接尾辞配列とその応用

nPk

abstractkeywords

1 記法

文字列は S とかき, 部分文字列は Rust の区間記法にしたがう. つまり, 表 1 の通りにする. 接尾辞配列 SA についても同じようにする.

S, T, ..	文字列	
|S|	文字列の長さ	
S[i]	文字	
S[i..j]	部分文字列	S[i]を含み, S[j]を含まない
S[i..]	接尾辞	S[i]を含む

2 接尾辞配列

文字列 S の接尾辞 S[i..]は頭文字の添え字 i で指定できる. 接尾辞（に対応する添え字）を辞書順に並べたものを接尾辞配列という. 接尾字配列上で二分探索を行うと, 任意の文字列 T の全部の出現個所を $O(|T|\log|S|)$ で求めることができる. 検索文字列についてオンラインに処理できる点で接尾辞配列は Z アルゴリズムや KMP 法よりも優れている.

2.1 文字

Rust 言語において文字は `T: Ord` と抽象化できる. 接尾辞配列においてすべての接尾辞の頭文字がソートされていることを考えれば, 計算量は最悪 $\Omega(|S|\log|S|)$ であることが分かる. 実際に, $O(1)$ の作業メモリを使って, 文字列 `&[T]` の接尾辞配列を $O(|S|\log|S|)$ で計算するアルゴリズムが存在する.

文字列を座標圧縮することを考える. このとき, `T = usize` である. 後述するように, 文字列 `&[usize]` の接尾辞配列は $\Theta(|S|)$ で求めることができる. 座標圧縮の計算量は $\Theta(|S|\log|S|)$ であるから, これによって最悪計算量が悪化することはない. 以下では, 文字は `usize` であるとする. [1]

[1] より小さな数値型を用いても良いが、アルゴリズムが複雑になってしまう。本稿で紹介するアルゴリズムでは `Vec<T>` の最大容量が `isize::MAX` であることを利用した最適化を施しており、副次的にアルゴリズムが単純になった。最上位ビットをフラグに使っているだけなので、2^{31} 文字以下ならば `u32` でも十分だが、型レベルの保証は得られない。

2.2 アルゴリズムの概要

2.2.1 L 型と S 型

定義 2.2.1.1: S[i..] $\gtrless$ S[i+1..] が成り立つとき, S[i]を L 型・S 型の接尾辞という. 末尾の番兵は S 型であるとする. 同様に, L 型・S 型の接尾辞の頭文字も L 型・S 型とする.

定理 2.2.1.1: S[i] $\gtrless$ S[i+1] のとき, S[i]は L 型・S 型である. S[i] $=$ S[i+1] のとき, 両者の型は一致する.

証明 *2.2.1.1*　前者は自明. S[i..k-1] $=$ S[i+1..k]が成り立つ最大の k を考える. これは 1 種類の文字からなる LCP の長さである. S[i+1..]が L 型・S 型のとき, S[k-1] $\gtrless$ S[k] が成り立つ.

系 2.2.1.1: 番兵は S 型なので, 文字列を末尾から走査することで全部の文字の型を$\Theta(|S|)$で判定できる. L 型文字の最上位ビットを立てておけば, 以降は $O(1)$ で判定できる.

2.2.2 誘導ソート

定理 2.2.2.1: S 型接尾辞がソート済みならば, L 型接尾辞も $\Theta(|S|)$ でソートできる. 逆も成り立つ.

証明 *2.2.2.1*　カウントソートにより, S 型接尾辞を SA の正しい場所に格納できる. SA を前から走査する. SA[i]が添え字であり, S[SA[i] $-$ 1]が L 型のとき, 対応するバケットの左端にSA[i] $-$ 1を書き込む. このとき, バケットの左端を指すカウンターをインクリメントする. L 型の定義より, アルゴリズムは正しい. 計算量は $\Theta(|S|)$ である.

定義 2.2.2.1: S[i-1..]が L 型であるような S 型接尾辞 S[i..]をとくに LMS 型（leftmost S type）という. 文字 S[i]についても同様に定義する.

系 2.2.2.1: LMS 型接尾辞がソートされているとき, 接尾辞配列を $\Theta(|S|)$ で構築できる.

2.2.3 部分問題の構築

定義 2.2.3.1: LMS 型文字から次の LMS 型文字までの部分列を LMS 型部分文字列という.

LMS 型接尾辞をソートするためには, LMS 型部分文字列ごとに比較すればよい. すべての LMS 型部分文字列を順序を保ったまま新しい文字で置き換えることができれば, サイズが高々 $\frac{|S|}{2}$ の問題に帰着できる.

定理 2.2.3.1: LMS 型部分文字列を線形時間でソートできる.

証明 *2.2.3.1* LMS 型文字はカウントソートで線形時間でソートできる. これに誘導ソートを適用すると, 頭文字を除いて最初の LMS 型文字までソートできる. つまり, LMS 型部分文字列を線形時間でソートできる.

系 2.2.3.1: 部分問題を線形時間で構築できる.

隣り合う LMS 型部分文字列の一致判定をとることで, $\Theta(|S|)$ ですべての LMS 型部分文字列をリネームできる.

定理 2.2.3.2: 線形時間で接尾辞配列を構築できる

証明 *2.2.3.2* 計算量は $T(n) \leq T\left(\frac{n}{2}\right) + \Theta(n) \leq \Theta(n)$ である.

2.3 アルゴリズムの詳細

2.4 非再帰アルゴリズム

紹介したアルゴリズムの再帰木は鎖型になっている. 行きがけと帰りがけでループを 2 つ用意すると非再帰で書けるはずである. ただし, $O(\log|S|)$ の作業メモリが必要かもしれない.

2.5 制約の緩和

本稿では座標圧縮を仮定したが, 文字列の種類が $O(|S|)$ であれば, 線形時間で接尾辞配列をもとめるアルゴリズムが存在する. ASCII 文字や Unicode 文字など, コンピューターで利用できる文字の種類は定数なので, 接尾辞配列を線形時間で計算できる.

3 参考文献

3.1 接尾辞配列の線形時間アルゴリズム

- LI, Zhize; LI, Jian; HUO, Hongwei. Optimal in-place suffix sorting. Information and Computation, 2022, 285: 104818. https://doi.org/10.1016/j.ic.2021.104818
- NONG, Ge; ZHANG, Sen; CHAN, Wai Hong. Two efficient algorithms for linear time suffix array construction. IEEE transactions on computers, 2010, 60.10: 1471-1484. https://doi.org/10.1109/TC.2010.188