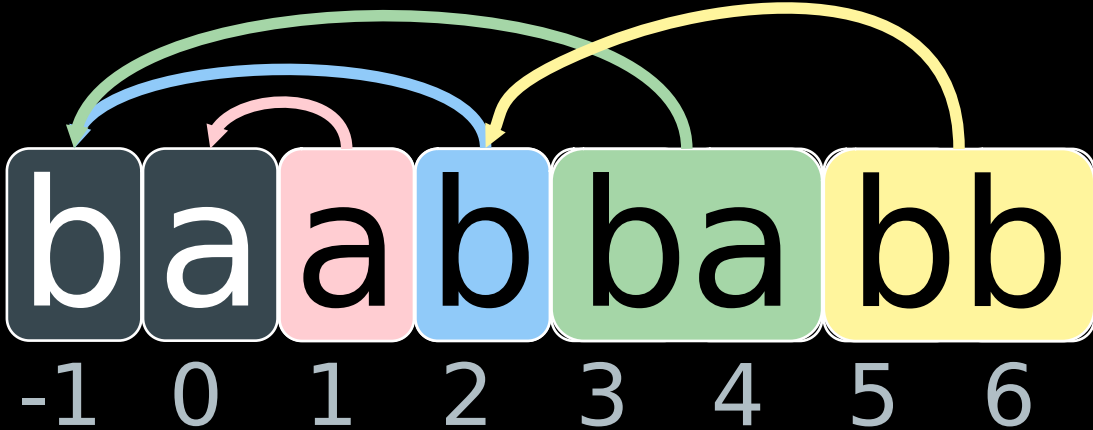


$T =$  $T =$ **b** **a** **a** **b** **b** **a** **b** **b**
-1 0 1 2 3 4 5 6

Lempel-Ziv 項の距離を 高次情報量で表現する符号

発表者
カップル ドミニク
東京医科歯科大学

共著
Gonzalo Navarro
University of Chile
Nicola Prezza
Ca' Foscari University

論文紹介
HOLZ: High-Order Entropy Encoding
of Lempel-Ziv Factor Distances,
Proc. DCC, pages 83-92, 2022

Lempel-Ziv 77 (LZ)

- 可逆圧縮の方法
- LZ の圧縮率は強い
 - gzip, 7z, zip で使われている
- LZ とは入力をテキストに扱い、左から右へ読み、
 - 読んだテキストを辞書に扱い、
 - 残ったテキストの接頭辞を辞書への参照で置換する

参照先の妥当性

参照を見つけないと、困る!

規定:

- テキストの前にすべて出現する文字を追加する
- 参照の長さは1文字以上になる

$T =$

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| b | a | a | b | b | a | b | b |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

LZ77 の実行

- LZ77 は参照として最長の候補を選択し、
- テキストを項 $T = F_1 \cdots F_z$ に分解する

ただし、 F_x は j から始まるとすると、 $\pi[-1..j-1]$ の範囲から始まり接頭辞の最長の共通接頭辞である

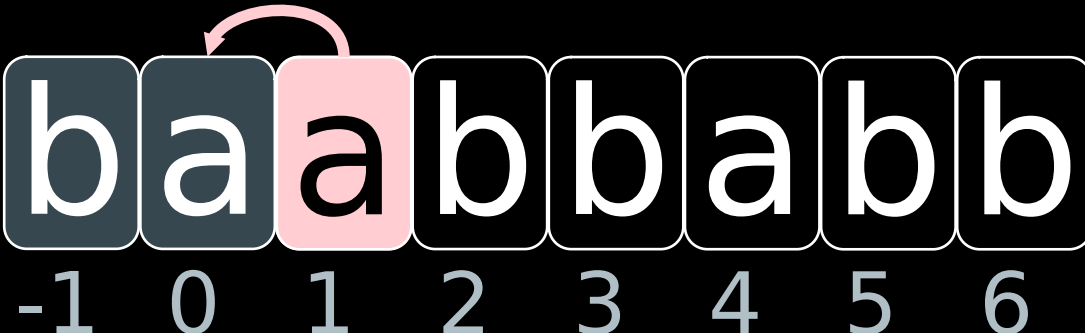
$T =$

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| b | a | a | b | b | a | b | b |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

LZ77 の実行

- LZ77 は参照として最長の候補を選択し、
- テキストを項 $T = F_1 \cdots F_z$ に分解する

ただし、 F_x は j から始まるとすると、 $\pi[-1..j-1]$ の範囲から始まり接頭辞の最長の共通接頭辞である

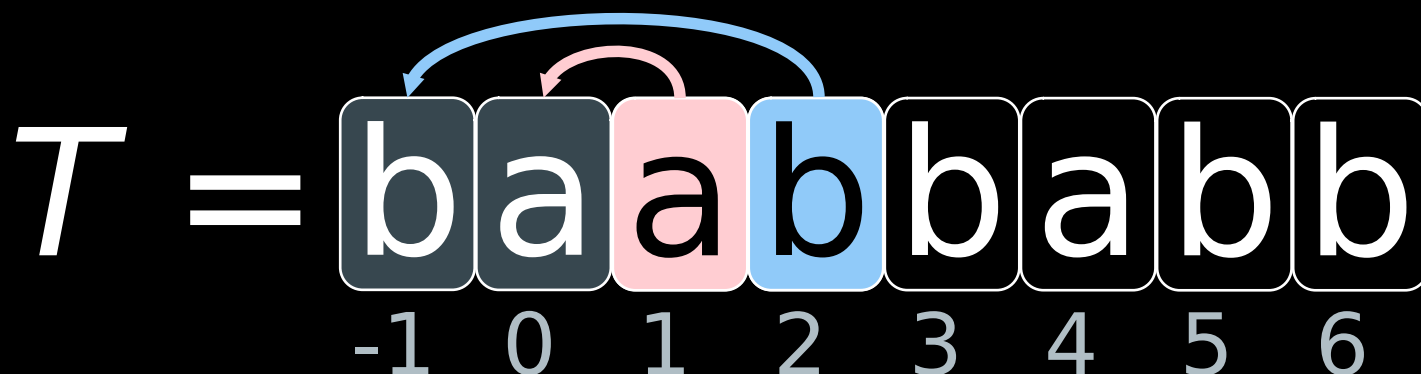
$T =$  $-1 \ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6$

The diagram shows the string $T = \text{baababb}$ with characters in boxes. The first 'a' at index 0 and the second 'a' at index 1 are highlighted in pink. A pink arrow points from the second 'a' back to the first 'a', indicating a backreference. Below the boxes are indices from -1 to 6.

LZ77 の実行

- LZ77 は参照として最長の候補を選択し、
- テキストを項 $T = F_1 \cdots F_z$ に分解する

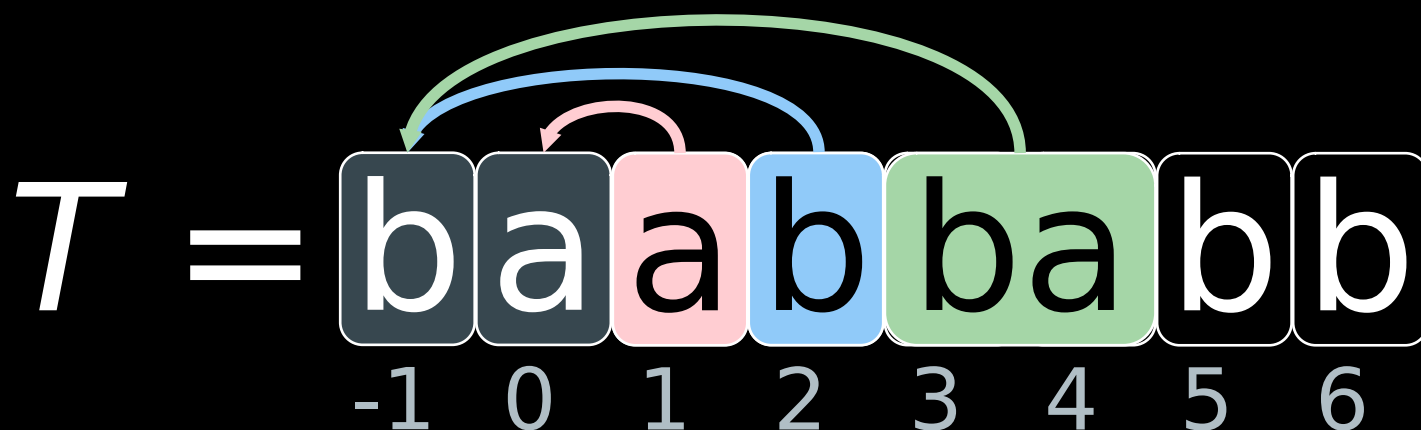
ただし、 F_x は j から始まるとすると、 $\pi[-1..j-1]$ の範囲から始まり接頭辞の最長の共通接頭辞である



LZ77 の実行

- LZ77 は参照として最長の候補を選択し、
- テキストを項 $T = F_1 \cdots F_z$ に分解する

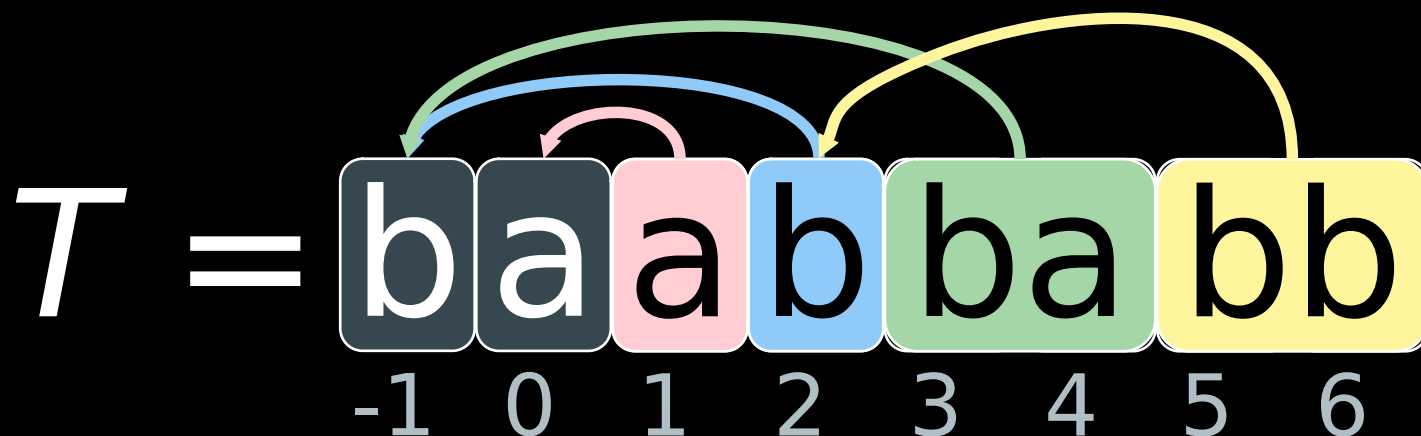
ただし、 F_x は j から始まるとすると、 $\pi[-1..j-1]$ の範囲から始まり接頭辞の最長の共通接頭辞である



LZ77 の実行

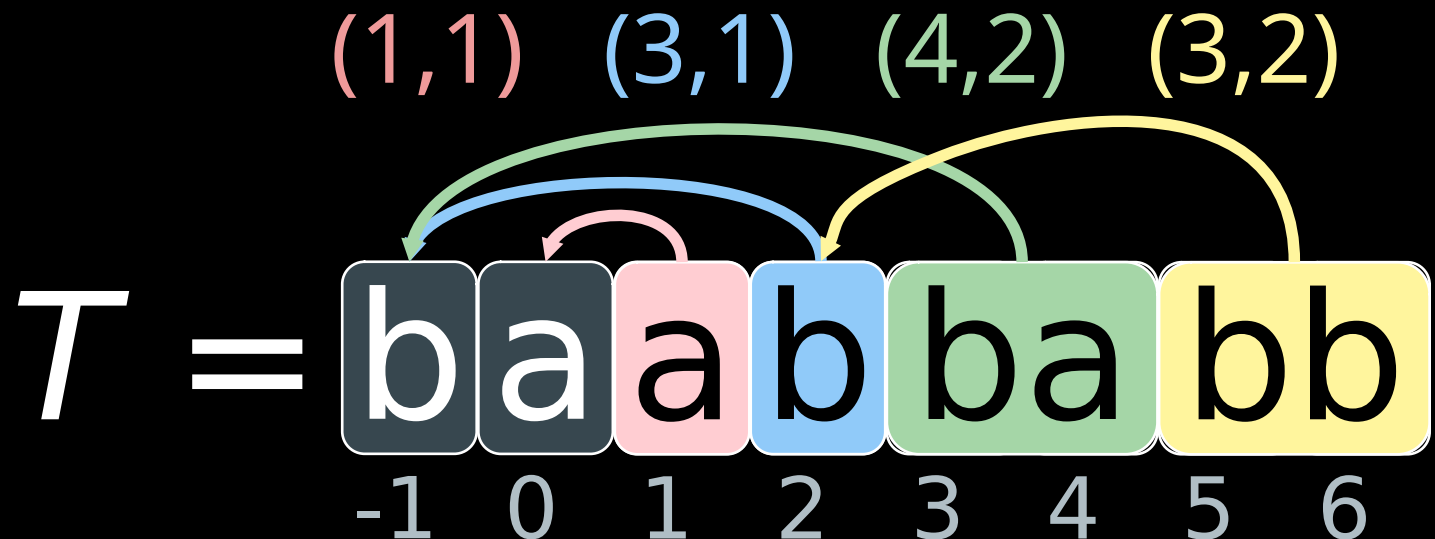
- LZ77 は参照として最長の候補を選択し、
- テキストを項 $T = F_1 \cdots F_z$ に分解する

ただし、 F_x は j から始まるとすると、 $\pi[-1..j-1]$ の範囲から始まり接頭辞の最長の共通接頭辞である



2つ組

- 各項は (参照先の距離、長さ) という2つ組で表現できる
- 圧縮を与えるため、2つ組の列を Elias γ code と いった universal code で符号化する



復元

- 参照先は読んだテキストを指す
⇒ 復元できる
- 問題: 距離は圧縮しにくい

(1,1) (3,1) (4,2) (3,2)

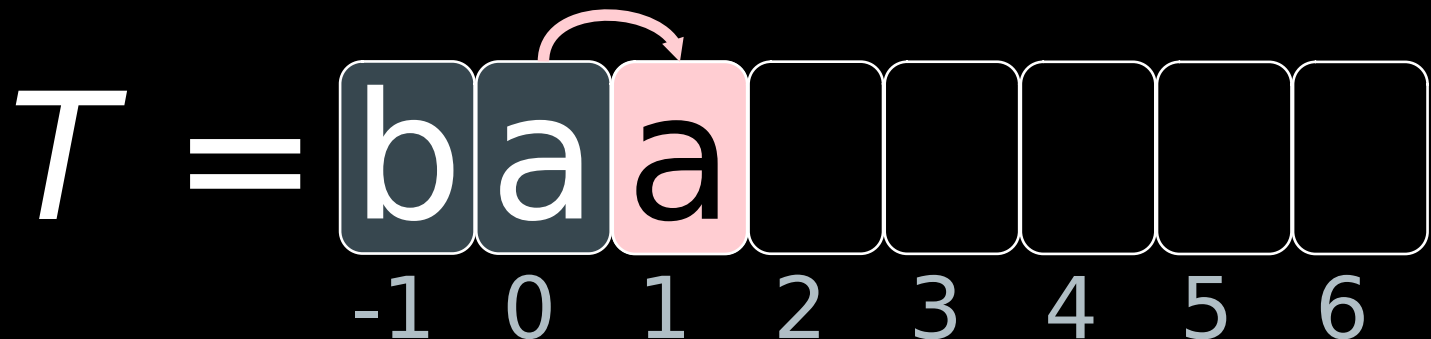
$T =$

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| b | a | | | | | | |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

復元

- 参照先は読んだテキストを指す
⇒ 復元できる
- 問題: 距離は圧縮しにくい

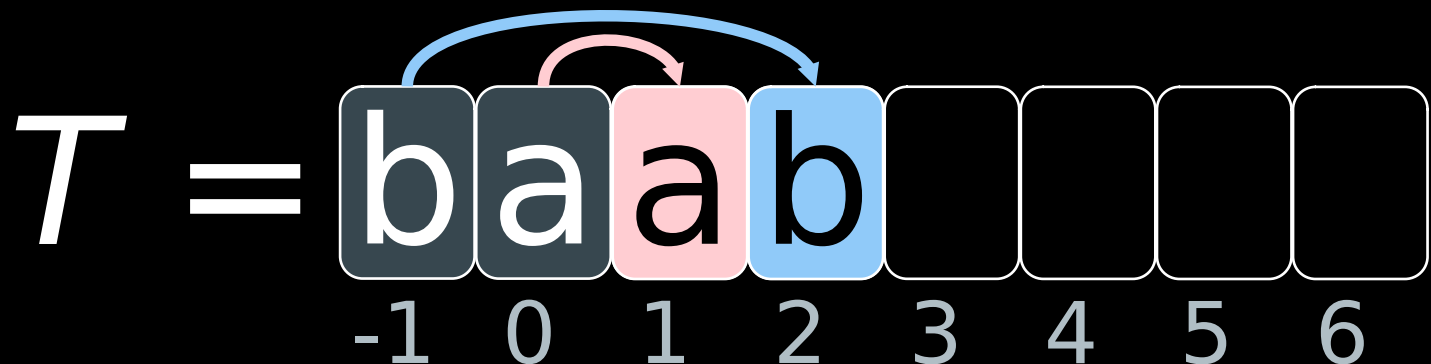
(1,1) (3,1) (4,2) (3,2)



復元

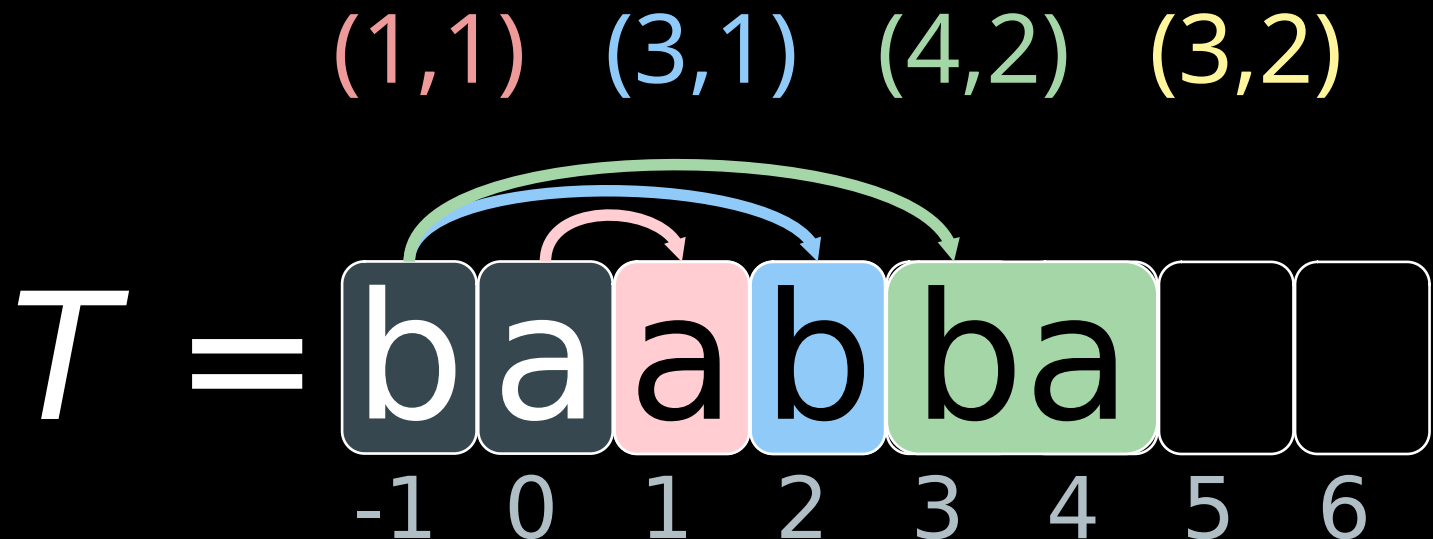
- 参照先は読んだテキストを指す
⇒ 復元できる
- 問題: 距離は圧縮しにくい

(1,1) (3,1) (4,2) (3,2)



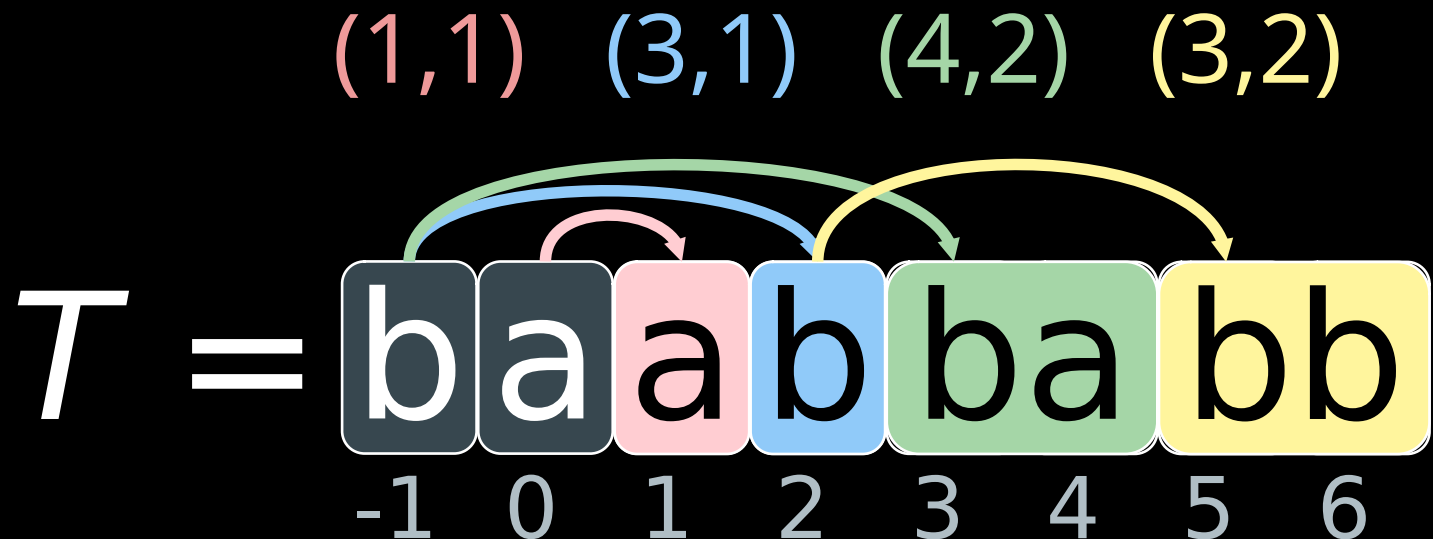
復元

- 参照先は読んだテキストを指す
⇒ 復元できる
- 問題: 距離は圧縮しにくい



復元

- 参照先は読んだテキストを指す
⇒ 復元できる
- 問題: 距離は圧縮しにくい



距離の表現

他の距離の表現を提案する:

- 項の開始位置と長さがすべて決定した後、
- 各項の参照先を読んだテキストの接頭辞にもとづいて距離を計算する
- その距離は接頭辞が colex 順序にソートされた順序で定まる
- **holz** 距離と呼ぶ (**h**igh **o**rd**er** **L**empel-**Z**iv)

colex 順序

= 逆文字列の辞書式順序に従ってソートする

例

aaab

abaa

aaba

aaba

abaa

bbba

bbba

aaab



辞書順序



colex 順序

他の用語

- $\pi[i..j]$: 部分文字列; 例: $\pi[0..2] = aab$
- $\pi[i..]$: 接尾辞; 例: $\pi[3..] = babb$
- $\pi[-1..j]$: 接頭辞
- ε : 空文字列 (長さ 0)

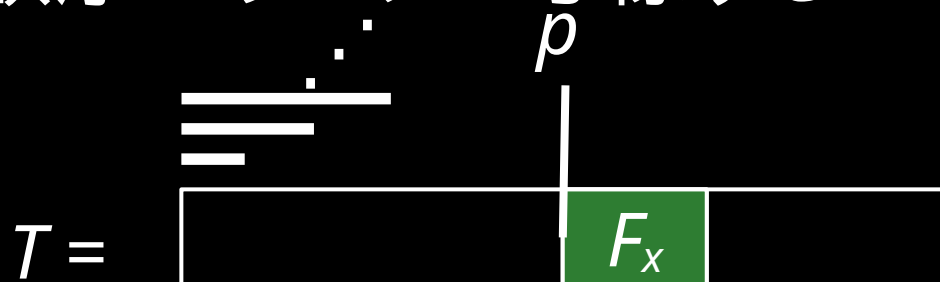
アルファベットはバイナリーだから、 T は -1 から始まる

$T =$

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| b | a | a | b | b | a | b | b |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

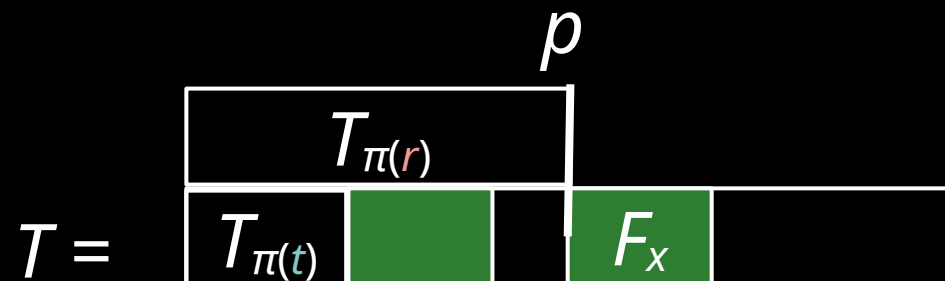
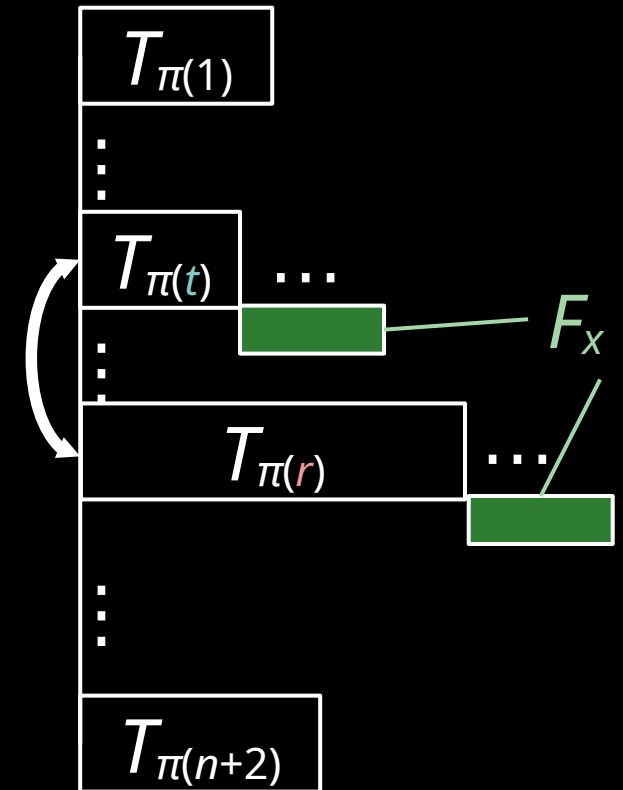
距離の計算：概念

- $T_p := \Pi[-1..p]$ (T の接頭辞)
- $T_{-2} = \varepsilon, T_{-1} = b, T_0 = ba, T_1 = baa \dots$
- $\Pi[p..]$ から始まる項 F_x の距離を計算したいとき：
- T_{-2}, \dots, T_{p-1} を colex 順序にソートする
- $T_{\pi(1)} < \dots < T_{\pi(p+2)}$ という順番を求める
- 順列 π は colex 順序のランクへ写像する



距離の計算

- $T_{\pi(1)} < \dots < T_{\pi(p+2)}$
- r は $\pi(r) = p-1$ を満たすランクを示す
- t は r に一番近い「 F_x が $\pi[\pi(t)+1..]$ の接頭辞」と満たすランク
- **holz** 距離は $r - t$ となる

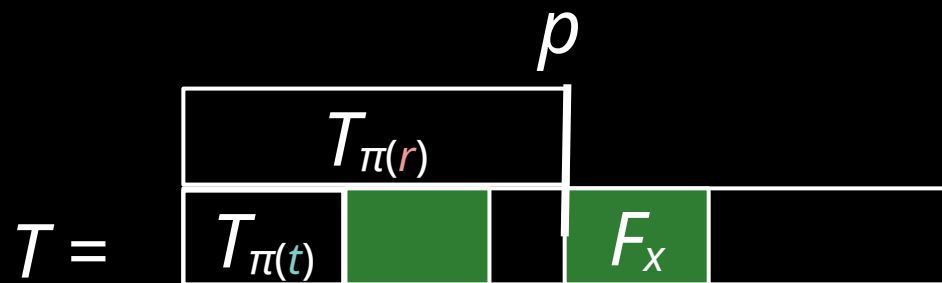
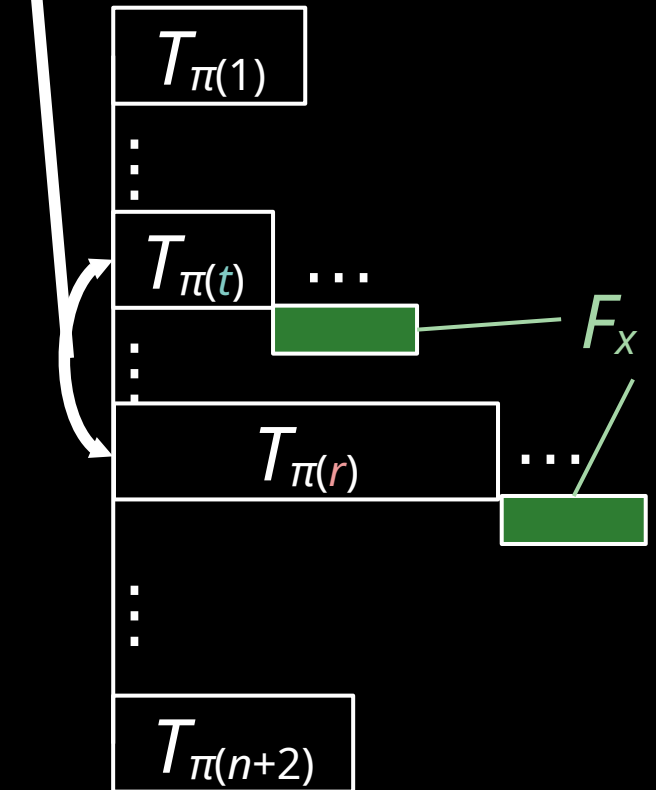


距離の計算

- $T_{\pi(1)} < \dots < T_{\pi(p+2)}$
- r は $\pi(r) = p-1$ を満たすランクを示す
- t は r に一番近い「 F_x が $\pi[\pi(t)+1..]$ の接頭辞」と満たすランク
- **holz** 距離は $r - t$ となる

F_x の
距離

$$= r - t$$



距離: 具体的な例

前処理

- $T_{-2} = \varepsilon$
- $T_{-1} = b$
- $T_0 = ba$

を colex 順序にソートする



$T =$

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| b | a | a | b | b | a | b | b |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

距離: 具体的な例

前処理

- $T_{-2} = \varepsilon$
- $T_{-1} = b$
- $T_0 = ba$

| | |
|---|--------------|
| 1 | $T_{-2} =$ |
| 2 | $T_0 = ba$ |
| 3 | $T_{-1} = b$ |

を colex 順序にソートする



$T =$

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| b | a | a | b | b | a | b | b |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

距離: 具体的な例

前処理

- $T_{-2} = \varepsilon$
- $T_{-1} = b$
- $T_0 = ba$

| | | |
|---|--------------|----------|
| | | 残った接尾辞 |
| 1 | $T_{-2} =$ | baabbabb |
| 2 | $T_0 = ba$ | abbabb |
| 3 | $T_{-1} = b$ | aabbabb |

を colex 順序にソートする

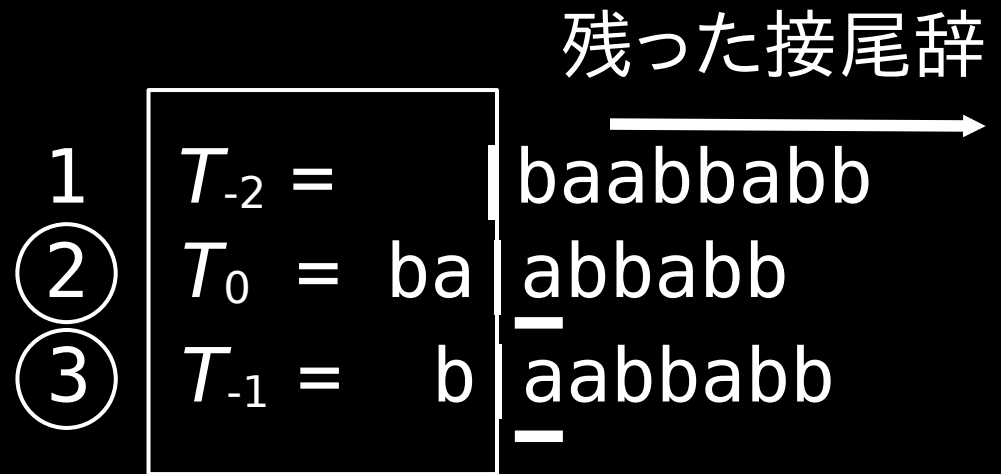


$T =$ **b** **a** **a** **b** **ba** **bb**

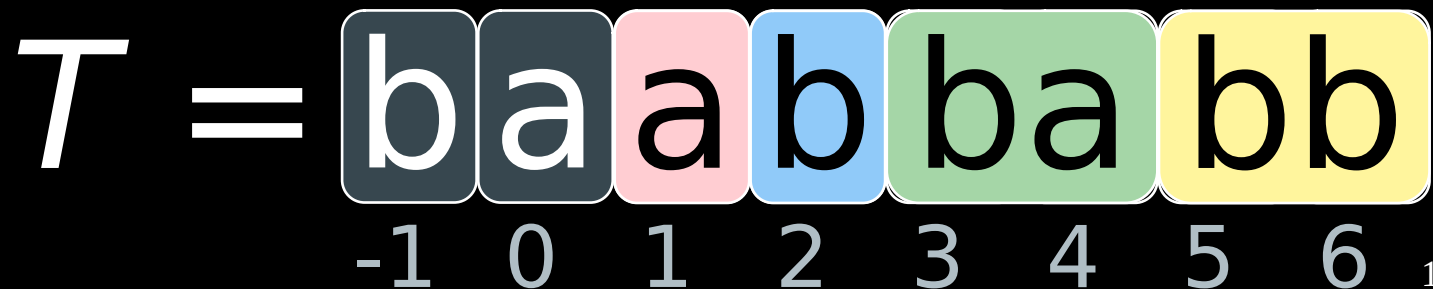
-1 0 1 2 3 4 5 6

距離: 具体的な例

- $F_1 = \pi[1]$ は最初の項
- F_1 の開始位置: $p=1$
- $T_{p-1} = T_0$ のランクは $r=2$
- T_0 の残った接尾辞の接頭辞は F_1
- T_{-1} のランクは $t=3$



$$r - t = 2 - 3 = -1$$



距離: 具体的な例

- T_1 を追加する
- $p = 2$
- $r = 2$
- $t = 1$
- $r - t = 1$

| | | |
|---|--------------|------------------|
| | | 残った接尾辞 |
| ① | $T_{-2} =$ | <u>b</u> aabbabb |
| ② | $T_1 =$ baa | <u>b</u> babb |
| 3 | $T_0 =$ ba | abbabb |
| 4 | $T_{-1} =$ b | aabbabb |



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

-1 0 1 2 3 4 5 6

距離: 具体的な例

- T_2 を追加する
- $p = 3$
- $r = 5$
- $t = 1$
- $r - t = 4$

| | | 残った接尾辞 |
|---|--------------|------------------|
| ① | $T_{-2} =$ | <u>ba</u> abbabb |
| 2 | $T_1 =$ baa | bbabb |
| 3 | $T_0 =$ ba | abbabb |
| 4 | $T_{-1} =$ b | aabbabb |
| ⑤ | $T_2 =$ baab | <u>ba</u> bb |

$T =$ **b** **a** **a** **b** **ba** **bb**

-1 0 1 2 3 4 5 6

距離: 具体的な例

- T_3 と T_4 を追加する
- $p = 5$
- $r = 4$
- $t = 2$
- $r - t = 2$

| | | |
|---|----------------|---------------|
| 1 | $T_{-2} =$ | baabbabb |
| 2 | $T_1 =$ baa | <u>bb</u> abb |
| 3 | $T_0 =$ ba | abbabb |
| 4 | $T_4 =$ baabba | <u>bb</u> |
| 5 | $T_{-1} =$ b | aabbabb |
| 6 | $T_2 =$ baab | babb |
| 7 | $T_3 =$ baabb | abb |



$T =$ **b** **a** **a** **b** **ba** **bb**

-1 0 1 2 3 4 5 6

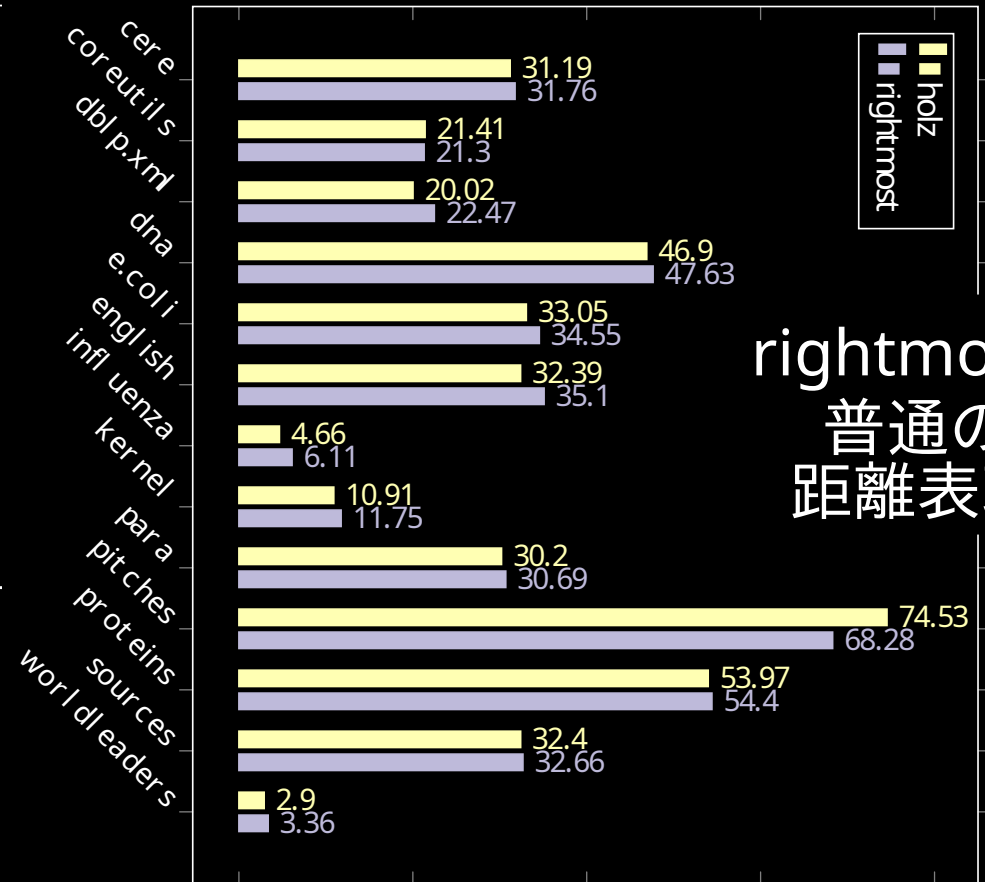
実験

- データセット: Pizza & Chili corpus
- 各データセットの 20 MB 接頭辞を LZ 分解し、
- 2つ組の列の Elias γ 符号化で圧縮し、
- 圧縮率を比較した

実験

| dataset | σ | z [K] | H_0 | H_2 | H_4 |
|--------------|----------|---------|-------|-------|-------|
| cere | 5 | 8492 | 2.20 | 1.79 | 1.78 |
| coreutils | 235 | 3010 | 5.45 | 2.84 | 1.31 |
| dblp.xml | 96 | 3042 | 5.22 | 1.94 | 0.89 |
| dna | 14 | 12706 | 1.98 | 1.92 | 1.91 |
| e.coli | 11 | 8834 | 1.99 | 1.96 | 1.94 |
| english | 143 | 5478 | 4.53 | 2.89 | 1.94 |
| influenza | 15 | 876 | 1.97 | 1.93 | 1.91 |
| kernel | 160 | 1667 | 5.38 | 2.87 | 1.47 |
| para | 5 | 8254 | 2.17 | 1.83 | 1.82 |
| pitches | 129 | 10407 | 5.62 | 4.28 | 2.18 |
| proteins | 25 | 8499 | 4.20 | 4.07 | 2.97 |
| sources | 111 | 4878 | 5.52 | 2.98 | 1.60 |
| worldleaders | 89 | 408 | 4.09 | 1.74 | 0.73 |

圧縮率 (低い=良い)



rightmost:
普通の
距離表現

符号化方法: Elias γ

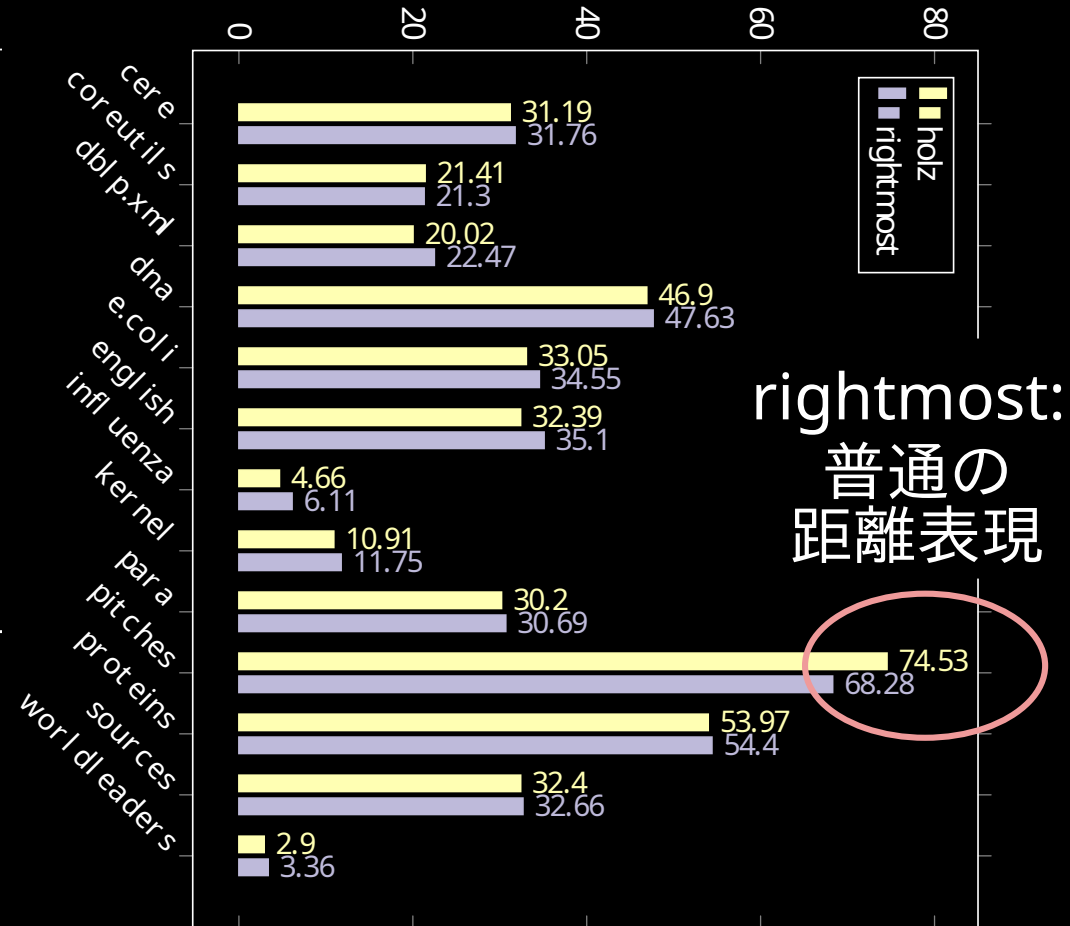
- z : 項の個数
- [K]: 10^3 (kilo)
- σ : アルファベットサイズ
- H_k : k 番目の経験的エントロピー

実験

| dataset | σ | z [K] | H_0 | H_2 | H_4 |
|--------------|----------|-------|-------|-------|-------|
| cere | 5 | 8492 | 2.20 | 1.79 | 1.78 |
| coreutils | 235 | 3010 | 5.45 | 2.84 | 1.31 |
| dblp.xml | 96 | 3042 | 5.22 | 1.94 | 0.89 |
| dna | 14 | 12706 | 1.98 | 1.92 | 1.91 |
| e.coli | 11 | 8834 | 1.99 | 1.96 | 1.94 |
| english | 143 | 5478 | 4.53 | 2.89 | 1.94 |
| influenza | 15 | 876 | 1.97 | 1.93 | 1.91 |
| kernel | 160 | 1667 | 5.38 | 2.87 | 1.47 |
| para | 5 | 8254 | 2.17 | 1.83 | 1.82 |
| pitches | 129 | 10407 | 5.62 | 4.28 | 2.18 |
| proteins | 25 | 8499 | 4.20 | 4.07 | 2.97 |
| sources | 111 | 4878 | 5.52 | 2.98 | 1.60 |
| worldleaders | 89 | 408 | 4.09 | 1.74 | 0.73 |

- z: 項の個数
- [K]: 10^3 (kilo)
- σ : アルファベットサイズ
- H_k : k 番目の経験的エントロピー

圧縮率 (低い=良い)



符号化方法: Elias γ

圧縮率の原因

なぜ **holz** 距離の値は期待的により低い？

概念の理由：

- 読んだ接頭辞を `colex` 順序にスートするのは参照先の直前の部分文字列に類似すれば、類似するほど距離の値は低い
- 逆 Burrows-Wheeler transform (BWT) のような扱い方
- 符号化 BWT は k 番目の経験的エントロピーを与える

計算量

問題: どう colex 順序でソートされている接頭辞を確保できる?

答え: dynamic BWT で:

- dynamic BWT で逆テキストを索引し、
読んだテキストの接頭辞を colex 順序で格納できる
- $n H_k + o(n \lg \sigma)$ 領域
- $O(n \lg n / \lg \lg n)$ 時間

H_k : k 番目の経験的エントロピー

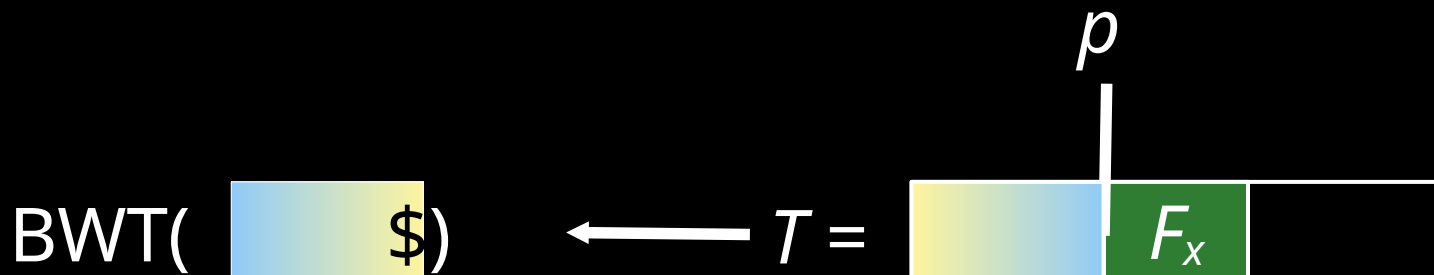
BWT での計算

1) $\pi[-1..n] = \text{baabbabb}$

2) $T^R\$ = \text{bbabbaab\$}$

3) 前処理 : $\text{BWT}(\text{ab\$})$ を格納

4) $\pi[p..]$ から始める項を計算するとき、 $\text{BWT}(T^R[n-p+1..n+2])$ を格納する



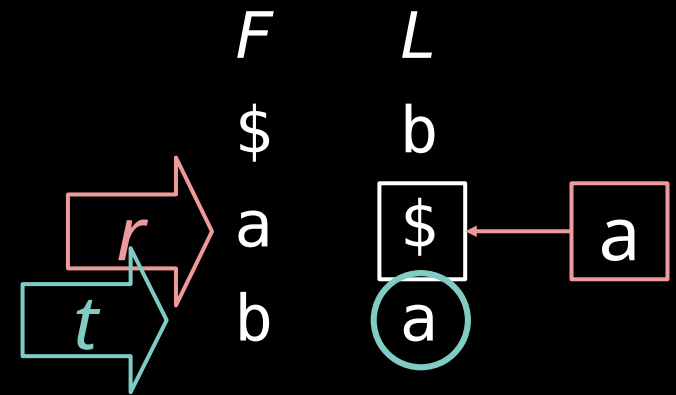
F_1 の距離の計算

BWT(ab\$)

| F | L |
|-----|-----|
| \$ | b |
| a | \$ |
| b | a |

r : \$ の場所

t : 参照先



文字の挿入

BWT(ab\$)

| F | L |
|-----|-----|
| \$ | b |
| a | \$ |
| b | a |

\$

a

BWT(aab\$)

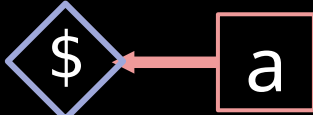
| F | L |
|-----|-----|
| \$ | b |
| a | \$ |
| a | a |
| b | a |

- 1) $L[i] = \$$ を挿入したい文字 c に置換する
- 2) 置換した後、 $L[i]$ は j 番目 c になると、
 $L[k]$ で $\$$ を挿入する
ただし、 $F[k]$ は F の j 番目の c

文字の挿入

BWT(ab\$)

| | |
|----------|----------|
| <i>F</i> | <i>L</i> |
| \$ | b |
| a | \$ |
| b | a |

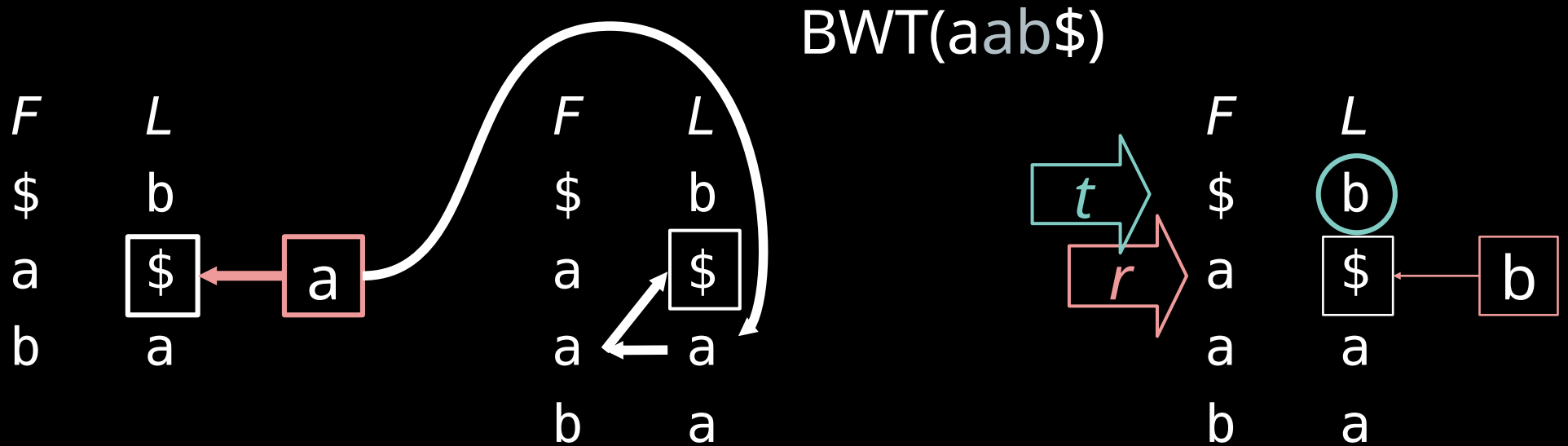


BWT(aab\$)

| | |
|----------|----------|
| <i>F</i> | <i>L</i> |
| \$ | b |
| a | \$ |
| a | a |
| b | a |

- 1) $L[i] = \$$ を挿入したい文字 c に置換する
- 2) 置換した後、 $L[i]$ は j 番目 c になると、
 $L[k]$ で $\$$ を挿入する
ただし、 $F[k]$ は F の j 番目の c

F_2 の距離の計算

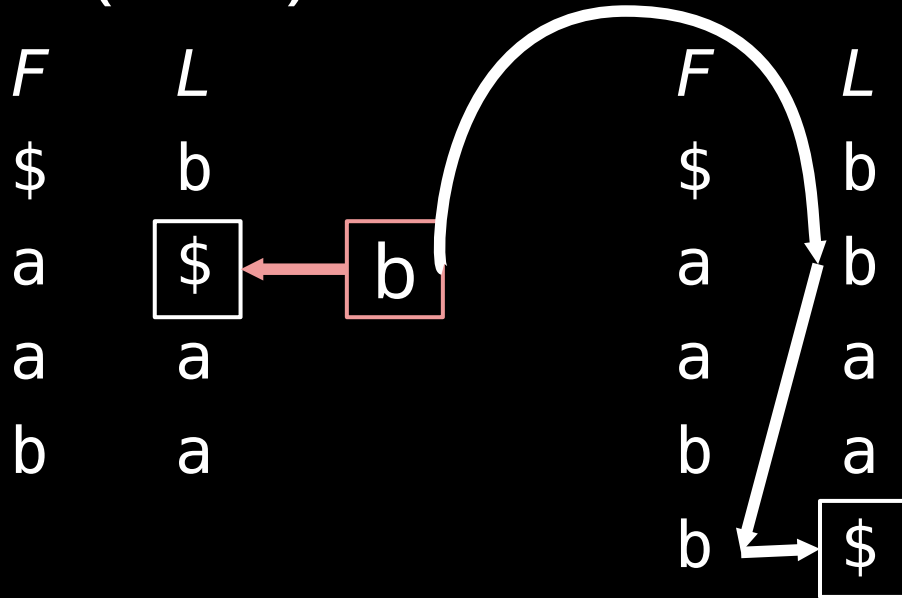


$$T = \text{b a a b b a b b}$$

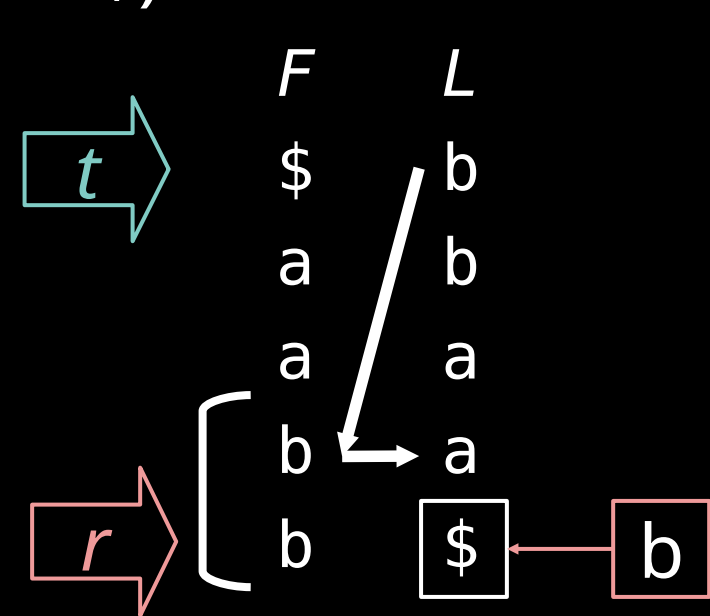
-1 0 1 2 3 4 5 6

F_3 の距離の計算

BWT(aab\$)



BWT(baab\$)

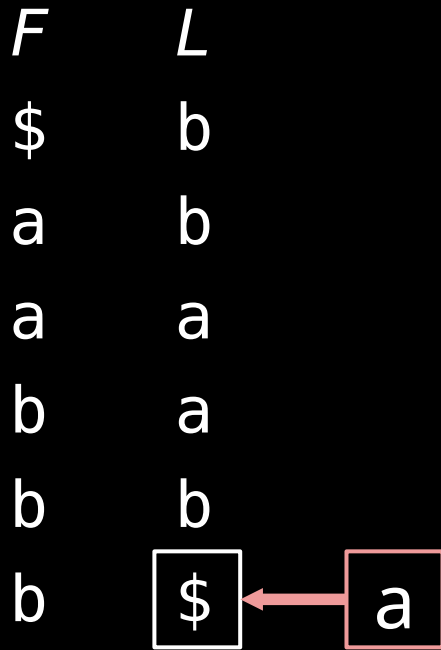


$T =$ b a a b b a b b

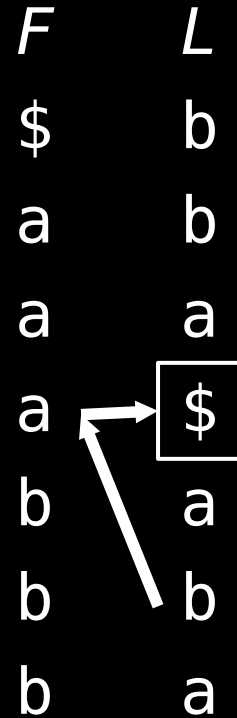
-1 0 1 2 3 4 5 6

F_4 の距離の計算

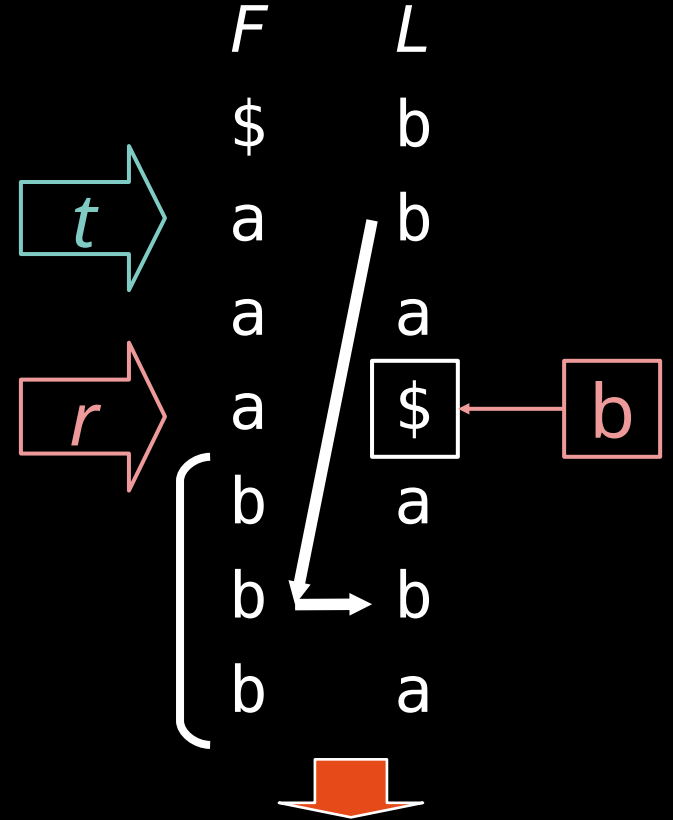
BWT(abaab\$)



BWT(babaab\$)



BWT(baab\$)



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

-1 0 1 2 3 4 5 6

まとめ

- LZ77 の分解を2つ組 (距離、長さ) の列で表現できる
 - 距離は一番圧縮しにくい
 - **holz** 距離を提案した:
読んだ接頭辞を colex 順序で格納しながら、並んだ接頭辞の範囲で距離を表現する
 - 実験によると、低いエントロピーで **holz** 距離のほうが役立つ
- 今度の研究
- dynamic BWT なら、速い実装がない
 - 原因: 存在の実装は pointer データ構造を使う