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], S[i..=j]	部分文字列
S[i..]	接尾辞

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。本稿で紹介する実装では `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、オリジナルと比べて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 理論

2 つの接尾辞 S[i..]と S[i+1..]は長さが異なるので, 大小関係が決まる. そこで, 接尾辞およびその頭文字の型を次のように定義する.

定義 2.2.1: $\mathrm{S[i..]} \gneqq \mathrm{S[i+1..]}$ が成り立つとき, S[i]を L 型・S 型の接尾辞という. ただし, 末尾の番兵は S 型であるとする. 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: $\mathrm{S[i]} \gneqq \mathrm{S[i+1]}$ のとき, S[i]は L 型・S 型である. $\mathrm{S[i]} = \mathrm{S[i+1]}$ のとき, 両者の型は一致する.

証明　前者は自明. 後者については, $\mathrm{S[i+k]} \neq \mathrm{S[i+k+1]}$が成り立つ最小の正整数 k を考えればよい. □

接尾辞配列 SA は当然頭文字についてソートされる. 同じ頭文字 S[i]をもつ接尾辞たちが記録される部分をバケット S[i]と呼ぶことにする.

系 2.2.1: 各バケットは L 型接尾辞と S 型接尾辞で区切られている.

証明　頭文字を固定して考える. この頭文字をもつ最小の S 型接尾辞を S[i..]とかくと, $\mathrm{S[i]} < \mathrm{S[i+1]}$が成り立つ. 同様に最大の L 型接尾辞を S[j..]とかくと, $\mathrm{S[j]} > \mathrm{S[j+1]}$が成り立つ. したがって, $\mathrm{S[i+1]} > \mathrm{S[j+1]}$である. □

L 型接尾辞 S[i..]の性質より, $\mathrm{S[i..]} > \mathrm{S[i+1..]}$が成り立つ. したがって, バケット S[i]に書き込まれる L 型接尾辞の 2 文字目以降からなる接尾辞はバケットの左側にある. 同様に S 型接尾辞の 2 文字目以降からなる接尾辞は対応するバケットよりも右側にある. これらの観察より次の定理を得る.

定理 2.2.2: 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　L 型接尾辞をソートすることにする. 接尾辞配列を昇順に走査し, 初期化済みの要素を探す. これを SA[i]とかく. もし S[SA[i]−1..]が L 型接尾辞ならば対応するバケットに前から詰めて書き込む. 同じバケットに属する接尾辞はその 2 文字目以降の部分からなる接尾辞についてもソートされているから, 各バケットに書き込まれた L 型接尾辞はソートされて

いる. バケットソートと累積和でバケットの左端を求めることができるので, アルゴリズムは線形時間で動作する. 逆も同じように証明できる. □

定理 2.2.2 の証明では S 型接尾辞のうち, 1 だけ長い接尾辞が L 型であるもののみを利用している. これより, 次の事実を得る.

定義 2.2.2: 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: 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.

証明　LMS 型接尾辞がソートされているとする. 定理 2.2.2 より, 接尾辞配列 SA を昇順に走査して L 型接尾辞をソートできる. さらに, 接尾辞配列を降順に走査して S 型接尾辞をソートすることができる. □

補題 2.2.1: LMS 型接尾辞は高々 $\left\lfloor \frac{|S|}{2} \right\rfloor$ 個しかない.

補題 2.2.1 より考えるべき接尾辞の数を半分以下にできましたが, 問題サイズは半分になっていません. 接尾辞の長さが高々 $|S|$ だからです. LMS 型接尾辞を比較することを考えると, 2 つの LMS 型文字に挟まれてできる部分文字列単位で比較すればよいことが分かります. これを利用して文字列を圧縮し, 問題のサイズを半分以下にすることが次の目標です.

定義 2.2.3: 部分文字列 S[i..=j]のうち, S[i]と S[j]のみが LMS 型であるものを LMS 型文字列という. とくに, 番兵も LMS 型部分文字列であるとする.

補題 2.2.2: LMS 型接尾辞と LMS 部分文字列の数は一致する.

LMS 部分文字列を順序を保ったまま新しい文字に置き換えるためには, これらをソートする必要があります. 逆に, LMS 部分文字列がソートされていれば

隣り合う 2 つの一致判定をとることで, 順序を保ったまま改名することができます.

定理 2.2.3: LMS 部分文字列を $\Theta(|S|)$ でソートできる.

証明　バケットソートにより, すべての LMS 文字を線形時間でソートできる. 系 2.2.2 の証明と同じようにして, 頭文字を除く最初の LMS 文字までソートできることが分かる. これはすべての LMS 部分文字列を含んでいる. □

系 2.2.3: 線形時間で接尾辞配列を構築できる.

証明　時間計算量は $T(n) \leq T(\frac{n}{2}) + \Theta(n) = \Theta(n)$ である. □

2.3 アルゴリズム

本節では作業メモリが $O(1)$ の線形時間接尾辞配列構築アルゴリズムを紹介します. 2 章 1 節 でも述べたように,

アルゴリズムの解説

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固定なので, 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