Generating De Bruijn Sequences: An Efficient Implementation

F.S. Annexstein

Abstract—This paper presents a concise and efficient implementation of a method of producing De Bruijn sequences. The implementation is based on a recursive method due to Lempel [5]. We provide code for a function that for each pair of integers $n \geq 2$ and $0 \leq x < 2^{n-2}$ returns a unique De Bruijn sequence of order-n. The implementation requires only $O(2^n)$ bit operations

Index Terms—Shift register sequences, De Bruijn graphs, computational complexity, recursive algorithms, NESL programming language.

1 Introduction

THE use of shift registers to produce sequences of zeros and ones with various randomness properties is well established (see, e.g., [1], [4]). In this paper, our concern is the generation of long shift register sequences, known as full cycles or De Bruijn sequences. An *order-n De Bruijn sequence* is a sequence of 2^n bits, which when arranged on a circle contains, as subsequences of consecutive bits, every n-bit sequence. Finding a De Bruijn sequence is equivalent to the problem of finding a Hamiltonian (directed) cycle in a shift-register graph, called a *De Bruijn graph*. A variety of algorithms for finding such sequences can be found in the survey papers of Fredrickson [3] and Ralston [6]. All the algorithms presented therein apparently require $\Omega(n2^n)$ bit operations (bit-ops) to generate an order-n De Bruijn sequence.

In this paper, we provide a particularly concise and efficient implementation for generating De Bruijn sequences. The implementation is based on a recursive method due to Lempel [5]. We provide code for a function **Generate-DeBruijn(n, x)** which takes an integer n, and returns a order-n De Bruijn sequence; a length n–2 bitstring x specifies a unique one out of a possible 2^{n-2} sequences to return. Our implementation requires only $O(2^n)$ bit-ops, whereas a naive implementation would require $\Omega(n2^n)$ bit-ops. In addition, our implementation enjoys extremely good locality of reference. Since nearly all memory references come from scanning arrays. For expository purposes, we provide working code for our implementation using the programming language NESL [2], an applicative language with transparent semantics. NESL's advantage is that it provides high-level scanning primitives, and also provides a simple mechanism for calculating complexity on various serial and parallel machine models. From the code it is readily apparent that a parallel implementation is also quite efficient: only n parallel steps are required to generate an order-n sequence. The NESL code we provide can be easily translated into C-code (available from the author). Using the C language implementation we can generate a De Bruijn sequence of size 2^{20} in under five seconds (wall clock time) on a Sparcstation IPC.

• *The author is with the Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, Cincinnati, OH 45221. E-mail: fred.annexstein@uc.edu.*

Manuscript received December 1994; revised May 1995.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96281.

2 Preliminaries

2.1 The De Bruijn Graph

The 2^n-node *De Bruijn graph* $\mathcal{D}_n$ is the digraph whose node-set is B^n, the set of all n-bit strings. Given a pair of strings $x, y \in B^n$, we write $x \to y$ if there is an arc in $\mathcal{D}_n$ from x to y. The arcs of $\mathcal{D}_n$ are defined as follows: for $a \in \{0, 1\}$ and $w \in B^{n-1}$, we have $a\,w \to w\,a$ and $aw \to w\bar{a}$, where $\bar{a} = 1 - a$. We use the following notation to denote certain string types: Let α denote the string of all zeros, let $\overline{\alpha}$ denote the string of all ones, and let β and $\overline{\beta}$ denote the unique pair of "alternating-strings" (i.e., the strings 10101... and 01010...). When referring to such strings, their lengths shall be clear from context. Note that in each graph $\mathcal{D}_n$ we have $\beta \to \overline{\beta}$ and $\overline{\beta} \to \beta$.

A pair of strings $x, y \in B^n$ are *conjugate* if they differ only in the first (leftmost) bit coordinate. A pair of cycles C_1, C_2 are *adjacent* in $\mathcal{D}_n$ if they are disjoint and there is a conjugate pair $x, \hat{x}$ such that $x \in C_1$ and $\hat{x} \in C_2$. It is easy to show that a pair of adjacent cycles are merged into a single cycle when the successors of an associated conjugate pair are interchanged.

A cycle of length k, or a k-cycle, is a sequence of k distinct strings $x^0, x^1, \ldots x^{k-1}$ where $x^i \to x^{(i+1) \bmod k}$, for each $0 \leq i < k$. It is convenient to represent a k-cycle by its *bit representation*: a sequence of k bits $[b_1\ b_2 \cdots b_k]$ where each b_i is the rightmost bit of the string $x^{i \bmod k}$. With this representation it is natural to view a k-cycle as beginning, or *oriented*, at the node $x^0 = b_{k-n+1} \cdots b_k$. The cycle is (re)constructed by traveling along the edges defined by each successive bit, that is, starting out at x^0 and traveling across the b_1 arc brings us to x^1, then across the b_2 arc brings us to x^2, and so forth, until the cycle is completed by traveling across the b_k arc which brings us back to x^0. Note that the string associated with each node on a k-cycle is contained as a substring of the bit representation $[b_1\ b_2 \cdots b_k]$ viewed as a circular list. Given an oriented k-cycle C containing string γ, let $|\gamma|_C$ $(0 \leq |\gamma|_C \leq k - 1)$ denote the number of arcs required to reach γ from the beginning of the cycle. Equivalently, $|\gamma|_C$ is the index (mod k) of the terminating (rightmost) bit of string γ in the bit representation of the k-cycle defined by C.

2.2 Lempel's Homomorphism

Consider the set of n-bit strings B^n, and let the mapping $D : B^n \to B^{n-1}$ be defined by $D(a_1, \ldots, a_n) = (a_1 + a_2), (a_2 + a_3), \ldots, (a_{n-1} + a_n)$, where addition is modulo 2. It is easily verified that D is a two-to-one onto mapping that induces a graph homomorphism $\mathcal{D}_n \to \mathcal{D}_{n-1}$ [5]. Each arc in $\mathcal{D}_{n-1}$ (with the exception of the two self-loops) is the image under D of a pair of node disjoint arcs in $\mathcal{D}_n$. Hence, by induction it follows that any path in $\mathcal{D}_{n-1}$ is the image under D of a pair of node disjoint paths. Lempel [5] showed that if the number of ones in a k-cycle $[b_1 \ldots b_k]$ of $\mathcal{D}_{n-1}$ is even, then the k-cycle is the image under D of a pair of node disjoint k-cycles in $\mathcal{D}_n$. It is easy to verify that the number of ones in a full 2^n-cycle in $\mathcal{D}_n$ is 2^{n-1}. Hence, for all $n > 1$ the pre-image under D of a full cycle $F = [a_1 a_2 \ldots a_{2^n}]$ (oriented at α) in $\mathcal{D}_n$ is a pair of disjoint cycles C, $\overline{C}$ of $\mathcal{D}_{n+1}$. By applying the inverse of the mapping D, we find that we can represent these cycles as follows: $C = [b_1 b_2 \cdots b_k]$ and $\overline{C} = [\bar{b}_1 \bar{b}_2 \ldots \bar{b}_k]$ where for each $1 \leq i \leq 2^n$, $b_i = (a_1 + a_2 + \cdots + a_i) \bmod 2$. The reader may easily verify that applying the mapping D to each

of the nodes in the cycle C (or cycle $\overline{C}$) produces the cycle F (see also Theorem 5 in [5]). We will assume, without loss of generality, that C is oriented at α and $\overline{C}$ is oriented at $\overline{\alpha}$.

We now describe an *inductive construction* whereby we join the pair $C, \overline{C}$ of cycles of $\mathcal{D}_{n+1}$ into a single full cycle. Let β denote the alternating-string that belongs to the cycle C. It follows that the complementary string $\overline{\beta}$ belongs to $\overline{C}$ since their images under D are identical, i.e., $D(\beta) = D(\overline{\beta}) = \overline{\alpha}$. Since $\beta \to \overline{\beta}$ and $\overline{\beta} \to \beta$, it follows that the conjugate $\hat{\beta}$ of β is the predecessor of $\overline{\beta}$ in $\overline{C}$, and the conjugate $\hat{\overline{\beta}}$ of $\overline{\beta}$ is the predecessor of β in C. Hence, the cycles C and $\overline{C}$ are adjacent, and thus by interchanging the successors of either one of these pair of conjugates we obtain a full cycle. Interchanging the successors of β and $\hat{\beta}$ is called a *type-0 interchange*, and interchanging the successors of $\overline{\beta}$ and $\hat{\overline{\beta}}$ is called a *type-1 interchange*. These two types of constructions are pictured in Fig. 1.

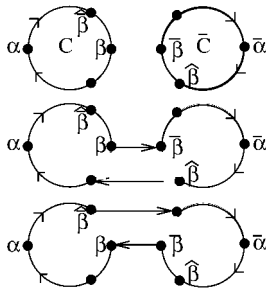

Fig. 1. Joining cycles to form a De Bruijn sequence. The top figure represents the pair of adjacent cycles $C, \overline{C}$ that are the pre-image of some full cycle. The middle figure represents the joining of these cycles via a type-0 interchange, and the bottom via a type-1 interchange.

To maximize efficiency, when generating sequences inductively, it will be important to compute $|\beta|_{\omega}$, the index of the alternating-string in each full cycle ω, so that we have a handle on where to join the pair of cycles needed to obtain the next inductively defined sequence. Recall that $D(\beta) = D(\overline{\beta}) = \overline{\alpha}$, hence, we can use knowledge of $|\overline{\alpha}|$ to inductively compute $|\beta|$, as illustrated by the following proposition.

PROPOSITION 1. *Let ω be any order-n De Bruijn sequence oriented at α, where $n \geq 2$. Let $C, \overline{C}$ denote the pair of adjacent cycles that are the pre-image of ω under D. Let ω^0 and ω^1 denote the order-$(n+1)$ De Bruijn sequences obtained via the type-0 and type-1 interchanges on the cycles $C, \overline{C}$, respectively. Then the following four identities hold.*

1) $|\beta|_{\omega^0} = |\beta|_C = |\overline{\alpha}|_{\omega}$
2) $|\beta|_{\omega^1} = |\beta|_C + 2^n = |\overline{\alpha}|_{\omega} + 2^n$
3) $|\overline{\alpha}|_{\omega^0} = 2^n + 1$
4) $|\overline{\alpha}|_{\omega^1} = 2^n - 1$

PROOF. Since $\overline{\alpha} = D(\beta)$, it is clear that $|\overline{\alpha}|_{\omega}$ the path-length from α to $\overline{\alpha}$ in ω is equal to $|\beta|_C$ the path-length from α to β in C. Since the construction of w^0 leaves the path in C from α to β intact, it follows that $|\beta|_C = |\beta|_{\omega^0}$, and the first identity holds.

The construction of ω^1 leaves the first $|\beta|_C - 1$ arcs in the path in C from α to the predecessor of β intact. Then the ω^1 path hops to and traverses the entire cycle $\overline{C}$ (an additional 2^n arcs). Then the path hops back, thus returning to $\beta \in C$. Hence, $|\beta|_{\omega^1}$ the length of path from α to β in ω^1 is precisely $|\beta|_C + 2^n$, and the second identity holds.

Recall that the number of arcs traversed in the cycle C when traveling from α to β plus the number of arcs traversed when traveling from β back to α is precisely 2^n. Of course, this latter summand is the same as the number of arcs traversed when traveling from $\overline{\beta}$ to $\overline{\alpha}$ in the cycle $\overline{C}$. The arcs traversed when traveling from α to $\overline{\alpha}$ in the cycle defined by ω^0 is precisely the arcs in C from α to β, followed by a hop over to $\overline{\beta} \in \overline{C}$, followed by the arcs traversed when traveling from $\overline{\beta}$ to $\overline{\alpha}$ in $\overline{C}$. Hence, the total number of arcs traversed is $2^n + 1$, and the third identity holds.

The fourth identity follows from an argument similar to that for the third. □

3 THE IMPLEMENTATION

The following function, Next-DeBruijn, takes as input three parameters `w,i,k`, where `w` is a De Bruijn sequence given by its bit representation (oriented at α), an integer `i` is required to be $|\overline{\alpha}|_w$ the path-length in `w` from the origin α to $\overline{\alpha}$, and a single bit `k` specifies the type of interchange to be performed (`k` = 0 specifies a type-0 and `k` = 1 specifies a type-1 interchange). The output of the function is a De Bruijn sequence of the next higher order. We use the NESL language to express the function since the language provides a number of built-in functions that compute *scans* or parallel prefix operations on a sequence. We use a function `xor_iscan` that returns a sequence that is the (inclusive) parallel prefix using the xor operator, i.e., if $x = x_1x_2 \ldots x_n$ is an n-bit string, then `xor_iscan`$(x) = y = y_1y_2 \ldots y_n$, where for each i we have that $y_i = x_1$ **xor** x_2 **xor** $\cdots$ **xor** x_i.

```
function Next-DeBruijn(w, i, k) =
let
  C = xor_iscan(w);          - - apply (inclusive)
                             prefix scan
  Cbar = {1 - a : a in C}; - - compute the bit-
                             complement of the sequence
  part1 = take (C, i - k); - - take the first i - k
                             bits from C
  part2 = drop (Cbar, i - 1 + k); - - drop the first
                             i - 1 + k bits from Cbar
  part3 = take (Cbar, i - 1 + k); - - take the first
                             i - 1 + k bits from Cbar
  part4 = drop (C, i - k);  - - drop the first i - k
                             bits from C
in part1 ++ part2 ++ part3 ++ part4- - concatenate
                             the list for output
```

PROOF OF CORRECTNESS. The fact that cycle C is defined by the bit sequence `C = xor_iscan(w)` follows from the inversion of Lempel's homomorphism (as described above) and the fact that we have oriented the cycle at α. The fact that cycle $\overline{C}$ is defined by `Cbar = {1 - a : a in C}`, the bit complement of `C`, follows similarly.

We consider the case where `k` = 0, i.e., the construction using the type-0 interchange. The correctness of the case where `k` = 1 follows from a similar argument. Recall that the inductive construction makes use of four subpaths defined by dividing the inductively defined cycles C and $\overline{C}$ into two pieces each. For the type-0 interchange, the first piece of the full cycle is the portion of C that defines a path

from α to β. We know from Proposition 1 that $|\beta|_w$ the path-length from α to β is `i` = $|\overline{\alpha}|_w$. Hence, by taking `part1` as the first `i` bits from `C` we isolate the subpath of C from α to β. The second subpath we need is the path in $\overline{C}$ from $\overline{\beta}$ to $\overline{\alpha}$. Note that appending the `i`th bit of `Cbar` to `part1` will bring the path to node $\overline{\beta}$. It follows that by dropping the first `i` -1 bits of `Cbar`, called `part2`, and appending this to `part1`, we achieve a path from α to $\overline{\alpha}$. Appending `part3`, the first `i` -1 bits of `Cbar`, finishes the path at the predecessor node of $\overline{\beta}$, and completes the visitation of all nodes in the cycle $\overline{C}$. Finally, appending `part4`, the last |`C`| $-$ `i` bits of `C`, brings us back to complete the cycle C by beginning at the successor of β and finishing at α; and thus, the full cycle is completed. □

We now define a function that will generate a De Bruijn sequence of any order. The function makes use of that fact that there are two ways (via the type-0 and type-1 interchanges) to build a De Bruijn sequence of order-n from one of order-($n-1$). The function takes as input an integer $n \geq 2$ and a sequence x of $n-2$ bits (indexed from 0 to $n-3$), one bit specifying the interchange type for each inductive step. For each input pair the function returns a unique De Bruijn sequence of order n.

```
function Generate-DeBruijn (n, x) =
if (n == 2) then [1,1,0,0]
else  if (n == 3) then
         if (x [0] == 0) then [1,0,1,1,1,0,0,0]
         else [1,1,1,0,1,0,0,0]
       else Next-DeBruijn (Generate-DeBruijn(n–1,x),
            2 ^ (n–2) + (-1) ^ (x[n–4]), x[n–3]);
```

PROOF OF CORRECTNESS. The proof follows by induction on n. The parameter associated with `i` in the call to **Next-DeBruijn** is the expression 2 ^ (n – 2) + (–1) ^ (x[n – 4]). Recall that this expression is required to be equal to $|\overline{\alpha}|_w$, where `w` is the De Bruijn sequence associated with the first parameter. This is indeed the case, for the expression corresponds to the identity for $|\overline{\alpha}|_w$ in Proposition 1 (identities 3 and 4), since the expression reflects that the sequence `w` was constructed via an interchange of type represented by the bit $x[n-4]$. Hence, the correctness of **Generate-DeBruijn** follows from the correctness of the **Next-DeBruijn** function, and the fact that the stopping conditions are easily verified. □

ACKNOWLEDGMENT

I thank Humberto Ortiz-Zuazaga for his help with the C-language implementation. This research has been supported in part by U.S. National Science Foundation Grant CCR–93–09470.

REFERENCES

[1] E.R. Berlekamp, *Algebraic Coding Theory*. Laguna Hills, Calif.: Aegean Park Press, 1984.
[2] G.E. Blelloch, "NESL: A Nested Data-Parallel Language," Technical Report CMU-CS-94, Carnegie Mellon Univ., 1994.
[3] H. Fredrickson, "A Survey of Full Cycle Algorithms," *SIAM Review*, vol. 24, pp. 195–221, 1982.
[4] S.W. Golomb, *Shift Register Sequences.* San Franciso: Holden-Day, 1967.
[5] A. Lempel, "On a Homomorphism of the De Bruijn Graph and Its Applications to the Design of Feedback Shift Registers," *IEEE Trans. Computers*, vol. 19, no. 12, pp. 1,204–1,209, Dec. 1970.
[6] A. Ralston, "De Bruijn Sequences—A Model Example of the Interaction of Discrete Mathematics and Computer Science," *Am. Math. Monthly*, vol. 55, no. 3, pp. 131–143, May 1982.