接尾辞木に基づくLZ77とLPF配列の変種の計算

クップル・ドミニク

M&D データ科学センター
東京医科歯科大学

論文紹介

- "Non-Overlapping LZ77 Factorization and LZ78 Substring Compression Queries with Suffix Trees." *Algorithms* 14(2): 44 (2021)

この発表

Lempel–Ziv 77 (LZ77) 分解の変種

- 重複なし Lempel–Ziv 77

longest previous factor 表 (LPF) の変種:

- LPnF: longest previous non-overlapping factor 表

研究の寄与

- LPnF の $2n$ ビット表現
- 上記の分解と表を $\mathcal{O}(n \lg n)$ 時間で計算できる少メモリーアルゴリズム

設定 & 例

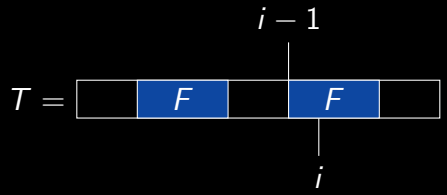
- T: 入力のテキスト, $n-1 := |T|$ は T の長さを示す
- Σ　は T のアルファベット, $\sigma := |\Sigma|$ は Σ のサイズを示す
- $\$ < c \; \forall \; c \in \Sigma$

i	1	2	3	4	5	6	7	8	9	10	11
$T\$$	a	b	b	a	b	b	a	b	a	b	$
LPF	0	0	1	5	4	3	2	3	2	1	0
LPnF	0	0	1	3	3	3	2	3	2	1	0

簡潔な表現

Sadakane'07: PLCP 配列の $2n$ ビット表現

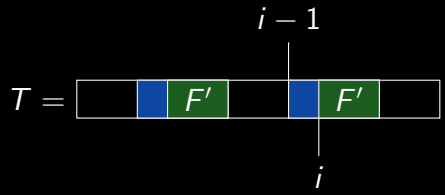
- PLCP[i] は $(T\$)[i..]$ と辞書式順序で直接の前の接尾辞の共通接頭辞の長さ
- $(T\$)[n] = \$ \Rightarrow \mathrm{PLCP}[n] = 0$
- $\mathrm{PLCP}[i] \leq n \; \forall \; i \in [1..n]$



簡潔な表現

Sadakane'07: PLCP 配列の $2n$ ビット表現

- PLCP[i] は $(T\$)[i..]$ と辞書式順序で直接の前の接尾辞の共通接頭辞の長さ
- $(T\$)[n] = \$ \Rightarrow \mathrm{PLCP}[n] = 0$
- $\mathrm{PLCP}[i] \leq n \; \forall \; i \in [1..n]$



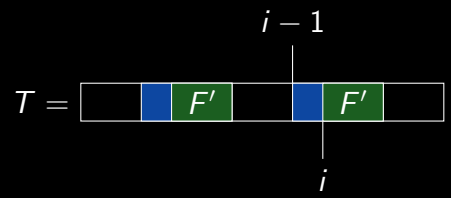
簡潔な表現

Sadakane'07: PLCP 配列の $2n$ ビット表現

- PLCP[i] は $(T\$)[i..]$ と辞書式順序で直接の前の接尾辞の共通接頭辞の長さ
- $(T\$)[n] = \$ \Rightarrow \mathrm{PLCP}[n] = 0$
- $\mathrm{PLCP}[i] \leq n \; \forall \; i \in [1..n]$
- $\mathrm{PLCP}[i] \geq \mathrm{PLCP}[i-1] - 1$
- よって、$\mathrm{PLCP}[i] - \mathrm{PLCP}[i-1] + 1 \geq 0$

$\Rightarrow$ 1 進数で値 $\mathrm{PLCP}[i] - \mathrm{PLCP}[i-1] + 1$ を格納した場合、以下のようになる

- $\sum_{i=1}^{n}(\mathrm{PLCP}[i] - \mathrm{PLCP}[i-1] + 1) = \mathrm{PLCP}[n] + n = n$、ただし $\mathrm{PLCP}[0] := 0$
- '1' の個数： n 、'0' の個数：n $\Rightarrow 2n$ ビット



同様に表現できる関連な配列

引用	配列	分解
Sadakane'07	PLCP	lcpcomp (Dinklage+'17)
Belazzougui and Cunial'14	matching statistics	Relative LZ (Ziv+'93)
Bannai,Inenaga,K.'17	LPF	LZ77
この発表	LPnF	重複なし LZ
K.'21	LPnrF	逆 LZ (Kolpakov+'09)

- 引用： $2n$ ビット表現の提案

既存研究

	引用	時間	ビット
LPnF:	Crochemore,Tischler'11	$\mathcal{O}(n)$	$\mathcal{O}(n \lg n)$
	Crochemore+'12	$\mathcal{O}(n)$	$\mathcal{O}(n \lg n)$
	Ohlebusch,Weber'19:	$\mathcal{O}(n)$	$\mathcal{O}(n \lg n)$

- $2n$ ビット表現で、$\mathcal{O}(n \lg n)$ ビットの計算領域は最適だと見なすことができなくなる
- 時間をなるべく増大させずに、計算領域を減らすことができる？

研究結果

- $\epsilon > 0$ は任意に選択可能である定数
- 基本時間量: $\mathcal{O}(\epsilon^{-1} n)$
- $t_{\mathsf{SA}} = \log_\sigma^\epsilon n$: 接尾辞配列の access 時間

分解	ビット	計算時間
重複なし LZ ・ LPnF	$(1+\epsilon) n \lg n + \mathcal{O}(n)$ $\mathcal{O}(\epsilon^{-1} n \lg \sigma)$	$\mathcal{O}(\epsilon^{-1} n)$ $\mathcal{O}(\epsilon^{-1} n t_{\mathsf{SA}})$

重複なし LZ 分解

重複なし LZ 分解

$T =$ a b b a b b a b a b

1 2 3 4 5 6 7 8 9 10

符号化：

重複なし LZ 分解

$T =$ a b b a b b a b a b

1 2 3 4 5 6 7 8 9 10

符号化：a

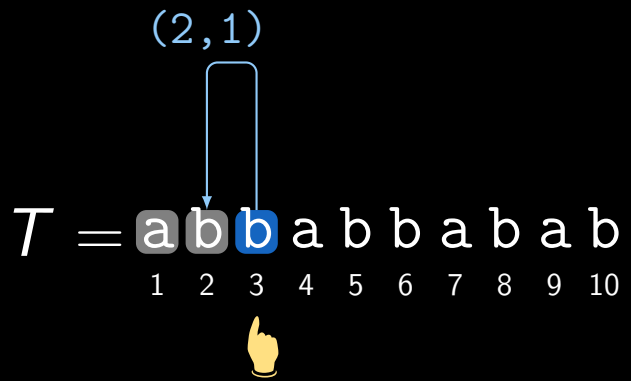
重複なし LZ 分解

$T =$ a b b a b b a b a b

1 2 3 4 5 6 7 8 9 10

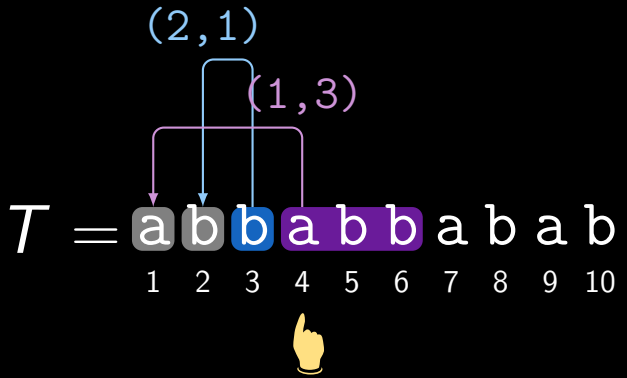
符号化：ab

重複なし LZ 分解



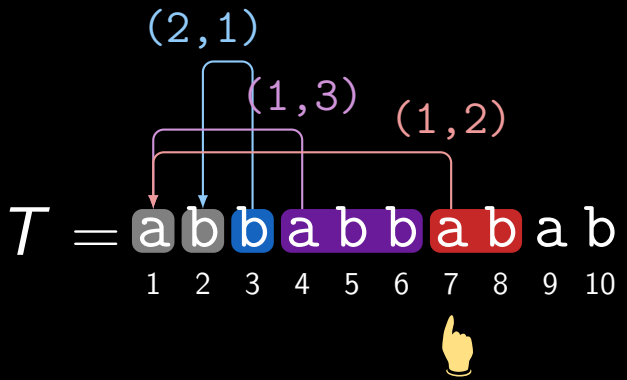
符号化：ab(2,1)

重複なし LZ 分解



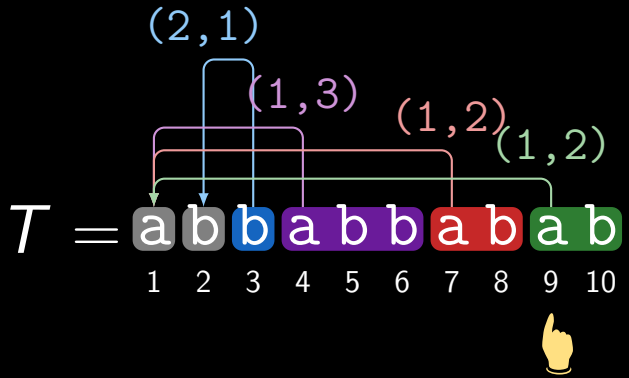
符号化：ab(2,1)(1,3)

重複なし LZ 分解

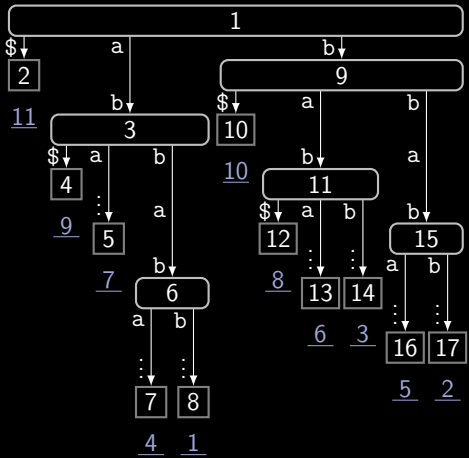


符号化：ab(2,1)(1,3)(1,2)

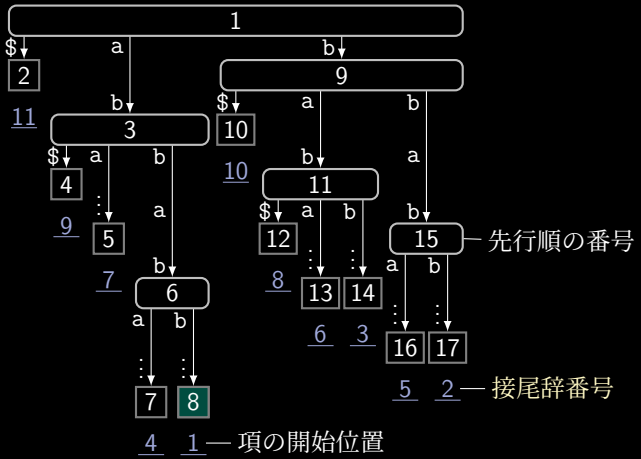
重複なし LZ 分解



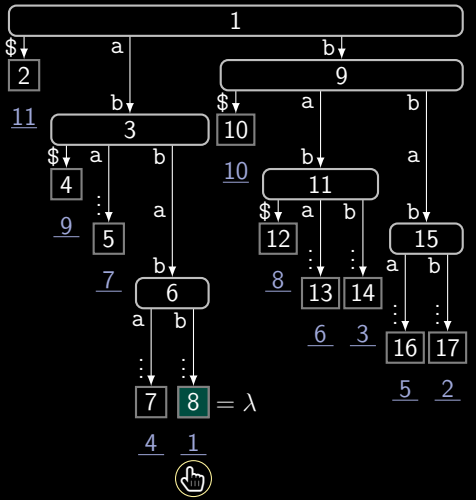
符号化：ab(2,1)(1,3)(1,2)(1,2)



$T = \texttt{abbabbabab}$

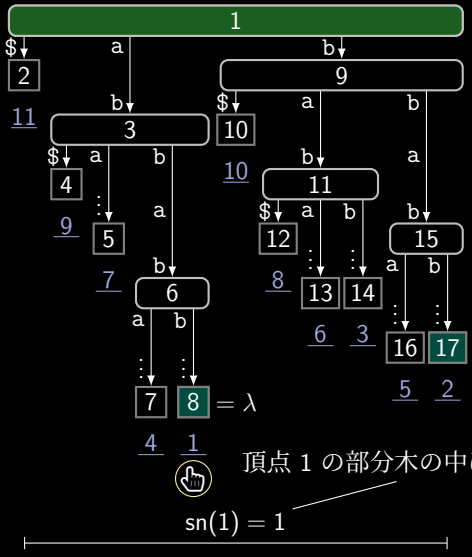


$T = \texttt{abbabbabab}$



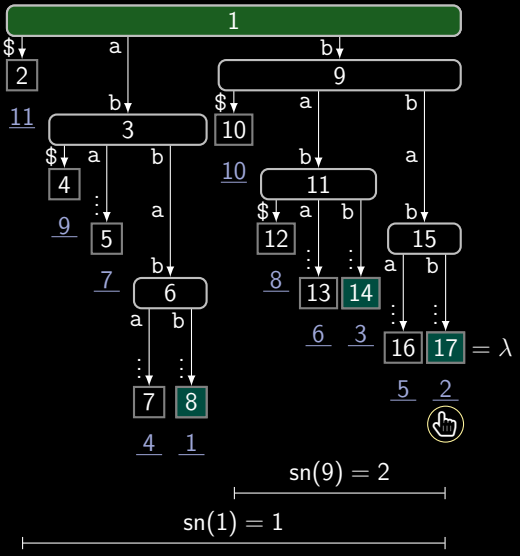
- 根から葉 λ まで辿る、ただし　λ の接尾辞番号は項の開始位置を一致する

$T =$ abbabbabab



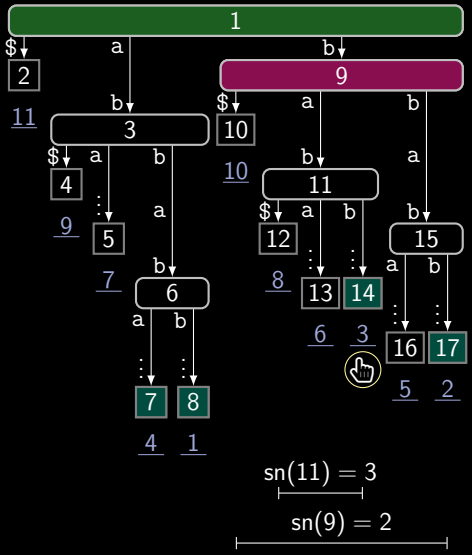
- 根から葉 λ まで辿る、ただし　λ の接尾辞番号は項の開始位置を一致する
- 部分木の最小の接尾辞番号は $< \mathsf{sn}(\lambda)$ に限り、続ける

$T = \mathtt{a}|\mathtt{bbabbabab}$



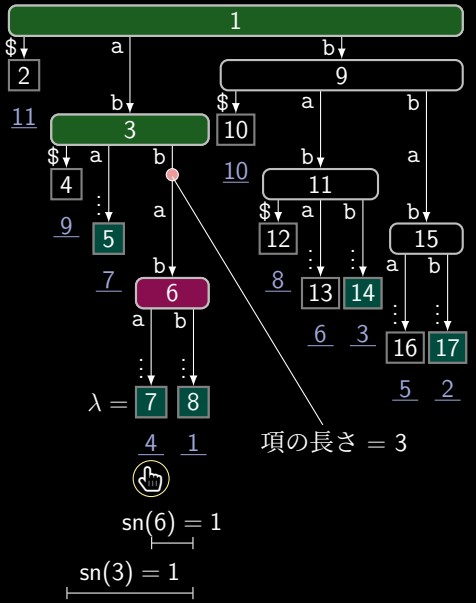
- 根から葉 λ まで辿る、ただし　λ の接尾辞番号は項の開始位置を一致する
- 部分木の最小の接尾辞番号は $< \mathrm{sn}(\lambda)$ に限り、続ける

$T = \mathrm{a}|\mathrm{b}|\mathrm{babbabab}$



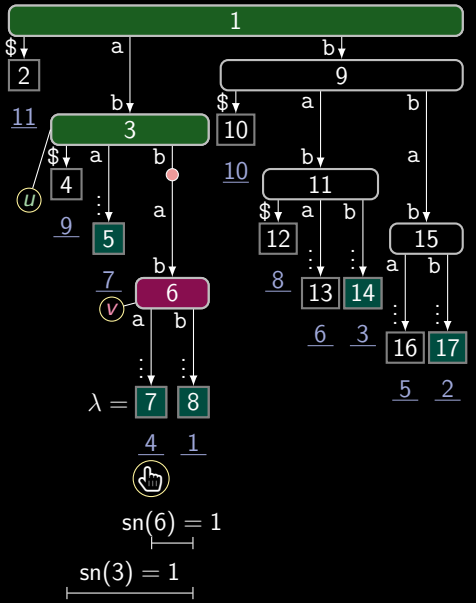
- 根から葉 λ まで辿る、ただし　λ の接尾辞番号は項の開始位置を一致する
- 部分木の最小の接尾辞番号は $< \mathsf{sn}(\lambda)$ に限り、続ける
- 辿り着いた最も下の頂点は参照を「目撃」する
- 参照の出現は重複しないと確認

$T = \mathsf{a}|\mathsf{b}|\mathsf{b}|\mathsf{abbabab}$



- 根から葉 λ まで辿る、ただし　λ の接尾辞番号は項の開始位置を一致する
- 部分木の最小の接尾辞番号は $< \mathrm{sn}(\lambda)$ に限り、続ける
- 辿り着いた最も下の頂点は参照を「目撃」する
- 参照の出現は重複しないと確認
- 重複したら、項を刈る

$T = \mathrm{a}|\mathrm{b}|\mathrm{b}|\mathrm{abb}|\mathrm{abab}$



- $\mathsf{sn}(w)$：頂点 w の部分木の中に最小の接尾辞番号
- $\mathsf{strdepth}(w)$：頂点 w の文字列深さ
- 頂点 v：辿り着いた最も下の頂点
- 頂点 u：v の親

項の長さは以下の選択の最大値

- $\min(\mathsf{sn}(\lambda) - \mathsf{sn}(v), \mathsf{strdepth}(v))$
- $\min(\mathsf{sn}(\lambda) - \mathsf{sn}(u), \mathsf{strdepth}(u))$

$T = \mathtt{a|b|b|abb|abab\$}$

重複なし LZ : 纏め

重複なし LZ を計算できるアルゴリズムを提案した。

- $\mathcal{O}(n)$ 回関数　sn$(\cdot)$ を呼び出す
- $\mathcal{O}(z)$ 回関数　strdepth$(\cdot)$　を呼び出す、ただし z は項の個数

LPnF を同様に計算できる

- 各葉を項の開始位置として扱う
- 根からではなく、接尾辞木リンクから計算し始めた場合、追加される文字列の長さに線形的な時間で木の走査が可能

$\Rightarrow$ $\mathcal{O}(n)$ 回 sn$(\cdot)$ 及び strdepth$(\cdot)$　を呼び出す

sn$(\cdot)$ または strdepth$(\cdot)$ を呼び出すための時間は?

接尾辞木

Farach–Colton+'00 の接尾辞木構築アルゴリズム

- $\mathcal{O}(n)$ 時間
- $\mathcal{O}(n \lg n)$ ビットの計算領域

少メモリーの $\mathcal{O}(n)$ 時間構築アルゴリズム:

1. Fischer+'18:
   - $(1+\epsilon)n \lg n + \mathcal{O}(n)$ ビット、ただし $\epsilon \in (0, 1]$
   - $\mathsf{sn}(\cdot)$ と $\mathsf{strdepth}(\cdot)$ を　$t_{\mathsf{SA}} = \mathcal{O}(1/\epsilon)$ 時間で計算できる
2. Munro+'17,Belazzougui+'20:
   - $\mathcal{O}(n \lg \sigma)$ ビット
   - 接尾辞配列のサンプリングで $t_{\mathsf{SA}} = \log_\sigma^\epsilon n$ を満たす

纏め

- アルゴリズムは $\mathcal{O}(n)$ 回関数　$\mathsf{sn}(\cdot)$ 及び $\mathsf{strdepth}(\cdot)$　を呼び出す
- 各関数呼び出しは t_{SA} 時間が必要

接尾辞木の表現

表現	領域 (ビット)	t_{SA}
1	$(1+\epsilon)n \lg n + 1/\epsilon^{-1}\,\mathcal{O}(n)$	$\mathcal{O}(1/\epsilon)$
2	$\mathcal{O}(n \lg \sigma)$	$\mathcal{O}(\log_\sigma^\epsilon n)$

未解決問題

- $\mathcal{O}(n \lg \sigma)$ ビット $+ \mathcal{O}(n)$ 時間で LPF 表の変種を計算できる？

$\mathcal{O}(r)$ ワードの領域, r は Burrows–Wheeler transform (BWT) の個数を示す

- matching statistics: Bannai,Gagie,I'20
- LPF: Prezza,Rosone'20
- 重複なし LZ77 なら、 Policriti,Prezza'18 のアルゴリズムを応用できる

  既存 項を計算しながら、 BWT を更新

  提案 その変わりに項の長さを決定したあと BWT を 更新
  - LPnF の計算について応用は現状困難:
    - 両方に、BWT に文字を挿入と削除すべき

    $\Rightarrow \sum_{i=1}^{n} \mathsf{LPnF}[i] = \mathcal{O}(n^2)$ の演算
- LPnF の $\mathcal{O}(r)$ 領域で計算できるアルゴリズムがまだ提案されていない

ご清聴ありがとうございました