A Loopless Generation of Bitstrings without p consecutive ones

Vincent Vajnovszki

LE2I – CNRS FRE 2309, Université de Bourgogne
B.P. 47 870 21078 DIJON-Cedex France
e-mail: vvajnov@u-bourgogne.fr

Abstract
Let $F_n^{(p)}$ be the set of all n-length bitstrings such that there are no p consecutive 1s. $F_n^{(p)}$ is counted with the pth order Fibonacci numbers and it may be regarded as the subsets of $\{1, 2, ..., n\}$ without p consecutive elements and bitstrings in $F_n^{(p)}$ code a particular class of trees or compositions of an integer. In this paper we give a Gray code for $F_n^{(p)}$ which can be implemented in a recursive generating algorithm, and finally in a loopless generating algorithm.
Key words: bitstrings, Fibonacci numbers, loopless generating algorithm, combinatorial Gray codes, subsets with a given restriction.

1 Introduction and motivation

In [6] Ehrlich formalized the notion of a *loopless* (or *loop-free*) generating algorithm where, after the initial sequence is generated, each succeeding sequence must be generated by at most a fixed number of operations, independent of the length of the generated sequences. A constant delay between outputs is particularly desirable in applications where the output of one computation serves as input to another.

There are many interesting loopless algorithms for generating combinatorial objects, for example: permutations and multiset permutations [5, 6, 15]; combinations [4, 6]; subsets [1]; integer partitions [7, 22]; trees [14, 16–18, 20, 24, 30]; well-formed parenthesis strings [21, 29, 31, 34]; and linear extensions of a poset [3].

In this paper we adopt a classic approach in order to obtain a loopless generating algorithm for the set $F_n^{(p)}$ of all n-length bitstrings without p consecutive 1s: we determine a Gray code for $F_n^{(p)}$ which can be implemented in an efficient recursive generating algorithm, then we demonstrate some 'regularities' in this Gray code which permit us to construct the loopless algorithm.

Unlike the Squire's Gray codes for *A-free strings* [28] (strings avoiding a fixed pattern) our Gray code lists strings in *graylex order* [4] (a generalization of lexicographic order). Also, in [33] Walsh presents a very general technique for producing loopless algorithms for Gray code lists satisfying: (1) all the strings with a given prefix form an interval of consecutive strings in the list (the list is *prefix partitioned*), and (2) not all the strings with the same first i letters have the same $i + 1$st letter. Our Gray code satisfies the first but not the last criteria, so our loopless generating algorithm is based on intrinsic combinatorial properties of generated objects.

Our motivation is two-fold: we add a new object to the catalogue of elementary combinatorial objects that can be generated by a loopless algorithm, and in contrast to the 'existential' approach of [19] we prove constructively that Fibonacci cubes contain Hamiltonian paths. Moreover, recently novel interconnection topologies are defined based on Gray codes on $F_n^{(p)}$ [11, 36] in reason of interesting values of their diameter, edge and node connectivity.

2 Preliminaries

In this section we define the main notions which occur through this paper: Gray code, operations over lists, binary Gray codes, order relations over bitstring sets and Fibonacci set.

A list $\mathcal{L}$ for a string set L is an ordered list of the elements of L. If the elements of $\mathcal{L}$ are in order such that the Hamming distance between two successive elements (i. e., the number of positions in which they differ) is bounded by a constant, then the list is called a Gray code list. For example, the Hamming distance between two successive bitstrings in the binary reflected Gray code list (see relation (1) below) is one and this distance is two in the Gray code list for the bitstrings which represent well-formed parenthesis strings [26, 34]. In the followings we adopt the definitions below.

- $first(\mathcal{L})$ is the first element in the list and $last(\mathcal{L})$ the last element on the list;
- $\overline{\mathcal{L}}$ is the list obtained by reversing $\mathcal{L}$, e. g., if $\mathcal{L} = l_1, l_2, ..., l_{n-1}, l_n$, then $\overline{\mathcal{L}} = l_n, l_{n-1}, \ldots, l_2, l_1$, and $first(\mathcal{L}) = last\left(\overline{\mathcal{L}}\right)$ and $first\left(\overline{\mathcal{L}}\right) = last(\mathcal{L})$;
- if α is an integer or a string, then α^n is the string which consists of n copies of α; and $\alpha \cdot \mathcal{L}$ is the list obtained by concatening α to each string of $\mathcal{L}$, e. g., if $\mathcal{L} = 01, 00, 10$ and $\alpha = 10$, then $\alpha \cdot \mathcal{L} = 1001, 1000, 1010$;
- if $\mathcal{L}'$ is a list, then $\mathcal{L} \circ \mathcal{L}'$ is the concatenation of the two lists, e. g., if $\mathcal{L} = 010, 000, 001$ and $\mathcal{L}' = 101, 100$, then $\mathcal{L} \circ \mathcal{L}' = 010, 000, 001, 101, 100$.

For the set B_n of all n-length bitstrings the binary reflected Gray code (BRGC for short) due to Frank Gray in 1953 [10] is defined by

$$\mathcal{B}_n = \begin{cases} \lambda & \text{if } n = 0, \\ 0 \cdot \mathcal{B}_{n-1} \circ 1 \cdot \overline{\mathcal{B}}_{n-1} & \text{if } n > 0 \end{cases} \tag{1}$$

where λ is the empty word, and the list $\mathcal{B}_n$ has $first(\mathcal{B}_n) = 0^n$ and $last(\mathcal{B}_n) = 10^{n-1}$.

The order in a list $\mathcal{L}$ may correspond to a natural order on the set of the elements of $\mathcal{L}$. For example, the order in the BRGC list is the grl-order (see the definition below), and the order in the binary tree integer sequences list is the lexicographic one [24].

Now you define two order relations on B_n denoted by $\prec^g$ and $\prec^l$, called *global reflected-lexicographical order* (*grl-order* for short), and *local reflected-lexicographical order* (*lrl-order* for short) respectively, and both are particular cases of *graylex order* [4]. The former is more natural than the latter and in Section 4 we give an lrl-ordered Gray code list for $F_n^{(p)}$. Notice that the grl-order is not a Gray code order for $F_n^{(p)}$.

Definition 1. Let $b = b_1 b_2 \cdots b_n$ and $b' = b'_1 b'_2 \cdots b'_n$ be bitstrings in B_n and i, $1 \le i \le n$, an index such that $b_i \ne b'_i$ but $b_j = b'_j$ for $1 \le j < i$. Then

1. $b \prec^g b'$ if and only if either
   (a) $\sum_{j=1}^{i-1} b_j$ is even and $b_i < b'_i$, or
   (b) $\sum_{j=1}^{i-1} b_j$ is odd and $b_i > b'_i$,
2. $b \prec^l b'$ if and only if either
   (a) $(i-1) + \sum_{j=1}^{i-1} b_j$ is even and $b_i < b'_i$, or
   (b) $(i-1) + \sum_{j=1}^{i-1} b_j$ is odd and $b_i > b'_i$.

We may construct a Gray code list for B_n with the bitstrings listed in *lrl-order* and not in grl-order as that given by the definition (1). Its definition is similar to (1), but we reverse the first list and not the last one

$$\mathcal{C}_n = \begin{cases} \lambda & \text{if } n = 0, \\ 0 \cdot \overline{\mathcal{C}}_{n-1} \circ 1 \cdot \mathcal{C}_{n-1} & \text{if } n > 0. \end{cases} \tag{2}$$

Here $first(\mathcal{C}_n) = 01^{n-1}$ and $last(\mathcal{C}_n) = 1^n$, and $\mathcal{C}_n$ may be obtained from $\mathcal{B}_n$ by replacing in $\mathcal{B}_n$ all zero bits in each string by a one bit and vice-versa, then reversing the obtained list.

Alternatively, we may define a Gray code list in terms of a Hamiltonian path over a set of strings. Let S be a string set, $k > 0$ an integer, and $G(S)$ be the graph with vertex set S, and edges connecting those vertices for which the Hamming distance is bounded by k. Finding a Gray code list $\mathcal{L}$ for S is equivalent to finding a Hamiltonian path in $G(S)$, and $\mathcal{L}$ lists the strings in S along this Hamiltonian path.

3 Fibonacci strings

Let $F_n^{(p)}$ be the set of all n-length bitstrings such that there are no p consecutive 1s, with $p \ge 2$. $F_n^{(p)}$ is called the pth *order* nth *Fibonacci set*, and the elements in $F_n^{(p)}$ are the pth *order n-length Fibonacci strings*. The set $F_n^{(p)}$ is defined recursively by

$$F_n^{(p)} = \begin{cases} \{\lambda\} & \text{if } \quad n = 0, \\ \{0,1\}^n & \text{if } 1 \le n < p, \\ 0 \cdot F_{n-1}^{(p)} \cup 10 \cdot F_{n-2}^{(p)} \cup \cdots \cup 1^{p-1}0 \cdot F_{n-p}^{(p)} & \text{if } \quad n \ge p \end{cases} \tag{3}$$

and [13, p. 287]

$$card(F_n^{(p)}) = f_{n+p}^{(p)} \tag{4}$$

where $f_n^{(p)}$ is the pth *order* nth *Fibonacci numbers* defined by [13, p. 269]

$$f_n^{(p)} = \begin{cases} 0 & \text{if } 0 \leq n < p-1, \\ 1 & \text{if } \quad n = p-1, \\ \sum_{j=n-p}^{n-1} f_j^{(p)} & \text{if } \quad n \geq p, \end{cases} \tag{5}$$

and the generating function for the sequence $\{f_n^{(p)}\}_{n\geq 0}$ is

$$\sum_{n\geq 0} f_n^{(p)} z^n = \frac{z^{p-1}}{1 - z^1 - z^2 - \ldots - z^p} = \frac{z^{p-1} - z^p}{1 - 2z + z^{p+1}}.$$

When $p = 2$ relation (5) gives the usual Fibonacci numbers, and $f_n^{(3)}$ and $f_n^{(4)}$ are called Tribonacci and Tetranacci numbers respectively [27, pp. 406, 423]. See Table 1 for the set $F_5^{(2)}$. It is easy to generate the Fibonacci set in lexicographical order, see [23] for an iterative constant amortized time generating algorithm for $F_n^{(2)}$.

A bitstring in $F_n^{(p)}$, $p \geq 2$, may be regarded as a subset of $[n] = \{1, 2, \ldots, n\}$ without p consecutive elements using the customary convention: an element is within the subset if and only if the bit in the Fibonacci string with its index is one. See [9, pp. 292 and 321] for two combinatorial interpretations of $F_n^{(2)}$ or Appendix 2 for $F_n^{(p)}$, $p \geq 2$.

4 Gray code for Fibonacci strings

Let $\mathcal{F}_n^{(p)}$ be the bitstring list defined by

$$\mathcal{F}_n^{(p)} = \begin{cases} \mathcal{C}_n & \text{if } 0 \leq n < p, \\ 0 \cdot \overline{\mathcal{F}}_{n-1}^{(p)} \circ 10 \cdot \overline{\mathcal{F}}_{n-2}^{(p)} \circ \cdots \circ 1^{p-1}0 \cdot \overline{\mathcal{F}}_{n-p}^{(p)} & \text{if } \quad n \geq p, \end{cases} \tag{6}$$

with $\mathcal{C}_n$ defined by relation (2). This is the expression in terms of list of relation (3) and it is not difficult to prove that $\mathcal{F}_n^{(p)}$ is a Gray code list for $F_n^{(p)}$ with the Hamming distance between two consecutive elements equal to one, and $\mathcal{F}_n^{(p)}$ lists the elements of $F_n^{(p)}$ in lrl-order. Remark that the list of $F_n^{(p)}$ in grl-order is not a Gray code; an example with small value of n proofs it. In terms of subsets, $\mathcal{F}_n^{(p)}$ lists the subsets of $[n]$ without p consecutive elements such that successive subsets differ by the deletion of an old element or the addition of a new one. See Table 1 and 2 for the lists $\mathcal{F}_5^{(2)}$ and $\mathcal{F}_4^{(3)}$, and their associated subsets.

Let $\chi^{(p)}$ be the $(p+1)$-length bitstring $1^{p-1}00$, and for $0 \leq j \leq p+1$, $\chi_j^{(p)}$ be its j-length prefix, i. e., $\chi_0^{(p)} = \lambda$, $\chi_1^{(p)} = 1$, $\chi_2^{(p)} = 11, \ldots, \chi_{p+1}^{(p)} = 1^{p-1}00$. The basic properties of the list $\mathcal{F}_n^{(p)}$ are embodied in the following Lemma.

Lemma 2. *1.* $first\left(\mathcal{F}_n^{(p)}\right) = 0\left(\chi^{(p)}\right)^{\lfloor\frac{n-1}{p+1}\rfloor}\chi^{(p)}_{(n-1)\bmod (p+1)}$*,*

2. $last\left(\mathcal{F}_n^{(p)}\right) = \left(\chi^{(p)}\right)^{\lfloor\frac{n-1}{p+1}\rfloor}\chi^{(p)}_{(n-1)\bmod (p+1)+1}$*, with* $\lfloor x \rfloor$ *the largest integer smaller then* x*,*

3. Two successive bitstrings in $\mathcal{F}_n^{(p)}$ *differ in exactly one position,*

4. $\mathcal{F}_n^{(p)}$ *lists the elements of* $F_n^{(p)}$ *in lrl-order.* □

An alternative way to define recursively the list $\mathcal{F}_n^{(p)}$ is

$$\mathcal{F}_n^{(p)} = \begin{cases} \lambda & \text{if } n = 0, \\ 0, 1 & \text{if } n = 1, \\ 0 \cdot \overline{\mathcal{F}}_{n-1}^{(p)} \circ 10 \cdot \overline{\mathcal{F}}_{n-2}^{(p)} \circ \cdots \circ 1^{p-1}0 \cdot \overline{\mathcal{F}}_{n-p}^{(p)} & \text{if } n > 1, \end{cases} \tag{7}$$

with the following conventions for negative values of n: (1) $\alpha \cdot \mathcal{F}_{-1}^{(p)}$ is the singleton list formed by the string α after the deletion of its last item, and (2) $\alpha \cdot \mathcal{F}_n^{(p)}$ is the empty list if $n < -1$.

The following recursive generating procedure is the algorithmic expression of relation (7). The array b, n and the order $p > 1$ are global variables, and the main call is $fib(n, up)$. For a simpler expression of the algorithm we admit that the fib procedure could write in the string b even after the index n, and the call of fib with values of n less then minus one has not any effect. A Java applet generating $\mathcal{F}_n^{(p)}$ is available at my web site [32].

```
procedure $fib(k, dir)$
if $k = 0$ or $k = -1$
then $PrintString$;
else if $k = 1$
    then if $dir = up$
         then $b[n] := 0$; $PrintString$;
              $b[n] := 1$; $PrintString$;
         else $b[n] := 1$; $PrintString$;
              $b[n] := 0$; $PrintString$;
         endif
    else if $k > 1$
         then if $dir = up$
              then for $j := 0$ to $p - 1$ do
                        for $u := 1$ to $min(j, k)$ do
                            $b[n - k + u] := 1$;
                        enddo
                        $b[n - k + j + 1] := 0$;
                        $fib(k - j - 1, down)$;
                   enddo
              else for $j := p - 1$ downto $0$ do
                        for $u := 1$ to $min(j, k)$ do
                            $b[n - k + u] := 1$;
                        enddo
                        $b[n-k+j+1] := 0$;
                        $fib(k-j-1, up)$;
                    enddo
                endif
            endif
        endif
endif
```

In the algorithm above each call of the generating procedure requires p recursive calls, thus *a priori* it has $O(p)$ time complexity which is not constant amortized time, unless p is a constant; an experimental study comfort this hypothesis. This algorithm lies on relation (7) and may be transformed in a constant amortized time by a simple transform of this relation as given below.

The case $n > 1$ in (7) may be expressed as

$$\begin{aligned}
\mathcal{F}_n^{(p)} &= 0 \cdot \overline{\mathcal{F}}_{n-1}^{(p)} \circ 10 \cdot \overline{\mathcal{F}}_{n-2}^{(p)} \circ \cdots \circ 1^{p-1}0 \cdot \overline{\mathcal{F}}_{n-p}^{(p)} \\
&= 0 \cdot \overline{\mathcal{F}}_{n-1}^{(p)} \circ 1 \cdot \underbrace{\left(0 \cdot \overline{\mathcal{F}}_{n-2}^{(p)} \circ \cdots \circ 1 \cdot \underbrace{\left(0 \cdot \overline{\mathcal{F}}_{n-p-1}^{(p)} \circ 10 \cdot \overline{\mathcal{F}}_{n-p}^{(p)} \right)}_{\mathcal{E}_{n,2}^{(p)}} \right)}_{\mathcal{E}_{n,p-1}^{(p)}} \\
&= \mathcal{E}_{n,p}^{(p)},
\end{aligned}$$

where $\mathcal{E}_{n,k}^{(p)}$ is defined recursively by

$$\mathcal{E}_{n,k}^{(p)} = \begin{cases} 0 \cdot \overline{\mathcal{F}}_{n-p-1}^{(p)} \circ 10 \cdot \overline{\mathcal{F}}_{n-p}^{(p)} & \text{if } \quad k = 2, \\ 0 \cdot \overline{\mathcal{F}}_{n-p-1+k}^{(p)} \circ 1 \cdot \mathcal{E}_{n,k-1}^{(p)} & \text{if } 2 < k \le p, \end{cases} \tag{8}$$

and with relation (7) we have

$$\mathcal{E}_{n,k}^{(p)} = \begin{cases} \lambda & \text{if } \quad n = 0 \text{ and } k = p, \\ 0, 1 & \text{if } \quad n = 1 \text{ and } k = p, \\ 0 \cdot \overline{\mathcal{E}}_{n-p-1,p}^{(p)} \circ 10 \cdot \overline{\mathcal{E}}_{n-p,p}^{(p)} & \text{if } \quad n > 1 \text{ and } k = 2, \\ 0 \cdot \overline{\mathcal{E}}_{n-p-1+k,p}^{(p)} \circ 1 \cdot \mathcal{E}_{n,k-1}^{(p)} & \text{if } n > 1 \text{ and } 2 < k \le p, \end{cases} \tag{9}$$

and $\mathcal{F}_n^{(p)} = \mathcal{E}_{n,p}^{(p)}$. In other words, $\mathcal{E}_{n,k}^{(p)}$, $k \le p$, is the $(n-p+k)$-length bitstring list obtained from $\mathcal{F}_n^{(p)}$ after the deletion of the prefix 1^{p-k} in all bitstring belonging in $\mathcal{F}_n^{(p)}$ with this prefix.

Procedure fib_e below is the implementation of relation (9); in this case the recursive generating procedure has no loops and it has a constant amortized time.

Indeed, it satisfies the Ruskey and van Baronaigien's [25] *'CAT'* (like *Constant Amortized Time*) principle listed below.

1. Every call results in the output of at least one object,
2. Excluding the computation done by recursive calls, the amount of computation of any call is proportional to the degree of a call,
3. The number of calls of degree one is linear in the number of generated objects,

where the *degree* of a call is the number of immediate recursive calls initiated by the current call. The call $fib\_e(n,p,up)$ produces the list $\mathcal{F}_n^{(p)} = \mathcal{E}_{n,p}^{(p)}$ and, as in the case of the procedure fib, variables n, p and b are global.

```
procedure $fib\_e(j,k,dir)$
if $j \le 0$
then $PrintString$;
else if $j = 1$
    then if $dir = up$
        then $b[n] := 0$; $PrintString$;
             $b[n] := 1$; $PrintString$;
        else $b[n] := 1$; $PrintString$;
             $b[n] := 0$; $PrintString$;
        endif
    else if $k = 2$
        then if $dir = up$
            then $b[n-j+1] := 0$; $fib\_e(j-1,p,down)$;
                 $b[n-j+1] := 1$; $b[n-j+2] := 0$; $fib\_e(j-2,p,down)$;
            else $b[n-j+1] := 1$; $b[n-j+2] := 0$; $fib\_e(j-2,p,up)$;
                 $b[n-j+1] := 0$; $fib\_e(j-1,p,up)$;
            endif
        else if $dir = up$
            then $b[n-j+1] := 0$; $fib\_e(j-1,p,down)$;
                 $b[n-j+1] := 1$; $fib\_e(j-1,k-1,up)$;
            else $b[n-j+1] := 1$; $fib\_e(j-1,k-1,down)$;
                 $b[n-j+1] := 0$; $fib\_e(j-1,p,up)$;
            endif
        endif
    endif
endif
end.
```

5 Loopless generating algorithm for $F_n^{(p)}$

In order to make the recursive generating algorithm loopless we need additional information about the generated bitstrings. In a bitstring b in $F_n^{(p)}$ we say that a one bit b_i is free if either (i) $i = n$, or (ii) $i = n-1$ and $b_n = 0$, or (iii) $i < n-1$, $b_{i+1} = 0$ and $b_{i+2} = 1$; a zero bit b_i is free if its right neighbor – if it exists –

is zero and if $i > 1$ then b_{i-1} is not the rightmost one bit in a contiguous 1s sequence of length $p-1$. In Tables 1.a and 1.b free bits in bitstrings belonging to $\mathcal{F}_5^{(2)}$ and $\mathcal{F}_4^{(3)}$, respectively, are in bold-face. Note that bitstrings $first(\mathcal{F}_n^{(p)})$ and $last(\mathcal{F}_n^{(p)})$ have only one free bit and any other bitstring in $\mathcal{F}_n^{(p)}$ has at least two free bits.

Table 1. The bitstrings in $\mathcal{F}_5^{(2)}$ and $\mathcal{F}_5^{(2)}$ in lrl-order and their corresponding subsets. Free bits are in bold-face and changed bits – the last or the last-but-one free bits – are underlined.

(a) The list $\mathcal{F}_5^{(2)}$

rank in lrl-order	bitstring	subset
1	0 1 0 0 $\underline{\mathbf{1}}$	{2,5}
2	0 1 0 $\underline{\mathbf{0}}$ **0**	{2}
3	0 $\underline{\mathbf{1}}$ 0 **1** 0	{2,4}
4	**0** **0** 0 $\underline{\mathbf{1}}$ 0	{4}
5	**0** **0** **0** **0** $\underline{\mathbf{0}}$	$\emptyset$
6	**0** **0** $\underline{\mathbf{0}}$ 0 **1**	{5}
7	**0** 0 **1** 0 $\underline{\mathbf{1}}$	{3,5}
8	$\underline{\mathbf{0}}$ 0 1 0 **0**	{3}
9	**1** 0 1 0 $\underline{\mathbf{0}}$	{1,3}
10	**1** 0 $\underline{\mathbf{1}}$ 0 **1**	{1,3,5}
11	1 0 **0** 0 $\underline{\mathbf{1}}$	{1,5}
12	1 0 **0** $\underline{\mathbf{0}}$ **0**	{1}
13	1 0 0 **1** 0	{1,4}

(b) The list $\mathcal{F}_4^{(3)}$

rank in lrl-order	bitstrings	subset	precedence array
1	0 1 $\underline{\mathbf{1}}$ 0	{2,3}	0120
2	0 1 **0** $\underline{\mathbf{0}}$	{2}	0100
3	0 $\underline{\mathbf{1}}$ 0 **1**	{2,4}	0101
4	**0** **0** 0 $\underline{\mathbf{1}}$	{4}	0001
5	**0** **0** $\underline{\mathbf{0}}$ **0**	$\emptyset$	0000
6	**0** 0 **1** $\underline{\mathbf{0}}$	{3}	0010
7	$\underline{\mathbf{0}}$ 0 1 **1**	{3,4}	0012
8	**1** 0 1 $\underline{\mathbf{1}}$	{1,3,4}	1012
9	**1** 0 $\underline{\mathbf{1}}$ **0**	{1,3}	1010
10	1 **0** **0** $\underline{\mathbf{0}}$	{1}	1000
11	1 $\underline{\mathbf{0}}$ 0 **1**	{1,4}	1001
12	1 **1** 0 $\underline{\mathbf{1}}$	{1,2,4}	1201
13	1 1 0 **0**	{1,2}	1200

Lemma 3 below yields a loopless generating algorithm. Let $H_n^{(p)}$ be the graph with vertex set $F_n^{(p)}$, with $first(\mathcal{F}_n^{(p)})$ and $last(\mathcal{F}_n^{(p)})$ connected to the bitstrings obtained by changing their only free bits, and any other bitstring b connected to two bitstrings – one obtained by changing the last free bit and the other by changing the last-but-one free bit in b.

Lemma 3. *Let $G(F_n^{(p)})$ be the graph with vertex set $F_n^{(p)}$, and edges connecting those vertices with Hamming distance equal to one (i. e., they differ in a single position). Then $H_n^{(p)}$ is a Hamiltonian path in $G(F_n^{(p)})$ and the list $\mathcal{F}_n^{(p)}$ defined by (6) is obtained covering the path $H_n^{(p)}$.*$\square$

The following loopless algorithm is a direct implementation of Lemma 3 and computes the successor of a bitstring b in the list $\mathcal{F}_n^{(p)}$. It employs the n-length arrays *pa* and *stack*. Array *pa* is called a precedence array and $pa[i]$ is the length of the contiguous sequence of 1s ending in position i if $b[i] = 1$, and 0 otherwise; array *stack* stores, in increasing order, the indices of the free bits in b. Integer *top* is the number of free bits in b and *ch* is the index of the bit in b which will be changed in order to obtain the next pth order Fibonacci string. Before the first call of *next*, b is initialized with $first(\mathcal{F}_n^{(p)})$ (see Lemma 1); *pa*, its precedence array, according to $pa[i] = 0$ if $b[i] = 0$, and $pa[i] = pa[i-1] + 1$ if $b[i] = 1$ and $i > 1$; and initially $ch = n - 1$ if n is a multiple of $p + 1$ and $ch = n$ otherwise, $stack[1] = ch$, and $top = 1$. After the initialization step the call of *next*, until $top = 1$, gives the list $\mathcal{F}_n^{(p)}$ with no loop statement between successive bitstrings.

```
procedure next
$b[ch] := 1 - b[ch]$;
update $top$ and arrays $stack$ and $pa$;
if $top \neq 1$
then  if $ch = stack[top]$
        then $ch := stack[top-1]$;
        else $ch := stack[top]$;
        endif
endif
end.
```

The difficulties hold in the update of the array *stack*, the list in increasing order of the free bits in b. The change of the bit index *ch* may induce the change of the status (becomes a free bit if it is not a one, or vice-versa) of bits index $ch-2$, $ch-1$ and $ch+1$. For example, to transform the first bitstring in $\mathcal{F}_5^{(2)}$ into its successor, bit index 4 becomes free (see Table 1); or to transform bitstring of rank 12, bits index 3 and 5 become not-free. In this case, the indices of the new free bits are pushed (in increasing order) in *stack* and the indices of the bits which are not more free are popped out of *stack*. This is possible with no loop statement since *ch* is the last or the last-but-one element in *stack*.

For $p = 2$ this algorithm generates the list $\mathcal{F}_n^{(2)}$. In this case the array *pa* is not required since $pa[i] = b[i]$ for all $1 \leq i \leq n$, and the algorithm can therefore be expressed in a simpler form, given in the Appendix 1.

6 Conclusions

We have presented a Gray code for the set of all n-length bitstrings without p successive ones, and efficient algorithms for generating these bitstrings. The algorithms are constant amortized time (constant on average) or loopless (constant in the worst case). These results add a new object to the list of combinatorial objects which may be efficiently generated and provide insight into the combinatorics of bitstrings with a given restriction.

This paper also shows how, for a given set, a counting recursive relation can be derived in a Gray code definition and easily expressed in a recursive generating algorithm and finally in a loopless generating algorithm.

Appendix 1

Procedure *next* which computes the successor of a bitstring b in the list $\mathcal{F}_n = \mathcal{F}_n^2$. For a simpler expression of the algorithm we consider that $b[0] = 0$.

```
procedure next
if $b[ch] = 1$
then $v := ch - 1$; $b[ch] := 0$
else  $v := ch - 2$; $b[ch] := 1$
endif
if $ch \geq 2$
then if $b[ch-2] \neq b[ch]$
        then {$ch - 1$ or $ch - 2$ is popped out of $stack$}
                if $ch \neq stack[top]$
                then $stack[top-2] := ch$;
                endif
                $stack[top-1] := stack[top]$; $top := top - 1$;
        else  {$ch - 1$ or $ch - 2$ is pushed in $stack$}
                $top := top + 1$; $stack[top] := stack[top-1]$;
                if $ch = stack[top-1]$
                then $stack[top-1] := v$
                else  $stack[top-1] := ch$; $stack[top-2] := v$;
                endif
        endif
endif
if $ch = n - 1$
then if $b[ch] = 0$
        then {$n$ is pushed in $stack$}
                $top := top + 1$; $stack[top] := n$;
        else  {$n$ is popped out of $stack$}
                $top := top - 1$;
        endif
endif
```

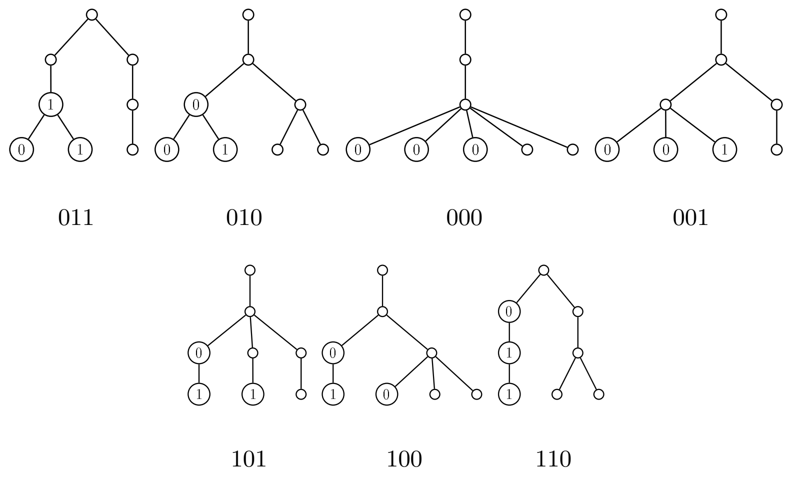
```
if $top \neq 1$
then if $ch = stack[top]$
        then $ch := stack[top-1]$;
        else $ch := stack[top]$;
        endif
endif
end.
```

Appendix 2

Geometrical interpretation

A (p,r)–tree, $1 < p < r$, is a tree with p levels and r nodes and all branches reaching to the level furthest from the root. The set of all (p,r)–trees is in a one-to-one correspondence with the bitstring set $F^{(p-1)}_{r-(p+1)}$, and we show it constructively. Let T be a (p,r)–tree, $1 < p < r$. We label nodes which have right sibling by 0 and all others by 1. Reading in post-order (recursively the subtrees left to right, then the root) the labels we obtain a r-length bitstring; since its p+1-length suffix is always 01^p we denote it by $\alpha 01^p$ and α is a Fibonacci string in $F^{(p-1)}_{r-(p+1)}$. Conversely, all bitstring in $F^{(p-1)}_{r-(p+1)}$ represents a unique (p,r)–tree. See Figure 1 for the seven (4,8)-trees and the Fibonacci bitstrings in $F^{(3)}_3$ associated with them. In this context, adding a i-length branch on the left side of a (p,r)–tree, $i < p$, corresponds to appending the prefix $1^{i-1}0$ to the bitstring which represents the tree.

Fig. 1. The seven $(4,8)$-trees and they bitstring representation.



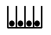
Combinatorial interpretation

Let $\mathcal{I}_n^{(p)}$ be the set of compositions of the integer n whose parts are only allowed to be taken from $\{1, 2, \ldots, p\}$ [8, pp. 15]. A composition in $\mathcal{I}_n^{(p)}$ is an integer sequence $n_1 n_2 \ldots n_k$ with $\sum_{i=1}^{k} n_i = n$ and $1 \le n_i \le p$, for $1 \le i \le k$. The transformation

$$n_1 n_2 \ldots n_k \rightsquigarrow 1^{n_1-1} 0 1^{n_2-1} 0 \ldots 1^{n_k-1}$$

is a one-to-one correspondence between $\mathcal{I}_n^{(p)}$ and $\mathcal{F}_{n-1}^{(p)}$.

Table 2. The set $\mathcal{I}_4^{(3)}$ of compositions of 4 with parts from $\{1, 2, 3\}$.

unlabeled balls into labeled boxes	sequence	bitstring
	13	011
	121	010
	1111	000
	112	001
	22	101
	211	100
	31	110

References

1. J.R. Bitner, G. Ehrlich and E.M. Reingold, Efficient generation of the binary reflected Gray code and its applications, *Commun. ACM* **19** (1976), 517–521.
2. G. Brightwell and P. Winkler, Counting linear extensions, *Order* **8** (1991), 225–242.
3. E.R. Canfield and S.G. Williamson, A loop-free algorithm for generating linear extensions of poset, *Order* **12** (1995), 57–75.
4. P.J. Chase, Combination generation and Graylex ordering, *Congr. Numer.* **69** (1989), 215–242.
5. N. Dershowitz, A simplified loop-free algorithm for generating permutations, *BIT* **15** (1975), 158–164.
6. G. Ehrlich, Loopless algorithms for generating permutations, combinations, and other combinatorial objects, *J. ACM* **20** (1973), 500–513.
7. T.I. Fenner and G. Loizou, A binary tree representation and related algorithms for generating integer partitions, *Comput. J.* **23** (1980), 332–337.

8. P. Flajolet and R. Sedgewick, Counting and Generating Functions, *Res. Rep. no. 1888, INRIA*, 1993. `http://pauillac.inria.fr/algo/flajolet/Publications/books.html`
9. R.L. Graham, D.E. Knuth and O. Patashnik, *Concrete Mathematics*, Second Edition, Reading, Massachusetts: Addison-Wesley, 1994.
10. F. Gray, *Pulse Code Communication*, U. S. Patent 2632058 (1953).
11. W-J. Hsu, Fibonacci cubes – a new interconnection topology, *IEEE Transactions on Parallel and Distributed Systems* **4**(1) (1993), 3–12.
12. J.T. Joichi, D.E. White and S.G. Williamson, Combinatorial Gray codes, *SIAM J. Comput.* **9**(1) (1980), 130–141.
13. D.E. Knuth, *The Art of Computer Programming. Vol. 3 Sorting and Searching*, Addison-Wesley, 1966.
14. J.F. Korsh, Loopless generation of k-ary tree sequences, *Information Processing Letters* **52** (1994), 243–147.
15. J.F. Korsh and S. Lipschutz, Generating multiset permutations in constant time, *J. Algorithms* **25** (1997), 321–335.
16. J.F. Korsh and S. Lipschutz, Shifts and loopless generation of k-ary trees, *Information Processing Letters* **65**(5) (1998), 235–240.
17. J.F. Korsh and P. LaFollette, Loopless generation of Gray codes for k-ary trees, *Information Processing Letters* **70**(1) (1999), 7–11.
18. J.F. Korsh and P. LaFollette, Multiset permutations and loopless generation of ordered trees with specified degree sequences, *J. Algorithms* **34**(2) (2000), 309–336.
19. J. Liu, W-J. Hsu and M.J. Chung, Generalized Fibonacci cubes are mostly Hamiltonian, *Journal of Graph Theory* **18**(8) (1994), 817–829.
20. J.M. Lucas, D. Roelants van Baronaigien and F. Ruskey, On rotations and the generation of binary trees, *J. Algorithms* **15**(1993), 343–366.
21. K. Mikawa and T. Takaoka, Generation of parenthesis strings by transpositions, in *Proc. CATS'97*, Sydney, Australia, February 3–4, 1997.
22. A. Nijenhuis and H.S. Wilf, *Combinatorial Algorithms*, Academic Press, 1975.
23. J.M. Pallo, On the listing and random generation of hybrid binary trees, *Intern. J. Comput. Math.* **50** (1994), 135–145.
24. D. Roelants van Baronaigien, A loopless algorithm for generating binary tree sequences, *Information Processing Letters* **39** (1991), 189–194.
25. D. Roelants van Baronaigien and F. Ruskey, Efficient generation of subsets with a given sum, *JCMCC* **14** (1993), 87–96.
26. F. Ruskey and A. Proskurowski, Generating binary trees by transpositions, *J. Algorithms* **11** (1990), 68–84.
27. N.J.A. Sloane, *A Handbook of Integer Sequences*, Academic Press, 1973.
28. M. Squire, Gray codes for A-free strings, *Electronic J. Combinatorics*, **3**(1996), paper R17.
29. V. Vajnovszki, Loopless generation of well-formed parenthesis strings, *Research Report Department IEM, University of Burgundy*, September 1997.
30. V. Vajnovszki, On the loopless generation of binary tree sequences, *Information Processing Letters* **68**(1998) 113–117.
31. V. Vajnovszki, Generating a Gray Code for P-sequences, to appear in *International Journal of Mathematical Algorithms.*
32. *http://www.u-bourgogne.fr/v.vincent/*
33. T.R. Walsh, A simple sequencing and ranking method that works on almost all Gray codes, *Res. Rep. no. 243, Department of Mathematics and Computer Science, University of Quebec at Montreal*, April 1995.

34. T.R. Walsh, Generation of well-formed parenthesis strings in constant worst-case time, *Journal of Algorithms* **29**(1) (1998), 651–673.
35. H.S. Wilf, *Combinatorial algorithms: An update*, SIAM, CBNS 55, 1989.
36. J. Wu, Extended Fibonacci Cubes, *IEEE Transactions on Parallel and Distributed Systems* **8**(12)(1997), 1203–1210.