Fast Enumeration of Run-Length-Limited Words

Yulia Medvedeva
Siberian State University of
Telecommunications and Informatics
Novosibirsk, Russia
Email: MJulja@gmail.com

Boris Ryabko
Institute of Computational Technologies of
Siberian Branch of Russian Academy of Science
and Siberian State University of
Telecommunications and Informatics,
Novosibirsk, Russia
Email: boris@ryabko.net

***Abstract*— An algorithm for enumeration and denumeration of run-length-limited words ($dklr$-sequences) is proposed. The complexity of the algorithm does not exceed $O(\log^3 n \log\log n)$, where n is the length of word, whereas known methods have the complexity that is not less than $cn,\ c > 0$.**

I. Introduction

In many telecommunication and storage systems there is a restriction on length of sequences of identical letters of a signal going sequentially. In this context, the problem of encoding and decoding (enumeration and denumeration) of run-length-limited words arises; see [1]. For example, one of such restrictions is the prohibition on two ones to go sequentially in a word. We say that sequences (or words) of zeroes and ones are dk-sequences if the length of any series of ones between zeroes is not less than d and not more than k. If also the length of a series of ones between the beginning of sequence and the first zero is not more than l and the length of a series of ones between the last zero and the end of sequence is not more than r then such sequence is called $dklr$-sequence. (We will notice that usually restrictions are imposed on a series of zeroes instead of ones, however, it is more convenient to us for the further statement to use the above-stated definition). We list as an example all sequences of length $n = 5$ with restriction $d = 0$, $k = 1$, i. e. such binary sequences of length 5 which do not contain two ones going sequentially: (00000), (00001), (00010), (00100), (00101), (01000), (01001), (01010), (10000), (10001), (10010), (10100), (10101).

For the first time the problem of enumerative coding of sets of run-length-limited words was considered in the paper of Kautz [2] where the method based on Fibonacci numbers was proposed. The generalization of this method is the method of enumeration of dk-sequences by Bahl and Tang [3], and its generalization, in turn, was proposed in [4]. Kurmaev developed and generalized the method of Bahl and Tang in [5], and then suggested a method of enumeration of $dklr$-sequences with restrictions on weight of sequences in [6]. It is important to notice that the time of encoding and decoding of one letter in all above-mentioned methods is not less than cn, where $c > 0$ is a constant, and the memory size grows polynomially in the length of a word n.

In this paper we propose a fast algorithm of enumeration of $dklr$-sequences for which the memory size grows polynomially (as well as in earlier known methods) in the length of words, whereas the time of encoding and decoding of one letter is $O(\log^3 n \log\log n)$, that is exponentially less than for previously known methods. (We notice that complexity of methods is estimated by the memory size (in bits) and the time of encoding and decoding of one letter measured by the number of one-bit words operations in realization on the RAM-machine, which is the model of a "usual" computer [7]). Let's notice also that the offered method uses the algorithm from [8] which is developed for enumeration of words in the alphabet $\{0,\ 1\}$ with the given length n and number of ones. Time of coding and decoding of one letter for this method is equal to $O(\log^3 n \log\log n)$.

II. Statement of the Problem and Review of Known Methods

The binary $dklr$-sequences of length n are considered. It is necessary to develop the algorithm which put each such word in correspondence with its number such that all numbers are presented in binary system by the strings of zeroes and ones of length $\lceil \log_2 |M| \rceil$, where M is the set of all words with the given restriction on length of series. This is a problem of enumeration. In turn, the restoration of word by its number is called denumeration. Let us consider an example. Let the length of words be $n = 5$ and the restriction on the lengths of series of ones be $d = 0$, $k = 1$, i. e. in a word two ones cannot go sequentially. There are 13 such words, see Table I. It is required to assign to these words their numbers of length $\lceil \log_2 13 \rceil = 4$. In the first column we write down all such words and in the second one their numbers are written in binary form.

We will briefly describe the method from [2] illustrating all stages on an example of enumeration of the words from Table I (length of word is $n = 5$, series of two and more ones are forbidden). The weight of position is determined for each position (which is a place of letter in a word numbered from right to left) using the following rule: for i-th position, $i = 1,\ \ldots,\ n$ the weight of a position is equal to $(i+1)$-th Fibonacci number. Remind that sequence of Fibonacci numbers $\{F_n\}$ is defined by a recurrent way: $F_1 = 1$, $F_2 = 1$, $F_{n+1} = F_n + F_{n-1}$, $n = 2, 3, \ldots$. The number of the word is computed as follows: the weights of positions of letter 1 are summed up. For example, if $n = 5$ then the weight $F_2 = 1$ corresponds to the first position, the weight $F_3 = 2$ corresponds to the

TABLE I

Words with restriction on length of series and their numbers: $n = 5$, $d = 0$, $k = 1$

Word	Number
00000	0000
00001	0001
00010	0010
00100	0011
00101	0100
01000	0101
01001	0110
01010	0111
10000	1000
10001	1001
10010	1010
10100	1011
10101	1100

second position, $F_4 = 3$ corresponds to the third position, $F_5 = 5$ corresponds to the fourth, and $F_6 = 8$ correspond to the fifth. It is easy to find a number of a word 01001 as follows: we add the weights of the first and fourth positions, because there is a letter 1 in these positions. As result the number of this word is $N = F_2 + F_5 = 1 + 5 = 6 = 0110_2$.

Algorithms from [3], [4], [5] are developed for a solution of more general problems: they allow to enumerate sequences with $d \geq 0$, $k \geq d$, $l \geq 0$, $r \geq 0$. In [6] the general problem of coding $dklr$-sequences with additional restrictions on their weight (i.e. on the total number of zeroes and ones in sequences) is considered. The general scheme of these algorithms is close to the method from [2] though weights of positions are calculated by other rules. It is possible to show, that the time of encoding and decoding of one letter by these methods is not less than cn, where $c > 0$ is a constant. Actually, for encoding it is necessary to look through each of n positions of encoded word and to sum up numbers (called the weights of positions and having length n bits), corresponding to positions of letter 1. A possible quantity of individual positions is proportional to n, therefore finding the number of a word of length n requires a computer to sum cn summands of n bits in length that gives cn operations over one-bit words per one letter of a word. The memory size grows polynomially in the length of a word because it is necessary to store n numbers which are the weights of positions and have the bit length n. It is important to notice that Fibonacci numbers and a number of others characteristics are calculated in advance and are stored in memory as a table which is used at encoding and decoding.

III. Enumeration of Run-Length-Limited Sequences

To simplify the description of the proposed method, in this section we will consider one special case of the enumeration for $dklr$-sequences that have restrictions: $d = 0$, $k > 0$, $l = 0$, $r = k$. As an example we consider enumeration of words of length $n = 6$ with restrictions $d = 0$, $k = r = 1$, $l = 0$. A list of all such words is: (000000), (000001), (000010), (000100), (000101), (001000), (001001), (001010), (010000), (010001), (010010), (010100), (010101).

Any of these words can be represented as a sequence of special subwords: (0), (01), ..., $(0\underbrace{1...1}_{k})$. The representation should be such that in the whole word the following letter after any subword is not "1". For example, a word (001001) is a sequence of subwords (0), (01), (0), (01). Each initial word can be matched to a word of the alphabet $A = \{a_0,\ a_1,\ ...,\ a_k\}$, by replacing subword (0) by a_0, subword (01) by a_1 and so on: $(0\underbrace{1...1}_{i})$ by a_i $(i = 0, ..., k)$. In our example, the word (001001) corresponds to a word $(a_0 a_1 a_0 a_1)$. Let us define the number of subwords $(0\underbrace{1...1}_{i})$ in the word as S_i $(i = 0, 1, 2, ..., k)$. In our example $S_0 = 2$, $S_1 = 2$. The set of all words of length n with restrictions $d = 0$, $k > 0$, $l = 0$, $r = k$ can be divided into some subsets which are defined by condition that two words belong to one such subset if and only if these two words have identical collections of S_0, S_1, S_2, ..., S_k. We denote a subset of words with a collection of S_0, S_1, S_2, ..., S_k by $m(S_0,\ S_1,\ S_2,\ ...,\ S_k)$. In our example $m(2, 2)$={(000101), (001001), (001010), (010001), (010010), (010100)}. It is easy to see that the subset $m(S_0,\ S_1,\ S_2,\ \ldots,\ S_k)$ consists of the words which are concatenations of all possible permutations of all subwords in corresponding numbers: S_0 subwords (0), S_1 subwords (01), S_2 subwords (011), ..., S_k subwords $(0\underbrace{1...1}_{k})$. In our example $m(2,\ 2)$ consists of the words which are concatenations of $S_0 = 2$ subwords (0) and $S_1 = 2$ subwords (01). The cardinality of subset of words defined by a collection $(S_0,\ S_1,\ S_2,\ ...,\ S_k)$ is calculated by the formula $|\,m(S_0,\ S_1,\ S_2,\ ...,\ S_k)\,| = (S_0 + S_1 + \ldots + S_k)!/(S_0!\,S_1! \ldots S_k!)$. In our example $|\,m(2,\ 2)\,| = (2 + 2)!/(2! \cdot 2!) = 6$.

For enumeration we need a table which is calculated once and then is used at encoding and decoding (as well as the table of Fibonacci numbers and other numbers in methods from [2], [3], [4], [5], [6]). The table construction consists of the following: by exhaustive search in lexicographic order we find all collections $(S_0,\ S_1,\ S_2,\ \ldots,\ S_k)$, such that

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

$$S_0 + 2\,S_1 + ... + (k+1)\,S_k = n.$$

There are all permissible collections, and a set $m(S_0,\ S_1,\ S_2,\ ...,\ S_k)$ is nonempty if and only if the collection $(S_0,\ S_1,\ S_2,\ ...,\ S_k)$ satisfies all conditions (1). In our example, permissible collections are ones that satisfy the

TABLE II

The table for the case $n = 6,\ d = 0,\ k = r = 1,\ l = 0$

S_0	S_1	$\nu(S_0,\ S_1)$
0	3	0
2	2	$\nu(0,\ 3) + \|m(0,\ 3)\| = 0 + (3+0)!/(3!\cdot 0!) = 1$
4	1	$\nu(2,\ 2) + \|m(2,\ 2)\| = 1 + (2+2)!/(2!\cdot 2!) = 7$
6	0	$\nu(4,\ 1) + \|m(4,\ 1)\| = 7 + (4+1)!/(1!\cdot 4!) = 12$

conditions $0 \le S_0 \le 6$, $0 \le S_1 \le \lfloor 6 - S_0 \rfloor$, $S_0 + 2\,S_1 = 6$, i.e. the following collections: (0, 3), (2, 2), (4, 1), (6, 0).

We build a matrix with $k + 1$ columns and M rows, where M is the number of permissible collections. We will denote sequences of permissible collections (S_i) arranged in a lexicographic order by σ_1, σ_2, ..., σ_M. The rows of the table correspond to the subsets of words $m(\sigma_1)$, $m(\sigma_2)$, $\ldots$, $m(\sigma_M)$ which are defined above. In a row j ($j = 1, ..., M$) the first k elements are S_i, $i = 0, 1, ..., k$. This is σ_j in explicit form. The last element of a row, which we denote by $\nu(\sigma_j)$, $j = 1..., M$, is calculated by the recurrent way: $\nu(\sigma_1) = 0$, $\nu(\sigma_j) = \nu(\sigma_{j-1}) + |m(\sigma_{j-1})|$, $1 < j \le M$.

Let us build the table for our example. We already have found the permissible collections $(S_0,\ S_1)$: (0, 3), (2, 2), (4, 1), (6, 0). Calculating the cardinalities of the sets for these collections: $|m(S_0,\ S_1)| = (S_0 + S_1)!/(S_0!\,S_1!)$ and calculating recurrently their $\nu(\sigma_j)$, we get Table II.

Notice that these calculations are done only once, prior to the beginning of encoding (and decoding).

Let us describe an numeration order of the words from the set of $dklr$-sequences. The first $|m(\sigma_1)|$ numbers (i.e. numbers 0, 1, $\ldots$, $|m(\sigma_1)| - 1$) correspond to words from the set $m(\sigma_1)$. The next $|m(\sigma_2)|$ numbers (i.e. numbers $|m(\sigma_1)|$, $\ldots$, $|m(\sigma_1)| + |m(\sigma_2)| - 1$) correspond to words from the set $m(\sigma_2)$, etc.: numbers $\sum_{i=1}^{j-1} |m(\sigma_i)|$, $\ldots$, $\sum_{i=1}^{j} |m(\sigma_i)| - 1$ correspond to words from the set $m(\sigma_j)$ ($j = 2$, $\ldots$, M).

Inside of each set $m(\sigma_j)$ the order of numbers of words correspond to the lexicographic order of the corresponding binary words of the alphabet A with order $a_0 < a_1 < ... < a_k$ defined on it.

In our example words correspond to numbers as it is shown in Table III.

Let us describe the algorithm of enumeration. We find a collection $(S_0,\ S_1,\ S_2,\ \ldots,\ S_k)$ for a numerated word and then find the value of the function $\nu(S_0,\ S_1,\ \ldots,\ S_k)$ with the help of the two-dimensional table constructed above. A search of the necessary row in the table is carried out by the binary search in numbers from the first $k + 1$ columns (it is possible since collections are arranged in the lexicographic order). In our example, for the numerated word (010001) we determine a collection (2, 2), then find $\nu(2,\ 2) = 1$ by search in Table II.

We associate the numerated word with a word of an alphabet $A = \{a_0,\ a_1,\ \ldots,\ a_k\}$ by substitution the subwords (0), (01), $\ldots$, $(0\underbrace{1\ldots1}_{k})$, (such that the following letter after any subword is not 1) with the letters a_0, a_1, $\ldots$, a_k, accordingly. In the example, we associate the word (001001) with $(a_0\,a_1\,a_0\,a_1)$.

TABLE III

Numbers of sequences for the case $n = 6,\ d = 0,\ k = r = 1,\ l = 0$

Set m	Interval of numbers	Numerated word	Word of alphabet A	Number
m (0,3)	0 ... 0	(010101)	$(a_1a_1a_1)$	0
m (1,2)	1 ... 6	(000101)	$(a_0a_0a_1a_1)$	1
		(001001)	$(a_0a_1a_0a_0)$	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 ... 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 ... 12	(000000)	$(a_0a_0a_0a_0a_0a_0)$	12

It is easy to see that in this new word the letter a_0 occurs S_0 times, each of letters a_i, $i = 1,\ 2,\ \ldots,\ k$ occurs S_i times.

Then we use a fast method of numeration from [8] to find a number of this word among all words consisting of the fixed number of the letters a_i, $i = 0,\ 1,\ \ldots,\ k$. (This algorithm is extensive enough, therefore we do not describe it in our paper and give the reference to work [8] for the full description). Let us denote this number by μ. In our example for the word $(a_0\,a_1\,a_0\,a_1)$ this number is $\mu = 1$. (By means of the fast algorithm from [8], words are enumerated in the lexicographic order, numbers begin with zero; in our example we can see, that if we numerate the words consisting of two letters a_0 and two letters a_1 in the lexicographic order, word $(a_0\,a_0\,a_1\,a_1)$ has number 0, word $(a_0\,a_1\,a_0\,a_1)$ has number 1, the word $(a_0\,a_1\,a_1\,a_0)$ has number 2, word $(a_1\,a_0\,a_0\,a_1)$ has number 3). The final number of a word is calculated by the following way: $N = \nu(S_0,\ S_1,\ \ldots,\ S_k) + \mu$. For our word (001001) we have $N = \nu(2, 2) + \mu = 1 + 1 = 2$.

Now let us obtain estimations of time complexity of the algorithm and the size of memory that we need for its realization. Firstly, let us estimate the memory size to store the table. The number of permissible collections defined by inequalities (1) and hence the number of rows in the table does not exceed n^{k+1}. Each row contains $k + 1$ numbers, the numbers are not exceeding n, since it is obvious, that $S_0,\ S_1,\ S_2,\ \ldots,\ S_k$ do not exceed n. Also each row contains only one number $\nu(S_0,\ S_1,\ \ldots,\ S_k)$. This number does not exceed 2^n because ν does not exceed the maximal number which, in turn, is less than 2^n. Thus the memory size needed for the table does not exceed $n^{k+1}\,((k+1)\,\log n + n) \le (k+2)\,n^{k+2} = O(n^{k+2})$.

Let us estimate the required time for the algorithm of enumeration. To find a collection (S_i) for a numerated word it is necessary to look up n letters of the word. Then the binary search among not more than n^{k+1} elements with the key of length $(k+1)\log n$ bits takes place. For it, we need $\log(n^{k+1})\,(k+1)\,\log n \le (k+1)^2\,\log^2 n$ time.

Then the algorithm from paper [8] is used. In this paper, it is proved that its complexity is $O(\log^3 n \log\log n)$ for one coded letter. For finding the final number, it is necessary to sum up two numbers of lengths not more than n bits. So, the time required for encoding of one letter is equal to $\frac{n+(k+1)^2\ \log^2 n+n}{n} + O(\log^3 \log\log n) = O(\log^3 n \log\log n)$. There is a denominator n in the first summand, because the time of realization of all steps of the algorithm, except the step with the use of the algorithm from [8], was calculated for the numeration of the whole word of length n.

As the result, all these arguments allow to describe the properties of the algorithm in the form of the following statement.

Theorem 1: *The proposed algorithm requires $O(n^{k+2})$ bits of the memory for encoding a word of length n with the restrictions $d = 0$, $k > 0$, $l = 0$, $r = k$, and encoding time is $O(\log^3 n \log\log n)$ bit operations per one encoded letter.*

Let us describe an algorithm of decoding (denumeration) of words of length n with considered restriction on lengths of series ($d = 0$, $k > 0$, $l = 0$, $r = k$). Word number N is given, it is required to find a word corresponding to this number. For example, it is given number $N = 2$, it is required to find a corresponding word of length $n = 6$ with restriction $d = 0$, $k = 1$, $l = 0$, $r = 1$. Using Table II, obtained above, we find a set σ_j, such that $\nu(\sigma_j) \leq N < \nu(\sigma_{j+1})$, ($j = 0$, $\ldots$, $k-1$) or $\nu(\sigma_j) \leq N$, ($j = k$). In our case we find $1 = \nu(2, 2) \leq 2 < 7 = \nu(4, 1)$, i.e. the required collection is $(2, 2)$. The collection is found by the binary search with value ν as the key. Further, we find $\mu = N - \nu(S_0, S_1, S_2, \ldots, S_k)$. In our case $\mu = 2 - 1 = 1$. Then using the fast algorithm of denumeration from [8] we find a word consisting from S_0 letters a_0, S_1 letters a_1, S_2 letters a_2, $\ldots$, S_k letters a_k with the number μ. Finally we replace a_0, a_1, $\ldots$, a_k by (0), (01), $\ldots$, $(0\underbrace{1\ldots1}_{k})$. In our example, the word is $(a_0a_1a_0a_1)$ and after replacement we get a desired word (010001).

Let us obtain estimations of a complexity of the algorithm. The memory size needed for denumeration is the same as for enumeration because the same table is used. We will estimate the time needed for the algorithm. On the first step, for the binary search of the collection $(S_0, S_1, S_2, \ldots, S_k)$ in the table, it is required to make $\log(n^{k+1})$ comparisons of words of length n, thus the binary search has complexity $\log(n^{k+1})\,n = (k+1)\,n\,\log n$ bit operations on a word of length n. Then for a subtraction $N - \nu(S_0, S_1, S_2, \ldots, S_k)$ it is required to make one operation over words of length n. Then the method of denumeration from [8] is used. In this paper it is proved that the time of decoding is $O(\log^3 n \log\log n)$ per one letter. Thus, the time required for decoding of one letter is equal to $\frac{(k+1)\,n\ \log(n)+n}{n} + O(\log^3 n \log\log n) = O(\log^3 n \log\log n)$. In the first summand, denominator n has appeared because for all steps of denumeration, except the step with the use of the algorithm from [8], the time was calculated for numeration of the whole word of length n.

This estimation allows us to describe the properties of the algorithm of decoding.

Theorem 2: *The proposed algorithm requires $O(n^{k+2})$ bits of the memory for decoding of a word of length n with restrictions $d = 0$, $k > 0$, $l = 0$, $r = k$, and decoding time is equal to $O(\log^3 n \log\log n)$ bit operations over one decoded letter.*

IV. Conclusion

We proved that the proposed algorithms have high speed for big block length n. The preliminary estimations show that the realization time of the proposed method is less than the time of algorithms from [2], [3], [4], [5] for length of the block above few tens.

In the paper we considered a partial case of $dklr$-sequences that have restrictions: $d = 0$, $k > 0$, $l = 0$, $r = k$, but this method can be extend to a general case of arbitrary parameters d, k, l, r. Similar theorems on complexity of the algorithm can be proved.

References

[1] Immink K.A.S. Codes for Mass Data Storage Systems, Shannon Foundation Publishers, Eindhoven, The Netherlands, 2004.

[2] Kautz W. Fibonacci codes for synchronisation control// IEEE Trans. Inform. Theory, V. 11, n 2, 1965, pp. 284-292.

[3] Tang D. T., Bahl L. R. Block Codes for a Class of Constrained Noiseless Channels//Inform. and Control. V. 17, n.5. 1970, pp. 436-461.

[4] Beenker G. F. M., Immink K. A. S. A Generalized Method for Encoding and Decoding Run-Length-Limited Binary Sequences//IEEE Trans. Inform. Theory, V. 29, n.3, 1983, pp. 751-754.

[5] Kurmaev O. F. Coding of sequences with the limited lengths of series//Problems of Information Transmission, v. 37, 2001, pp. 35-43.

[6] Kurmaev O. F. Numerical coding of sequences with restrictions on lengths of series of zero and weight// Problems of Information Transmission, v. 38, 2002, pp. 4-8.

[7] Aho A.V., Hopcroft L.E., Ullman J.D. The Design and Analysis of Computer Algorithms, Addison-Wesley, 1976.

[8] Ryabko B.Ya. The fast enumeration of combinatorial objects. //Discrete Math.and Applications, v.10, n2, 1998. (see also http://arxiv.org/abs/cs.CC/0601069).