2026/1/13 SIGAL206

極大閉部分文字列の
オンライン計算アルゴリズム

柴田 紘希, 梅崎 陽生, 中島 祐人, 稲永 俊介

（九州大学）

発表スライド

文字列中の極大連続反復

極大連続反復: 以下の3つの条件を満たすような同じ部分文字列の組

1. **連続性**: 2つの出現の間にそれらと同じ文字列が出現しない
2. **右極大性**: それぞれの右隣の文字が一致しない
3. **左極大性**: それぞれの左隣の文字が一致しない

極大連続反復

文字列中の極大連続反復

極大連続反復: 以下の3つの条件を満たすような同じ部分文字列の組

1. **連続性**: 2つの出現の間にそれらと同じ文字列が出現しない
2. **右極大性**: それぞれの右隣の文字が一致しない
3. **左極大性**: それぞれの左隣の文字が一致しない

文字列中の極大連続反復

極大連続反復: 以下の3つの条件を満たすような同じ部分文字列の組

1. **連続性**: 2つの出現の間にそれらと同じ文字列が出現しない
2. **右極大性**: それぞれの右隣の文字が一致しない
3. **左極大性**: それぞれの左隣の文字が一致しない

右極大性なし

極大閉部分文字列

時間の都合で厳密な定義は省略
（スライドのAppendixに記載してます）

極大閉部分文字列: 極大性を持つ閉部分文字列

- 極大連続反復と一対一対応を持つことが知られている

以降では、極大連続反復を計算するアルゴリズムを説明します☺

極大連続反復に関する既存研究

- 極大連続反復をオフラインに $O(n \log n)$ 時間で計算するアルゴリズムが存在 [Badkobeh et al., 2024]
- オンライン手法（＝文字の追加に対応した手法）は未知

今回示した定理

文字列の先頭への文字追加に対し、1文字あたりならし $O(\log n)$ 時間で極大連続反復をオンラインに計算できる。

※入力文字列の長さを n と表記

文字列中の極大連続反復（再掲）

極大連続反復: 以下の3つの条件を満たすような同じ部分文字列の組

1. **連続性**: 2つの出現の間にそれらと同じ文字列が出現しない
2. **右極大性**: それぞれの右隣の文字が一致しない
3. **左極大性**: それぞれの左隣の文字が一致しない

左極大性以外の条件を満たす文字列（**右極大連続反復**）の変化を考える！

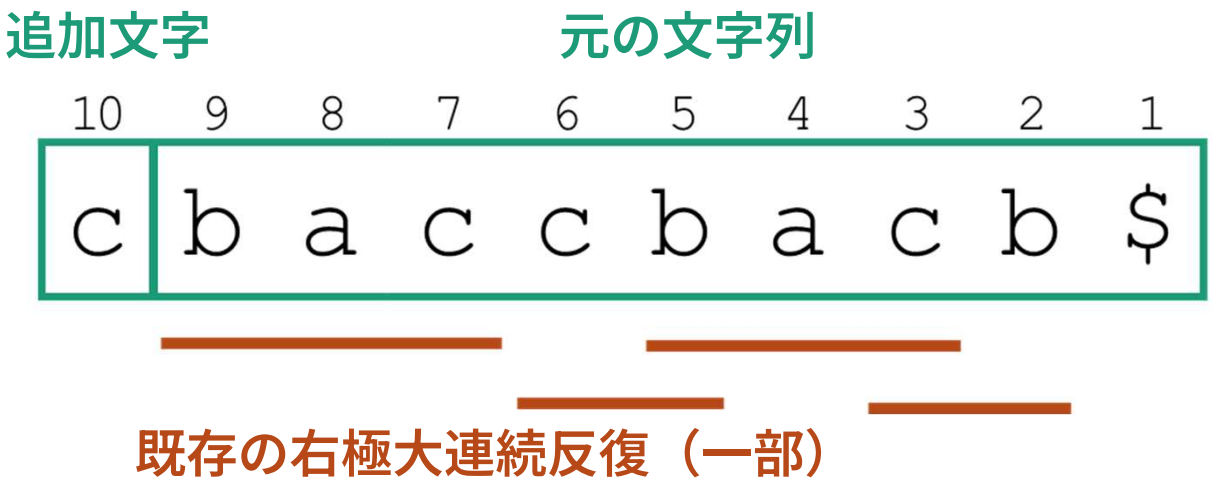
（左極大性は後からチェックすればよい）

右極大連続反復を考える利点

観察

元の文字列における右極大連続反復は、先頭への文字追加後も右極大連続反復のままである。

→ 新規発生する右極大連続反復だけ考えれば良く、扱いやすい！

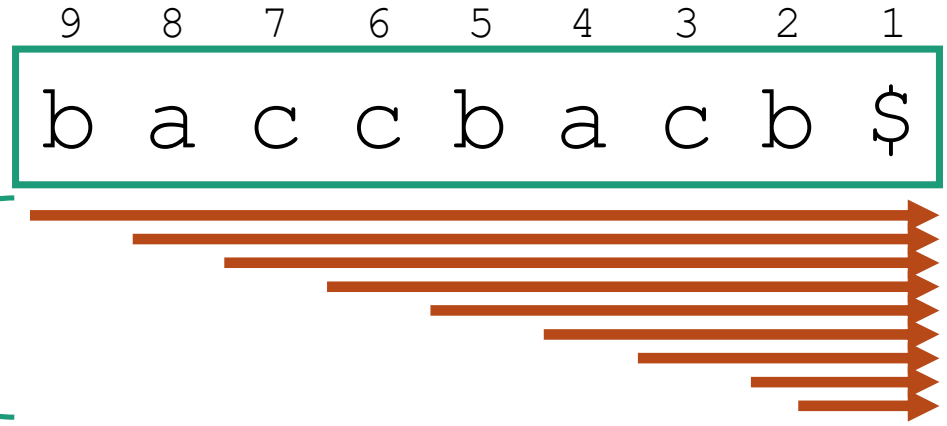


接尾辞木

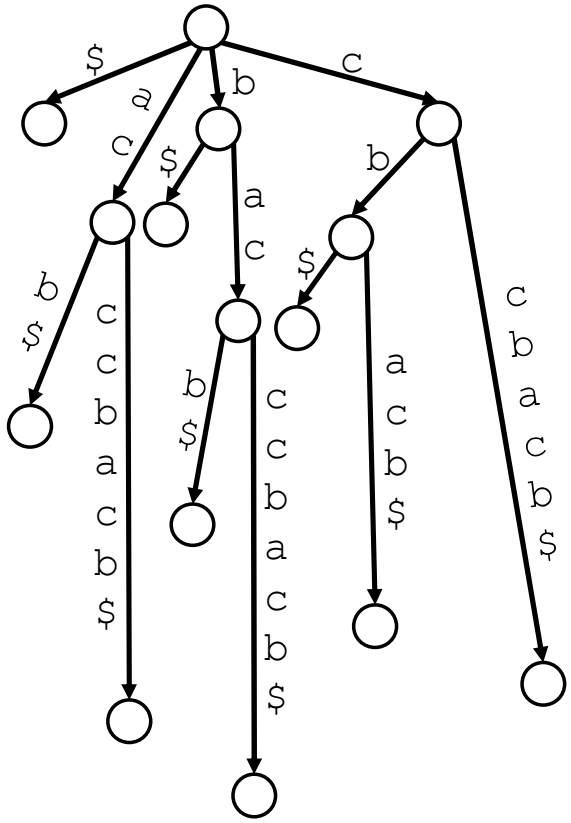
枝分かれのない頂点を縮約したトライ

接尾辞木: 文字列の接尾辞を表すコンパクトトライ

- 各葉が文字列の接尾辞を表している
- 頂点数は $O(n)$ であり、$O(n)$ 領域で表現できる



接尾辞木の
各葉へのパスが
表す文字列



baccbacb$の接尾辞木

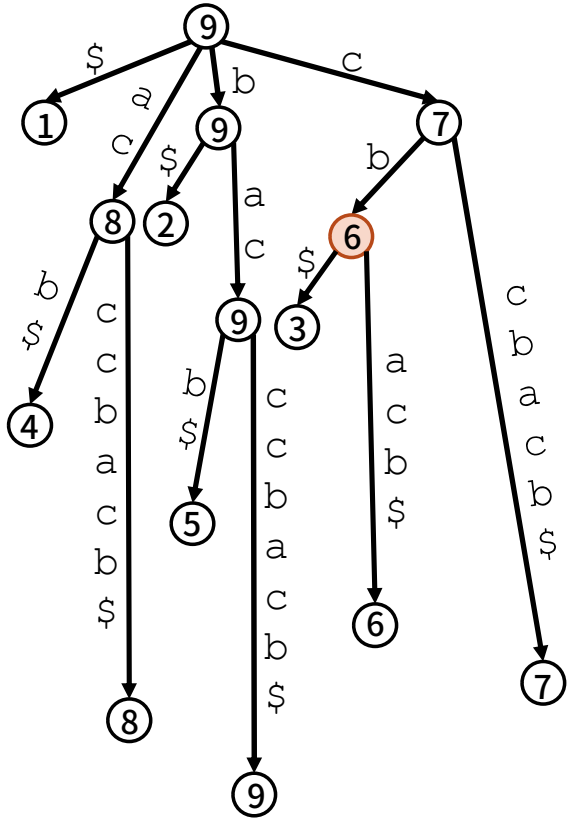
※入力文字列の末尾文字がuniqueな文字 $ で終わることを仮定

接尾辞木へのラベル付け

頂点が表す文字列の最左出現の位置で頂点にラベル付けする

- 図中の赤い頂点（文字列 cb を表す頂点）の例
  - cb は位置 6,3 の2カ所に出現
  - 最左出現の位置は 6 なので、6 でラベル付け

9	8	7	6	5	4	3	2	1
b	a	c	c	b	a	c	b	$



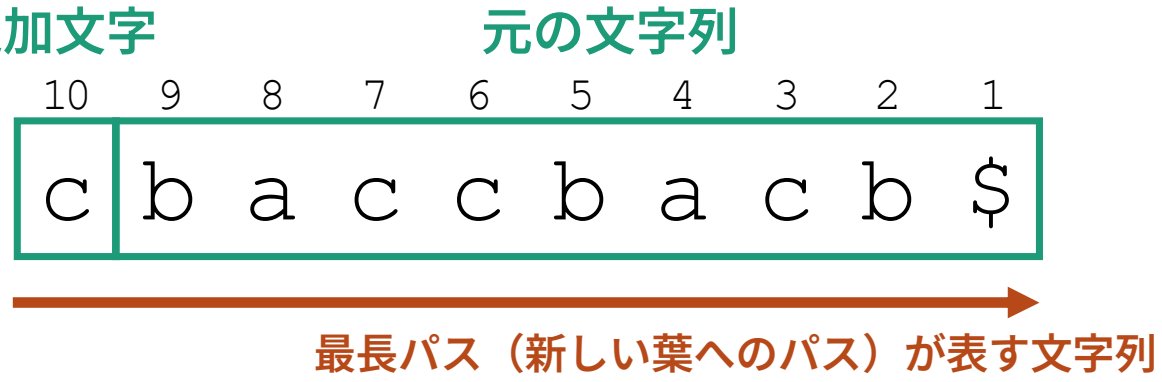
baccbacb$の接尾辞木

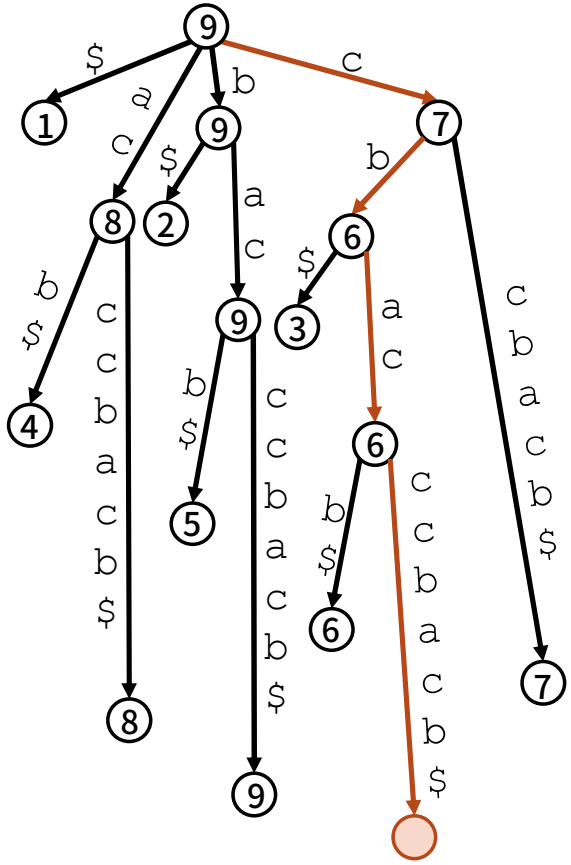
※indexは右側から1,2,…と数える（左側に文字を追加するため）

文字の先頭追加による接尾辞木の更新

更新後の文字列全体を表す新しい葉を1つ追加することで接尾辞木を更新できる

→ この際に新しい右極大連続反復を計算したい！





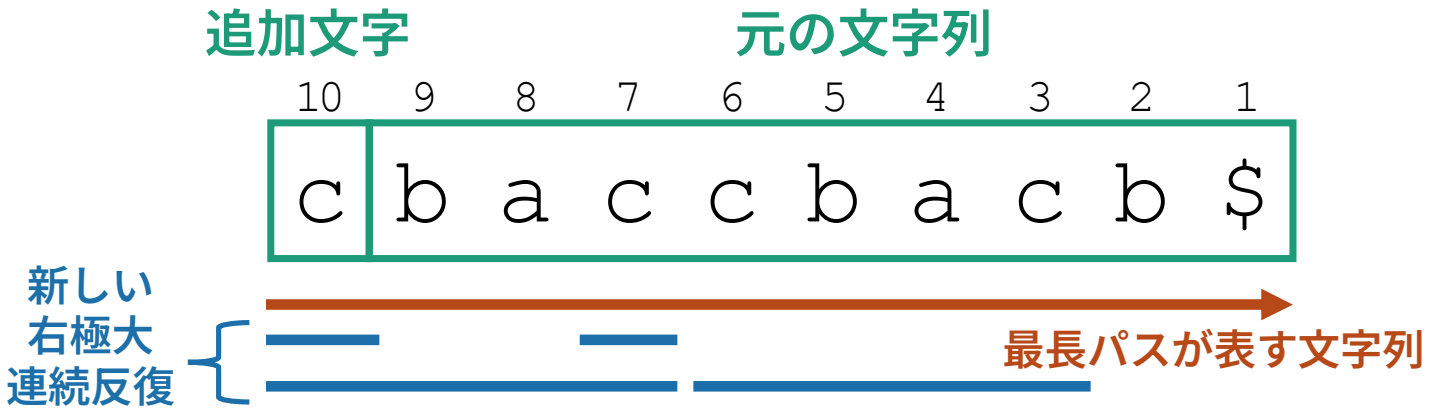
cbaccbacb$の接尾辞木

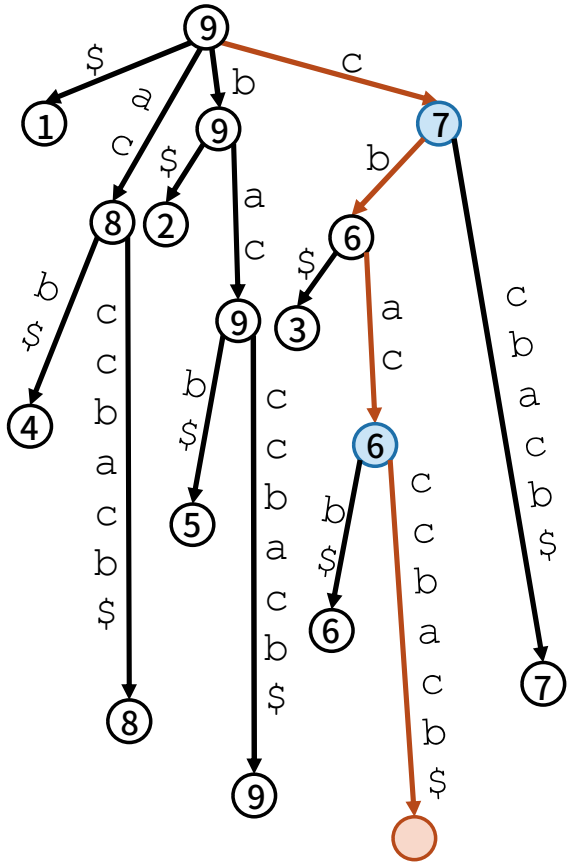
新しい右極大連続反復に関する性質

観察から導ける性質

新しく生まれる右極大連続反復は、以下のようになる:

- **反復の文字列**: 最長パス上に現れる、「最長パスの1つ下の頂点とのラベルが異なる」頂点に対応
- **左側の出現位置**: 更新後の文字列の接頭辞として現れる
- **右側の出現位置**: 頂点ラベルの値と一致





cbaccbacb$の接尾辞木

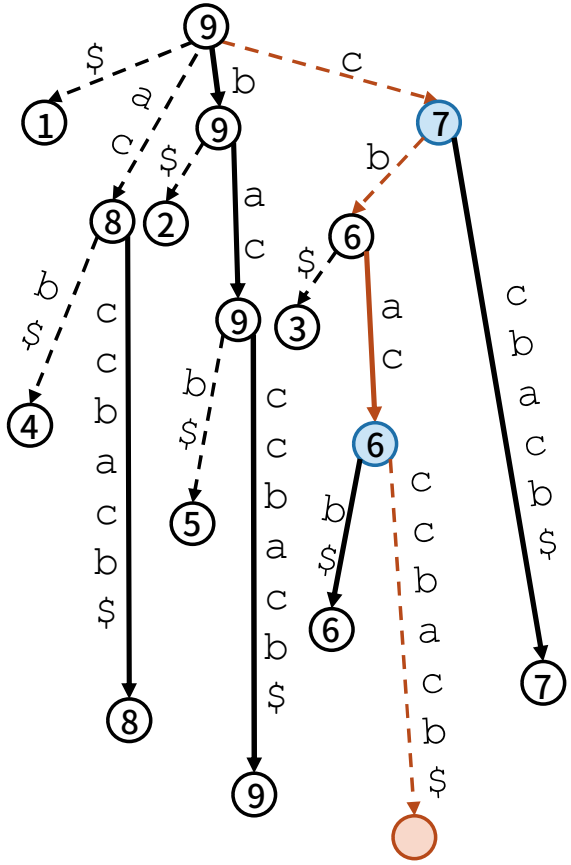
Heavy EdgeとLight Edge

接尾辞木のラベルが同じ頂点を結ぶ辺をHeavy Edge、
そうでない頂点をLight Edgeとする

観察4

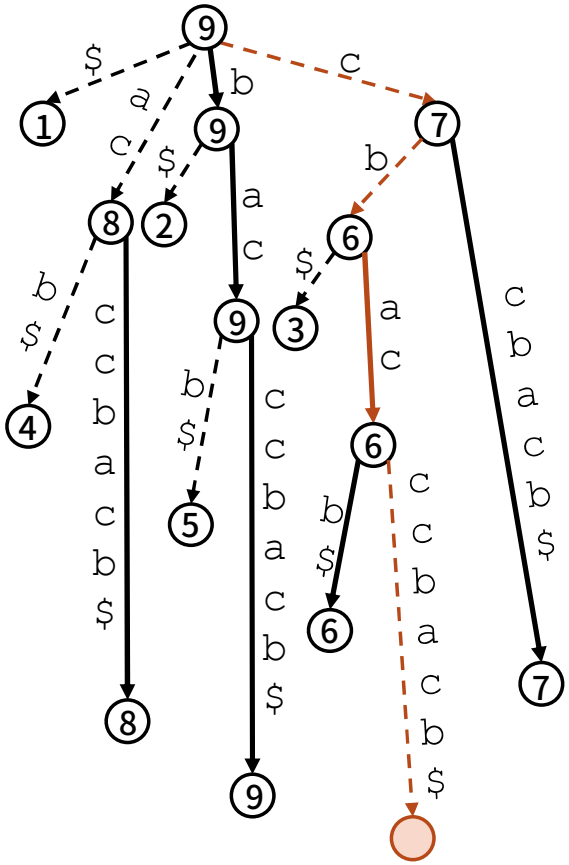
新しい右極大連続反復に対応する頂点は
Light Edgeの上側端点に限られる。

→ 最長パス上のLight Edgeを高速に列挙できれば、
新規右極大連続反復も列挙できる！

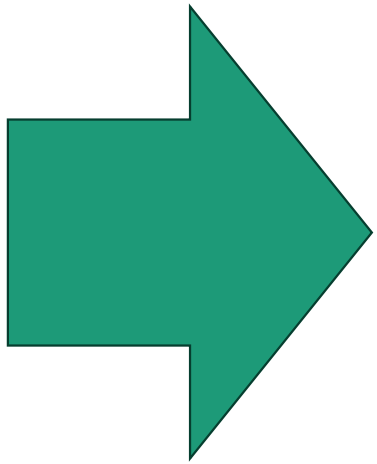


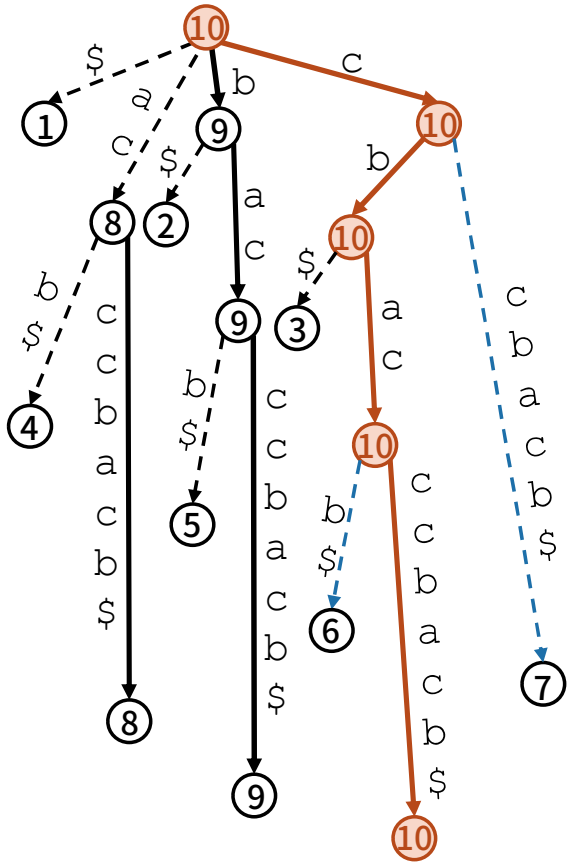
cbaccbacb$の接尾辞木

接尾辞木の頂点ラベル・辺の更新



頂点ラベル・辺属性の更新前





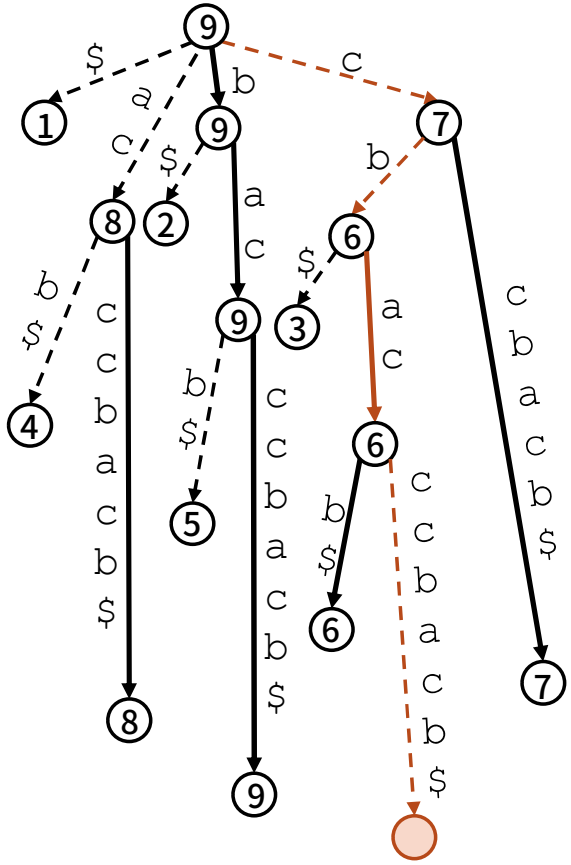
頂点ラベル・辺属性の更新後

接尾辞木に対する操作のまとめ

接尾辞木に対して以下の操作を行えればよい:

1. 頂点への葉の追加
2. 根から葉のパス上のLight Edgeを列挙
3. 列挙したLight EdgeをHeavy Edgeに切り替え、既存Heavy Edgeを適切にLight Edgeに切り替え

実は、これらはLink-Cut Tree [Sleator & Tarjan, 1983]の内部で行っている処理と完全に同一であり、ならし $O(\log n)$ 時間で行える！



cbaccbacb$の接尾辞木

アルゴリズム全体の計算量

文字追加の際に行う各手続きの計算量:

- 接尾辞木への葉の追加: ならし $O(\log \sigma) \subseteq (\log n)$
- 右極大連続反復の列挙: ならし $O(\log n)$
- Heavy/Light Edgeの更新: ならし $O(\log n)$

} Weinerのアルゴリズム

} Link-Cut Tree

以上より、文字追加に対してならし $O(\log n)$ 時間で処理を行える

- 左極大性の判定を適宜行うことで、極大連続反復の列挙も同じ計算量で行える
- 極大連続反復・極大閉部分文字列の一対一対応より、極大閉部分文字列の列挙も可能

※ σ は文字の種類数

まとめ

極大連続反復のオンライン計算アルゴリズムを紹介

- 片側極大な連続反復・極大閉部分文字列・etc…にも使える
- 1文字追加あたりならし $O(\log n)$ 時間で、既存オフライン手法と同等の速度

研究のポイント: 自然に考察を行った結果、本質的にLink-Cut Treeと同等の構造が現れた！

- LCTをブラックボックスとして使う手法自体はたくさんある
- LCTの内部構造がそのまま現れる例はかなり珍しい（？）

Appendix

基本的な定義

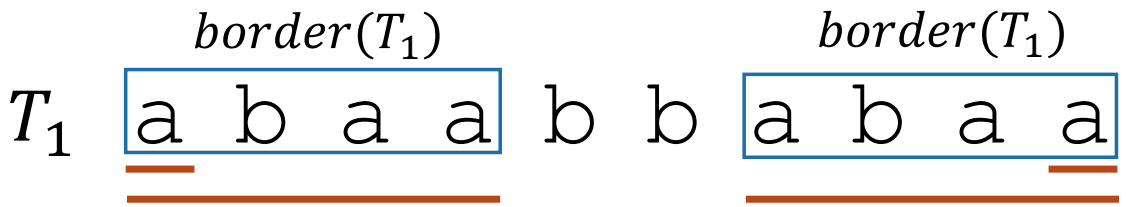
- **文字列**: 有限集合 Σ 上の列 $T \in \Sigma^*$
  - 今回は $|\Sigma| \in O\big(\mathrm{poly}(n)\big)$ で順序付きアルファベットを仮定
- T の**部分文字列**: T 中の連続する部分列
  - i 要素目〜 j 要素目の部分文字列を $T[i..j]$ と表記
- T の**接頭辞**: T の先頭部分を取り出した部分文字列 $T[1..j]$
- T の**接尾辞**: T の末尾部分を取り出した部分文字列 $T[i..n]$

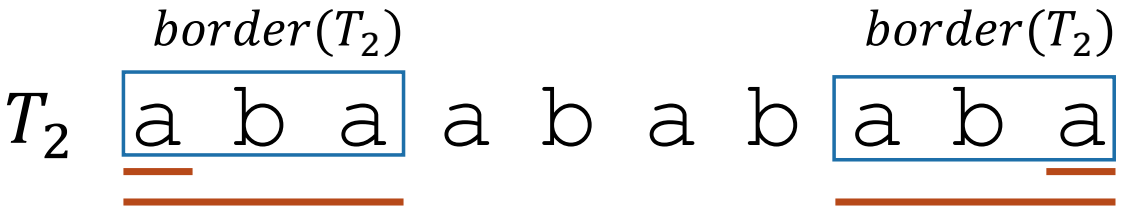
※本発表では文字列 T の長さを n と表記する

ボーダー (border)

定義: ボーダー

文字列 T の（真の）接頭辞かつ（真の）接尾辞であるような文字列を T の**ボーダー**と呼び、 T の最長ボーダーを $border(T)$ と表す。



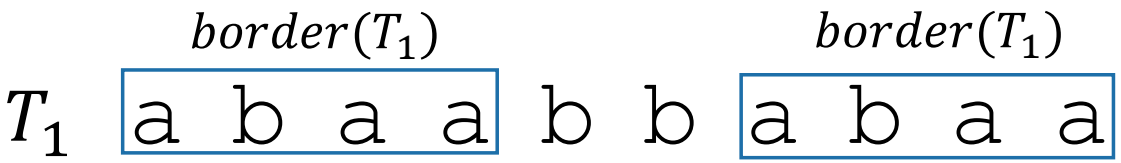


閉文字列 (closed string)

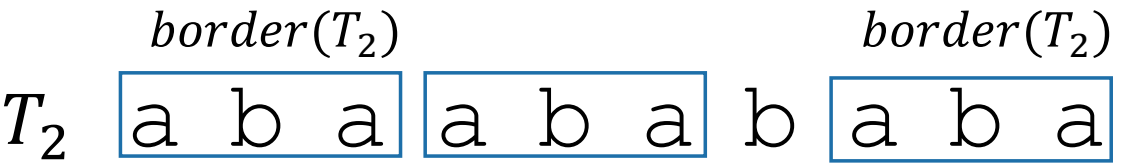
定義: 閉文字列

T にボーダーが存在し、T 中の $border(T)$ の出現回数がちょうど2回であるような文字列 T を閉文字列 (closed string) と呼ぶ。

※閉でない文字列を開文字列 (open string) と呼ぶ



T_1 は閉文字列



T_2 は開文字列

※今回は長さ1の文字列を開文字列と定義

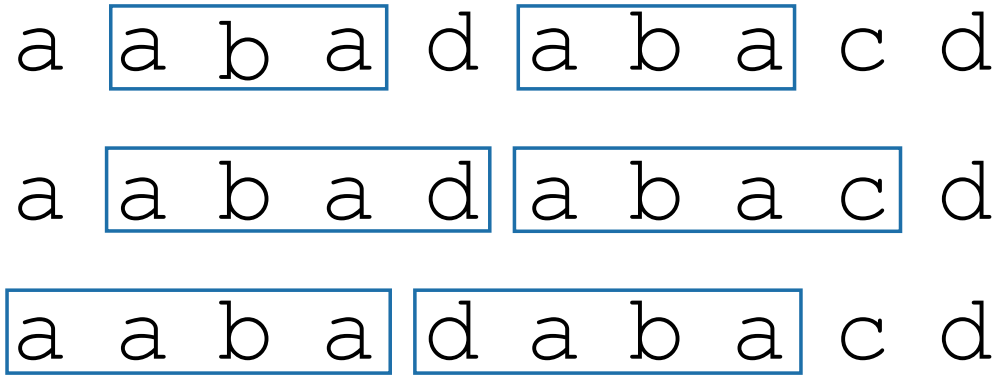
極大連続反復

定義: 極大連続反復

T 中の同じ部分文字列の連続する出現の組 $T[i..i'], T[j..j']$ $(i < j)$ が $T[i-1..i'] \neq T[j-1..j'], T[i-1..i'+1] \neq T[j..j'+1]$ を満たすとき、$T[i..i'], T[j..j']$ の組を T の**極大連続反復**と呼ぶ。

右上図青枠部分が極大連続反復

- 右に1文字伸ばすと不一致（中央図）
- 左に1文字伸ばすと不一致（下図）



極大閉部分文字列 (maximal closed substring, MCS)

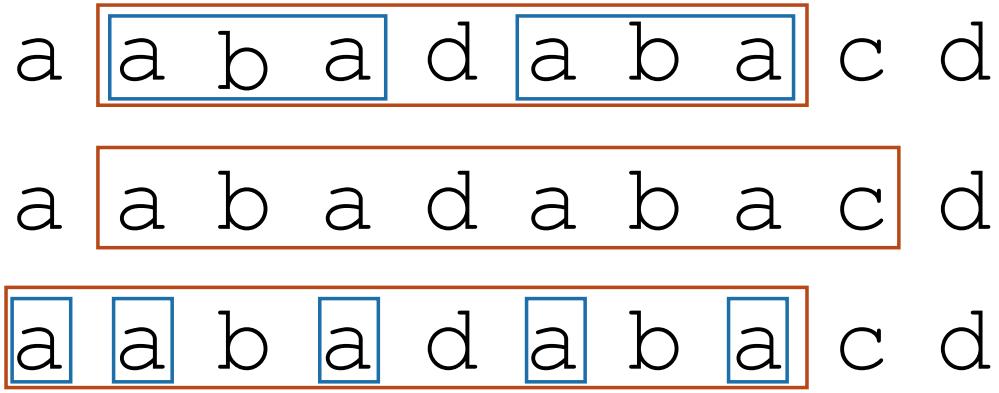
定義: 極大閉部分文字列

T 中の閉部分文字列 $T[i..j]$ について $T[i-1..j], T[i..j+1]$ がともに開部分文字列であるとき、$T[i..j]$を**極大閉部分文字列 (maximal closed substring, MCS)** と呼ぶ。

※片方向の拡張が開部分文字列な場合**右極大・左極大**な閉部分文字列と呼ぶ

右上図赤枠部分がMCSになっている

- 右に1文字伸ばすと開文字列（中央図）
- 左に1文字伸ばすと開文字列（下図）

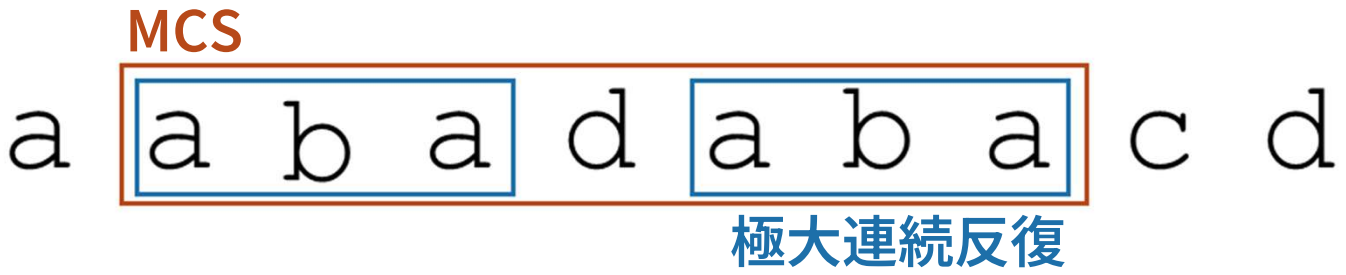


極大閉部分文字列と極大連続反復の関係

極大閉部分文字列 (MCS)・極大連続反復は**一対一対応**をもつ:

- $T[i..j']$ がMCSなら、 $T[i..j']$ の最長ボーダーの出現の組が極大連続反復
- $T[i..i'], T[j..j']$ が極大連続反復なら、 $T[i..j']$ は $T[i..i'] = T[j..j']$ を最長ボーダーとするMCS

→ 極大連続反復を全て求めることでMCSを全て求められる！（逆も成り立つ）



既存研究: MCS・極大連続反復の個数の上界・下界

- **上界: 片側極大に限っても $O(n \log n)$**
  - $O(n \log n)$ 時間の列挙アルゴリズムの存在から成り立つ
- **下界: $\Omega(n \log n)$**
  - ランダムな2進文字列の極大連続反復の個数の期待値が $\Theta(n \log n)$

    [Kosolobov, 2024]

今回扱った問題: 極大連続反復のオンライン計算問題

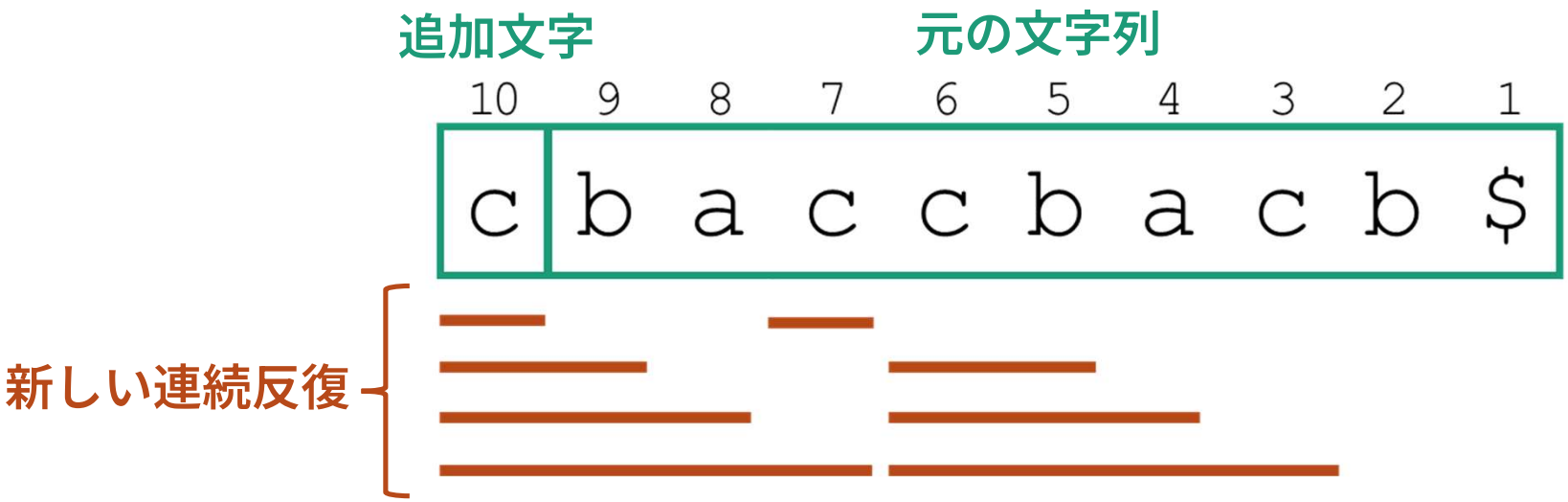
- 文字列 T（初期状態では空文字列）に対し、操作列 $c_1, \ldots, c_n$ $(c_i \in \Sigma)$ がオンラインで与えられる
- i 回目の操作では、文字列 T を Tc_i へと更新する
- どの更新操作の後でも、以下の処理を指定の計算量で行える必要がある:
  - T の極大連続反復の<u>個数の出力</u>（定数時間）
  - T の極大連続反復の<u>列挙</u>（極大連続反復の個数に対して線形時間）

観察A: 連続反復の新規出現

観察A

先頭文字追加で新しく出現する連続反復は以下の条件を満たす。

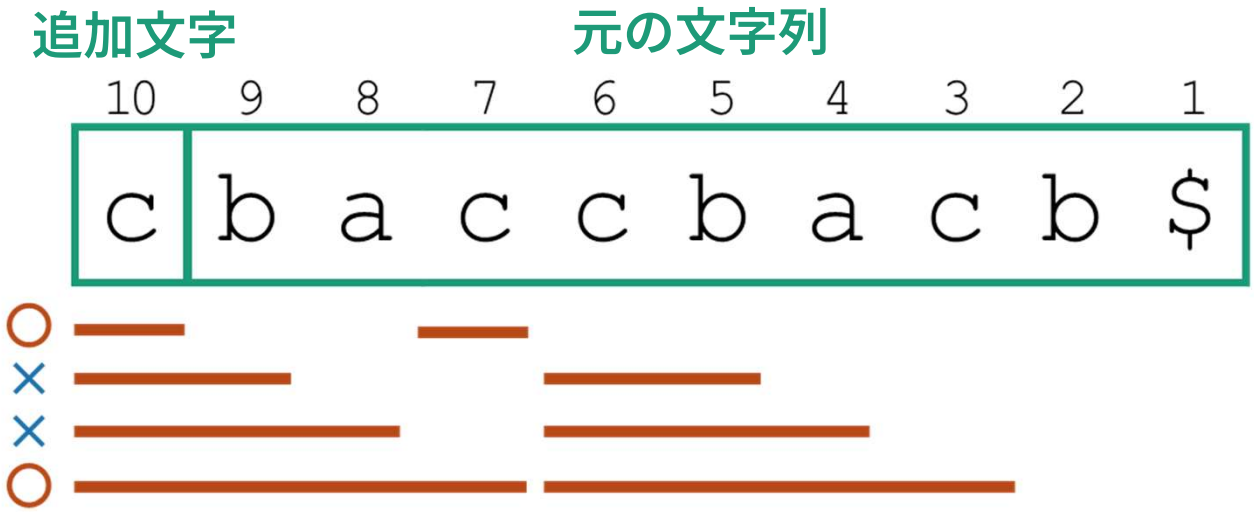
1. 左側の出現: 更新後の文字列の**接頭辞**
2. 右側の出現: 元の文字列中での**最左の出現**



観察B: 右極大である必要十分条件

観察B

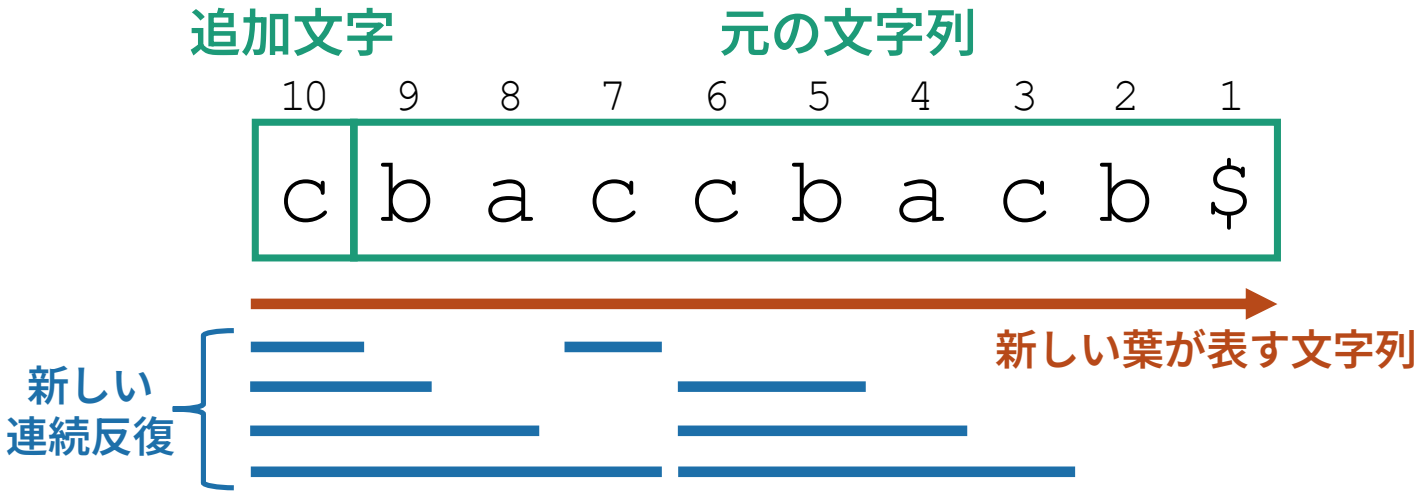
先頭文字追加で新しく出現する連続反復が右極大である必要十分条件は、「1文字長い接頭辞に対応する新規連続反復と、右側出現の開始位置が異なること」である。

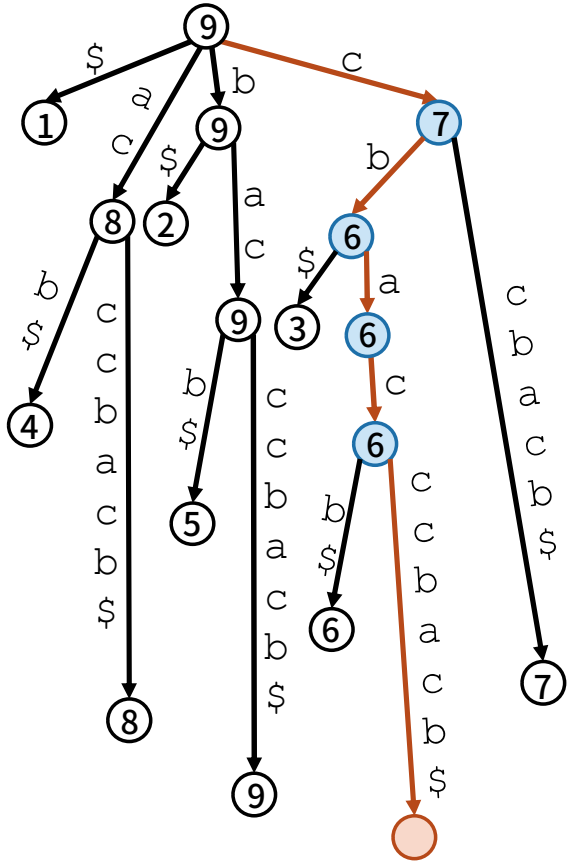


新しい連続反復の出現位置

観察Aから導ける性質

- 新しい連続反復は、接尾辞木の根～新しい葉を結ぶパス上の文字列となる
- 新しい連続反復の左側の出現位置は、その文字列に対応する頂点のラベルと一致する



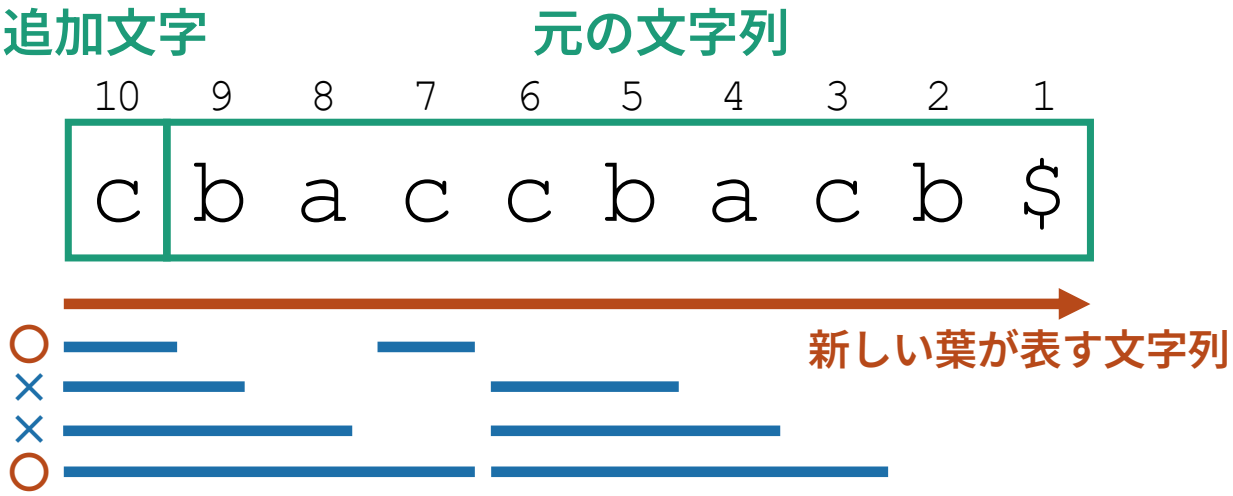


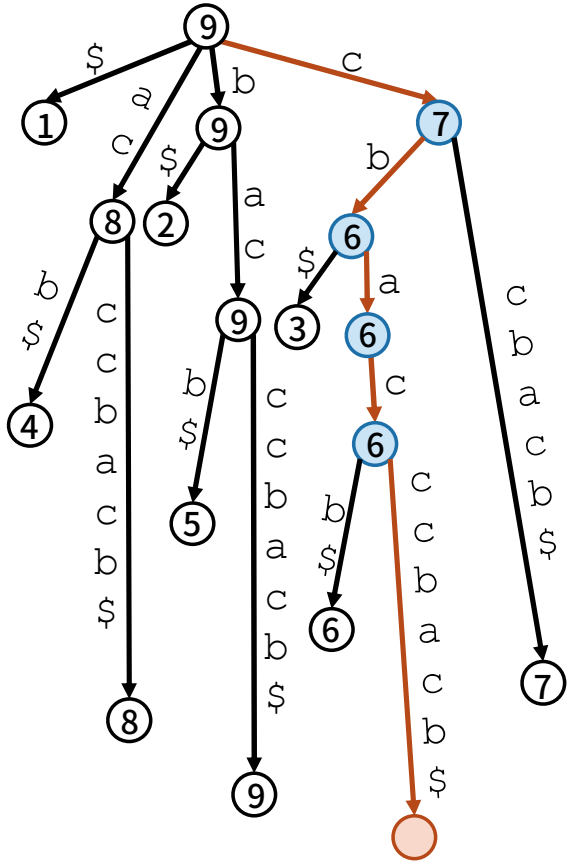
cbaccbacb$の接尾辞木

新しい連続反復の右極大性

観察Bから導ける性質

新しい連続反復が右極大である必要十分条件は、「文字列に対応する接尾辞木の頂点のラベルが、根～新しい葉を結ぶパス上の1つ下の頂点とラベルが異なること」である。



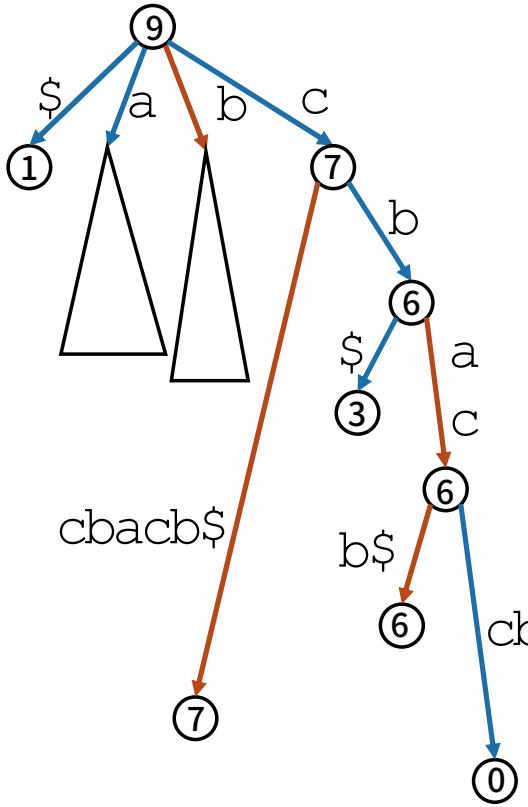


cbaccbacb$の接尾辞木

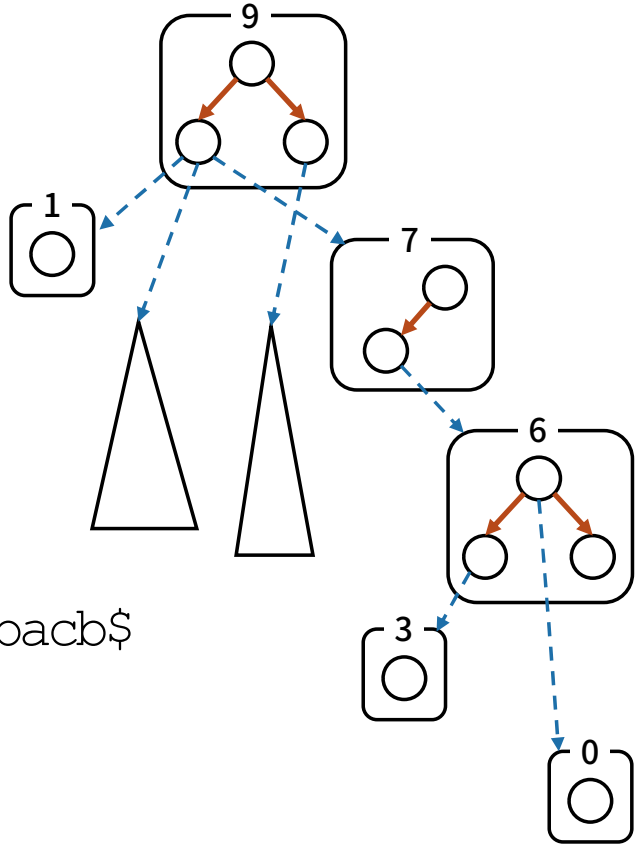
Link-Cut Treeを用いた接尾辞木の表現

各連結成分に属する頂点列をSplay木で管理し、Splay木を頂点とする木 (Link-Cut Tree [Sleator & Trarjan, 1983])を構築する

- 各Splay木はHeavy Edgeからなる連結成分（パスを成す）を表現
- Light edgeはSplay木同士を結ぶ辺で表現



cbaccbacb$の接尾辞木
（一部省略）



接尾辞木に対応するLink-Cut Tree（一部省略）

Link-Cut Treeの更新の計算量

定理 [Sleator & Tarjan, 1983]

頂点数 m の木を表現するLink-Cut Treeに以下の一連の操作を行う計算量は償却 $O(\log m)$ である。

1. 新規の葉頂点の追加
2. 根〜新規葉頂点を結ぶSplay Treeの列挙
3. 根〜新規葉頂点 を結ぶパス上の辺を全てheavy edgeに変化させ、パス性を保つように既存のheavy edgeをlight edgeに更新

この定理により、1文字追加あたり**ならし** $O(\log n)$ **時間**でLink-Cut Treeをメンテナンスできる