ISSN 0032-9460, Problems of Information Transmission, 2010, Vol. 46, No. 4, pp. 369–378. © Pleiades Publishing, Inc., 2010.
Original Russian Text © Yu.S. Medvedeva, B.Ya. Ryabko, 2010, published in Problemy Peredachi Informatsii, 2010, Vol. 46, No. 4, pp. 130–139.

SOURCE CODING

Fast Enumeration Algorithm for Words with Given Constraints on Run Lengths of Ones[1]

Yu. S. Medvedeva[a] and B. Ya. Ryabko[b]

[a] *Siberian State University of Telecommunication and Information Science*
brainwashed@yandex.ru
[b] *Siberian State University of Telecommunication and Information Science*
Institute of Computational Technologies, Siberian Branch
of the Russian Academy of Sciences, Novosibirsk
boris@ryabko.net

Received May 4, 2008; in final form, June 1, 2010

Abstract—We propose an algorithm for enumeration and denumeration of words with given constraints on run lengths of ones ($dklr$-sequences). For large n, operation time of the algorithm (per symbol of a sequence) is at most $O(\log^3 n \log\log n)$, where n is the length of enumerated words, whereas for the best known methods it is at least cn, $c > 0$.

DOI: 10.1134/S0032946010040095

1. INTRODUCTION

In many telecommunication and data storage systems there exist constraints on the length of sequences (runs) of consecutive identical symbols. In this connection, the problem arises of encoding and decoding (enumeration and denumeration) of run-length-limited words. For instance, one of such constraints is forbidding two consecutive ones in a word.

A sequence of zeros and ones is said to be a dk-sequence if the length of any run of ones between two zeros is not less than d and not greater than k. As an example, here is a list of all sequences of length $n = 5$ with the constrains $d = 1$ and $k = 2$: (01010), (01011), (01101), (10101), (10110), (11010), and (11011).

If, besides these constraints, the run length of ones from the beginning of a sequence to the first zero is at most l, and the run length of ones from the last zero to the end of a sequence is at most r, such a sequence is referred to as a $dklr$-sequence. (Note that often constraints are imposed on run lengths of zeros; however, it is more convenient for us to use the above definition in what follows.)

By enumerative encoding (or enumeration), we mean finding the number of a word in the set of $dklr$-sequences of a given length; in turn, reconstructing a word given its number is referred to as denumeration. (Elements of the set are assumed to be ordered in a certain way, and numbers of elements are defined with respect to this order.)

The enumerative encoding problem was first considered in works by Kautz [2], where a method based on Fibonacci numbers was proposed. Table 1 presents lexicographically ordered words of length $n = 4$ with the following constraints on the run lengths of ones: $d = 0$ and $k = 1$ (i.e., these are words with no two consecutive ones; their total number is eight).

The idea of the Kautz enumeration method is as follows: sequences of so-called position weights (which are integers assigned to digit positions of enumerated words) are found. In our example, the

[1] Supported in part by the Russian Foundation for Basic Research, project no. 09-07-00005.
Main results of the paper were announced in [1].

Table 1. Words of length $n = 4$ with constraints $d = 0$, $k = 1$

Number	Word
0	0000
1	0001
2	0010
3	0100
4	0101
5	1000
6	1001
7	1010

position weights coincide with Fibonacci numbers and are, respectively, 1, 2, 3, and 5. The number of a word is obtained by summing weights of its nonzero positions, which are numbered from right to left. Thus, the number of 0101 is obtained as follows: $0 \times 5 + 1 \times 3 + 0 \times 2 + 1 \times 1 = 4$, which coincides with its number in Table 1.

A generalization of this method is the enumeration method for dk-sequences proposed in [3], which, in turn, was generalized in [4] (see also [5]). These methods enumerate sequences with $d \geq 0$, $k \geq d$, $l \geq 0$, and $r \geq 0$. Further development and generalization of the method of [3] was undertaken in [6], and an enumeration method for $dklr$-sequences with constraints on their weight (i.e., on the total numbers of zeros and ones) was proposed in [7]. The general scheme of operation of these algorithms is close to [2], though positions weights are computed by other rules.

It is important to note that for all the above-mentioned methods the encoding/decoding time per symbol is at least cn, where $c > 0$ is a constant, and memory grows polynomially with n. Indeed, in encoding one has to look through n positions of a word and compute the sum of weights of nonzero positions. Here the possible number of ones is proportional to n, and the binary representation of Fibonacci numbers (or weight coefficients on other methods) also amounts to n bits. Thus, to find a word of length n one has to compute the sum of cn terms of n bits each, which results in cn bit operations per symbol of an enumerated word. Since Fibonacci numbers (and analogous weight coefficients in other methods) muts be stored, memory required for encoding grows polynomially with n.

In the present paper, we propose a fast enumeration algorithm for $dklr$-sequences with polynomial memory (as for previously known algorithms) but with encoding/decoding time per symbol of the order of $O(\log^3 n \log\log n)$, which is exponentially smaller than for previously known methods. Note that complexity of methods is estimated by the required memory (in bits) and encoding/decoding time per symbol, which is measured by the number of bit operations in implementation on a RAM machine, which is a model for a "usual" computer [8].

Note that the proposed method employs an algorithm of [9] designed for enumeration/denumeration of elements of a set W of words over a finite alphabet A of a given length N with a given number of occurrences of each symbol A in words of W. For example, for $A = \{0, 1\}$ and $N = 4$, the set $W = \{(0011, 0101, 0110, 1001, 1010, 1100\}$ contains all words with two ones (and zeros). In enumeration, for each word of W, its number with respect to the lexicographic order is found; in denumeration, given a number (i.e., an integer in the range $0, \ldots, |W| - 1$), the corresponding word $w \in W$ is found. Encoding/decoding time per symbol of a word for this method is $O(\log^3 N \log\log N)$ for large N, and memory grows polynomially with N.

Below, in Section 2, we describe our method for a subclass of $dklr$-sequences; Section 3 considers the general case. We choose such way of presentation, since in our opinion it allows us to simplify description of the method.

2. ENUMERATION OF WORDS WITH CONSTRAINTS ON THE MAXIMUM RUN LENGTH OF ONES

To simplify the description of the method, we first consider a particular case of enumeration of $dklr$-sequences with $d = 0$, $k > 0$, $l = 0$, and $r = k$. Note that this problem is of a certain independent interest, since, when joining several words into one sequence, the latter does not contain runs of more than k ones due to the condition $l = 0$.

As an example, we will consider enumeration of sequences of length $n = 6$ with the constraints $d = 0$, $k = r = 1$, and $l = 0$. Here is a list of all such words: (000000), (000001), (000010), (000100), (000101), (001000), (001001), (001010), (010000), (010001), (010010), (010100), and (010101).

Every word of length n cab be divided into a sequence of subwords of the form $a_0 = (0)$, $a_1 = (01)$, …, $a_k = (0\underbrace{1\ldots1}_{k})$ (i.e., each subword consists of a single zero followed by a run of ones) and then put into correspondence with a word over the alphabet $A = \{a_0, a_1, \ldots, a_k\}$. Denote by S_i the number of subwords of the form $(0\underbrace{1\ldots1}_{0\le i\le k})$ in a word of length n. For example, for the word (000101) we have $S_0 = 2$ and $S_1 = 2$. Then the set of all words of length n with the constraints $d = 0$, $k > 0$, $l = 0$, and $r = k$ is divided into subsets defined by the condition that two words belong to one such subset if and only if these two words have the same composition $S_0, S_1, S_2, \ldots, S_k$. Denote the set defined by the tuple $S_0, S_1, S_2, \ldots, S_k$ by $m(S_0, S_1, S_2, \ldots, S_k)$. In our example,

$$m(2,2) = \{(000101), (001001), (001010), (010001), (010010), (010100)\}.$$

It easily seen that the set $m(S_0, S_1, S_2, \ldots, S_k)$ consists of words that are concatenations of all possible permutations of S_0 words (0), S_1 words (01), S_2 words (011), …, and S_k words $(0\underbrace{1\ldots1}_{k})$. In our case $m(2,2)$ consists of concatenations of $S_0 = 2$ words (0) and $S_1 = 2$ words (01). The cardinality of a set specified by a tuple $(S_0, S_1, S_2, \ldots, S_k)$ is computed as the number of permutations with repetition:

$$|m(S_0, S_1, S_2, \ldots, S_k)| = \frac{(S_0 + S_1 + \ldots + S_k)!}{S_0!\, S_1! \ldots S_k!}.$$

In our example, $|m(2,2)| = (2+2)!/(2!\cdot 2!) = 6$.

Denote the number of different tuples $(S_0, S_1, \ldots, S_k)$ by M. Establish a lexicographic order on this set, and introduce the notation $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_M$ for its elements.

One can easily check that the set $m(S_0, S_1, S_2, \ldots, S_k)$ is nonempty if and only if the tuple $(S_0, S_1, S_2, \ldots, S_k)$ satisfies the following conditions:

$$\begin{aligned}
&0 \le S_0 \le n,\\
&0 \le S_1 \le \lfloor (n - S_0)/2 \rfloor,\\
&0 \le S_2 \le \lfloor (n - S_0 - 2\,S_1)/3 \rfloor,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&0 \le S_k \le \lfloor (n - S_0 - 2\,S_1 - 3\,S_2 - \ldots - k\,S_{k-1})/(k+1) \rfloor,
\end{aligned} \tag{1}$$

and also $S_0 + 2\,S_1 + \ldots + (k+1)S_k = n$. In our example such tuples are those satisfying the conditions $0 \le S_0 \le 6$, $0 \le S_1 \le \lfloor (6 - S_0)/2 \rfloor$, and $S_0 + 2S_1 = 6$, i.e., the following: $(0,3)$, $(2,2)$, $(4,1)$, and $(6,0)$.

For enumeration, we need a table, which is computed once and then used in encoding and decoding (like the table of Fibonacci numbers and other series in methods of [2–7]). Construction

Table 2. $n = 6$, $d = 0$, $k = r = 1$, $l = 0$

S_0	S_1	$\boldsymbol{\nu}(S_0, S_1)$
0	3	0
2	2	$\boldsymbol{\nu}(0,3) + \lvert m(0,3)\rvert = 0 + (3+0)!/(3! \cdot 0!) = 1$
4	1	$\boldsymbol{\nu}(2,2) + \lvert m(2,2)\rvert = 1 + (2+2)!/(2! \cdot 2!) = 7$
6	0	$\boldsymbol{\nu}(4,1) + \lvert m(4,1)\rvert = 7 + (4+1)!/(1! \cdot 4!) = 12$

Table 3. $n = 6$, $d = 0$, $k = r = 1$, $l = 0$

Set m	Interval of numbers	Enumerated word	Word of alphabet A	Number
$m(0,3)$	$0 \ldots 0$	(010101)	$(a_1a_1a_1)$	0
$m(1,2)$	$1 \ldots 6$	(000101)	$(a_0a_0a_1a_1)$	1
		(001001)	$(a_0a_1a_0a_1)$	2
		(001010)	$(a_0a_1a_1a_0)$	3
		(010001)	$(a_1a_0a_0a_1)$	4
		(010010)	$(a_1a_0a_1a_0)$	5
		(010100)	$(a_1a_1a_0a_0)$	6
$m(4,2)$	$7 \ldots 11$	(000001)	$(a_0a_0a_0a_0a_1)$	7
		(000010)	$(a_0a_0a_0a_1a_0)$	8
		(000100)	$(a_0a_0a_1a_0a_0)$	9
		(001000)	$(a_0a_1a_0a_0a_0)$	10
		(010000)	$(a_1a_0a_0a_0a_0)$	11
$m(6,0)$	$12 \ldots 12$	(000000)	$(a_0a_0a_0a_0a_0a_0)$	12

of the table consists in the following: in the lexicographic order, we look through all admissible tuples $(S_0, S_1, \ldots, S_k)$ to obtain a two-dimensional table with $k+1$ columns and with the number of rows equal to the number of admissible tuples $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_M$. Rows of the table correspond to subsets $m(\boldsymbol{\sigma}_1)$ $m(\boldsymbol{\sigma}_2)$, $\ldots$, $m(\boldsymbol{\sigma}_M)$ of words, defined above. In row j, $j = 1, \ldots, M$, the first k elements are the jth tuple $(S_0, S_1, \ldots, S_k)$, i.e., $\boldsymbol{\sigma}_j$. The last element of the row, which we denote by $\boldsymbol{\nu}(\boldsymbol{\sigma}_j)$, is computed recursively: $\boldsymbol{\nu}(\boldsymbol{\sigma}_1) = 0$, $\boldsymbol{\nu}(\boldsymbol{\sigma}_j) = \boldsymbol{\nu}(\boldsymbol{\sigma}_{j-1}) + |m(\boldsymbol{\sigma}_{j-1})|$.

Note that this representation was earlier used in [5]; however, the complexity of the enumeration method considered there is asymptotically greater than that of our method.

Let us form a table for our example (see Table 2). We have already found admissible tuples (S_0, S_1): $(0,3)$, $(2,2)$, $(4,1)$, and $(6,0)$. Finding the cardinalities $|m(S_0, S_1)| = (S_0 + S_1)!/(S_0!\, S_1!)$ of sets corresponding to these tuples and recursively computing their cumulative sums, we obtain the results presented in Table 2. Note that these computations are made only once, before the enumeration/denumeration process.

Now we describe the order in which words of the set of *dklr*-sequences will be enumerated. The first $|m(\boldsymbol{\sigma}_1)|$ numbers (i.e., $0, 1, \ldots, |m(\boldsymbol{\sigma}_1)| - 1$) correspond to words of the set $m(\boldsymbol{\sigma}_1)$; the next $|m(\boldsymbol{\sigma}_2)|$ numbers (i.e., $|m(\boldsymbol{\sigma}_1)|, \ldots, |m(\boldsymbol{\sigma}_1)| + |m(\boldsymbol{\sigma}_2)| - 1$), to words of the set $m(\boldsymbol{\sigma}_2)$, etc. The numbers $\sum_{i=1}^{j-1} |m(\boldsymbol{\sigma}_i)|, \ldots, \sum_{i=1}^{j} |m(\boldsymbol{\sigma}_i)| - 1$ correspond to words of the set $m(\boldsymbol{\sigma}_j)$, $j = 2, \ldots, M$. Within each set m, the numbers, as well as words, are ordered lexicographically assuming that $a_0 < a_1 < \ldots < a_k$.

In our case, the numbers are arranged as shown in Table 3.

Now we describe the enumeration algorithm. For an enumerated word, we define its composition $(S_0, S_1, S_2, \ldots, S_k)$ and find the value $\boldsymbol{\nu}(S_0, S_1, \ldots, S_k)$ from the table. Finding the appropriate row in the table is made using binary search over the numbers in the first $k+1$ columns as a key (this is possible since the compositions are ordered lexicographically). In our example, for the word (010001) we find the composition $(2, 2)$, and then, by searching through Table 2, determine $\boldsymbol{\nu}(2, 2) = 1$. The word over the alphabet $A = \{a_0, a_1, \ldots, a_k\}$ corresponding to the enumerated word is found as follows: we replace its subwords (0), (01), …, $(0\underbrace{1\ldots1}_{k})$ (such that the next symbol after the subword is not one) by the symbols $a_0, a_1, \ldots, a_k$, respectively. For example, the word (001001) corresponds to $(a_0a_1a_0a_1)$. It is easily seen that each of the symbols a_i, $i = 0, 2, \ldots, k$, occurs in this new word precisely S_i times.

Then we use the method from [9] for enumeration/denumeration of so-called constant-weight words, defined as follows. Let us be given an alphabet $A = \{\alpha_1, \ldots, \alpha_k\}$, $k > 1$. Denote by $W_N^{\gamma_1,\ldots,\gamma_k}$ the set of length-N words over this alphabet such that the symbol α_i occurs in each word $w \in W_N^{\gamma_1,\ldots,\gamma_k}$ exactly γ_i times, $i = 1, \ldots, k$, where $\sum\limits_{i=1}^{k} \gamma_i = N$. In enumeration by the method of [9], for each word w in $W_N^{\gamma_1,\ldots,\gamma_k}$, its number under the lexicographic order on words of this set is found. In decoding (denumeration), the inverse operation is performed: given an integer z in the range $0, 1, \ldots, |W_N^{\gamma_1,\ldots,\gamma_k}| - 1$, a word in $W_N^{\gamma_1,\ldots,\gamma_k}$ is found whose number is z. For large N, the encoding/decoding time per one symbol of words in $W_N^{\gamma_1,\ldots,\gamma_k}$ is $O(\log^3 N \log\log N)$. For instance, for $A = \{0, 1\}$ and $N = 4$ the set $W_4^{2,2}$ contains all words with two ones (and zeros): $W_4^{2,2} = \{(0011), (0101), (0110), (1001), (1010), (1100)\}$. The number of the word (0110) is $2 = (010)_2$, and the denumeration algorithm, given the number 011, finds the word $(1001) \in W_4^{2,2}$.

A detailed description of the algorithm is given in [9]; here we note that the sets $m(i, j)$ defined above contain "constant-weight" words over the alphabet $\{a_0, a_1, \ldots, a_k\}$. In other words, each $m(i, j)$ contains all words with a fixed number of symbols $a_0, a_1, \ldots, a_k$ (this is the sense of introducing the sets $m(i, j)$). The partition of the original *dklr*-set of words into subsets is made in order to enumerate words within a subset using the fast method from [9].

We return to our example. Using the method of [9], the words are enumerated in the lexicographic order, starting with 0; in our example, $(a_0a_0a_1a_1)$ has number 0, $(a_0a_1a_0a_1)$ has number 1, etc. The final number of a word u is computed as follows: $N(u) = \boldsymbol{\nu}(S_0, S_1, \ldots, S_k) + \boldsymbol{\mu}$. For instance, $N(a_0a_1a_0a_1) = \boldsymbol{\nu}(2, 2) + \boldsymbol{\mu} = 1 + 1 = 2$.

Now we describe the decoding (denumeration) algorithm for words of length n with the run-length constraints $d = 0$, $k > 0$, $l = 0$, and $r = k$. A number N of a word is given; it is required to find the corresponding word. For instance, the number $N = 2$ is given, and it required to find the corresponding word of length $n = 6$ with the constraints $d = 0$, $k = 1$, $l = 0$, and $r = 1$. Using Table 2, we find a composition $\boldsymbol{\sigma}_j$ such that $\boldsymbol{\nu}(\boldsymbol{\sigma}_j) \le N < \boldsymbol{\nu}(\boldsymbol{\sigma}_{j+1})$, $j = 0, \ldots, t-1$, or $\boldsymbol{\nu}(\boldsymbol{\sigma}_j) \le N$, $j = t$. In our example, using Table 3 we find $1 = \boldsymbol{\nu}(2, 2) \le 2 < 7 = \boldsymbol{\nu}(4, 1)$; i.e., the desired composition is $(2, 2)$. The composition is found by binary search over the value of $\boldsymbol{\nu}$ as a key. Then we find $\boldsymbol{\mu} = N - \boldsymbol{\nu}(S_0, S_1, S_2, \ldots, S_k)$. In our case, $\boldsymbol{\mu} = 2 - 1 = 1$. Then, with the use of the fast denumeration algorithm of [9], we find a word consisting of S_0 symbols a_0, S_1 symbols a_1, S_2 symbols a_2, …, S_k symbols a_k with number $\boldsymbol{\mu}$. In our example, this is $(a_0a_1a_0a_1)$. We replace $a_0, a_1, \ldots, a_k$ with (0), (01), …, $(0\underbrace{1\ldots1}_{k})$. In the example, we obtain (010001), which is the sought-for word.

3. GENERAL ENUMERATION METHOD FOR *dklr*-SEQUENCES

Now we generalize the described method to *dklr*-sequences, $d \ge 0$, $k \ge d$, $l \ge 0$, $r \ge 0$, where d is the minimum run length of ones between zeros, k is the maximum run length of ones between

Table 4. $n = 9$, $d = 1$, $k = 2$, $l = 2$, $r = 2$

L	R	S_1	S_2	$\boldsymbol{\nu}(L, R, S_1, S_2)$
0	0	1	2	0
0	0	4	0	$\boldsymbol{\nu}(0,0,1,2) + \lvert m(0,0,1,2)\rvert = 0 + (1+2)!/(1!\cdot 2!) = 3$
0	1	2	1	$\boldsymbol{\nu}(0,0,4,0) + \lvert m(0,0,4,0)\rvert = 3 + (4+0)!/(4!\cdot 0!) = 4$
0	2	3	0	$\boldsymbol{\nu}(0,1,2,1) + \lvert m(0,1,2,1)\rvert = 4 + (2+1)!/(2!\cdot 1!) = 7$
0	2	0	2	$\boldsymbol{\nu}(0,2,0,2) + \lvert m(0,2,0,2)\rvert = 7 + (0+2)!/(0!\cdot 2!) = 8$
1	0	2	1	$\boldsymbol{\nu}(0,2,3,0) + \lvert m(0,2,3,0)\rvert = 8 + (3+0)!/(3!\cdot 0!) = 9$
1	1	3	0	$\boldsymbol{\nu}(1,0,2,1) + \lvert m(1,0,2,1)\rvert = 9 + (2+1)!/(2!\cdot 1!) = 12$
1	1	0	2	$\boldsymbol{\nu}(1,1,0,2) + \lvert m(1,1,0,2)\rvert = 12 + (0+2)!/(0!\cdot 2!) = 13$
1	2	1	1	$\boldsymbol{\nu}(1,1,3,0) + \lvert m(1,1,3,0)\rvert = 13 + (3+0)!/(3!\cdot 0!) = 14$
2	0	0	2	$\boldsymbol{\nu}(1,2,1,1) + \lvert m(1,2,1,1)\rvert = 14 + (1+1)!/(1!\cdot 1!) = 16$
2	0	3	0	$\boldsymbol{\nu}(2,0,0,2) + \lvert m(2,0,0,2)\rvert = 16 + (0+2)!/(0!\cdot 2!) = 17$
2	1	1	1	$\boldsymbol{\nu}(2,0,3,0) + \lvert m(2,0,3,0)\rvert = 17 + (3+0)!/(3!\cdot 0!) = 18$
2	2	2	0	$\boldsymbol{\nu}(2,1,1,1) + \lvert m(2,1,1,1)\rvert = 18 + (1+1)!/(1!\cdot 1!) = 20$

zeros, and l and r are the maximum run lengths of ones at the head and tail of a word, i.e., from the beginning of a word to the first zero and from the last zero to the end of a word, respectively.

As an example, we will consider words of length 9 with the constraints $d = 1$, $k = 2$, $l = 2$, and $r = 2$.

Denote by $L \le l$ the leading run length on ones, i.e., the number of ones from the beginning of a word to the first zero, and by $R \le r$, the trailing run length of ones, i.e., the number of ones from the rightmost zero to the end of the word. For instance, for the word (010110101) these values are 0 and 1, respectively. The parameters $S_d, S_{d+1}, \ldots, S_k$ are defined as follows: delete from the initial word the first L symbols (ones) and the last $R+1$ symbol (i.e., the last R ones and the preceding zero), and for the word thus obtained compute the values $S_d, S_{d+1}, \ldots, S_k$ in the same way as in Section 2. Let us show how these values are found for the word (010110101). First we delete zero first symbols and two last symbols of the word. Then for the obtained word (010110101) we find S_1 and S_2: divide the word (0101101) into a sequence of subwords (01), (011), (01), compute the number of words (01) and (011) in this sequence, and thus obtain $S_1 = 2$ and $S_2 = 1$. Replacing the subwords $(0\underbrace{1\ldots1}_{i})$ in this sequence with a_i, $i = d, d+1, \ldots, k$, we can put into correspondence to each word, as in the case described in Section 2, a word over the alphabet $A = \{a_d, a_{d+1}, \ldots, a_k\}$. In our case, to the word (0101101) we put into correspondence the word $(a_1 a_2 a_1)$. A table is constructed in the same way as in Section 2, but now rows of the table correspond to tuples $(L, R, S_d, S_{d+1}, \ldots, S_k)$. Such tuples are admissible if the following conditions are fulfilled:

$$
\begin{aligned}
&0 \le L \le l,\\
&0 \le R \le r,\\
&0 \le S_d \le \lfloor (n - L - R)/(d+1) \rfloor,\\
&0 \le S_{d+1} \le \lfloor (n - L - R - (d+1)S_d)/(d+2) \rfloor,\\
&0 \le S_{d+2} \le \lfloor (n - L - R - (d+1)S_d - (d+2)S_{d+1})/(d+3) \rfloor,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&0 \le S_k \le \lfloor (n - L - R - (d+1)S_d - (d+2)S_{d+1} - (d+3)S_{d+2} - \ldots - k\,S_{k-1})/(k+1) \rfloor,\\
&(d+1)S_d + (d+2)S_{d+1} + \ldots + (k+1)S_k + R + L = n.
\end{aligned}
\qquad (2)
$$

Denote the number of such tuples by M, and denote the sequence of admissible tuples $(L, R, S_d, S_{d+1}, \ldots, S_k)$ ordered lexicographically by $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_M$. The set defined by the tuple $(L, R, S_d, S_{d+1}, \ldots, S_k)$ will be denoted by $m(L, R, S_0, S_1, S_2, \ldots, S_k)$. Thus, the set $m(L, R, S_d, S_{d+1}, \ldots, S_k)$ contains all words with L leading and R trailing ones such that after deleting the first L and last $R+1$ symbols they become permutations of S_d words $(0\underbrace{1\ldots1}_{d})$, S_{d+1} words $(0\underbrace{1\ldots1}_{d+1})$, $\ldots$, S_k words $(0\underbrace{1\ldots1}_{k})$. The cardinality of the set corresponding to a tuple $(L, R, S_d, S_{d+1}, \ldots, S_k)$ is computed as

$$|m(L, R, S_d, S_{d+1}, \ldots, S_k)| = \frac{(S_d + S_{d+1} + \ldots + S_k)!}{S_d!\, S_{d+1}! \ldots S_k!}.$$

Then we construct a two-dimensional array with $k + 4 - d$ columns and M rows as follows: rows of the table correspond to the sets of words $m(\boldsymbol{\sigma}_1), m(\boldsymbol{\sigma}_2), \ldots, m(\boldsymbol{\sigma}_M)$ defined above. In row j, $j = 1, \ldots, M$, the first $k + 3 - d$ elements are the jth tuple $(L, R, S_d, S_{d+1}, \ldots, S_k)$, i.e., $\boldsymbol{\sigma}_j$. The last element of the row $\boldsymbol{\nu}(\boldsymbol{\sigma}_j)$, $j = 1, \ldots, M$, is computed recursively: $\boldsymbol{\nu}(\boldsymbol{\sigma}_1) = 0$ and $\boldsymbol{\nu}(\boldsymbol{\sigma}_j) = \boldsymbol{\nu}(\boldsymbol{\sigma}_{j-1}) + |m(\boldsymbol{\sigma}_{j-1})|$, $1 < j \le M$.

Let us construct a table for our example (see Table 4). Numbers of words are arranged similarly to the order described in Section 2. The first $|m(\boldsymbol{\sigma}_1)|$ numbers (i.e., $0, 1, \ldots, |m(\boldsymbol{\sigma}_1)|-1$) correspond to words from the set $m(\boldsymbol{\sigma}_1)$, the next $|m(\boldsymbol{\sigma}_2)|$ numbers (i.e., $|m(\boldsymbol{\sigma}_1)|, \ldots, |m(\boldsymbol{\sigma}_1)| + |m(\boldsymbol{\sigma}_2)| - 1$), to words of the set $m(\boldsymbol{\sigma}_2)$, etc. (the numbers $\sum_{i=1}^{j-1} |m(\boldsymbol{\sigma}_i)|, \ldots, \sum_{i=1}^{j} |m(\boldsymbol{\sigma}_i)| - 1$ correspond to words of the set $m(\boldsymbol{\sigma}_j)$, $j = 2, \ldots, M$). Within each set m, words are ordered lexicographically (with $a_d < a_{d+1} < \ldots < a_k$).

For our example, the numbers are arranged as shown in Table 5.

Let us describe the enumeration method. As an example, we consider finding the number of the word (010110101) among words with $n = 9$, $d = 1$, $k = 2$, $l = 2$, and $r = 2$. For the enumerated word, we find the values $L \le l$ and $R \le r$. Deleting the first L and last $R + 1$ symbols, we find $S_d, S_{d+1}, \ldots, S_k$. In our case, for the word (010110101) we have $L = 0$ and $R = 1$, since the word starts with no ones and ends with a single one. Then, deleting from (010110101) the first $L = 0$ symbols and the last $R + 1 = 2$ symbols, we obtain the word (0101101) and find $S_1 = 2$ and $S_2 = 1$. From the table, we find $\boldsymbol{\nu}(L, R, S_d, S_{d+1}, \ldots, S_k)$ that corresponds to the obtained composition. In our case, from Table 4 we find $\boldsymbol{\nu}(0, 1, 2, 1) = 4$. Then, to the binary word obtained after deletion, we put into correspondence a word over the alphabet $A = \{a_d, a_{d+1}, \ldots, a_k\}$: replace its subsequences $(0\underbrace{1\ldots1}_{d})$, $(0\underbrace{1\ldots1}_{d+1})$, $\ldots$, $(0\underbrace{1\ldots1}_{k})$ (such that the next symbol after the sequence is not 1) with the symbols $a_d, a_{d+1}, \ldots, a_k$, respectively. In our example, (0101101) corresponds to the word $(a_1 a_2 a_1)$. After that we apply the enumeration method of [9] to find the number $\boldsymbol{\mu}$ of this word among all words with the fixed number of occurrences of of the symbols a_i, $i = d, d+1, \ldots, k$, of the alphabet with the order $a_d < a_{d+1} < \ldots < a_k$. In our case, we find the number of the word $(a_1 a_2 a_1)$ among all words consisting of two symbols a_1 and one a_2. It is easily seen that $(a_1 a_1 a_2)$ has number 0, the word $(a_1 a_2 a_1)$ has number 1, and the word $(a_2 a_1 a_1)$ has number 2. Thus, in our case we have $\boldsymbol{\mu} = 1$. The we find the resulting number: $N = \boldsymbol{\nu}(L, R, S_d, S_{d+1}, \ldots, S_k) + \boldsymbol{\mu}$. In our case, $N = \boldsymbol{\nu}(0, 1, 2, 1) + \boldsymbol{\mu} = 4 + 1 = 5$.

The complexity of the method is described as follows.

Theorem 1 (encoding complexity). *The memory required for encoding a word of length n with the constraints $d \ge 0$, $k \ge d$, $l \ge 0$, and $r \ge 0$ with the use of the proposed algorithm is $O(n^{k-d+2})$ bits.*

The time required for encoding a word of length n with the constraints $d \ge 0$, $k \ge d$, $l \ge 0$, and $r \ge 0$ with the use of the proposed algorithm is $O(\log^3 n \log\log n)$ bit operations per symbol.

Table 5. $n = 9$, $d = 1$, $k = 2$, $l = 2$, $r = 2$

Set m	Interval of numbers	Enumerated word	Word of alphabet A	Number
$m(0,0,1,2)$	$0 \dots 2$	(010110110)	$(a_1a_2a_2)$	0
		(011010110)	$(a_2a_1a_2)$	1
		(011011010)	$(a_2a_2a_1)$	2
$m(0,0,4,0)$	$3 \dots 3$	(010101010)	$(a_1a_1a_1a_1)$	3
$m(0,1,2,1)$	$4 \dots 6$	(010101101)	$(a_1a_1a_2)$	4
		(010110101)	$(a_1a_2a_1)$	5
		(011010101)	$(a_2a_1a_1)$	6
$m(0,2,0,2)$	$7 \dots 7$	(011011011)	(a_2a_2)	7
$m(0,2,3,0)$	$8 \dots 8$	(010101011)	$(a_1a_1a_1)$	8
$m(1,0,2,1)$	$9 \dots 11$	(101010110)	$(a_1a_1a_2)$	9
		(101011010)	$(a_1a_2a_1)$	10
		(101101010)	$(a_2a_1a_1)$	11
$m(1,1,0,2)$	$12 \dots 12$	(101101101)	(a_2a_2)	12
$m(1,1,3,0)$	$13 \dots 13$	(101010101)	$(a_1a_1a_1)$	13
$m(1,2,1,1)$	$14 \dots 15$	(101011011)	(a_1a_2)	14
		(101101011)	(a_2a_1)	15
$m(2,0,0,2)$	$16 \dots 16$	(110110110)	(a_2a_2)	16
$m(2,0,3,0)$	$17 \dots 17$	(110101010)	$(a_1a_1a_1)$	17
$m(2,1,1,1)$	$18 \dots 19$	(110101101)	(a_1a_2)	18
		(110110101)	(a_2a_1)	19
$m(2,2,2,0)$	$20 \dots 20$	(110101011)	(a_1a_1)	20

Proof. The proof is based on estimating the algorithm operation time and required memory. First we consider the memory for storing the table. The number of admissible tuples, defined by inequalities (2), and therefore of rows of the table is at most lrn^{k-d+1}. Each row contains two numbers such that one of them is at most l and the other, at most r, and $k+1-d$ numbers not greater than n, since $S_d, S_{d+1}, S_{d+2}, \dots, S_k$ are obviously not greater than n. Also, each row contains a number $\boldsymbol{\nu}(S_d, S_{d+1}, \dots, S_k)$. This number is at most 2^n, since $\boldsymbol{\nu}$ is not greater than the largest number, which, in turn, is less than 2^n. Thus, the memory (in bits) required for the table is at most

$$lrn^{k-d+1}((k-d+1)\log n + \log l + \log r + n) = O(n^{k-d+2}).$$

Now we estimate enumeration time. To find the tuple $(L, R, S_d, S_{d+1}, \dots, S_k)$ for a given word, we have to look through n symbols of the word. Then we perform binary search among at most lrn^{k-d+1} elements with a key of length $(k-d+1)\log n + \log l + \log r$ bits. The time required for that is $\log(lrn^{k-d+1})((k-d+1)\log n + \log l + \log r) = O(\log^2 n)$. Then we use the algorithm of [9], whose operation time (per symbol of the input word) is $O(\log^3 n \log\log n)$. To compute the resulting number, we have to add two numbers of length at most n bits. Thus, the time required for encoding one symbol is

$$\frac{n + O(\log^2 n) + n}{n} + O(\log^3 n \log\log n) = O(\log^3 n \log\log n). \quad \triangle$$

Now we describe the denumeration algorithm. Given a number N of a word, it is required to find the word corresponding to this number. For example, given the number $N = 5$, we have to find the corresponding word of length $n = 9$ with the constraints $d = 1$, $k = 2$, $l = 2$, and $r = 2$. Using Table 4, we find a tuple $\boldsymbol{\sigma}_j$ such that $\boldsymbol{\nu}(\boldsymbol{\sigma}_j) \le N < \boldsymbol{\nu}(\boldsymbol{\sigma}_{j+1})$, $j = 0, \ldots, k-1$, or $\boldsymbol{\nu}(\boldsymbol{\sigma}_j) \le N$, $j = k$. In our example, for $N = 5$ we find from Table 5 that $4 = \boldsymbol{\nu}(0, 1, 2, 1) \le 5 < 7 = \boldsymbol{\nu}(0, 2, 3, 0)$; i.e., the desired tuple is $(0, 1, 2, 1)$. The tuple is found by binary search over the value of $\boldsymbol{\nu}$ as a key. Then we compute $\boldsymbol{\mu} = N - \boldsymbol{\nu}(L, R, S_d, S_{d+1}, \ldots, S_k)$. In our case, $\boldsymbol{\mu} = 5 - 4 = 1$. Then, using the denumeration algorithm of [9], we find a word consisting of S_d symbols a_d, S_{d+1} symbols a_{d+1}, $\ldots$, S_k symbols a_k that has number $\boldsymbol{\mu}$. In our case, this is $(a_1 a_2 a_1)$. Replace a_d, a_{d+1}, $\ldots$, a_k with $(0\underbrace{1\ldots1}_{d})$, $(0\underbrace{1\ldots1}_{d+1})$, $\ldots$, $(0\underbrace{1\ldots1}_{k})$, respectively. In our example we obtain (0101101). Then we add to this word a head consisting of L ones and a tail consisting of zero and R ones. In our case, we add to (0101101) a head of $L = 0$ ones and a tail of zero and $R = 1$ ones, thus obtaining the word (010110101).

The decoding complexity is estimated as follows.

Theorem 2 (decoding complexity). *The memory required for decoding a word of length n with the constraints $d \ge 0$, $k \ge d$, $l \ge 0$, and $r \ge 0$ with the use of the proposed algorithm is $O(n^{k-d+2})$ bits.*

The time required for decoding a word of length n with the constraints $d \ge 0$, $k \ge d$, $l \ge 0$, and $r \ge 0$ with the use of the proposed algorithm is $O(\log^3 n \log\log n)$ bit operations per symbol.

Proof. The memory required for denumeration coincides with that for enumeration. Let us estimate the algorithm operation time. At the first step, in the binary search of an appropriate tuple $(L, R, S_d, S_{d+1}, S_{d+2}, \ldots, S_k)$ in the table, we have to perform $\log(lrn^{k+1})$ comparison operations on words of length n. Thus, the binary search has the complexity of $\log(lrn^{k-d+1})n = O(n \log n)$ bit operations. Then, to perform the subtraction $N - \boldsymbol{\nu}(L, R, S_d, S_{d+1}, S_{d+2}, \ldots, S_k)$, we make one operation on words of length n. Then, to obtain the final result, we use the denumeration algorithm of [9], which requires $O(\log^3 n \log\log n)$ time per symbol. Thus, the time required for decoding one symbol is

$$\frac{O(n \log n) + n}{n} + O(\log^3 n \log\log n) = O(\log^3 n \log\log n). \quad \triangle$$

We note in conclusion that the proposed algorithms have high throughput for large values of the block length n. Preliminary estimates show that the performance surpasses that of previously known algorithms for block lengths n of the order of several hundred.

REFERENCES

1. Medvedeva, Yu. and Ryabko, B., Fast Enumeration of Run-Length-Limited Words, in *Proc. 2009 IEEE Int. Sympos. on Information Theory (ISIT'2009), Seoul, Korea*, 2009, pp. 640–643.
2. Kautz, W.H., Fibonacci Codes for Synchronization Control, *IEEE Trans. Inform. Theory*, 1965, vol. 11, no. 2, pp. 284–292.
3. Tang, D.T. and Bahl, L.R., Block Codes for a Class of Constrained Noiseless Channels, *Information and Control*, 1970, vol. 17, no. 5, pp. 436–461.
4. Beenker, G.F.M. and Immink, K.A.S., A Generalized Method For Encoding and Decoding Run-Length-Limited Binary Sequences, *IEEE Trans. Inform. Theory*, 1983, vol. 29, no. 5, pp. 751–754.
5. Datta, S. and McLaughlin, S.W., An Enumerative Method for Runlength-Limited Codes: Permutation Codes, *IEEE Trans. Inform. Theory*, 1999, vol. 45, no. 6, pp. 2199–2204.
6. Kurmaev, O.F., Coding of Run-Length-Limited Sequences, *Probl. Peredachi Inf.*, 2001, vol. 37, no. 3, pp. 34–43 [*Probl. Inf. Trans.* (Engl. Transl.), 2001, vol. 37, no. 3, pp. 216–225].

7. Kurmaev, O.F., Enumerative Coding for Constant-Weight Binary Sequences with Constrained Run-Length of Zeros, *Probl. Peredachi Inf.*, 2002, vol. 38, no. 4, pp. 3–9 [*Probl. Inf. Trans.* (Engl. Transl.), 2002, vol. 38, no. 4, pp. 249–254].

8. Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Reading: Addison-Wesley, 1976. Translated under the title *Postroenie i analiz vychislitel'nykh algoritmov*, Moscow: Mir, 1979.

9. Ryabko, B.Ya., Fast Enumeration of Combinatorial Objects, *Diskret. Mat.*, 1998, vol. 10, no. 2, pp. 101–119 [*Discrete Math. Appl.* (Engl. Transl.), 2998, vol. 8, no. 2, pp. 163–182].