Bitwise Operations Related to a Combinatorial Problem on Binary Matrices

Krasimir Yankov Yordzhev

Faculty of Mathematics and Natural Sciences, South-West University "N. Rilsky", Blagoevgrad, Bulgaria
Email: yordzhev@swu.bg

Abstract— Some techniques for the use of bitwise operations are described in the article. As an example, an open problem of isomorphism-free generations of combinatorial objects is discussed. An equivalence relation on the set of square binary matrices having the same number of units in each row and each column is defined. Each binary matrix is represented using ordered n-tuples of natural numbers. It is shown how by using the bitwise operations can be implemented an algorithm that gets canonical representatives which are extremal elements of equivalence classes relative to a double order on the set of considered objects.

Index Terms— Programming language, bitwise operations, isomorphism-free generations of combinatorial objects, binary matrix, equivalence relation, factor-set, cardinality.

I. INTRODUCTION

The present study is thus especially useful for students educated to become programmers as well as for their lecturers. A meaningful example for the advantages of using bitwise operations for creating effective algorithms in programming is presented in this article. We will consider an open combinatorial problem on binary matrices and its solution using the algorithm for some values of the integer parameters n and k. To implement the algorithm, we will use essentially bitwise operations.

The use of bitwise operations is a powerful method used in C/C++ and Java programming languages. Unfortunately, in the widespread books on this topic there is incomplete or no description for the work of the bitwise operations. The aim of this article is to correct this lapse to a certain extent and present a meaningful example of a programming task, where the use of bitwise operations is appropriate in order to facilitate the work and to increase the effectiveness of the respective algorithm.

This work is an extension and complement to [1].

A *binary* (or *boolean*, or *(0,1)-matrix*) is a matrix whose all elements belong to the set $\mathcal{B} = \{0,1\}$. With $\mathcal{B}_n$ we will denote the set of all $n \times n$ binary matrices.

Some algorithms for isomorphism-free generations of combinatorial objects are discussed in detail in [2]. In our work we will consider a problem of this type. Its formulation is as follows: A set of binary matrices $\mathcal{L} \subseteq \mathcal{B}_n$ is given. In $\mathcal{L}$ is defined an equivalence relation.

An algorithm which did not study every element of the set $\mathcal{L}$, and which receives one representative of each equivalence class to be described. For this purpose, we will use significantly bitwise operations.

In Section II we formulate the problem and we give some well known results. In Section IV we will describe in detail an algorithm for computer solution of the formulated problem. Section III is only for reference.

II. PRELIMINARIES AND PROBLEM FORMULATION

Let n and k be positive integers. We let Λ_n^k denote the set of all $n \times n$ binary matrices in each row and each column of which there are exactly k in number 1's. Let us denote with

$$\lambda(n, k) = |\Lambda_n^k| \qquad (1)$$

the number of all elements of Λ_n^k.

There is not any known formula to calculate the $\lambda(n, k)$ for all n and k. There are formulas for the calculation of the function $\lambda(n, k)$ for each n for relatively small values of k; more specifically, for $k = 1$, $k = 2$ and $k = 3$. We do not know any formula to calculate the function $\lambda(n, k)$ for $k > 3$ and for all positive integer n.

It is easy to prove the following well-known formula

$$\lambda(n, 1) = n! \qquad (2)$$

The following formula

$$\lambda(n, 2) = \sum_{2x_2+3x_3+\cdots+nx_n=n} \frac{(n!)^2}{\prod_{r=2}^n x_r!(2r)^{x_r}} \qquad (3)$$

is well known [3].

One of the first recursive formulas for the calculation of $\lambda(n, 2)$ appeared in [4] (see also [5, p. 763]).

$$\lambda(1,2) = 0, \quad \lambda(2,2) = 1, \quad \lambda(3,2) = 6; \qquad (4)$$

for $n \geq 4$,

$$\lambda(n, 2) = \frac{1}{2}n(n-1)^2[(2n-3)\lambda(n-2,2) + (n-2)^2\lambda(n-3,2)] \qquad (5)$$

Another recursive formula for the calculation of $\lambda(n, 2)$ occurs in [6].

$$\lambda(1,2) = 0, \quad \lambda(2,2) = 1; \tag{6}$$

for $n \geq 3$

$$\lambda(n, 2) = (n - 1)n\lambda(n - 1,2) + \frac{(n-1)^2 n}{2}\lambda(n - 2,2) \tag{7}$$

The next recursive system is to calculate $\lambda(n, 2)$:

$$\lambda(1,2) = 0, \lambda(2,2) = 1, \tag{8}$$

for $n \geq 2$

$$\lambda(n + 1,2) = n(2n - 1)\lambda(n, 2) + n^2\lambda(n - 1,2) - \pi(n + 1); \tag{9}$$

$$\pi(1) = \pi(2) = \pi(3) = 0, \pi(4) = 9; \tag{10}$$

for $n \geq 4$

$$\pi(n + 1) = (n - 1)^2 4[8(n - 2)(n - 3)\lambda(n - 2,2 + (n - 2)^2\lambda(n - 3,2) - 4\pi(n - 1)] \tag{11}$$

where $\pi(n)$ identifies the number of a special class of Λ_n^2-matrices [7].

The following formula is an explicit form for the calculation of $\lambda(n, 3)$.

$$\lambda(n, 3) = \frac{n!^2}{6^n} \sum \frac{(-1)^\beta (\beta + 3\gamma)! 2^\alpha 3^\beta}{\alpha! \beta! \gamma!^2 6^\gamma}, \tag{12}$$

where the sum is done as regards all $\frac{(n+2)(n+1)}{2}$ solutions in nonnegative integers of the equation $\alpha + \beta + \gamma = n$ [8]. As it is noted in [9], the above formula does not give us good opportunities to study behavior of $\lambda(n, 3)$.

Let $A, B \in \Lambda_n^k$. We will say that $A : B$, if A is obtained from B by moving some rows and/or columns. Obviously, the relation defined like that is an equivalence relation. We denote with

$$\mu(n, k) = \left| \Lambda_{n/\sim}^k \right| \tag{13}$$

the number of equivalence classes on the above defined relation.

Problem 1 *Find $\mu(n, k)$ for given integers n and k, $1 \leq k < n$.*

The task of finding the number of equivalence classes for all integers n and k, $1 \leq k < n$ is an open scientific problem. We partially solve this problem by making a computer program to find this number for some (not great) values of n and k. Moreover, using bitwise operations, our algorithm will receive one representative from each equivalence class without examining the whole set Λ_n^k.

III. BITWISE OPERATIONS

Bitwise operations can be applied for integer data type only, i.e. they cannot be used for float and double types. For the definition of the bitwise operations and some of their elementary applications could be seen, for example, in [10, 11, 12].

We assume, as usual that bits numbering in variables starts from right to left, and that the number of the very right one is 0.

Let x,y and z are integer variables or constants of one type, for which bits are needed. Let x and y are initialized (if they are variables) and let the assignment z = x & y; (*bitwise AND*), or z = x | y; (*bitwise inclusive OR*), or z = x ^ y; (*bitwise exclusive OR*), or z = ~x; (*bitwise NOT*) be made. For each $i = 0,1,2, \ldots, w - 1$, the new contents of the i-th bit in z will be as it is presented in the Table I.

TABLE I.
BITWISE OPERATIONS

i-th bit of x	i-th bit of y	i-th bit of z = x & y;	i-th bit of z = x | y;	i-th bit of z = x ^ y;	i-th bit of z = ~x;
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

In case that k is a nonnegative integer, then the statement z = x<<k (*bitwise shift left*) will write $(i + k)$ in the bit of z the value of the k bit of x, where $i = 0, 1, \ldots, w - k - 1$, and the very right k bits of x will be filled by zeroes. This operation is equivalent to a multiplication of x by 2^k.

The statement z=x>>k (*bitwise shift right*) works the similar way. But we must be careful if we use the programming language C or C++, as in various programming environments this operation has different interpretations – somewhere k bits of z from the very left place are compulsory filled by 0 (logical displacement), and elsewhere the very left k bits of z are filled with the value from the very left (sign) bit; i.e. if the number is negative, then the filling will be with 1 (arithmetic displacement). Therefore it is recommended to use unsigned type of variables (if the opposite is not necessary) while working with bitwise operations. In the Java programming language, this problem is solved by introducing the two different operators: z=x>>k and z=x>>>k [12].

Bitwise operations are left associative.

The priority of operations in descending order is as follows: ~ (bitwise NOT); the arithmetic operations * (multiply), / (divide), % (remainder or modulus); the arithmetic operations + (addition) - (subtraction); the bitwise operations << and >>; the relational operations <, >, <=, >=, ==, !=; the bitwise operations &,^ and |; the logical operations && and ||.

To compute the value of the i -th bit of an integer variable x we can use the function:

```
int BitValue(int x, unsigned int i)
{
    return ( (x & 1<<i) == 0 ) ? 0 : 1;
}
```

The next function prints an integer in binary notation. We don't consider and we don't print the sign of integer. For this reason we work with $|n|$.

```
void DecToBin(int n)
{
    n = abs(n);
    int b;
    int d = sizeof(int)*8 - 1;
    while ( d>0 && (n & 1<<(d-1) ) == 0 ) d--;
    while (d>=0)
        {
        b= 1<<(d-1) & n ? 1 : 0;
        cout<<b;
        d--;
        }
}
```

The following function calculates the number of 1's in the binary representation of an integer n. Again we ignore the sign of the number.

```
int NumbOf_1(int n)
{
    n = abs(n);
    int temp=0;
    int d = sizeof(int)*8 - 1;
    for (int i=0; i<d; i++)      if (n & 1<<i) temp++;
    return temp;
}
```

IV. DESCRIPTION AND IMPLEMENTATION OF THE ALGORITHM

Let $\mathbb{N}$ be the set of natural numbers and let

$$\mathcal{T}_n = \{\langle x_1, x_2, \ldots, x_n \rangle \mid x_i \in \mathbb{N}, i = 1, 2, \ldots, n\} \tag{14}$$

An one to one corresponding

$$\varphi : \mathcal{B}_n \overset{\sim}{\to} \mathcal{T}_n \tag{15}$$

which is based on the binary presentation of the natural numbers, is described in [1]. If $A \in \mathcal{B}_n$ and $\varphi(A) = \langle x_1, x_2, \ldots x_n \rangle$, then i-th row of A is integer x_i written in binary notation.

In [13], it is proved that the representation of the elements of $\mathcal{B}_n$ using ordered n -tuples of natural numbers leads to making a fast and saving memory algorithms.

Let $A \in \mathcal{B}_n$ and let

$$\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle = \varphi(A). \tag{16}$$

Then we denote

$$\mathbf{x}^t = \varphi(A^t), \tag{17}$$

where $A^t \in \mathcal{B}_n$ is the transpose of the matrix A.

Let

$$\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \tag{18}$$

and let

$$\mathbf{x}^t = \langle y_1, y_2, \ldots, y_n \rangle. \tag{19}$$

Then we will call **x** a *canonical element*, if

$$x_1 \leq x_2 \leq \cdots \leq x_n \tag{20}$$

and

$$y_1 \leq y_2 \leq \cdots \leq y_n. \tag{21}$$

Proposition 1 *There is un unique canonical element in every equivalence class of factor-set $\Lambda_{n/:}^k$.*

The proof of proposition 1 is within the reach of any student who has successfully studied the properties of the binary system concept and we will miss it here.

Proposition 1 is the base of our algorithm, which we describe in brief below. For its implementation, we will use also the functions shown in Section III.

As it is well known, there are exactly 2^n nonnegative integers, which are presented with no more than n digits in binary notation. We need to select all of them, which have exactly k 1's in binary notation. Their number is $\binom{n}{k} \ll 2^n$. We could use the function NumbOf_1(int) from section 3, but then we have to use it for each integer from the interval $[0, 2^n - 1]$, i.e. 2^n times. We will describe an algorithm that directly receives the necessary elements without checking whether any integer $m \in [0, 2^n - 1]$ satisfies the conditions. We will remember the result in the array p[] of size $c = \binom{n}{k}$. Moreover, the obtained array is sorted in ascending order and there are no duplicate elements. The algorithm is based on the fact that the set of all ordered m-tuples

$$\mathcal{B}^m = \langle b_1, b_2, \dots, b_m \rangle, \; b_i \in \mathcal{B} = \{0,1\}, \quad (22)$$

$$i = 1,2, \dots, m, \; m = 1,2, \dots, n,$$

is partitioned into two disjoint subsets

$$\mathcal{B}^m = \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset, \quad (23)$$

where

$$\mathcal{M}_1 = \{\langle b_1, b_2, \dots, b_m \rangle \mid b_1 = 0\} \quad (24)$$

and

$$\mathcal{M}_2 = \{\langle b_1, b_2, \dots, b_m \rangle \mid b_1 = 1\}. \quad (25)$$

The described recursive algorithm again uses bitwise operations

```
void DataNumb(int p[], unsigned int n, int k, int& c)
{
    if (k<=0)
    {
    c = 1;
    p[0] = 0;
    }
    else if (k==n)
        {
        c = 1;
        p[0] = (1<<n)-1;   // p[0]= 2^n -1
        }
    else
```

```
    {
    int p1[10000], p2[10000];
    int c1, c2;
    DataNumb(p1, n-1, k, c1);
    DataNumb(p2, n-1, k-1, c2);
    c = c1+c2;
    for (int i=0; i<c1; i++)
            p[i] = p1[i];
    for (int i=0; i<c2; i++)
            p[c1+i] = p2[i] | 1<<(n-1);
    }
}
```

We also will use bitwise operations in constructing the next two functions.

The function int n_tuple(int[], int, int, int) gets all $t = \binom{n+k-1}{k}$ (combinations with repetitions) ordered n-tuples $\langle x_1, x_2, \dots, x_n \rangle$, where $0 \le x_1 \le x_2 \le \dots \le x_n < c$, x_i, $i = 1,2, \dots, n$ are elements of sorted array p[] of size c. As a result, the function returns the number of canonical elements.

The function bool check(int[], int) refers to the use of each received n-tuples. It examines whether this is a canonical element and prints it.

```
bool check(int x[], int n, int k)
{
    int yj;        // the integer representing column (n-j)
    int y0=0;      // integer preceding column j
    int b;
    for (int j=n-1; j>=0; j--)
    {
        yj=0;
        for (int i=0; i<n; i++)
        {
            b = 1<<j & x[i] ? 1 : 0;
            yj |= b << (n-1-i);
        }
        if (yj<y0 || (NumbOf_1(yj) != k)) return false;
        y0 = yj;
    }

    // We have received a canonical element. Print it:

    for (int i=0; i<n; i++) cout<<x[i]<<" ";
    cout<<'\n';
    return true;
}
```

```
int n_tuple(int p[], int n, int k, int c)
{
    int t=0;
    int a[n], x[n];
    int indx = n-1;
    for (int i=0; i<n; i++) a[i]=0;
    while (indx >= 0)
    {
```

```
        for (int i=indx+1; i<n; i++)   a[i] = a[indx];
        for (int i=0; i<n; i++)   x[i] = p[a[i]];
         if(check(x,n,k)) t++;
        indx = n-1;
        a[indx]++;
        while (indx>=0 && a[indx]==c)
        {
                indx--;
                a[indx]++;
        }
    } return t;
}
```

The description of the main function, we leave to the reader.

V. CONCLUSIONS

The number of equivalence classes for $1 \leq k < n \leq 9$ are given in Table II, which is obtained through the work of the algorithms described in this paper.

The ideas described in this article can be used for finding the cardinality of other factor-sets of binary matrices

TABLE II
THE NUMBER OF EQUIVALENCE CLASSES FOR $1 \leqslant K < N \leqslant 9$

k \ n	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1
2		1	2	5	13	42	155	636
3			1	3	25	272	4 070	79 221
4				1	5	161	7 776	626 649
5					1	8	1 112	287 311
6						1	13	8 787
7							1	21
8								1

REFERENCES

[1] K. Yordzhev, An example for the use of bitwise operations in programming: *Mathematics and education in mathematics*, **38** (2009), 196-202.

[2] I. Bouyukliev, About Algorithms for Isomorphism-free generations of Combinatorial objects: *Mathematics and education in mathematics*, **38** (2009), 51-60.

[3] V. E. Tarakanov, Combinatorial problems on binary matrices: *Combinatorial Analysis*, Moscow, Moscow State University, **5** (1980), 4-15 (in Russian).

[4] H. Anand, V. C. Dumir and H. Gupta, A combinatorial distribution problem: *Duke Math. J.* **33** (1966), 757-769.

[5] H. Gupta and G. L. Nath, Enumeration of stochastic cubes: *Notices of the Amer. Math. Soc.* **19** (1972) A-568.

[6] I. Good and J. Grook, The enumeration of arrays and generalization related to contingency tables: *Discrete Math*, **19** (1977), 23-45.

[7] K. Yordzhev, Combinatorial problems on binary matrices: *Mathematics and education in mathematics*, **24** (1995), 288-296.

[8] M. L. Stein and P. R. Stein, Enumeration of stochastic matrices with integer elements: Los

Alamos Scientific Laboratory Report LA-4434, 1970.

[9] R. P. Stanley, Enumerative combinatorics. V.1, Wadword & Brooks, California, 1986.

[10] S.R. Davis, *C++ for dummies*. IDG Books Worldwide, 2000.

[11] B.W. Kernigan, D.M. Ritchie, *The C programming Language*. AT&T Bell Laboratories, 1998.

[12] H. Schildt, *Java 2 A Beginner's Guide*. McGraw-Hill, 2001.

[13] H. Kostadinova and K. Yordzhev, A Representation of Binary Matrices: *Mathematics and education in mathematics*, **39** (2010), 198-206.

Prof. Dr. Krasimir Yordzhev is a lecturer in computer science, programming and discrete mathematics at the Department of Computer Science, Faculty of Mathematics and Natural Sciences, South-West University, Blagoevgrad, Bulgaria. In research, his current interests include theoretical computer science and in particular the study of algorithms and their complexity using analytical, combinatorial or probabilistic methods. Dr. Yordzhev received his PhD degree in the Faculty of Cybernetics, Kiev State University, Ukraine. He is a member of Editorial Board of International Journal of Modern Education and Computer Science (IJMECS) and Associate Editor of International Journal of Education and Management Engineering (IJEME).