Algorithms for De Bruijn Sequences – A Case Study in the Empirical Analysis of Algorithms

M. VASSALLO[1] AND A. RALSTON[2]

[1] *Suny College at Fredonia, N.Y., USA*

[2] *Department of Computer Science, SUNY, University at Buffalo, 226 Bell Hall, Buffalo, New York 14260, USA*

The following paper presents an analysis of algorithms for De Bruijn sequences. It is unusual in that the analysis is carried out empirically rather than analytically.

Received June 1990, revised March 1991

1. THE ANALYSIS OF ALGORITHMS

Most published work about the analysis of algorithms is analytic because, when analytic results can be obtained, they are usually decisive, often elegant and sometimes beautiful. However, such analyses are not always available either because they are mathematically intractable or because the analysis is indecisive. In such cases, empirical analysis may be used to distinguish the effectiveness of one algorithm from another. Indeed, this situation is more common than the computer science literature might lead you to believe. This note is an example of such an empirical analysis.

2. DE BRUIJN SEQUENCES

Definition. Let $S = \{0, 1, \ldots, m\}$ be an alphabet of $m+1$ symbols and consider all words of length n from S. Let $L = (m+1)^n$. An (m, n) de Bruijn sequence $B_{m,n}$ is a sequence

$$a_1 a_2 a_3, \ldots, a_L$$

with each $a_i \varepsilon S$ such that every word w of length n from S is realized as

$$a_i a_{i+1}, \ldots, a_{i+n-1} \quad (0 \leqslant i \leqslant L) \qquad (1)$$

for exactly one i, where each subscript in (1) is to be interpreted in modulo L.

For example, with $m = 1$ and $n = 3$ so that $L = 8$, the sequence 00010111 contains all binary sequences of length 3 with 110 and 100 being obtained by wraparound from the end of the sequence to the beginning.

The existence of de Bruijn sequences for any m and n can be proved using graph theory or finite field theory [3] but the algorithms suggested by these proofs are clearly very inefficient from both time and space perspectives. The known good algorithms for generating de Bruijn sequences all depend on a strictly combinatorial approach.

3. THREE COMBINATORIAL ALGORITHMS FOR DE BRUIJN SEQUENCES

An early combinatorial algorithm due to Martin [2] was inefficient because it required L units of memory. By contrast, the three algorithms to be considered here are all essentially memoryless since each requires $O(n)$ units of storage. Improving the space attributes also improves the timing properties by requiring much less memory referencing. The three algorithms to be considered here all require $O(L)$ time, clearly the minimum possible.

Two of the three algorithms depend upon the following two definitions:

Definition. Let $S = s_1, s_2, \ldots, s_n$ and let $T = s_1 s_2, \ldots, s_j, j < n$. Denote by T^k the subsequence of k consecutive repetitions of T. If $S = T^k$ when $k > 1$, we say that S is periodic with repetition (or periodicity) k and T is its periodic reduction. If there is no T with $k > 1$ for which $S = T^k$, we say that S is aperiodic.

Definition. The lexically largest permutation, $\pi_1, \pi_2, \ldots, \pi_n$, of an n-set $\alpha_1, \alpha_2, \ldots, \alpha_n$, is such that $\pi_1 \geqslant \pi_2 \geqslant \ldots \geqslant \pi_n$. (Thus, the lexically largest permutation is the numerically largest). If $S = s_1 s_2, \ldots, s_n$ and $T = t_1 t_2, \ldots, t_n$ are two strings of length n, then $S \geqslant T$ iff $s_i = t_i$, $i = 1, \ldots, j$, and $s_{j+1} > t_{j+1}$. Equality holds only when $j = n$. A necklace S of length n is an n-sequence with the property that $S >= T$ for every n-sequence T which is a cyclic permutation of S. Thus, if $S = s_1, s_2, \ldots, s_n$ is a necklace, then

$$S \geqslant s_i s_{i+1}, \ldots, s_n s_1, \ldots, s_{i-1}$$

for $2 \leqslant i \leqslant n$.

The first algorithm, due to Fredricksen and Maiorana [1], outputs the lexically largest (m, n) de Bruijn sequence by generating successive necklaces of length n.

Algorithm FM

Input: two positive integers m, n

Step 1.
Start with the empty string.

Step 2. (Iterative Step)
Generate the necklaces of length n whose first symbol is m, in decreasing lexicographic order. Append each necklace to the string already generated if it is aperiodic or its periodic reduction otherwise.

Step 3. (Recursive Step)
If $m = 1$ then append 0 to the string already generated, else append $B^{(FM)}_{m-1,n}$ to the string already generated.

Output: $B^{(FM)}_{m,n}$

Since Algorithm FM requires the generation of successive necklaces, it requires a subalgorithm for this purpose which can be found in [3].

As an example of Algorithm FM, let $n = 4$ and $m = 2$. We obtain:

$$B_{2,4}^{(FM)} = 2\ 2221\ 2220\ 2211\ 2210\ 2201\ 2200\ 21\ 2120\ 2111\ 2110\ 2101\ 2100\ 20\ 2011\ 2010\ 2001\ 2000\ 1\ 1110\ 1100\ 10\ 1000\ 0,$$

where each group shown is a necklace or its periodic reduction.

A second algorithm, Algorithm R, due to Ralston [4], is a variation on Algorithm FM which uses a double recursive approach (see Appendix).

For the third algorithm, due to Xie [5], we need to introduce two more definitions.

Definition. Let $L1 = (m+1)^{n-1}$. If a sequence

$$0t_1 t_2, \ldots, t_{L1-1}$$

is such that for any j ($1 \leqslant j \leqslant L1$), there exists another sequence $j_1 j_2, \ldots, j_s$ ($s \leqslant L1$) where

$$j_s = 0, \quad j_k \neq 0 \quad (k < s)$$

and

$$j_1 = j, \quad j_{k+1} = ((m+1) * j_k + t_{j_k}) \mod L1 \quad (1 \leqslant k < s) \tag{2}$$

then $0t_1 t_2, \ldots, t_{L1-1}$ is called a label (m, n).

Note: $000, \ldots, 0$ ($L1$ 0's) is always a label.

As an example, 0001 is a label (3, 2) since there are sequences 10 for $j = 1$, 20 for $j = 2$, and 310 for $j = 3$. However, 0003 is not a label (3, 2) since for $j = 3$, the sequence generated by (2) is 3333.

Definition. A look-up table is a two-dimensional array consisting of $L1$ rows. Each row consists of an index i, $0 \leqslant i \leqslant L1-1$, and a sequence of symbols $i_0, i_1, \ldots, i_m$ in which each sequence is a permutation of the set $\{0, 1, \ldots, m\}$.

Let us now consider the special look-up table shown in Table 1 in which $0t_1 t_2, \ldots, t_{L1-1}$ is a label (m, n) and $xx, \ldots, x$ in row i is $01 \ldots (t_{i-1})(t_{i+1}) \ldots m$.

Table 1

Index	Sequence				
0	0	1	2	...	m
1	t_1	x	x	...	x
.	.	.	.	.	.
.	.	.	.	.	.
$L1-1$	t_{L1-1}	x	x	...	x

Xie proves that Algorithm X, associated with Table 1, generates a (m, n) de Bruijn sequence. In particular, if $0t_1, t_2, \ldots, t_{L1-1} = 00, \ldots, 0$ then the sequence generated is the lexically largest (m, n) de Bruijn sequence.

Algorithm X

Input: positive integers m, n

Step 1.

Generate a label (m, n) represented by $0t_1, t_2, \ldots, t_{L1-1}$

Step 2.

Construct Table 2 as described above

Step 3.

Start with the index $i = 0$ and set the sequence to be generated to be empty

Step 4. (Iterative Step)

If the row with index i in the look-up table is empty, stop; otherwise append the rightmost symbol s in this row to the sequence already generated, remove this symbol from the row and evaluate

$$i \leftarrow ((m+1) * i + s) \mod L1$$

Output: an (m, n) de Bruijn sequence

For example, with $m = 2$, $n = 3$, $L1 = 9$. If we take the label in Step 1 to be all zeros, then the look-up table is:

Table 2

Index	0	1	2	3	4	5	6	7	8
Sequence	012	012	012	012	012	012	012	012	012

Applying Step 4, we obtain:

$$B_{2,3}^{(X)} = 222122021121020120011101000.$$

This remarkable but rather unintuitive algorithm is, as we shall see in the next section, very efficient because of the extremely simple calculation in Step 4.

4. RESULTS

We implemented Algorithms FM, R and X in Pascal on a VAX 11/785 and a Burroughs B7900. Since the results were similar on both computers, we give the results only for the former. The implementation, so far as possible, had similar characteristics with respect to modularity and data structures. The lines of code for the three algorithms were as follows: FM-155, R-318, X-75. The table which follows gives the results for the three algorithms for various values of m and n.

Table 3. Time to generate $B_{m,n}$ on a VAX 11/785 (cpu secs)

m	2	2	2	2	2	2	3	3	3
n	3	4	5	6	7	8	3	4	5
FM	2.47	2.71	3.41	5.91	13.97	39.87	2.63	3.41	6.93
R	2.62	2.80	3.57	5.93	13.54	39.22	2.74	3.47	6.90
X	0.10	0.16	0.41	1.19	3.54	10.60	0.14	0.44	1.64
m	3	4	4	5	5	6	6	7	7
n	6	3	4	3	4	3	4	3	4
FM	22.13	2.83	4.88	3.23	7.73	3.66	12.01	4.31	18.98
R	21.72	2.86	4.87	3.19	7.29	3.63	11.69	4.26	17.97
X	6.32	0.26	1.05	0.36	2.01	0.58	3.58	0.86	6.12

The superiority of Xie's algorithm is clear, particularly so for increasing values of n for each m. This superiority is not obvious from the algorithms themselves nor could an analytic analysis have shown this since all three algorithms are $O(L)$ for time. A posteriori we may infer that the simplicity and elegance of the idea in Xie's algorithm accounts for its computational efficiency.

Empirical studies of the performance of algorithms whose analytic orders are the same will not usually result in quite such definitive results as obtained in this case. Still computer scientists and mathematicians should be willing to take an experimental approach to the analysis of algorithms when analytic approaches are not fruitful.

5. APPENDIX

Definition. An aperiodic block is a string of aperiodic necklaces of decreasing numerical value with m's in the same position in each necklace (e.g. with $m = 2$, $n = 4$, we get 2211 2210 2201 2200).

Definition. Let T be the periodic reduction of a periodic necklace S with repetition k (e.g. $S = 2121$, $T = 21$, $k = 2$). Let t be the number of $(m-1)$s in T, $u = m^t - 1$ and $h_0 < h_1 < h_2 < \ldots < h_u$ be the strings with ms in the same position as in T and nowhere else (e.g. $t = 1$, $u = 2^1 - 1 = 1$, $h_0 = 20 < h_1 = 21$). Let $B^{(R)}_{u,k}$ be the de Bruijn sequence generated by Algorithm R below for $m = u$, $n = k$ (e.g. for $u = 1$, $k = 2$, this turns out to be $B^{(R)}_{1,2} = 1\ 10\ 0$). A periodic block is one in which each symbol j in $B_{u,k}$ is replaced by h_j (e.g. 21 2120 20).

Definition. A group G_j is the sequence of periodic and aperiodic blocks whose initial j symbols are each m, ordered by the value of the initial block of length n in each. For example, with $m = 2$, $n = 4$, $j = 1$, then

$G_1 =$ 2121 2020 (periodic block)
2111 2110 2101 2100 2011 2010 2001 2000 (aperiodic block)

Then here is **Algorithm R**:

Input: two positive integers m, n
Step 1.
Start with the string consisting of the single symbol m
Step 2. (Iterative Step)
For $j = n-1, n-2, \ldots, 1$ append G_j to the string already generated. (This will generally involve a recursive call to the algorithm to compute $B^{(R)}_{u,k}$)
Step 3. (Recursive Step)
If $m = 1$, then append 0 to the string already generated; else, append $B^{(R)}_{m-1,n}$ to the string already generated
Output: $B^{(R)}_{m,n}$

For $n = 4$, $m = 2$, the following output is obtained (Ralston [3]):

$B^{(R)}_{2,4} = 2$ [Step 1]
2221 2220 $[G_3]$
2211 2210 2201 2200 $[G_2]$
2121 2020 2111 2110 2101 2100 2011 2010 2001 2000 $[G_1]$
1 1110 1100 10 1000 $[B^{(R)}_{1,4}]$
0

REFERENCES

1. H. Fredricksen and J. Maiorana, Necklaces of beads in k colors and k-ary de Bruijn sequences, *Discrete Math.*, **23**, 207–210 (1978).
2. M. H. Martin, A problem in arrangements, *Bull. Amer. Math. Soc.*, **40**, 859–864 (1934).
3. A. Ralston, De Bruijn Sequences – A Model Example of the Interaction of Discrete Mathematics and Computer Science, *Mathematics Magazine*, **55**(3) (May 1982).
4. A. Ralston, A New Memoryless Algorithm for De Bruijn Sequences, *J. Algorithms*, **2**, 50–62 (1981).
5. S. Xie, Notes on De Bruijn Sequences, *J. Disc. App. Math.* **16**, 157–177 (1987).