Efficient Construction of Long Orientable Sequences

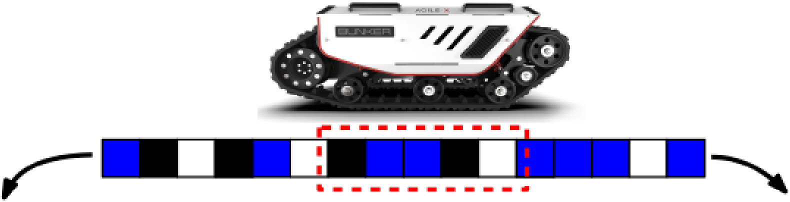
Daniel Gabrić and Joe Sawada

University of Guelph

June 27, 2024

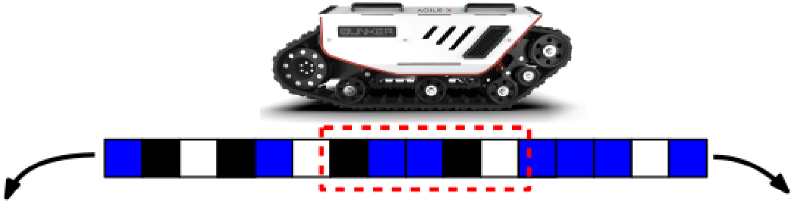
A scenario

- Consider an autonomous robot with limited sensors on a cyclic track.



A scenario

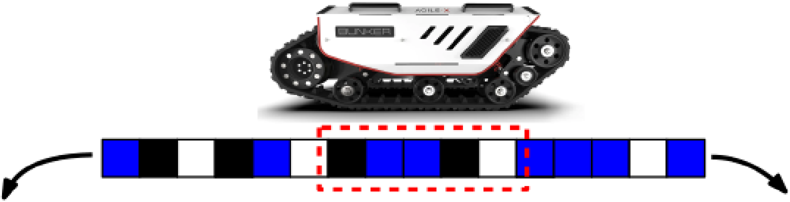
- Consider an autonomous robot with limited sensors on a cyclic track.



- To determine its location on the track, labelled with a finite set of colored squares, the robot scans a window of n squares directly beneath it.

A scenario

- Consider an autonomous robot with limited sensors on a cyclic track.



- To determine its location on the track, labelled with a finite set of colored squares, the robot scans a window of *n* squares directly beneath it.
- Suppose the track is labelled with only white and black squares.
- What constraints on the sequence of white and black squares allow the robot to uniquely determine its position *and* orientation?

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.
- To uniquely determine position and orientation, it is sufficient for w to contain every length-n binary string at most once *in either direction*.

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.
- To uniquely determine position and orientation, it is sufficient for w to contain every length-n binary string at most once *in either direction*.
- Let w^R denote the reversal of w.

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.
- To uniquely determine position and orientation, it is sufficient for w to contain every length-n binary string at most once *in either direction*.
- Let w^R denote the reversal of w.

Definition 1

We say that a binary circular string w is an *orientable sequence of order n* (or an $\mathcal{OS}(n)$) if every length-n string occurs at most once in w, and if a length-n string u occurs in w, then u^R is not a substring of w.

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.
- To uniquely determine position and orientation, it is sufficient for w to contain every length-n binary string at most once *in either direction*.
- Let w^R denote the reversal of w.

Definition 1

We say that a binary circular string w is an *orientable sequence of order n* (or an $\mathcal{OS}(n)$) if every length-n string occurs at most once in w, and if a length-n string u occurs in w, then u^R is not a substring of w.

Example 2

Consider $\mathcal{S} = 001011$.

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.
- To uniquely determine position and orientation, it is sufficient for w to contain every length-n binary string at most once *in either direction*.
- Let w^R denote the reversal of w.

Definition 1

We say that a binary circular string w is an *orientable sequence of order n* (or an $\mathcal{OS}(n)$) if every length-n string occurs at most once in w, and if a length-n string u occurs in w, then u^R is not a substring of w.

Example 2

Consider $\mathcal{S} = 001011$. In the forward direction, including the wraparound, $\mathcal{S}$ contains the six 5-tuples 00101, 01011, 10110, 01100, 11001, and 10010;

Orientable sequences

- Let w be a binary circular string corresponding to the sequence of white and black squares on a cyclic track.
- To uniquely determine position and orientation, it is sufficient for w to contain every length-n binary string at most once *in either direction*.
- Let w^R denote the reversal of w.

Definition 1

We say that a binary circular string w is an *orientable sequence of order n* (or an $\mathcal{OS}(n)$) if every length-n string occurs at most once in w, and if a length-n string u occurs in w, then u^R is not a substring of w.

Example 2

Consider $\mathcal{S} = 001011$. In the forward direction, including the wraparound, $\mathcal{S}$ contains the six 5-tuples 00101, 01011, 10110, 01100, 11001, and 10010; in the reverse direction $\mathcal{S}$ contains 11010, 10100, 01001, 10011, 00110, and 01101. Since each substring is unique, $\mathcal{S}$ is an $\mathcal{OS}(5)$ with length (period) six.

Background

- Dai, Martin, Robshaw, and Wild [Cryptography and Coding III (1993)] first introduced orientable sequences.

Background

- Dai, Martin, Robshaw, and Wild [Cryptography and Coding III (1993)] first introduced orientable sequences.
- They give upper U_n and lower L_n bounds on the maximum M_n length of an $\mathcal{OS}(n)$.

Background

- Dai, Martin, Robshaw, and Wild [Cryptography and Coding III (1993)] first introduced orientable sequences.
- They give upper U_n and lower L_n bounds on the maximum M_n length of an $\mathcal{OS}(n)$.
- They show the existence of an $\mathcal{OS}(n)$ with asymptotically optimal length.

Background

- Dai, Martin, Robshaw, and Wild [Cryptography and Coding III (1993)] first introduced orientable sequences.
- They give upper U_n and lower L_n bounds on the maximum M_n length of an $\mathcal{OS}(n)$.
- They show the existence of an $\mathcal{OS}(n)$ with asymptotically optimal length.
  - Did not give a constructive proof.

Background

- Dai, Martin, Robshaw, and Wild [Cryptography and Coding III (1993)] first introduced orientable sequences.
- They give upper U_n and lower L_n bounds on the maximum M_n length of an $\mathcal{OS}(n)$.
- They show the existence of an $\mathcal{OS}(n)$ with asymptotically optimal length.
  - Did not give a constructive proof.
  - Left the problem of efficiently generating such a sequence open.

Background

Dai et al. showed that L_n is the following, where μ is the Möbius function:

$$L_n = \left(2^{n-1} - \frac{1}{2}\sum_{d|n}\mu(n/d)\frac{n}{d}H(d)\right), \quad \text{where} \quad H(d) = \frac{1}{2}\sum_{i|d} i\left(2^{\lfloor \frac{i+1}{2} \rfloor} + 2^{\lfloor \frac{i}{2} \rfloor + 1}\right).$$

Background

Dai et al. showed that L_n is the following, where μ is the Möbius function:

$$L_n = \left(2^{n-1} - \frac{1}{2}\sum_{d|n} \mu(n/d)\frac{n}{d}H(d)\right), \quad \text{where} \quad H(d) = \frac{1}{2}\sum_{i|d} i\left(2^{\lfloor \frac{i+1}{2} \rfloor} + 2^{\lfloor \frac{i}{2} \rfloor + 1}\right).$$

Their upper bound U_n is the following:

$$U_n = \begin{cases} 2^{n-1} - \frac{41}{9}2^{\frac{n}{2}-1} + \frac{n}{3} + \frac{16}{9} & \text{if } n \bmod 4 = 0, \\ 2^{n-1} - \frac{31}{9}2^{\frac{n-1}{2}} + \frac{n}{3} + \frac{19}{9} & \text{if } n \bmod 4 = 1, \\ 2^{n-1} - \frac{41}{9}2^{\frac{n}{2}-1} + \frac{n}{6} + \frac{20}{9} & \text{if } n \bmod 4 = 2, \\ 2^{n-1} - \frac{31}{9}2^{\frac{n-1}{2}} + \frac{n}{6} + \frac{43}{18} & \text{if } n \bmod 4 = 3. \end{cases}$$

Background

Dai et al. showed that L_n is the following, where μ is the Möbius function:

$$L_n = \left(2^{n-1} - \frac{1}{2}\sum_{d|n} \mu(n/d)\frac{n}{d}H(d)\right), \quad \text{where} \quad H(d) = \frac{1}{2}\sum_{i|d} i\left(2^{\lfloor \frac{i+1}{2} \rfloor} + 2^{\lfloor \frac{i}{2} \rfloor + 1}\right).$$

Their upper bound U_n is the following:

$$U_n = \begin{cases} 2^{n-1} - \frac{41}{9}2^{\frac{n}{2}-1} + \frac{n}{3} + \frac{16}{9} & \text{if } n \bmod 4 = 0, \\ 2^{n-1} - \frac{31}{9}2^{\frac{n-1}{2}} + \frac{n}{3} + \frac{19}{9} & \text{if } n \bmod 4 = 1, \\ 2^{n-1} - \frac{41}{9}2^{\frac{n}{2}-1} + \frac{n}{6} + \frac{20}{9} & \text{if } n \bmod 4 = 2, \\ 2^{n-1} - \frac{31}{9}2^{\frac{n-1}{2}} + \frac{n}{6} + \frac{43}{18} & \text{if } n \bmod 4 = 3. \end{cases}$$

- Mitchell and Wild [IEEE Transactions on Information Theory (2022)] recently gave a recursive construction that generates an $\mathcal{OS}(n)$ from an $\mathcal{OS}(n-1)$.

Background

Dai et al. showed that L_n is the following, where μ is the Möbius function:

$$L_n = \left(2^{n-1} - \frac{1}{2} \sum_{d|n} \mu(n/d) \frac{n}{d} H(d) \right), \quad \text{where} \quad H(d) = \frac{1}{2} \sum_{i|d} i \left(2^{\lfloor \frac{i+1}{2} \rfloor} + 2^{\lfloor \frac{i}{2} \rfloor + 1} \right).$$

Their upper bound U_n is the following:

$$U_n = \begin{cases} 2^{n-1} - \frac{41}{9} 2^{\frac{n}{2}-1} + \frac{n}{3} + \frac{16}{9} & \text{if } n \bmod 4 = 0, \\ 2^{n-1} - \frac{31}{9} 2^{\frac{n-1}{2}} + \frac{n}{3} + \frac{19}{9} & \text{if } n \bmod 4 = 1, \\ 2^{n-1} - \frac{41}{9} 2^{\frac{n}{2}-1} + \frac{n}{6} + \frac{20}{9} & \text{if } n \bmod 4 = 2, \\ 2^{n-1} - \frac{31}{9} 2^{\frac{n-1}{2}} + \frac{n}{6} + \frac{43}{18} & \text{if } n \bmod 4 = 3. \end{cases}$$

- Mitchell and Wild [IEEE Transactions on Information Theory (2022)] recently gave a recursive construction that generates an $\mathcal{OS}(n)$ from an $\mathcal{OS}(n-1)$.
- The length of their constructed $\mathcal{OS}(n)$ is not of asymptotically optimal length.
- Their construction requires storing the whole string.

Main results: a successor rule

- Let Σ be some finite nonempty alphabet.
- Let w be a circular string over Σ such that for some $n \geq 1$, every length-n substring of w occurs exactly once.

Main results: a successor rule

- Let Σ be some finite nonempty alphabet.
- Let w be a circular string over Σ such that for some $n \geq 1$, every length-n substring of w occurs exactly once.

Definition 3

A function $g : \Sigma^n \to \Sigma$ is a *successor rule* for w, if $ug(u)$ is a substring of w for any length-n substring u of w.

Main results: a successor rule

- Let Σ be some finite nonempty alphabet.
- Let w be a circular string over Σ such that for some $n \geq 1$, every length-n substring of w occurs exactly once.

Definition 3

A function $g : \Sigma^n \to \Sigma$ is a *successor rule* for w, if $ug(u)$ is a substring of w for any length-n substring u of w.

- For example, the function $g : \{0,1\}^3 \to \{0,1\}$ defined by $g(\mathrm{a}_1\mathrm{a}_2\mathrm{a}_3) = (\mathrm{a}_1 + 1) \bmod 2$ is a successor rule for 000111.

Main results: a successor rule

- Let Σ be some finite nonempty alphabet.
- Let w be a circular string over Σ such that for some $n \geq 1$, every length-n substring of w occurs exactly once.

Definition 3

A function $g : \Sigma^n \to \Sigma$ is a *successor rule* for w, if $ug(u)$ is a substring of w for any length-n substring u of w.

- For example, the function $g : \{0, 1\}^3 \to \{0, 1\}$ defined by $g(\mathrm{a}_1\mathrm{a}_2\mathrm{a}_3) = (\mathrm{a}_1 + 1) \bmod 2$ is a successor rule for 000111.

- We develop a successor rule that can be used to generate an $\mathcal{OS}(n)$ of length L_n in $O(n)$ time per bit using $O(n)$ space.

Main results: a successor rule

- Let Σ be some finite nonempty alphabet.
- Let w be a circular string over Σ such that for some $n \geq 1$, every length-n substring of w occurs exactly once.

Definition 3

A function $g : \Sigma^n \to \Sigma$ is a *successor rule* for w, if $ug(u)$ is a substring of w for any length-n substring u of w.

- For example, the function $g : \{0,1\}^3 \to \{0,1\}$ defined by $g(\mathtt{a}_1\mathtt{a}_2\mathtt{a}_3) = (\mathtt{a}_1 + 1) \bmod 2$ is a successor rule for 000111.

- We develop a successor rule that can be used to generate an $\mathcal{OS}(n)$ of length L_n in $O(n)$ time per bit using $O(n)$ space.
  - Space complexity is logarithmic in L_n.
  - Time complexity (per bit) is logarithmic in L_n.

Strategy

Strategy

- Take a class of short and easy-to-describe orientable sequences with different length-n substrings.

Strategy

- Take a class of short and easy-to-describe orientable sequences with different length-n substrings.
- Use cycle-joining to join all these “short” orientable sequences into one large orientable sequence of length L_n.

Cycle-joining

- Cycle-joining is often used to construct de Bruijn sequences from disjoint cycle covers of the de Bruijn graph.

Cycle-joining

- Cycle-joining is often used to construct de Bruijn sequences from disjoint cycle covers of the de Bruijn graph.
- Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_m$ and $\beta = \mathtt{b}_1\mathtt{b}_2\cdots\mathtt{b}_\ell$ be circular strings.

Cycle-joining

- Cycle-joining is often used to construct de Bruijn sequences from disjoint cycle covers of the de Bruijn graph.
- Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_m$ and $\beta = \mathtt{b}_1\mathtt{b}_2\cdots\mathtt{b}_\ell$ be circular strings.
- If the sets S_1 and S_2 of length-n substrings of α and β respectively are disjoint, and α and β share a string of length $n-1$ in common, then α and β can be cycle-joined to a string γ with the set of its length-n substrings being $S_1 \cup S_2$.

Cycle-joining

- Cycle-joining is often used to construct de Bruijn sequences from disjoint cycle covers of the de Bruijn graph.
- Let $\alpha = \mathtt{a}_1\mathtt{a}_2 \cdots \mathtt{a}_m$ and $\beta = \mathtt{b}_1\mathtt{b}_2 \cdots \mathtt{b}_\ell$ be circular strings.
- If the sets S_1 and S_2 of length-n substrings of α and β respectively are disjoint, and α and β share a string of length $n-1$ in common, then α and β can be cycle-joined to a string γ with the set of its length-n substrings being $S_1 \cup S_2$.
- Suppose α and β share the length-$(n-1)$ substring u.

Cycle-joining

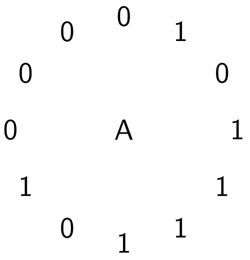
- Cycle-joining is often used to construct de Bruijn sequences from disjoint cycle covers of the de Bruijn graph.
- Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_m$ and $\beta = \mathtt{b}_1\mathtt{b}_2\cdots\mathtt{b}_\ell$ be circular strings.
- If the sets S_1 and S_2 of length-n substrings of α and β respectively are disjoint, and α and β share a string of length $n-1$ in common, then α and β can be cycle-joined to a string γ with the set of its length-n substrings being $S_1 \cup S_2$.
- Suppose α and β share the length-$(n-1)$ substring u.
- Let a, b, c, and d be symbols such that aub is a substring of α and cud is a substring of β.

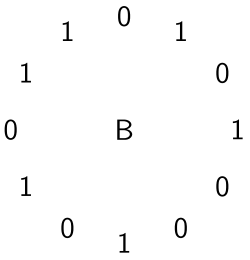
Cycle-joining

- Cycle-joining is often used to construct de Bruijn sequences from disjoint cycle covers of the de Bruijn graph.
- Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_m$ and $\beta = \mathtt{b}_1\mathtt{b}_2\cdots\mathtt{b}_\ell$ be circular strings.
- If the sets S_1 and S_2 of length-n substrings of α and β respectively are disjoint, and α and β share a string of length $n-1$ in common, then α and β can be cycle-joined to a string γ with the set of its length-n substrings being $S_1 \cup S_2$.
- Suppose α and β share the length-$(n-1)$ substring u.
- Let a, b, c, and d be symbols such that aub is a substring of α and cud is a substring of β.
- Then α and β can be cycle-joined by exchanging the successor of au with the successor of cu, so that aud and cub are substrings of the cycle-joined result.

Cycle-joining example

Let $n = 6$.

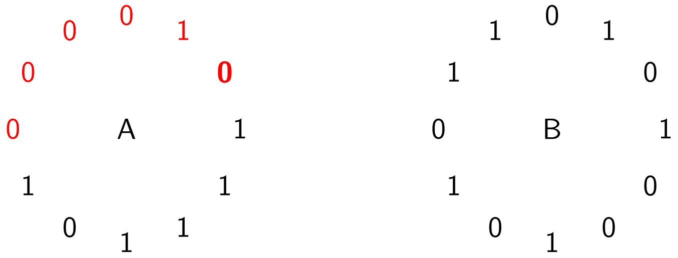




$A = 000010111101$ $B = 001010110101$

Cycle-joining example

Let $n = 6$.

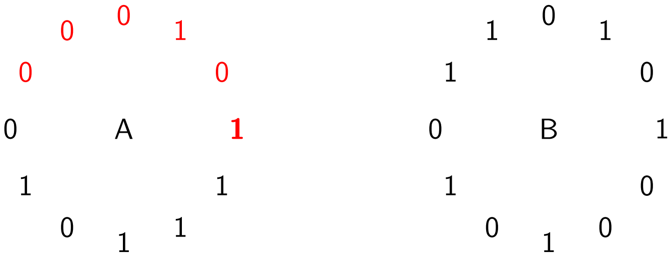


$A = 000010111101$ $\quad B = 001010110101$

A cycle-join $B = \mathbf{0}$

Cycle-joining example

Let $n = 6$.

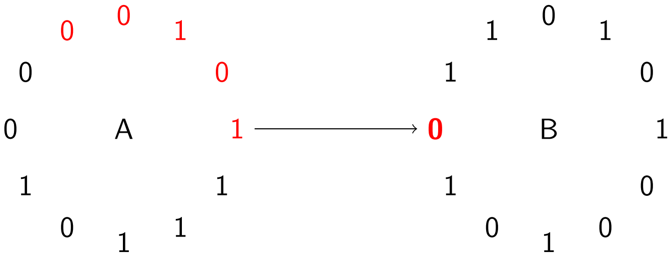


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 01$**1**

Cycle-joining example

Let $n = 6$.

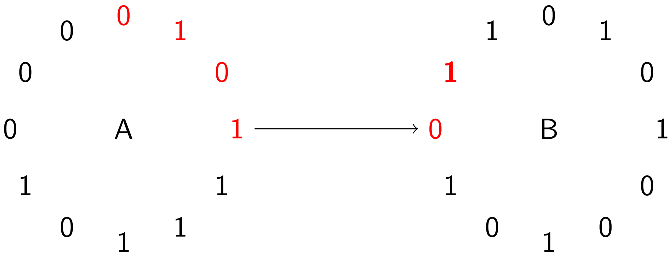


$A = 000010111101$ $\quad$ $B = 001010110101$

A cycle-join $B = 01\mathbf{0}$

Cycle-joining example

Let $n = 6$.

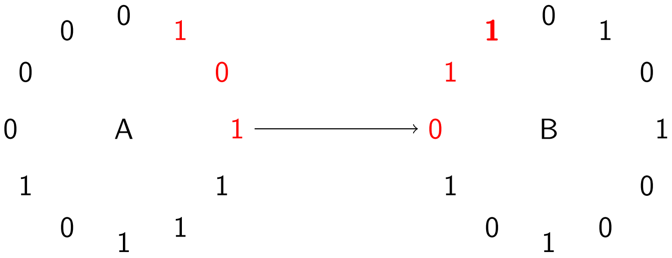


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 010\mathbf{1}$

Cycle-joining example

Let $n = 6$.

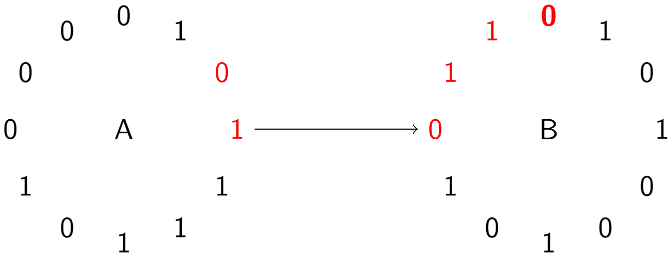


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 0101\mathbf{1}$$

Cycle-joining example

Let $n = 6$.

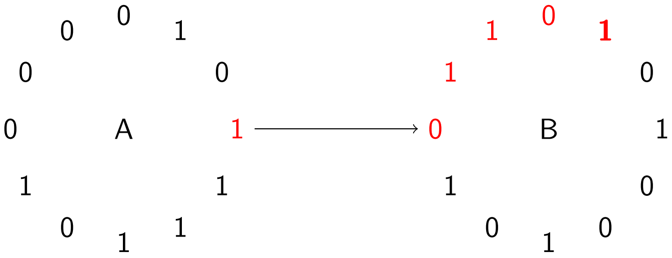


$$A = 000010111101 \qquad B = 001010110101$$

A cycle-join $B = 01011\mathbf{0}$

Cycle-joining example

Let $n = 6$.

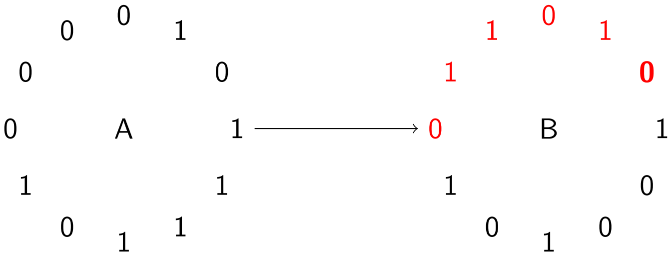


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 010110\mathbf{1}$

Cycle-joining example

Let $n = 6$.

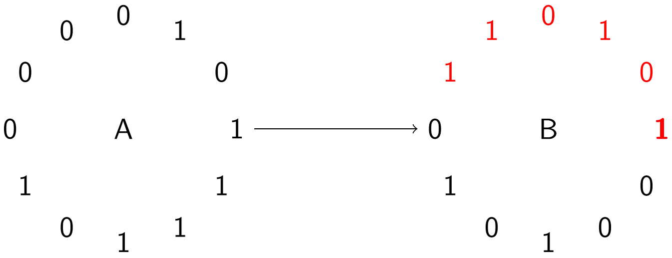


$A = 000010111101$ $\quad$ $B = 001010110101$

A cycle-join $B = 01011010$

Cycle-joining example

Let $n = 6$.

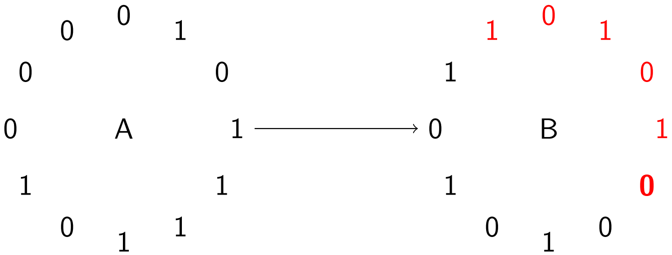


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 01011010\mathbf{1}$

Cycle-joining example

Let $n = 6$.

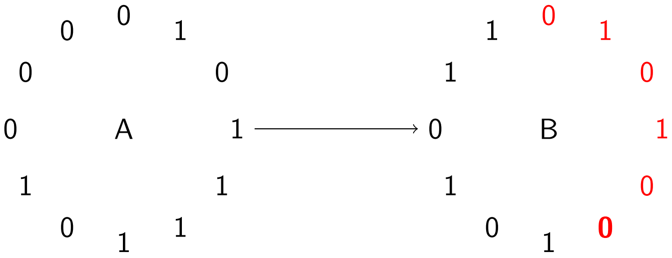


$A = 000010111101$ $\qquad$ $B = 001010110101$

A cycle-join $B = 010110101\mathbf{0}$

Cycle-joining example

Let $n = 6$.

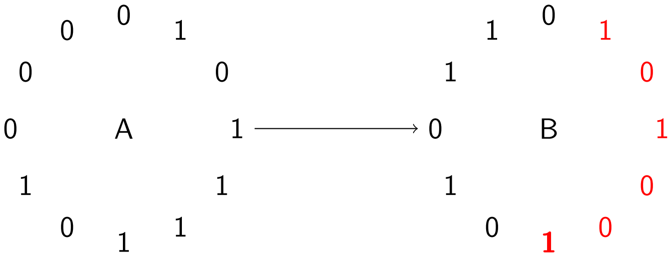


$A = 000010111101$ $\quad$ $B = 001010110101$

A cycle-join $B = 01011010100$**0**

Cycle-joining example

Let $n = 6$.

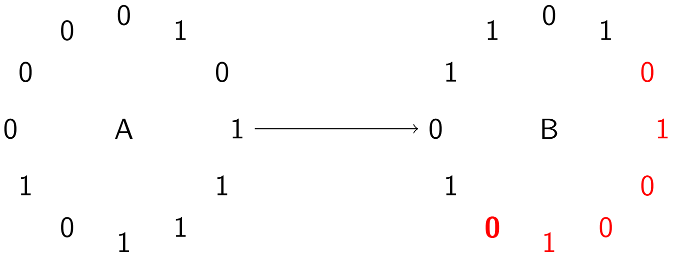


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 01011010100\mathbf{1}$

Cycle-joining example

Let $n = 6$.

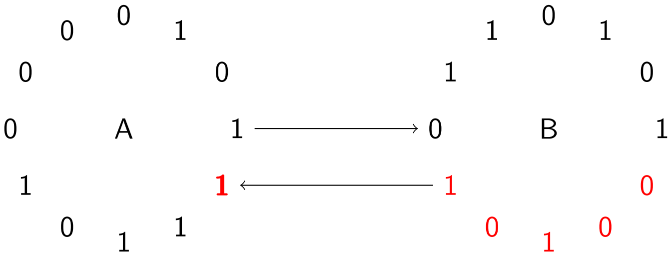


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 01011010100\mathbf{10}$$

Cycle-joining example

Let $n = 6$.

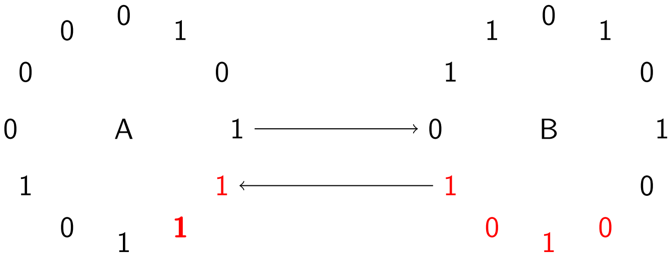


$A = 000010111101$ $\qquad$ $B = 001010110101$

A cycle-join $B = 0101101010010\mathbf{1}1$

Cycle-joining example

Let $n = 6$.

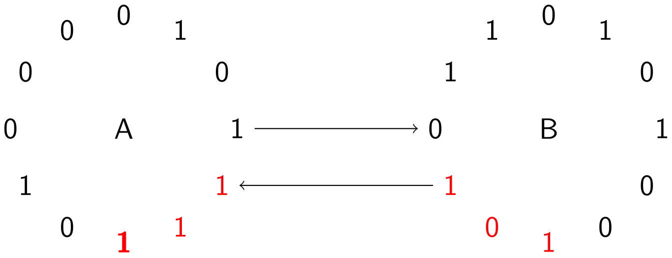


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 0101101010010111$$

Cycle-joining example

Let $n = 6$.

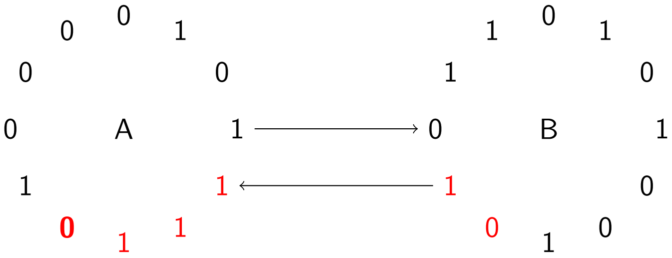


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 0101101010010111\mathbf{1}$

Cycle-joining example

Let $n = 6$.

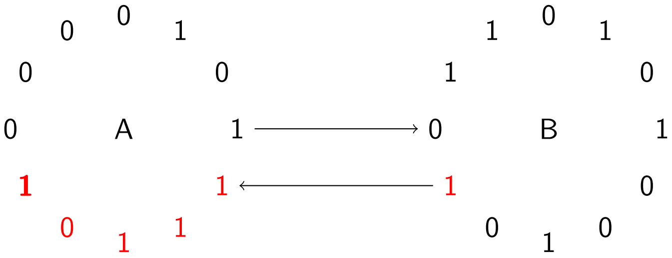


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 0101101010010111\mathbf{1}0$$

Cycle-joining example

Let $n = 6$.

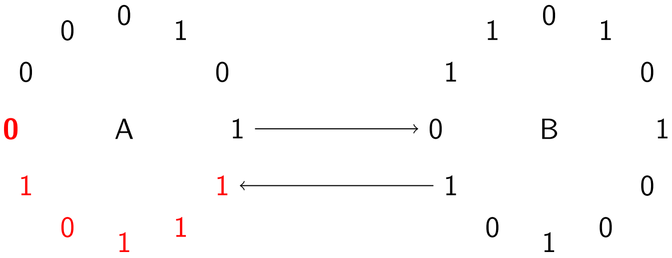


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 010110101001011110\mathbf{1}$$

Cycle-joining example

Let $n = 6$.

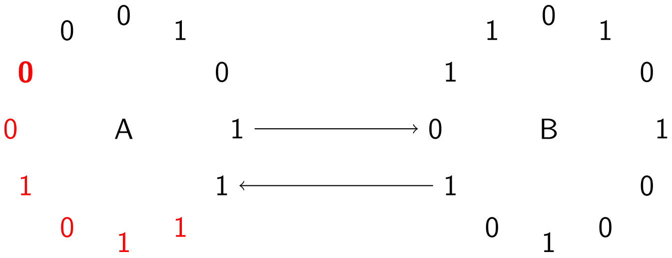


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 01011010100101111010$$

Cycle-joining example

Let $n = 6$.

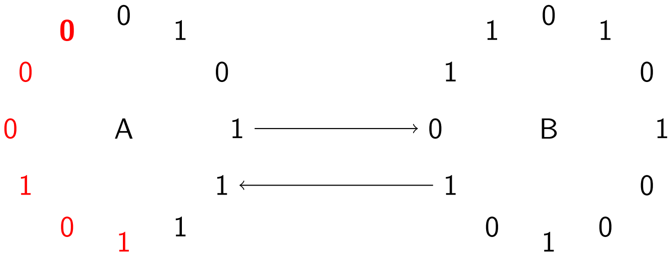


$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 0101101010010111101000$

Cycle-joining example

Let $n = 6$.

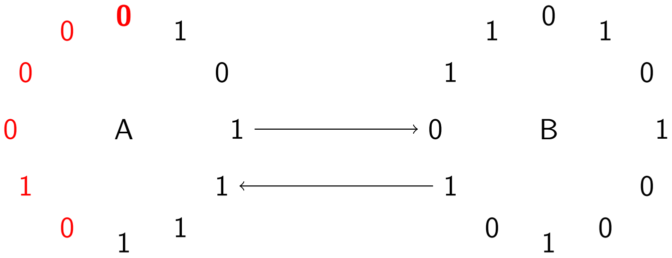


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 010110101001011110100\mathbf{0}$$

Cycle-joining example

Let $n = 6$.

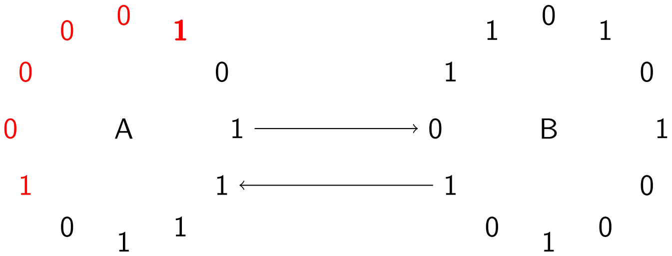


$$A = 000010111101 \qquad B = 001010110101$$

$$A \text{ cycle-join } B = 0101101010010111101000\mathbf{0}$$

Cycle-joining example

Let $n = 6$.



$A = 000010111101$ $B = 001010110101$

A cycle-join $B = 01011010100101111010000$**1**

Bracelets and Necklaces

Let α be a length-n string.

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.
  - For example, the necklace class of the string 001011 is

$$A = \{001011, 010110, 101100, 011001, 110010, 100101\}.$$

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.
  - For example, the necklace class of the string 001011 is

$$A = \{001011, 010110, 101100, 011001, 110010, 100101\}.$$

- We say that α is a *necklace* if it is the lexicographically smallest string in its necklace class.

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.
  - For example, the necklace class of the string 001011 is

$$A = \{001011, 010110, 101100, 011001, 110010, 100101\}.$$

- We say that α is a *necklace* if it is the lexicographically smallest string in its necklace class.
- A *bracelet class* is an equivalence class under rotation *and* reversal.

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.
  - For example, the necklace class of the string 001011 is

$$A = \{001011, 010110, 101100, 011001, 110010, 100101\}.$$

- We say that α is a *necklace* if it is the lexicographically smallest string in its necklace class.
- A *bracelet class* is an equivalence class under rotation *and* reversal.
  - For example, the bracelet class of 001011 is

$$A \cup \{110100, 101001, 010011, 100110, 001101, 011010\}.$$

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.
  - For example, the necklace class of the string 001011 is

$$A = \{001011, 010110, 101100, 011001, 110010, 100101\}.$$

- We say that α is a *necklace* if it is the lexicographically smallest string in its necklace class.
- A *bracelet class* is an equivalence class under rotation *and* reversal.
  - For example, the bracelet class of 001011 is

$$A \cup \{110100, 101001, 010011, 100110, 001101, 011010\}.$$

- We say that α is a *bracelet* if it is the lexicographically smallest string in its bracelet class.

Bracelets and Necklaces

Let α be a length-n string.

- A *necklace class* is an equivalence class under rotation.
  - For example, the necklace class of the string 001011 is

$$A = \{001011, 010110, 101100, 011001, 110010, 100101\}.$$

- We say that α is a *necklace* if it is the lexicographically smallest string in its necklace class.
- A *bracelet class* is an equivalence class under rotation *and* reversal.
  - For example, the bracelet class of 001011 is

$$A \cup \{110100, 101001, 010011, 100110, 001101, 011010\}.$$

- We say that α is a *bracelet* if it is the lexicographically smallest string in its bracelet class.
- For example, the string 001011 is both a necklace and bracelet, and the string 001101 is a necklace but not a bracelet.

Symmetric and asymmetric bracelets

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1\beta_2$.

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1\beta_2$.
- If a necklace or bracelet is not symmetric, it is said to be *asymmetric*.
- For example...

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1\beta_2$.
- If a necklace or bracelet is not symmetric, it is said to be *asymmetric*.
- For example...
  - The string 0011011 is a symmetric bracelet.

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1\beta_2$.
- If a necklace or bracelet is not symmetric, it is said to be *asymmetric*.
- For example...
  - The string 0011011 is a symmetric bracelet.
  - The string 0001011 is an asymmetric bracelet.

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1\beta_2$.
- If a necklace or bracelet is not symmetric, it is said to be *asymmetric*.
- For example...
  - The string 0011011 is a symmetric bracelet.
  - The string 0001011 is an asymmetric bracelet.
  - The string 0001101 is an asymmetric necklace, but not an asymmetric bracelet.

Symmetric and asymmetric bracelets

- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1 \beta_2$.
- If a necklace or bracelet is not symmetric, it is said to be *asymmetric*.
- For example...
  - The string 0011011 is a symmetric bracelet.
  - The string 0001011 is an asymmetric bracelet.
  - The string 0001101 is an asymmetric necklace, but not an asymmetric bracelet.
- Dai et al. proved the existence of an algorithm to cycle-join all asymmetric bracelets into an $\mathcal{OS}(n)$ with asymptotically optimal length.

Symmetric and asymmetric bracelets

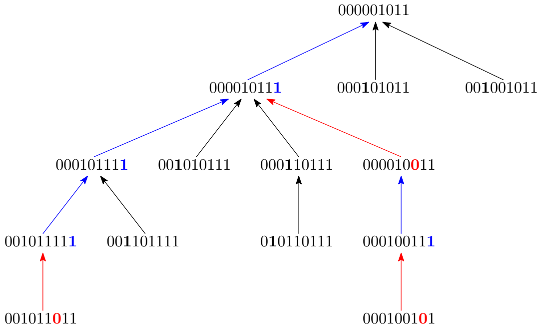
- A necklace α is *symmetric* if it belongs to the same necklace class as α^R.
  - By this definition, a symmetric necklace is necessarily a bracelet.
  - By a well-known folklore result, a necklace is symmetric if and only if there exist two palindromes β_1, β_2 such that $\alpha = \beta_1\beta_2$.
- If a necklace or bracelet is not symmetric, it is said to be *asymmetric*.
- For example...
  - The string 0011011 is a symmetric bracelet.
  - The string 0001011 is an asymmetric bracelet.
  - The string 0001101 is an asymmetric necklace, but not an asymmetric bracelet.
- Dai et al. proved the existence of an algorithm to cycle-join all asymmetric bracelets into an $\mathcal{OS}(n)$ with asymptotically optimal length.
- They did not explicitly give an algorithm to construct such an $\mathcal{OS}(n)$.

Parent rule

Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n$ be an asymmetric bracelet of length n.

- first1(α) be the necklace $\mathtt{a}_1\cdots\mathtt{a}_{i-1}\mathbf{0}\mathtt{a}_{i+1}\cdots\mathtt{a}_n$, where i is the index of the first 1 in α;
- last1(α) be the necklace in the necklace class of $\mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_{n-1}\mathbf{0}$;
- last0(α) be the necklace $\mathtt{a}_1\cdots\mathtt{a}_{j-1}\mathbf{1}\mathtt{a}_{j+1}\cdots\mathtt{a}_n$, where j is the index of the last 0 in α.

$$\text{par}(\alpha) = \begin{cases} \text{first1}(\alpha), & \text{if first1}(\alpha) \text{ is asymmetric;} \\ \text{last1}(\alpha), & \text{if first1}(\alpha) \text{ is not asymmetric and last1}(\alpha) \text{ is asymmetric;} \\ \text{last0}(\alpha), & \text{otherwise.} \end{cases}$$



Successor-rule g to construct an $\mathcal{OS}(n)$ of length L_n

- Let $\mathbf{A}(n)$ denote the set of length-n asymmetric bracelets.
- Let $\mathbf{S}(n)$ denote the set of all rotations of strings in $\mathbf{A}(n)$.

Let $\alpha = \mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n \in \mathbf{S}(n)$ and let

- $\beta_1 = 0^{n-i}\mathbf{1}\mathtt{a}_2\cdots\mathtt{a}_i$ where i is the largest index of α such that $\mathtt{a}_i = 1$ (first 1);
- $\beta_2 = \mathtt{a}_2\mathtt{a}_3\cdots\mathtt{a}_n\mathbf{1}$ (last 1);
- $\beta_3 = \mathtt{a}_j\mathtt{a}_{j+1}\cdots\mathtt{a}_n\mathbf{0}1^{j-2}$ where j is the smallest index of α such that $\mathtt{a}_j = 0$ and $j > 1$ (last 0).

Let

$$g(\alpha) = \begin{cases} \overline{\mathtt{a}}_1, & \text{if } \beta_1 \text{ and first1}(\beta_1) \text{ are in } \mathbf{A}(n); \\ \overline{\mathtt{a}}_1, & \text{if } \beta_2 \text{ and last1}(\beta_2) \text{ are in } \mathbf{A}(n), \text{ and first1}(\beta_2) \text{ is not in } \mathbf{A}(n); \\ \overline{\mathtt{a}}_1, & \text{if } \beta_3 \text{ and last0}(\beta_3) \text{ are in } \mathbf{A}(n), \text{ and neither first1}(\beta_3) \text{ nor last1}(\beta_3) \text{ are in } \mathbf{A}(n); \\ \mathtt{a}_1, & \text{otherwise.} \end{cases}$$

Theorem 4

The function g is a successor rule that generates an $\mathcal{OS}(n)$ with length L_n for the set $\mathbf{S}(n)$ in $O(n)$-time per bit using $O(n)$ space.

Conclusions and Open problems

Conclusions and Open problems

- We resolved an open problem by Dai et al., by describing a procedure to generate a binary orientable sequence of order n with asymptotically optimal length in $O(n)$ time per bit, using $O(n)$ space.

Conclusions and Open problems

- We resolved an open problem by Dai et al., by describing a procedure to generate a binary orientable sequence of order n with asymptotically optimal length in $O(n)$ time per bit, using $O(n)$ space.
- A natural open problem, related to the autonomous robot application, is decoding or unranking orientable sequences.

Conclusions and Open problems

- We resolved an open problem by Dai et al., by describing a procedure to generate a binary orientable sequence of order n with asymptotically optimal length in $O(n)$ time per bit, using $O(n)$ space.
- A natural open problem, related to the autonomous robot application, is decoding or unranking orientable sequences.
- That is, given an arbitrary length-n substring of an $\mathcal{OS}(n)$, efficiently determine where in the sequence this substring is located.

Conclusions and Open problems

- We resolved an open problem by Dai et al., by describing a procedure to generate a binary orientable sequence of order n with asymptotically optimal length in $O(n)$ time per bit, using $O(n)$ space.
- A natural open problem, related to the autonomous robot application, is decoding or unranking orientable sequences.
- That is, given an arbitrary length-n substring of an $\mathcal{OS}(n)$, efficiently determine where in the sequence this substring is located.
- There has been no progress in this direction.

Conclusions and Open problems

- We resolved an open problem by Dai et al., by describing a procedure to generate a binary orientable sequence of order n with asymptotically optimal length in $O(n)$ time per bit, using $O(n)$ space.
- A natural open problem, related to the autonomous robot application, is decoding or unranking orientable sequences.
- That is, given an arbitrary length-n substring of an $\mathcal{OS}(n)$, efficiently determine where in the sequence this substring is located.
- There has been no progress in this direction.
- Even in the well-studied area of de Bruijn sequences, only a few efficient decoding algorithms have been discovered.

Conclusions and Open problems

- We resolved an open problem by Dai et al., by describing a procedure to generate a binary orientable sequence of order n with asymptotically optimal length in $O(n)$ time per bit, using $O(n)$ space.
- A natural open problem, related to the autonomous robot application, is decoding or unranking orientable sequences.
- That is, given an arbitrary length-n substring of an $\mathcal{OS}(n)$, efficiently determine where in the sequence this substring is located.
- There has been no progress in this direction.
- Even in the well-studied area of de Bruijn sequences, only a few efficient decoding algorithms have been discovered.
- See `https://debruijnsequence.org/db/orientable` for a complete implementation of the successor rule.