

準リアルタイム接尾辞木構築に関する 応用について

January 10, 2026

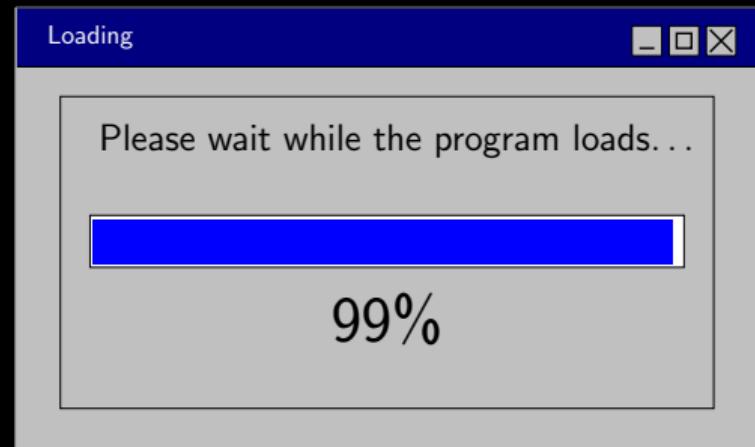
Dominik Köppel¹ Gregory Kucherov²

¹: 山梨大学 コンピュータ理工学

²: ギュスターヴ・エッフェル大学

背景

- 古典的な問題：パターン照合，テキスト比較，データ圧縮など
- 大規模なテキスト集合に対して（例：DNA配列，リポジトリなど）
しかし：
- ほとんどのアルゴリズムはオフラインで動作：いかなる出力も生成される前に全体のテキストを必要とする
- 全入力が処理されるまで待つ必要がある！
- 実世界の現象：バーが99%に達したときにアルゴリズムが本格的に動き始め，ユーザーは待たされる…

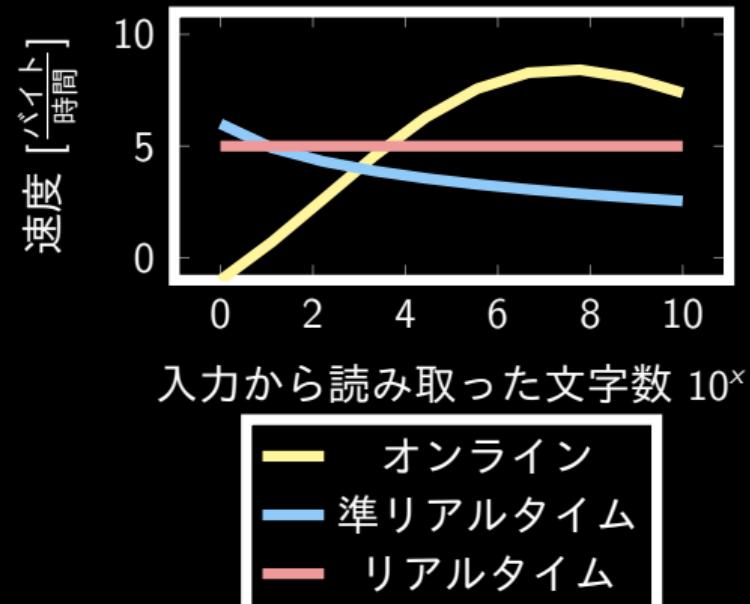


オンラインアルゴリズム

定義 (オンラインアルゴリズム)

新しい文字が到着するたびに、利用可能な解を漸進的に更新する

- ほとんどのオンラインアルゴリズムは平均化されている：1文字あたりの最悪時間は高くなる可能性がある！
 - リアルタイムアルゴリズム：1文字あたりの最悪時間は定数
- ここでは：
- 準リアルタイムアルゴリズム：
1文字あたりの最悪時間は入力サイズに対して対数的



リアルタイムアルゴリズム

リアルタイムで何ができるか？

- スライディングウィンドウを用いたパターン照合 Galil'76
- 回文認識 Galil'76
- Lempel-Ziv 78分解 Hartman,Rodeh'85

準リアルタイムで何ができるか？

- 接尾辞木構築 Breslauer,Italiano'13
- Weinerのアルゴリズム Weiner'73に基づく

Weinerの接尾辞木構築

入力：

- 長さ n のテキスト $T[1..n]$ (整数アルファベット Σ 上)
- 文字は右から左へ到着する
- $T[n] = \$$ は一意の終端文字とする

出力： T の接尾辞木 $ST(T)$

目標： $T[j]$ が到着したとき， $ST(T[j+1..])$ から $ST(T[j..])$ を構築する
($j = n, n-1, \dots, 1$)

例 ($T = aababaa\$$)

Weinerの接尾辞木構築

入力：

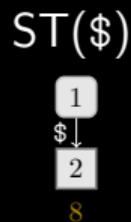
- 長さ n のテキスト $T[1..n]$ (整数アルファベット Σ 上)
- 文字は右から左へ到着する
- $T[n] = \$$ は一意の終端文字とする

出力： T の接尾辞木 $ST(T)$

目標： $T[j]$ が到着したとき， $ST(T[j+1..])$ から $ST(T[j..])$ を構築する
($j = n, n-1, \dots, 1$)

例 ($T = aababaa\$$)

- $\$$ から開始



Weinerの接尾辞木構築

入力：

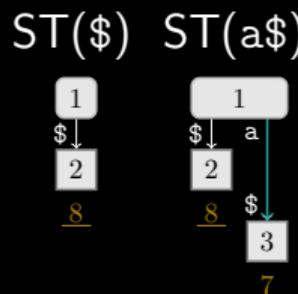
- 長さ n のテキスト $T[1..n]$ (整数アルファベット Σ 上)
- 文字は右から左へ到着する
- $T[n] = \$$ は一意の終端文字とする

出力： T の接尾辞木 $ST(T)$

目標： $T[j]$ が到着したとき， $ST(T[j+1..])$ から $ST(T[j..])$ を構築する
($j = n, n-1, \dots, 1$)

例 ($T = aababaa\$$)

- $\$$ から開始
- $a\$$ を挿入



Weinerの接尾辞木構築

入力：

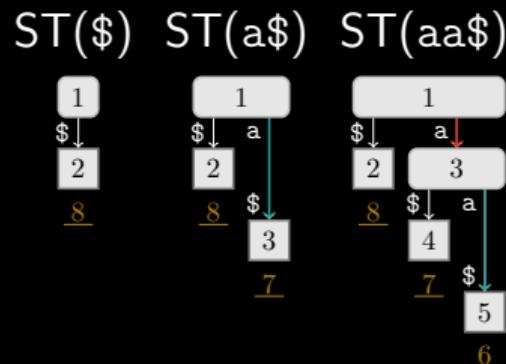
- 長さ n のテキスト $T[1..n]$ (整数アルファベット Σ 上)
- 文字は右から左へ到着する
- $T[n] = \$$ は一意の終端文字とする

出力： T の接尾辞木 $ST(T)$

目標： $T[j]$ が到着したとき， $ST(T[j+1..])$ から $ST(T[j..])$ を構築する
($j = n, n-1, \dots, 1$)

例 ($T = aababaaa\$$)

- $\$$ から開始
- $a\$$ を挿入
- $aa\$$ を挿入



Weinerの接尾辞木構築

入力：

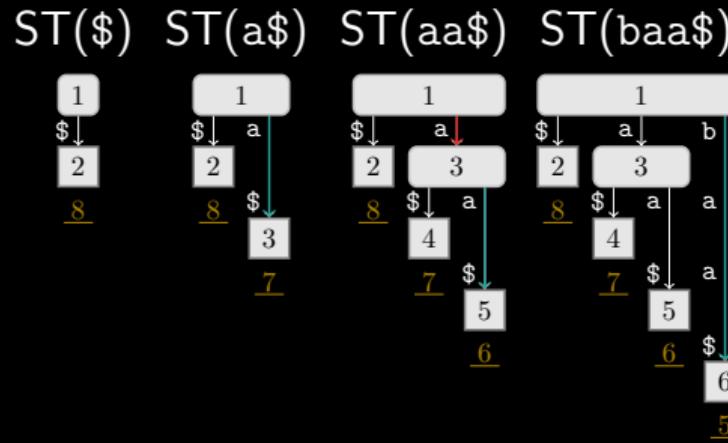
- 長さ n のテキスト $T[1..n]$ (整数アルファベット Σ 上)
- 文字は右から左へ到着する
- $T[n] = \$$ は一意の終端文字とする

出力： T の接尾辞木 $ST(T)$

目標： $T[j]$ が到着したとき， $ST(T[j+1..])$ から $ST(T[j..])$ を構築する
($j = n, n-1, \dots, 1$)

例 ($T = aababaaa\$$)

- $\$$ から開始
- $a\$$ を挿入
- $aa\$$ を挿入
- $baa\$$ を挿入

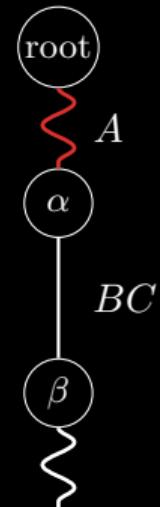


挿入点

$T[j..]$ を $\text{ST}(T[j+1..])$ に挿入するとき、 ST をどこで更新するか？

素朴な解法

1. $\text{ST}(T[j+1..])$ で根から $T[j..]$ に一致する最長経路を検索する

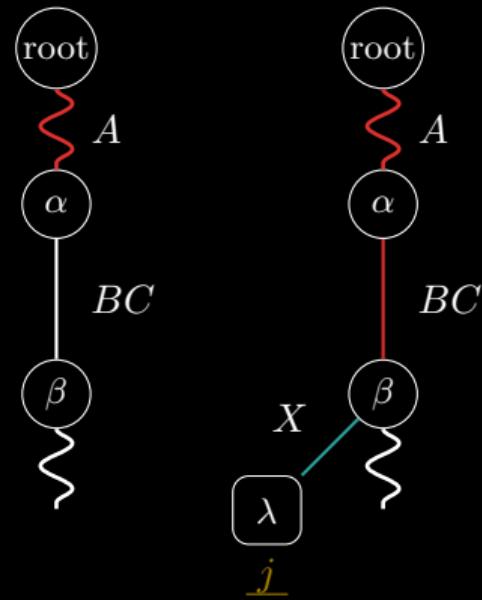


挿入点

$T[j..]$ を $\text{ST}(T[j+1..])$ に挿入するとき、 ST をどこで更新するか？

素朴な解法

1. $\text{ST}(T[j+1..])$ で根から $T[j..]$ に一致する最長経路を検索する
2. $T[j..] = ABCX$ の場合、頂点 β で分岐する



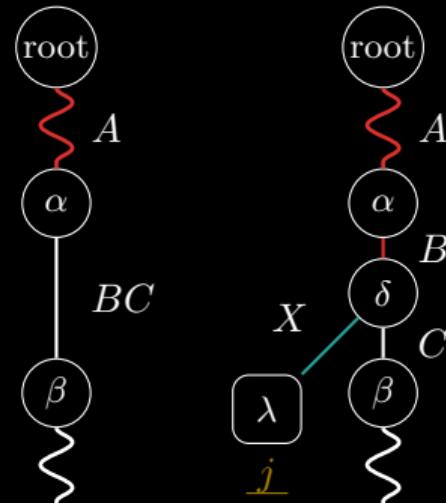
挿入点

$T[j..]$ を $\text{ST}(T[j+1..])$ に挿入するとき、 ST をどこで更新するか？

素朴な解法

1. $\text{ST}(T[j+1..])$ で根から $T[j..]$ に一致する最長経路を検索する
2. $T[j..] = ABCX$ の場合、頂点 β で分岐する
3. $T[j..] = ABX$ の場合、
 - 3.1 頂点 (α, β) を分割して頂点 δ を作成する
 - 3.2 頂点 δ で分岐する

いずれの場合も、分岐点を挿入点と呼ぶ。



挿入点問題

重要な観察

- 接尾辞 $T[j..]$ は $ST(T[j..])$ において葉 λ で表される
- $ST(T[j+1..])$ を $ST(T[j..])$ に更新するには、 λ を挿入する必要がある
- これには $ST(T[j+1..])$ における $T[j..]$ の挿入点を見つける必要がある

問題 (INSERTIONPOINT)

入力: 接尾辞木 $ST(T[j+1..])$ と文字 $T[j]$.

出力: $T[j+1..]$ に出現する $T[j..]$ の最長接頭辞の位置.

t_{SU} を INSERTIONPOINT を解く時間計算量とする.

時間計算量 t_{SU}

素朴なアプローチ：

- $ST(T[j+1..])$ を根から $T[j..]$ に一致するまで走査する
- $O(n)$ 時間

より良い方法はあるか？

| t_{SU} | 参考文献 |
|---|-------------------------|
| $O(\lg \sigma)$ 平均 | Weiner'73 |
| $O(\lg n)$ | Amir,Kopelowitz,+ '05 |
| $O(\sigma \log \log n)$ | Breslauer,Italiano'13 |
| $O(\log \log n + \log \log \sigma)$ 期待 | Kopelowitz'12 |
| $O(\log \log n + \frac{\log^2 \log \sigma}{\log \log \log \sigma})$ | Fischer,Gawrychowski'15 |
| $O(\log \log n)$ ただし $\sigma = O(\log^{1/4} n)$ | Kucherov,Nekrich'17 |

挿入点について

INSERTIONPOINTはどんな情報を保持するか？

- $T[j..n]$ と $ST(T[j + 1..n])$ 内の $T[j..n]$ に辞書順で最も近い接尾辞との最長共通接頭辞

最長繰り返し接頭辞問題を解くことができる！

問題 (LONGESTREPEATINGPREFIX)

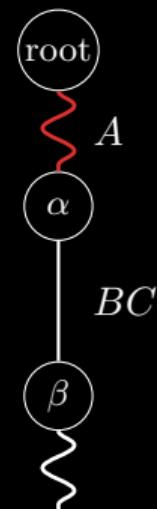
入力: 接尾辞木 $ST(T[j + 1..])$ と文字 $T[j]$.

出力: $T[j..]$ の最長繰り返し接頭辞の長さ.

INSERTIONPOINTを用いてLONGESTREPEATINGPREFIXを解く

アルゴリズム

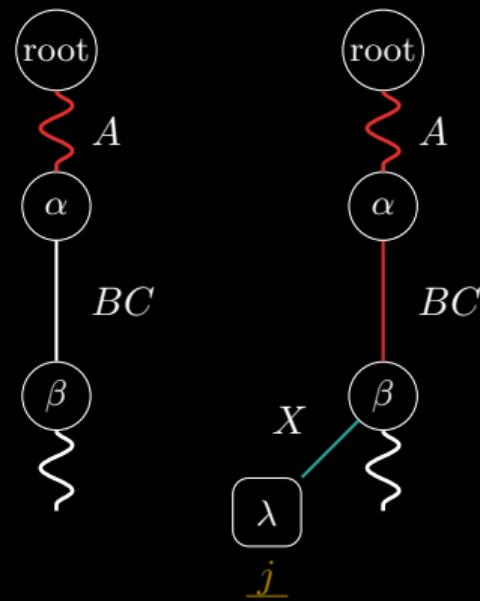
1. $ST(T[j + 1..])$ において $T[j..]$ のINSERTIONPOINTを計算する



INSERTIONPOINTを用いてLONGESTREPEATINGPREFIXを解く

アルゴリズム

1. $ST(T[j + 1..])$ において $T[j..]$ のINSERTIONPOINTを計算する
2. INSERTIONPOINTが β ならば $|ABC|$ を返す



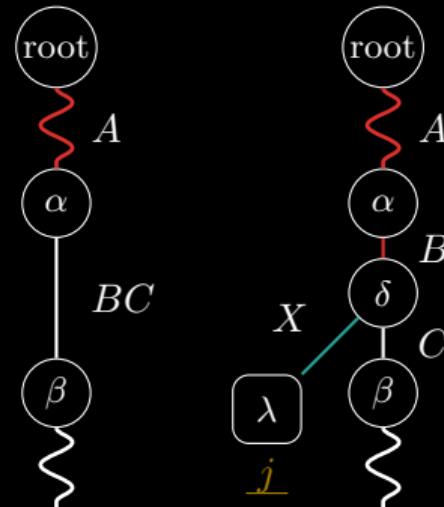
INSERTIONPOINTを用いてLONGESTREPEATINGPREFIXを解く

アルゴリズム

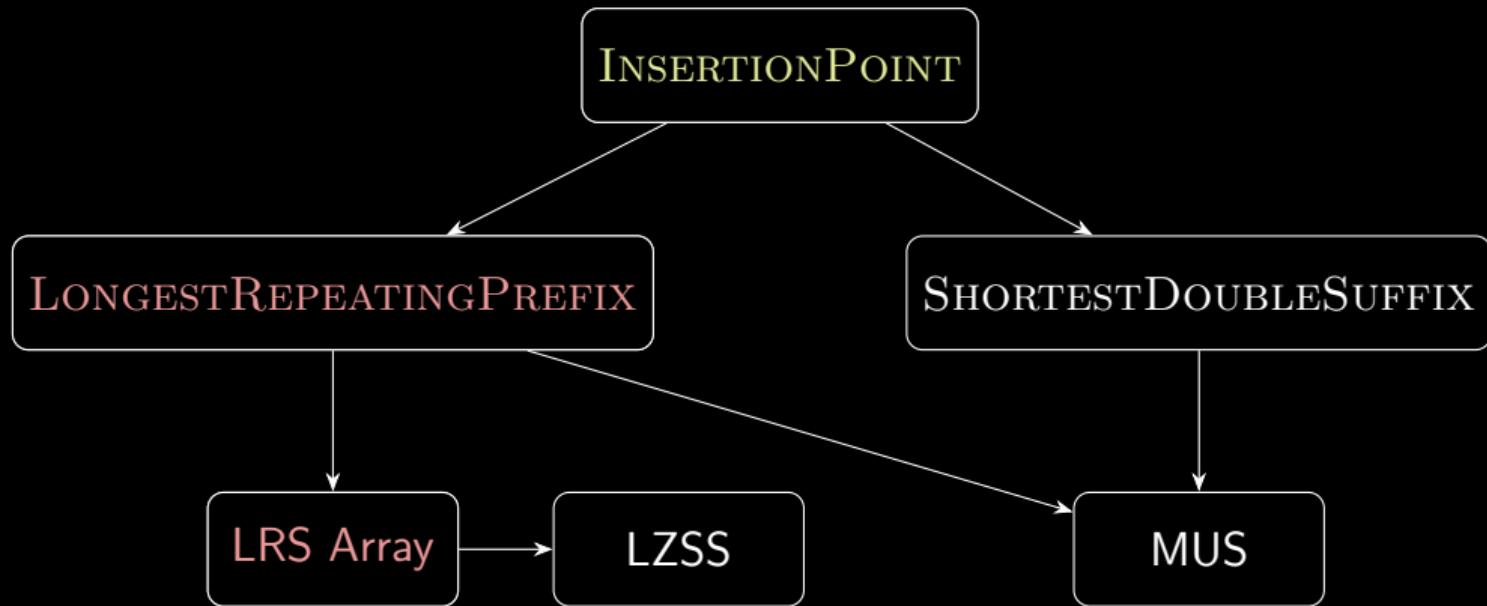
1. $ST(T[j + 1..])$ において $T[j..]$ のINSERTIONPOINTを計算する
2. INSERTIONPOINTが β ならば $|ABC|$ を返す
3. INSERTIONPOINTが δ ならば $|AB|$ を返す

いずれの場合も：INSERTIONPOINTの文字深さを出力する

他に何ができるか？



最長繰り返し接尾辞配列



最長繰り返し接尾辞配列

LONGEST REPEATING PREFIX を使って計算できる：

定義 (最長繰り返し接尾辞配列 (LRS))

文字列 $T[1..n]$ の最長繰り返し接尾辞配列 (LRS) は配列 $LRS[1..n]$ であり, 各位置 $j \in [1..n]$ に対して, $LRS[j]$ は $T[1..j]$ の中で少なくとも2回出現する最長の接尾辞の長さである.

LRS をオンラインで計算する既知の解法, 1文字あたりの時間

| 時間 | 参考文献 |
|------------------|-------------------------|
| $O(\log^3 n)$ | Okanohara, Sadakane '08 |
| $O(\log^2 n)$ 平均 | Prezza, Rosone '20 |
| $O(t_{SU})$ | この研究 |

準リアルタイムLRS計算

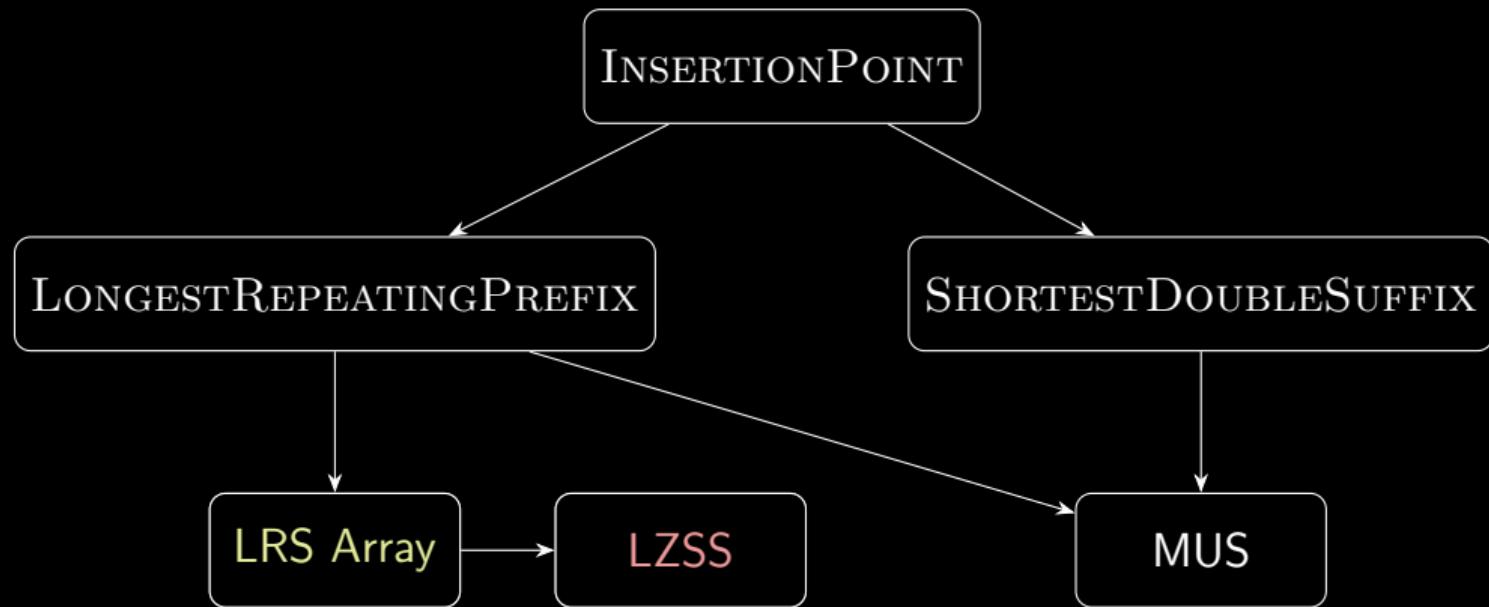
Amir,Landau,+’02の考え方従う：

- 反転テキスト $\overleftarrow{T} = T[n]T[n-1]\dots T[1]$ 上で ST をオンラインで計算する
- LONGESTREPEATINGPREFIXクエリを使って LRS を計算する

定理

LRS を1文字あたり $O(t_{SU})$ 最悪時間でオンライン計算できる.

LZSS計算



Lempel–Ziv–Storer–Szymanski (LZSS)

$T = a\ a\ b\ a\ b\ a\ a\ \$$
1 2 3 4 5 6 7 8



符号化：

- 最長の読み取り可能な部分文字列を後方参照で置き換える

Lempel–Ziv–Storer–Szymanski (LZSS)

$T = \boxed{a} a b a b a a \$$

1 2 3 4 5 6 7 8



符号化： a

- 最長の読み取り可能な部分文字列を後方参照で置き換える
- 単一の文字から開始する（置き換えなし）

Lempel–Ziv–Storer–Szymanski (LZSS)



符号化 : $a(1, 1)$

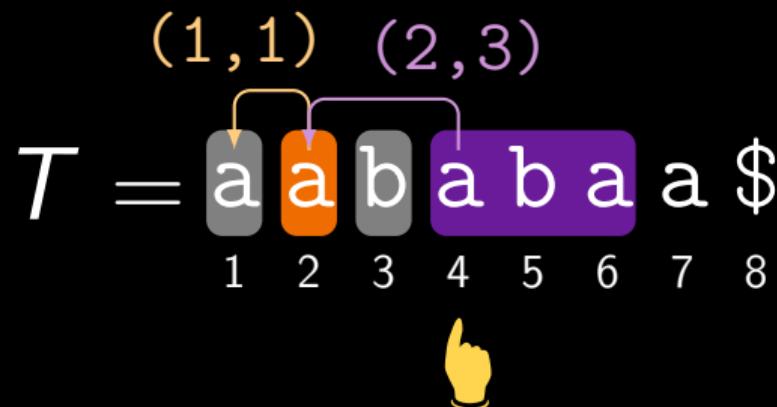
- 最長の読み取り可能な部分文字列を後方参照で置き換える
- 単一の文字から開始する（置き換えなし）
- 最長の以前の出現で置き換える（重複は許可される）

Lempel–Ziv–Storer–Szymanski (LZSS)



- 最長の読み取り可能な部分文字列を後方参照で置き換える
- 単一の文字から開始する（置き換えなし）
- 最長の以前の出現で置き換える（重複は許可される）

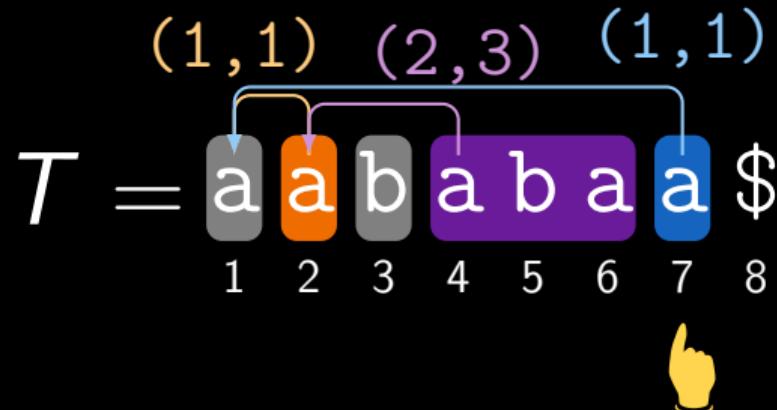
Lempel–Ziv–Storer–Szymanski (LZSS)



符号化 : $a(1, 1)b(2, 3)$

- 最長の読み取り可能な部分文字列を後方参照で置き換える
- 単一の文字から開始する（置き換えなし）
- 最長の以前の出現で置き換える（重複は許可される）

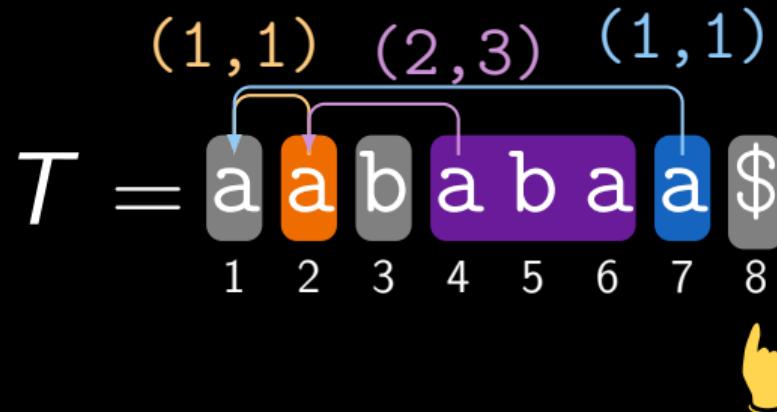
Lempel–Ziv–Storer–Szymanski (LZSS)



符号化 : $a(1,1)b(2,3)(1,1)$

- 最長の読み取り可能な部分文字列を後方参照で置き換える
- 単一の文字から開始する（置き換えなし）
- 最長の以前の出現で置き換える（重複は許可される）

Lempel–Ziv–Storer–Szymanski (LZSS)



符号化 : $a(1,1)b(2,3)(1,1)\$$

- 最長の読み取り可能な部分文字列を後方参照で置き換える
- 単一の文字から開始する（置き換えなし）
- 最長の以前の出現で置き換える（重複は許可される）

LZSS計算

LZSSをオンラインで計算する既知の解法、1文字あたりの時間：

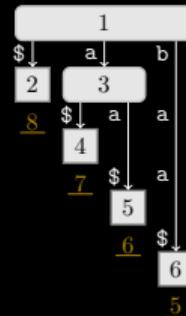
| 時間 | 参考文献 |
|---------------------|-----------------------|
| $O(\log \sigma)$ 平均 | Gusfield'97 |
| $O(t_{SU})$ | この研究 |
| $O(\log^3 n)$ | Okanohara,Sadakane'08 |
| $O(\log^2 n)$ 平均 | Starikovskaya'12 |
| $O(\log n)$ 平均 | Yamamoto+'14 |

注：後者の解法はコンパクトな空間を使用する

準リアルタイム LZSS 計算

- 仮定: $j = 3$, すなわち
 $T = \text{aababaa}$ の $T[1..3] = \text{aab}$ を
処理済み
- $\text{ST}(\overleftarrow{T[1..3]}) = \text{ST}(\text{baa})$ を持つ
- 次の要素 F が $T[4..] = \text{abaa}$ か
ら始まるこことを知っている

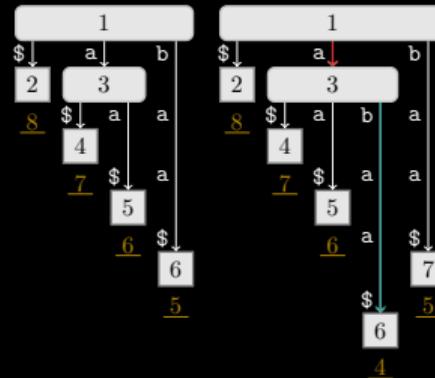
目標: $p = 4$ から始まる F の長さを
計算する



準リアルタイム LZSS 計算

目標 : $p = 4$ から始まる F の長さを計算する

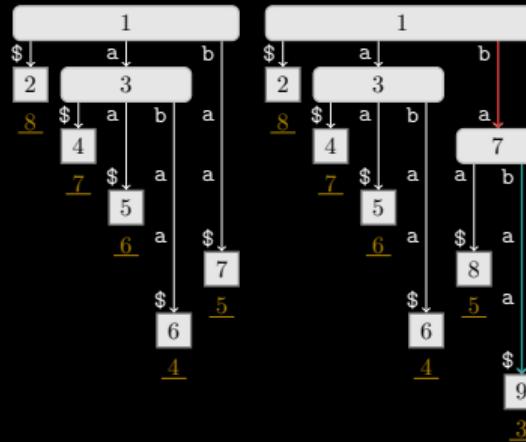
- $j \leftarrow 4$ を $\text{ST}(\overleftarrow{T[1..j]}) = \text{ST}(\text{abaa})$ に更新
- 挿入点の位置は $L = \text{a}$
- $|L| \geq p + 1 - j = 1$ のため : 続行



準リアルタイム LZSS 計算

目標 : $p = 4$ から始まる F の長さを計算する

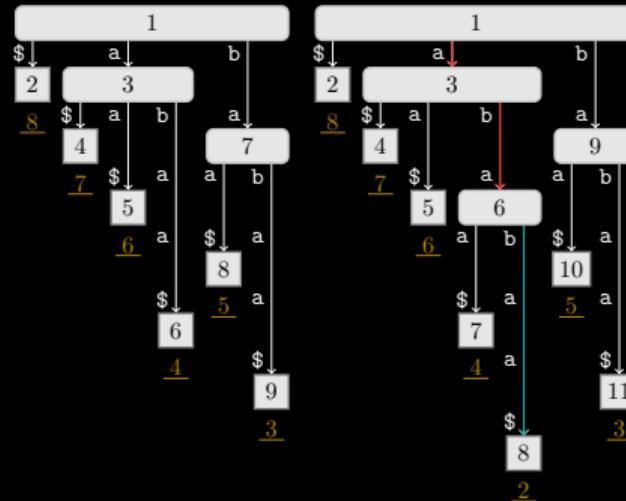
- $j \leftarrow 5$ を $\text{ST}(\overleftarrow{T[1..j]}) = \text{ST}(\text{babaa})$ に更新
- 挿入点の位置は $L = \text{ba}$
- $|L| \geq p + 1 - j = 2$ のため : 続行



準リアルタイム LZSS 計算

目標 : $p = 4$ から始まる F の長さを計算する

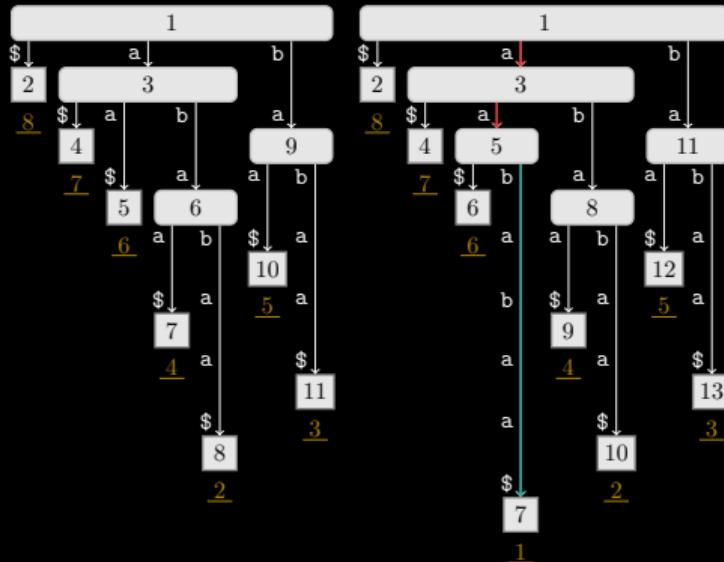
- $j \leftarrow 6$ を
 $\text{ST}(T[1..j]) = \text{ST}(\text{ababaa})$ に更新
- 挿入点の位置は $L = \text{aba}$
- $|L| \geq p + 1 - j = 3$ のため : 続行



準リアルタイム LZSS 計算

目標 : $p = 4$ から始まる F の長さを計算する

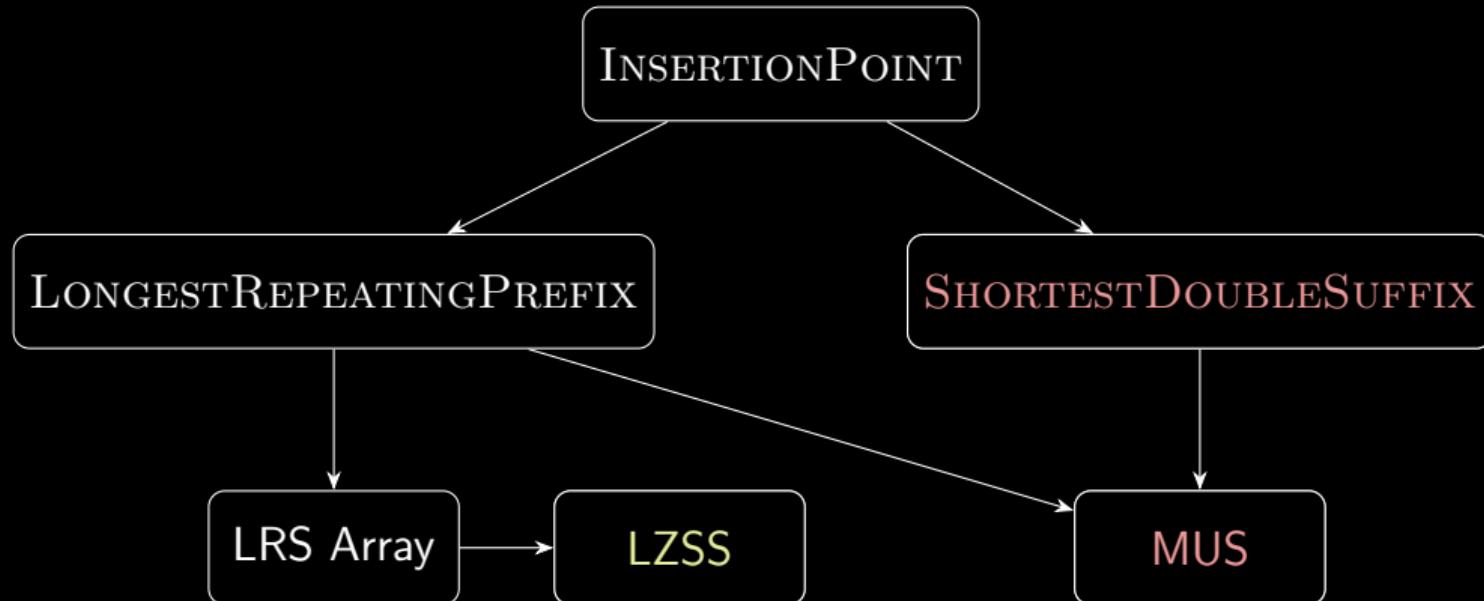
- $j \leftarrow 7$ を
 $\text{ST}(T[1..j]) = \text{ST}(\text{aababaa})$ に更新
- 挿入点の位置は $L = \text{aa}$
- $|L| < p + 1 - j = 4$ のため : 停止
- これで分かる : $|F| = p - j = 3$



LZSS計算（続き）

定理

T のLZSS分解を1文字あたり $O(t_{SU})$ 最悪時間でオンライン計算できる。



最小一意部分文字列 (MUS)

LONGEST REPEATING PREFIX を使って計算できる：

定義 (最小一意部分文字列 (MUS))

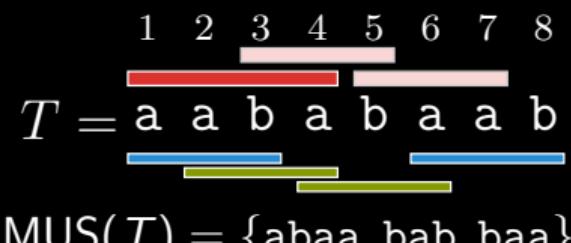
$T[1..n]$ の最小一意部分文字列は部分文字列 $T[\ell..r]$ であり、次を満たす

- $T[\ell..r]$ は T において正確に1回出現
- $T[\ell+1..r]$ と $T[\ell..r-1]$ は T で少なくとも2回出現 ($\ell < r$ の場合)

$MUS(T[1..j])$ を $T[1..j]$ のすべてのMUSの集合とする。

例 ($T[1..4] = \text{aab}$ はMUS)

- $T[1..4] = \text{aab}$ は1回出現
- $T[1..3] = \text{ab}$ は2回出現
- $T[2..4] = \text{ab}$ は2回出現



既存の研究

- MUS をオンラインで計算する既知の解法, 1文字あたりの時間

| 時間 | 参考文献 |
|--------------------|-----------|
| $O(\lg \sigma)$ 平均 | Mieno+'20 |
| $O(t_{SU})$ | この研究 |

拡張可能配列

n 個の要素を持つ整数配列 $A[1..n]$ であり, 以下をサポートする

- $A.grow()$: A のサイズを増加させる
- $A[j]$: エントリ $A[j]$ へのランダム読み書きアクセス

拡張可能配列 :

| 時間 | 参考文献 |
|-----------|-------------|
| $O(1)$ 平均 | 配列倍増 |
| $O(1)$ | Brodnik+'99 |

Mieno+'20のアプローチ

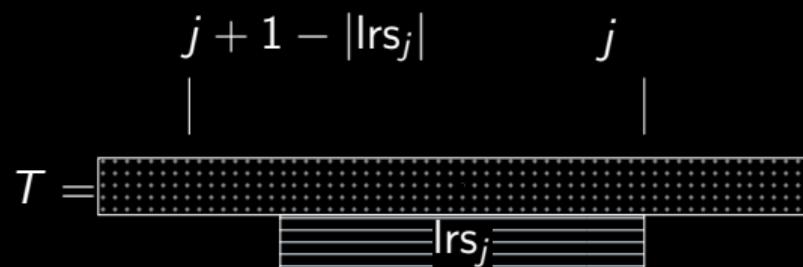
$\text{MUS}(T[1..j])$ は $\text{MUS}(T[1..j-1])$, lrs_j , sds_j から計算できることを示した

定義 (最短2重接尾辞 sds_j)

$T[1..j]$ において正確に2回出現する最短の接尾辞であり, 存在しない場合もある.

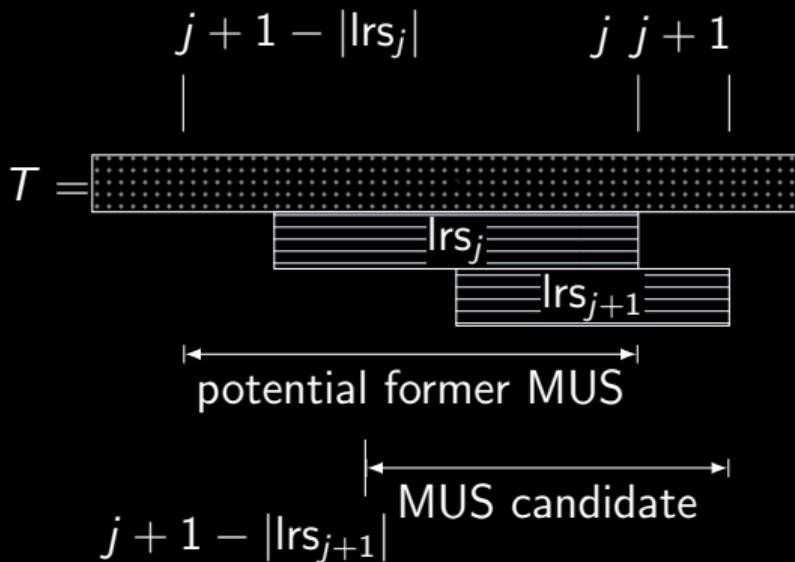
オンラインMUSアルゴリズム

- 新しい文字 $T[j + 1]$ を読み込む



オンラインMUSアルゴリズム

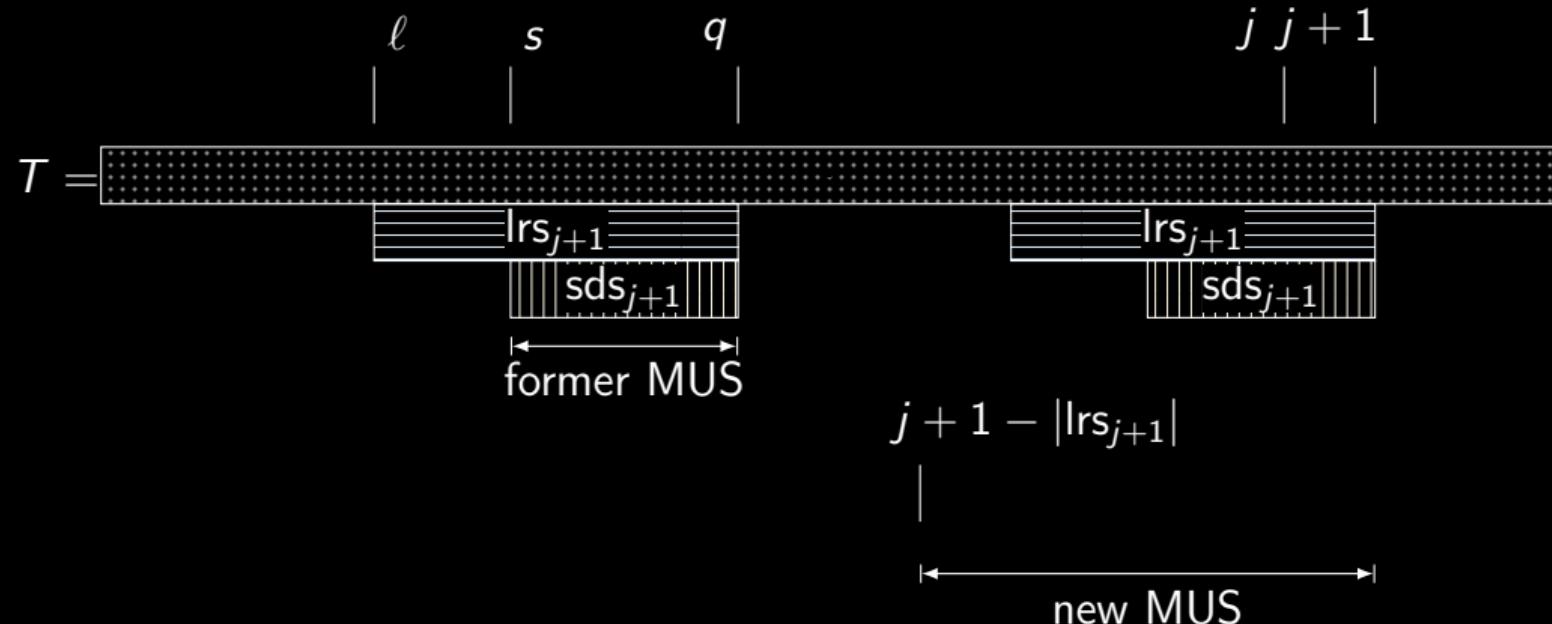
1. 新しい文字 $T[j + 1]$ を読み込む
2. $|lrs_{j+1}| \leq lrs_j|$ の場合、新しいMUS $[j + 1 - |lrs_{j+1}|..j + 1]$ を追加する
 - sds_{j+1} が存在しない場合、完了
 - これから： sds_{j+1} が存在すると仮定する



場合 : sds_{j+1} が存在する

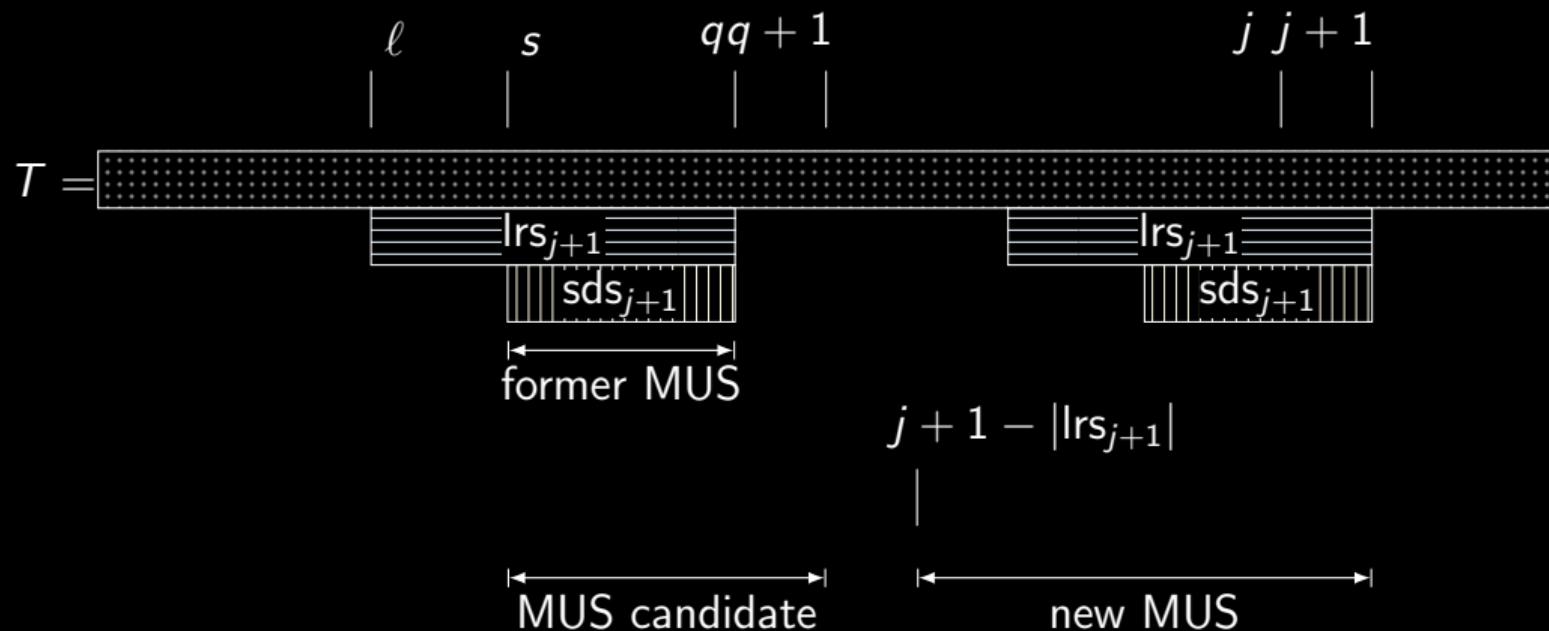
- sds_{j+1} の非接尾辞出現 $[s..q]$ を見つける :

- s : $ST(\overleftarrow{T[1..j]})$ で既に存在していた sds_{j+1} の位置の葉のラベル
- $q = s + |sds_{j+1}| - 1$



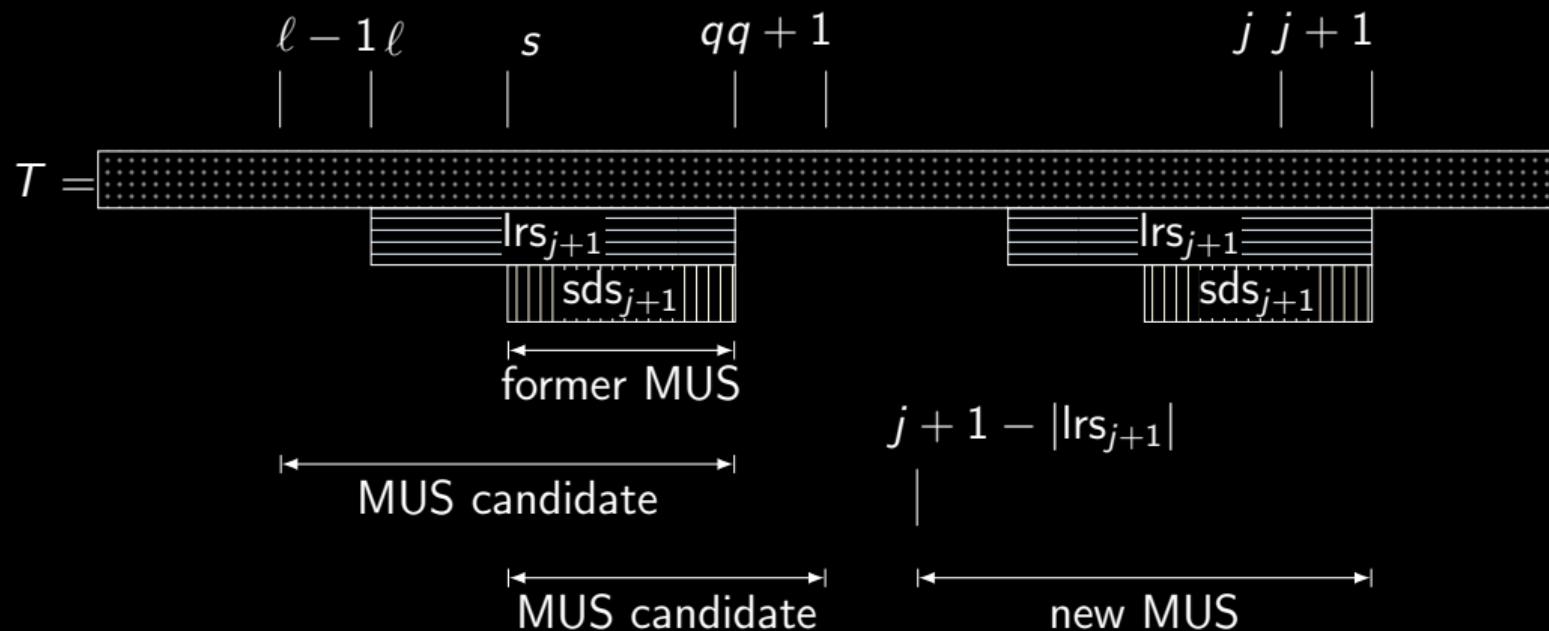
場合 : sds_{j+1} が存在する

- 以前のMUS $[s..q]$ を削除する
- 位置 $q + 1$ で終わるMUSが存在しない場合 : MUS $[s..q + 1]$ を追加する



場合 : sds_{j+1} が存在する

- 以前のMUS $[s..q]$ を削除する
- 位置 $q + 1$ で終わるMUSが存在しない場合 : MUS $[s..q + 1]$ を追加する
- $q - |\text{lrs}_{j+1}| \geq 1$ かつ位置 $q - |\text{lrs}_{j+1}|$ で始まるMUSが \emptyset の場合 : MUS $[q - |\text{lrs}_{j+1}|..q]$ を追加する



INSERTIONPOINT からの帰着

問題 (SHORTESTDOUBLE_SUFFIX sds_j)

入力: 接尾辞木 $ST(\overleftarrow{T[..j]})$ と文字 $T[j + 1]$.

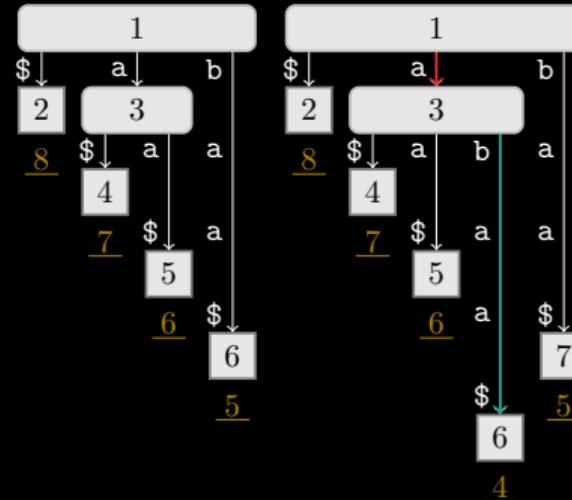
出力: $\overleftarrow{T[1..j]}$ において正確に1回の出現を持つ $\overleftarrow{T[..j + 1]}$ の最短接頭辞 (存在する場合).

SHORTESTDOUBLE_SUFFIX は INSERTIONPOINT を用いて解くことができる!

INSERTIONPOINT からの帰着

アルゴリズム

もし挿入点が $ST(T[j + 1..])$ で既に存在する頂点である場合：
LONGESTREPEATINGPREFIX は $T[j..]$ において少なくとも3回出現し、 sds_j は存在しない



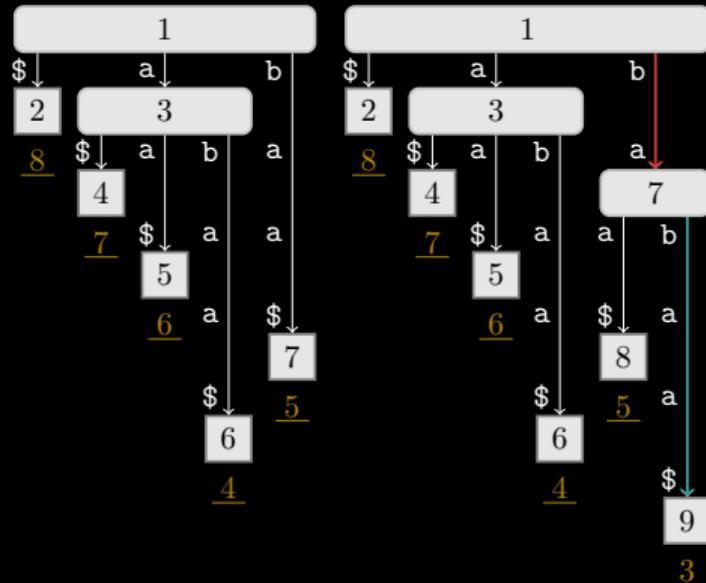
- $T = \text{aababaaa}$
- $ST(\text{baa})$ から $ST(\text{abaa})$ への更新

INSERTIONPOINT からの帰着

アルゴリズム

そうでない場合：

- LONGESTREPEATINGPREFIXは $ST(T[j..])$ において正確に2つの葉を持つ頂点になる
- sds_j が存在し、その位置はLONGESTREPEATINGPREFIXの位置への親辺上にある



- $T = \text{aababaa}$
- $ST(\text{abaa})$ から $ST(\text{babaa})$ への更新

結論

定理

$MUS(T[1..j])$ を1文字あたり $O(t_{SU})$ 最悪時間でオンライン計算できる.

まとめ

- リアルタイム ST 構築は未解決だが、準リアルタイムで可能
- 多くの応用 (LRS, LZSS, MUS, ...) はINSERTIONPOINTを用いて準リアルタイムで解ける
- 示していない：反転LZ分解の準リアルタイム計算

ご清聴ありがとうございました