ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 2 — Приложна математика и информатика
Том 89, 1995

ANNUAIRE DE L'UNIVERSITE DE SOFIA „ST. KLIMENT OHRIDSKI“

FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 2 — Mathématiques Appliquée et Informatique
Tome 89, 1995

SOME EXAMPLES OF LEXICOGRAPHIC ORDER ALGORITHMS AND SOME OPEN COMBINATORIAL PROBLEMS

DIMITAR L. VANDEV

A general reasoning based on the lexicographic order is studied. It helps to create algorithms for generation of sets of words having certain natural and good properties. Several examples are considered and the performance of the proposed algorithms is calculated. An open combinatorial problem regarding the set of partitions arises.

Keywords: enumerating algorithms, lexicographic order functions.
1991/95 Mathematics Subject Classification: 68E05, 65C20.

1. INTRODUCTION

There are numerous examples of sets of words — vectors of natural numbers, which as one set of entities may be used for some computational purposes: sets of all permutations, combinations and many others. In many cases one needs to go across such a set and perform some computations for each member. (See [3] for many examples in combinatorial calculations.)

The problem of the efficient generation of all elements of a class of combinatorial configurations with given properties is considered as an important problem in the theory of algorithms. The generation in a prescribed lexicographical order is one of the most investigated cases, see [5, 4].

In the present paper an attempt is made to use the lexicographic order of these words as a tool for creating enumerating (or generating) algorithms. It turns out that the proposed scheme is useful also for calculating the performance of the

algorithms. In some cases it is possible to calculate it easily, while in others an open problem arises.

A part of these examples were presented as a short communication at the Seminar of Statistical Data Analysis in Varna, Bulgaria, see [7].

2. DEFINITIONS AND NOTATIONS

Let N be the set of natural numbers $\{0, 1, \ldots, n\}$. Call N alphabet. Denote by $S = S(m, n)$ the m-times Cartesian product of the set N. The elements of the set S are called words with a fixed length m and a common alphabet N. It is clear what a lexicographic order in this set means. One word is called "larger" than another if its first (after the common beginning of the two words) letter is larger than the corresponding letter of the second word. Note that the set of numbers (with leading zeroes) with digits from N is ordered in the same manner.

For any subset W of the set S this order induces the same order for the elements of W. To make things more formal, we shall call the word formed by the first k letters of the word w prefix and denote it by $w(k)$. The notation $w(l, k)$, $l \leq k$, will be used to denote the set of letters in the places from l to k. For prefixes with fixed length we have the same induced from S linear order. Formally, the empty word $w(0)$ is the unique element of the set $S(0, n)$. We introduce two sets of mappings (projections preserving the order) from S onto W. If the word belongs to W, these mappings will preserve its prefix. In the following we shall consider the set W fixed.

Definition 1. For $s \in S$, $z = \mathbf{First}(k, s)$ is the first member $z \in W$, such that $z(k) \geq s(k)$, if it exists. In any other case it is the first member of W.

Definition 2. For $s \in S$, $z = \mathbf{Last}(k, s)$ is the last member $z \in W$, such that $z(k) \leq s(k)$, if it exists. In any other case it is the last member of W.

If $w \in W$, $z = \mathbf{First}(k, w)$ is the first member of W and $z = \mathbf{Last}(k, w)$ is the last member of W with the same prefix $z(k) = w(k)$. So the element $w_0 = \mathbf{First}(0, w)$ is the very first in W and $l_0 = \mathbf{Last}(0, w)$ — the very last in the global linear order.

We shall introduce also a mapping $\mathbf{Increase}(k, w)$, which will be used to increase only the k-th letter of the word w. This mapping is not defined for all elements of W or S. Moreover, its result (when defined) is not obliged to belong to the set W.

Definition 3. We say that $\mathbf{Increase}(k, w)$ equals the smallest word $z \in S$, such that $w(k) < z(k)$, if this word exists. In any other case it is not defined.

Obviously, not for all elements of W this definition will lead to increasing of only and exactly the k-th letter.

3. MAIN RESULTS

First we shall prove some simple consequences of these natural definitions. Then an algorithm will be presented and a theorem about its completeness will be proved. Then a simple theorem which helps to estimate the effectivity of the

algorithm will be stated. It will concern the mean number of steps needed to produce the next to w word in W.

Lemma 1. *Both mappings* **First** *and* **Last** *are well defined and idempotent with fixed* k: $\mathbf{First}(k, \mathbf{First}(k, s)) = \mathbf{First}(k, s)$, $\forall s \in S$.

Proof. Denote by $l_0 = \mathbf{Last}(0, s)$. Then the set S may be split into two subsets: $S = S_1 + S_2 = \{s : s(k) \leq l_0(k)\} + \{s : s(k) > l_0(k)\}$.

When $s \in S_1$, the image $z = \mathbf{First}(k, s)$ exists, because it may be represented as intersection of non-empty subsets of W. It is unique because of the linear order. When $s \in S_2$, we have $w_0 = \mathbf{First}(k, s)$ according to the definition. The second statement is obvious because of the definition too. The same argument works for the mapping **Last**. □

Lemma 2. *If* $z = \mathbf{Last}(k, w)$, *then for each* $i \geq k$, $z = \mathbf{Last}(i, z)$. *If* $z = \mathbf{First}(k, w)$, *then for each* $i \geq k$, $z = \mathbf{First}(i, z)$.

Proof. Suppose that $z = \mathbf{Last}(k, w)$, but $z < \mathbf{Last}(i, z)$ for $i > k$. We have $w \leq z = \mathbf{Last}(k, z)$. Then $\mathbf{Last}(k, z) < \mathbf{Last}(i, z)$. However, the first k letters of these two words coincide — which is a contradiction. The same reasoning works for the dual statement. □

Let fix $w \in W$ and consider the set of equations $w = \mathbf{Last}(k, w)$. Note that $w = \mathbf{Last}(m, w)$ is true for each w. According to Lemma 2, there exists a minimal k for which this equality holds. For the last word in W we shall have $k = 0$. Again the same is true for the first word in W (in this case the mapping **First** should take place in the equations).

These considerations give us the possibility to construct the following algorithm for consecutive computing of the 'next' to w word in the set W:

```
function next(word)
1  k = n;
2       while word = Last(k, word);
3            k = k − 1;
4       end_while;
5            if k = 0 stop;
6       word := Increase(k, word);
7       word := First(k, word);
end
```

This algorithm needs some explanations. Lines 1–4 perform the search for the largest k for which $w \neq \mathbf{Last}(k, w)$. As the purpose of these lines is to find the integer k, it seems natural to combine them into a function: $k = \mathbf{Last\_Not\_Last}(w)$. Another reason to make this will be seen in the examples — in most cases it is easier to calculate the function **Last_Not_Last** than the mapping **Last**. The number k may be easily interpreted as the position of the first letter changing when moving from the given word to the next one in a lexicographically ordered set W. Line 5 prevents the use of the program after the last word $l_0 = \mathbf{Last}(0, w)$ has been reached. Line 6 increases the k-th letter of the word to the next letter allowed

(given the prefix $w(k-1)$). Line 7 simply uses the mapping **First**, but the prefix is now one letter longer — $w(k)$.

Usage of the function **Last_Not_Last** simplifies the algorithm:

```
function next(word)
  k = Last_Not_Last(word);
      If k = 0 stop;
  word := Increase(k, word));
  word := First(k, word);
end
```

Theorem 1. *Starting with* $w_0 = \mathbf{First}(0, w)$, *the above algorithm exhausts all elements of the set* W, *i.e.* $l_0 = \mathbf{Last}(0, w)$ *is reached.*

Proof. The first thing is to check the possibility to define and use the mapping **Increase** properly. Let $k > 0$. As the element w is not equal to $\mathbf{Last}(k, w)$, there exists a word $z \in W$, such that $z(k) = w(k)$ and $w < z$. Let us choose the next to w element $z \in W$. Suppose now that the k–th elements of z and w are equal. This means that $z(k) = w(k)$ and we have $w < z$, but $\mathbf{Last}(k, w) = w$. This is a contradiction. Thus, there exists a word $z \in W \subset S$ with greater k-th letter. Such a word exists in S. So in our algorithm we may use the function **Increase** when the proper $k > 0$ is found.

Now we shall show that no word $z \in W$ may be skipped by the algorithm. If $m = 1$, the statement follows from the definitions of **Increase** and **First**. If **Increase** does not produce a word from W, then this will be done by **First**.

The induction on m uses the fact that each part of the set W with a fixed prefix uses the same definitions of the functions **First**, **Last**, **Increase**. If for any fixed first letter the algorithm is exhaustive, it will be exhaustive for the whole set. □

Comment 1. The study of this simple proof shows that the definition of **Increase** may be made more complicated — not simply to increase the corresponding letter, but to choose it in such a way that the corresponding prefix "belongs" to W. The function **First** does not need to be defined for any word in S. For the index k achieved at the first step, there always exists a number in the alphabet put at the k-th place in the word, so that $\mathbf{First}(k, w) := \mathbf{First}(z, w)$ is well defined. This situation is effectively explored in some of the examples below.

Comment 2. It is easy to see that if one defines the mapping **Increase** to do nothing when $k = 0$, the proposed algorithm will loop infinitely across the set W starting from the beginning again and again.

Comment 3. The same argument may be used for the statement concerning the reverse order. The mapping **First** may be replaced by **Last**, the mapping **Increase** — by the correspondingly defined mapping *Decrease*. All the statements above will remain true except for the order — it will become the inverse order. There is one more formal duality in the lexicographical order – the interchanging of the letters. The most natural interchanging is to read the word backward. Then

the first letters of the word are changed while the last are kept fixed. We shall call such an order dual. With any set four different orderings into the set S can be defined. The definitions above are to be changed correspondingly for any such order. Any of these orderings may be useful to consider when an enumeration is performed.

Theorem 2. *Denote by W_k the set of all different prefixes $w(k)$ of words in W. We shall assume that $|W_0| = 1$ and $W_n = W$. Suppose that for each $k = 0, 1, 2, \ldots, n-1$, we have $|W_k|/|W_{k+1}| < q < 1$. Then according to the uniform distribution on W the expected number of steps to reach the address of change $\mathbf{E}(n-k)$ fulfills the inequality*

$$\mathbf{E}(n-k) < \frac{2}{1-q}.$$

Proof. The set W may be represented as a graph — a tree with totally $N+1$ vertices and N edges. Each vertex corresponds to a fixed prefix. Then the total way of our algorithm to generate all the members of the set is proportional to $2N$. Denote $r_k = |W_k|$. The expected number $\mathbf{E}(n-k)$ may be represented in the form

$$\mathbf{E}(n-k) \le \frac{2N}{|W|} \le \frac{2}{r_n}(r_0 + r_1 + r_2 + \cdots + r_n)$$

$$= 2\left(\frac{r_0 r_1 \ldots r_{n-1}}{r_1 r_2 \ldots r_n} + \frac{r_1 r_2 \ldots r_{n-1}}{r_2 r_3 \ldots r_n} + \cdots + 1\right) \le 2\frac{1-q^{n+1}}{1-q}.$$

Comment 4. It is clear that the assumption of the theorem may be weakened in a number of ways. For example, it is enough for k to run over the set $0, 1, \ldots, n-j$. Then the expectation will be limited by $j + 1/(1-q)$.

As we shall see in the next section, despite that the assumption is not fulfilled in many cases, the average number of steps remains finite. On the other hand, the set consisting of two words $\{1, 1, 1, \ldots, 1\}$ and $\{2, 1, 1, \ldots, 1\}$ will need an expected number of steps proportional to n. It is an open question, what, in general, happens to the expected number of steps when all the dualities mentioned in Comment 3 are explored.

4. EXAMPLES

In the next examples we shall construct the mappings **First**, **Last**, *Increase* and the function **Last_Not_Last** for different subsets of S. We shall try also to calculate the computational complexity of the algorithms. In fact, one needs only the distribution of k — "the place of first change" in lexicographically ordered words. It is clear that the proposed algorithm will be as effective as closer to n the expectation of this place is situated. For that purpose one has to calculate also the size of the corresponding data set and assume uniform probability on it. So, the mean effort for constructing the next element should represent the complexity of the embedding of the data set in the given order. It will be seen that the use of different embeddings is of primary interest.

The first two examples have been extensively studied in [4, 5]. Here they are mentioned only to show that the idea we use leads to natural and well-known algorithms.

4.1. PERMUTATIONS OF N ELEMENTS

In Table 1 a part of the set of all permutations of 5 elements is shown in a lexicographic order.

TABLE 1. Part of permutations of 5 elements

1 2 3 4 5	1 3 2 4 5	1 4 2 3 5	...
1 2 3 5 4	1 3 2 5 4	1 4 2 5 3	...
1 2 4 3 5	1 3 4 2 5	1 4 3 2 5	...
1 2 4 5 3	1 3 4 5 2	1 4 3 5 2	...
1 2 5 3 4	1 3 5 2 4	1 4 5 2 3	...
1 2 5 4 3	1 3 5 4 2	1 4 5 3 2	...

It is clear that the mapping **Last** simply orders all the elements of w after (and including) the k-th one in a decreasing order, while for the mapping **First** this order is increasing.

The function **Last_Not_Last** finds the smallest k such that after it all elements are in a decreasing order. Denote $j = n - k$. Then it is clear that j runs from 1 to n. For a given k, this function needs j subtractions and comparisons.

The mapping **Increase** is more complicated. It takes the next larger than $w(k, k)$ integer from the set of integers $w(k, n)$ and should replace it with $w(k, k)$. The last step **First** is equivalent to inversion of the sequence of the last j integers.

Theorem 2 can be applied to the set of permutations with k running up to $n-j$ and $q = 1/j!$. However, it might be interesting to calculate exactly the expected number of steps of the algorithm. This is done in [5, Section 5.1], in the terms of transpositions and comparisons. The expected number of integer calculations then is proportional to $(e - 1)$ and remains finite as $n \to \infty$.

4.2. SUBSETS OF M ELEMENTS OUT OF SET OF N

In Table 2 the set of all subsets of 4 elements, taken from the set of 6 elements, is shown in a lexicographic order. One calls the objects combinations of n elements of class 4. Here the letters are kept in an increasing order inside the word — they should not coincide.

TABLE 2. Subsets of 4 elements out of 6

1 2 3 4	1 3 4 5	2 3 4 6
1 2 3 5	1 3 4 6	2 3 5 6
1 2 3 6	1 3 5 6	2 4 5 6
1 2 4 5	1 4 5 6	3 4 5 6
1 2 4 6	2 3 4 5	

The mapping **Last** changes the last $m-k$ elements of w into the largest elements of N, while **First** sets these elements to the smaller ones following the k-th element of w. The function **Last_Not_Last** finds the largest k such that $w(k,k)$ is not equal to $n-(m-k)$. The mapping **Increase** simply adds 1 to the corresponding element of w. This is also well-known algorithm [5, Section 5.2.2].

The number of combinations of n elements class m is $\binom{n}{m}$. In a similar way, as in permutations, we find the number of calculations as a function of $j=m-k$ to be about $4j$. The distribution of j is also easy to construct:

$$p_j = \frac{\binom{n-m-1-j}{j}}{\binom{n}{m}}.$$

So, we come to even stronger result, namely, that with the growth of $n-m$ the expected number of calculations decreases.

Here the application of Theorem 2 is also possible, which yields $|W_k| = \binom{n-(m-k)}{k}$.

4.3. PARTITIONS OF AN INTEGER I

To generate the set W of all partitions of a given integer n into a sum of number integers is an easy problem for this algorithm. In Table 3 all partitions of 10 into the sum of up to 4 numbers are given (except the trivial 000 10). This presentation allows to split easily W into partitions of exactly 2, 3 and 4 non-zero numbers. These subsets follow consecutively. In the next examples other representations will be used.

TABLE 3. Partitions of 10 into up to 4 members

0 0 1 9	0 1 3 6	1 1 2 6
0 0 2 8	0 1 4 5	1 1 3 5
0 0 3 7	0 2 2 6	1 1 4 4
0 0 4 6	0 2 3 5	1 2 2 5
0 0 5 5	0 2 4 4	1 2 3 3
0 1 1 8	0 3 3 4	2 2 2 4
0 1 2 7	1 1 1 7	2 2 3 3

The mapping **Last** distributes the remaining portion of n into maximum equal portions among the remaining numbers after the k-th one. The mapping **First** states all these numbers to $w(k,k)$ and the remainder from m is added to the last number. The function **Last_Not_Last** finds the largest k such that $w(m,m) - w(k,k) \geq 2$. The mapping **Increase** simply increases by 1 the corresponding element of w. This algorithm is due to Hindenburg (see [1, Section 14.3]).

In order to apply the theorem, we have to calculate $w_k = |W_k|$. This is not an easy problem, however. Consider the unlimited case — all partitions of n will be fixed as words of length n with non decreasing elements.

One sees that the number w_k is a sum of partition numbers, subjected to two kinds of restrictions — concerning the maximum number of elements and the size of the largest element.

We have $w_0 = 1$ and $w_k = |w_{k-1}| + 1$ until $n - k \geq [n/2]$. Starting from $[n/2] + 1$ until $n - k = [n/3]$, we have $w_k = w_{k-1} + 2$ or $w_k = w_{k-1} + 3$. Here the choice depends on the remainder of the division on 3.

This example shows that in this case Theorem 2 is not applicable. Indeed, $w_{[n/2]+1}/w_{[n/2]} = 1 + 1/n$ and it tends to one as n increases. Nevertheless, we hope that the average number of steps is finite in this representation also. The exact statement remains an open problem.

4.4. PARTITIONS OF AN INTEGER II

Here we use the representation which follows from the formula

$$\sum_{i=1}^{n} in_i = n.$$

Here the numbers n_i represent the number of members of size i in a particular partition. The number of members in each partition is $\sum_{i=1}^{n} n_i$. It is clear how to convert one representation into another. Table 4 contains the set of all partitions of 10 into members less than 8, but in another lexicographical order according to the new presentation. Instead of the restriction on the number of members, we now pose a restriction on the maximal member of the partition.

Here we shall exploit the dual order and present the same partitions in an order with a fixed suffix. In this order it is easy to add the additional restriction on the maximum member of the partition, say, it is equal to 7: starting with $a_1 = N$ this new algorithm produces all partitions of 10 to members not bigger than 7.

TABLE 4. All partitions of 10 into numbers up to 7

10 0 0 0 0 0 0	4 0 2 0 0 0 0	0 0 2 1 0 0 0	4 0 0 0 0 1 0
8 1 0 0 0 0 0	2 1 2 0 0 0 0	2 0 0 2 0 0 0	2 1 0 0 0 1 0
6 2 0 0 0 0 0	0 2 2 0 0 0 0	0 1 0 2 0 0 0	0 2 0 0 0 1 0
4 3 0 0 0 0 0	1 0 3 0 0 0 0	5 0 0 0 1 0 0	1 0 1 0 0 1 0
2 4 0 0 0 0 0	6 0 0 1 0 0 0	3 1 0 0 1 0 0	0 0 0 1 0 1 0
0 5 0 0 0 0 0	4 1 0 1 0 0 0	1 2 0 0 1 0 0	3 0 0 0 0 0 1
7 0 1 0 0 0 0	2 2 0 1 0 0 0	2 0 1 0 1 0 0	1 1 0 0 0 0 1
5 1 1 0 0 0 0	0 3 0 1 0 0 0	0 1 1 0 1 0 0	0 0 1 0 0 0 1
3 2 1 0 0 0 0	3 0 1 1 0 0 0	1 0 0 1 1 0 0	
1 3 1 0 0 0 0	1 1 1 1 0 0 0	0 0 0 0 2 0 0	

The function **Last_Not_Last** finds first k from the beginning such that $k \leq l = \sum_{i=1}^{k-1} i.a_i$. Then a_k is increased by one. **First** nullifies all elements in the beginning and makes $a_1 = l - k$.

Let us try to estimate the performance of the algorithm. As usual $P(n) = |W|$ is the number of partitions of n. Denote by r_k the number of different suffixes in the words of the set W. It is clear that $r_{n-1} = P(n), r_0 = 1, r_1 = 2$.

Consider now the case: $3 \leq k \leq n-1$. It is clear that the set R_k may be mapped into the set R_{k+1}, so that the additional element (at place $m = n - k$) is zero. However, it is possible to make one more mapping of the same set R_k into elements of R_{k+1} with non-zero elements at the same place m. This can be done if every suffix is shifted left until the first non-zero element occupies the place m. The remaining portion of the suffix is filled with zeros. The only exception for this second mapping is the suffix consisting of zeros only. So we have the inequality

$$r_k/r_{k+1} \leq 1/2 + 1/r_{k+1} \leq 5/6.$$

According to Theorem 2 this means that the mean number of steps is finite and does not increase as $n \to \infty$. The exact value of this mean, as well as the exact distribution of the number of steps remain an open problem in this presentation also.

4.5. BELL POLYNOMIALS

For exact definitions see [3, Ch. I, p. 10]. This example is included merely to illustrate the use of the algorithm working in the reverse order. Consider the set of all vectors of natural numbers satisfying the following two equalities:

$$\sum_{i=1}^{\infty} k_i = m, \qquad \sum_{i=1}^{\infty} i k_i = n.$$

The summation above is assumed to be infinite for simplicity. It is clear that only the first $n - k + 1$ elements may be non-zero. This set is of some interest in many applications. In addition to computing Bell polynomials, it is used in the distributions of order k. Again we have partitions and the problem could be solved using the first representation of partitions in Section 4.3 and then screening partitions with number of members less than m. However, we shall give here an explicit solution.

In Table 5 the solutions for $n = 13$ and $m = 6$ are shown in reverse order. The reverse order is chosen because of the simplicity of the mapping **Last** in this case.

TABLE 5. Bell Polynomials $n = 13$ and $m = 6$

5 0 0 0 0 0 0 1	3 0 2 1 0 0 0 0
4 1 0 0 0 0 1 0	2 3 0 0 1 0 0 0
4 0 1 0 0 1 0 0	2 2 1 1 0 0 0 0
4 0 0 1 1 0 0 0	2 1 3 0 0 0 0 0
3 2 0 0 0 1 0 0	1 4 0 1 0 0 0 0
3 1 1 0 1 0 0 0	1 3 2 0 0 0 0 0
3 1 0 2 0 0 0 0	0 5 1 0 0 0 0 0

The reverse algorithm will be used. The mapping *Decrease* decreases by 1 the corresponding element of w. The mapping **Last** sets all elements with indexes

greater than k to zero, then $w(k+1,k+1) = m_k - 1$, and finally adds 1 to the number in position $k+n_k-(k+1)*m_k$. Here m_k and n_k are the values of the sums in the definition of Bell polynomials, taken over indexes greater than k. The function $First\_Not\_\mathbf{First}$ finds the largest k such that $w(k,k) > 0$ and $w(k,k) \leq max - 2$. Here max is the index of rightmost non-zero element.

In order to estimate the performance of the algorithm in this situation, we shall point out that the representation of partitions given in Section 4.3 is much more economical. Moreover, the additional restriction $m = \sum_{i=1}^{\infty} k_i$ in this case is in concordance with the presentation. It means that it makes no sense to use this second representation if one needs effectivity.

4.6. PARTITIONS WITH AN ADDITIONAL RESTRICTION [1]

Let us consider the algorithm for the following partition problem with the additional restriction:

$$\sum_{i=1}^{l} k_i = m, \qquad \sum_{i=1}^{l} {k_i}^2 = n.$$

By using the algorithm described in Section 4.3 and simply screening the second equation, an easy solution could be given of this problem. As an example, the results are presented in Table 6 for words of length 15, $m = 16$ and several values of n.

TABLE 6. $\sum_{i=1}^{15} k_i = 16, \sum_{i=1}^{15} {k_i}^2 = n$

n	
18	1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
20	0 1 1 1 1 1 1 1 1 1 1 1 1 2 2
22	0 0 1 1 1 1 1 1 1 1 1 1 2 2 2
	0 1 1 1 1 1 1 1 1 1 1 1 1 1 3
24	0 0 0 1 1 1 1 1 1 1 1 2 2 2 2
	0 0 1 1 1 1 1 1 1 1 1 1 1 2 3
26	0 0 0 0 1 1 1 1 1 1 2 2 2 2 2
	0 0 0 1 1 1 1 1 1 1 1 1 2 2 3
28	0 0 0 0 0 1 1 1 1 2 2 2 2 2 2
	0 0 0 0 1 1 1 1 1 1 1 2 2 2 3
	0 0 0 1 1 1 1 1 1 1 1 1 1 3 3
	0 0 1 1 1 1 1 1 1 1 1 1 1 1 4

It would be interesting to investigate the combinatorial properties of this set and to study the properties of the algorithm in this case.

[1] The author is grateful to Prof. G. Zbaganu who mentioned this problem at the 8-th Seminar on Statistical Data Analysis, Varna, 1992, and then helped to reformulate the algorithm.

Consider the set $F_n^{(m)}$ of words of fixed length n, consisting of zeroes and ones, and having the property that they do not contain m or more than m consecutive ones. (This example was proposed by P. Mateev.)

It may be easily proved that the cardinalities $f_n^{(m)}$ of $F_n^{(m)}$ satisfy the following recurrent relation:

$$f_n^{(m)} = \sum_{i=1}^{m} f_{n-i}^{(m)}. \tag{4.1}$$

When $m = 2$, these numbers form the well-known Fibonacci sequence. For arbitrary m and starting conditions $f_0^{(m)} = 0$ and $f_1^{(m)} = 1$, Gabai [2] called them Fibonacci numbers of order M. Philippou [6] calculated them as sums of multinomial coefficients. For the words of zeroes and ones, however, the starting conditions are $f_0^{(m)} = 1, f_1^{(m)} = 1$, as with the original Fibonacci numbers. For the particular case $m = 3$ we have $f_n^{(3)} = 1, 1, 2, 4, 7, 13, 24, \ldots; n = 0, 1, 2, \ldots$ In Table 7 all the $f_5^{(3)}$ zero-one words are presented.

TABLE 7. Fibonacci words for $n = 5$ and $m = 3$

0 0 0 0 0	0 1 0 0 1	1 0 0 1 1
0 0 0 0 1	0 1 0 1 0	1 0 1 0 0
0 0 0 1 0	0 1 0 1 1	1 0 1 0 1
0 0 0 1 1	0 1 1 0 0	1 0 1 1 0
0 0 1 0 0	0 1 1 0 1	1 1 0 0 0
0 0 1 0 1	1 0 0 0 0	1 1 0 0 1
0 0 1 1 0	1 0 0 0 1	1 1 0 1 0
0 1 0 0 0	1 0 0 1 0	1 1 0 1 1

The algorithm for generating such n-tuples is extremely simple. The function **Last_Not_Last** has to find the first zero preceded by less than $m-1$ ones, **Increase** puts one at this place and **First** nullifies all elements with greater indexes.

Here the mean value of the needed calculations is obviously proportional to the place $j = n - k$ of the zero to change. Denote this mean value by j_n. The above recurrence relation then leads to a new relation for the mean values. In order to obtain this relation, we shall use the proof of the recurrence formula (4.1). All n-words may be divided into M disjoint subsets $S_1, S_2, \ldots, S_m$ (we suppose that n is large enough). The l-th subset S_l has an arbitrary prefix and last numbers are fixed:

$$S_l = \{w \in F_n^{(m)} : (w_1, w_2, \ldots, w_{n-l}, 0, 1, 1, 1, \ldots, 1)\}.$$

These subsets cover the whole set $F_n^{(m)}$. The cardinalities of the sets are clearly $f_{n-l}^{(m)}$. In each subset the algorithm stops at the first 0, performing exactly l steps, or enters the prefix looking for the next available zero. In the first case the prefix

should end with exactly m numbers — a zero and $m-1$ ones. Its cardinality equals to $f^{(m)}_{n-l-m}$. So we come to the formula

$$j_n = \frac{\sum_{i=1}^{m}((j_{n-i}+i)(f^{(m)}_{n-i} - f^{(m)}_{n-i-m}) + if^{(m)}_{n-i-m})}{\sum_{i=1}^{m} f^{(m)}_{n-i}}$$

$$= \sum_{i=1}^{m} w_i(j_{n-i} - j_{n-m-i}) \quad + \sum_{i=1}^{m} iw_i.$$

The proportions $w_l = f^{(m)}_{n-l}/f^{(m)}_n$, $l = 1, 2, \ldots, m$, are fixed as $n \to \infty$. The sequence j_n then obviously converges to the finite number $\sum_{i=1}^{m} iw_i * \lim(f_{n+m}/f_n)$.

5. PERFORMANCE AND CONCLUSION

Both forms of the algorithm have quite different performances. For the first it is quadratic in j. One hardly expects a comparison of two words of length j to be made for shorter time. It may be expected a good performance from the second form when the function **Last_Not_Last**, as well as the mapping **First** depend linearly of j. In all examples above this was made possible.

In the case when the distribution of j has a finite mean, not depending of n, the asymptotic properties of the algorithm are extremely nice.

We do not know the distribution of j in the case of Bell polynomials and the performance of the presented algorithms in this case remains an open problem which would be interesting to be solved.

It is clear that building up programs in such a way, one can hardly expect that they will be fast without some additional efforts. However, in all cases above it turned out that only slight modifications were needed to make the programs work quite satisfactorily.

Another useful hint may be to try the other orders to change the mappings **First**, **Last** and **Increase**, correspondingly, and to see what will happen to the program. It may became shorter and faster.

Acknowledgements. The author is grateful to K. Manev, whose useful comments have improved the presentation. The work was partially supported by the Bulgarian Ministry of Education, Science and Technology under Grant No MM440/94.

REFERENCES

1. Andrews, G. E. The theory of partitions. Encyclopedia of Mathematics and its Applications, vol. 2, Addison-Wesley, Reading, Massachusetts, 1976.
2. Gabai, H. Generalized Fibonacci k-sequences. *Fibonacci Quarterly*, **8**(1), 1970, 31–38.

3. Kaufmann, A. Introduction á la Combinatorique en Vue des Applications. Dunod, Paris, 1968.
4. Lipski, W. Kombinatorics for Programists. Wydwnictwa Naukowo-Techniczne, Warszawa, 1982 (in Russian).
5. Reingold, E. M., J. Nivergeld, N. Deo. Combinatorial Algorithm. Theory and Practice. Prentice-Hall, Englewood Cliffs, 1977.
6. Philippou, A. N. A note on the Fibonacci sequence of order k and multinomial coefficients. *Fibonacci Quarterly*, **21**(2), 1983, 82–86.
7. Vandev, D. L. Alphabetical ordering — useful tool for creating algorithms. In: *Statistical Data Analysis — SDA92, Proceedings*, Seminar on Statistical Data Analysis, Varna, Bulgaria, September 1992, 121–126.

Received on September 23, 1996
Revised on November 5, 1997

Faculty of Mathematics and Informatics
"St. Kl. Ohridski" University of Sofia
5 blvd J. Bourchier
BG-1164 Sofia, Bulgaria
e-mail: vandev@fmi.uni-sofia.bg