Dynamic Fully-Compressed Suffix Trees

Luís M. S. Russo Gonzalo Navarro Arlindo L. Oliveira

INESC-ID/IST
{**lsr,aml**}**@algos.inesc-id.pt**

Dept. of Computer Science, University of Chile
gnavarro@dcc.uchile.cl

19th Annual Symposium on Combinatorial Pattern Matching

Outline

1. Motivation
   - The Problem We Studied
   - Previous Work and FCST's
   - Fully-Compressed Suffix Tree Basics
2. Dynamic FCST's
   - The problem
   - Dynamic CSA's
   - Updating the sampling
3. Conclusions
   - Summary

Suffix Trees are Important

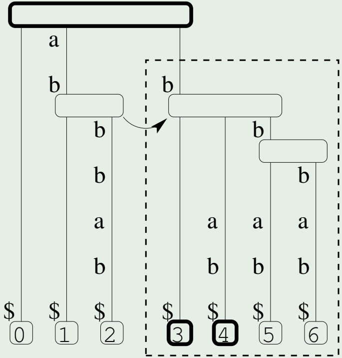
Suffix trees are important for several string problems:

- pattern matching
- longest common substring
- super maximal repeats
- bioinformatics applications
- etc

Suffix Trees are Important

27 min

Example (Suffix Tree for *abbbab*)



***A*:** 6 4 0 5 3 2 1

Representation Problems

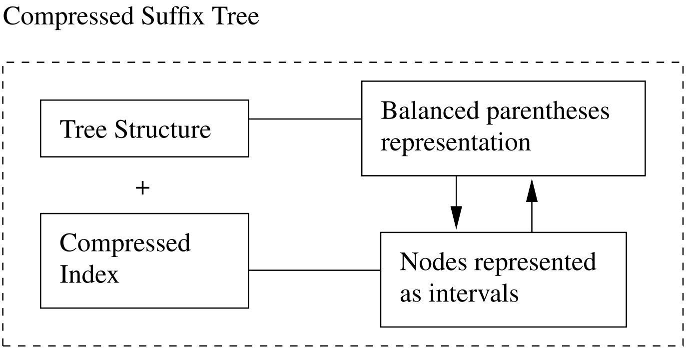
Problem (Suffix Trees need too much space)

Pointer based representations require $O(n \log n)$ bits.

This is much larger than the indexed string.
State of the art implementations require $[8, 10]n \log \sigma$ bits.

Compressed Representations 25 min

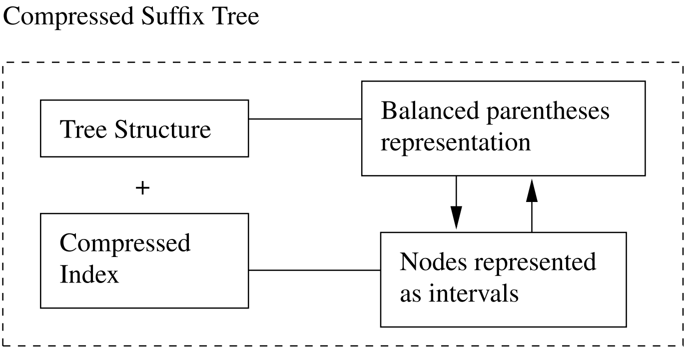
Sadakane proposed a way to represent compressed suffix trees, in $nH_k + 6n + o(n \log \sigma)$ bits.



Compressed Representations

25 min

A dynamic representation, by Chan *et al.*, requires $nH_k + \Theta(n) + o(n \log \sigma)$ bits and suffers an $O(\log n)$ slowdown.

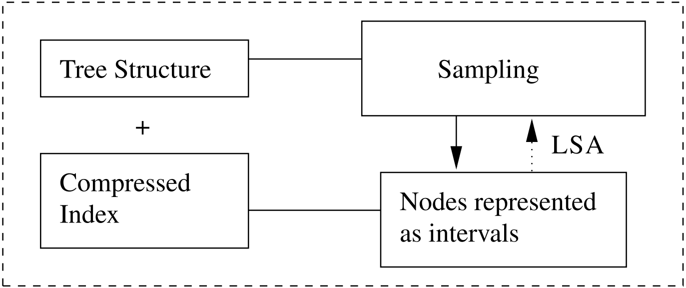


Compressed Representations

25 min

- The Fully-Compressed suffix tree representation requires only $nH_k + o(n \log \sigma)$ bits.
- The representation uses the following scheme:

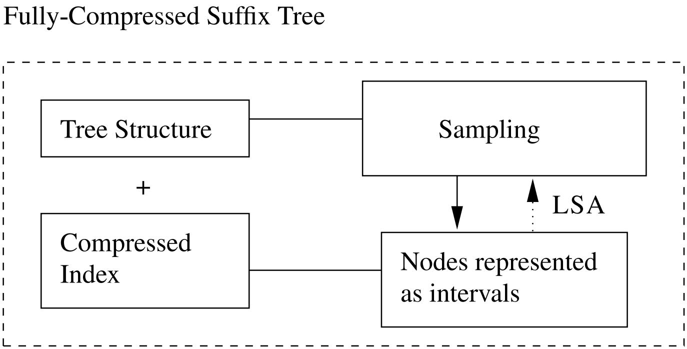
Fully-Compressed Suffix Tree



Compressed Representations

25 min

We present dynamic FCST's that require only $nH_k + o(n \log \sigma)$ bits with a $O(\log n)$ slowdown.



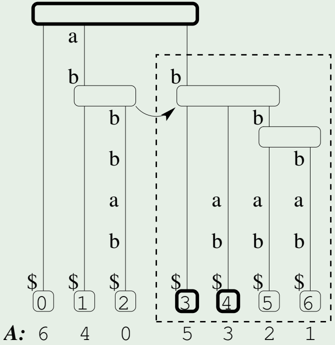
Node Representation

23 min

A node represented as an interval of leaves of a suffix tree.

Example

Interval [3, 6] represents node *b*.



Compressed Indexes

22 min

Compressed indexes are compressed representations of the leaves of a suffix tree.
Their success relies on:

- `Succinct structures`, based on RANK and SELECT.
- `Data compression`, that represent T in $O(uH_k)$ bits.

Examples

FM-index, Compressed Suffix Arrays, LZ-index, etc.

Sadakane used compressed suffix arrays.
We need a compressed index that supports ψ and LF.
For example the Alphabet-Friendly FM-Index.

Suffix Tree self-similarity

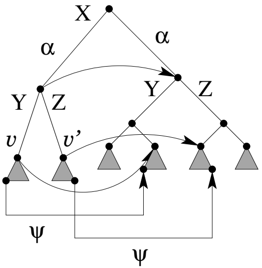
LCA and SLINK

21 min

Lemma

When $\mathrm{LCA}(v, v') \neq \textsc{Root}$ *we have that:*

$$\mathrm{S}\textsc{Link}(\mathrm{LCA}(v, v')) = \mathrm{LCA}(\mathrm{S}\textsc{Link}(v), \mathrm{S}\textsc{Link}(v'))$$



This self-similarity explains why we can store only some nodes.

Sampling

18 min

FCST's use a sampling such that in any sequence

- v
- $\textsc{SLink}(v)$
- $\textsc{SLink}(\textsc{SLink}(v))$
- $\textsc{SLink}(\textsc{SLink}(\textsc{SLink}(v)))$
- $\ldots$

of size δ there is at least one sampled node.

Fundamental lemma

17 min

Lemma

If $\text{SLINK}^r(\text{LCA}(v, v')) = \text{ROOT}$*, and let* $d = \min(\delta, r+1)$*.*
Then $\text{SDEP}(\text{LCA}(v, v')) =$

$$\max_{0 \le i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\}$$

Proof.

$$\begin{aligned}\text{SDEP}(\text{LCA}(v, v'))\\ &= i + \text{SDEP}(\text{SLINK}^i(\text{LCA}(v, v')))\\ &= i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\\ &\ge i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\end{aligned}$$

The last inequality is an equality for some $i \le d$. $\square$

Fundamental lemma

17 min

Lemma

If $\text{SLINK}^r(\text{LCA}(v, v')) = \text{ROOT}$, *and let* $d = \min(\delta, r + 1)$*.*
Then $\text{SDEP}(\text{LCA}(v, v')) =$

$$\max_{0 \leq i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\}$$

Proof.

$\text{SDEP}(\text{LCA}(v, v'))$

$$= i + \text{SDEP}(\text{SLINK}^i(\text{LCA}(v, v')))$$
$$= i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$$
$$\geq i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$$

The last inequality is an equality for some $i \leq d$. $\square$

Fundamental lemma

17 min

Lemma

If $\textsc{SLink}^r(\mathrm{LCA}(v, v')) = \textsc{Root}$, *and let* $d = \min(\delta, r + 1)$.
Then $\textsc{SDep}(\mathrm{LCA}(v, v'))?$

$$\max_{0 \le i < d}\{i + \textsc{SDep}(\mathrm{LCSA}(\textsc{SLink}^i(v), \textsc{SLink}^i(v')))\}$$

Proof.

$$\begin{aligned}
&\textsc{SDep}(\mathrm{LCA}(v, v')) \\
&\quad = i + \textsc{SDep}(\textsc{SLink}^i(\mathrm{LCA}(v, v'))) \\
&\quad = i + \textsc{SDep}(\mathrm{LCA}(\textsc{SLink}^i(v), \textsc{SLink}^i(v'))) \\
&\quad \ge i + \textsc{SDep}(\mathrm{LCSA}(\textsc{SLink}^i(v), \textsc{SLink}^i(v')))
\end{aligned}$$

The last inequality is an equality for some $i \le d$. □

Fundamental lemma

17 min

Lemma

If $\text{SLINK}^r(\text{LCA}(v, v')) = \text{ROOT}$, *and let* $d = \min(\delta, r+1)$.
Then $\text{SDEP}(\text{LCA}(v, v'))$?

$$\max_{0 \le i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\}$$

Proof.

$$\begin{aligned}
&\text{SDEP}(\text{LCA}(v, v')) \\
&\quad = i + \text{SDEP}(\text{SLINK}^i(\text{LCA}(v, v'))) \\
&\quad = i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
&\quad \ge i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))
\end{aligned}$$

The last inequality is an equality for some $i \le d$. □

Fundamental lemma

17 min

Lemma

If $\text{SL}\textsc{ink}^r(\text{LCA}(v, v')) = \text{R}\textsc{oot}$, *and let* $d = \min(\delta, r+1)$.
Then $\text{SD}\textsc{ep}(\text{LCA}(v, v')) \geq$
$$\max_{0 \leq i < d}\{i + \text{SD}\textsc{ep}(\text{LCSA}(\text{SL}\textsc{ink}^i(v), \text{SL}\textsc{ink}^i(v')))\}$$

Proof.

$$\begin{aligned}
&\text{SD}\textsc{ep}(\text{LCA}(v, v')) \\
&\quad = i + \text{SD}\textsc{ep}(\text{SL}\textsc{ink}^i(\text{LCA}(v, v'))) \\
&\quad = i + \text{SD}\textsc{ep}(\text{LCA}(\text{SL}\textsc{ink}^i(v), \text{SL}\textsc{ink}^i(v'))) \\
&\quad \geq i + \text{SD}\textsc{ep}(\text{LCSA}(\text{SL}\textsc{ink}^i(v), \text{SL}\textsc{ink}^i(v')))
\end{aligned}$$

The last inequality is an equality for some $i \leq d$. $\square$

Fundamental lemma

17 min

Lemma

If $\text{SLINK}^r(\text{LCA}(v, v')) = \text{ROOT}$, *and let* $d = \min(\delta, r + 1)$.
Then $\text{SDEP}(\text{LCA}(v, v')) =$

$$\max_{0 \le i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\}$$

Proof.

$$\begin{aligned}
&\text{SDEP}(\text{LCA}(v, v')) \\
&\quad = i + \text{SDEP}(\text{SLINK}^i(\text{LCA}(v, v'))) \\
&\quad = i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
&\quad \ge i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))
\end{aligned}$$

The last inequality is an equality for some $i \le d$. $\square$

Kernel Operations

12 min

With the previous lemma FCST's compute the following operations:

- $\text{SDEP}(v) = \text{SDEP}(\text{LCA}(v, v)) = \max_{0 \leq i < d}\{i + \text{SDEP}(\text{LCSA}(\psi^i(v_l), \psi^i(v_r)))\}$.
- $\text{LCA}(v, v') =$
  $\text{LF}(v[0..i-1],$
  $\text{LCSA}(\psi^i(\min\{v_l, v'_l\}), \psi^i(\max\{v_r, v'_r\}))),$
  for the i in the lemma.
- $\text{SLINK}(v) = \text{LCA}(\psi(v_l), \psi(v_r))$

Dynamic FCST's

11 min

Problem (FCST's are static)

How to insert or remove a text T from a FCST that is indexing a collection $\mathcal{C}$ of texts ?

- Use Weiner's algorithm or delete suffixes from the largest to the biggest.
- Update the CSA and the sampling.

Dynamic FCST's

11 min

Problem (FCST's are static)

How to insert or remove a text T from a FCST that is indexing a collection $\mathcal{C}$ of texts ?

- Use Weiner's algorithm or delete suffixes from the largest to the biggest.
- Update the CSA and the sampling.

Dynamic FCST's

11 min

Problem (FCST's are static)

How to insert or remove a text T from a FCST that is indexing a collection $\mathcal{C}$ of texts ?

- Use Weiner's algorithm or delete suffixes from the largest to the biggest.
- Update the CSA and the sampling.

Dynamic FCST's

10 min

Use a dynamic CSA's.

Theorem (Mäkinen, Navarro)

A dynamic CSA over a collection $\mathcal{C}$ can be stored in $nH_k(\mathcal{C}) + o(n \log \sigma)$ bits, with times $t = \Psi = O(((\log_\sigma \log n)^{-1} + 1) \log n)$, $\Phi = O((\log_\sigma \log n) \log^2 n)$, and inserting/deleting texts T in $O(|T|(t + \Psi))$.

Lets take a closer look at the sampling.

Dynamic FCST's

10 min

Use a dynamic CSA's.

Theorem (Mäkinen, Navarro)

A dynamic CSA over a collection $\mathcal{C}$ can be stored in $nH_k(\mathcal{C}) + o(n \log \sigma)$ bits, with times $t = \Psi = O(((\log_\sigma \log n)^{-1} + 1) \log n)$, $\Phi = O((\log_\sigma \log n) \log^2 n)$, and inserting/deleting texts T in $O(|T|(t + \Psi))$.

Lets take a closer look at the sampling.

Reverse tree

9 min

- How do we guarantee the sampling condition, with at most $O(n/\delta)$ nodes?
- We use a purely conceptual reverse tree.

Definition

The **reverse tree** $\mathcal{T}^R$ is the minimal labeled tree that, for every node v of a suffix tree, contains a node v^R denoting the reverse string of the path-label of v.

Reverse tree

9 min

- How do we guarantee the sampling condition, with at most $O(n/\delta)$ nodes?
- We use a purely conceptual reverse tree.

Definition

The **reverse tree** $\mathcal{T}^R$ is the minimal labeled tree that, for every node v of a suffix tree, contains a node v^R denoting the reverse string of the path-label of v.

Reverse tree

9 min

- How do we guarantee the sampling condition, with at most $O(n/\delta)$ nodes?
- We use a purely conceptual reverse tree.

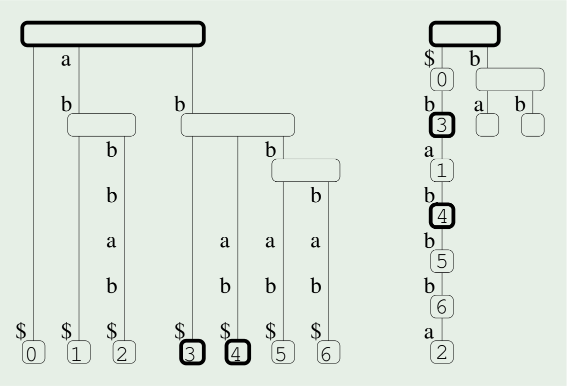
Definition

The **reverse tree** $\mathcal{T}^R$ is the minimal labeled tree that, for every node v of a suffix tree, contains a node v^R denoting the reverse string of the path-label of v.

Reverse tree

8 min

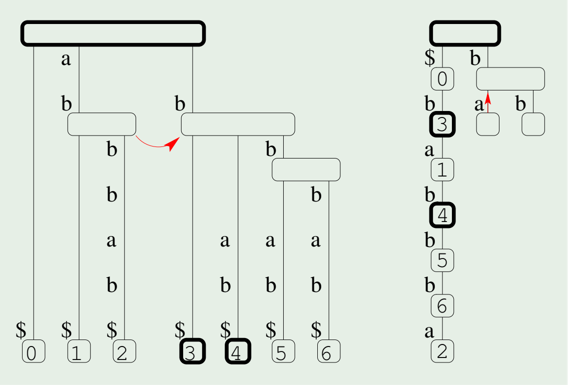
Example (Suffix Tree for *abbbab* and its reverse tree)



Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

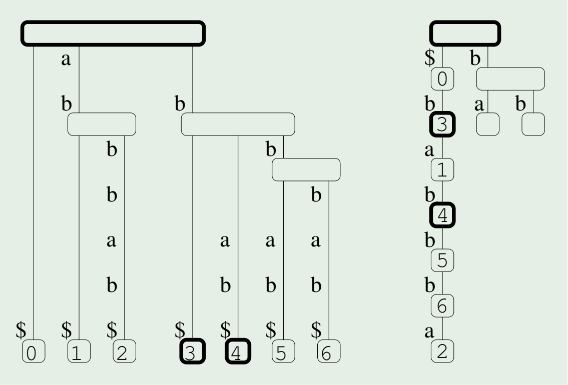


Note that the SLINK's correspond to moving upwards on the reverse tree.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

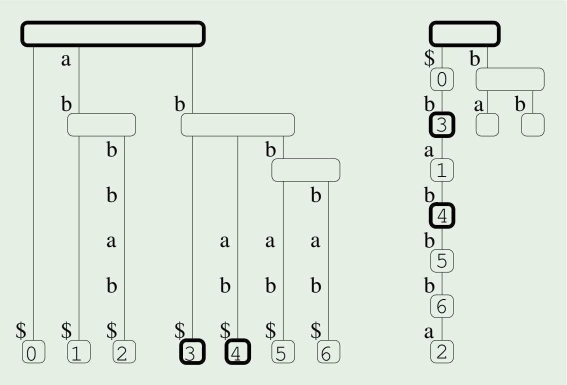


We sample the nodes for which $\text{TDEP}(v^R) \equiv_{\delta/2} 0$ and $\text{HEIGHT}(v^R) \geq \delta/2$.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

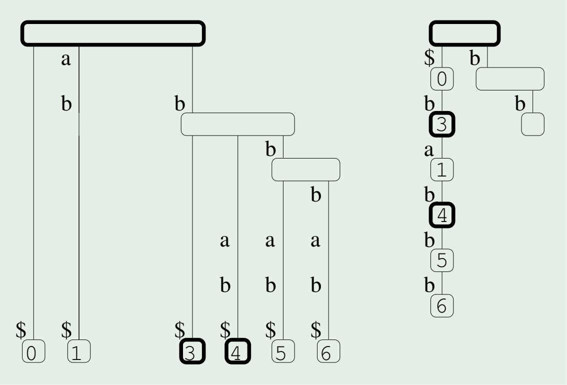


What happens when nodes are inserted or deleted ?

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

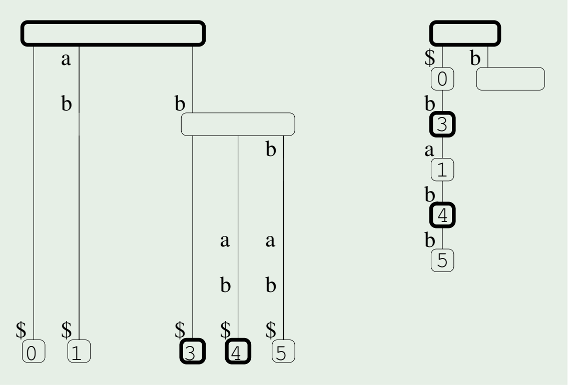


Only the leaves of the reverse tree change.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

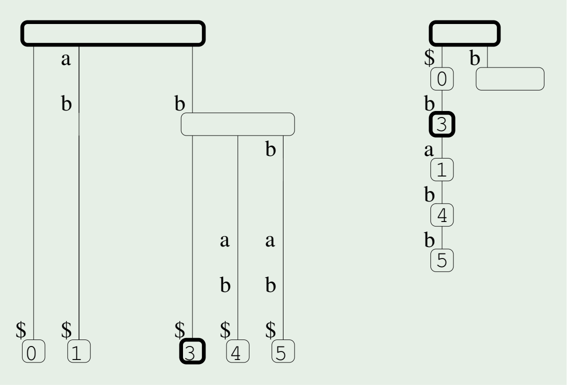


This sampling does not respect the $\textsc{Height}(v^R) \geq \delta/2$ condition.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

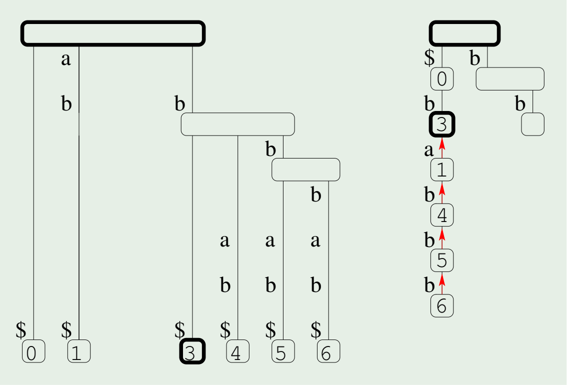


To insert a node we do an upwards scan and sample nodes if necessary.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

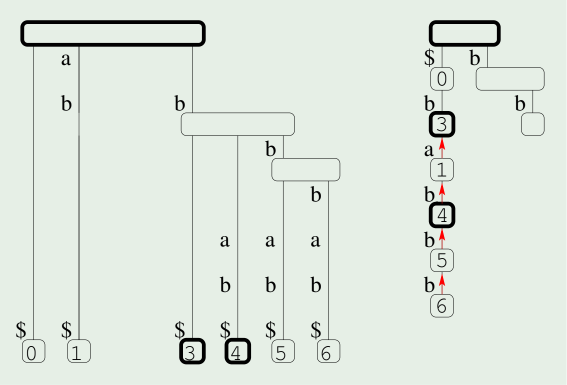


To insert a node we do an upwards scan and sample nodes if necessary.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

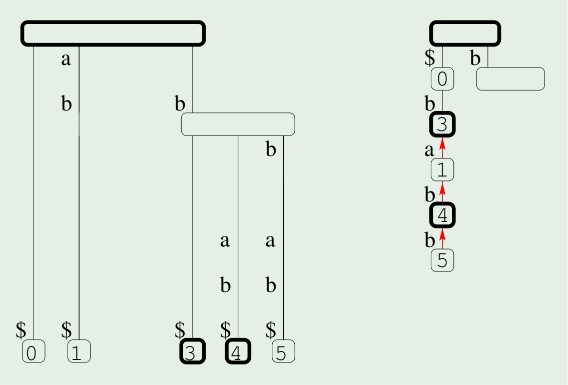


To insert a node we do an upwards scan and sample nodes if necessary.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)

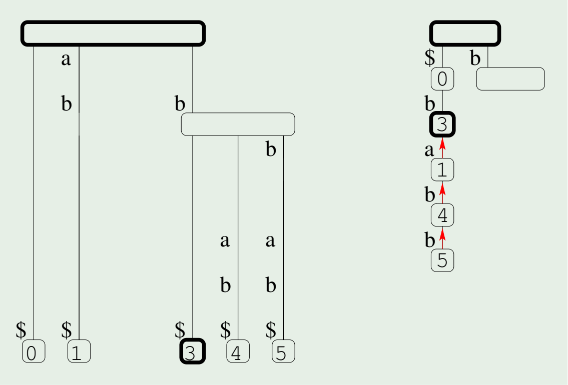


To delete a node we keep reference counters to guarantee that it is safe to unsample a node.

Reverse tree

8 min

Example (Suffix Tree for *abbbab* and its reverse tree)



To delete a node we keep reference counters to guarantee that it is safe to unsample a node.

Other contributions

2 min

- We study the problem of a changing $\lceil logn \rceil$.
- We give a new way to compute LSA.
- We obtain a generalized branching, that determines $v_1 . v_2$ for nodes v_1 and v_2 and can be computed directly over CSA's in the sample time as regular branching.

Other contributions

2 min

- We study the problem of a changing $\lceil log n \rceil$.
- We give a new way to compute LSA.
- We obtain a generalized branching, that determines $v_1 . v_2$ for nodes v_1 and v_2 and can be computed directly over CSA's in the sample time as regular branching.

Other contributions

2 min

- We study the problem of a changing $\lceil logn \rceil$.
- We give a new way to compute LSA.
- We obtain a generalized branching, that determines $v_1 . v_2$ for nodes v_1 and v_2 and can be computed directly over CSA's in the sample time as regular branching.

Summary

1 min

We presented dynamic fully-compressed suffix trees that:

- occupy $uH_k + o(u \log \sigma)$ bits.
- supports usual operations in a reasonable time.

Acknowledgments

0 min

- Veli Mäkinen and Johannes Fisher for pointing out the generalized branching problem.
- FCT grant SFRH/BPD/34373/2006 and project ARN, PTDC/EIA/67722/2006.
- Millennium Institute for Cell Dynamics and Biotechnology, Grant ICM P05-001-F, Mideplan, Chile.

Acknowledgments

0 min

Thanks for listening.