# Fully-Compressed Suffix Trees

Luís M. S. Russo

CITI (FCT/UNL)
**lsr@di.fct.unl.pt**

O Gosto pela Matemática - Uma Década de Talentos

# Outline

# Pattern Matching Problem 29 min

**Problem (Exact Matching)**

*Find pattern $P$ in text $T$.*

**Example**

1. $T = acctgcgctagct$, $n = 13$, $\sigma = 4$
2. $acctgcgctagct$, $P = c$, $m = 1$, $occ = 5$
3. $acctgcgctagct$, $P = cct$, $m = 3$, $occ = 1$

# Pattern Matching Problem

29 min

**Problem (Exact Matching)**

*Find pattern $P$ in text $T$.*

**Example**

1. $T = acctgcgctagct$, $n = 13$, $\sigma = 4$
2. $acctgcgctagct$, $P = c$, $m = 1$, $occ = 5$
3. $acctgcgctagct$, $P = cct$, $m = 3$, $occ = 1$

# Pattern Matching Problem

29 min

## Problem (Exact Matching)

*Find pattern $P$ in text $T$.*

## Example

1. $T = acctgcgctagct$, $n = 13$, $\sigma = 4$
2. $acctgcgctagct$, $P = c$, $m = 1$, $occ = 5$
3. $acctgcgctagct$, $P = cct$, $m = 3$, $occ = 1$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases,* $O(m + n)$.

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases,* $O(m + n)$.

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases, $O(m+n)$.*

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m+occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems 27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases,* $O(m + n)$.

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases,* $O(m + n)$.

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases,* $O(m + n)$.

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

## Problem (On-line Limitations)

*On-line solutions are not viable for large text databases,* $O(m + n)$.

## Example (On-line scan)

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems 27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases,* $O(m + n)$.

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# Text Database Problems

27 min

**Problem (On-line Limitations)**

*On-line solutions are not viable for large text databases, $O(m + n)$.*

**Example (On-line scan)**

- Scanning 3Gb DNA sequences takes too long.
- $T = acctgcgctagct$
- We need indexes that have $O(m + occ)$ time.
- $P = cct$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

**Example (Suffix Tree for *atttat*)**

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

**Example (Suffix Tree for *atttat*)**

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

**Example (Suffix Tree for *atttat*)**

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

**Example (Suffix Tree for *atttat*)**

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

**Example (Suffix Tree for *atttat*)**

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$

# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

**Example (Suffix Tree for *atttat*)**

atttat$
tttat$
ttat$
tat$
at$
t$
$

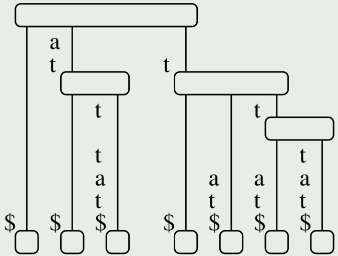
# The Suffix Tree

26 min

- Suffix trees have optimal $O(m + occ)$ performance.
- What is a suffix tree ?
- Gather the suffixes & build a tree.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$



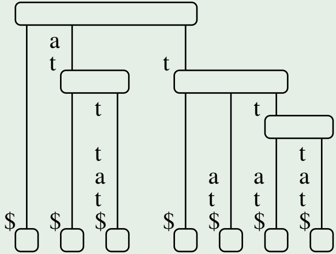
# The Suffix Tree

24 min

- Searching means descending from the ROOT & reporting leaves bellow.
- Suffix trees have problems.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$



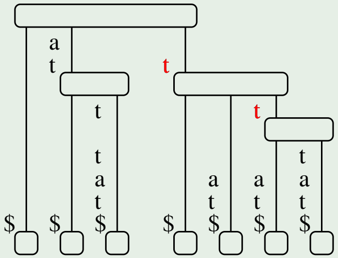
# The Suffix Tree

24 min

- Searching means descending from the ROOT & reporting leaves bellow.
- Suffix trees have problems.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$



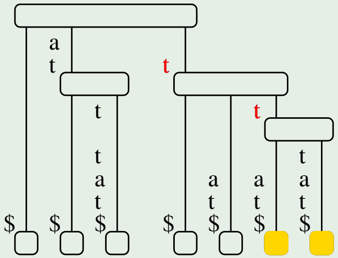
# The Suffix Tree

24 min

- Searching means descending from the ROOT & reporting leaves bellow.
- Suffix trees have problems.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$



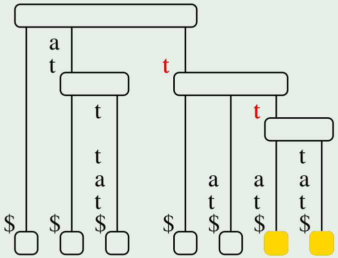
# The Suffix Tree

24 min

- Searching means descending from the ROOT & reporting leaves bellow.
- Suffix trees have problems.

## Example (Suffix Tree for *atttat*)

atttat$
tttat$
ttat$
tat$
at$
t$
$



# Suffix Tree Representation 22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

## Example (Suffix Tree for *atttat*)

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation 22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation 22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation

22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\ldots$ |
| t$ | $+2$ | $\ldots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation

22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $= 28$ | $= (n+2)(n+1)/2$ |
| | | $= O(n^2)$ |

# Suffix Tree Representation 22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\ldots$ |
| t$ | $+2$ | $\ldots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation 22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

## Example (Suffix Tree for *atttat*)

| | | |
|---|---|---|
| atttat$ | 7 | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation



- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation

22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | 7 | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\ldots$ |
| t$ | $+2$ | $\ldots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation 22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

## Example (Suffix Tree for *atttat*)

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\dots$ |
| t$ | $+2$ | $\dots$ |
| $ | $+1$ | $+1$ |
| | $=28$ | $=(n+2)(n+1)/2$ |
| | | $=O(n^2)$ |

# Suffix Tree Representation

22 min

- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

## Example (Suffix Tree for *atttat*)

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\ldots$ |
| t$ | $+2$ | $\ldots$ |
| $ | $+1$ | $+1$ |
| | $= 28$ | $= (n+2)(n+1)/2$ |
| | | $= O(n^2)$ |

# Suffix Tree Representation

22 min

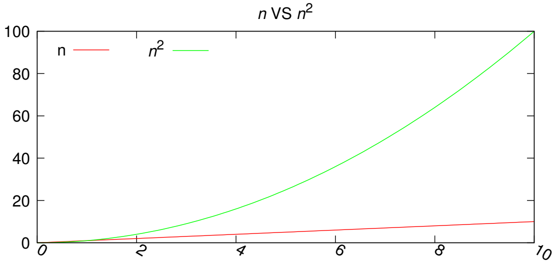
- Naive construction takes $O(n^2)$ space & time
- Efficient algorithms using pointers and amortized analysis take $O(n)$ space & time.

**Example (Suffix Tree for *atttat*)**

| | | |
|---|---|---|
| atttat$ | $7$ | $n+1$ |
| tttat$ | $+6$ | $+n$ |
| ttat$ | $+5$ | $+(n-1)$ |
| tat$ | $+4$ | $+(n-2)$ |
| at$ | $+3$ | $\ldots$ |
| t$ | $+2$ | $\ldots$ |
| $ | $+1$ | $+1$ |
| | $= 28$ | $= (n+2)(n+1)/2$ |
| | | $= O(n^2)$ |

# What is the difference ? 20 min

There is no way to index 3Gb DNA with an $O(n^2)$ algorithm

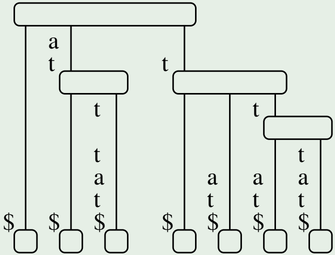


# Suffix Link

19 min

- Not going to explain Ukkonen's algorithm
- Key concept SLINK, always exists

## Example (Suffix Tree for *atttat*)

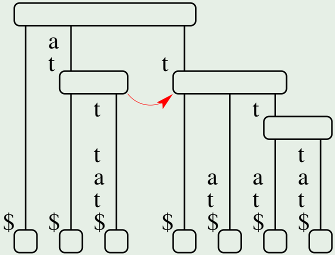


# Suffix Link

19 min

- Not going to explain Ukkonen's algorithm
- Key concept SLINK, always exists

**Example (Suffix Tree for *atttat*)**



# Representation Problems

17 min

**Problem (Suffix Trees need too much space)**

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

- This is much larger than the indexed string, $n \log \sigma$ bits.
- State of the art implementations require $[8, 10] \times 8n$ bits.
- This is 48 Gb for 3 Gb of DNA, i.e. for the human genome.
- Modern representations use data compression techniques.
- Compressing the tree requires finding regularities.

# Representation Problems

17 min

**Problem (Suffix Trees need too much space)**

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

- This is much larger than the indexed string, $n \log \sigma$ bits.
- State of the art implementations require $[8, 10] \times 8n$ bits.
- This is 48 Gb for 3 Gb of DNA, i.e. for the human genome.
- Modern representations use data compression techniques.
- Compressing the tree requires finding regularities.

# Representation Problems

17 min

**Problem (Suffix Trees need too much space)**

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

- This is much larger than the indexed string, $n \log \sigma$ bits.
- State of the art implementations require $[8, 10] \times 8n$ bits.
- This is 48 Gb for 3 Gb of DNA, i.e. for the human genome.
- Modern representations use data compression techniques.
- Compressing the tree requires finding regularities.

# Representation Problems

17 min

**Problem (Suffix Trees need too much space)**

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

- This is much larger than the indexed string, $n \log \sigma$ bits.
- State of the art implementations require $[8, 10] \times 8n$ bits.
- This is 48 Gb for 3 Gb of DNA, i.e. for the human genome.
- Modern representations use data compression techniques.
- Compressing the tree requires finding regularities.

# Representation Problems

17 min

**Problem (Suffix Trees need too much space)**

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

- This is much larger than the indexed string, $n \log \sigma$ bits.
- State of the art implementations require $[8, 10] \times 8n$ bits.
- This is 48 Gb for 3 Gb of DNA, i.e. for the human genome.
- Modern representations use data compression techniques.
- Compressing the tree requires finding regularities.

# Representation Problems

17 min

**Problem (Suffix Trees need too much space)**

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

- This is much larger than the indexed string, $n \log \sigma$ bits.
- State of the art implementations require $[8, 10] \times 8n$ bits.
- This is 48 Gb for 3 Gb of DNA, i.e. for the human genome.
- Modern representations use data compression techniques.
- Compressing the tree requires finding regularities.

# Lowest Common Ancestors 15 min

- The LCA operation.
- can be computed in $O(1)$ time, $O(n)$ space.
- No dark magic, just good algorithms ;-)

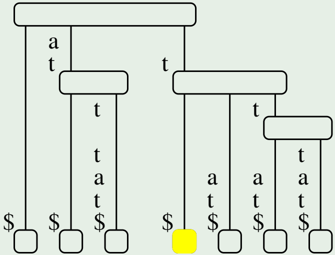
**Example (Suffix Tree for *atttat*)**

# Lowest Common Ancestors

15 min

- The LCA operation.
- can be computed in $O(1)$ time, $O(n)$ space.
- No dark magic, just good algorithms ;-)

## Example (Suffix Tree for *atttat*)

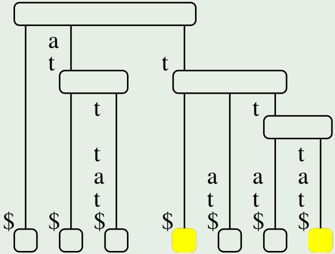


# Lowest Common Ancestors

15 min

- The LCA operation.
- can be computed in $O(1)$ time, $O(n)$ space.
- No dark magic, just good algorithms ;-)

**Example (Suffix Tree for *atttat*)**

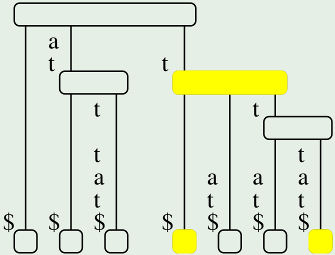


# Lowest Common Ancestors

15 min

- The LCA operation.
- can be computed in $O(1)$ time, $O(n)$ space.
- No dark magic, just good algorithms ;-)

## Example (Suffix Tree for *atttat*)

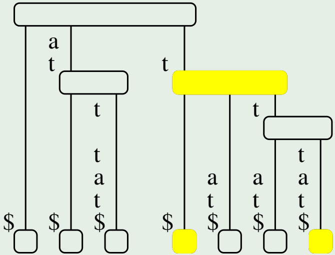


# Lowest Common Ancestors

15 min

- The LCA operation.
- can be computed in $O(1)$ time, $O(n)$ space.
- No dark magic, just good algorithms ;-)

## Example (Suffix Tree for *atttat*)

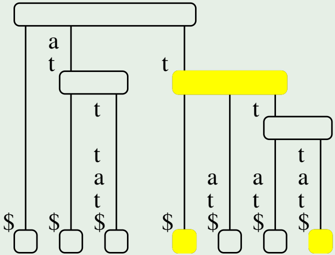


# Lowest Common Ancestors

15 min

- The LCA operation.
- can be computed in $O(1)$ time, $O(n)$ space.
- No dark magic, just good algorithms ;-)

## Example (Suffix Tree for *atttat*)



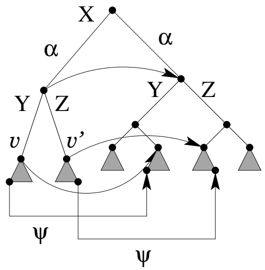
# LCA and SLINK

12 min

**Lemma**

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

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



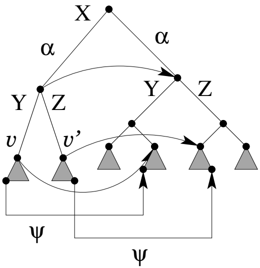

Using this lemma we do not need to store all the nodes, only some sampled ones.

# LCA and SLINK 12 min

## Lemma

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

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




Using this lemma we do not need to store all the nodes, only some sampled ones.

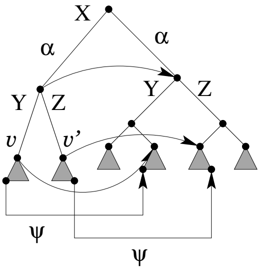
# LCA and SLINK

12 min

**Lemma**

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

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




Using this lemma we do not need to store all the nodes, only some sampled ones.

# Fundamental lemma

10 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.**

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

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

# Fundamental lemma

10 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.**

$\text{SDEP}(\text{LCA}(v, v'))$
$$= i + \text{SDEP}(\text{SLINK}^i(\text{LCA}(v, v')))$$
$$= i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$$
$$\ge i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$$
The last inequality is an equality for some $i \le d$. □

# Fundamental lemma

10 min

## Lemma

*If* $\text{SL}\textsc{ink}^r(\text{LCA}(v, v')) = \textsc{Root}$*, 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 $i \leq d$. $\square$

# Fundamental lemma

10 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.**

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

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

# Fundamental lemma

10 min

## Lemma

*If* $\text{SLINK}^r(\text{LCA}(v, v')) = \text{ROOT}$*, and let* $d = \min(\delta, r+1)$*.*
*Then* $\text{SDEP}(\text{LCA}(v, v')) \geq$
$$\max_{0 \leq i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\}$$

## Proof.

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

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

# Fundamental lemma

10 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.**

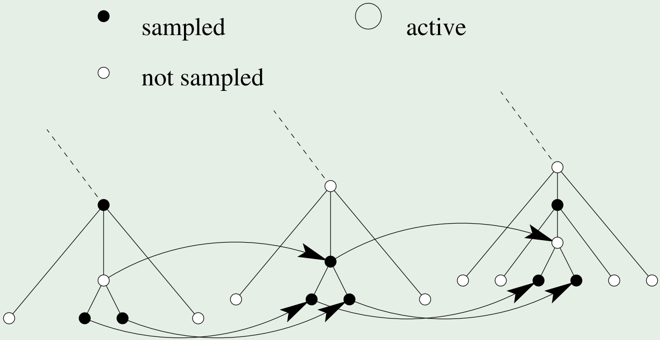
$$\begin{aligned}
&\text{SDEP}(\text{LCA}(v, v')) \\
&\quad = i + \text{SDEP}(\text{SLINK}^i(\text{LCA}(v, v'))) \\
&\quad = i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
&\quad \ge i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))
\end{aligned}$$

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

# Fundamental lemma

7 min

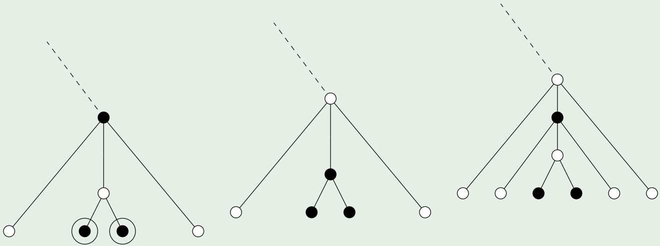
## Example ($\delta = 3$)



# Fundamental lemma

7 min

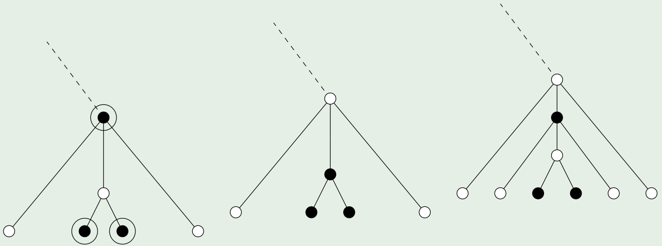
## Example ($\delta = 3$)



# Fundamental lemma

7 min

## Example ($\delta = 3$)

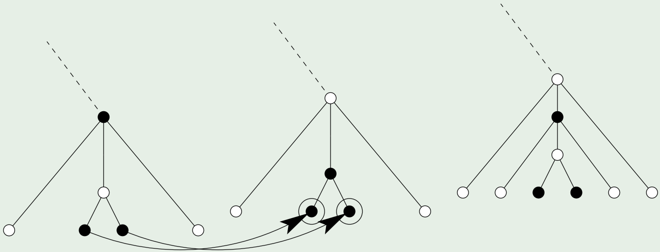


SDEP : 5

# Fundamental lemma

7 min

## Example ($\delta = 3$)

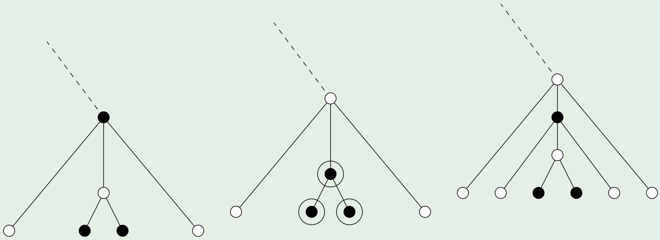


$\text{SDEP}: \;\; 5$

# Fundamental lemma

7 min

## Example ($\delta = 3$)

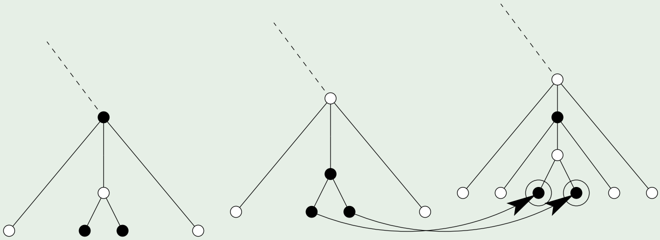


SDEP : 5 10

# Fundamental lemma

7 min

## Example ($\delta = 3$)

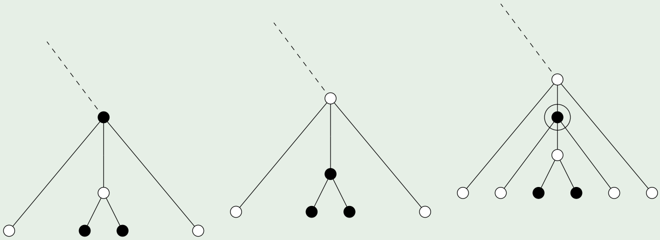


SDEP : 5 10

# Fundamental lemma

7 min

## Example ($\delta = 3$)

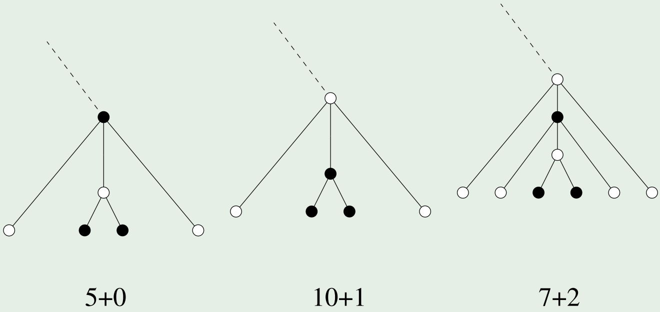


SDEP : 5 10 7

# Fundamental lemma

7 min

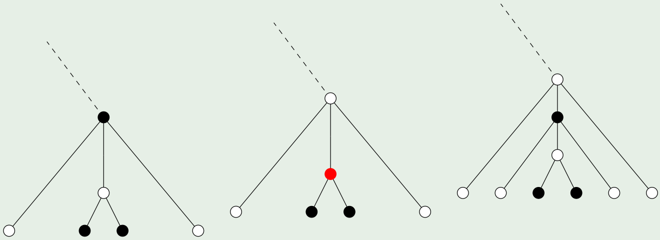
## Example ($\delta = 3$)



# Fundamental lemma

7 min

## Example ($\delta = 3$)



5+0

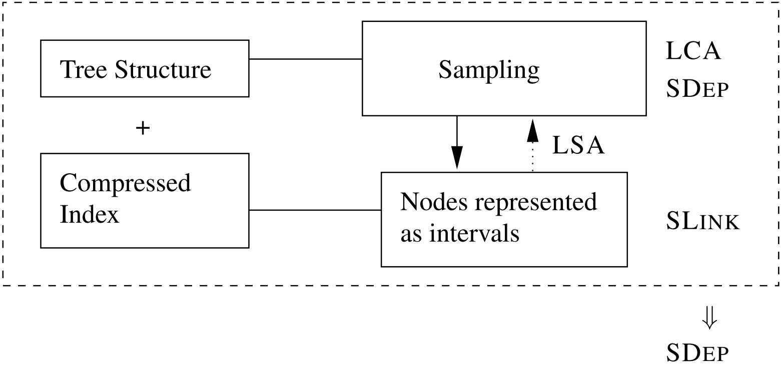
10+1

7+2

# Entangled Operations

6 min

Why is the lemma important ?



# Experimental Results

3 min

| Space in MBs | File | FCST | FFCST | LSA | LSAF | CST |
|---|---|---|---|---|---|---|
| Pitches | 53 | 44 | 134 | 43 | 50 | 214 |
| Proteins | 63 | 50 | 121 | 48 | 56 | 204 |
| DNA | 100 | 57 | 142 | 54 | 69 | 287 |
| XML | 100 | 55 | 198 | 52 | 67 | 316 |

| CHILD Operation | Pitches | Proteins | DNA | XML |
|---|---|---|---|---|
| FCST | 1.3E-2 | 3.4E-3 | 1.6E-3 | 8.7E-3 |
| FFCST | 9.4E-3 | 5.7E-3 | 7.2E-1 | 1.9E-2 |
| LSA | 7.6E-3 | 2.7E-3 | 2.E-3 | 9.7E-3 |
| LSAF | 1.3E-2 | 3.5E-3 | 1.6E-3 | 8.9E-3 |
| CST | 5.4E-4 | 4.2E-4 | 1.2E-4 | 6.4E-4 |

# Experimental Results

3 min

| Space in MBs | File | FCST | FFCST | LSA | LSAF | CST |
|---|---|---|---|---|---|---|
| Pitches | 53 | 44 | 134 | 43 | 50 | 214 |
| Proteins | 63 | 50 | 121 | 48 | 56 | 204 |
| DNA | 100 | 57 | 142 | 54 | 69 | 287 |
| XML | 100 | 55 | 198 | 52 | 67 | 316 |

| CHILD Operation | Pitches | Proteins | DNA | XML |
|---|---|---|---|---|
| FCST | 1.3E-2 | 3.4E-3 | 1.6E-3 | 8.7E-3 |
| FFCST | 9.4E-3 | 5.7E-3 | 7.2E-1 | 1.9E-2 |
| LSA | 7.6E-3 | 2.7E-3 | 2.E-3 | 9.7E-3 |
| LSAF | 1.3E-2 | 3.5E-3 | 1.6E-3 | 8.9E-3 |
| CST | 5.4E-4 | 4.2E-4 | 1.2E-4 | 6.4E-4 |

# Summary

2 min

We presented a representation of suffix trees that:

- occupies $n \log \sigma + o(u \log \sigma)$ bits.
- in fact it is even better $nH_k + o(u \log \sigma)$ bits.
- supports usual operations in a reasonable time.
- current prototypes show that this performance holds in practice.

# Summary 2 min

We presented a representation of suffix trees that:

- occupies $n \log \sigma + o(u \log \sigma)$ bits.
- in fact it is even better $nH_k + o(u \log \sigma)$ bits.
- supports usual operations in a reasonable time.
- current prototypes show that this performance holds in practice.

# Summary

2 min

We presented a representation of suffix trees that:

- occupies $n \log \sigma + o(u \log \sigma)$ bits.
- in fact it is even better $nH_k + o(u \log \sigma)$ bits.
- supports usual operations in a reasonable time.
- current prototypes show that this performance holds in practice.

# Summary

2 min

We presented a representation of suffix trees that:

- occupies $n \log \sigma + o(u \log \sigma)$ bits.
- in fact it is even better $nH_k + o(u \log \sigma)$ bits.
- supports usual operations in a reasonable time.
- current prototypes show that this performance holds in practice.