のためのデータ構造

本論文では Sadakane のアルゴリズム[3]を用いる.

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.
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.

Sadakane [3]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 [7].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.

上記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 3 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: $D[i]$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 $D_2[i]=D[mi]$).$m=O(\log k)$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.

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 D'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,D'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.

5. 実験

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/ 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.

5.1 データ構造

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 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未圧縮の 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 1).これは圧縮接尾辞配列のサイズも含んでいる.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 8bpc 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 0.20 倍,配列 D を格納する手法(naive)の約 0.68 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.Muthukrishnan のデータ構造はサイズが大きく,検索時間も naive 法よりも遅いため以下の実験では用いない.

表 1 文書列挙問い合わせのデータ構造のサイズ

データ構造	サイズ(bpc)
Muthukrishnan	62.401
naive	18.871
本論文	12.901

次に,代表的なデータ構造 3 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 2 で，N は配列 D を圧縮せずに格納するもの(naive)，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(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 2 文書列挙問い合わせの時間(1000 回)

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

5.2 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 2 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 に加えて圧縮接尾辞配列の(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) の高々1.74 倍であるため，実用的であると思われる．

6. 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 [3]．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．

[謝辞] この研究の一部は文部科学省科学研究費の援助を受けた．

[文献]

[1] Muthukrishnan, S: “Efficient Algorithms for Document Retrieval Problems,” in Proc. SODA, pp.657-666 (2002).
[2] Salton, G., Wong, A., and Yang, C.S.: “A Vector Space Model for Automatic Indexing,” Communications of the ACM, 18(11):613-620 (1975).
[3] Sadakane, K.: “Space-Efficient Data Structures for Flexible Text Retrieval Systems,” in Proc. ISAAC 2002, LNCS 2518, pp.14-24 (2002).
[4] Grossi, R and Vitter, J.S.: “Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching,” in ACM Symp. on Theory of Computing, pp.397-406 (2000).
[5] Sadakane, K.: “Compressed Text Databases with Efficient Query Algorithms based on the Compressed Suffix Array,” in Proc. ISAAC00, LNCS1969, pp.410-421 (2000).
[6] Bender, M. and Farach-Colton, M.: “The LCA Problem Revisited,” in Proc. LATIN, LNCS1776, pp.88-94 (2000).
[7] Munro, J.I.: “Tables,” in Proc. FSTTCS, LNCS1180, pp.37-42 (1996).

定兼 邦彦 Kunihiko SADAKANE
2000 年東京大学大学院理学系研究科情報科学専攻終了．2000 年 4 月より東北大学大学院情報科学研究科助手．2003 年 4 月より九州大学大学院システム情報科学研究院助教授．情報処理学会，日本データベース学会会員．

渡邉 大輔 Daisuke WATANABE
2003 年東北大学工学部情報工学科卒業．2003 年 4 月より富士通東日本コミュニケーションシステムズ㈱勤務．