THE AMERICAN MATHEMATICAL MONTHLY

Volume 117, Number 8 October 2010

Gerald B. Folland	A Tale of Topology	663
Jack S. Calcut	Grade School Triangles	673
Mark A. Conger, Jason Howald	A Better Way to Deal the Cards	686
Tom M. Apostol, Mamikon A. Mnatsakanian	Tanvolutes: Generalized Involutes	701
Leonard F. Klosinski, Gerald L. Alexanderson, Loren C. Larson	The Seventieth William Lowell Putnam Mathematical Competition	714
NOTES		
Alin Bostan, Philippe Dumas	Wronskians and Linear Independence	722
Abbas M. Alhakim	A Simple Combinatorial Algorithm for de Bruijn Sequences	728
Albert Nijenhuis	Short Gamma Products with Simple Values	733
Miyeon Kwon	Dimension, Linear Functionals, and Norms in a Vector Space	738
PROBLEMS AND SOLUTIONS		741
REVIEWS		
Marion Deutsche Cohen	*Graph Theory: A Problem Oriented Approach.* By Daniel A. Marcus	749
EDITOR'S ENDNOTES		754

AN OFFICIAL PUBLICATION OF THE MATHEMATICAL ASSOCIATION OF AMERICA

The American Mathematical Monthly

ISSN: 0002-9890 (Print) 1930-0972 (Online) Journal homepage: https://www.tandfonline.com/loi/uamm20

A Simple Combinatorial Algorithm for de Bruijn Sequences

Abbas M. Alhakim

To cite this article: Abbas M. Alhakim (2010) A Simple Combinatorial Algorithm for de Bruijn Sequences, The American Mathematical Monthly, 117:8, 728-732

To link to this article: https://doi.org/10.4169/000298910X515794

Published online: 13 Dec 2017.

A Simple Combinatorial Algorithm for de Bruijn Sequences

Abbas M. Alhakim

Abstract. This paper presents a combinatorial method to construct a de Bruijn sequence for any order n. The method is similar to that of the well-known prefer-one algorithm. The resulting sequences are compared and both methods are shown to be special cases of a unifying graph construction that is capable of generating all de Bruijn sequences.

1. INTRODUCTION. A binary de Bruijn sequence of order n is a string of bits in which every possible string of n bits occurs exactly once as a consecutive substring. As an example the sequence 0001011100 is a de Bruijn sequence of order 3. It is known that the last $n-1$ bits always coincide with the first $n-1$ bits, so it is customary to trim the last $n-1$ bits and think of the remaining bits as being placed on a circle; this accounts for the practice of referring to these sequences as de Bruijn cycles. Although the existence of such sequences is not obvious, it is well known that they exist for all orders n and that the number of distinct sequences is $2^{2^{n-1}-n}$; see de Bruijn [**3**]. Observe that this number is 1, 1, 2, 16, and 2048 for $n = 1, \dots, 5$ respectively.

A de Bruijn digraph $G_n = (V_n, E_n)$ of order n has the same vertex set as a hypercube, i.e., the 2^n binary strings, but the edges are different. For two vertices $\mathbf{x} = x_1 \cdots x_n$ and $\mathbf{y} = y_1 \cdots y_n$, $(\mathbf{x}, \mathbf{y})$ is an edge if and only if $y_i = x_{i+1}$ for $i = 1, \dots, n-1$. The de Bruijn digraphs of orders 2 and 3 are shown in Figure 1. One aspect of de Bruijn sequences makes them similar to Gray codes. A Gray code makes a Hamiltonian cycle in a hypercube while a de Bruijn sequence is equivalent to a Hamiltonian

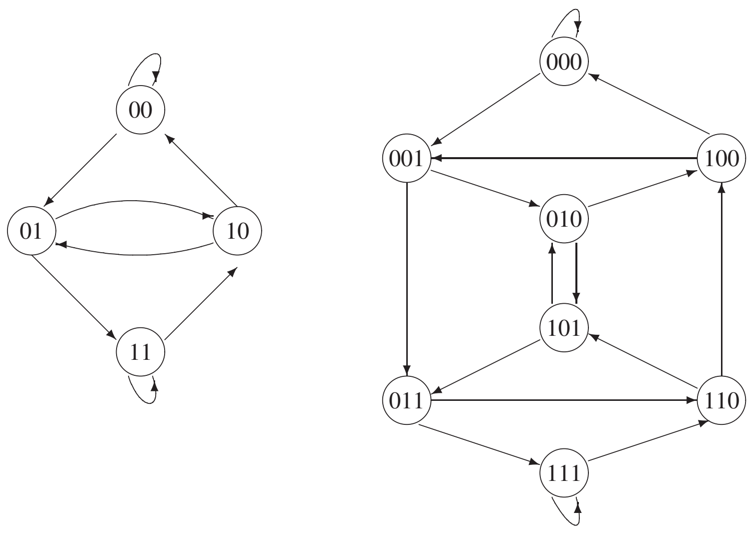

Figure 1. De Bruijn digraphs of orders 2 and 3 respectively.

doi:10.4169/000298910X515794

cycle in the de Bruijn digraph. That is, if $x_1 \cdots x_m$ is a de Bruijn sequence of order n then $x_1 \cdots x_n, x_2 \cdots x_{n+1}, \ldots, x_{m-n+1} \cdots x_m, x_1 \cdots x_n$ is a Hamiltonian cycle in G_n. A de Bruijn digraph is obviously Eulerian because every vertex has exactly two inward edges and two outward edges. Interestingly, Eulerian circuits and Hamiltonian cycles are intimately related in the case of de Bruijn digraphs: if we label an edge $(x_1 \cdots x_{n-1}, x_2 \cdots x_n)$ in G_{n-1} as $x_1 \cdots x_n$ then we can see that the consecutive edges of an Eulerian circuit in G_{n-1} form a Hamiltonian cycle in G_n.

These combinatorial objects have been used to design memory wheels [**7**], where a rotating wheel is labeled at equal distances by a sequence of 0s and 1s. There are n sensors affixed next to the rotating wheel in such a way that they can read n consecutive bits of the sequence. The memory wheel problem (also known as the rotating drum problem) is to come up with a sequence so that the sensors uniquely determine the location of the wheel. Another rather entertaining application is to break a hypothetical key-lock system with a minimal number of attempts [**4**]. Besides these recreational problems, de Bruijn sequences have been used in diverse fields such as the generation of pseudorandom sequences. In fact, maximal period linear feedback shift register sequences (LFSR) are de Bruijn sequences without the all-zero pattern, referred to as punctured de Bruijn sequences, and they have been popular in engineering applications because they can be implemented efficiently by hardware devices called linear feedback shift registers (LFSR); see Golomb [**5**].

The first conference on de Bruijn cycles was held in 2004 with a theme of advancing interest and knowledge in the area of de Bruijn cycles and their many generalizations (they belong to a larger family of combinatorial objects called universal cycles; see [**1, 2, 6, 8**]).

2. GENERATING DE BRUIJN SEQUENCES.

There are various methods for generating de Bruijn cycles, but the most popular ones use linear recurrences that yield sequences with almost full period (maximal period LFSR); this is based on primitive polynomials in the Galois field $GF(2)$. There are also graphical and combinatorial methods. Fredricksen [**4**] is an excellent exposition that surveys the known methods of generation. In his survey, Fredricksen [**4**, p. 207] writes "When the mathematician on the street is presented with the problem of generating a full cycle, one of three things happens: he gives up, or produces a sequence based on a primitive polynomial, or produces the [prefer-one sequence]. Only rarely is a new algorithm proposed." The prefer-one algorithm is a very simple method amazingly capable of generating a full cycle. For any positive integer $n \geq 1$, the algorithm puts n 0s, and proceeds after this by proposing 1 for the next bit and accepting it when the word formed by the last n bits has not been encountered previously in the sequence; otherwise 0 is placed. The algorithm stops when neither 0 nor 1 brings a new word.

Fredricksen cites five authors who discovered the prefer-one algorithm. It is perhaps worth mentioning that the author of this note rediscovered the same algorithm as a solution to a Hamiltonian problem relating to the graph of a certain simple Markov chain.

A similar combinatorial algorithm, also given in Fredricksen [**4**, p. 212], is named the prefer-same algorithm but is more elaborate than the prefer-one. In this note we present and prove yet another algorithm of the same family, which we refer to as the prefer-opposite algorithm. To the best of the author's knowledge, it has not been published anywhere, despite the fact that it enjoys better properties than its more famous cousins, as will be explained later in this note. For a bit b and a positive integer i we will use the terminology $\bar{b} = 1 - b$ and $b^i = \underbrace{b \cdots b}_{i}$.

Prefer-Opposite Algorithm.

1. Let $a_1 = \cdots = a_n = 0$.
2. $i = n + 1$.
3. If $a_{i-n+1} \cdots a_{i-1}\bar{a}_{i-1}$ is a pattern that has not appeared before then let $a_i = \bar{a}_{i-1}$, increment i by 1 and repeat Step 3.
4. Otherwise, if $a_{i-n+1} \cdots a_{i-1}a_{i-1}$ is a pattern that has not appeared earlier in the sequence then let $a_i = a_{i-1}$, increment i by 1 and go to Step 3.
5. Otherwise, stop.

Lemma 2.1. *The algorithm does not produce the pattern* 1^n *for any* $n \geq 2$.

Proof. Assume that 1^n occurs in the sequence. Observing that G_n has a self-loop that occurs at the vertex 1^n (and another at 0^n), the only possible predecessor of 1^n in the sequence is 01^{n-1}—otherwise 1^n would occur twice in a row. However, because we are using the prefer-opposite algorithm, if the string 01^{n-1} occurs it must be immediately followed by a 0, so that the next string is $1^{n-1}0$. ■

The next result shows that 1^n is the only string that is not produced by the algorithm.

Theorem 2.2. *For any integer* $n \geq 2$, *the prefer-opposite algorithm generates a cycle of size* $2^n - 1$ *that includes each string of size n exactly once, except the all-one string.*

Proof. We will prove that the algorithm works by verifying two steps: (1) the algorithm terminates at 10^{n-1}, and (2) all strings except 1^n are included. To see (1), suppose the algorithm terminates just after the substring $x_1 \cdots x_n \neq 10^{n-1}$ is realized. This means that $x_2 \cdots x_n\bar{x}_n$ and $x_2 \cdots x_nx_n$ must have appeared earlier in the sequence (in the listed order due to preferring the opposite). Since $x_2 \cdots x_n \neq 0^{n-1}$, $x_2 \cdots x_n\bar{x}_n$ must be preceded by some bit b. That is, both $bx_2 \cdots x_n\bar{x}_n$ and $\bar{b}x_2 \cdots x_nx_n$ occur as substrings in the sequence. Since either b or $\bar{b}$ equals x_1, $x_1 \cdots x_n$ occurs twice before termination. This contradiction establishes assertion (1).

To prove (2), suppose a word $\mathbf{w} = x_1 \cdots x_n \neq 1^n$ does not occur. We consider two cases:

Case 1. $x_n = 0$. Since 0^n already occurs at the beginning of the sequence, $\mathbf{w} \neq 0^n$. So there must be at least one 1 in $\mathbf{w}$. Let i be the largest index such that $x_i = 1$. Then $1 < i < n$ where the first inequality follows by assertion (1), and $\mathbf{w} = x_1 \cdots x_{i-1}10^{n-i}$. We claim that $x_2 \cdots x_{i-1}10^{n-i+1}$ does not occur. To see why, suppose it does occur. Then since this sequence is not all 0s, the prefer-opposite strategy implies that $x_2 \cdots x_{i-1}10^{n-i}1$ must occur earlier, so $x_2 \cdots x_{i-1}10^{n-i}$ must occur twice. Since this sequence is not all 0s, it does not occur at the beginning, so each time it occurs it is preceded by something. Therefore both $0x_2 \cdots x_{i-1}10^{n-i}$ and $1x_2 \cdots x_{i-1}10^{n-i}$ must occur, and one of these matches $\mathbf{w}$, which does not occur, so we have a contradiction. Repeating this reasoning $i - 1$ times, we conclude that 10^{n-1} does not occur. But this contradicts assertion (1), which tells us that the sequence ends with 10^{n-1}.

Case 2. $x_n = 1$. Since $\mathbf{w} \neq 1^n$, we can let i be the largest index such that $x_i = 0$. Then $i < n$ and $\mathbf{w} = x_1 \cdots x_{i-1}01^{n-i}$. We now imitate the reasoning in case 1 to conclude that 01^{n-1} does not occur. But then $1^{n-1}0$ does not occur either, as it can only occur by immediately following either 01^{n-1} or 1^n, neither of which occurs. By case 1, the statement that $1^{n-1}0$ does not occur leads to a contradiction. ■

Proposition 2.3. *For any $n \geq 2$, the longest run of* 1*s*, 1^{n-1}, *is the last run of* 1*s in the prefer-opposite sequence.*

Proof. By Theorem 2.2 we know that 01^{n-1} must occur. The proposition claims that the algorithm halts just after realizing the string $01^{n-1}0^{n-1}$. Suppose that this is not the case. Then 01^{n-1} is followed (in a nonoverlapping way) by the string 0^j1 for some $1 \leq j < n-1$. This implies that the string 1^i0^j1, where $i + j = n - 1$, occurs after 01^{n-1}.

Since the algorithm cannot halt at this string, it must be followed by either a 0 or a 1. If it is followed by a 1 then the string $1^{i-1}0^j11$ occurs immediately after 1^i0^j1. If it is followed by a 0, then $1^{i-1}0^j10$ occurs immediately after 1^i0^j1 and therefore $1^{i-1}0^j11$ occurs at some later point, by Theorem 2.2 and the prefer-opposite strategy. In both cases, the string $1^{i-1}0^j11$ occurs later than 1^i0^j1. Applying this argument $n-2$ times implies that 01^{n-1} occurs subsequently for a second time, a contradiction. ■

To produce a full de Bruijn sequence, the algorithm can therefore be adjusted so as to count the number of 1s in the last $n-1$ bits of the running end of the sequence. When this count is $n-1$, we append 10^{n-1} and exit.

Table 1 displays the prefer-opposite sequence as well as the prefer-one sequence without the last $n-1$ 0s for $n = 1, \ldots, 5$. It can be seen that for $n \leq 4$ one of these sequences is the reverse of the other. This is not the case though for $n = 5$.

Table 1. Prefer-one and prefer-opposite sequences with an appended 1 and the last $n-1$ zeroes trimmed.

n	Prefer-one sequence	Prefer-opposite sequence
1	01	01
2	0011	0011
3	00011101	00010111
4	0000111101100101	0000101001101111
5	00000111110111001101011000101001	00000101011010010001100111011111

By construction, the prefer-one sequence is biased toward 1s, while the prefer-opposite tends to keep the balance between 1s and 0s. In fact, Fredricksen [**4**] states that the prefer-one sequence has most of its 1s in the beginning. For $n = 59$ the first 10^6 bits are 90 percent 1s, and the first fourth of the sequence for $n = 22$ is 60 percent 1s. We found that the corresponding percentages of 1s within the first fourths of the prefer-opposite sequences are very close to 50 percent for $n = 10, \ldots, 20$.

3. A GRAPH-THEORETIC PERSPECTIVE.

The above arguments constitute a well-rounded and elegant proof that the prefer-opposite algorithm always forms a full cycle. However, there is another perspective—namely graph-theoretic—that has the added benefit of showing particularly the relationship between the prefer-one and the prefer-opposite algorithms. In fact, they both are very special cases of a general way of generating *all* possible full cycles.

As indicated earlier, there is a one-to-one correspondence between the set of de Bruijn sequences of order n and the set of Eulerian circuits in G_{n-1}. Moreover, there is a one-to-one correspondence between Eulerian circuits in G_{n-1} and spanning trees of G_{n-1} rooted at the vertex 0^{n-1}. Recall that a rooted tree has exactly one root.

In our trees the edges will be directed toward the root, so the root is the only vertex in the tree that is not the initial vertex of any edge in the tree. Fredricksen [**4**] describes in detail a general algorithm that produces a de Bruijn sequence of order n from a rooted spanning tree of G_{n-1}. Briefly, using this algorithm we traverse G_{n-1} by starting at the root and following an edge that is not in the spanning tree. If this edge has been used we follow the edge in the tree. We stop when we get to a vertex whose two outward edges have been used. This defines an Eulerian circuit in G_{n-1} whose edges form a de Bruijn cycle in G_n.

The spanning tree that corresponds to the prefer-one algorithm is defined as follows. Starting with a vertex $x_1 \cdots x_{n-1} \neq 0^{n-1}$, choose the edge that connects to $x_2 \cdots x_{n-1}0$. That is, we construct the prefer-zero spanning tree. It is obviously a spanning tree rooted at 0^{n-1} because there is a unique path from every node to 0^{n-1} whose length depends on the location of the last 1 in the vertex.

Similarly, if one draws the arcs not preferred by the prefer-opposite algorithm, we get the "prefer-same" arcs. That is, starting with a vertex $x_1 \cdots x_{n-1}$ that is different from both 0^{n-1} and 1^{n-1}, we select the edge that goes to $x_2 \cdots x_{n-1}x_{n-1}$. This time we obtain two distinct roots, 0^{n-1} and 1^{n-1}. A vertex in G_{n-1} is definitely connected to one or the other of these two roots. Thus we get a disconnected forest of two trees rooted at 0^{n-1} and 1^{n-1}. A vertex $x_1 \cdots x_{n-1}$ belongs to the tree rooted at 0^{n-1} or 1^{n-1} if its last bit is respectively 0 or 1. Connecting the vertex 1^{n-1} to $1^{n-2}0$ merges the two trees into one spanning tree rooted at 0^{n-1}. Since the edges in this tree are the ones not preferred by the prefer-opposite algorithm, this new edge corresponds to the insertion of an extra 1 after the string 01^{n-1} as discussed after the proof of Proposition 2.3.

ACKNOWLEDGMENTS. I would like to thank the referees whose invaluable comments and detailed review helped improve the clarity of exposition throughout this paper.

REFERENCES

1. F. Chung, P. Diaconis, and R. Graham, Universal cycles for combinatorial structures, *Discrete Math.* **110** (1992) 43–59. doi:10.1016/0012-365X(92)90699-G
2. J. Cooper and R. Graham, Generalized de Bruijn cycles, *Ann. Comb.* **8** (2004) 1325. doi:10.1007/s00026-004-0201-y
3. N. G. de Bruijn, A combinatorial problem, *Koninklijke Nederlandse Akademie v. Wetenschappen* **49** (1946) 758–764.
4. H. Fredricksen, A survey of full length nonlinear shift register cycle algorithms, *SIAM Rev.* **24** (1982) 195–221. doi:10.1137/1024041
5. S. Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, CA, 1967.
6. D. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations, 1st Edition*, Addison-Wesley Professional, Boston, MA, 2005.
7. T. Koshy, *Discrete Mathematics with Applications*, Academic Press, Burlington, MA, 2004.
8. C. Savage, A survey of combinatorial gray codes, *SIAM Rev.* **39** (1997) 605–629. doi:10.1137/S0036144595295272

Department of Mathematics, American University of Beirut, 3 Dag Hammarskjold Plaza, 8th Floor, New York, NY 10017-2303 USA
aa145@aub.edu.lb