A de Bruijn Sequence Construction by Concatenating Cycles of the Complemented Cycling Register

Daniel Gabric[1] and Joe Sawada[2]

[1] University of Guelph, Canada, dgabric@uoguelph.ca
[2] University of Guelph, Canada, jsawada@uoguelph.ca

Abstract. We present a new de Bruijn sequence construction based on co-necklaces and the complemented cycling register (CCR). A co-necklace is the lexicographically smallest string in an equivalence class of strings induced by the CCR. We prove that a concatenation of the cycles of the CCR forms a de Bruijn sequence when the cycles are ordered in colexicographic order with respect to their co-necklace representatives. We also give an algorithm that produces the de Bruijn sequence in $O(1)$-time per bit. Finally, we prove that our construction has a discrepancy bounded above by $2n$.

1 Introduction

Let $\mathbf{B}(n)$ be the set of binary strings of length n. It is well known that the pure cycling register, which takes a binary string and outputs its first bit, partitions $\mathbf{B}(n)$ into equivalence classes under rotation. The lexicographically smallest representative of each equivalence class is called a necklace. For $n = 5$, the eight necklace equivalence classes are listed in columns as follows:

00000	**0**0001	**0**0011	**0**0101	**0**0111	**0**1011	**0**1111	**1**1111.
	00010	**0**0110	**0**1010	**0**1110	**1**0110	**1**1110	
	00100	**0**1100	**1**0100	**1**1100	**0**1101	**1**1101	
	01000	**1**1000	**0**1001	**1**1001	**1**1010	**1**1011	
	10000	**1**0001	**1**0010	**1**0011	**1**0101	**1**0111	

The first string in each equivalence class is its necklace representative and the necklaces are listed from left to right in lexicographic order. For each equivalence class, observe that the string obtained by concatenating the first bit from each string yields the longest aperiodic prefix of the necklace representative. Now consider the string of length $2^5 = 32$ obtained by concatenating these highlighted bits (top down, then left to right):

$$0\ 00001\ 00011\ 00101\ 00111\ 01011\ 011111.$$

Amazingly, when considered cyclicly, this constructed string contains every string in $\mathbf{B}(5)$ as a substring exactly once. Strings with this property for a given n are called de Bruijn sequences.

In this paper, we show a similar property with respect to the complemented cycling register (CCR), which takes a binary string and outputs the complement of the first bit. The CCR partitions $\mathbf{B}(n)$ into equivalence classes of size up to $2n$. We call the lexicographically smallest string in each such equivalence class a co-necklace. For $n = 5$, the four co-necklace equivalence classes are listed in columns as follows:

00000	**0**0010	**0**0100	**0**1010
00001	**0**0101	**0**1001	**1**0101.
00011	**0**1011	**1**0011	
00111	**1**0111	**0**0110	
01111	**0**1110	**0**1101	
11111	**1**1101	**1**1011	
11110	**1**1010	**1**0110	
11100	**1**0100	**0**1100	
11000	**0**1000	**1**1001	
10000	**1**0001	**1**0010	

Observe that the co-necklace representative is positioned at the top of each class, and the classes are ordered in lexicographic order with respect to the co-necklaces. Within each equivalence class, each successive string is a left rotation of the string above it after complementing the final bit. For each equivalence class, let α denote the co-necklace and observe that the string obtained by concatenating together the first bit from each string yields the longest aperiodic prefix of $\alpha\overline{\alpha}$, where $\overline{\alpha}$ denotes the complement of α. We call such a string a *cycle of the CCR*. Consider the string obtained by concatenating these cycles of the CCR (top down, then left to right):

$$\mathbf{0}000011111\ 000101\mathbf{1101}\ \mathbf{0}0100110\mathbf{11}\ \mathbf{01}.$$

This string is not a de Bruijn sequence since it contains the substring 11010 twice. It also contains the substring 10100 twice and is missing 01010 and 10101. However, observe what happens if we list the equivalence classes in colexicographic order with respect to the co-necklace representatives. This listing is obtained by swapping the second and third classes from the previous lexicographic order example. Using this colexicographic order and concatenating the CCR cycles together yields the following de Bruijn sequence

$$0000011111\ 0010011011\ 0001011101\ 01.$$

Main result: The main result of this paper is to provide a simple proof that this construction using co-necklaces and CCR cycles, for arbitrary n, yields a de Bruijn sequence. We provide an algorithm to generate the co-necklaces of length n in colexicographic order that runs in $O(n)$-amortized time per string. This allows us to generate the de Bruijn sequence in $O(1)$-time per it. Additionally, we demonstrate that the constructed de Bruijn sequences have a very nice property: they have discrepancy bounded above by $2n$.

In the next subsection, we provide some history of de Bruijn sequence constructions and the complemented cycling register. Then in Section 2, we present formal definitions of our key objects and notation. In Section 3, we prove our main result which includes implementation details and an analysis. Finally in Section 4 we discuss an interesting property of our constructed de Bruijn sequences.

1.1 History

The lexicographic necklace concatenation approach was first presented by Fredricksen and Maiorana [13]. An algorithm to generate necklaces in lexicographic order [12] was shown to run in $O(1)$-amortized time per necklace [19] which means the corresponding de Bruijn sequence can be generated in $O(1)$-amortized time per bit. Rather surprisingly, it was only recently discovered that a similar algorithm works for necklaces in colexicographic order [6]. Since necklaces can also be generated in colexicographic order in $O(1)$-time [21], the corresponding de Bruijn sequence can also be generated in $O(1)$-amortized time per bit.

Other methods for generating de Bruijn sequences of order n include greedy approaches and successor-based approaches. A major drawback of the greedy approaches [2, 7, 10, 17] is they all require an exponential amount of space. The first successor-rule based approach was by Fredricksen [11] for the lexicographically smallest de Bruijn sequence, which also happens to be equivalent to the lexicographic necklace concatenation approach. Additional successor based approaches [8, 9, 16] generate de Bruijn sequences requiring $O(n^2)$-time per bit. In particular, the constructions by Etzion [8] and Huang [16] are also based on the CCR. A more recent construction [20] based on the PCR requires $O(n)$-time to compute each successive bit, and an optimization allows the entire sequence to be generated in $O(1)$-amortized time per bit.

There is a well known correspondence between co-necklaces of order n and necklaces of order n that contain an odd number of 1s. A discussion from [4] describes how representatives from these equivalence classes can be generated in $O(1)$-amortize time per string. However, there is no efficient algorithm known to generate the lexicographically smallest representatives, the co-necklaes, in either lexicographic or colexicographic order. The enumeration sequence for co-necklaces A000016 was one of the first listed in the Online Encyclopedia of Integer Sequences [1].

2 Background and definitions

A *necklace* is the lexicographically smallest string in an equivlance class of strings under rotation. Let $\overline{\alpha}$ be the complement of the string α. We say that α is a *co-necklace* if $\alpha\overline{\alpha}$ is a necklace. Let $\mathbf{coneck}(\alpha)$ denote the set containing all length n substrings of the circular string $\alpha\overline{\alpha}$. For example,

$$\mathbf{coneck}(00000) = \{00000, 00001, 00011, 00111, 01111, 11111, 11110, 11100, 11000, 10000\}.$$

Let $\mathbf{coN}(n)$ be the set of all co-necklaces of length n, so $\mathbf{coN}(5) = \{00000, 00010, 00100, 01010\}$. Clearly every distinct **coneck** set contains a co-necklace, which is it's lexicographically least element. It is also well known that $\{\mathbf{coneck}(\alpha) \mid \alpha \in \mathbf{coN}(n)\}$ is a partition of $\mathbf{B}(n)$ [4, 8, 15, 18]. The number of co-necklaces of length n is the same as the number of cycles of the CCR (with respect to n) as well as the number of necklaces of length n with an odd number of 1s and is given by the formula [14]

$$|\mathbf{coN}(n)| \;=\; \frac{1}{2n}\sum_{\text{odd } d|n} \phi(d)2^{n/d}, \tag{1}$$

where ϕ is Euler's totient function.

Given a non-empty subset $\mathbf{S}$ of $\mathbf{B}(n)$, a *universal cycle* for $\mathbf{S}$ is a sequence of length $|\mathbf{S}|$ that contains every string in $\mathbf{S}$ as a substring exactly once when the string is viewed circularly. A universal cycle is a *de Bruijn* sequence in the case that $\mathbf{S} = \mathbf{B}(n)$.

The *aperiodic prefix* of α denoted by $ap(\alpha)$ is the shortest prefix $a_1a_2\cdots a_i, i \in \{1, 2, \ldots, n\}$ such that $\alpha = (a_1a_2\cdots a_i)^{\frac{n}{i}}$. We say α is *periodic* if $|ap(\alpha)| < |\alpha|$ and is *aperiodic* if $|ap(\alpha)| = |\alpha|$.

Lemma 1. *If $\alpha = a_1a_2\cdots a_n$ is a binary string and $\alpha\overline{\alpha}$ is periodic, then $\frac{|\alpha\overline{\alpha}|}{|ap(\alpha\overline{\alpha})|}$ is odd.*

Proof. The proof is by contradiction. Assume $\alpha\overline{\alpha}$ is periodic and $\frac{|\alpha\overline{\alpha}|}{|ap(\alpha\overline{\alpha})|} = 2k$ is even. Then $\alpha\overline{\alpha} = (ap(\alpha\overline{\alpha}))^{2k} = (ap(\alpha\overline{\alpha}))^{k}(ap(\alpha\overline{\alpha}))^{k} = \alpha\alpha$, a contradiction. □

Lemma 2. *If $\alpha = a_1a_2\cdots a_n$ is a binary string and $\beta = ap(\alpha\overline{\alpha})$, then $\beta = a_1a_2\cdots a_i\overline{a_1a_2\cdots a_i}$ for some $1 \le i \le n$.*

Proof. If $\alpha\overline{\alpha}$ is aperiodic, then $\beta = \alpha\overline{\alpha} = a_1a_2\cdots a_n\overline{a_1a_2\cdots a_n}$. If $\alpha\overline{\alpha}$ is periodic, then $ap(\alpha\overline{\alpha}) = a_1a_2\cdots a_j$ for some $j \le n$ and by Lemma 1 the value j must be even and $\alpha\overline{\alpha} = (a_1a_2\cdots a_j)^{2k+1} = (a_1a_2\cdots a_j)^{k}(a_1a_2\cdots a_j)(a_1a_2\cdots a_j)^{k}$. Thus, it follows that $a_1a_2\cdots a_{j/2} = \overline{a_{j/2+1}\cdots a_{j-1}a_j}$. □

Let $\alpha = a_1a_2\cdots a_n$ and $\beta = b_1b_2\cdots b_n$ be two distinct binary strings of equal length. Then α comes before β in *colexicographic* (colex) order if $a_i < b_i$ for the largest i where $a_i \ne b_i$.

Lemma 3. *If $\alpha = a_1a_2\cdots a_n$ and $\beta = b_1b_2\cdots b_n$ are consecutive co-necklaces in colex order, where α comes before β and j is the smallest index where $b_j = 1$, then $a_{j+1}a_{j+2}\cdots a_n = b_{j+1}b_{j+2}\cdots b_n$.*

Proof. The proof is by contradiction. Suppose $a_{j+1}a_{j+2}\cdots a_n \ne b_{j+1}b_{j+2}\cdots b_n$. Then there exists some largest $i > j$ such that $a_i \ne b_i$. Since α comes before β in colex order, $a_i = 0$ and $b_i = 1$. However, since β is a co-necklace, then $\gamma = 0^i b_{i+1}b_{i+2}\cdots b_n$ will also be a co-necklace, since 0^i is the largest run of 0's in γ. But this means γ comes between α and β in colex order, which is a contradiction. Thus $a_{j+1}a_{j+2}\cdots a_n = b_{j+1}b_{j+2}\cdots b_n$. □

3 De Bruijn sequence construction

In this section we present a de Bruijn sequence construction obtained by concatenating the cycles of the CCR. The construction generalizes to produce universal cycles for certain subsets of $\mathbf{B}(n)$.

Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the first m co-necklaces of length n in colex order. Let

$$\mathcal{U}_{m,n} = ap(\alpha_1\overline{\alpha_1})ap(\alpha_2\overline{\alpha_2})\cdots ap(\alpha_m\overline{\alpha_m}).$$

When $m = |\mathbf{coN}(n)|$, let $\mathcal{DB}_n = \mathcal{U}_{m,n}$.

Theorem 4. *Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the first m co-necklaces of order n in colex order, where $m \ge 1$. Then $\mathcal{U}_{m,n}$ is a universal cycle for $\bigcup_{k=1}^{m} \mathbf{coneck}(\alpha_k)$ and $\overline{\alpha_m}$ is a suffix of $\mathcal{U}_{m,n}$.*

Proof. The proof is by induction. In the base case when $m = 1$, $\alpha_1 = 0^n$. Clearly $ap(\alpha_1\overline{\alpha_1}) = 0^n1^n$ is a universal cycle that contains all the strings in $\mathbf{coneck}(0^n)$, and $\overline{\alpha_1} = 1^n$ is its suffix. For $m \geq 1$, assume $\mathcal{U}_{m,n}$ is a universal cycle for $\bigcup_{k=1}^{m} \mathbf{coneck}(\alpha_k)$ with suffix $\overline{\alpha_m}$. Consider $\mathcal{U}_{m+1,n} = \mathcal{U}_{m,n}ap(\alpha_{m+1}\overline{\alpha_{m+1}})$, where $\alpha_m = a_1a_2\cdots a_n$ and $\alpha_{m+1} = 0^j1b_{j+2}b_{j+3}\cdots b_n$ where $j+1$ is the smallest index where $b_{j+1} = 1$. Let $\beta = ap(\alpha_{m+1}\overline{\alpha_{m+1}}) = 0^j1b_{j+2}b_{j+3}\cdots b_{|\beta|}$. First we show that $\overline{\alpha_{m+1}}$ is a suffix of $\mathcal{U}_{m+1,n}$. If $\alpha_{m+1}\overline{\alpha_{m+1}}$ is aperiodic, then by definition $\overline{\alpha_{m+1}}$ is a suffix of $\mathcal{U}_{m+1,n}$. If $\alpha_{m+1}\overline{\alpha_{m+1}}$ is periodic, we know that $\overline{\alpha_m}$ appears as a suffix of $\mathcal{U}_{m,n}$ by the inductive hypothesis. Also by Lemma 3 we see that $\overline{a_{j+2}a_{j+3}\cdots a_n} = \overline{b_{j+2}b_{j+3}\cdots b_n}$, and this implies that a suffix of $\overline{\alpha_m}$ is $(b_1b_2\cdots b_{|\beta|})^k$ where $\alpha_{m+1}\overline{\alpha_{m+1}} = (b_1b_2\cdots b_{|\beta|})^{2k+1}$ (a result of Lemma 1). Lemma 2 tells us that $\overline{\alpha_{m+1}}$ is a suffix of $\overline{\alpha_m}\beta = (b_1b_2\cdots b_{|\beta|})^{k+1}$, so $\overline{\alpha_{m+1}}$ is a suffix of $\mathcal{U}_{m+1,n}$. Now we prove that $\mathcal{U}_{m+1,n}$ is a universal cycle for $\bigcup_{k=1}^{m+1} \mathbf{coneck}(\alpha_k)$. By the inductive hypothesis, $\mathcal{U}_{m+1,n}$ will contain all the strings in $\bigcup_{k=1}^{m} \mathbf{coneck}(\alpha_k)$ except for possibly the strings $\{\overline{a_2a_3\cdots a_n}0, \overline{a_3a_4\cdots a_n}00, \ldots, \overline{a_n}0^{n-1}\}$ which were involved in the wraparound. First, we show they still exist as substring in the cyclic $\mathcal{U}_{m+1,n}$ By Lemma 3, $\overline{a_{j+2}a_{j+3}\cdots a_n} = \overline{b_{j+2}b_{j+3}\cdots b_n}$. Because we already showed that $\overline{\alpha_{m+1}}$ is a suffix of $\mathcal{U}_{m+1,n}$, this implies that each string in $\{\overline{a_{j+2}a_{j+3}\cdots a_n}0^{j+1}, \overline{a_{j+3}a_{j+4}\cdots a_n}0^{j+2}, \ldots, \overline{a_n}0^{n-1}\}$ occurs as a substring in the wrap-around of the cyclic $\mathcal{U}_{m+1,n}$. Furthermore, the strings $\{\overline{a_2a_3\cdots a_n}0, \overline{a_3a_4\cdots a_n}00, \ldots, \overline{a_{j+1}\cdots a_n}0^j\}$ exist within $\mathcal{U}_{m+1,n}$ because β has prefix 0^j. Finally, we show that all strings in $\mathbf{coneck}(\alpha_{m+1})$ occur as a substring in $\mathcal{U}_{m+1,n}$. Those that are not trivially substrings of β occur either in the wrap-around or have their prefix as a suffix in $\mathcal{U}_{m,n}$ and suffix in a prefix of β. The latter case covers each string in $\{\overline{b_{j+2}b_{j+3}\cdots b_n}0^j1, \overline{b_{j+3}b_{j+4}\cdots b_n}0^j1b_{j+2}, \ldots, \overline{b_{x+1}\cdots b_n}0^j1b_{j+2}\cdots b_x\}$, where $x = n$ if $|\beta| > n$ and $x = |\beta|$ otherwise. Since the length n suffix of $\mathcal{U}_{m+1,n}$ is $\overline{\alpha_{m+1}} = \overline{b_1b_2\cdots b_n}$, and $\alpha_1 = 0^n$, the strings $\{1^{j-1}0\overline{b_{j+2}\cdots b_n}0, 1^{j-2}0\overline{b_{j+2}\cdots b_n}00\ldots, 0\overline{b_{j+2}\cdots b_n}0^j\}$ occur in the wraparound of $\mathcal{U}_{m+1,n}$. Thus, every string in $\bigcup_{k=1}^{m+1} \mathbf{coneck}(\alpha_k)$ appears as a substring in the cyclic string $\mathcal{U}_{m+1,n}$. Therefore since the length of $\mathcal{U}_{m+1,n}$ is equal to $|\bigcup_{k=1}^{m+1} \mathbf{coneck}(\alpha_k)|$, $\mathcal{U}_{m+1,n}$ is a universal cycle for $\bigcup_{k=1}^{m+1} \mathbf{coneck}(\alpha_k)$. □

Corollary 5. *$\mathcal{DB}_n$ is a de Bruijn sequence of order n.*

3.1 Efficient implementation

In order to construct $\mathcal{DB}_n$, we must first generate co-necklaces in colex order. A naïve algorithm will consider all strings $\alpha \in \mathbf{B}(n)$ in colex order and test if $\alpha\overline{\alpha}$ is a necklace. Such a necklace test can be computed in $O(n)$-time [3]. Since there are $\Theta(2^n/n)$ co-necklaces of length n by Equation 1 this approach will result in each co-necklace being generated in $O(n^2)$-amortized time. We will present an algorithm that improves this method by a factor of n.

Our strategy is to apply a standard recursive algorithm to generate strings in colex order, building the global string $\alpha = a_1a_2\cdots a_n$ from right to left one bit at a time. Such an algorithm is given in Algorithm 1 with the following modifications optimized for co-necklace generation.

– Keep track of the length of the current run of 0s in the parameter $curZero$.
– Keep track of the longest substring of the form 0^* in the parameter $maxZero$.
– Terminate the recursion when the length of the remaining prefix of α to be completed, given by parameter t, is less than or equal to $maxZero$. This is because a longest run of 0s must be at the start of any co-necklace.[3] The algorithm can be further optimized by keeping track of the current run of the form 1^*. At this point, the prefix $a_1a_2\cdots a_t$ is set to 0^t and then we test if $\alpha\overline{\alpha}$ is a co-necklace using the boolean function IsNECKLACE.

The function PRINT outputs the string passed as input and the function MAX returns the larger of its two integer inputs. After initializing $a_n = 0$, since all co-necklaces end with 0, the initial call to generate all co-necklaces of length n is GENERATECONECK$(n-1, 1, 1)$.

[3] Because a longest run of the form 0^* or 1^* must be at the start of a co-necklace, the algorithm can be further optimized by keeping track of the longest current run of the form 1^*. However, it will not affect the asymptotic analysis.

Algorithm 1 An algorithm to generate all co-necklaces of length n in colex order.

```
1: procedure GENERATECONECK($t, curZero, maxZero$)
2:     if $t \leq maxZero$ then
3:         $a_1a_2\cdots a_t \leftarrow 0^t$
4:         if ISNECKLACE($\alpha\overline{\alpha}$) then PRINT($\alpha\overline{\alpha}$)
5:     else
6:         $a_t \leftarrow 0$
7:         GENERATECONECK($t-1, curZero+1$, MAX($curZero+1, maxZero$))
8:         $a_t \leftarrow 1$
9:         GENERATECONECK($t-1, 0, maxZero$)
```

Recall that IsNECKLACE can be implemented in $O(n)$-time and also note that setting the prefix $a_1a_2\cdots a_t$ also requires at most $O(n)$-time. Thus, since the recursion always has a branch factor of two, the total work done by the algorithm will be $O(n)$ times the number of strings α generated before the necklace test. An example computation tree for $n = 6$ is shown in Figure 1. Each such string $\alpha = a_1a_2\cdots a_n$ is constructed to have a longest run of 0s at the start of the string and $a_n = 0$. Now observe that $a_na_1a_2\cdots a_{n-1}$ is the prefix of a necklace (called a prenecklace). In particular, $a_na_1a_2\cdots a_{n-1}1$ is clearly a necklace since the longest number of 0s occurs uniquely at the start of the string. The number of prenecklaces of length n is known to be $\Theta(2^n/n)$ [4], which in turn is proportional to the number of co-necklaces of length n as mentioned earlier. Thus, the total work done by the algorithm is bounded by $O(n)$ times the number of co-necklaces of length n.

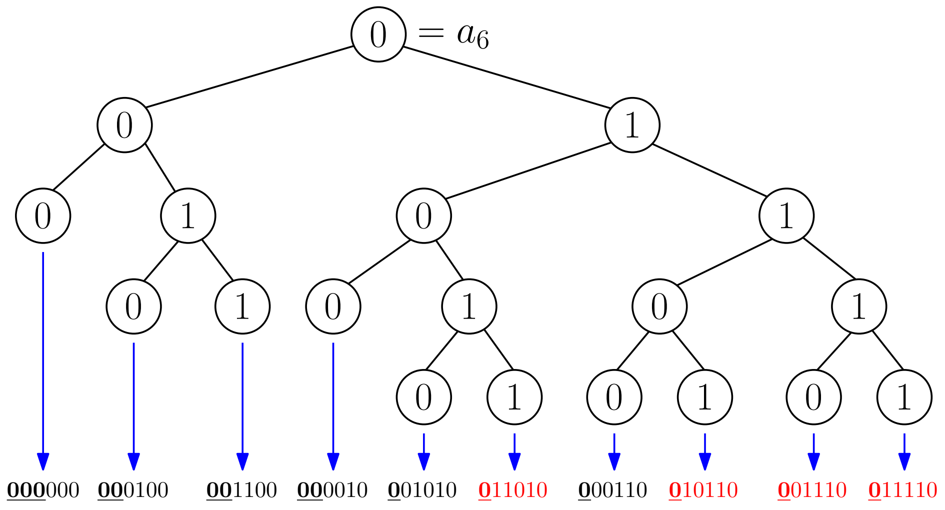

Fig. 1. Computation tree for GENERATECONECK for $n = 6$. Observe that the six co-necklaces in **coN**(6) are listed in colex order: 000000, 000100, 001100, 000010, 001010, 000110.

Theorem 6. *The set of all co-necklaces of length n can be listed in colex order in $O(n)$-amortized time per string.*

To apply the algorithm GENERATECONECK to construct $\mathcal{DB}_n$, we only need to determine $p = |ap(\alpha\overline{\alpha})|$ for each co-necklace α, then pass $a_1a_2\cdots a_p$ to the function PRINT instead of $\alpha\overline{\alpha}$. A complete C implementation to generate $\mathcal{DB}_n$ is provided in the appendix. Since the value p can be computed in $O(n)$ time, we obtain the following corollary.

Corollary 7. *The de Bruijn sequence $\mathcal{DB}_n$ can be constructed in $O(1)$-amortized time per bit.*

It remains an open problem to generate co-necklaces in colex (or lexicographic) order in $O(1)$-amortized time per string.

4 Discrepancy

In this section we focus on a measure studied by Cooper and Heitsch [5] known as the discrepancy of a de Bruijn sequence. In particular, they show that the discrepancy of the lexicographically smallest de Bruijn sequence, which happens to be obtained via the necklace concatenation approach discussed earlier, is $\Theta(\frac{2^n \log n}{n})$.

Let α be a binary string. Let $\mathit{diff}(\alpha)$ denote the absolute difference between the number of 1s and number of 0s in α. Thus $\mathit{diff}(011011) = 2$ because there are four 1s and two 0s, and the absolute difference is 2. The *discrepancy* of α, denoted $D(\alpha)$, is defined to be the maximum value of $\mathit{diff}(\beta)$ over all substrings β of α. For example, the discrepancy of 101001110110 is 4 because $\mathit{diff}(111011) = 4$ and 111011 is a substring that results in the maximal difference.

Theorem 8. *For all $n \geq 1$, $n \leq D(\mathcal{DB}_n) < 2n$.*

Proof. Clearly n is the lower bound for the discrepancy of any de Bruijn sequence, since they all must contain the substring 0^n. All substrings contained within $ap(\alpha\overline{\alpha})$ for a co-necklace α clearly have difference less than n, unless $\alpha = 0^n$ in which case the difference is n. Any other substring will be of the form

$$\sigma \; ap(\alpha_i\overline{\alpha_i}) \; ap(\alpha_{i+1}\overline{\alpha_{i+1}}) \; \cdots \; ap(\alpha_j\overline{\alpha_j}) \; \tau$$

for some $i \leq j$ where σ is a suffix of $ap(\alpha_{i-1}\overline{\alpha_{i-1}})$ and τ is a prefix of $ap(\alpha_{j+1}\overline{\alpha_{j+1}})$. Thus since $\mathit{diff}(ap(\alpha\overline{\alpha})) = 0$, the difference of the substring must be less than $2n$. □

In fact, $D(\mathcal{DB}_n)$ approaches $2n$ as n gets large. Consider the substring of $\mathcal{DB}_n$ starting from $\overline{\alpha_1} = 1^n$ and ending with $\alpha_j = 0^i(1^{i-1}0)^r$ where $i(r+1) = n$. Note that α_j is a co-necklace and $\alpha_j\overline{\alpha_j}$ is aperiodic, so such a substring exists. The difference of the string between this prefix and suffix is 0, as outlined in the proof of Theorem 8. Thus, the difference of the entire substring is given by $n + \mathit{diff}(\alpha_j)$ since α_j has more 1s than 0s. Since $\mathit{diff}(\alpha_j) = r(i-1) - r - i$, by solving for $r = n/i - 1$ we get $\mathit{diff}(\alpha_j) = \frac{i-2}{i}(n) - 2i + 2$. Thus considering i to be any constant, as n goes to infinity $D(\mathcal{DB}_n)$ approaches $n + \frac{i-2}{i}n$.

References

1. The On-Line Encyclopedia of Integer Sequences, published electronically at https://oeis.org, 2010, sequence A000016.
2. A. Alhakim. A simple combinatorial algorithm for de Bruijn sequences. *The American Mathematical Monthly*, 117(8):728–732, 2010.
3. K. S. Booth. Lexicographically least circular substrings. *Inform. Process. Lett.*, 10(4/5):240–242, 1980.
4. K. Cattell, F. Ruskey, J. Sawada, M. Serra, and C. Miers. Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF(2). *J. Algorithms*, 37(2):267–282, 2000.
5. J. Cooper and C. Heitsch. The discrepancy of the lex-least de Bruijn sequence. *Discrete Mathematics*, 310:1152–1159, 2010.
6. P. B. Dragon, O. I. Hernandez, and A. Williams. The grandmama de Bruijn sequence for binary strings. In *Proceedings of LATIN 2016: Theoretical Informatics: 12th Latin American Symposium, Ensenada, Mexico*, pages 347–361. Springer Berlin Heidelberg, 2016.
7. C. Eldert, H. Gray, H. Gurk, and M. Rubinoff. Shifting counters. *AIEE Trans.*, 77:70–74, 1958.
8. T. Etzion. Self-dual sequences. *Journal of Combinatorial Theory, Series A*, 44(2):288 – 298, 1987.
9. T. Etzion and A. Lempel. Construction of de Bruijn sequences of minimal complexity. *IEEE Transactions on Information Theory*, 30(5):705–709, September 1984.
10. L. Ford. A cyclic arrangement of M-tuples. *Report No. P-1071, Rand Corporation, Santa Monica, California, April 23*, 1957.
11. H. Fredricksen. Generation of the Ford sequence of length 2^n, n large. *J. Combin. Theory Ser. A*, 12(1):153–154, 1972.
12. H. Fredricksen and I. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Math.*, 61(2):181 – 188, 1986.
13. H. Fredricksen and J. Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Math.*, 23:207–210, 1978.
14. S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.
15. E. R. Hauge. On the cycles and adjacencies in the complementary circulating register. *Discrete Mathematics*, 145(1):105 – 132, 1995.
16. Y. Huang. A new algorithm for the generation of binary de Bruijn sequences. *J. Algorithms*, 11(1):44–51, 1990.

17. M. H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40(12):859–864, 1934.
18. G. L. Mayhew and S. W. Golomb. Characterizations of generators for modified de Bruijn sequences. *Advances in Applied Mathematics*, 13(4):454 – 461, 1992.
19. F. Ruskey, C. Savage, and T. M. Y. Wang. Generating necklaces. *J. Algorithms*, 13:414–430, 1992.
20. J. Sawada, A. Williams, and D. Wong. A surprisingly simple de Bruijn sequence construction. *Discrete Math.*, 339:127–131, 2016.
21. J. Sawada, A. Williams, and D. Wong. Necklaces and Lyndon words in colexicographic and reflected Gray code order. *submitted manuscript*, 2017.

Appendix - C code

```
#include <stdio.h>
#define MAX(a,b) (a)>(b)?(a):(b)
int N,b[1000];

//--------------------------------------------------------------
// Returns 0 if string is not necklace, and index of
// longest aperiodic prefix if is necklace
//--------------------------------------------------------------
int IsNecklace(int b[], int n) {
    int i, p=1;

    for (i=2; i<=n; i++) {
        if (b[i-p] > b[i]) return 0;
        if (b[i-p] < b[i]) p = i;
    }
    if (n % p != 0) return 0;
    return p;
}

void Gen(int t, int curZero, int maxZero){
    int i,p;
    if(t <= maxZero){
        for (i=N+1;i<=2*N;++i) b[i]=1-b[i-N];
        p = IsNecklace(b,2*N);
        for (i=1; i<=p; i++) printf("%d", b[i]);
    }
    else {
        b[t]=0;
        Gen(t-1,curZero+1,MAX(curZero+1,maxZero));
        b[t]=1;
        Gen(t-1,0,maxZero);
    }
}

int main(){
    printf("Enter N: ");scanf("%d",&N);
    b[N] = 0;
    Gen(N-1,1,1);
    printf("\n");
    return 0;
}
```