A Survey of De Bruijn Sequences

Kishen N Gowda (17110074)
IIT Gandhinagar
kishen.gowda@iitgn.ac.in
Link to presentation: `http://bit.ly/dm-debruijn`

Abstract

A de Bruijn sequence is a cyclic sequence of order n on a k sized alphabet A in which every possible string of size n built on A is present exactly once. This is denoted by B(k,n). De Bruijn Sequences have many applications in the field of Bio-informatics. In this paper, I will survey de Bruijn Sequences with an emphasis on how they are constructed. I will be discussing two methods - one is using de Bruijn Graphs and Euler cycles, and the other is using Inverse Burrows-Wheeler Transform. I have also highlighted some elementary proofs, applications and extensions.

1 Introduction

In Combinatorics, a de Bruijn sequence of order n on a size k alphabet A is a cyclic sequence in which every possible length n string on A occurs exactly once as a sub-string (i.e., as a contiguous subsequence). This sequence is generally denoted by $B(k, n)$.

Let us consider the following example.

Example

let $A = \{0, 1\}$

Now, consider the string 0011. This string covers all possible length 2 strings on A.

0011 0011 0011 0011

So, the given sequence 0011 is a de Bruijn Sequence of order 2 on a size 2 Alphabet A.

We can denote this as $B(2, 2)$.

The sequences are named after the Dutch mathematician *Nicolaas Govert de Bruijn*. According to him, the existence of de Bruijn sequences for each order together with the above properties were first proved, for the case of alphabets with two elements, i.e. $B(2, n)$, by *Camille Flye Sainte-Marie* in 1894, whereas the generalization to larger alphabets is originally due to *Tatyana van Aardenne-Ehrenfest* and himself.

2 History

The earliest known example of a de Bruijn sequence comes from Sanskrit prosody where, since the work of *Pingala*, each possible three-syllable pattern of long and short syllables is given a name, such as '*y*' for short–long–long and '*m*' for long–long–long. To remember these names, the mnemonic ***yamātārājabhānasalagām*** is used, in which each three-syllable pattern occurs starting at its name: '*yamātā*' has a short–long–long pattern, '*mātārā*' has a long–long–long pattern, and so on, until '*salagām*' which has a short–short–long pattern. This mnemonic, equivalent to a de Bruijn sequence on binary 3-tuples, is of unknown antiquity, but is at least as old as Charles Philip Brown's 1869 book on Sanskrit prosody that mentions it and considers it "an ancient line, written by *Panini*.

The first appearance of this problem was in the French problem journal *L'Intermédiaire des Mathématiciens* given by *A. de Rivière* which questioned about the existence of a circular

arrangement of zeroes and ones of size 2^n that contains all 2^n binary sequences of length n. The problem was solved (in the affirmative), along with the count of $2^{2^{n-1}-n}$ distinct solutions, by *Camille Flye Sainte-Marie* in the same year. This was largely forgotten. *Martin* (1934) proved the existence of such cycles for general alphabet size in place of 2, with an algorithm for constructing them.

In 1944, *Kees Posthumus* conjectured the count $2^{2^{n-1}-n}$ for binary sequences, de Bruijn proved the conjecture in 1946, through which the problem became famous.

3 Elementary Proofs

Let us see some elementary proofs of de Bruijn Sequences.

1) Length of a de Bruijn Sequence

Theorem

▶ **Theorem 1.** *The length of $B(n,k)$ is k^n.*

Proof. The total number of strings of size n possible on an alphabet of size k is k^n. Now, if we start constructing the de Bruijn sequence, each of the k^n string has to be present in the sequence and start at some point in the sequence. Also, each sequence should appear exactly once. Therefore, the length of the de Bruijn sequence is k^n. ◀

2) Existence of de Bruijn Sequences

Theorem

▶ **Theorem 2.** *There are de Bruijn sequences for any positive integers n and k.*

Proof. Refer [1]. ◀

3) Number of distinct de Bruijn Sequences

Theorem

▶ **Theorem 3.** *The number of distinct de Bruijn Sequences B(k,n) is* $\frac{(k!)^{k^{n-1}}}{k^n}$

▶ Note. The proof of theorem 3 is beyond the scope of this paper. Refer [6].

4 Constructing De Bruijn Sequences

There are multiple ways to construct de Bruijn sequences. We will see two methods, one using graphs and the other using BWT. For pseudo codes refer **Appendix**.

Method 1: Using de Bruijn Graphs

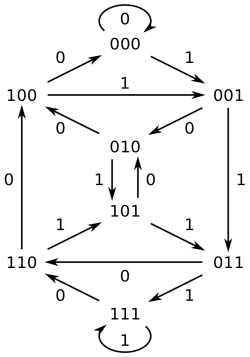
Definition

De Bruijn Graph: *In graph theory, an n dimensional de Bruijn graph of k symbols is a directed graph representing overlaps between sequences of symbols. It has k^{n-1} vertices, consisting of all possible length n sequences of the given symbols; the same symbol may appear multiple times in a sequence.*

Let's see an example.

Example

The following is the graph on the binary alphabet $\{0, 1\}$ and covers all sequences of size 4 possible on the binary alphabet, as edges. The sequences can be read as the vertex followed by the value on the edge.



As we can notice, for constructing the de Bruijn sequence we require a path which starts at a vertex of the graph and visits every edge exactly once and finally ends at the same point. This is nothing but a Eulerian circuit. Therefore, we have designed a pretty natural algorithm for generating de Bruijn sequences. Let us state the method.

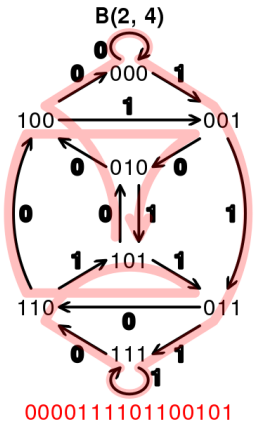
Method:

1. For given k size alphabet A and n, construct a de Bruijn graph.
2. Select any vertex, find a Eulerian circuit. Let $v_1.e_1e_2e_3.....e_{k^{n-1}}.v_1$ be the Eulerian circuit, where v_1 is the start vertex.

3. Now, concatenate the values on the edges in the sequence of edges above. The resulting sequence is the de Bruijn sequence $B(k, n)$.

Example

The following is one of the possible Eulerian circuits of the de Bruijn Graph constructed on the binary alphabet and sequence length 4.
By writing the sequence of edge values from the Eulerian circuit, we get the de Bruijn Sequence $B(2, 4)$.



► Note. The proof of existence of Eulerian circuit for the de Bruijn Graph is proved while proving the existence of de Bruijn sequences. See [1] for more details.

Method 2: By Inverse Burrows-Wheeler Transform

Before we go ahead with the method, let us see some definitions.

1) Lyndon Words

Definition

Lyndon Word: *A Lyndon word is a non-empty string that is strictly smaller in lexicographic order than all of its rotations.*

For example, the Lyndon words over the binary alphabet 0,1, sorted by length and then lexicographically within each length class, form an infinite sequence (A001037 in OEIS) that begins as,

$$0, 1, 01, 001, 011, 0001, 0011, 0111, 00001, 00011, 00101, 00111, 01011, 01111, ...$$

The first string that does not belong to this sequence, "00", is omitted because it is periodic (it

consists of two repetitions of the substring "0"); the second omitted string, "10", is aperiodic but is not minimal in its permutation class as it can be cyclically permuted to the smaller string "01".

2) Burrows-Wheeler Transform (BWT)

Definition

Burrows-Wheeler Transform: The Burrows–Wheeler transform (BWT)*, also called block-sorting compression, is an algorithm used to prepare data for use with data compression techniques. The BWT rearranges a character string into runs of similar characters.*

For example, consider the string "ˆBANANA|". By BWT, it gets transformed to "BNNˆAA|A".

Transformation				
Input	All rotations	Sort in lexical order	Take the last column	Output
ˆBANANA|	ˆBANANA| |ˆBANANA A|ˆBANAN NA|ˆBANA ANA|ˆBAN NANA|ˆBA ANANA|ˆB BANANA|ˆ	ANANA|ˆB ANA|ˆBAN A|ˆBANAN BANANA|ˆ NANA|ˆBA NA|ˆBANA ˆBANANA| |ˆBANANA	ANANA|ˆ**B** ANA|ˆBA**N** A|ˆBANA**N** BANANA|ˆ NANA|ˆB**A** NA|ˆBAN**A** ˆBANANA**|** |ˆBANAN**A**	BNNˆAA|A

3) Bijective Variant of BWT

Definition

Bijective Variant of Burrows-Wheeler Transform: *The bijective transform is computed by factoring the input into a non-increasing sequence of Lyndon words; such a factorization exists and is unique by the Chen–Fox–Lyndon theorem [3], and may be found in linear time*

For example, the string ˆBANANA is broken into Lyndon words as (ˆ) (B) (AN) (AN) (A).

Bijective Transformation				
Input	All rotations	Sort alphabetically	Last column of rotated Lyndon word	Output
^BANANA|	^^^^.... (^) BBBB.... (B) ANAN... (AN) NANA... (NA) ANAN... (AN) NANA... (NA) AAAA.... (A)	AAAA.... (A) ANAN... (AN) ANAN... (AN) BBBB.... (B) NANA... (NA) NANA... (NA) ^^^^.... (^)	AAAA.... (**A**) ANAN... (A**N**) ANAN... (A**N**) BBBB.... (**B**) NANA... (N**A**) NANA... (N**A**) ^^^^.... (^)	ANNBAA^|

Based on the above definitions, we can claim that,

Claim

▷ Claim. An inverse Burrows—Wheeler transform on a word w generates a multi-set of equivalence classes consisting of strings and their rotations. These equivalence classes of strings each contain a Lyndon word as a unique minimum element, so the inverse Burrows—Wheeler transform can be considered to generate a set of Lyndon words.

Proof. Refer [7] ◀

4) Connection to de Bruijn sequences

Theorem

▶ **Theorem 4.** *If one concatenates together, in lexicographic order, all the Lyndon words that have length dividing a given number n, the result is the de Bruijn sequence B(k,n).*

Proof. Refer [5]. ◀

Therefore, from the above definitions and Theorem 4, we get another method for the construction of de Bruijn Sequences. Let's state it.

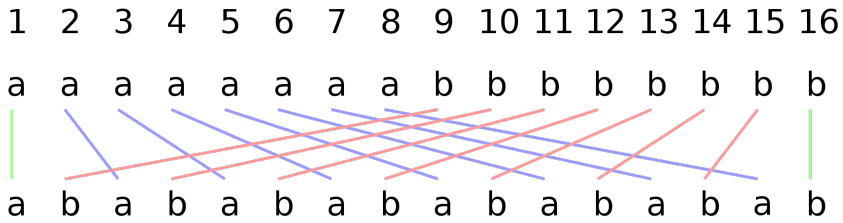
Method:

1. For given k size alphabet A and n, construct the string w by repeating A k^{n-1} times.
2. Sort the characters in w to yield the string w'.

3. Position the string w' above the string w, and map each letter's position in w' to its position in w while preserving order. This process defines the standard permutation.
4. Write this permutation in cycle notation with the smallest position in each cycle first, and the cycles sorted in increasing order.
5. For each cycle, replace each number with the corresponding letter from string w' in that position.
6. Each cycle has now become a Lyndon word, and they are arranged in lexicographic order, so dropping the parentheses yields the first de Bruijn sequence.

Example

For example, to construct the smallest B(2,4) de Bruijn sequence of length $2^4 = 16$, repeat the alphabet (ab) 8 times yielding $w = abababababababab$. Sort the characters in w, yielding $w' = aaaaaaaabbbbbbbb$. Position w' above w as shown, and map each element in w' to the corresponding element in w by drawing a line. Number the columns as shown so we can read the cycles of the permutation:



Starting from the left, the cycles are:
(1) (2 3 5 9) (4 7 13 10) (6 11) (8 15 14 12) (16).
Then, replacing each number by the corresponding letter in w' from that column yields: (a)(aaab)(aabb)(ab)(abbb)(b).
These are all of the Lyndon words whose length divides 4, in lexicographic order, so dropping the parentheses gives $B(2,4) = aaaabaabbababbbb$.

5 Applications

1. The sequence can be used to shorten a brute-force attack on a PIN-like code lock that does not have an "enter" key and accepts the last n digits entered. For example, a digital

door lock with a 4-digit code would have $B(10, 4)$ solutions, with length 10000. Therefore, only at most $10000 + 3 = 10003$ (as the solutions are cyclic) presses are needed to open the lock. Trying all codes separately would require $4 \times 10000 = 40000$ presses.

2. The symbols of a de Bruijn sequence written around a circular object (such as a wheel of a robot) can be used to identify its angle by examining the n consecutive symbols facing a fixed point. This angle-encoding problem is known as the "rotating drum problem".
3. De Bruijn cycles are of general use in neuroscience and psychology experiments that examine the effect of stimulus order upon neural systems, and can be specially crafted for use with functional magnetic resonance imaging (fMRI).
4. Creating cool games and tricks.
5. Wide applications in Bioinformatics.

6 Extensions

Let us see some extensions of de Bruijn Sequences.

1) f-fold de Bruijn Sequences

Definition

f-fold de Bruijn Sequences: *f-fold n-ary de Bruijn sequence is an extension of the notion n-ary de Bruijn sequence, such that the sequence of the length fk^n contains every possible subsequence of the length n exactly f times.*

For example, for $n = 2$ the cyclic sequences 11100010 and 11101000 are two-fold binary de Bruijn sequences.

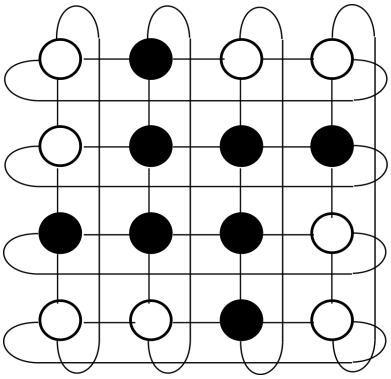
The number of two-fold de Bruijn sequences is denoted by N_n. Some of the known values of N_n are $N_1 = 2$, $N_2 = 5$, $N_3 = 72$, and $N_4 = 43768$.

2) De Bruijn Torus

Definition

De Bruijn Torus: *A de Bruijn torus is a toroidal array with the property that every k-ary m-by-n matrix occurs exactly once.*

Such a pattern can be used for two-dimensional positional encoding in a fashion analogous to that described above for rotary encoding. Position can be determined by examining the m-by-n matrix directly adjacent to the sensor, and calculating its position on the de Bruijn torus. The smallest possible binary "square" de Bruijn torus, depicted right, denoted as $(4, 4; 2, 2)_2$ de Bruijn torus (or simply as B_2), contains all 2×2 binary matrices.



7 Conclusion

In this paper, we have seen two methods for constructing de Bruijn Sequences $B(k, n)$, one was very intuitive but inefficient whereas the other was non-intuitive but efficient. We have also highlighted some of the applications of de Bruijn sequences, which makes it's construction important.

References

[1] Jacob Bower. *18.312: Algebraic Combinatorics - Lecture 21*. 2011. DOI: http://pi.math.cornell.edu/~levine/18.312/alg-comb-lecture-21.pdf.

[2] Nicolaas Govert de Bruijn. *A Combinatorial Problem*. 1946. DOI: http://www.dwc.knaw.nl/DL/publications/PU00018235.pdf.

[3] M. Hazewinkel. "Chen-Fox-Lyndon Factorization for Words over Partially Ordered Sets". In: *Journal of Mathematical Sciences* 131.6 (Dec. 2005), pp. 6027–6031. ISSN: 1573-8795. DOI: 10.1007/s10958-005-0458-7. URL: https://doi.org/10.1007/s10958-005-0458-7.

[4] M. H. Martin. "A problem in Arrangements". In: (1934). DOI: http://www.ams.org/journals/bull/1934-40-12/S0002-9904-1934-05988-3/S0002-9904-1934-05988-3.pdf.

[5] Eduardo Moreno. "On the theorem of Fredricksen and Maiorana about de Bruijn sequences". In: (2003). DOI: https://core.ac.uk/download/pdf/82741740.pdf.

[6] Vladimir Raphael Rosenfeld. “Enumerating De Bruijn Sequences”. In: (). DOI: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.9804&rep=rep1&type=pdf.

[7] Marinella Sciortino. *Suffixes, Conjugates and Lyndon words*. 2013. DOI: http://pages.di.unipi.it/rosone/talks/DLT_2013.pdf.

Appendix

Below is a recursive implementation of *Inverse BWT*, which returns $B(k, n)$ for given alphabet A of size k and n, in python.

```
        def de_bruijn(k, n):
            """
            de Bruijn sequence for alphabet of size k
            and subsequences of length n.
            """
            try:
                '''
                let's see if k can be cast to an integer
                if so, make our alphabet a list
                '''
                intk = int(k)
                alphabet = list(map(str, range(k)))

            except (ValueError, TypeError):
                alphabet = k
                k = len(k)

            a = [0] * k * n
            sequence = []

            def db(t, p):
                if t > n:
                    if n % p == 0:
                        sequence.extend(a[1:p + 1])
                else:
                    a[t] = a[t - p]
                    db(t + 1, p)
                    for j in range(a[t - p] + 1, k):
                        a[t] = j
                        db(t + 1, t)
            db(1, 1)
            return "".join(alphabet[i] for i in sequence)
```