Some Applications of Binary Lunar Arithmetic

Van Vinh Dang
Hochiminh city University of Technology
Hochiminh city, Vietnam
dangvvinh@hcmut.edu.vn

Nataliya Dodonova
Samara National Research University
Samara, Russia
ndodonova@bk.ru

Mikhail Dodonov
Samara National Research University
Samara, Russia
dodonoff@mail.ru

Svetlana Korabelshchikova
Northern (Arctic) Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
s.korabelsschikova@narfu.ru

***Abstract*—In general, we formulate the problem of extracting *n*-th order roots from a language as follows: for a given language *A* and a given $n \in N$ it is required to find all the languages *B*, such that $A = B^n$. This theoretical problem is closely related to the practical task of decoding messages, where it is necessary to find a partition of the encoded message into elementary codes that correspond to certain characters of the original alphabet. We previously found a solution to the problem of extracting n-th order roots for languages of a special kind, which is containing all the possible words of the length from $n \cdot n_1$ to $n \cdot n_2$ ($n_1 \leq n_2$). We reduced the problem under consideration to a knapsack problem and solved it by the method of software implementation of the algorithms proposed in the article. We found out that the number of roots depends on the values of n and k, where k is the cardinal number of the set $\{n_1, n_1+1, \ldots, n_2\}$. We obtained quantitative estimates of the number of *n*-th power roots for different values of *n* and *k*. For *n* = 2, the sequence of the number of square roots from the language coincid with the sequence published on the website of the online encyclopedia of integer sequences http://oeis.org/A191701. In binary lunar arithmetic, this sequence is the number of binary numbers *x* of length *k* such that x^2 has no zeros. In the article, we established and theoretically proved the correspondence between operations in binary lunar arithmetic and in the ring of polynomials with integer coefficients. Based on the established correspondence, we developed algorithms that allows us to solve both the special problem of finding roots from a language and the more general knapsack problem using operations in binary lunar arithmetic. We compared the software implementation of the proposed algorithm with the well-known classical alternative solutions to the knapsack problem. Using the discrete Fourier transform, we were able to improve the asymptotic complexity of the original algorithm.**

Keywords—dismal arithmetic, binary lunar arithmetic, knapsack problem

I. INTRODUCTION

American scientists D. Applegate, M. Lebrun, and N. J. A. Sloane introduced the concept of "lunar arithmetic" in [1], and originally they used the term "dismal arithmetic", subsequently replacing it with a less pessimistic "lunar arithmetic".

In the so-called lunar arithmetic, we replace the addition and multiplication of numbers by the operations of calculating the maximum and minimum, respectively. For example, 2 + 7 = 7, and 2·7=2. With multi-digit numbers, the addition operation is performed by columns, i.e., a maximum is selected in each column. For instance 6179 + 348 = 6379. The rule for multiplication of two multi-digit numbers is that the first number multiplies every digit of the second number and then the results are added together, i.e., we select the maximum in each digit (column):

			6	1	7	9
		x		3	4	8
			6	1	7	8
		4	1	4	4	
	+	3	1	3	3	
		3	4	6	4	7	8

You can learn more about the features of lunar arithmetic in [1]. We are most interested in the binary case when operations on numbers are performed according to the rules formulated above, but the numbers belong to the set {0, 1}.

In the past, in [2, 3], we considered the problem of extracting all the roots from special kind of languages. In general, the problem of extracting the *n*-th root from a language can be formulated as follows: for a given language *A* and given $n \in N$, it is required to find all languages *B* such that $A = B^n$. In this case, we call the language *B* the *n*-th root of the language *A*. This theoretical problem is closely related to the practical problem of decoding messages, where it is necessary to find the division of the encoded message into elementary codes corresponding to the characters of the original alphabet.

Let us introduce the necessary notations. Let *M* be a finite subset of the set of natural numbers, let Σ - be an alphabet, possibly infinite. We consider languages of the form $\Sigma(M)$ containing all the *i*-digit long words over the alphabet Σ, where $i \in M$. We call the set *M*, defining the language $\Sigma(M)$, - the set of indices. The solution to the problem of extracting all the roots was obtained for languages $A=\Sigma[t_1;t_2]$, containing all the words of length from t_1 to t_2 ($t_1 \leq t_2$). Let us show two examples.

Example 1. Let Σ be an arbitrary alphabet. We can extract 5 square roots from the language $A=\Sigma[2;14]$ with sets of indices {1, 2, 3, 4, 5, 6, 7}, {1, 2, 3, 4, 6, 7}, {1, 2, 4, 5, 6, 7}, {1, 2, 4, 6, 7} and {1, 2, 3, 5, 6, 7}.

Let's consider, for example, the language $B=\Sigma(M)$, where $M=\{1, 2, 4, 6, 7\}$. We can show, that $B^2=A$. The multiplicative operation here is concatenation. Since language B contains all 1-digit long words, language B^2 contains concatenations of 1-digit long words, that is, all 2-digit long words. Similarly, language B^2 contains all words of length 4, 8, 12, and 14. We can get all the 3-digit long words by concatenating words of length 1 and 2 ($1, 2 \in M$), we can get all the 5-digit long words by concatenating words of length 1 and 4 ($1, 4 \in M$), and so on for the remaining natural numbers from the interval [2, 14].

Example 2.

Let us find all the cube roots from the language $A=\Sigma[27;42]$.

Obviously, the set $M_1=\{9, 10, 11, 12, 13, 14\}$ defines a cubic root $B=\Sigma[9;14]$ from A. Similarly to the example 1, we can verify that the set $M_2 = \{9, 10, 13, 14\}$ defines another cube root from the language A. Therefore, the roots will be also two other languages with sets of indices $M_3=\{9, 10, 11, 13, 14\}$ and $M_4=\{9, 10, 12, 13, 14\}$.

In general, the n–th root from the language $A=\Sigma[t_1;t_2]$ is extracted successfully if and only if t_1 and t_2 are divisible by n. Let M be a subset of the set of natural numbers $\{n_1, n_1 + 1, \ldots, n_2\}$, where $n_1=t_1/n$, $n_2=t_2/n$. The language $B=\Sigma(M)$, containing all words of length a_j, $a_j \in M$, is an n–th root from the language A if and only if the condition is satisfied:

			1	0	1	1
		x		1	0	1
			1	0	1	1
		0	0	0	0	
+	1	0	1	1		
	1	0	1	1	1	1

$$(\forall i)\ (n \cdot n_1 \le i \le n \cdot n_2 \rightarrow i = a_1 + a_2 + \ldots + a_n), \quad (1)$$

where $a_1, a_2, \ldots, a_n$ are some elements from M (not necessarily different).

The task of checking condition (1) is equivalent to the specific case of the unlimited knapsack problem. We can solve this problem by the method of dynamic programming, finding the answer in polynomial time. Condition (1) is obviously satisfied for the set $\{n_1, n_1 + 1, \ldots, n_2\}$ and corresponds to the trivial root $B=\Sigma[n_1,n_2]$, containing all words of length from n_1 to n_2. Verification of condition (1) for other subsets of the set $\{n_1, n_1 + 1, \ldots, n_2\}$ was carried out by the method of a software implementation of well-known algorithms for solving the unlimited knapsack problem. Table 1 shows the results of the program, i.e., the number of n-th roots from languages of the form $A=\Sigma[t_1;t_2]$ for some n and k ($k = n_2 – n_1 + 1$). For k from 1 to 3, the values are 1.

TABLE I. THE NUMBERS OF N–TH ROOTS IN DIFFERENT K

***n* \k**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
2	1	2	3	5	9	15	28	50	95	174	337
3	1	2	4	7	13	25	49	95	185	365	721
4	1	2	4	8	15	29	57	113	225	447	889
5	1	2	4	8	16	31	61	121	241	481	961
6	1	2	4	8	16	32	63	125	249	497	993
7	1	2	4	8	16	32	64	127	253	505	1009

We found the sequence for $n = 2$ on the website http://oeis.org/A191701, which, in turn, refers to the work [1]. D. Applegate, M. Lebrun and N. J. A. Sloane obtained this sequence for k from 1 to 41:

1, 1, 1, 1, 2, 3, 5, 9, 15, 28, 50, 95, 174, 337, 637, 1231, 2373, 4618, 8974, 17567, 34387, 67561, ... (2)

In binary lunar arithmetic, this sequence is the number of binary numbers x of length k such that x^2 has no zeros. In this paper, we will establish the reason for this coincidence, interpret our results in terms of lunar arithmetic, and apply the established relation to solve the knapsack problem.

II. BINARY LUNAR ARITHMETIC AND OPERATIONS ON POLYNOMIALS

Let the initial set be {0, 1}. In the binary case, the lunar addition (finding a maximum) corresponds to the disjunction, and the multiplication (finding a minimum) corresponds to the conjunction. Let us consider lunar operations over multi-digit binary numbers. When adding in each column, select the maximum: 1011 + 101 = 1111. When multiplying, we follow the rule of multi-digit numbers multiplication in the lunar arithmetic shown above.

Thus, 1011· 101= 101111 (when summing by columns we find the maximum).

We see that the original vector 1011, when multiplied by 1, remains unchanged, shifting to the left by the position of this bit. This multiplication is similar to a multiplication in the ring $Z[x]$ of polynomials. If the original polynomial $f(x)$ is multiplied by x^i, then all its coefficients will remain unchanged, only i zeros will be added to the right, which corresponds to a shift of the coefficient vector $f(x)$ by i positions to the left. We associate the binary vector 1011 with the polynomial x^3+x+1 and vector 101 with the polynomial x^2+1. We get:

$$(x^3+x+1)\cdot 1= x^3+x+1=1011$$
$$(x^3+x+1)\cdot x^2= x^5+x^3+ x^2=101100$$

Adding the coefficients at the equal powers, we have $x^5+2x^3+x^2+x+1=102111$. In the polynomial ring, when calculating the coefficients at the equal power, we perform the usual addition, and in the lunar binary arithmetic if there is 1 in the column, the result is 1, if all the numbers are zeros, the result is 0. Thus, the following theorem is true.

Theorem 1. Let us compare the polynomial

$$a_n x^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$$

to the binary vector $(a_n,a_{n-1},\ldots,a_1,a_0)$. When the mapping is set in this way, operations in binary lunar arithmetic correspond to operations on polynomials in the ring Z[x] with subsequent replacement of non-zero coefficients by 1.

Proof.

Without loss of generality, we consider two binary vectors $a=(a_n,a_{n-1},\ldots,a_1,a_0)$ and $b=(b_n,b_{n-1},\ldots,b_1,b_0)$ of the same length. Otherwise, we can add to the shorter vector zeros on the left. Then $a+b=(a_n \vee b_n, a_{n-1} \vee b_{n-1},\ldots,a_1 \vee b_1, a_0 \vee b_0)$ and 0 will be in the i-th position if and only if $a_i= b_i=0$. On the other hand, for the polynomials $a(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$ and $b(x)=b_nx^n+b_{n-1}x^{n-1}+\ldots+b_1x+b_0$, we have $a(x)+ b(x)= (a_n+ b_n) x^n+(a_{n-1}+b_{n-1})x^{n-1}+\ldots+ (a_1+b_1)x+a_0+b_0$ Since the initial vectors a and b are binary, the coefficients of the polynomial $a(x)+ b(x)$ belong to the set {0,1,2}, moreover the coefficient x^i equals 0 if and only if $a_i= b_i=0$. For this reason, after replacing all nonzero coefficients by 1, the coefficient vector of the polynomial $a(x)+ b(x)$ is equal to the vector $a+b$.

The corresponding results for multiplication can be derived similarly.

III. BINARY LUNAR ARITHMETIC AND THE PROBLEM OF FINDING ALL THE ROOTS OF THE LANGUAGE

Next, we interpret the content of the sequence (1). According to the author's terminology, this sequence represents the number of *k*-digit long binary vectors. The lunar squares of these vectors consist of all ones. The lunar square is the result of multiplying the vector by itself.

Example 3. Lunar squares of the following 5 vectors with the same length $k = 7$: 1111111, 1111011, 1101111, 1101011 and 1110111 are $1111111111111=1^{(13)}$, therefore, the seventh term in the sequence (1) is 5. For all other 7-digit long binary vectors, the lunar square will contain zeros.

Let us move to the polynomial operations.

For instance, we associate the polynomial $a(x)=x^6 + x^5 + x^3 + x + 1$ with the fourth vector a=1101011. After multiplying the polynomial with itself, we get

$$(x^6 + x^5 + x^3 + x + 1)\,(x^6 + x^5 + x^3 + x + 1) =$$
$$=c_{12}x^{12} + c_{11}x^{11} + c_{10}x^{10} + c_9x^9 + c_8x^8 + c_7x^7 + c_6x^6 +$$
$$+ c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0.$$

Following the rule of theorem 1, we consider all non-zero coefficients equal to 1.

$$c_0= a_0\cdot a_0=1,$$
$$c_1= a_0\cdot a_1+ a_1\cdot a_0=1+1=1,$$
$$c_2= a_0\cdot a_2+ a_1\cdot a_1+ a_2\cdot a_0=0+1+0=1, \text{ and so on.}$$

We get $c_0= c_1=\ldots= c_{12}=1$.

The power x^i can be represented as the sum of the powers of the factors, that is as a sum of powers of a polynomial $a(x)$, if and only if this power is present in the product $a(x)\cdot a(x)$. For some, not necessarily different, j and k we obtain $i=j+k$, where $j, k\in\{0, 1, 3, 5, 6\}$.

In the polynomial $a(x)\cdot a(x)$, all the coefficients c_i are nonzero, hence any i from 0 to 12 can be represented as the sum of two (not necessarily different) numbers from {0, 1, 3, 5, 6}.

Let us recall that the problem of extracting all the roots from the language $A=\Sigma[t_1;t_2]$ is reduced to the same question. In the example 1 we extracted 5 square roots from the language $A=\Sigma[2;14]$ with the sets of indices: {1, 2, 3, 4, 5, 6, 7}, {1, 2, 3, 4, 6, 7}, {1, 2, 4, 5, 6, 7}, {1, 2, 4, 6, 7} and {1, 2, 3, 5, 6, 7}. We associate these subsets of the set {1, 2, 3, 4, 5, 6, 7} to its binary characteristic vector ($\alpha_1, \alpha_2, \ldots, \alpha_7$), so we obtain 5 binary 7-digit long vectors from the example 3.

Let us interpret the results obtained in Table 1 in terms of lunar arithmetic.

It is not difficult to verify that the second sequence of the number of cubic roots from languages of the form $A=\Sigma[t_1;t_2]$ coincides with the number of *k*-digit long binary numbers, whose lunar cubes consist of ones.

In particular, for k = 6, the lunar cubes of four vectors: 110011, 111011, 110111, and 111111 give a unit vector (it will be 16 -digit long). These vectors are characteristic vectors for the sets of indices from Example 2.

In the general, the following theorem is true.

Theorem 2. Let Σ be an arbitrary alphabet. Let k be the cardinal number of the set $\{n_1, n_1+1,..., n_2\}$, i.e. $k = n_2 – n_1 + 1$. Let M be a subset of the set $\{n_1, n_1 + 1, ..., n_2\}$, and $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ be the binary characteristic vector of the subset M. The language $B=\Sigma(M)$ is a n–th root from the language $A=\Sigma[n\cdot n_1;n\cdot n_2]$ if and only if in binary lunar arithmetic α^n has no zeros.

The Theorem 2 provides another method to verify whether a subset M is a set of indices for some root from a language of the form $A=\Sigma[t_1;t_2]$.

IV. APPLICATION OF LUNAR ARITHMETIC TO SOLVING THE KNAPSACK PROBLEM

The classical knapsack problem in general can be formulate as follows: given a set of items, each with a weight and a value, select a certain subset of objects so that we get maximum total cost, subject to the restrictions on the total weight.

The unlimited knapsack problem is a generalization of the classical problem when any item can be taken any number of times. We consider a special case of this problem when the value of the item is a natural number, and equal to its weight. Therefore, we may not take into account the cost of items, but only their weight. We also add the condition that the taken items must be equal exactly M. We formulate this problem in mathematical language.

The knapsack problem 1. Given N items. The capacity of the knapsack is W, M - the number of items to be taken, and the natural numbers $w_1, w_2, \ldots, w_N$. - are weights corresponding to the items. Find a set of values $x_1, x_2, \ldots, x_N$, where x_i is the number of taken items of a certain type, and such that:

1. $x_1 w_1 + \ldots + x_N w_N \le W;$
2. $x_1 + \ldots + x_N = M;$
3. $x_1 w_1 + \ldots + x_N w_N$ has a maximum.

Note. All the weights of objects are different. If there are any objects with the same weight, then we will always take from them only some specific one. Such an assumption will not change anything in problem 1, since any item can be selected any number of times.

Algorithm

Step 1. Create a polynomial $y = x^{w_1} + x^{w_2} + \cdots + x^{w_N}$ according to the selected set of weights $\hat{w}= \{w_1, w_2, \ldots, w_N\}$.

Step 2. Associate the polynomial y with the binary vector u of its coefficients. Then, in the vector u, for all u_i, the following condition holds $u_i=1$, if $i \in \hat{w}$; otherwise $u_i=0$.

Step 3. Raise the vector u to the power M in the lunar binary arithmetic. We get the binary vector $z = u^M$.

In the vector z, all nonzero z_k denote that there exists a set of exactly M objects, not necessarily different, whose sum of weights is k. This statement justifies the next step.

Step 4. Choose the maximum number k satisfying the conditions: $k \le W$ and $z_k \neq 0$.

Step 5. Find the linear combination method$x_1 w_1 + \ldots + x_N w_N=k$ using the backstepping.

Example 4. Given 4 items with weights 2, 3, 6, 7, and the capacity of the knapsack W=17. Find exactly 3 items.

Step 1. Create the polynomial $y = x^2 + x^3 + x^6 + x^7$.

Step 2. Associate the polynomial y with the binary vector u=(11001100) of its coefficients.

Step 3. Compute u^3 in the lunar arithmetic, we obtain:

u^2= (111011101110000), or (by the theorem 1) $y^2 = x^{14} + x^{13} + x^{12} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4$.

u^3= (11111111111111000000) = ($1^{16}0^6$).

Step 4. k=17.

Step 5. 17=10+7; 10=7+3. Hence, 17=7+3+7, so that we need to choose 2 items weighing 7 and one item weighing 3.

There is a second solution 17=14+3; 14=7+7. Hence 17=7+7+3, which, in fact, coincides with the first solution.

We evaluate the asymptotic complexity of this algorithm. Let us denote by $w_{max} = max\{ w_i \mid 1 \le i \le N\}$ the weight of the heaviest item, that is, the maximum power of the polynomial y. Then $w_{max} \cdot M$ is the maximum power of the polynomial $y^M=z$. The latter operation can be implemented in a naive way, which involves sequentially calculating the powers of y from *2* to M, or using the "binary power raising" technique, which allows raising any number to the power of n in O ($log\ n$) multiplications instead of n multiplications in the usual sequential multiplication. Using this technique, the asymptotic complexity of this algorithm is $O((w_{max} \cdot M)^2 \cdot log\ (M))$, where $(w_{max} \cdot M)^2$ is the algorithmic complexity of multiplying two polynomials, $log\ M$ is the algorithmic complexity of binary exponentiation.

We can use the Fast Fourier Transform algorithm (FFT) [4, 5] to multiply two polynomials. To multiply two polynomials in binary lunar arithmetic, based on Theorem 1, it suffices to multiply the polynomials over the standard complex field and, after obtaining the resulting polynomial, change the coefficients that are greater than one by one. Multiplication of polynomials in binary lunar arithmetic using FFT has asymptotic complexity O ($w_{max} \cdot log(w_{max})$). Thus, it turned out to improve the asymptotic complexity of the original algorithm to O ($w_{max} \cdot log(w_{max}) \cdot log(M)$). We also note that FFT can be implemented using the parallel algorithm from [6].

V. Comparative analysis of solutions to the knapsack problem

A comparative analysis of the program execution time among various solutions to the problem 1 on the knapsack problem was carried out by the 2nd year master specialized in 01.04.02 "Applied Mathematics and Computer Science" A. I. Chesnokov. Testing was carried out on computing M-th power of random polynomials of power N. There are the restrictions on N and M in experiments:

Test 1 – N = 500, M = 500;
Test 2 – N = 500, M = 1000;
Test 3 – N = 1000, M = 500;
Test 4 – N = 1000, M = 1000;
Test 5 – N = 1500, M = 500;
Test 6 – N = 1500, M = 1000;
Test 7 – N = 2000, M = 500;
Test 8 – N = 2000, M = 1000.

There were 7 different solutions to the knapsack problem 1 [7,8].

- Solution A is a solution using dynamic programming;
- Solution B is a solution using dynamic programming using bitmasks;
- Solution C is a solution using the product of polynomials in lunar arithmetic. The product of polynomials is computed using the fast Fourier transform. The M-th power of a polynomial is computed in a naive way;
- Solution D is a solution using the product of polynomials in lunar arithmetic. The product of polynomials is produced using the fast Fourier transform. The M-th power of a polynomial is computed using binary exponentiation;
- Solution E is a solution using the product of polynomials in lunar arithmetic. The product of polynomials is produced using the fast Fourier transform. The M-th power of a polynomial was raised using binary exponentiation. The parallel FFT algorithm was used [6]. Calculations are made on 2, 4, or 8 processors.

Testing was conducted on a cluster of CAFU, which has the following characteristics:

- 20 computing nodes;
- on each node there are 2 ten-core Intel Xeon processors and 64 GB of RAM;
- additionally installed mathematical coprocessors Intel Xeon Phi 5110P on the eighth node;
- The internal computer network for computing: Infiniband 56 Gb / s;
- FEFS network file system (Fujitsu Exabyte File System) with a capacity of more than 50 TB and a bandwidth of 1.67 GB / s (13.36 GB / s);
- cluster performance on the CPU in the LINPACK test 8.02 Tflops; on CPU + Xeon Phi 7.68 Tflops, cumulative 15.7 Tflops;

The results of the execution time of the programs are presented in table 2.

TABLE II. Testing results

Test	Solution A, seconds.	Solution B, seconds.	Solution C, seconds	Solution D, seconds	Solution E 2 processors, seconds	Solution E 4 processors, seconds	Solution E 8 processors, seconds
1	46.780	0.340	14.800	0.180	0.153	0.137	0.131
2	187.260	1.760	62.300	0.380	0.319	0.275	0.264
3	192.670	1.250	32.990	0.370	0.316	0.277	0.265
4	772.610	5.610	137.330	0.830	0.657	0.585	0.550
5	445.530	1.820	58.310	1.000	0.654	0.571	0.540
6	1793.230	8.160	268.720	1.900	1.476	1.246	1.244
7	779.010	3.280	77.380	0.910	0.689	0.576	0.566
8	3133.810	13.120	322.010	1.940	1.512	1.301	1.241

According to table 2, we can make a conclusion that the solution to problem 1 using FFT in binary lunar arithmetic and binary exponentiation is faster than the other solutions. The speed of the parallel FFT algorithm in all the tests increases noticeable, especially with the increasing number of threads (in some cases, significantly).

VI. Conclusion

Let us summarize the results. In the theoretical part of the work, we established a relation between operations in binary lunar arithmetic and in the ring of polynomials with integer coefficients (Theorem 1). We also created an association between binary numbers x of length k for which x^n does not contain zeros in binary lunar arithmetic and sets of indices of roots of the n-th power from a language of a special form (Theorem 2). A generalization of the problem of extracting the n-th root from a language of a special kind is a

mathematical model of a special case of the problem of an unbounded knapsack problem (Problem 1). To solve this problem, we developed an algorithm based on lunar binary arithmetic.

We made the software implementation of the proposed algorithm in several variants, using optimization methods of calculations and parallel technologies. Table 2 gives us a chance to compare the solution obtained by the authors with other known solutions (solution A and solution B) of the unbounded knapsack problem.

The knapsack problem belongs to the class NP–complete problems. This means that there is no polynomial algorithm to obtain the exact result (solution). The results presented in table 2 demonstrate that, despite this pessimistic fact, in some cases, a solution to such a problem can be obtained in seconds.

The results obtained in this research can be applied to solve many problems related to the knapsack problem. In particular, the problem of estimating the number of different cyclic codes with given parameters was solved in [9] using this method. Some additional algorithms were proposed in [10,11,12].

REFERENCES

[1] D. Applegate, M. LeBrun and N.J.A. Sloane, “Dismal Arithmetic,” Arxiv preprint: 1107.1130v2.pdf, 2011.

[2] S.Yu. Korabel’shchikova, A.I. Chesnokov and A.G. Tutygin, “On primitive roots from languages of a special type,” Proceedings of the IX international conference Discrete models in the theory of control systems, pp. 116-118, 2015.

[3] B.F. Melnikov, S.Yu. Korabelshchikova and V.N. Dolgov, “On the task of extracting the root from the language,” International Journal of Open Information Technologies, vol. 7, no. 3, pp. 1-6, 2019.

[4] G. Nussbaumer, “Fast Fourier Transform and convolution algorithms of computation,” Moscow: Radio and communications, 1985.

[5] S.Yu. Korabel’shchikova and A.I. Chesnokov, “Lunar arithmetic algorithms and fast Fourier transform in a knapsack problem,” Heuristic Algorithms and Distributed Computing, vol. 2, no. 3, pp. 41-49, 2015.

[6] V.V. Voevodin and Vl.V. Voevodin, ‘Parallel computing,” SPb.: BHV-Petersburg, 2002.

[7] D. Pisinger, ”Knapsack problems,” 1995.

[8] S. Martelo and P. Toth, “Knapsack problems,” Wiley, 1990.

[9] S.Yu. Korabel’shchikova and A.I. Chesnokov, “On the number of different cyclic codes of a given length,” Vector science TSU, vol. 4, no. 26, pp. 25-26, 2013.

[10] S.Y. Korabelshchikova, L.V. Zyablitseva, B.F. Melnikov and S.V. Pivneva, “Linear codes and some their applications,” Journal of Physics: Conference Series electronic edition, 012174, 2018.

[11] B.F. Mel’nicov, S.Yu. Korabel’shchikova, “Algorithms for computing the number of error-correcting codes of a general and special form,” Informatization and communication, vol. 1, pp. 55-60, 2019.

[12] V.M. Chernov, “Fibonacci, tribonacci, ..., hexanacci and parallel "error-free" machine arithmetic,” Computer Optics, vol. 43, no. 6, pp. 1072-1078, 2019. DOI: 10.18287/2412-6179-2019-43-6-1072-1078.