2024年度夏のLAシンポジウム

CDAWGによる LZ78部分文字列圧縮

柴田 紘希（九州大学）

ドミニク クップル（山梨大学）

本研究の概要

研究でやったこと

CDAWGを用いて、LZ78分解の部分文字列圧縮を省領域で高速に行う手法を提案した

- CDAWG: 文字列の省領域索引
- LZ78分解: 文字列の圧縮表現のひとつ
- 部分文字列圧縮:事前にテキスト文字列が与えられ、部分文字列の圧縮結果を返すクエリを高速に処理する問題

準備: LZ78分解と部分文字列圧縮問題

LZ78分解 [Lempel & Ziv, '78]

- 文字列の圧縮手法の一つ
- 文字列 T を特定のアルゴリズムにしたがって

  $T = F_0 F_1 \dots F_f$ に分解する（ただし、F_0 は空文字列）

$$T = \text{abbabaaaab} = F_0 F_1 \dots F_f$$

$$F = (\varepsilon, \text{a}, \text{b}, \text{ba}, \text{baa}, \text{ab})$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T = \mathrm{abbabaaab} = F_0 F_1 \ldots F_f \; (F_0 = \varepsilon)$$

$$T' = \mathrm{abbabaaab}$$

$$F = (\varepsilon)$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T' = \text{abbabaaab}$$
$$F = (\varepsilon, \text{a})$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}| .. n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T' = \text{abbabaaab}$$
$$F = (\varepsilon, \text{a}, \text{b})$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T' = \text{abbabaaab}$$
$$F = (\varepsilon, \text{a}, \text{b}, \text{ba})$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T' = \text{abbabaaab}$$
$$F = (\varepsilon, \text{a}, \text{b}, \text{ba}, \text{baa})$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T' = \text{abbabaaab}$$
$$F = (\varepsilon, \text{a}, \text{b}, \text{ba}, \text{baa}, \text{ab})$$

LZ78分解 [Lempel & Ziv, '78]

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  - $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ とし、T' の接頭辞となる最長の F_j $(j < i)$ を求める
  - $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ とする

$$T' = \mathrm{abbabaaab}$$
$$F = (\varepsilon, \mathrm{a}, \mathrm{b}, \mathrm{ba}, \mathrm{baa}, \mathrm{ab})$$

※もし $T' = F_j$ となったら $F_i = F_j$ とする

LZ78分解 [Lempel & Ziv, '78]

- Factor F_i は整数 j_i と文字c_iを用いて (j_i, c_i) と表現できる
  - (j_i, c_i) の列 F' から元の文字列 T を復元できるため、この列が圧縮表現になる！

$$T = \mathrm{abbabaaab}$$
$$F = (\varepsilon, \mathrm{a}, \mathrm{b}, \mathrm{ba}, \mathrm{baa}, \mathrm{ab})$$
$$F' = ((0, \mathrm{a}), (0, \mathrm{b}), (2, \mathrm{a}), (3, \mathrm{a}), (\mathrm{a}, \mathrm{b}))$$

部分文字列圧縮問題

- 長さ n のテキスト T が事前に与えられる
  - T についての索引構造を事前に構築しておいてよい
- 以下のクエリを処理:
  - 入力: 整数 l, r $(1 \leq l \leq r \leq n)$
  - 出力: $T[l..r]$ の圧縮表現

$T = \text{abbabaaab}, (l, r) = (2, 7)$

→ $T[l..r]$ のLZ78分解は $(\text{b}, \text{ba}, \text{baa})$

LZ78分解の部分文字列圧縮

LZ78分解の部分文字列圧縮 [Köppl, '21]

LZ78分解の部分文字列圧縮は、1クエリあたり

$O(z_{l,r})$ time・ $O(n)$ space

で解くことができる

※ $n = |T|$ で、 $z_{l,r}$ は $T[l..r]$ のLZ78分解のfactor数

- 接尾辞木（Suffix Tree）を用いた手法

本研究はこの手法の省領域化

※ワードサイズ $\Omega(n)$ のword-RAM modelを仮定、空間計算量はワード数で表記

発表の流れ

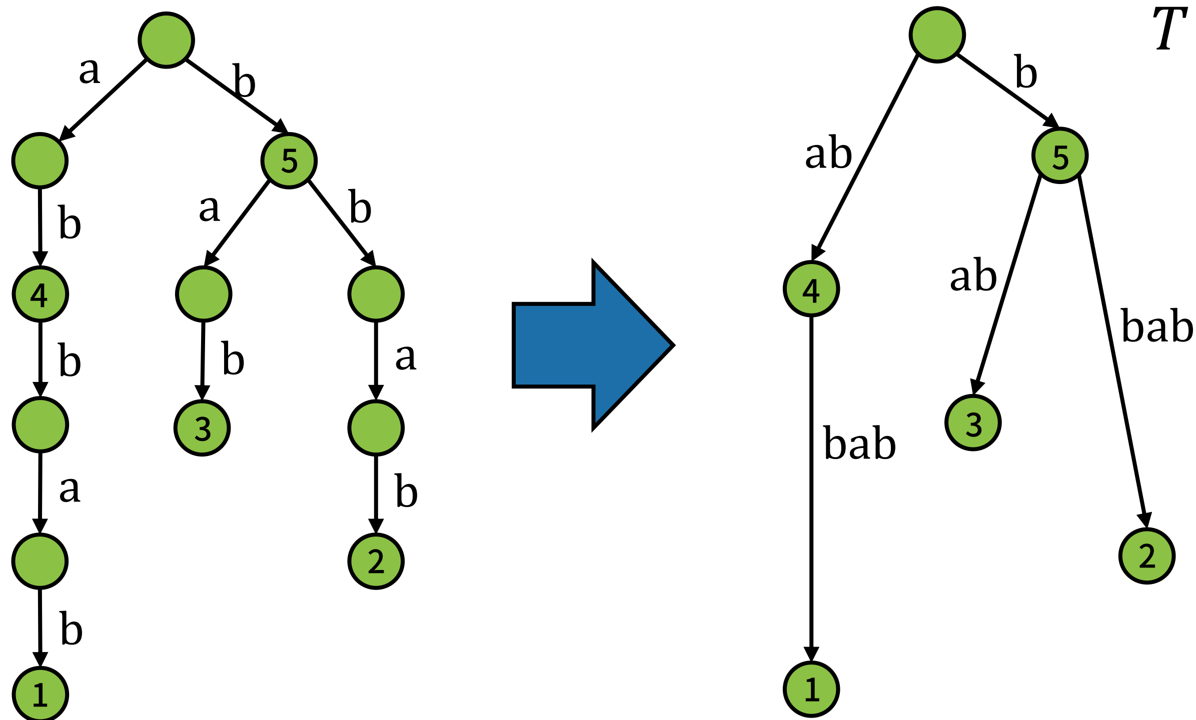
1. 接尾辞木によるLZ78分解の計算
2. 1. の部分文字列圧縮版
3. CDAWGによるLZ78分解の計算
4. 3. の部分文字列圧縮版

既存手法: 接尾辞木を使ったLZ78分解

+ 部分文字列圧縮

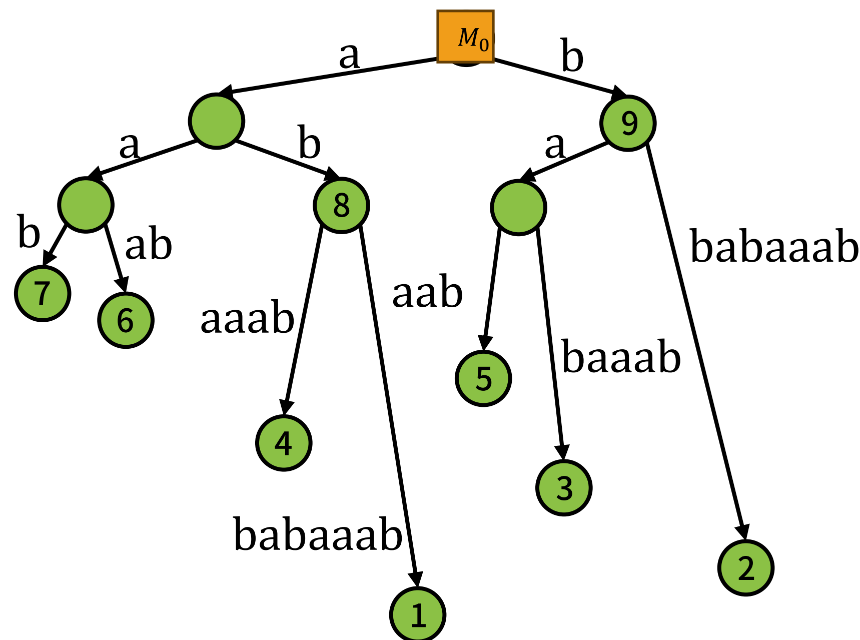
接尾辞木 (Suffix Tree)

文字列の接尾辞集合を表すトライ木から、接尾辞を表さない出次数1のノードをひとまとめにすることで得られる辺ラベル付き木



接尾辞木によるLZ78分解の計算

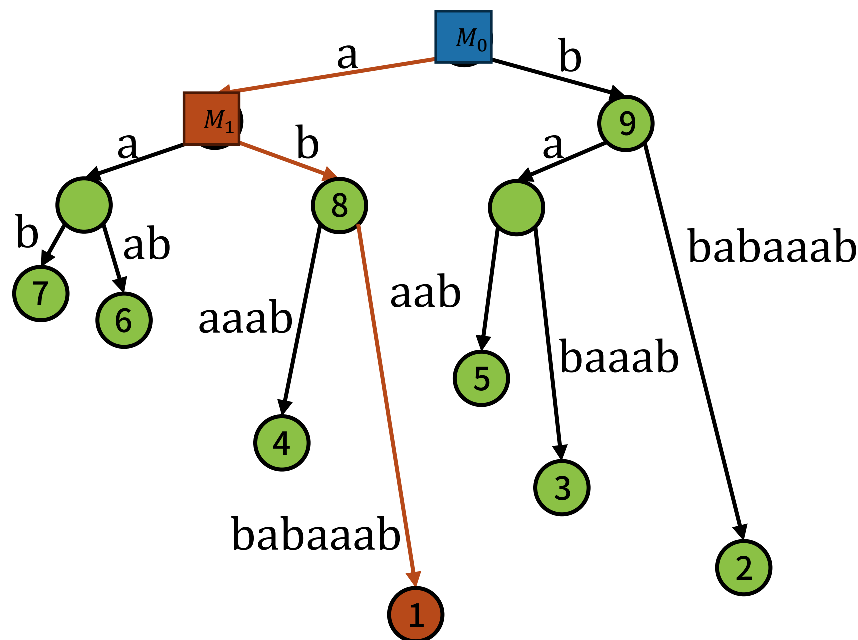
- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ を表す接尾辞木上のパスを見つける
  2. パス上に存在する中で最深のマーク M_j を探す
  3. $F_i = F_j T'[|F_j| + 1]$ として、接尾辞木中の F_i に対応する地点にマーク M_i をつける



$T' = \text{abbabaaab}$
$F = (\varepsilon)$

接尾辞木によるLZ78分解の計算

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ を表す接尾辞木上のパスを見つける
  2. パス上に存在する中で最深のマーク M_j を探す
  3. $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ として、接尾辞木中の F_i に対応する地点にマーク M_i をつける

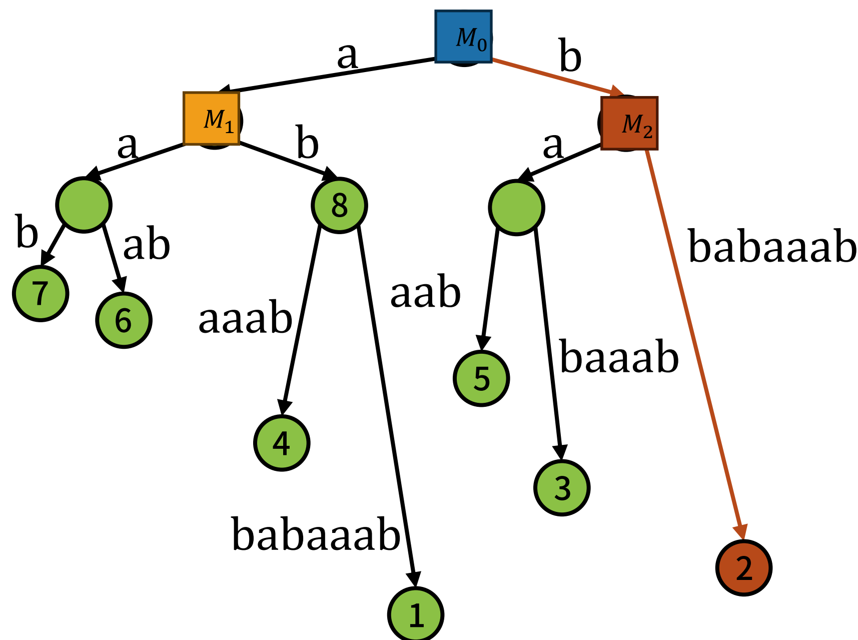


$T' =$ abbabaaab

$F = (\varepsilon, \mathrm{a})$

接尾辞木によるLZ78分解の計算

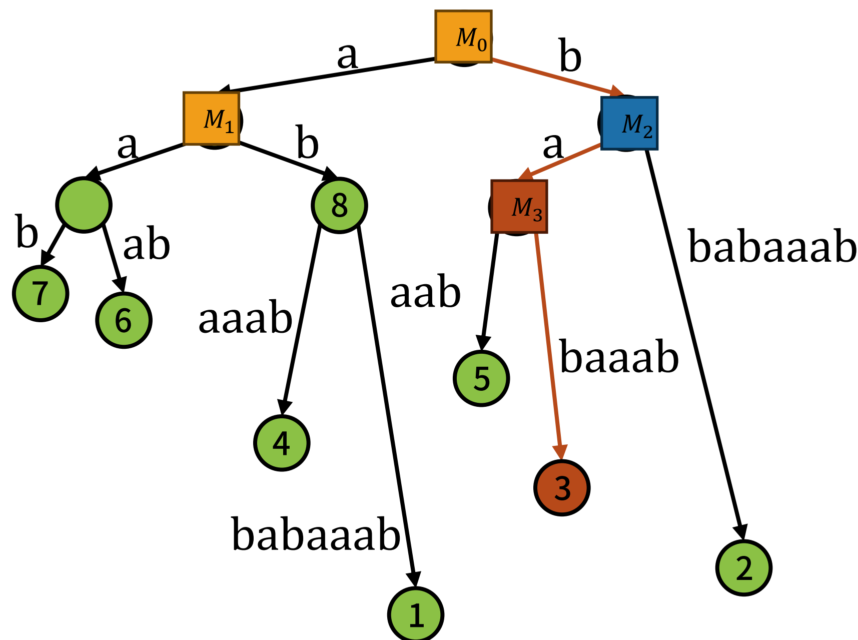
- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ を表す接尾辞木上のパスを見つける
  2. パス上に存在する中で最深のマーク M_j を探す
  3. $F_i = F_j T'[|F_j| + 1]$ として、接尾辞木中の F_i に対応する地点にマーク M_i をつける



$T' = $ abbabaaab
$F = (\varepsilon, \mathrm{a}, \mathrm{b})$

接尾辞木によるLZ78分解の計算

- $F_i\ (i \geq 1)$ を計算するアルゴリズムは以下の通り:
  1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ を表す接尾辞木上のパスを見つける
  2. パス上に存在する中で最深のマーク M_j を探す
  3. $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ として、接尾辞木中の F_i に対応する地点にマーク M_i をつける

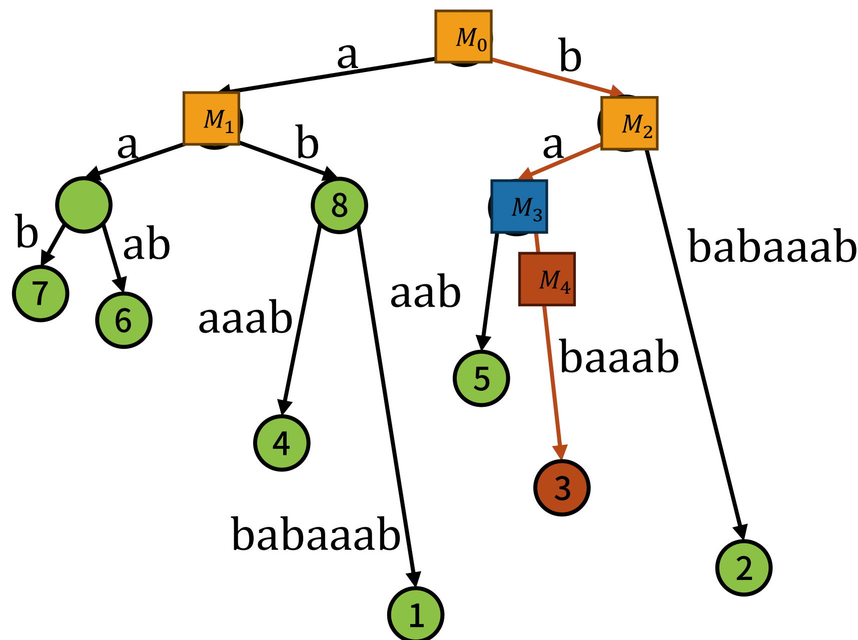


$T' = $ abbabaaab

$F\ = (\varepsilon, \mathrm{a}, \mathrm{b}, \mathrm{ba})$

接尾辞木によるLZ78分解の計算

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ を表す接尾辞木上のパスを見つける
  2. パス上に存在する中で最深のマーク M_j を探す
  3. $F_i = F_j T'\left[\left|F_j\right| + 1\right]$ として、接尾辞木中の F_i に対応する地点にマーク M_i をつける

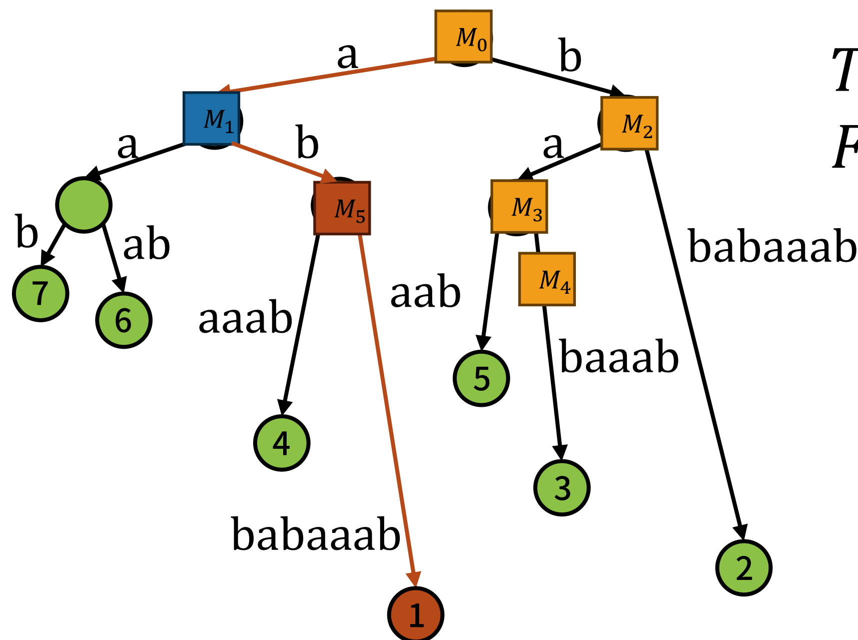


$T' =$ abbababaaab

$F = (\varepsilon, \mathrm{a}, \mathrm{b}, \mathrm{ba}, \mathrm{baa})$

接尾辞木によるLZ78分解の計算

- F_i $(i \geq 1)$ を計算するアルゴリズムは以下の通り:
  1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ を表す接尾辞木上のパスを見つける
  2. パス上に存在する中で最深のマーク M_j を探す
  3. $F_i = F_j T'[|F_j| + 1]$ として、接尾辞木中の F_i に対応する地点にマーク M_i をつける



$T' =$ abbababaaab

$F = (\varepsilon, \text{a}, \text{b}, \text{ba}, \text{baa}, \text{ab})$

接尾辞木によるLZ78分解の計算

アルゴリズム中で用いるテクニック

- T' を表すパス上の最深マークを見つける・マークをつける
  - ▲ Lowest Marked Ancestor Problemであり、O(1) 時間で処理 できる [Westbrook, 1992]
- F_i に対応する接尾辞木中の位置を得る
  - ▲ 接尾辞木上のWeighted Level Ancestor Problemであり、O(1) 時間で求まる [Gawrychowski et al., 2014, Bellazougui et al., 2021]

時間・空間計算量

前処理も $O(n)$ 時間

- 接尾辞木・LMA索引・WLA索引は全て $O(n)$ spaceなため、全体でも $O(n)$ space
- マーク地点の取得・マークが定数時間で行えるため、factorあたり $O(1)$ 時間

⇒ 全体の計算量は 空間 $O(n)$ ・時間 $O(z)$

※ z は T のLZ分解のfactor数

接尾辞木によるLZ78分解の部分文字列圧縮

部分文字列圧縮への対応: とても簡単

- 始点 l の対応: 前述のアルゴリズムの T' を

  $T' = T[\,l + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ に書き換えれば良い

- 終点 r の対応: 通常通り計算して、 $T[l..r]$ の範囲をはみ出したら末尾のfactorを削れば良い

計算量: $O(z_{l,r})$ time・$O(n + z_{l,r})$ space

※索引自体に $O(n)$ space必要で、作業領域が $O(z_{l,r})$ space

※ $z_{l,r}$ は $T[l..r]$ のLZ分解のfactor数

提案手法: CDAWGを使ったLZ78分解

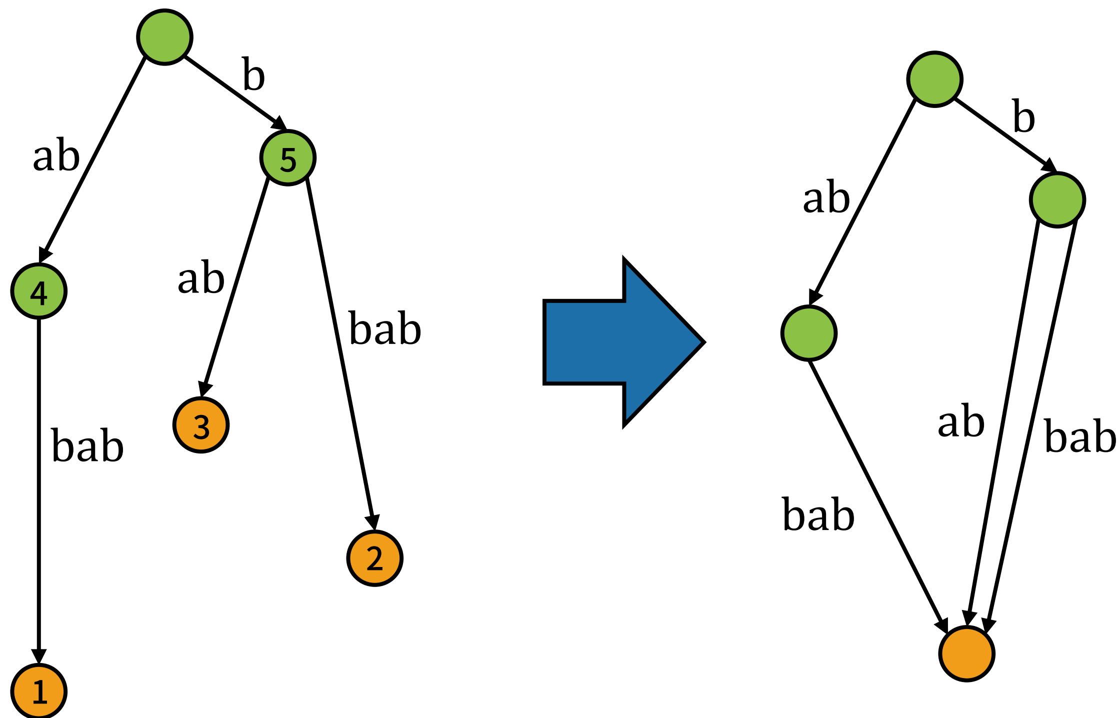
+ 部分文字列圧縮

CDAWG (Compacted Directed Acyclic Word Graph)

接尾辞木の同型な部分木をひとまとめにすることで得られる

辺ラベル付きDAG

$T = \mathrm{abbab}$

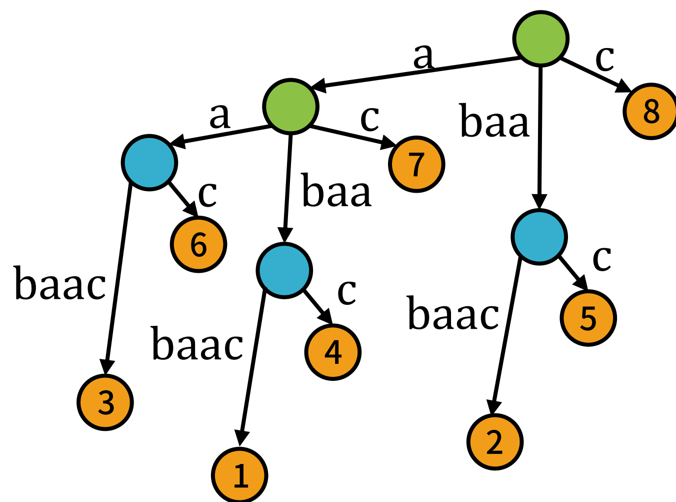


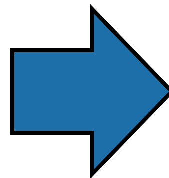
CDAWG (Compacted Acyclic Word Graph)

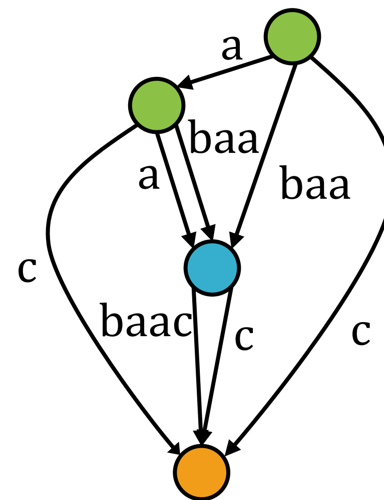
接尾辞木の<u>同型な部分木をひとまとめにする</u>ことで得られる

辺ラベル付きDAG

T = abaabaac





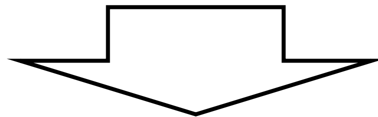


CDAWGによる文字列圧縮

- e : CDAWGの辺の数
  - 接尾辞木の辺数は $2n-1$ 以下なので、 $e \leq 2n-1$

性質: 文字列が繰り返し構造を持つと e は小さくなり、

$e \in \Theta(\log n)$ であるような文字列も存在



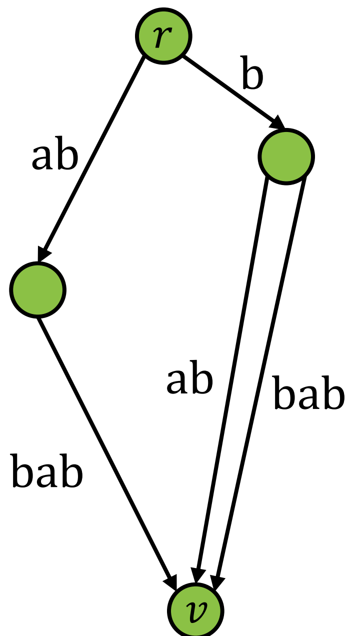
$O(e)$ spaceで索引を作ることができれば、元文字列より省領域な**圧縮索引**になる！

接尾辞木→CDAWG で置き換える際の課題

先行研究のアルゴリズムをCDAWGでシミュレートしたい → 課題あり

1. CDAWGでは祖先が一意に定まらない
2. CDAWGでは1つの頂点が複数文字列を表す

⇒ 別のアプローチを用いて解決する



$T = \mathrm{abbab}$
$S(v) = \{\mathrm{abbab}, \mathrm{bbab}, \mathrm{bab}\}$

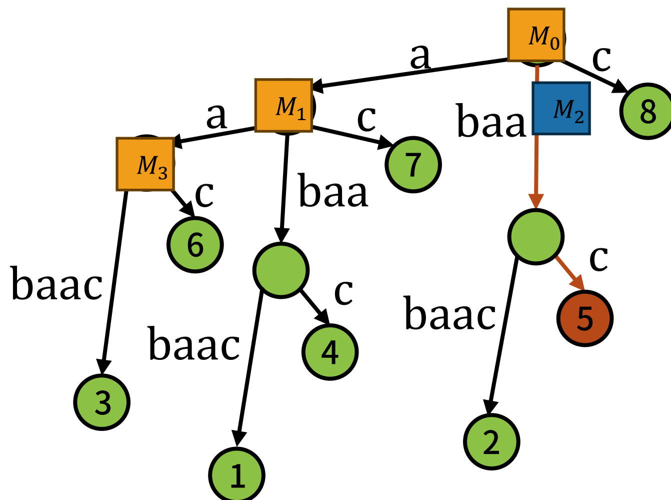
v の入次数は3で、祖先は2頂点ある

※ $S(v)$: CDAWGのroot r → 頂点 v のパスが表す文字列の集合

Lowest Marked Ancestorの帰着

先行研究のアルゴリズム中で扱うMarked Ancestor は以下のようなもの:

1. 頂点 v にマークをつける
2. 葉 ℓ を選び、ℓ から根へのパス上で最も深いマークを見つける



※ 辺 (u, v) へのマークは頂点 v へのマークとしてよい

※ 一般には接尾辞を表す頂点が葉とは限らないが、
T の末尾にuniqueな文字を追加することで
葉であることを保証できる

Lowest Marked Ancestorの帰着

アルゴリズム中で扱うMarked Ancestor Problemは以下のようなもの:

1. 頂点 v にマークをつける
2. 葉 ℓ を選び、ℓ から根へのパス上で最も深いマークを見つける

辺の探索順序は辞書順にしておく

木の葉を行きがけ順に並べて $\ell_1, \ell_2, \dots, \ell_n$ とすることで、上の2つの処理は以下の2つの処理に置き換えられる:

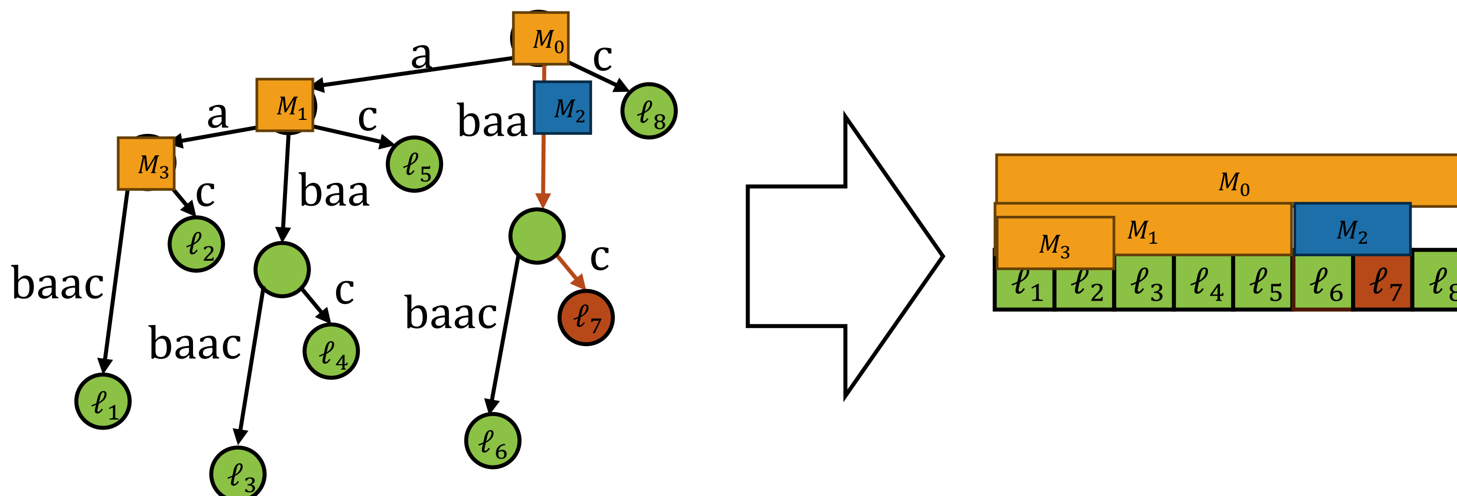
1. v の子孫である葉 $\ell_a, \ell_{a+1}, \dots, \ell_b$ に重み $\mathrm{depth}(v)$ でマークを付ける
2. ℓ_k についたマークのうち、重みが最大のものを求める

※$\mathrm{depth}(v)$: 頂点 v の深さ

Lowest Marked Ancestorの帰着

木の葉を行きがけ順（辺順序は辞書順）に並べて $\ell_1, \ell_2, \dots, \ell_n$ とする

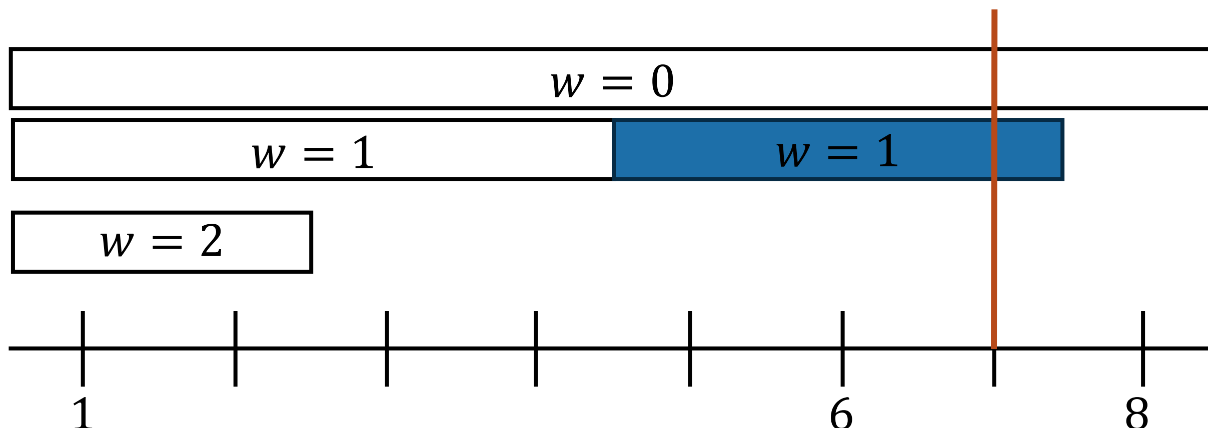
1. v の子孫である葉 $\ell_a, \ell_{a+1}, \dots, \ell_b$ に重み depth(v) でマークを付ける
2. ℓ_k についたマークのうち、重みが最大のものを求める



Lowest Marked Ancestorの帰着

問題をさらに抽象化すると、以下の問題に帰着できる:

1. 重み w の区間 $[a, b]$ を集合に追加
2. 整数 k を包含するような区間のうち、重みが最大のものを求める



※頂点に対応する $[a, b]$ や葉に対応する k は高速に求まると仮定

Lowest Marked Ancestorの帰着

問題をさらに抽象化すると、以下の問題に帰着できる:

1. 重み w の区間 $[a, b]$ を集合に追加
2. 整数 k を包含するような区間のうち、重みが最大のものを求める

この問題は1次元のstabbing-max problemであり、以下の条件を満たすようなデータ構造が存在 [Nekrich, 2011]

- 各クエリの時間計算量: ならし $O(\log n)$
- 空間計算量: $O(m)$ words

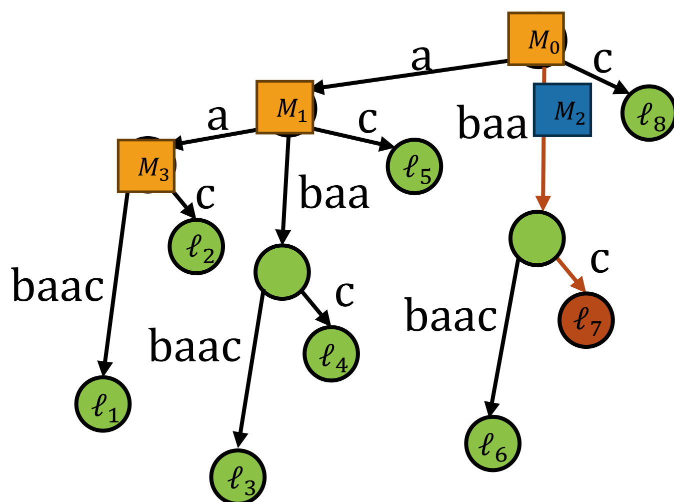
※ m : 区間の個数

$T[l..r]$ に対応する区間の取得

以下の2つを高速に処理する必要がある:

1. (l,r) が与えられたとき、$T[l..r]$ に対応する葉の区間 $[a,b]$ を求める
2. i が与えられたとき、$T[i..n]$ に対応する葉 ℓ_k を求める

葉を辞書順に並べると、これらは接尾辞配列上の操作に対応



M_3 M_1 M_0 M_2

SA[i]	$T[\mathrm{SA}[i],n]$
3	aabaac
6	aac
1	abaabaac
4	abaac
7	ac
2	baabaac
5	baac
8	c

$T[l..r]$ に対応する区間の取得

以下の性質を満たすCDAWG-basedの圧縮索引が存在する

[Bealazzougui and Cunial, CPM 2017]

- $T[i], \mathrm{ISA}[i]$の取得: $O(\log n)$ time
- $T[l..r]$ に対応する区間の取得: $O(\log n)$ time
- 空間計算量: $O(e)$ space

※ISA: 接尾辞配列 SA の逆配列 ($\mathrm{ISA}\big[\mathrm{SA}[i]\big] = i$)

CDAWGによる計算: 処理のまとめ

以下のような手続きの繰り返しでLZ78のfactorを計算できる:

1. $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ に対応する SA 上の位置
   $k = \mathrm{ISA}[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|]$ を探す
2. k を包含する重み w が最大の区間 $[a, b]$ を求める
3. $F_i = F_j T'[w + 1]$ とする
4. F_i に対応する SA 上の区間 $[a, b]$ を探し、重み $w + 1$ の区間 $[a, b]$ を追加

$T' =$ abaabaac

$F = (\varepsilon, \mathrm{a}, \mathrm{b}, \mathrm{aa})$

SA[i]	$T[\mathrm{SA}[i], n]$
3	aabaac
6	aac
1	abaabaac
4	abaac
7	ac
2	baabaac
5	baac
8	c

M_3 M_1 M_0 M_2

CDAWGによる計算: 処理のまとめ

以下のような手続きの繰り返しでLZ78のfactorを計算できる:

1. $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ に対応する SA 上の位置
   $k = \mathrm{ISA}[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|]$ を探す
2. k を包含する重み w が最大の区間 $[a, b]$ を求める
3. $F_i = F_j T'[w+1]$ とする
4. F_i に対応する SA 上の区間 $[a, b]$ を探し、重み $w+1$ の区間 $[a, b]$ を追加

$T' = $ abaabaac

$F = (\varepsilon, \mathrm{a}, \mathrm{b}, \mathrm{aa})$

M_3 M_1 M_0 M_2

$\mathrm{SA}[i]$	$T[\mathrm{SA}[i], n]$
3	aabaac
6	aac
1	abaabaac
4	abaac
7	ac
2	baabaac
5	baac
8	c

CDAWGによる計算: 処理のまとめ

以下のような手続きの繰り返しでLZ78のfactorを計算できる:

1. $T' = T[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ に対応する SA 上の位置 $k = \mathrm{ISA}[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|]$ を探す
2. k を包含する重み w が最大の区間 $[a, b]$ を求める
3. $F_i = F_j T'[w + 1]$ とする
4. F_i に対応する SA 上の区間 $[a, b]$ を探し、重み $w + 1$ の区間 $[a, b]$ を追加

$T' = \text{abaabaac}$

$F = (\varepsilon, \text{a}, \text{b}, \text{aa}, \text{ba})$

M_3 M_1 M_0 M_2

$\mathrm{SA}[i]$	$T[\mathrm{SA}[i], n]$
3	aabaac
6	aac
1	abaabaac
4	abaac
7	ac
2	baabaac
5	baac
8	c

CDAWGによる計算: 処理のまとめ

以下のような手続きの繰り返しでLZ78のfactorを計算できる:

1. $T' = T[1 + |F_0| + |F_1| + \cdots, |F_{i-1}|..n]$ に対応する SA 上の位置
   $k = \mathrm{ISA}[1 + |F_0| + |F_1| + \cdots + |F_{i-1}|]$ を探す
2. k を包含する重み w が最大の区間 $[a, b]$ を求める
3. $F_i = F_j T'[w + 1]$ とする
4. F_i に対応する SA 上の区間 $[a, b]$ を探し、重み $w + 1$ の区間 $[a, b]$ を追加

計算量:

- 時間: ならし $O(\log n)$ / factor → 全体で $O(z \log n)$ time
- 空間: $O(e + z)$ space
  - e（CDAWGの辺数）はCDAWG-basedの索引の分
  - z（T のLZ分解のfactor数）はstabbing-maxのためのデータ構造の分

factorの個数分だけ区間を保存する

※ n は T の長さ

LZ78部分文字列圧縮への対応

接尾辞木の場合と同様に、部分文字列圧縮に対応できる

- 始点 l の対応: 前述のアルゴリズムの T' を

  $T' = T[\,l + |F_0| + |F_1| + \cdots + |F_{i-1}|..n]$ に書き換えれば良い

- 終点 r の対応: 通常通り計算して、$T[l..r]$ の範囲を

  はみ出したら末尾のfactorを削れば良い

接尾辞木と同じ

計算量: $O\left(z_{l,r} \log n\right)$ time・$O\left(e + z_{l,r}\right)$ space

クエリの度にstabbing-maxのためのデータ構造を構築する

※索引自体は $O(e)$ space、作業領域が $O(z_{l,r})$ space

※ $z_{l,r}$ は $T[l..r]$ のLZ分解のfactor数

まとめ・展望

LZ78分解の部分文字列圧縮を以下の計算量で行う手法を提案:

- **時間:** $O(z_{l,r} \log n)$
- **空間:** $O(e)$

既存手法（$O(z_{l,r})$ time, $O(n)$ space）からの省領域化を達成

展望

- **他の問題も $O(e)$ sizeの索引で解けないか**
  - CDAWG上のパスクエリ等も圧縮領域で行えるため、応用は幅広い
- **CDAWG以外の索引・他の部分文字列圧縮への一般化**
  - 1文字アクセス・ISA・SA範囲の3処理が行える索引ならCDAWGの代わりに使える
  - 他の部分文字列圧縮もこの枠組みで解けるかもしれない（一部は既知）