Computing k-th Lyndon Word and Decoding Lexicographically Minimal de Bruijn Sequence

Tomasz Kociumaka,
Jakub Radoszewski, Wojciech Rytter

University of Warsaw

CPM 2014
Moscow, June 18, 2014

Lyndon words

Definition (Lyndon, 1954)

A *Lyndon word* is a word that is strictly smaller than all its nontrivial cyclic rotations.

Lyndon words

Definition (Lyndon, 1954)

A *Lyndon word* is a word that is strictly smaller than all its nontrivial cyclic rotations.

Examples:

- 0010011,
- 000001,
- 1.

Lyndon words

Definition (Lyndon, 1954)

A *Lyndon word* is a word that is strictly smaller than all its nontrivial cyclic rotations.

Examples:

- 0010011,
- 000001,
- 1.

Non-examples:

- 010011 ($010011 > 001101$)
- 001001 ($001001 = (001)^2$)

Enumerating Lyndon words

Notation:

Σ a fixed finite totally-ordered alphabet,

$\mathcal{L}$ all Lyndon words in Σ^+,

$\mathcal{L}_n$ all Lyndon words in Σ^n.

Enumerating Lyndon words

Notation:

Σ a fixed finite totally-ordered alphabet,

$\mathcal{L}$ all Lyndon words in Σ^+,

$\mathcal{L}_n$ all Lyndon words in Σ^n.

We define

$$Lynd(w) \; = \; \{x \in \mathcal{L}_{|w|} \; : \; x \leq w\}.$$

Enumerating Lyndon words

Notation:

Σ a fixed finite totally-ordered alphabet,

$\mathcal{L}$ all Lyndon words in Σ^+,

$\mathcal{L}_n$ all Lyndon words in Σ^n.

We define

$$Lynd(w) \;=\; \{x \in \mathcal{L}_{|w|} \;:\; x \le w\}.$$

Example:

$$\Sigma = \{0, 1\},$$
$$\mathcal{L}_6 = \{000001, 000011, 000101, 000111, 001011$$
$$001101, 001111, 010111, 011111\},$$

Enumerating Lyndon words

Notation:

Σ a fixed finite totally-ordered alphabet,

$\mathcal{L}$ all Lyndon words in Σ^+,

$\mathcal{L}_n$ all Lyndon words in Σ^n.

We define

$$Lynd(w) \;=\; \{x \in \mathcal{L}_{|w|} \;:\; x \leq w\}.$$

Example:

$$\begin{aligned}
\Sigma &= \{0,1\},\\
\mathcal{L}_6 &= \{000001, 000011, 000101, 000111, 001011\\
&\quad 001101, 001111, 010111, 011111\},\\
|Lynd(001110)| &= 6,
\end{aligned}$$

Enumerating Lyndon words

Notation:

Σ a fixed finite totally-ordered alphabet,

$\mathcal{L}$ all Lyndon words in Σ^+,

$\mathcal{L}_n$ all Lyndon words in Σ^n.

We define

$$Lynd(w) \;=\; \{x \in \mathcal{L}_{|w|} \;:\; x \le w\}.$$

Example:

$$\Sigma = \{0, 1\},$$
$$\mathcal{L}_6 = \{000001, 000011, 000101, 000111, 001011$$
$$001101, 001111, 010111, 011111\},$$
$$|Lynd(001110)| = 6,$$
$$|Lynd(011010)| = 8.$$

Our results (Lyndon words)

Theorem

Given a word w of length n the value $|Lynd(w)|$ can be computed in $\mathcal{O}(n^3)$ time.

Our results (Lyndon words)

Theorem

Given a word w of length n the value $|Lynd(w)|$ can be computed in $\mathcal{O}(n^3)$ time.

Technical assumptions:

- word-RAM model,
- $\Sigma = \{0, 1, \dots, \sigma - 1\}$, σ fits in a machine word.

Our results (Lyndon words)

Theorem

Given a word w of length n the value $|Lynd(w)|$ can be computed in $\mathcal{O}(n^3)$ time.

Technical assumptions:

- word-RAM model,
- $\Sigma = \{0, 1, \ldots, \sigma - 1\}$, σ fits in a machine word.

Corollary

The k-th lexicographically smallest Lyndon word in $\mathcal{L}_n$ can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Proof.

Binary search over Σ^n. $\square$

Motivation

- Lyndon words have numerous applications:

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,
  - text algorithms,

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,
  - text algorithms,
  - algebra.

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,
  - text algorithms,
  - algebra.
- Generating k-th smallest element is a natural question for any combinatorial object;

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,
  - text algorithms,
  - algebra.
- Generating k-th smallest element is a natural question for any combinatorial object;
  - no polynomial-time algorithm known previously for $\mathcal{L}_n$.

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,
  - text algorithms,
  - algebra.
- Generating k-th smallest element is a natural question for any combinatorial object;
  - no polynomial-time algorithm known previously for $\mathcal{L}_n$.
- Generalization of the known formula:

$$|\mathcal{L}_n| = \frac{1}{n}\sum_{d|n}\mu\left(\frac{n}{d}\right)\sigma^d.$$

Motivation

- Lyndon words have numerous applications:
  - combinatorics of words,
  - text algorithms,
  - algebra.
- Generating k-th smallest element is a natural question for any combinatorial object;
  - no polynomial-time algorithm known previously for $\mathcal{L}_n$.
- Generalization of the known formula:

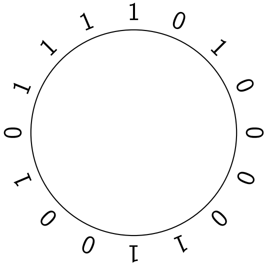
$$|\mathcal{L}_n| = \frac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d}) \sigma^d.$$

- Applications to decoding the minimal de Bruijn sequence.

De Bruijn sequences

Definition

A *de Bruijn sequence of rank n* is a cyclic sequence in which every word from Σ^n appears as a subword exactly once.

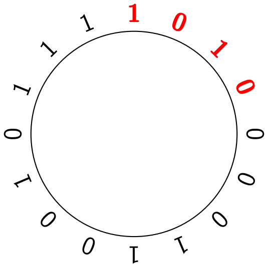


$$n = 4, \ \Sigma = \{0, 1\}$$

De Bruijn sequences

Definition

A *de Bruijn sequence of rank n* is a cyclic sequence in which every word from Σ^n appears as a subword exactly once.

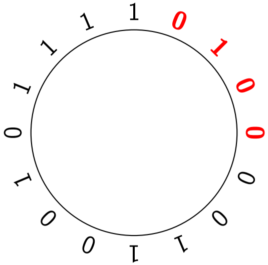


$$n = 4,\ \ \Sigma = \{0, 1\}$$

De Bruijn sequences

Definition

A *de Bruijn sequence of rank n* is a cyclic sequence in which every word from Σ^n appears as a subword exactly once.

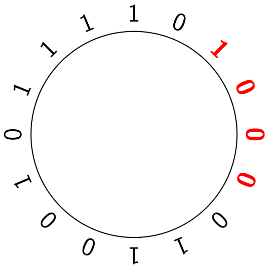


$$n = 4,\ \Sigma = \{0, 1\}$$

De Bruijn sequences

Definition

A *de Bruijn sequence of rank n* is a cyclic sequence in which every word from Σ^n appears as a subword exactly once.

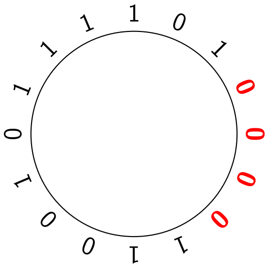


$$n = 4,\ \ \Sigma = \{0, 1\}$$

De Bruijn sequences

Definition

A *de Bruijn sequence of rank n* is a cyclic sequence in which every word from Σ^n appears as a subword exactly once.



$n = 4, \ \Sigma = \{0, 1\}$

Lexicographically minimal de Bruijn sequence

Notation:

Σ fixed alphabet,

dB_n the lexicographically minimal de Bruijn sequence over Σ of rank n.

Lexicographically minimal de Bruijn sequence

Notation:

Σ fixed alphabet,

dB_n the lexicographically minimal de Bruijn sequence over Σ of rank n.

For $\Sigma = \{0, 1\}$ and $n = 6$ the sequence dB_6 is:

0000001000011000101000111001001011
001101001111010101110110111111

Lexicographically minimal de Bruijn sequence

Notation:

Σ fixed alphabet,

dB_n the lexicographically minimal de Bruijn sequence over Σ of rank n.

For $\Sigma = \{0, 1\}$ and $n = 6$ the sequence dB_6 is:

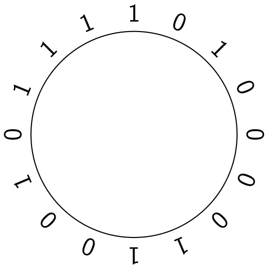
0000001000011000101000111001001011
001101001111010101110110111111

Theorem (Fredricksen, Maiorana, 1978)

The sequence dB_n is the concatenation, in lexicographic order, of all Lyndon words over Σ whose length divides n.

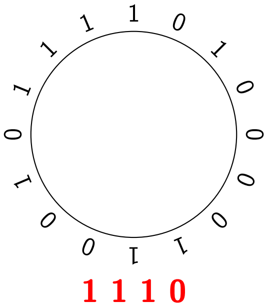
Decoding de Bruijn sequences

A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.

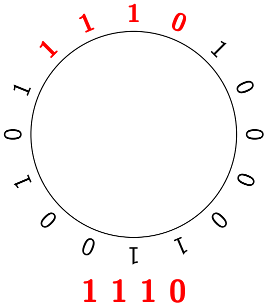


Decoding de Bruijn sequences

A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.

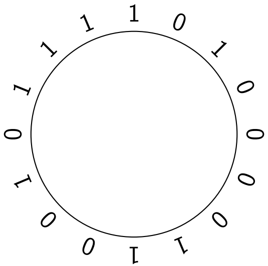


A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



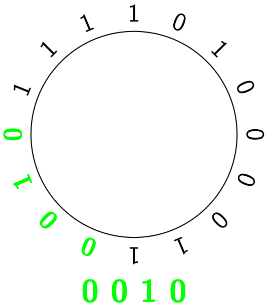
Decoding de Bruijn sequences

A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



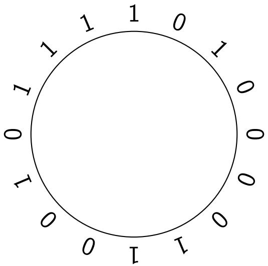
0 0 1 0

A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



Decoding de Bruijn sequences

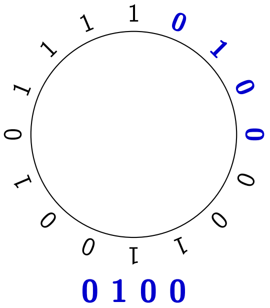
A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



0 1 0 0

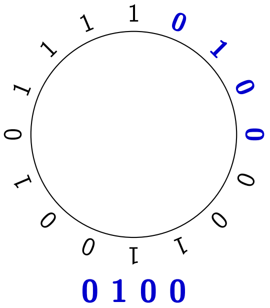
Decoding de Bruijn sequences

A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



Decoding de Bruijn sequences

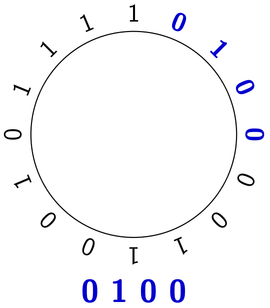
A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



Applications:

- position sensing schemes.

A *decoding algorithm* finds the position of an arbitrary word from Σ^n in a given de Bruijn sequence in polynomial time.



Applications:

- position sensing schemes.

Previous work:

- polynomial-time decoding schemes for several types of de Bruijn sequences:
  - Paterson & Robshaw, 1995
  - Mitchell, Etzion and Paterson, 1996
  - Tuliani, 2001

Our results (minimal de Bruijn sequences)

Using the correspondence between Lyndon words and lex. min. de Bruijn sequences we show the following:

Theorem

There exists an $\mathcal{O}(n^3)$-time decoding algorithm for dB_n.

Theorem

For any n the k-th symbol of dB_n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Main algorithm

We sketch the proof of the following result:

Theorem

For any word w of length n, the value $|Lynd(w)|$ can be computed in $poly(n)$ time.

Main algorithm: auxiliary reduction

We sketch the proof of the following result:

Theorem

For any word w of length n, the value $|Lynd(w)|$ can be computed in $poly(n)$ time.

For a word $w \in \Sigma^+$, denote the lexicographically minimal cyclic rotation of w by $\langle w \rangle$.

Lemma

For any word $w \in \Sigma^n$ the lexicographically maximal $w' \in \Sigma^n$ such that $\langle w' \rangle = w' \leq w$ can be found in $\mathcal{O}(n^2)$ time.

Main algorithm: auxiliary reduction

We sketch the proof of the following result:

Theorem

For any word w of length n, the value $|Lynd(w)|$ can be computed in $poly(n)$ time.

For a word $w \in \Sigma^+$, denote the lexicographically minimal cyclic rotation of w by $\langle w \rangle$.

Lemma

For any word $w \in \Sigma^n$ the lexicographically maximal $w' \in \Sigma^n$ such that $\langle w' \rangle = w' \leq w$ can be found in $\mathcal{O}(n^2)$ time.

$Lynd(w') = Lynd(w)$, so we can assume that $\langle w \rangle = w$.

Formula for $|Lynd(w)|$

Define

$$CS(w) = \{x \in \Sigma^{|w|} : \langle x \rangle \leq w\}.$$

Lemma

If $\langle w \rangle = w$ *then*

$$|Lynd(w)| = \tfrac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d})\, |CS(w[1..d])|$$

Formula for $|Lynd(w)|$

Define

$$CS(w) = \{x \in \Sigma^{|w|} : \langle x \rangle \leq w\}.$$

Lemma

If $\langle w \rangle = w$ *then*

$$|Lynd(w)| = \frac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d})\,|CS(w[1..d])|$$

Example. Let $w = 010111$.

$$CS(w[1..1]) = \{0\}, \quad CS(w[1..2]) = \{00, 01, 10\},$$

$$CS(w[1..3]) = \{000, 001, 010, 100\}, \quad |CS(w)| = 54$$

$$|Lynd(w)| = \tfrac{1}{6} \cdot \big(\mu(1)\,|CS(w)| + \mu(2)\,|CS(w[1..3])| + \mu(3)\,|CS(w[1..2])| + \mu(6)\,|CS(w[1..1])|\big) = \tfrac{1}{6} \cdot (54 - 4 - 3 + 1) = 8.$$

$CS(w)$ as a language

Define a language $L(w)$ as follows:
$x \in L(w)$ if there exists a subword z of x such that $z \leq w$ but z is not a proper prefix of w.

$CS(w)$ as a language

Define a language $L(w)$ as follows:
$x \in L(w)$ if there exists a subword z of x such that $z \leq w$ but z is not a proper prefix of w.

Fact

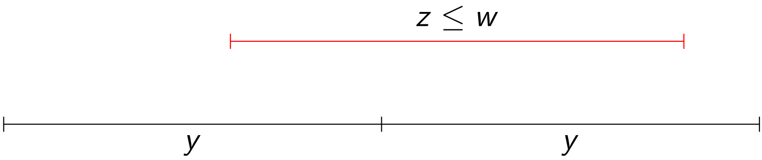
If $\langle w \rangle = w$ then $CS(w) = \sqrt{L(w)} \cap \Sigma^n$, where $\sqrt{L} = \{y : y^2 \in L\}$.

$CS(w)$ as a language

Define a language $L(w)$ as follows:
$x \in L(w)$ if there exists a subword z of x such that $z \leq w$ but z is not a proper prefix of w.

Fact

If $\langle w \rangle = w$ then $CS(w) = \sqrt{L(w)} \cap \Sigma^n$, where $\sqrt{L} = \{y : y^2 \in L\}$.



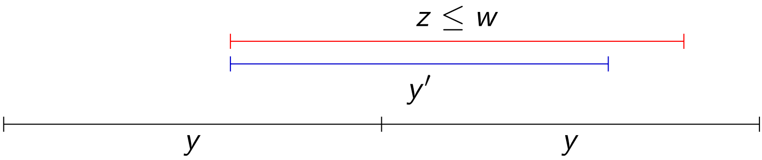
$y^2 \in L(w)$, $|y| = n$

$CS(w)$ as a language

Define a language $L(w)$ as follows:
$x \in L(w)$ if there exists a subword z of x such that $z \leq w$ but z is not a proper prefix of w.

Fact

If $\langle w \rangle = w$ then $CS(w) = \sqrt{L(w)} \cap \Sigma^n$, where $\sqrt{L} = \{y : y^2 \in L\}$.

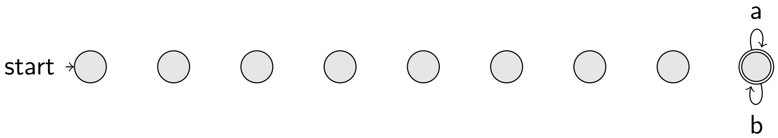


$y^2 \in L(w)$, $|y| = n$ $\qquad$ $y' \leq w$, hence $y \in CS(w)$

Deterministic automaton recognizing $L(w)$

A has $n+1$ states: one for each proper prefix of w, and an auxiliary accepting state AC. The transitions are defined as follows: $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

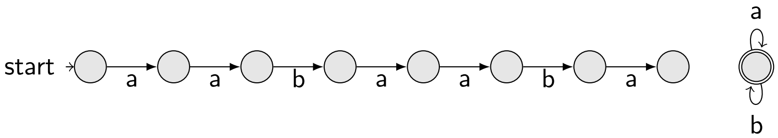
$$\delta(w_{(i)}, c) = \left\{ \right.$$



Deterministic automaton recognizing $L(w)$

A has $n+1$ states: one for each proper prefix of w, and an auxiliary accepting state AC. The transitions are defined as follows: $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

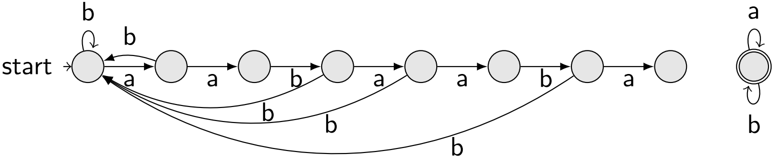
$$\delta(w_{(i)}, c) = \begin{cases} w_{(i+1)} & \text{if } c = w[i+1] \text{ and } i \neq n-1, \\ & \\ & \end{cases}$$



Deterministic automaton recognizing $L(w)$

A has $n+1$ states: one for each proper prefix of w, and an auxiliary accepting state AC. The transitions are defined as follows: $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

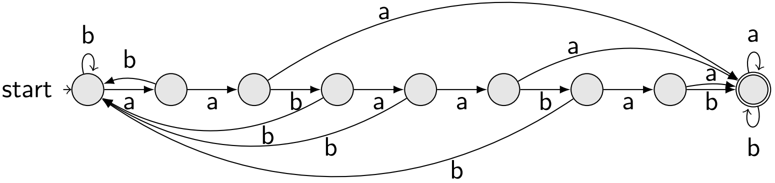
$$\delta(w_{(i)}, c) = \begin{cases} w_{(i+1)} & \text{if } c = w[i+1] \text{ and } i \neq n-1, \\ w_{(0)} & \text{if } c > w[i+1], \\ & \end{cases}$$



Deterministic automaton recognizing $L(w)$

A has $n+1$ states: one for each proper prefix of w, and an auxiliary accepting state AC. The transitions are defined as follows: $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

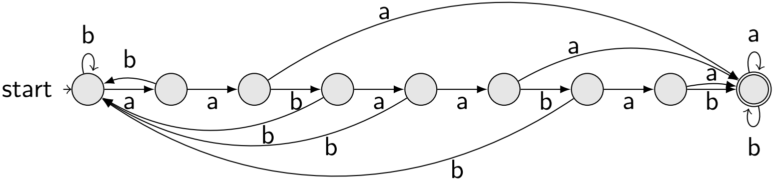
$$\delta(w_{(i)}, c) = \begin{cases} w_{(i+1)} & \text{if } c = w[i+1] \text{ and } i \neq n-1, \\ w_{(0)} & \text{if } c > w[i+1], \\ AC & \text{otherwise.} \end{cases}$$



Deterministic automaton recognizing $L(w)$

A has $n+1$ states: one for each proper prefix of w, and an auxiliary accepting state AC. The transitions are defined as follows: $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

$$\delta(w_{(i)}, c) = \begin{cases} w_{(i+1)} & \text{if } c = w[i+1] \text{ and } i \neq n-1, \\ w_{(0)} & \text{if } c > w[i+1], \\ AC & \text{otherwise.} \end{cases}$$



Fact

Let a word w be its own minimal rotation, i.e. $\langle w \rangle = w$. If $w[1..j]$ is a border of $w[1..i]$, then $w[j+1] \leq w[i+1]$.

Dynamic programming

For $A = (Q, q_0, F, \delta)$ and $q, q' \in Q$ define

$$L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}.$$

Dynamic programming

For $A = (Q, q_0, F, \delta)$ and $q, q' \in Q$ define

$$L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}.$$

Fact

For any automaton $A = (Q, q_0, F, \delta)$ we have

$$|\sqrt{L(A)} \cap \Sigma^n| = \sum_{q \in Q, q' \in F} |L_A(q_0, q) \cap L_A(q, q') \cap \Sigma^n|.$$

Dynamic programming

For $A = (Q, q_0, F, \delta)$ and $q, q' \in Q$ define

$$L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}.$$

Fact

For any automaton $A = (Q, q_0, F, \delta)$ we have

$$|\sqrt{L(A)} \cap \Sigma^n| = \sum_{q \in Q, q' \in F} |L_A(q_0, q) \cap L_A(q, q') \cap \Sigma^n|.$$

One can compute $|L_A(q_0, q) \cap L_A(q', q'') \cap \Sigma^m|$ for all $q, q', q'' \in Q$ and $m \leq n$ in $O(poly(n))$ time.

Dynamic programming

For $A = (Q, q_0, F, \delta)$ and $q, q' \in Q$ define

$$L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}.$$

Fact

For any automaton $A = (Q, q_0, F, \delta)$ we have

$$|\sqrt{L(A)} \cap \Sigma^n| = \sum_{q \in Q, q' \in F} |L_A(q_0, q) \cap L_A(q, q') \cap \Sigma^n|.$$

One can compute $|L_A(q_0, q) \cap L_A(q', q'') \cap \Sigma^m|$ for all $q, q', q'' \in Q$ and $m \leq n$ in $O(poly(n))$ time.

To obtain $\mathcal{O}(n^3)$ time complexity, we use an alternative method that exploits the structure of A.

Decoding de Bruijn Sequence

Let $dB_n = \lambda_1 \dots \lambda_p$, where $\lambda_i \in \mathcal{L}$ and $|\lambda_i| \mid n$.

The proof of theorem of Fredricksen and Maiorana provides, for each $w \in \Sigma^n$, λ_k such that w is a subword of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

Decoding de Bruijn Sequence

Let $dB_n = \lambda_1 \dots \lambda_p$, where $\lambda_i \in \mathcal{L}$ and $|\lambda_i| \mid n$.

The proof of theorem of Fredricksen and Maiorana provides, for each $w \in \Sigma^n$, λ_k such that w is a subword of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

1. Find λ_{k-1} and λ_{k+1} (using the FKM algorithm).

Decoding de Bruijn Sequence

Let $dB_n = \lambda_1 \dots \lambda_p$, where $\lambda_i \in \mathcal{L}$ and $|\lambda_i| \mid n$.

The proof of theorem of Fredricksen and Maiorana provides, for each $w \in \Sigma^n$, λ_k such that w is a subword of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

1. Find λ_{k-1} and λ_{k+1} (using the FKM algorithm).
2. Perform pattern matching for w in $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

Decoding de Bruijn Sequence

Let $dB_n = \lambda_1 \dots \lambda_p$, where $\lambda_i \in \mathcal{L}$ and $|\lambda_i| \mid n$.

The proof of theorem of Fredricksen and Maiorana provides, for each $w \in \Sigma^n$, λ_k such that w is a subword of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

1. Find λ_{k-1} and λ_{k+1} (using the FKM algorithm).
2. Perform pattern matching for w in $\lambda_{k-1}\lambda_k\lambda_{k+1}$.
3. The occurrence of λ_k in dB_n *ends* at position $|CS(\lambda_k^{n/|\lambda_k|})|$.

Decoding de Bruijn Sequence

Let $dB_n = \lambda_1 \dots \lambda_p$, where $\lambda_i \in \mathcal{L}$ and $|\lambda_i| \mid n$.

The proof of theorem of Fredricksen and Maiorana provides, for each $w \in \Sigma^n$, λ_k such that w is a subword of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

1. Find λ_{k-1} and λ_{k+1} (using the FKM algorithm).
2. Perform pattern matching for w in $\lambda_{k-1}\lambda_k\lambda_{k+1}$.
3. The occurrence of λ_k in dB_n *ends* at position $|CS(\lambda_k^{n/|\lambda_k|})|$.

Decoding de Bruijn Sequence

Let $dB_n = \lambda_1 \dots \lambda_p$, where $\lambda_i \in \mathcal{L}$ and $|\lambda_i| \mid n$.

The proof of theorem of Fredricksen and Maiorana provides, for each $w \in \Sigma^n$, λ_k such that w is a subword of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

1. Find λ_{k-1} and λ_{k+1} (using the FKM algorithm).
2. Perform pattern matching for w in $\lambda_{k-1}\lambda_k\lambda_{k+1}$.
3. The occurrence of λ_k in dB_n *ends* at position $|CS(\lambda_k^{n/|\lambda_k|})|$.

$\left|CS\left(\lambda_k^{n/|\lambda_k|}\right)\right|$ can be computed in $\mathcal{O}(n^3)$ time which yields the $\mathcal{O}(n^3)$-time decoding algorithm.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.
- The k-th lexicographically smallest Lyndon word of length n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.
- The k-th lexicographically smallest Lyndon word of length n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.
- An $\mathcal{O}(n^3)$-time decoding algorithm for dB_n.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.
- The k-th lexicographically smallest Lyndon word of length n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.
- An $\mathcal{O}(n^3)$-time decoding algorithm for dB_n.
- The k-th symbol of dB_n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.
- The k-th lexicographically smallest Lyndon word of length n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.
- An $\mathcal{O}(n^3)$-time decoding algorithm for dB_n.
- The k-th symbol of dB_n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.
- The k-th lexicographically smallest Lyndon word of length n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.
- An $\mathcal{O}(n^3)$-time decoding algorithm for dB_n.
- The k-th symbol of dB_n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Further work:

- Improve the running time of the algorithms.

Summary

Our results:

- The value $|Lynd(w)|$ can be computed in $\mathcal{O}(|w|^3)$ time.
- The k-th lexicographically smallest Lyndon word of length n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.
- An $\mathcal{O}(n^3)$-time decoding algorithm for dB_n.
- The k-th symbol of dB_n can be computed in $\mathcal{O}(n^4 \log \sigma)$ time.

Further work:

- Improve the running time of the algorithms.
    - replace dynamic programming over A
      by Fast Fourier Transform in the computation of $CS(w)$.

Thank you for your attention!