Succinct Data Structures

Luís M. S. Russo

Data Storm Big Data Summer School 2014

Outline

1. Motivation
   - Trees take too much space
   - Suffix Trees
   - Compressed Representations
2. FCST Representation
   - Performance
   - The kernel Operations
   - Further Operations
3. Conclusions
   - Summary

Implementing trees with pointers

```
typedef struct node {
Item item;
struct node *l;
struct node *r;
} *link;
```

- Requires 2 x 32 bits per node
- or 2 x 64, on 64 bit machine

Implementing trees with pointers

```
typedef struct node {
Item item;
struct node *l;
struct node *r;
} *link;
```

- Requires 2 x 32 bits per node
- or 2 x 64, on 64 bit machine

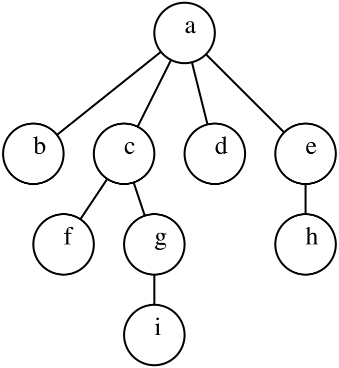
Implementing trees with pointers

```
typedef struct node {
Item item;
struct node *l;
struct node *r;
} *link;
```

- Requires 2 x 32 bits per node
- or 2 x 64, on 64 bit machine

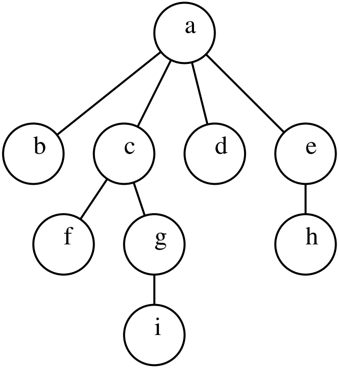
Succinct Data Structures

- Representantions that require optimal space.
- What is the minimal number of bits to represent a tree?
- With 2 bits per node, using parenthesis.
- (a(b)(c(f)(g(i))(d)(e(h)))



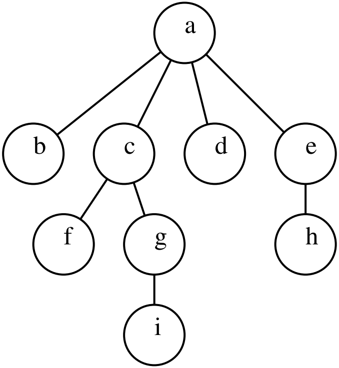
Succinct Data Structures

- Representantions that require optimal space.
- What is the minimal number of bits to represent a tree?
- With 2 bits per node, using parenthesis.
- (a(b)(c(f)(g(i))(d)(e(h)))



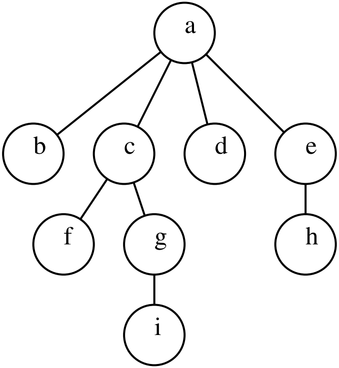
Succinct Data Structures

- Representantions that require optimal space.
- What is the minimal number of bits to represent a tree?
- With 2 bits per node, using parenthesis.
- (a(b)(c(f)(g(i))(d)(e(h)))



Succinct Data Structures

- Representantions that require optimal space.
- What is the minimal number of bits to represent a tree?
- With 2 bits per node, using parenthesis.
- (a(b)(c(f)(g(i))(d)(e(h)))

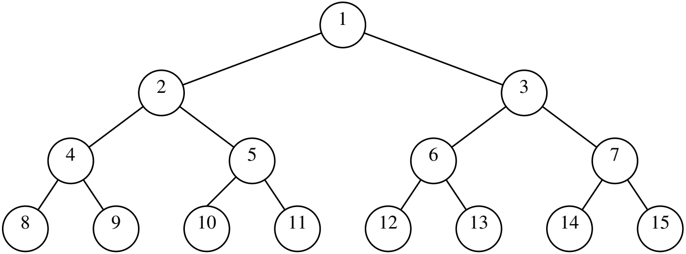


Succinct Data Structures

- We still want to navigate to child and parent.
- Recall heaps.
- Child(i) = 2i ; Parent(i) = i/2 ;

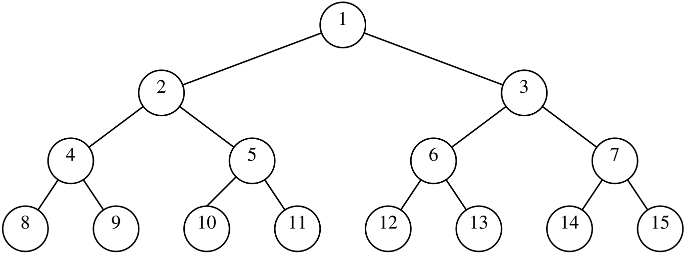
Succinct Data Structures

- We still want to navigate to child and parent.
- Recall heaps.
- Child(i) = 2i ; Parent(i) = i/2 ;



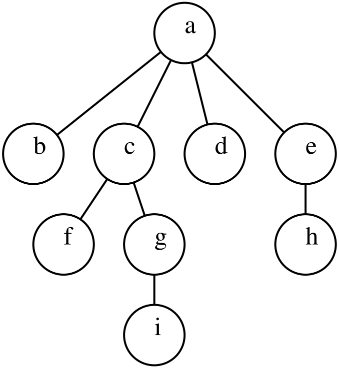
Succinct Data Structures

- We still want to navigate to child and parent.
- Recall heaps.
- Child(i) = 2i ; Parent(i) = i/2 ;



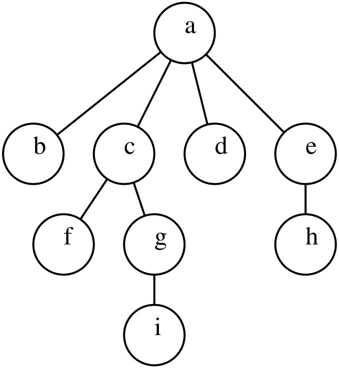
Succinct Data Structures

- Level-Order Unary Degree Sequence (LOUDS) representation
- 1a01111b0c011d0e01f0g01h0i0
- Store only the bits, not the letters.



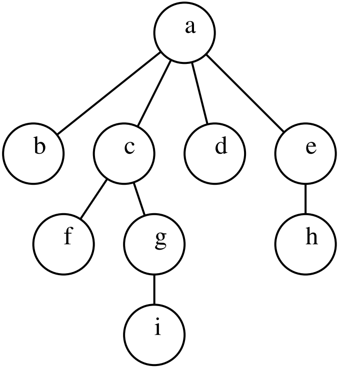
Succinct Data Structures

- Level-Order Unary Degree Sequence (LOUDS) representation
- 1a01111b0c011d0e01f0g01h0i0
- Store only the bits, not the letters.



LOUDS

- 1a01111b0c011d0e01f0g01h0i0
- parent(i) = select1(rank0(i)) ; child = select0(rank1(i)) ;
- rank1 counts the numbers of 1's
- select1 finds the *i*-th 1.



Rank and Select

Rank and select can be computed efficiently.
For Rank use sparse arrays for higher bits.

Bitmap	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0
Rank1	1	1	2	3	4	5	5	5	6	7	7	7	8	8	8
HiBits	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

- Hence Rank can be computed in $O(1)$ in $n + o(n)$ bits.
- Select can also be computed in $O(1)$.
- Binary searches are used in practice, $O(\log n)$ time.
- Rank and Select are the building blocks of Succinct Data Structures.

Rank and Select

Rank and select can be computed efficiently.
For Rank use sparse arrays for higher bits.

Bitmap	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0
Rank1	1	1	2	3	4	5	5	5	6	7	7	7	8	8	8
HiBits	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

- Hence Rank can be computed in $O(1)$ in $n + o(n)$ bits.
- Select can also be computed in $O(1)$.
- Binary searches are used in practice, $O(\log n)$ time.
- Rank and Select are the building blocks of Succinct Data Structures.

Rank and Select

Rank and select can be computed efficiently.
For Rank use sparse arrays for higher bits.

Bitmap	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0
Rank1	1	1	2	3	4	5	5	5	6	7	7	7	8	8	8
HiBits	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

- Hence Rank can be computed in $O(1)$ in $n + o(n)$ bits.
- Select can also be computed in $O(1)$.
- Binary searches are used in practice, $O(\log n)$ time.
- Rank and Select are the building blocks of Succinct Data Structures.

Rank and Select

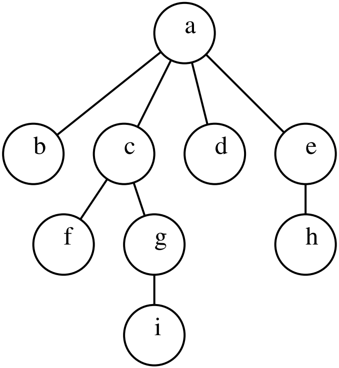
Rank and select can be computed efficiently.
For Rank use sparse arrays for higher bits.

Bitmap	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0
Rank1	1	1	2	3	4	5	5	5	6	7	7	7	8	8	8
HiBits	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

- Hence Rank can be computed in $O(1)$ in $n + o(n)$ bits.
- Select can also be computed in $O(1)$.
- Binary searches are used in practice, $O(\log n)$ time.
- Rank and Select are the building blocks of Succinct Data Structures.

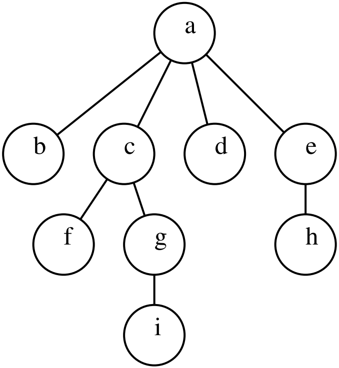
Lowest Common Ancestor

- LOUDS is a functional tree representation.
- How about fancier operations ? Lowest Common Ancestors.



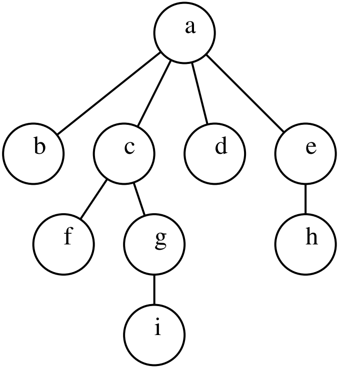
Lowest Common Ancestor

- LOUDS is a functional tree representation.
- How about fancier operations ? Lowest Common Ancestors.



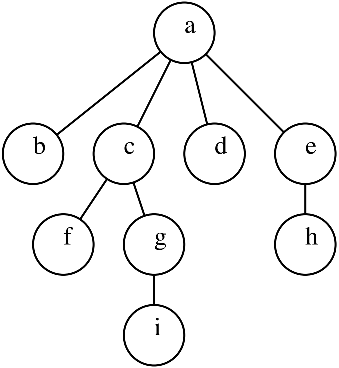
Lowest Common Ancestor

- Let us go back to balanced parenthesis.
- (a(b)(c(f)(g(i))(d)(e(h)))
- 1a2b12c3f23g4i3212d12e3h210



Lowest Common Ancestor

- Reduce LCA to the minimum in an interval
- (a(b)(c([f)(g(i]))(d)(e(h)))
- 1a2b12c3[f23g4i]3212d12e3h210



Range Minimum Queries

- Preprocess a sequence, and find interval minimum in $O(1)$
- 12123[234]3212123210
- Using an $O(n^2)$ table, too much space
- Scanning, too slow.

Range Minimum Queries

- Preprocess a sequence, and find interval minimum in $O(1)$
- 12123[234]3212123210
- Using an $O(n^2)$ table, too much space
- Scanning, too slow.

Range Minimum Queries

- Preprocess a sequence, and find interval minimum in $O(1)$
- 12123[234]3212123210
- Using an $O(n^2)$ table, too much space
- Scanning, too slow.

Range Minimum Queries

- Use a table for queries of size 2^i.
- Takes $O(n \log^2 n)$ bits, and $O(1)$ query time.
- Drop $O(\log n)$ factors by sampling.

S	1	2	1	2	3	2	3	4	3	2	1	2	1	2	3	2	1
2	1	1	1	2	2	2	3	3	2	1	1	1	1	2	2	1	0
4	1	1	1	2	2	2	2	1	1	1	1	1	1	1	0	0	0
8	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Range Minimum Queries

- Use a table for queries of size 2^i.
- Takes $O(n \log^2 n)$ bits, and $O(1)$ query time.
- Drop $O(\log n)$ factors by sampling.

S	1	2	1	2	3	2	3	4	3	2	1	2	1	2	3	2	1
2	1	1	1	2	2	2	3	3	2	1	1	1	1	2	2	1	0
4	1	1	1	2	2	2	2	1	1	1	1	1	1	1	0	0	0
8	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Range Minimum Queries

- Use a table for queries of size 2^i.
- Takes $O(n \log^2 n)$ bits, and $O(1)$ query time.
- Drop $O(\log n)$ factors by sampling.

S	1	2	1	2	3	2	3	4	3	2	1	2	1	2	3	2	1	0
2	1	1	1	2	2	2	3	3	2	1	1	1	1	2	2	1	0	0
4	1	1	1	2	2	2	2	1	1	1	1	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

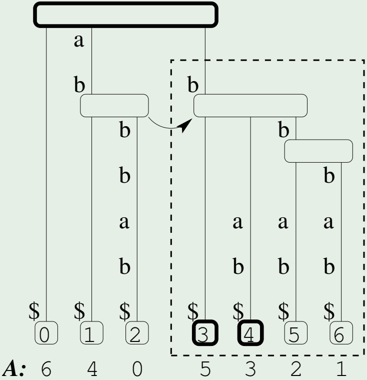
Suffix Trees are Important

Suffix trees are important for several string problems:

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

Suffix Trees are Important

Example (Suffix Tree for *abbbab*)



Representation Problems

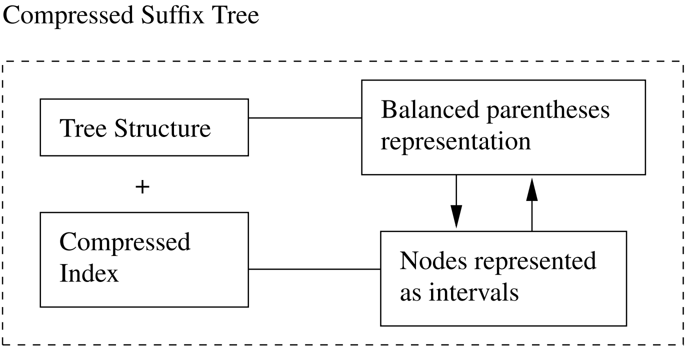
Problem (Suffix Trees need too much space)

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

This is much larger than the indexed string.
State of the art implementations require $[8, 10]n \times 32$ bits.

Compressed Representations

Sadakane proposed a new way to represent suffix trees.



Compressed Suffix Tree

Tree Structure

+

Compressed Index

Balanced parentheses representation

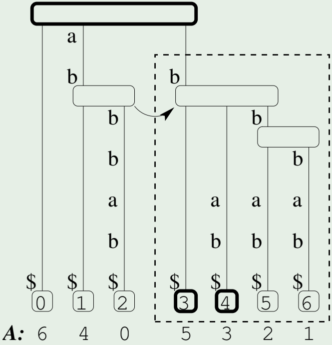
Nodes represented as intervals

Node Representation

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

Example

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



Compressed Indexes

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.

Overall Performance

$\sigma = O(\mathrm{polylog}(n))$

	Sadakane's	FCST
Space in bits	$nH_k + 6n + o(n \log \sigma)$	$nH_k + o(n \log \sigma)$
SDEP/LOCATE	$\log n \log \log n$	$\log n \log \log n$
COUNT/ANCESTOR	1	1
PARENT/FCHILD/	1	$\log n \log \log n$
SLINK	1	$\log n \log \log n$
SLINK^i	$\log n \log \log n$	$\log n \log \log n$
LETTER(v, i)	$\log n \log \log n$	$\log n \log \log n$
LCA	1	$\log n \log \log n$
CHILD	$(\log \log n) \log n$	$(\log \log n)^2 \log_\sigma n$
TDEP	1	$(\log n \log \log n)^2$
TLAQ	1	$(\log n \log \log n)^2$
SLAQ	—	$\log n \log \log n$
WEINERLINK	1	1

Overall Performance

$\sigma = O(\mathrm{polylog}(n))$

	Sadakane's	FCST
Space in bits	$nH_k + 6n + o(n \log \sigma)$	$nH_k + o(n \log \sigma)$
SDep/Locate	$\log n \log \log n$	$\log n \log \log n$
Count/Ancestor	1	1
Parent/FChild/	1	$\log n \log \log n$
SLink	1	$\log n \log \log n$
SLink^i	$\log n \log \log n$	$\log n \log \log n$
Letter(v, i)	$\log n \log \log n$	$\log n \log \log n$
LCA	1	$\log n \log \log n$
Child	$(\log \log n) \log n$	$(\log \log n)^2 \log_\sigma n$
TDep	1	$(\log n \log \log n)^2$
TLAQ	1	$(\log n \log \log n)^2$
SLAQ	—	$\log n \log \log n$
WeinerLink	1	1

Overall Performance

$\sigma = O(\text{polylog}(n))$

	Sadakane's	FCST
Space in bits	$nH_k + 6n + o(n \log \sigma)$	$nH_k + o(n \log \sigma)$
SDEP/LOCATE	$\log n \log \log n$	$\log n \log \log n$
COUNT/ANCESTOR	1	1
PARENT/FCHILD/	1	$\log n \log \log n$
SLINK	1	$\log n \log \log n$
SLINK^i	$\log n \log \log n$	$\log n \log \log n$
LETTER(v, i)	$\log n \log \log n$	$\log n \log \log n$
LCA	1	$\log n \log \log n$
CHILD	$(\log \log n) \log n$	$(\log \log n)^2 \log_\sigma n$
TDEP	1	$(\log n \log \log n)^2$
TLAQ	1	$(\log n \log \log n)^2$
SLAQ	—	$\log n \log \log n$
WEINERLINK	1	1

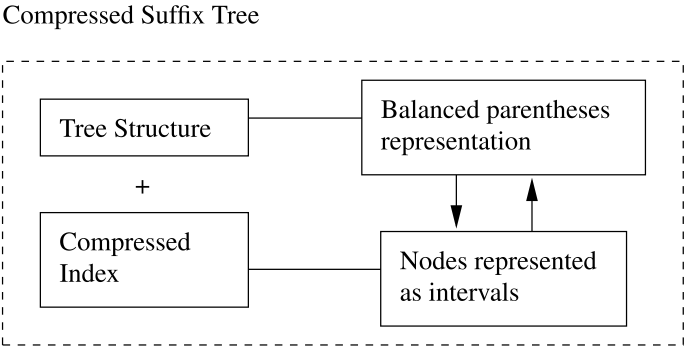
Overall Performance

$\sigma = O(\mathrm{polylog}(n))$

	Sadakane's	FCST
Space in bits	$nH_k + 6n + o(n \log \sigma)$	$nH_k + o(n \log \sigma)$
SDEP/LOCATE	$\log n \log \log n$	$\log n \log \log n$
COUNT/ANCESTOR	1	1
PARENT/FCHILD/	1	$\log n \log \log n$
SLINK	1	$\log n \log \log n$
SLINK^i	$\log n \log \log n$	$\log n \log \log n$
LETTER(v, i)	$\log n \log \log n$	$\log n \log \log n$
LCA	1	$\log n \log \log n$
CHILD	$(\log \log n) \log n$	$(\log \log n)^2 \log_\sigma n$
TDEP	1	$(\log n \log \log n)^2$
TLAQ	1	$(\log n \log \log n)^2$
SLAQ	—	$\log n \log \log n$
WEINERLINK	1	1

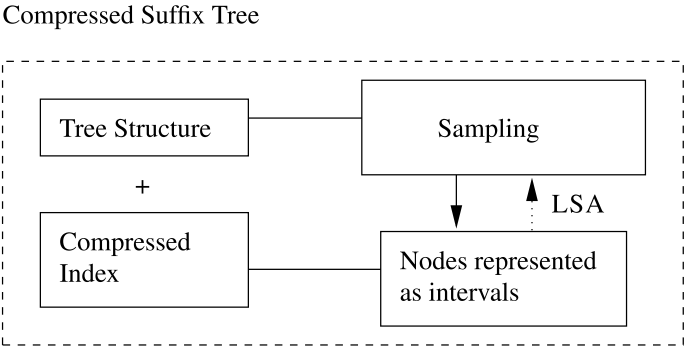
Sampling

We use sampling instead of balanced parentheses.



Sampling

We use sampling instead of balanced parentheses.



Compressed Suffix Tree

Tree Structure

\+

Compressed Index

Sampling

Nodes represented as intervals

Sampling

The sampling has the property that in any sequence

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

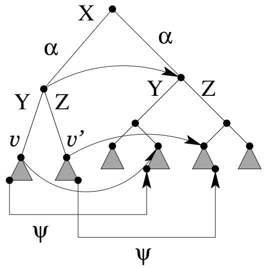
of size δ there is at least one sampled node.

LCA and SLINK

Lemma

When LCA$(v, v') \neq$ ROOT *we have that:*

$$\text{SLINK}(\text{LCA}(v, v')) = \text{LCA}(\text{SLINK}(v), \text{SLINK}(v'))$$



Fundamental lemma

Lemma

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

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

Proof.

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

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

Fundamental lemma

Lemma

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

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

Proof.

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

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

Fundamental lemma

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

Lemma

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

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 $i \leq d$. ☐

Fundamental lemma

Lemma

If $\textsc{SLink}^r(\mathrm{LCA}(v, v')) = \textsc{Root}$*, and let* $d = \min(\delta, r+1)$.
Then $\textsc{SDep}(\mathrm{LCA}(v, v')) \geq$
$$\max_{0 \leq i < d}\{i + \textsc{SDep}(\mathrm{LCSA}(\textsc{SLink}^i(v), \textsc{SLink}^i(v')))\}$$

Proof.

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

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

Fundamental lemma

Lemma

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

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

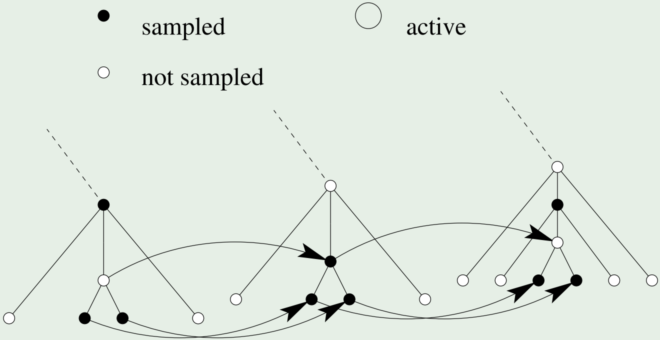
Proof.

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

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

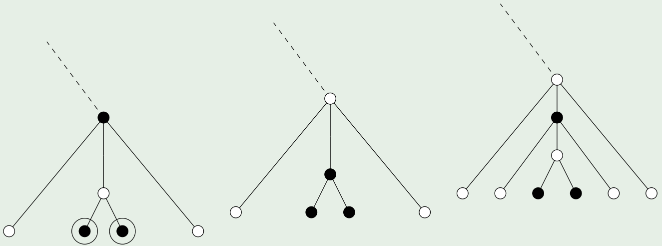
Fundamental lemma

Example ($\delta = 3$)



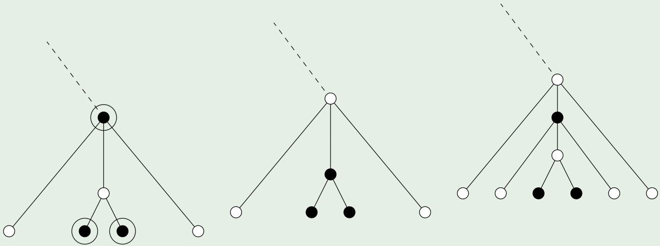
Fundamental lemma

Example ($\delta = 3$)



Fundamental lemma

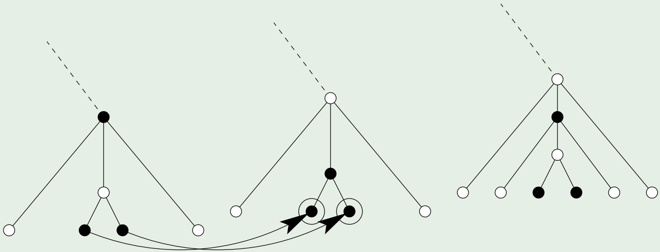
Example ($\delta = 3$)



$\text{SDEP} : \; 5$

Fundamental lemma

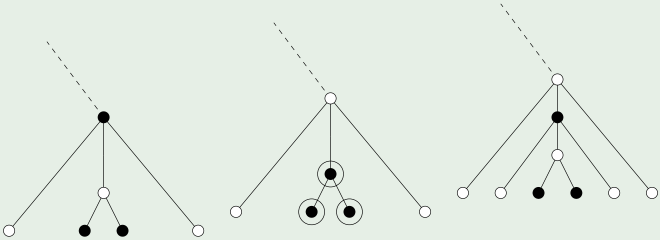
Example ($\delta = 3$)



$\textsc{SDep}: \ 5$

Fundamental lemma

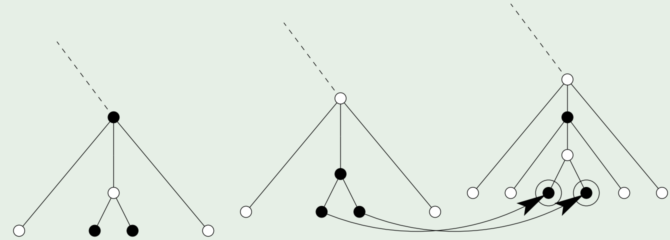
Example ($\delta = 3$)



SDEP : 5 10

Fundamental lemma

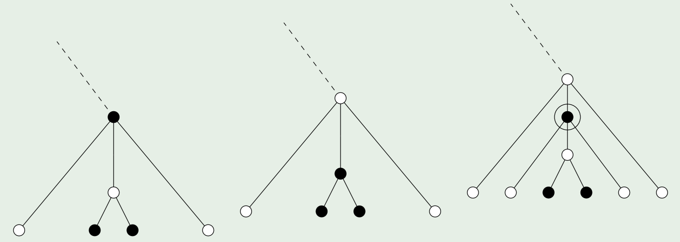
Example ($\delta = 3$)



$\text{SDEP}: \quad 5 \qquad\qquad 10$

Fundamental lemma

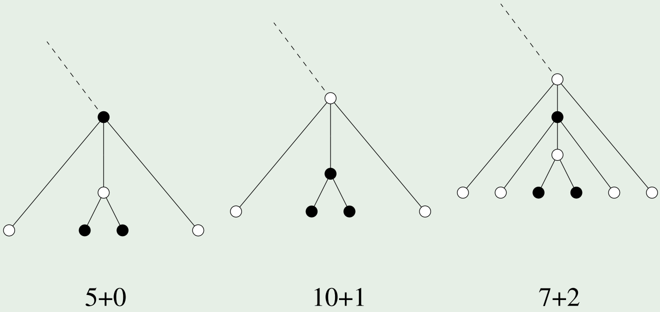
Example ($\delta = 3$)



SDEP : 5 10 7

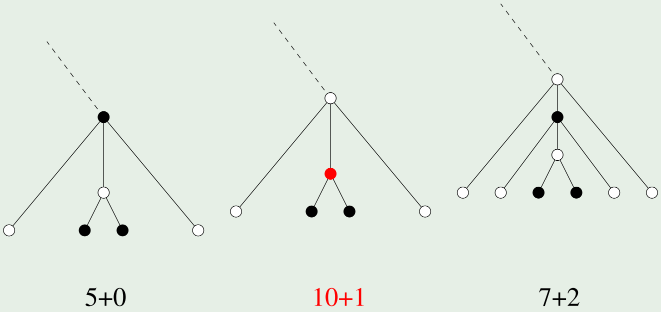
Fundamental lemma

Example ($\delta = 3$)



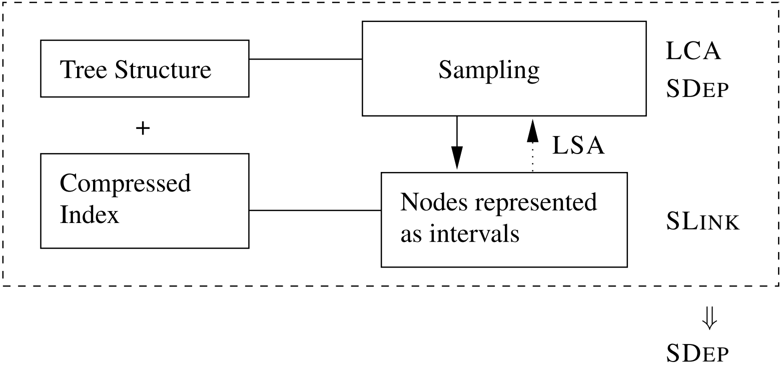
Fundamental lemma

Example ($\delta = 3$)



Entangled Operations

Why is the lemma important ?



Entangled Operations

The lemma allows us to compute other operations:

- $\textsc{SDep}(v) = \textsc{SDep}(\textsc{LCA}(v, v))$.
- $\textsc{SLink}(v) = \textsc{LCA}(\psi(v_l), \psi(v_r))$,
  $\textsc{SLink}^i(v) = \textsc{LCA}(\psi^i(v_l), \psi^i(v_r))$.
- $\textsc{LCA}(v, v') =$
  $\textsc{LF}(v[0..i-1],$
  $\textsc{LCSA}(\textsc{SLink}^i(v), \textsc{SLink}^i(v')))$,
  for the i in the lemma.

$\textsc{SLink}$ depends on LCA and LCA on $\textsc{SLink}$.

Entangled Operations

The lemma allows us to compute other operations:

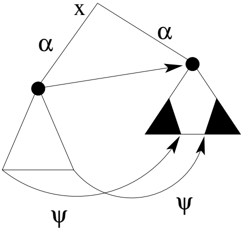
- $\textsc{SDep}(v) = \textsc{SDep}(\text{LCA}(v, v))$.
- $\textsc{SLink}(v) = \text{LCA}(\psi(v_l), \psi(v_r))$,
  $\textsc{SLink}^i(v) = \text{LCA}(\psi^i(v_l), \psi^i(v_r))$.
- $\text{LCA}(v, v') =$
  $\quad \text{LF}(v[0..i-1],$
  $\quad\quad \text{LCSA}(\textsc{SLink}^i(v), \textsc{SLink}^i(v')))$,
  for the i in the lemma.

$\textsc{SLink}$ depends on LCA and LCA on $\textsc{SLink}$.

Entangled Operations

The lemma allows us to compute other operations:

- $\mathrm{SD}{\scriptstyle\mathrm{EP}}(v) = \mathrm{SD}{\scriptstyle\mathrm{EP}}(\mathrm{LCA}(v, v))$.
- $\mathrm{SL}{\scriptstyle\mathrm{INK}}(v) = \mathrm{LCA}(\psi(v_l), \psi(v_r))$,
  $\mathrm{SL}{\scriptstyle\mathrm{INK}}^i(v) = \mathrm{LCA}(\psi^i(v_l), \psi^i(v_r))$.
- $\mathrm{LCA}(v, v') =$
  $\mathrm{LF}(v[0..i-1],$
  $\mathrm{LCSA}(\mathrm{SL}{\scriptstyle\mathrm{INK}}^i(v), \mathrm{SL}{\scriptstyle\mathrm{INK}}^i(v')))$,
  for the i in the lemma.

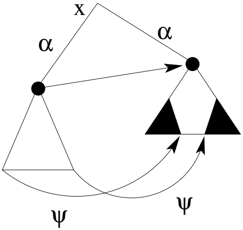


SLINK depends on LCA and LCA on SLINK.

Entangled Operations

The lemma allows us to compute other operations:

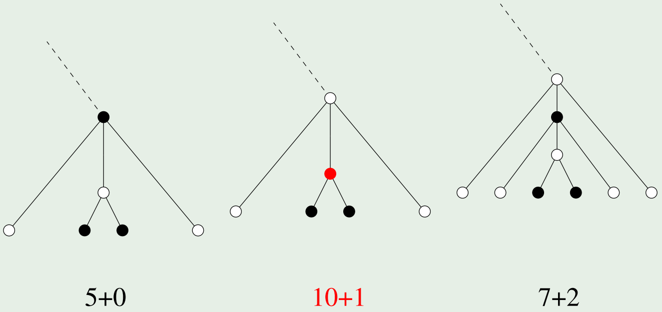
- $\text{SD}{\small\text{EP}}(v) = \text{SD}{\small\text{EP}}(\text{LCA}(v, v))$.
- $\text{SL}{\small\text{INK}}(v) = \text{LCA}(\psi(v_l), \psi(v_r))$,
  $\text{SL}{\small\text{INK}}^i(v) = \text{LCA}(\psi^i(v_l), \psi^i(v_r))$.
- $\text{LCA}(v, v') =$
  $\quad \text{LF}(v[0..i-1],$
  $\quad\quad \text{LCSA}(\text{SL}{\small\text{INK}}^i(v), \text{SL}{\small\text{INK}}^i(v')))$,
  for the i in the lemma.



SLINK depends on LCA and LCA on SLINK.

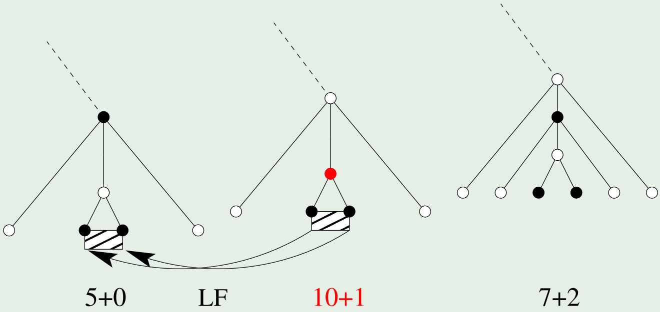
Entangled Operations

Example ($\delta = 3$)



Entangled Operations

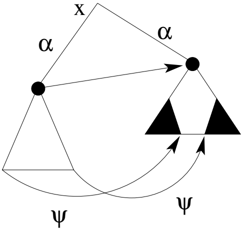
Example ($\delta = 3$)



Entangled Operations

The lemma allows us to compute other operations:

- $\text{SD}\textsc{ep}(v) = \text{SD}\textsc{ep}(\text{LCA}(v, v))$.
- $\text{SL}\textsc{ink}(v) = \text{LCA}(\psi(v_l), \psi(v_r))$,
  $\text{SL}\textsc{ink}^i(v) = \text{LCA}(\psi^i(v_l), \psi^i(v_r))$.
- $\text{LCA}(v, v') =$
  $\text{LF}(v[0..i-1],$
  $\text{LCSA}(\text{SL}\textsc{ink}^i(v), \text{SL}\textsc{ink}^i(v')))$,
  for the i in the lemma.

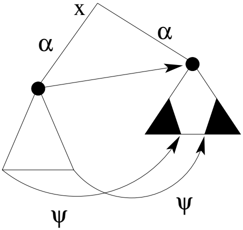


SLink depends on LCA and LCA on SLink.

Entangled Operations

The lemma allows us to compute other operations:

- $\text{SDEP}(v) = \text{SDEP}(\text{LCA}(v, v))$.
- $\text{SLINK}(v) = \text{LCA}(\psi(v_l), \psi(v_r))$,
  $\text{SLINK}^i(v) = \text{LCA}(\psi^i(v_l), \psi^i(v_r))$.
- $\text{LCA}(v, v') =$
  $\text{LF}(v[0..i-1],$
  $\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$,
  for the i in the lemma.



SLINK depends on LCA and LCA on SLINK.

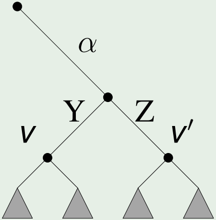
Breaking the Cycle

To avoid this circular dependency we use the next lemma.

Lemma

$$\mathrm{LCA}(v, v') = \mathrm{LCA}(\min\{v_l, v'_l\}, \max\{v_r, v'_r\})$$

Example



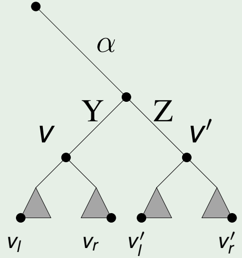
Breaking the Cycle

To avoid this circular dependency we use the next lemma.

Lemma

$$\mathrm{LCA}(v, v') \quad = \quad \mathrm{LCA}(\min\{v_l, v_l'\}, \max\{v_r, v_r'\})$$

Example



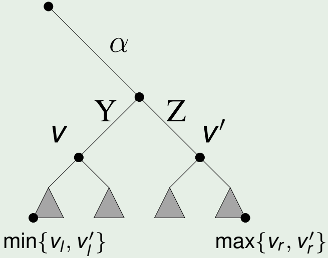
Breaking the Cycle

To avoid this circular dependency we use the next lemma.

Lemma

$$\mathrm{LCA}(v, v') = \mathrm{LCA}(\min\{v_l, v_l'\}, \max\{v_r, v_r'\})$$

Example



Breaking the Cycle

Hence we can use ψ instead of $\textsc{SLink}$.
Therefore LCA no longer depends on $\textsc{SLink}$.
The following operations simplify:

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

Breaking the Cycle

Hence we can use ψ instead of SLINK.
Therefore LCA no longer depends on SLINK.
The following operations simplify:

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

Breaking the Cycle

Hence we can use ψ instead of SLINK.
Therefore LCA no longer depends on SLINK.
The following operations simplify:

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

Further Operations

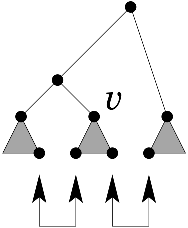
With these base operations we can also compute:

- $\textsc{Letter}(v, i) = \textsc{SLink}^i(v)[0] = \psi^i(v_l)[0]$
- $\textsc{Parent}$ is either $\mathrm{LCA}(v_l - 1, v_l)$ or $\mathrm{LCA}(v_r, v_r + 1)$, whichever is lowest.

Further Operations

With these base operations we can also compute:

- $\textsc{Letter}(v, i) = \textsc{SLink}^i(v)[0] = \psi^i(v_l)[0]$
- $\textsc{Parent}$ is either $\text{LCA}(v_l - 1, v_l)$ or $\text{LCA}(v_r, v_r + 1)$, whichever is lowest.



Further Operations

- CHILD can be computed with LETTER and binary searches.
- We can also use the fundamental lemma as $\textsc{Child}(v, X) = \mathrm{LF}(v[0..i-1], \textsc{Child}(\textsc{SLink}^i(v), X))$
- The branching is computed over child lists in the sampled tree.
- We proposed a compromise between these approaches.

Further Operations

- CHILD can be computed with LETTER and binary searches.
- We can also use the fundamental lemma as $\textsc{Child}(v, X) = \mathrm{LF}(v[0..i-1], \textsc{Child}(\textsc{SLink}^i(v), X))$
- The branching is computed over child lists in the sampled tree.
- We proposed a compromise between these approaches.

Further Operations

- $\textsc{Child}$ can be computed with $\textsc{Letter}$ and binary searches.
- We can also use the fundamental lemma as $\textsc{Child}(v, X) = \mathrm{LF}(v[0..i-1], \textsc{Child}(\textsc{SLink}^i(v), X))$
- The branching is computed over child lists in the sampled tree.
- We proposed a compromise between these approaches.

Summary

We presented a representation of suffix tree that:

- Occupies $uH_k + o(u \log \sigma)$ bits.
- Supports usual operations in a reasonable time.
- Recently the time was improved by $O(\log n)$.

Summary

Practical implementations available.

- https://github.com/simongog/sdsl-lite
- http://www.cs.helsinki.fi/group/suds/cst/
- http://pizzachili.dcc.uchile.cl/

Acknowledgments

- Thanks for listening.
- Questions ?