2024/11/29 AFSA 2024年度第2回領域集会

トライの接尾辞木に対する高速な更新手法

柴田 紘希（九州大学）

トライの接尾辞木に対する高速な更新手法

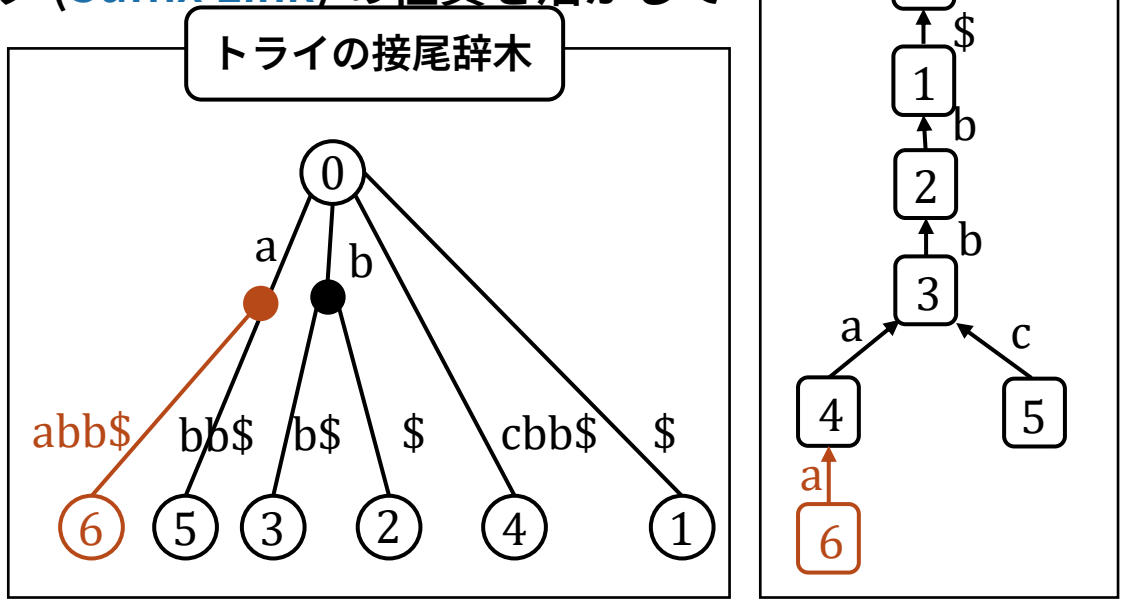
トライの接尾辞木: トライに対して高速な部分文字列検索ができる索引

- 文字列集合に対する検索などに使える

やったこと: トライの更新時にトライの接尾辞木を高速に更新する手法を提案

研究の面白さ: 接尾辞木のリンク (Suffix Link) の性質を活かして

木上の探索+データ構造で解く



研究の概要

- **トライの接尾辞木**: トライ中の全ての「頂点→根」へのパス文字列を表すコンパクトトライ
  - コンパクトトライ: 枝分かれのある頂点を縮約したトライ木

研究でやったこと

- トライへの頂点追加時に**接尾辞木を高速に更新する**手法を提案

	更新の時間計算量
[Breslauer, 1998]	$O(\sigma)$
[Funakoshi et al., 2024]	$O(\log n)$
提案手法	$O(\log \sigma + f)$

n: 接尾辞集合の要素数

σ: 文字の種類数

$$f = \frac{(\log \log n)^2}{\log \log \log n}$$

トライの頂点に対応する文字列

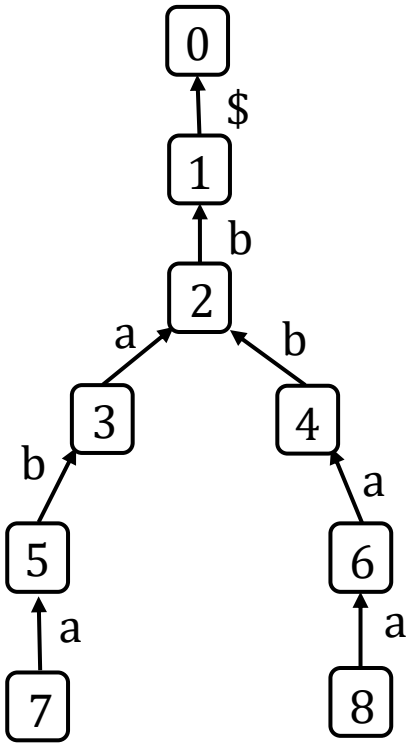
定義: トライの頂点に対応する文字列 = トライのある頂点から根までのパス文字列

- 通常のトライと逆向きに文字を読む

このようなトライをBackward Trieと呼ぶ

頂点に対応する文字列に関する性質:

- トライの頂点に対応する文字列は木の頂点数と同じだけ存在
- 文字列 S がトライのある頂点に対応する文字列なら、$S[i..|S|]$ もトライのある頂点に対応する文字列になる

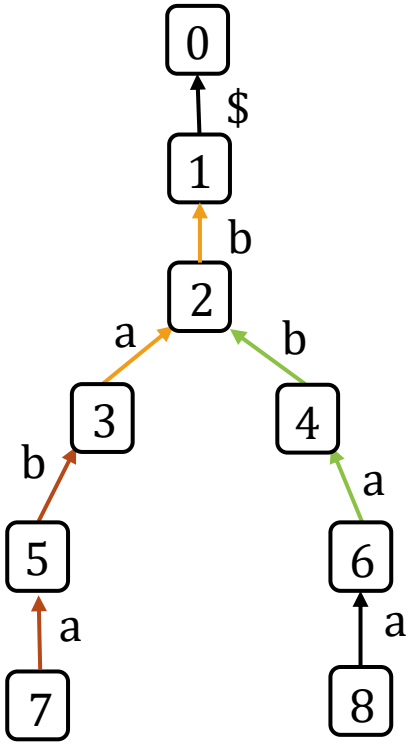


トライの部分パス検索問題

トライの部分パス検索問題

- 入力: トライ $\mathcal{T}$ とパターン文字列 P
- 出力: $\mathcal{T}$ 中で（根方向への）パス文字列が P となるパスの始点の一覧

右図のトライに対して $P = \mathrm{ab}$ で検索を行った場合、
出力は $\{3, 6, 7\}$



ユースケース: 文字列集合の部分文字列検索

文字列集合の部分文字列検索問題

- **入力: 文字列集合** $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ **とパターン文字列** P
- **出力:** $\mathcal{S}$ **中の** P **の出現位置の集合** $S\,\{\,(i,j) \mid S_i[j..j+|P|-1\} = P\,\}$

入力例: $\mathcal{S} = \{S_1, S_2, S_3, S_4\}, P = \text{correct}$

$S_1 = \text{correct}\text{ly}, S_2 = \text{in}\text{correct}, S_3 = \text{in}\text{correct}\text{ness}, S_4 = \text{disconnected}$

出力例: $\{(1,1),(2,3),(3,3)\}$

この問題は<u>トライの部分パス検索問題に帰着できる</u>

（S_1, S_2, S_3, S_4 を表す頂点が存在するようなトライに対して検索を行えばよい）

ユースケース: 文字列集合の部分文字列検索

この問題を索引を用いて解く場合、2種類の解法がある:

1. S の要素を全て繋げた文字列に対して適当な索引を作る
   - $O(\sum_{s \in S} |S|)$ だけ空間計算量がかかる
2. S を元に構築したトライに対して部分パス検索索引を作る
   - 空間計算量はトライのサイズに線形

→トライは重複する接頭辞を省略するので、より省領域で検索ができる！

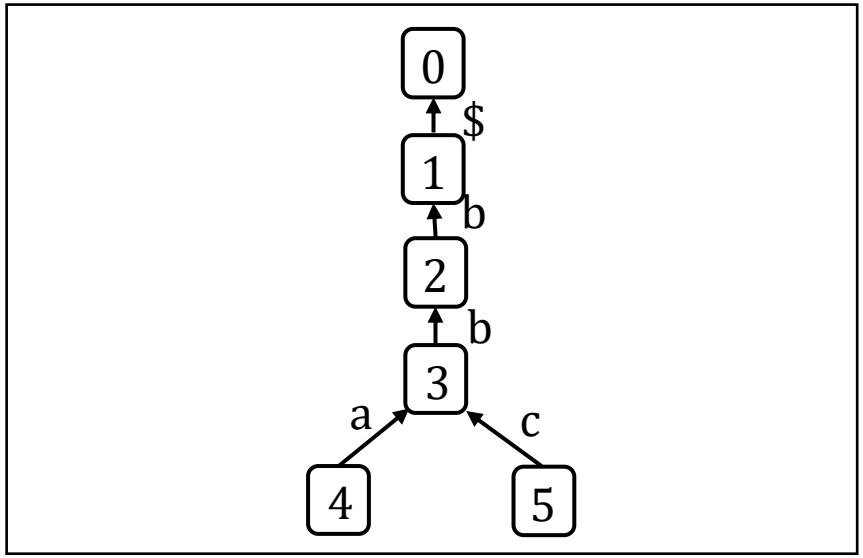
- 例: $S = \{\mathrm{a}^{100}\mathrm{b}, \mathrm{a}^{100}\mathrm{c}, \mathrm{a}^{100}\mathrm{d}\}$ に対してトライを作ると、前者の方法の約3倍省領域

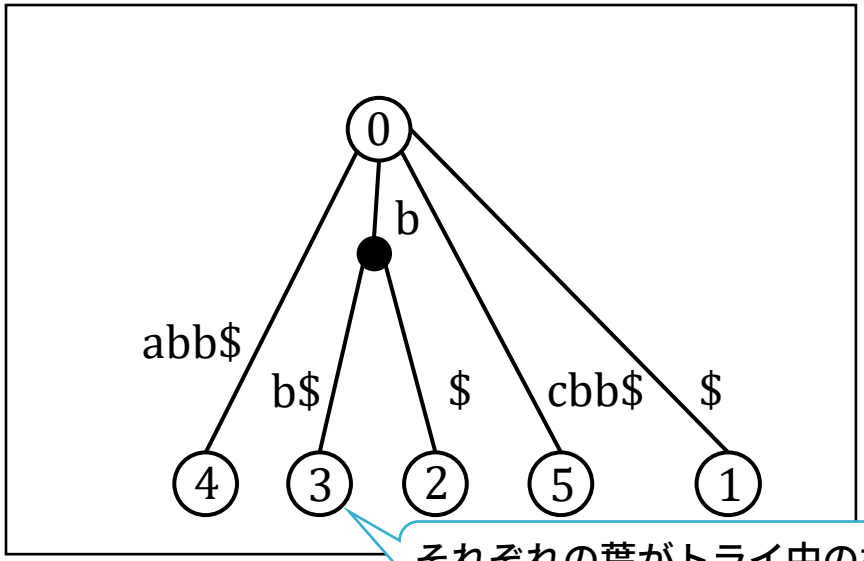
トライの接尾辞木

今回はトライの接尾辞木しか扱わないため、
以降では「接尾辞木」と略して表記する

トライの接尾辞木: トライの全頂点に対応する文字列を表すコンパクトトライ

- 通常の「文字列に対する接尾辞木」を一般化した定義
- トライの接尾辞木があればトライの部分文字列検索問題が高速に解ける
- 辺ラベルを工夫すれば、トライの頂点数に対して線形領域で表現できる



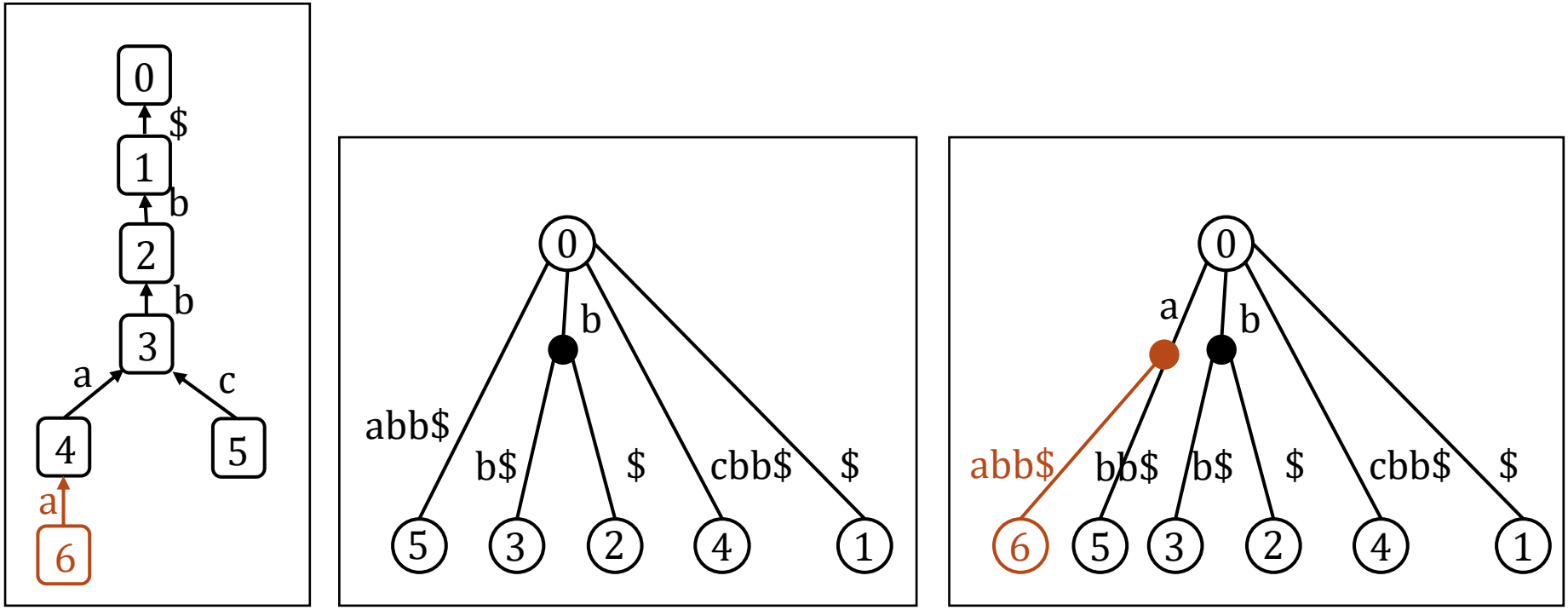


それぞれの葉がトライ中の根
でない頂点と一対一対応

トライとそれに対応する接尾辞木

トライへの頂点追加

下図のように、トライに新しい頂点を1つ追加する更新操作をした場合の接尾辞木の更新について考える

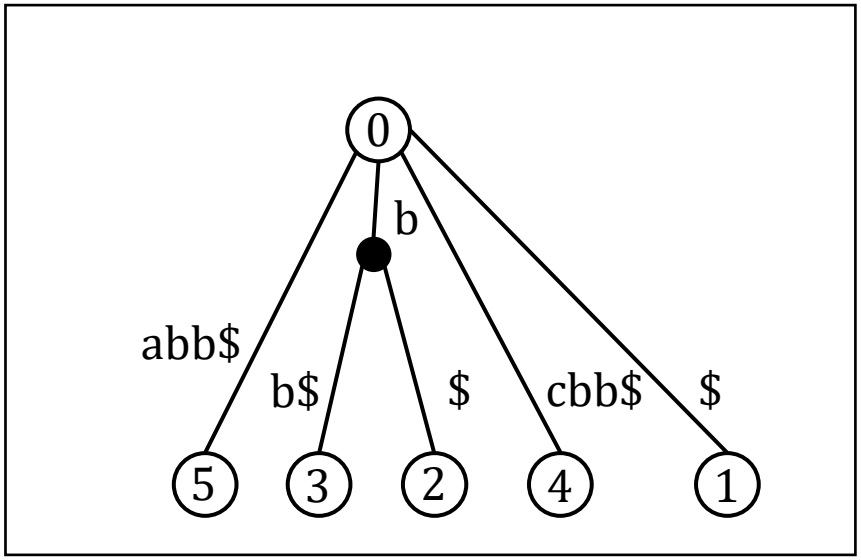


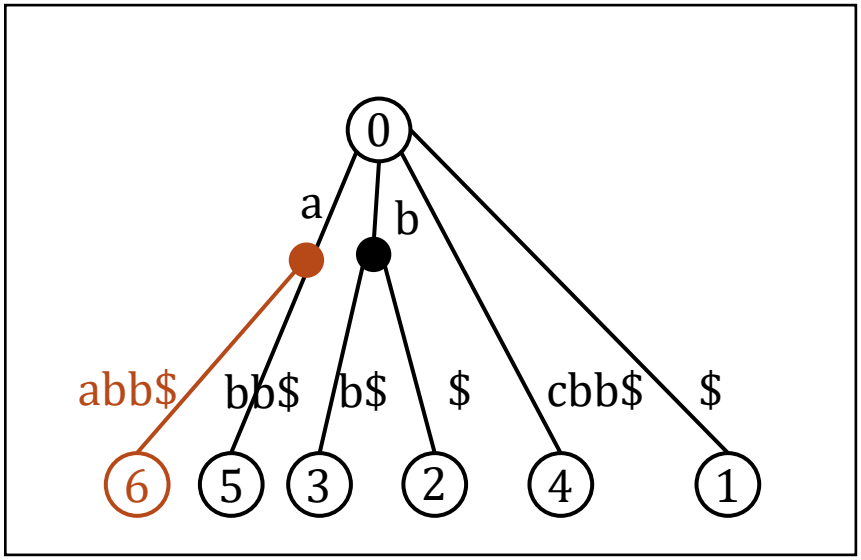
左図のトライに aabb$ を加えた場合の接尾辞木の更新（中央が更新前・右が更新後）

文字列追加時の接尾辞木の更新

やりたいこと: トライへの頂点追加時に接尾辞木を高速に更新したい

- トライの根以外の頂点と接尾辞木の葉が一対一対応するので、葉頂点を丁度一つ（と内部頂点を高々一つ）新しく増やす必要がある





前ページのトライに aabb$ を加えた場合の接尾辞木の更新（右が更新後）

Suffix Tree Oracle

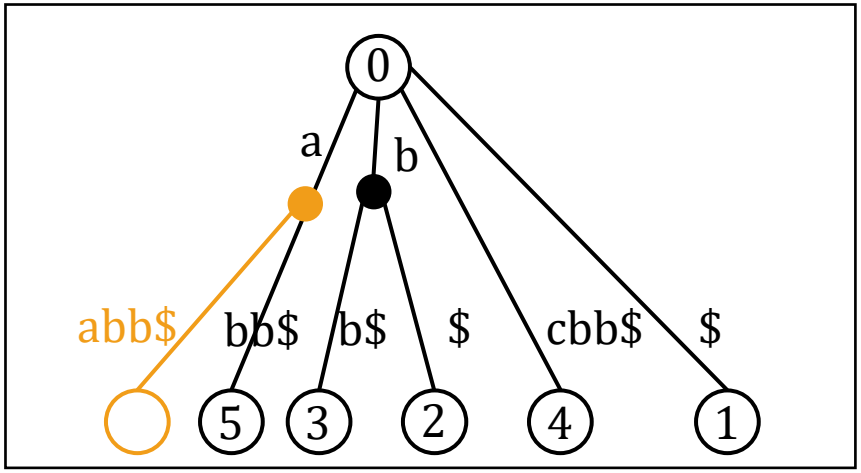
接尾辞木の更新のためには、Suffix Tree Oracleを高速に計算できればよい

以降では、Suffix Tree Oracleの高速な計算方法について議論する

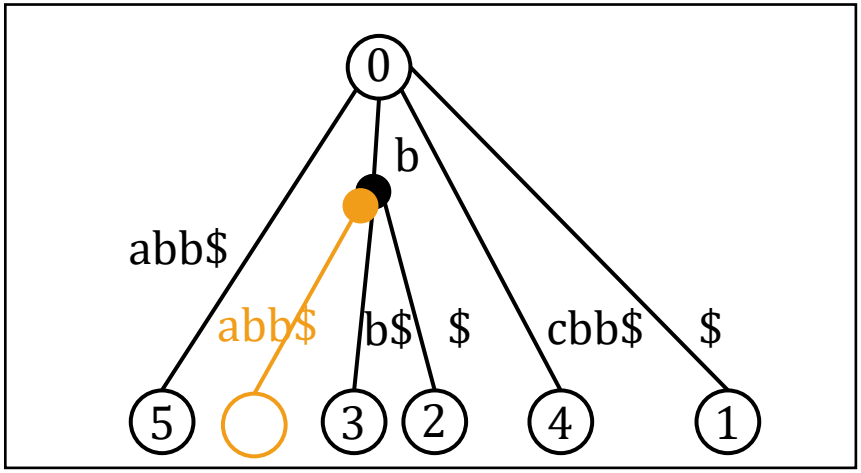
Suffix Tree Oracle

既存の文字列の先頭に1文字追加した文字列

- 入力: 接尾辞木, 追加する文字列 cS（S はトライのある頂点を表す文字列）
- 出力: 接尾辞木に cS を表すノードを追加する際、どの辺のどの位置を分割すべきか



aabb$ を加えた場合



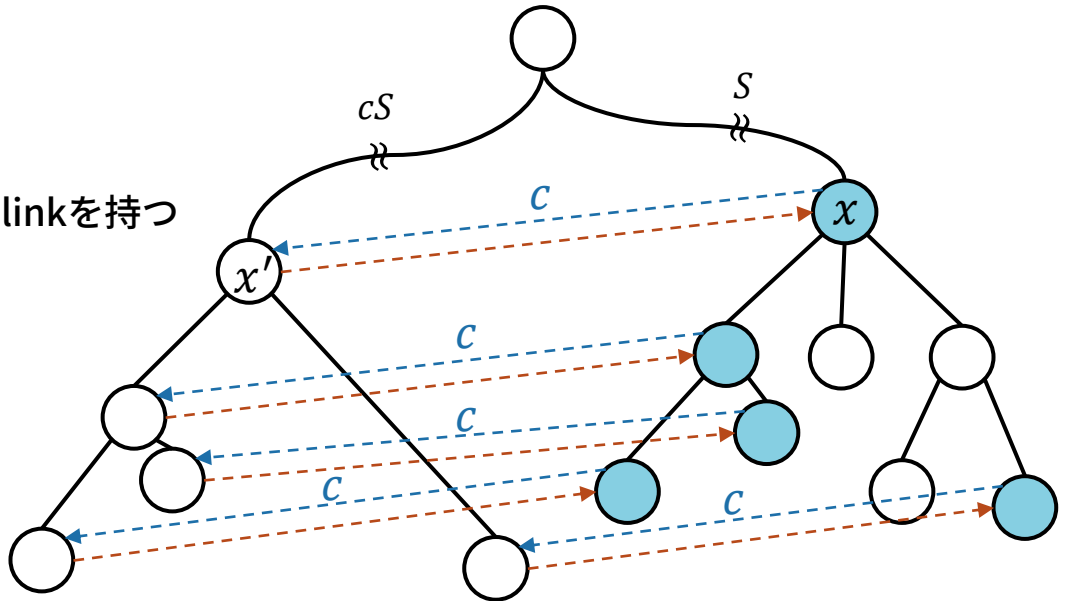
babb$ を加えた場合

Suffix link / Reverse suffix link

接尾辞木中のある頂点 x が文字列 S を表し、別の頂点 x' が S の先頭に1文字 c を追加した文字列 cS を表すとき、

- x' のsuffix link先は x
- x のラベル c でのreverse suffix link先は x'

水色の頂点: ラベル c のreverse suffix linkを持つ

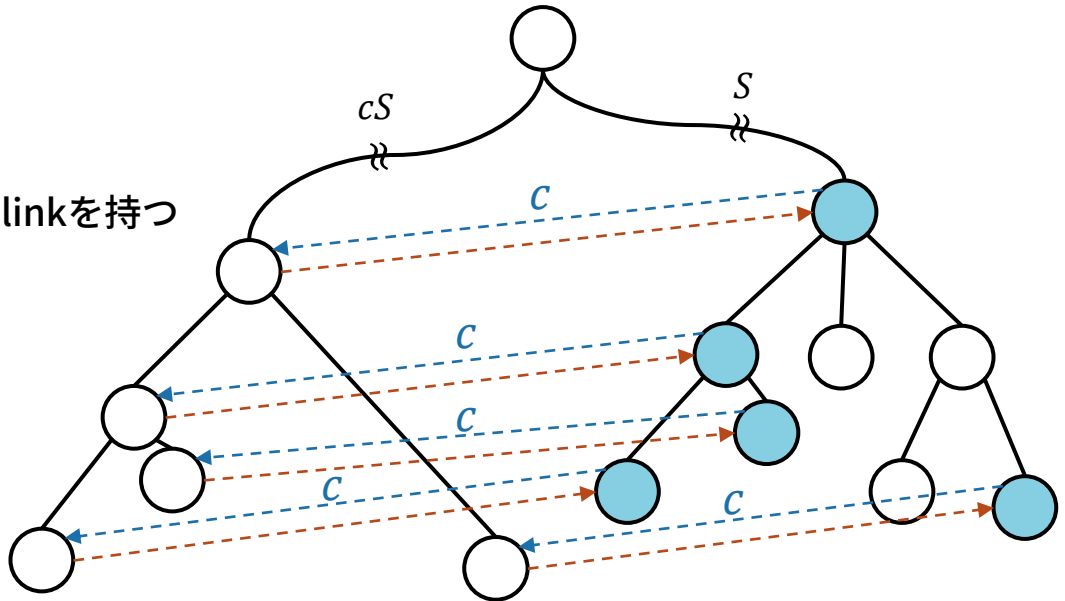


Suffix linkの基本性質

性質1: 接尾辞木の頂点 v がラベル c のreverse suffix linkを持つ必要十分条件は、以下の2つのどちらが成り立つこと

- v が葉であり、$\mathrm{str}(v') = c \cdot \mathrm{str}(v)$ を満たす葉頂点 v' が存在する
- v が内部頂点であり、ラベル c のreverse suffix linkを持つ v の子孫2つのLCA（最深共通祖先）になっている

水色の頂点: ラベル c のreverse suffix linkを持つ

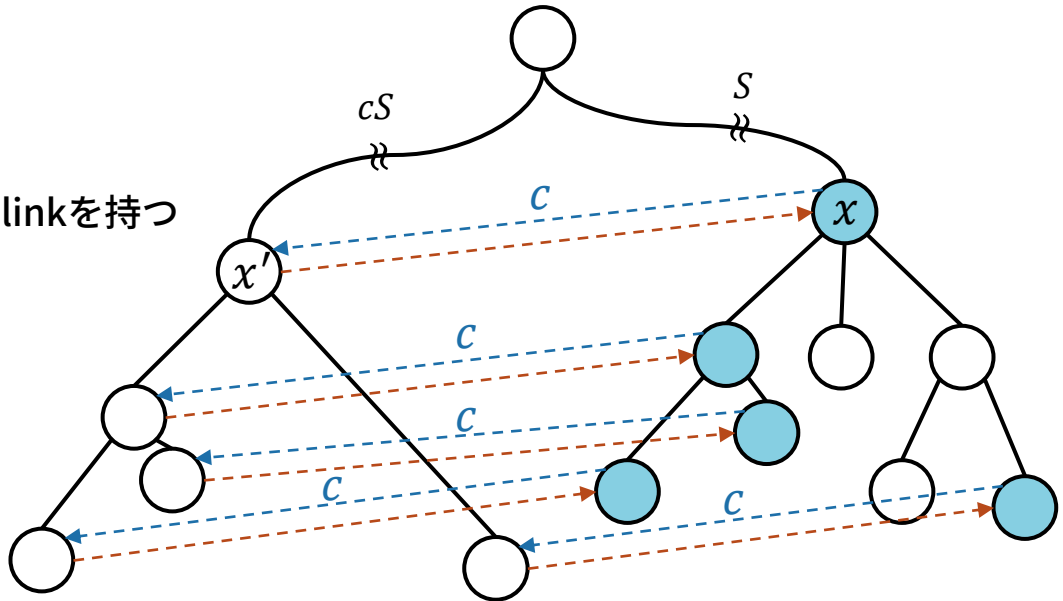


Suffix linkの基本性質

頂点 x のラベル c でのreverse suffix link先を x' とする

性質2: x' の部分木は、x の部分木から辺ラベル c のreverse suffix linkを持つ頂点のみを抜き出して縮約した木と同型になり、各頂点は一対一対応する

水色の頂点: ラベル c のreverse suffix linkを持つ



Suffix Tree Oracleの計算

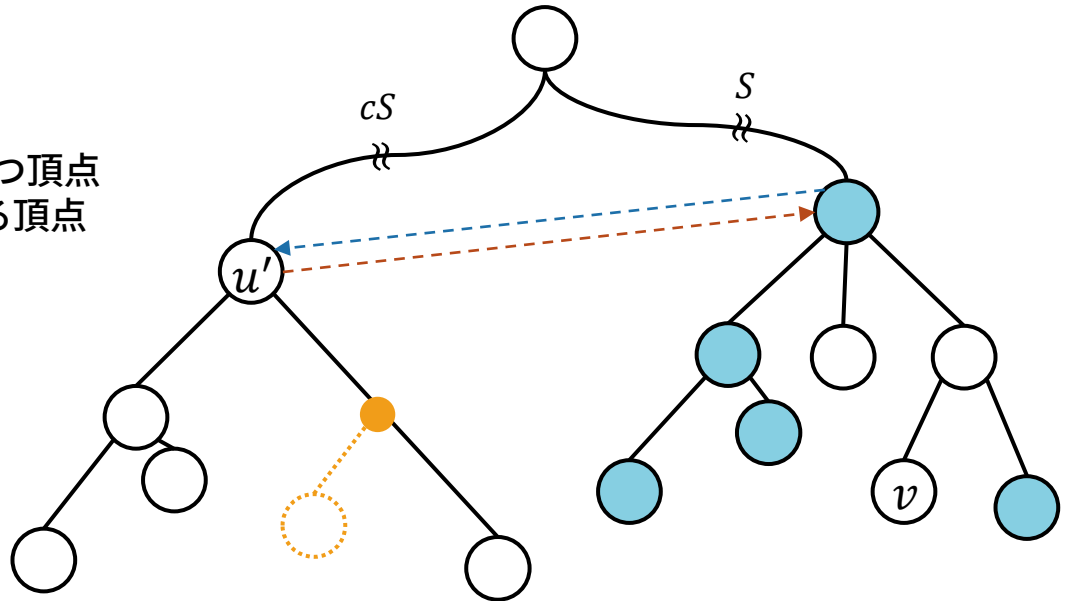
接尾辞木に追加する文字列は必ず
葉 v と文字 c を用いてこの形で表せる

文字列 $c \cdot \mathrm{str}(v)$ を追加する場合のSuffix Tree Oracleの計算を考える

Suffix Tree Oracleの出力となる辺の上側端点 u' を高速に求めたい

- 上側端点が分かれば、そこから適切な辺ラベルで子を探すことで下側端点も分かる
- 辺の分割位置（辺の何文字目で分割すれば良いか）については後で求める

水色: ラベル c のreverse suffix linkを持つ頂点
橙色: suffix tree oracleの出力と追加する頂点



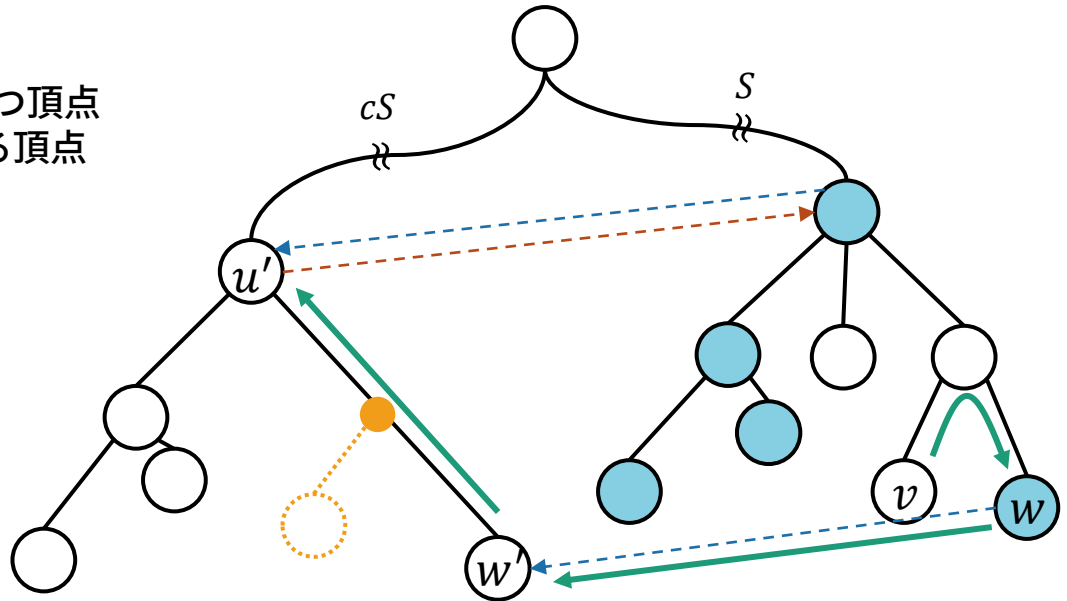
Suffix Tree Oracleの上端点の計算

性質3: ラベル c でのreverse suffix linkを持つ頂点のうち、木を深さ優先探索したときに v の次に訪問する頂点を w とおき、そのラベル c でのreverse suffix link先の頂点を w' とする。このとき、 w' もしくはその親はsuffix tree oracleの出力辺の上側端点 u' となる。

（ v の直前に訪問した頂点、としても同様に成り立つ）

水色: ラベル c のreverse suffix linkを持つ頂点
橙色: suffix tree oracleの出力と追加する頂点

右図: w' の親 $= u'$ になる例



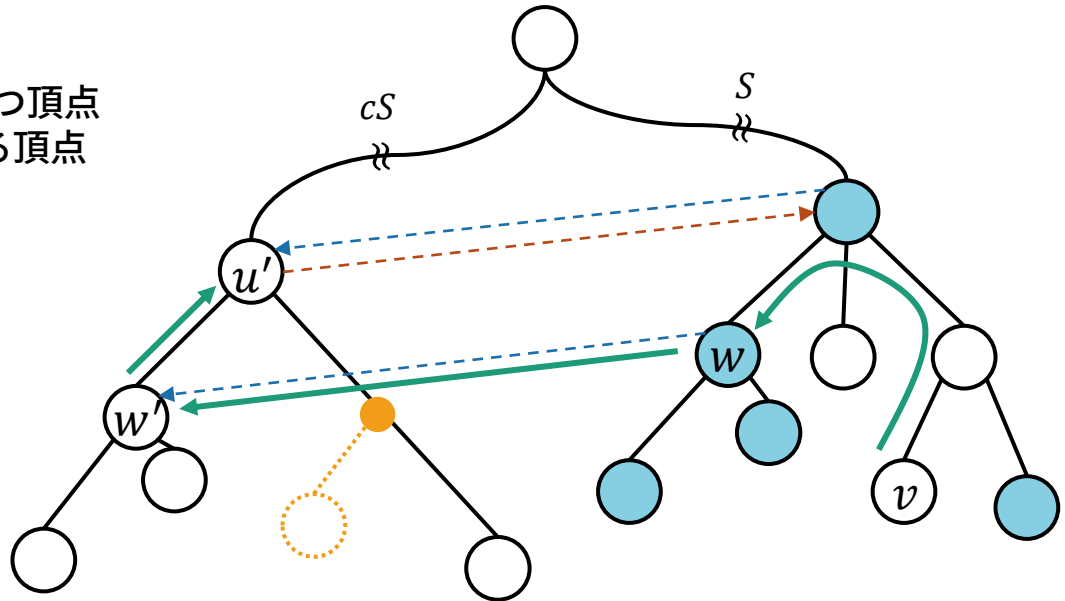
Suffix Tree Oracleの上端点の計算

性質3: ラベル c でのreverse suffix linkを持つ頂点のうち、木を深さ優先探索したときに v の次に訪問する頂点を w とおき、そのラベル c でのreverse suffix link先の頂点を w' とする。このとき、 w' もしくはその親はsuffix tree oracleの出力辺の上側端点 u' となる。

（ v の直前に訪問した頂点、としても同様に成り立つ）

水色: ラベル c のreverse suffix linkを持つ頂点
橙色: suffix tree oracleの出力と追加する頂点

右図: w' の親 $= u'$ になる例



Colored Predecessor Problemへの帰着

性質3より、「ラベル c のreverse suffix linkを持つような、深さ優先探索の訪問順において頂点 v の直後にある頂点」を高速に求められれば、suffix link oracleの出力となる辺が特定できる

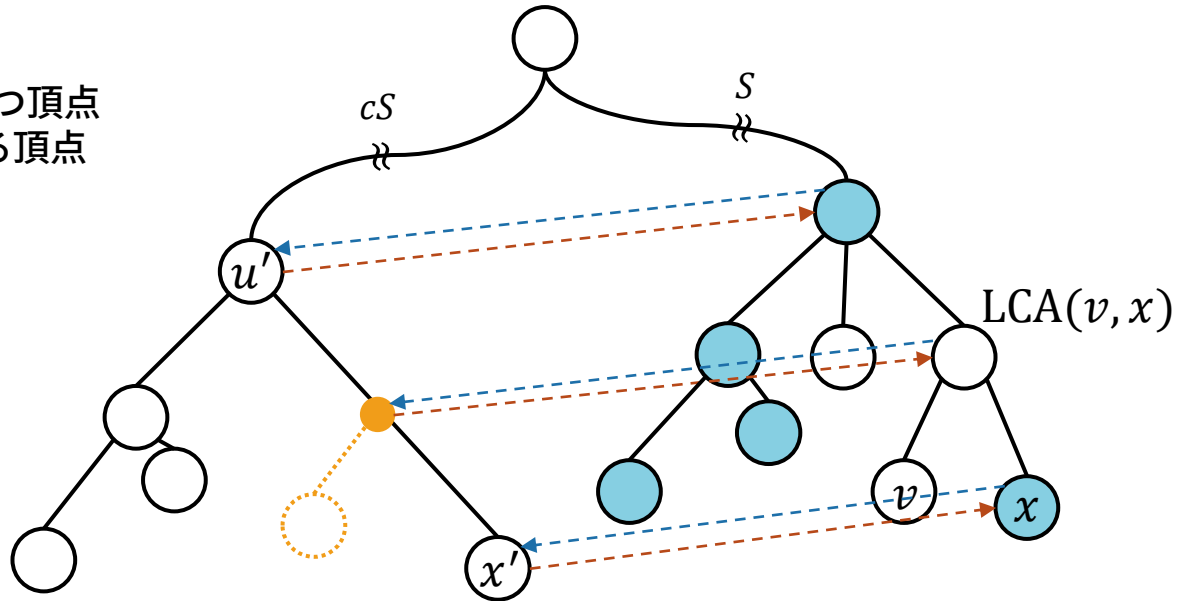
この問題は Colored Predecessor Problem と呼ばれ、要素数を n としたとき、要素追加とクエリ処理が最悪 $O\left(\frac{(\log\log n)^2}{\log\log\log n}\right)$ 時間で行えるデータ構造が存在する [Fischer and Gawrychowski, 2018]

辺の分割位置の計算

Suffix tree oracleの出力辺 (u', x')が分かっても辺の何文字目で分割すれば良いかはまだ分からないが、これは下の性質を用いて決定可能

性質4: x' のsuffix link先の頂点を x とすると、接尾辞木の更新によって新たにできる内部頂点のsuffix link先は v と x の最深共通祖先 $\mathrm{LCA}(v, x)$ となる

水色: ラベル c のreverse suffix linkを持つ頂点
橙色: suffix tree oracleの出力と追加する頂点

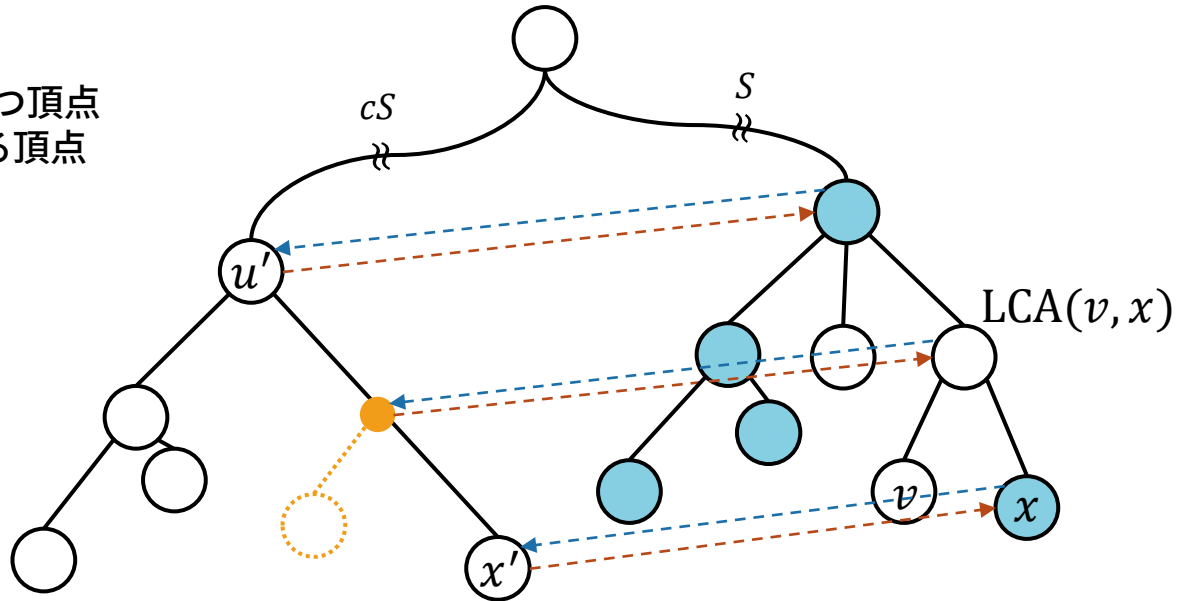


辺の分割位置の計算

性質4より、 $\mathrm{LCA}(v, x)$ が計算できれば、suffix link oracleの分割位置の深さを $\mathrm{depth}\big(\mathrm{LCA}(v, x)\big) + 1$ として計算できる

木の更新とLCAの計算が最悪定数時間・線形領域で行えるデータ構造が存在しているため [Cole and Hahiharan, 2005]、これを直接用いる

水色: ラベル c のreverse suffix linkを持つ頂点
橙色: suffix tree oracleの出力と追加する頂点



計算方法のまとめ

以下の手順でsuffix tree oracleの出力を計算できる:

1. 深さ優先探索順のColored Predecessorによって分割する辺を特定する
   - Colored Predecessorと辺の探索があり、計算量は $O\left(\frac{(\log\log n)^2}{\log\log\log n} + \log\sigma\right)$ 時間
2. LCAを求めて辺の分割位置を特定する
   - 定数時間で行える

Oracleの計算後は接尾辞木と関連するデータ構造を更新すれば良い

- 接尾辞木の更新は $O(\log\sigma)$ 時間で行える
- 関連するデータ構造の更新は $O\left(\frac{(\log\log n)^2}{\log\log\log n}\right)$ 時間

まとめ

- 接尾辞集合に対する接尾辞木の高速な更新手法を提案
  - Colored Predecessor ProblemやLCAに帰着して解いている
  - 既存手法より高速
- 今後の展望: さらなる高速化
  - 通常の文字列に対する接尾辞木にはさらに早い手法が存在
  - ↑の手法は直接一般化できないが、工夫の余地はありそう

	更新の時間計算量
[Breslauer, 1998]	$O(\sigma)$
[Funakoshi et al., 2024]	$O(\log n)$
提案手法	$O(\log \sigma + f)$

n: 接尾辞集合の要素数

σ: 文字の種類数

$$f = \frac{(\log \log n)^2}{\log \log \log n}$$

補足: 性質3の例外ケース

性質3: ラベル c でのreverse suffix linkを持つ頂点のうち、木を深さ優先探索したときに v の次に訪問する頂点を w とおき、そのreverse suffix link先の頂点を w' とする。このとき、w' もしくはその親はsuffix tree oracleの出力辺の上側端点 w' となる。

「直前の頂点」「直後の頂点」でそれぞれ w' を調べてこれらの深さを確認することで、$w' = u'$ になる例も対処できる

水色: ラベル c のreverse suffix linkを持つ頂点
橙色: suffix tree oracleの出力と追加する頂点

右図: $w' = u'$ になる例