接尾辞配列に関する直交領域探索のための省空間なウェーブレット木

数理情報学専攻 48-186213 舘下　正和
指導教員　定兼 邦彦 教授

1　はじめに

より巨大な文字列データを扱う上で，メモリに格納可能な省空間なデータ構造を用いることが重要になっている．その中でも接尾辞配列 [3] は，高速な文字列検索のために有用なデータ構造で，様々な文字列アルゴリズムに用いられている．本研究では，接尾辞配列を用いたアルゴリズムの中でも，接尾辞配列に関する直交領域探索という問題を考える．

2　問題設定

2.1　接尾辞配列

長さ n の文字列 T 中の文字を $T[i]$ $(i = 1, 2, .., n)$ とする．T について，接尾辞 T_j $(j = 1, 2, ..., n)$ を文字列 $T[j..n] = T[j]T[j+1] \cdots T[n]$ と定義し，T の接尾辞配列 $SA[1..n]$ を，T_j が辞書順で小さいほうから i 番目 $(i = 1, ..., n)$ であるとき，$SA[i] = j$ とする．この配列は，1 から n までの値の順列とみなすことができ，全体で $n \log n$ ビット必要である．

2.2　接尾辞配列に関する直交領域探索

ある長さ n の文字列の接尾辞配列 SA について，n 個の点の集合 $P = \{(i, SA[i]) \mid i \in [1, n]\}$ を定義する．この点は，接尾辞の辞書順と，その接尾辞の文字列での開始位置が表現されている．この 2 次元空間内の点の集合から，問い合わせ領域に含まれる点の数を求めることを頻度問い合わせという．領域 $[x_1, x_2] \times [y_1, y_2]$ を問い合わせることで，接尾辞の辞書順の範囲 $[x_1, x_2]$ と文字列での開始位置の範囲 $[y_1, y_2]$ の両方を満たす点を求めることができる．

3　既存研究

一般的な 2 次元直交領域探索に関して，ウェーブレット木を用いた以下の手法が知られている．

3.1　ウェーブレット木

長さ n の文字列 $S[1..n]$ に関して，$rank$ や $select$ といった演算が可能なデータ構造がウェーブレット木 [1] である．文字列 $S[1..n]$ を表すウェーブレット木 $WT(S)$ は木の各ノードにビットベクトルを格納する完全二分木である．文字 $S[i]$ $(1 \leq i \leq n)$ を表す符合の j ビット目 $(1 \leq j \leq n)$ は木の深さ $j-1$ のノードに分散して格納される (根の深さを 0 とする)．

具体的には以下のように構成される．ノード v のビットベクトルを B_v で表す．ウェーブレット木の根ノード ϵ に格納するビットベクトル B_ϵ は S の各文字の符号の最上位ビットを S での出現順に並べたものとする．S を 2 つの文字列 S_0 と S_1 に分割し，S_0 (S_1) には符号の最上位ビットが 0 (1) の文字を S での出現順に並べる．ノード 0 と 1 をつくり，根ノード ϵ の左の子と右の子とする．ノード 0 には，文字列 S_0 の各文字の符号の上から 2 ビット目を順に並べたビットベクトル B_0 を格納する．そして，S_0 の文字を上から 2 ビット目に従って S_{00} と S_{01} に分割する．各ノードで同様のことを行っていく．

3.2　ウェーブレット木による直交領域探索

まず，点の集合 $P = \{p_1, p_2, ..., p_n\}$ を文字列で表す．P の点を x 座標の小さな順に並べ，その y 座標を文字だとみなして文字列 S を作り，このウェーブレット木を構成する．問い合わせ領域を $Q = [x_1, x_2] \times [y_1, y_2]$ とすると，$|P \cap Q|$ は，$S[x_1..x_2]$ の中の文字 c で $c \in [y_1, y_2]$ となるものの数を求めればいい．ウェーブレット木の根ノードから探索を行い，各ビットベクトルの $rank$ 計算を繰り返すことで計算できる．

空間計算量は，ビットベクトルの索引を追加したウェーブレット木を用いるため，$n \log n + \mathrm{o}(n \log n)$ ビットである．時間計算量は，探索するノード数はウェーブレット木の高さに比例し，$\mathrm{O}(\log n)$ 個であるため，全体の計算量は $\mathrm{O}(\log n)$ 時間である．

4　提案手法

本研究では圧縮接尾辞配列というデータ構造を用いて，ウェーブレット木と接尾辞配列との共通した構造を利用することで，より省空間なデータ構造を提案する．

4.1　圧縮接尾辞配列

圧縮接尾辞配列 [2] は，接尾辞配列を圧縮したものである．アルファベットサイズが σ である長さ n の文字列に対して，接尾辞配列のサイズは $n \log n$ ビットであ

るが，これを $O(n \log \sigma)$ ビットにできる．ただし，要素の取り出しには $O(\log n)$ 時間かかる．

4.2　ウェーブレット木と接尾辞配列の共通構造

ウェーブレット木の深さ k には，2^k 個のビットベクトルが存在するが，各ビットベクトルは，最初の k ビット目までが同じとなる接尾辞配列の部分配列に対応している．つまり，ビットベクトル $B_{00\cdots00}$ に対応する部分配列は $[1, \frac{n}{2^k}]$ の範囲で，$B_{00\cdots01}$ は $[\frac{n}{2^k}+1, \frac{2n}{2^k}]$，$\cdots$，$B_{11\cdots11}$ は $[\frac{(2^k-1)n}{2^k}+1, n]$ の範囲に対応する．接尾辞配列の値は文字列における開始位置であるので，深さ k の各ノードに対応する部分配列は，文字列中の長さ $\frac{n}{2^k}$ の部分文字列の接尾辞配列となる．つまり，ビットベクトル $B_{00\cdots00}$ に対応する部分配列は $T[1, \frac{n}{2^k}]$ の接尾辞配列，ビットベクトル $B_{00\cdots01}$ に対応する部分配列は $T[\frac{n}{2^k}+1, \frac{2n}{2^k}]$ の接尾辞配列，$\cdots$，$B_{11\cdots11}$ に対応する部分配列は $T[\frac{(2^k-1)n}{2^k}+1, n]$ の接尾辞配列を求めることで記録できる．

4.3　提案するデータ構造

接尾辞配列 SA を文字列とみなし，ウェーブレット木を作る．まず，根ノード B_ϵ を長さ Δ の $\lceil \frac{n}{\Delta} \rceil$ 個のブロック B^j_ϵ $(j = 0, ..., \lfloor \frac{n}{\Delta} \rfloor)$ に分割する．すると，各ノード v のビットベクトル B_v についても，根ノード B_ϵ の各ブロック B^j_ϵ に対応するブロック B^j_v が $\lceil \frac{n}{\Delta} \rceil$ 個できる．この j 番目のブロック B^j_v に対し，

- データ構造 $B^j_v.rank_0$：
  - ブロック B^j_v の左端までのビットベクトル B_v 中の 0 の数
- データ構造 $B^j_v.blocknum$：
  - ブロック B^j_v の左端までのビットベクトル B_v 中に格納されているビットの数

を格納する．あるビットベクトルの $rank_0$ 演算は，根ノードにおける対応する位置から長さ Δ 分 (ブロック 1 個分) の圧縮接尾辞配列を復元し，その範囲での $rank_0$ の値と，そのブロックの前までの 0 の $rank$ の値 (データ構造 $B^j_v.rank_0$ の値) を足し合わせることで計算できる．

ここで，深さ $k = \epsilon \log n$ ごとに，ビットベクトルに対応する接尾辞配列の部分配列を圧縮接尾辞配列として記録する．すると，深さ dk $(d = 0, 1, ..., \frac{1}{\epsilon} - 1)$ の 2^{dk} 個の各ノード v' を新たに長さ Δ の $\lceil \frac{n}{\Delta \cdot 2^{dk}} \rceil$ 個のブロック $B^j_{v'}$ $(j = 0, ..., \lfloor \frac{n}{\Delta \cdot 2^{dk}} \rfloor)$ に区切りなおすことができる．深さ dk から $(d+1)k$ の各ノード v'' のビットベクトル $B_{v''}$ についても，深さ dk の祖先ノード v_{dk} の各ブロック $B^j_{v_{dk}}$ に対応するブロック $B^j_{v''}$ が $\lceil \frac{n}{\Delta \cdot 2^{dk}} \rceil$ 個できる．この j 番目のブロック $B^j_{v''}$ に対し，上の場合と同様に，$rank_0$ と $blocknum$ を格納する．

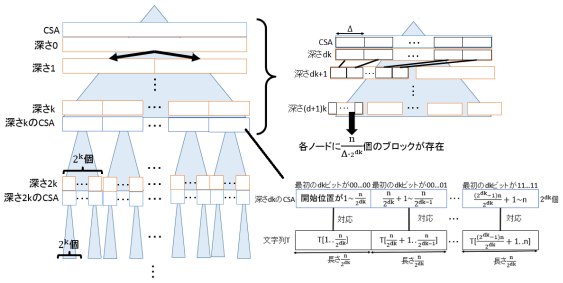

図 1. 提案データ構造.

$\Delta = n^\epsilon \log n$ とすると，アルファベットサイズが σ である長さ n の文字列に対し，接尾辞配列に関する直交領域探索の頻度問い合わせが $O(n^\epsilon \log^2 n)$ 時間で可能である $O(\frac{1}{\epsilon} \cdot n \log \sigma)$ ビットのデータ構造ができる．

5　数値実験

数値実験を行うことにより，提案手法は 10% 近くのサイズまで圧縮できるが，計算時間のオーバーヘッドは大きいことが分かった．また，問い合わせ領域の x 軸方向の範囲が大きくなるほど提案手法は効果があり，また，n が大きくなるほど提案手法の圧縮効果があることが分かった．

6　まとめ

アルファベットサイズが σ である長さ n $(\sigma \ll n)$ の文字列に対して，接尾辞配列に関する直交領域探索の頻度問い合わせが $O(n^\epsilon \log^2 n)$ 時間で可能である $O(\frac{1}{\epsilon} \cdot n \log \sigma)$ ビットの従来より省空間なデータ構造ができた．また，このデータ構造を利用して range-LCP クエリへの省空間データ構造も構築できた．

参考文献

[1] B. Chazelle: “A functional approach to data structures and its use in multidimensional searching,” *SIAM Journal on Computing*, **17**(3), pp. 427-462, 1988.

[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*, pp. 397–406, 2000.

[3] U. Manber and G. Myers: “Suffix arrays: A new method for on-line string searches,” *SIAM Journal on Computing*, **387**(3), pp. 332–347, 1993.