A New Algorithm for Generating Necklaces Extended Abstract

Terry MinYih Wang
Carla D. Savage*
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695

July 12, 1990

Abstract

In this paper we present an algorithm for generating the lexicographically smallest representatives of all the n-bead necklaces in k-colors. The time required is $O(nN_k^n)$, where N_k^n is the number of necklaces with n beads of k colors. To our knowledge, this is the first algorithm for this problem which has been proved to asymptotically improve the obvious $O(n * k^n)$ approach of examining all k-ary n-tuples and rejecting those which are not lexicographically smallest rotations. Our algorithm has another advantage over a competitive one proposed by Fredericksen, Kessler, and Maiorana in that it can be implemented so that there is a gap of at most one non-necklace examined between any two necklaces generated, whereas the FKM algorithm can have gaps as large as $\lfloor (n-1)/2 \rfloor$. Although no good upper bound was established for FKM, we show experimentally that its performance appears to be as least as good as ours for $k > 2$.

1 Introduction

In this paper, we consider the problem of efficiently generating necklaces. A *necklace of n beads in k colors* is an equivalence class of k-ary n-tuples under rotation. That is, if σ denotes the rotation

$$\sigma(x_{n-1} \ldots x_0) = x_{n-2} \ldots x_0 x_{n-1}$$

*This work was supported in part by the National Science Foundation Grant No. CCR8906500.

then the k-ary n-tuples x and y are in the same equivalence class or *necklace* if and only if $\sigma^j(x) = y$ for some integer j.

Let N_k^n denote the number of n-bead necklaces in k colors. Since no necklace can contain more than n elements, and the necklaces a^n for $0 \leq a \leq k-1$ have only one element,

$$N_k^n \geq k + ((k^n) - k))/n$$

and equality holds when n is prime. So, in applications where every rotation of a given k-ary n-tuple represents the same phenomenon, it would be more efficient to work with only one representative of each necklace, rather than all k^n of the n-tuples. A necklace is thus identified with its representative.

A simple and elegant algorithm was proposed in [FrMa] and [FrKe] to generate for each necklace the lexicographically largest element. We will refer to this as the *FKM algorithm*. A disadvantage of the FKM algorithm is that there can be gaps in which as many as $\lfloor (n-1)/2 \rfloor$ non-necklaces are examined between any two necklaces generated. For the case $k = 2$ the authors of [FrKe] prove an upper bound of $\lfloor (n-1)/2 \rfloor N_k^n$ on the number of n-tuples generated and examined by the algorithm. The time spent generating and checking each element is $O(n)$, giving an overall time for the algorithm of $O(n^2 N_k^n)$. However, this does not establish an asymptotic improvement over the direct approach of generating *every* k-ary n-tuple and checking each one to see if it is the representative of its necklace, at a cost of time $O(n)$ per check.

In this paper, we describe a new algorithm which generates the lexicographically smallest element in each necklace. The number of k-ary n-tuples generated and examined by our algorithm is at most *twice* the total number of necklaces, at a cost of $O(n)$ per n-tuple. This gives a total time of $O(nN_k^n)$. Thus, in contrast to the FKM algorithm, we can prove our algorithm gives an asymptotic improvement over the obvious approach of testing in linear time every k-ary n-tuple. Our algorithm also has the advantage that it can be implemented so that there is a gap of at most one non-necklace examined between any two necklaces generated.

We also present the results of our experiments comparing the actual performance of our algorithm against the FKM algorithm. The indication is that (1) our algorithm is better than FKM for $k = 2$, (2) FKM is at least comparable to our algorithm for $k > 2$ and (3) the upper bound on FKM from [FrKe] is almost certainly not tight.

In Section 2 we describe our basic algorithm for the case $k = 2$, prove that it examines no more than $2N_k^n$ of the n-tuples, discuss how to implement it to

minimize gaps, and compare it to FKM. In Section 3 we extend to arbitrary k and prove that it is still possible to stay within the $2N_k^n$ bound.

Most proofs have been omitted in this extended abstract.

2 Necklaces of Beads in Two Colors

Our idea for generating necklaces of two-color beads was inspired by a result in [LiHiCa] that a certain variation on the shuffle-exchange graph is Hamiltonian.

When $k = 2$, the n-tuples are bit strings which we regard as elements of $\{0,1\}^n$. Let $\alpha\beta$ denote the concatenation of bitstrings α and β and let 0^t denote the bitstring of length t in which every symbol is 0.

Define a function τ on $\{0,1\}^n$ by

$$\tau(x) = \begin{cases} \alpha 0 & \text{if } x = \alpha 1 \text{ for some } \alpha \in \Sigma^{n-1} \\ \alpha 1 & \text{if } x = \alpha 0 \text{ for some } \alpha \in \Sigma^{n-1} \end{cases}$$

As in the previous section, $\sigma(x)$ denotes the rotation of string x one position left.

We choose the *lexicographically smallest* string as the representative of a necklace and refer to this string itself as a necklace. A tree of necklaces is generated as follows. Starting with the string $x = 0^n$ as root, we generate as the children of x all those necklaces of the form $\tau\sigma^j(x)$ for some j satisfying $1 \leq j \leq n-1$. This procedure is applied recursively to each child of x. We show first that every necklace will be generated.

Lemma 1 *Every n-bead necklace will appear in the tree with root 0^n.*

Proof. Let x be an n-bead necklace. We show by induction on t, the number of ones in x, that x is in the tree. If $t = 0$ then $x = 0^n$, which is the root of the tree. For $t > 0$, note that $x = \alpha 1$ for some $\alpha \in \{0,1\}^{n-1}$ since x must be the lexicographically smallest in its class. Thus $x = \tau\sigma(0\alpha)$. The representative y of the class containing 0α has $t-1$ ones, so must be in the tree, by induction. Since 0α is a rotation of y, $x = \tau\sigma^j(y)$, for some j, so x will be a child of y in the tree. □

Although at first glance it would appear that we need to examine *every* string of the form $\alpha 1$ with this procedure, this is not the case. We show in Theorem 1 below that if x, $\tau(x)$, and $\sigma(x)$ are not necklaces, then $\tau\sigma(x)$ is not a necklace. It follows that for a *necklace* $y \neq 00\ldots01$, that if $\tau\sigma^j(y)$ is *not* a necklace for some j, then $\sigma^{j+1}(y) = y$, $\tau\sigma^{j+1}(y)$ is also *not a necklace*.

As a result, from a necklace y, we generate the $\tau\sigma^j(y)$, starting with $j = 1$, until the first j is found for which $\tau\sigma^j(y)$ is not a necklace. By Theorem 1, we are guaranteed that at this point all children of y have been found.

Lemma 2 *If $x = 0^t1\alpha1$, where $t > 0$ and $\alpha \in \{0,1\}^*$, then*

(a) if there are $\beta, \gamma \in \{0,1\}^*$ such that $\alpha = \beta 0^{t+1}\gamma$ then x is not a necklace.

(b) if x is not a necklace, then there are $\beta, \gamma \in \{0,1\}^*$ such that $\alpha = \beta 0^t \gamma$.

Theorem 1 *For bitstring $x \in \{0,1\}^n$, if x, $\tau(x)$, and $\sigma(x)$ are not necklaces, then $\tau\sigma(x)$ is not a necklace.*

We summarize below the recursive algorithm for generating the lexicographically smallest representatives of the n-bead necklaces in two colors. Note that by Theorem 1 and as discussed above, while generating the children of the necklace y in the tree, at most one n-tuple is examined which is *not* a necklace. Thus the total number of n-tuples examined is at most $2N_2^k$.

This algorithm makes use of the fact that a necklace of the form $y = 1\alpha$ will not have any children.

Two Color Algorithm

```
procedure search(y);
        begin
                output(y);
                done := false;
                while (not(done) and
                    (the first symbol of y is 0)) do begin
                        y := sigma(y);
                        x := tau(y);
                        if x is a necklace then search(x)
                                else done := true
                        end
        end;

main program
        begin
                y := 00...0;
                output(y);
```

```
            y := tau(y);
            search(y)
    end.
```

A recursive implementation will give rise to *gaps* in this algorithm: as necklaces are implicitly stacked, it could occur that for several consecutive necklaces on the stack, the next candidate "child" to be tested turns out to be a non-necklace. We can avoid this gap with a nonrecursive implementation in which a necklace y is pushed on the stack only after checking that the next n-tuple x to be examined when y is popped is actually a necklace. In this case, x can be saved on the stack with y to avoid generating and checking x twice.

Finally, note that in the two color case 01^{n-1} is the only necklace of the form 01α which has a child in the tree. Thus we could search in the Two Color algorithm for children of y only when y has the form 00α and then output 1^n at the end of the main procedure. This helps for small values of n by delaying the asymptotic behavior.

With this modification incorporated, our algorithm is compared with FKM in the table of Figure 1. The ratio of non-necklaces examined by the algorithms to the total number of necklaces is compared.

3 Necklaces of Beads in k Colors

In the general case of generating necklaces of beads with k colors, let $\Sigma = \{0, 1, \ldots k-1\}$. For each $a \in \Sigma \setminus \{0\}$, define a function τ_a on elements of Σ^n by

$$\tau_a(x) = \begin{cases} \alpha a & \text{if } x = \alpha 0 \text{ for some } \alpha \in \Sigma^{n-1} \\ \text{undefined} & \text{otherwise} \end{cases}$$

We propose to generate a tree of all n-bead k-color necklaces as follows. Starting from the root $y = 0^n$, generate as children of y all *necklaces* of the form $\tau_a \sigma^t(y)$ for $a \in \Sigma \setminus \{0\}$, $1 \le t \le n-1$. This procedure is then applied recursively to the children of y. We can prove as in Lemma 1 that all necklaces will be in the tree with root 0^n. We focus now on minimizing the number of non-necklaces examined when generating the children of a necklace y.

Lemma 2 of Section 2 is still valid and can be restated as follows.

Lemma 3 *If $x = 0^t a \alpha b$ where $t > 0, \alpha \in \Sigma^*$, and $a, b \in \Sigma \setminus \{0\}$ then*

(a) if there are $\beta, \gamma \in \Sigma^*$ such that $\alpha = \beta 0^{t+1} \gamma$ then x is not a necklace.

(b) if x is not a necklace, then there are $\beta, \gamma \in \Sigma^*$ such that $\alpha = \beta 0^t \gamma$.

n	N_2^n	B/N for FKM	B/N for Our Algorithm
1	2	0	0
2	3	0	0
3	4	.250	0
4	6	.333	0
5	8	.750	.250
6	14	.643	.286
7	20	1.050	.500
8	36	.972	.556
9	60	1.117	.667
10	108	1.093	.722
11	188	1.191	.798
12	352	1.122	.824
13	632	1.179	.870
14	1182	1.147	.892
15	2192	1.153	.914
16	4116	1.138	.931
17	7712	1.141	.945
18	14602	1.126	.955
19	27596	1.125	.964
20	52488	1.115	.971
21	99880	1.111	.977
22	190746	1.104	.981
23	765768	1.100	.985
24	699252	1.195	.988
25	1342184	1.092	.990

Figure 1: Comparison of our algorithm to FKM for $k = 2$. (N is the number of necklaces and B is the number of non-necklaces examined.)

We now show that a variation on Theorem 1 holds for k colors.

Theorem 2 *If $x \in \Sigma^n$ is not a necklace and if $\tau_a(x)$, $a \in \Sigma \setminus \{0\}$, is either (i) undefined or (ii) is not a necklace, then $\tau_a\sigma(x)$ is either undefined or is not a necklace.*

By Theorem 2, when generating the children of necklace y, one need only examine, for each $a \in \Sigma \setminus \{0\}$, the strings

$$\tau_a\sigma^1(y), \tau_a\sigma^2(y), \ldots$$

until the smallest t is found for which $\tau_a\sigma^t(y)$ is undefined or is not a necklace. This would imply at most $k-1$ non-necklaces need be examined when generating the children of y.

However, we can do better. At necklace y, for each j satisfying $1 \leq j \leq n-1$, let $s(j)$ be the largest $l \leq n-1$ for which $\tau_l\sigma^j(y)$ is either not defined or is not a necklace. Theorem 2 shows that

$$s(1) \leq s(2) \leq \ldots$$

but in fact we have the following.

Lemma 4 *If $s(j) > 0$ for some j, then $s(j+1) = k-1$.*

Finally, we make use of the following lemma, a variation of which appears in [FrMa].

Lemma 5 *For $x = 0\alpha \in \Sigma^n$, and $a \in \Sigma \setminus \{0\}$, if $\tau_a\sigma(x)$ is not a necklace, the neither is $\tau_{a-1}\sigma(x)$.*

Putting these results together we have the following.

Theorem 3 *Let the candidate children of necklace y be generated in the order:*

$$\tau_{k-1}\sigma^1(y), \tau_{k-2}\sigma^1(y), \ldots, \tau_{s(1)}\sigma^1(y),$$

$$\tau_{k-1}\sigma^2(y), \tau_{k-2}\sigma^2(y), \ldots, \tau_{s(2)}\sigma^2(y),$$

$$\vdots$$

Then all children of y have been generated by the time we examine the first element which is undefined or not a necklace.

Theorem 2 establishes that no more than $2N_k^n$ of the n-tuples are generated and examined while growing the tree of necklaces.

As in the two color case, we can implement this algorithm nonrecursively in such a way that no more than one non-necklace will be examined between any two necklaces. We can again postpone the asymptotic behavior by generating children of necklace y only if y is of the form $y = 00\alpha$; necklaces with no '0' beads can be generated recursively as $k-1$-color necklaces. Our algorithm, with this modification incorporated, is compared with FKM in Figure 2.

4 Final Remarks

Two obvious open questions remain. There should be a way to prove that the FKM algorithm actually examines at most cN_k^n of the n-tuples for some constant c. Extrapolating the results in Figure 2, it seems that for $k > 2$ the FKM algorithm may always have B/N smaller than our algorithm. It would be nice to be able to prove this.

More important, is there a way to generate all necklaces in total time $O(N_k^n)$, ignoring the time for output? There are several combinatorial classes for which such a result is possible using Gray code - like algorithms ([Gi], [Jo], [Lu], [Ru], [Sa]). Alternatively, some clever encoding scheme, taking advantage of the structure of the tree, may allow one to reduce or eliminate the $O(n)$ cost of necklace testing in our algorithm.

References

[FrKe] H. Fredericksen and I. J. Kessler, "An algorithm for generating necklaces of beads in two colors," *Discrete Mathematics* 61 (1986) 181-188.

[FrMa] H. Fredericksen and J. Maiorana, "Necklaces of beads in k colors and k-ary de Bruijn sequences," *Discrete Mathematics* 23, No. 3 (1978) 207-210.

[Gi] E. N. Gilbert, "Gray codes and paths on the n-cube," *Bell Systems Technical Journal* (1958) 815-826.

[Jo] S. M. Johnson, "Generation of permutations by adjacent transpositions," *Math. Comp.* 17 (1963) 282-285.

n	k	N_2^n	B/N for FKM	B/N for Our Algorithm
3	3	11	.273	0
4	3	24	.333	.042
5	3	51	.569	.176
6	3	130	.508	.231
7	3	315	.613	.308
8	3	834	.580	.360
9	3	2195	.595	.413
10	3	5934	.581	.460
11	3	16107	.582	.506
12	3	44368	.571	.547
13	3	122643	.568	.586
14	3	341802	.562	.621
15	3	956635	.558	.655
16	3	2690844	.554	.684
4	4	70	.286	.043
5	4	208	.413	.115
6	4	700	.377	.153
7	4	2344	.410	.190
8	4	8230	.393	.223
9	4	29144	.393	.255
10	4	104968	.385	.287
11	4	381304	.381	.318
5	5	629	.318	.097
6	5	2635	.294	.105
7	5	11165	.305	.127
8	5	48915	.294	.149
9	5	217045	.292	.170
6	6	7826	.235	.075
7	6	39996	.242	.091
8	6	210126	.235	.105
7	7	117655	.201	.068
8	7	720916	.195	.079

Figure 2: Comparison of our algorithm to FKM for $k > 2$. (N is the number of necklaces and B is the number of non-necklaces examined.)

[LiHiCa] W. Liu, T. H. Hildebrandt, and R. Cavin III, "Hamiltonian cycles in the shuffle-exchange network," *IEEE Transactions on Computers*, to appear.

[Lu] J. Lucas, "The rotation graph of binary trees is Hamiltonian," *Journal of Algorithms* 8 (1987) 503-535.

[Ru1] F. Ruskey, "Adjacent interchange generation of combinations," *Journal of Algorithms* 9 (1988) 162-180.

[Sa1] C. Savage, "Gray code sequences of partitions," *Journal of Algorithms* 10, No. 4 (1989).