LA-4 柔軟な文書検索のためのコンパクトなデータ構造
Space-Efficient Data Structures for Flexible Document Retrieval

定兼 邦彦*
Kunihiko Sadakane

1 はじめに

本論文は柔軟な文書検索のためのコンパクトなデータ構造を提案する．現在広く用いられている文書検索システムでは，*tf*idf* スコア [9] に従い文書をランク付けすることで検索の精度を高めている．このスコアは単語 p と文書 d に対するスコア $tf(p,d)$ と $idf(p)$ の積で定義される．前者は文書 d 中の単語 p の出現頻度，後者は総文書数を k, p を含む文書数を n_p とすると $\log \frac{k}{n_p}$ と定義される．このスコアを計算するためのデータ構造としては転置ファイル [2] が用いられている．これは各単語ごとにそれを含む文書の番号を列挙したものであり，単語の頻度も格納してあるため *tf*idf* スコアを高速に計算できる．ただし任意の文字列に対して計算することができないという欠点がある．

文字列検索の分野では接尾辞木 ([4] 参照) が有名である．これは任意の文字列の出現個所や頻度を高速に求められるデータ構造であるが，ある文字列 p を含む文書を高速に列挙すること (文書列挙問い合わせ) ができない．Muthukrishnan [6] は接尾辞木と区間最小値問合わせのデータ構造を用いた最適時間アルゴリズムを考案したが，転置ファイルと比べてデータ構造のサイズが大きいという欠点がある．また，*tf*idf* スコアの計算もできない．

本論文では文書列挙問い合わせおよび任意の文字列に対する *tf*idf* スコア計算のためのコンパクトなデータ構造を提案する．検索システム中の文書数を k, 文書の総長を n とすると，主結果は以下の通りである．

定理 1 任意の文字列 p に対し，それを含む q 個の文書を $O(|p|+q\log^{\epsilon} n)$ 時間 (ϵ は $0<\epsilon\leq 1$ の定数) で列挙できる．

定理 2 任意の文字列 p と文書 d に対し，$tf(p,d)$ を $O(|p|)$ 時間で計算できる．また，p を含む q 個の文書全てに対する $tf(p,d)$ を $O(|p|+q\log^{\epsilon} n)$ 時間で計算できる．

定理 3 任意の文字列 p に対し，$idf(p)$ を $O(|p|)$ 時間で計算できる．

これらの問い合わせを実現するデータ構造のサイズは $2|CSA|+10n+o(n)$ ビットであり，$O(n)$ 時間で構成できる．ここで，$|CSA|$ はデータベース中の全ての文書に対する圧縮接尾辞配列 [3] のサイズであり，通常は文書サイズ以下になる [7]．また，文書サイズは通常 $8n$ ビットであるため，このデータ構造のサイズは文書サイズの約 3 倍である．既存手法で用いられている接尾辞木のサイズは文書サイズの 10 倍以上であり，また，Muthukrishnan のデータ構造のサイズはその約 2 倍程度であるため，本論文のデータ構造のサイズはそれらより非常に小さい．また，計算量は高々 $O(\log^{\epsilon} n)$ 倍になるだけである．このように，本研究のデータ構造は任意の文字列に対する *tf*idf* スコアを高速に計算できるため，転置ファイルなどのデータ構造よりも柔軟な検索を行える．

*東北大学大学院情報科学研究科 sada@dais.is.tohoku.ac.jp

2 文書列挙問い合わせのためのデータ構造

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 の接尾辞木は図 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が T 中のどの位置にあるかを表す．これは接尾辞配列 ([4] 参照) と呼ばれ，SA で表す．配列 D の要素 $D[i]=j$ は接尾辞木で左から i 番目の葉が文書 d_j に含まれることを示す．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される．そのような j が存在しない場合は $C[i]=-D[i]$ と定義する．

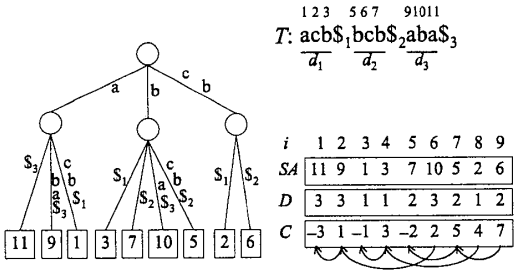

図 1: "acb$\$_1bcb\$_2aba\$_3$" に対する接尾辞木と文書列挙問い合わせのためのデータ構造．

Muthukrishnan のアルゴリズム [6] はまず p に対応する接尾辞配列の範囲 $[l,r]$ を求め，次に配列 C の $C[l,r]$ の範囲で l 未満の数 $C[i]$ に対応する $D[i]$ の値を列挙する．本論文ではこれらのデータ構造のサイズを圧縮する．まず，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 [8]．

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 [8] 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．

Muthukrishnan のアルゴリズムでは文書を重複して出力しないために C の値を用いていた．本研究では C の値自身は保存していないため，長さ k のビットベクトルを用い

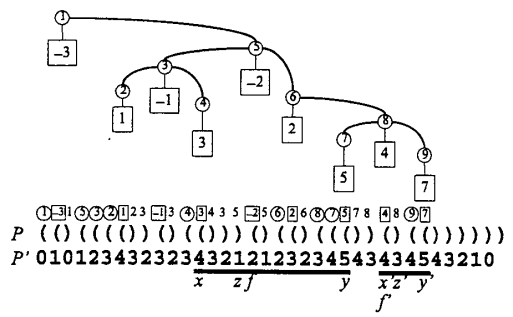

図 2: 本論文での配列 C の表現方法.

る．文書番号に対応するビットが 0 の場合のみ文書を出力し，ビットを 1 にする．全ての文書を出力した後でビットを 0 に戻す．

3 tf*idf スコアの計算法

3.1 $tf(p,d)$ の計算法

文書全体を表す文字列 T に対する圧縮接尾辞配列の他に，各文書に対する圧縮接尾辞配列も用意する．後者のサイズの合計は前者のサイズとほぼ等しいため，合計で $2|CSA|$ となる．文書 d の圧縮接尾辞配列を用いれば $tf(p,d)$ は $O(|p|)$ 時間で求まる．但し，p を含む全ての文書 (q 個) に対し $tf(p,d)$ を計算する場合，この方法では $O(|p|q)$ 時間 かかってしまう．

q 個の文書に対し $tf(p,d)$ をそれぞれ計算する場合，まず T の接尾辞配列中の p に対応する範囲 $[l,r]$ を $O(|p|)$ 時間で求める．次に，配列 $D[l,r]$ 中の異なる値について最左要素と最右要素の添え字を列挙する．最左要素を求めるには上述の配列 C での最小値を求めるデータ構造を用いる．最右要素は $C'[i]$ が $j>i$ かつ $D[j]=D[i]$ となる j のうち最小のものとして定義される配列 C' 中の区間の最大値を求めるデータ構造を用いて求める．データ構造のサイズは 2 倍になり $8n+o(n)$ ビットである．

配列 $D[l,r]$ 中で $D[i]=d$ である添え字のうち最小と最大のものを i,j とする．すると $SA[i], SA[j]$ を求めることによりそれらの T 中の位置が求まり，さらに文書 d 中の位置 x,y が求まる．ここで文書 d に対する接尾辞配列の逆関数を用いると位置 x,y の文字列 p の接尾辞配列中の位置 i',j' が $O(\log^{\epsilon} n)$ 時間で求まる [7]．すると $tf(p,d)=j'-i'+1$ で計算できる．なお，最左要素と最右要素を小さい順にソートする必要があるが，これは要素数を q とすると $O(q \log\log q)$ 時間で行える [1].

3.2 $idf(p)$ の計算法

$idf(p)$ を求めるには Hui のアルゴリズム [5] の考えを用いる．まず，接尾辞木を図 3 のように二分木に変更し，葉に D を置く．そして，各内部ノードにその左右の部分木の葉どちらにも存在する値を書き，その個数のみを格納する．数はビットベクトル E に 1 進法で格納する．長さは高々 $2n$ である．このとき，p に対応する接尾辞配列の範囲を $[l,r]$ とすると，$idf(p)$ は $r-l+1$ から E で $r-1$ 番目の 1 と $l-1$ 番目の 1 の間にある 0 の数を引くことで求まる．

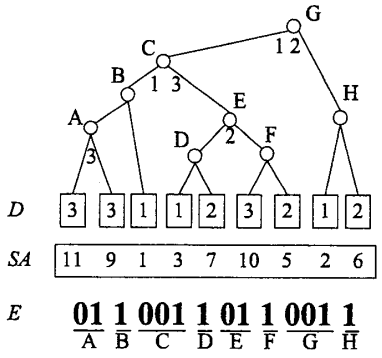

図 3: $idf(p)$ を求めるデータ構造.

参考文献

[1] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in Linear Time? In *ACM Symposium on Theory of Computing*, pages 427–436, 1995.

[2] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987.

[3] 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.

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

[5] L. Hui. Color Set Size Problem with Applications to String Matching. In *Proc. of the 3rd Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, LNCS 644, pages 227–240, 1992.

[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 1969 in LNCS, pages 410–421, 2000.

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

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