МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2009
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2009
Proceedings of the Thirty Eighth Spring Conference of
the Union of Bulgarian Mathematicians
Borovetz, April 1–5, 2009

AN EXAMPLE FOR THE USE OF BITWISE OPERATIONS IN PROGRAMMING

Krasimir Yordzhev

This piece of work presents a meaningful example for the advantages of using bitwise operations for creating effective algorithms in programming. A task connected with mathematic modeling in weaving industry is examined and computed.

1. Introduction. The use of bitwise operations is a powerful method provided by C/C++ programming languages. Unfortunately in widespread books on this topic there is incomplete or no description for the work of the bitwise operations [2, 4, 5, 9, 11]. The aim of the article is to correct this lapse to a certain extent and present a meaningful example of a programming problem, where the use of bitwise operations is appropriate in order to facilitate the work and to increase the effectiveness of the respective algorithm.

On the other hand the algorithm specified here could have a good practical application for computing a known combinatorial task connected with the classification of the various textile structures.

2. Problem formulation. Let us denote by Bn the set of all $n \times n$ *binary* matrices, i.e. matrices composed by n rows and n columns, all elements of which are either 0 or 1. It is a well-known fact that the number of all matrices of Bn is equal to 2^{n^2}. Let $A, B \in Bn$. We will say, that A and B are equivalent and we will write $A \sim B$, if B is obtained from A as a result of sequential cyclic move of the last row or column at a first place. It is easy to see that the so described relation is an equivalence relation. In this way we come to a formulation of the following programming problem:

Problem 1. *Write a program for with assigned positive integer n returns one representative of each equivalence class in Bn concerning the above mentioned equivalence relation.*

As a result from the solution of problem 1 we will also compute a combinatorial problem to find the number of all equivalence classes in Bn regarding the equivalence relation $\sim$, i.e. for finding the cardinal number of the factor set $Bn_{/\sim}$.

This problem is applicable in wavering industry. With the help of the elements of Bn the various threads interweaving of a certain weaver structure could be coded, and with this coding by using two equivalent matrices the weaving of one and the same fabric is coded, because of cyclic recurrence of the repetition of interweaving [6, 8].

From a practical point of view just matrices with at least one 0 and at least one 1 in each row and each column have meaning. Let us mark with Qn the set of all matrices of that kind, $Qn \subset Bn$. The next problem which we are going to compute is a bit more difficult version of problem 1.

Problem 2. *Write a program that for assigned positive integer n returns one representative of each equivalence class in Qn concerning the above mentioned equivalence relation.*

3. Bitwise operations in C/C++. 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 in C/C++ and some of their elementary applications could be seen, for example, in [1, 3, 7, 10].

We assume as usually 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 be integer variables of one type, for which w bits are needed. Let x and y be initialized and let the assignment $z = x\alpha y$ is made, where α is one of the operations & (**bitwise AND**), | (**bitwise inclusive OR**) or $\wedge$ (**bitwise exclusive OR**). For each $i = 0, 1, \dots, w-1$ the new contents of the i bit in z will be as it is presented in the following table:

The i bit of x	The i bit of y	The i bit of $x\&y$	The i bit of $x\|y$	The i bit of $x \wedge y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

In case that $z =\sim x$, if the i bit of x is 0, then the i bit of z becomes 1, and if the i bit of x is 1, then the i bit of z becomes 0, $i = 0, 1, \dots, w-1$.

In case that k is a nonnegative integer, then the statement $z = x \ll k$; (**bitwise shift left**) will write in the $(i+k)$ bit of z the value of the k bit of x, where $i = 0, 1, \dots, w-k-1$, and the very right k bits of z will be filled by zeroes. This operation is equivalent to a multiplication of x by 2^k. The statement $z = x \gg k$ works in similar way (**bitwise shift right**). But we must be careful here, because in various programming environments (see for example in [7]) 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.

Directly from the definition of the operation bitwise shift left follows the effectiveness of the following function computing 2^k, where k is a nonnegative integer:

```
unsigned int Power2(unsigned int k) {
      return 1<<k;
}
```

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

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

Bitwise operations are left associative.

The priority of operations in descending order is as follows: *bitwise complement* $\sim$; *arithmetic operations* $*$ (multiply), / (divide), % (remainder or modulus); *arithmetic operations* $+$ (binary plus or add) $-$ (binary minus or subtract); the *bitwise operations* << and >>; *relational operations* <, >, <=, >=, ==, !=; *bitwise operations* &, $\wedge$ and |; *logical operations* && and ||.

4. Algorithm realization. Each $n \times n$ binary matrix A can be coded with the help of vector (array) of n nonnegative integers $v = (v_0, v_1, \dots, v_{n-1})$, where $0 \leq v_i \leq 2^n - 1$ for each i: $0 \leq i \leq n-1$. One-to-one correspondence is realized through binary presentation of natural numbers, i.e. the i row of the matrix A is v_i in binary system. The row i of A will be completely nil if and only if $v_i = 0$; and all elements of the i row of A will be equal to 1 if and only if $v_i = 2^n - 1$. In other words, it is a necessary and sufficient condition for each $i = 0, 1, \dots, n-1$ to be realized $1 \leq v_i \leq 2^n - 2$, in order to obtain at least one 0 and at least one 1 in each row. In order to obtain at least one 0 in each column of the matrix A, it is necessary and sufficient that the bitwise AND of all numbers, representing the rows of A to be equal to 0. In order to obtain at least one 1 in each column of the matrix A it is necessary and sufficient that the bitwise inclusive OR of all numbers, representing the rows of A to be equal to $2^n - 1$, i.e. to be equal to the number which is written in binary system with n ones.

Thus we obtain the following function, which checks whether the array of n integers $v = (v_0, v_1, \dots, v_{n-1})$ represents a matrix of Qn, or not

```
int IsQn(unsigned int v[], unsigned int n) {
   // Returns 1, if with v a matrix in  Qn  is coded
   // Returns 0, otherwise
      for (int i=0; i <= n-1; i++)
         if (v[i]<1 || $ v[i] > (1<<n)-2) return 0;
      int x,y;
      x = (1<<n) -1;
      y=0;
      for (int j=0; j <= n-1; j++) {
         x = x & v[j];
         y = y | v[j];
      }
      if (x != 0) return 0;
      if (y != (1<<n)-1) return 0;
      return 1;
}
```

Let x be an integer, for which we are certain that it belongs to the interval $0 \leq x \leq 2^n - 1$, i.e. there is no need of more than n digits 0 or 1 for its binary code. Then to present x in binary system (see the function BitValue described in the previous section), written with the aid of exactly n digits 0 or 1 and eventually with a certain number of insignificant zeroes at the beginning, we can use the following function:

```
void BinPrn(int x, unsigned int k) {
   int z;
   for (int i = k-1; i >= 0; i--) {
      z = x & (1<<i);
      if (z == 0) cout<<'0';
      else cout<<'1';
   }
   cout<<'\n';
}
```

Let us examine the set

$$V = \{(v_0, v_1, \ldots, v_{n-1}) \mid 0 \leq v_i \leq 2^n - 1, i = 0, 1, \ldots, n-1\}.$$

All elements of V can be sorted in ascending lexicographic order. The essence of the proposed by us algorithm is to obtain sequentially all elements of V in the same increasing order from the smallest one to the biggest one and right after obtaining them to check whether this element is minimal according to the lexicographic order in the class of equivalence. At last we will separate just the minimal in their class of equivalency elements and they will be the only representatives of each equivalency class in the sets Bn and Qn (which was required in Problems 1 and 2). For this purpose we will design a function IsMin, which will return 1, if the input argument is minimal in the class of equivalency to which it belongs to, and 0 otherwise. But before that we need the following auxiliary function CicleMove, which for assigned nonnegative integers x and n returns a number, which is obtained from x by moving all bits one position to the right, beginning with the moving of the very right bit to the place of the bit $n-1$. In this case we will be helped by the bitwise operations.

```
unsigned int CicleMove(unsigned int x,unsigned int n)
   unsigned int b0 = x & 1; // Record the value
                      //of the very right bit of x
   x=x & ((1<<n)-1);    // Replaces all bits to the
      //left from the on with number (n-1) with 0
   return (x >> 1)$\vert $(b0 << n-1);
}
```

The following auxiliary function will also be useful for the computing of the main problem:

```
Int IsLess(unsigned int u[],unsigned int v[],int n)
{
// Return 1, if according to lexicographic order
   //u[0] u[1] \ldots u[n-1] < v[0] v[1] \ldots v[n-1]
// Return 0, otherwise
   int i = 0;
 while ((u[i] == v[i]) && (i<n-1)) i++;
   if (u[i] < v[i]) return 1;
      else return 0;
}
```

The above mentioned function IsMin could look as follows:

```
int IsMin(unsigned int v[], unsigned int n) {
// Return 1, if according to lexicographic order
//v[0] v[1] \ldots v[n-1]is minimal in its class of equivalency
// Return 0, otherwise
   unsigned int u[32], v1[32];
   for (int i = 0; i <= n-1; i++) v1[i] = v[i];
   for (int j = 0; j <= n-1; j++) {
      for (int i = 1; i <= n-1; i++) {
         for (int s = 0; s <= n-1; s++) {
            int s1 = (s+i) % n;
            u[s] = v1[s1];
         }
         if (IsLess(u,v,n) ) return 0;
      }
      for (int i=0; i <= n-1; i++) v1[i]=CicleMove(v1[i],n);
   }
   return 1;
}
```

Taking the advantages of the above described functions we propose the following computing of Problems 1 and 2 (for $n = 4$, for example). In order to be brief here we will not print all the elements obtained, and we will obtain their numbers only. For the hard-copy itself for each row of any of the obtained matrices we can take advantage of the above described procedure BinPrn and after organizing of a cycle by the number of the row to print the whole matrix as well.

```
int main() {
   const int n=4;
   int i;
   unsigned long int NBn = 0; // Number of elements in Bn
   unsigned long int NQn = 0; // Number of elements in Qn
   unsigned long int NBnEq = 0; // Number of the
                      //classes of equivalency in Bn
   unsigned long int NQnEq = 0; // Number of the
                      //classes of equivalency in Qn
   unsigned int v[n];
   int r=(1<<n)-1;}
   for (i = 0; i<n; i++) v[i]=0;
   do {
      i=n-1;
      for (int k=0; k<=r; k++) {
         v[i]=k;
         NBn++;
         if (IsQn(v,n) ) NQn++;
         if (IsMin(v,n) ) {
            NBnEq++;
            if (IsQn(v,n)) NQnEq++;
         }
      }
      while (v[i]==r) i--;
         if (i>=0) {
            v[i]++;
            for (int k=n-1; k>i; k--) {
               v[k]=0;
            }
         }
      } while ( i>=0 );
      cout<<"Number of elements in Bn "<<NBn<<'\n';
      cout<<"Number of elements in Qn "<<NQn<<'\n';
      cout<<"Number of the classes of equivalency in Bn "<<NBnEq<<'\n';
      cout<<"Number of the classes of equivalency in Qn "<<NQnEq<<'\n';
   return 0;
}
```

The results from the above described program for some values of n can be summarized in the following table.

n	1	2	3	4	5	6
$\|Bn\|$	2	16	512	65 536	33 554 432	$2^{36} > 2^{32} - 1$
$\|Qn\|$	0	2	102	22 874	17 633 670	$> 2^{32} - 1$
$\|Bn_{/\sim}\|$	2	7	64	4 156	1 342 208	1 908 897 152
$\|Qn_{/\sim}\|$	0	1	14	1 446	705 366	1 304 451 482

REFERENCES

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

[2] C. S. Horstmann Computing concepts with C++ essentials. John Wiley & Sons, 1999.

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

[4] П. Азълов. Информатика: Езикът C++ в примери и задачи за 9-10 клас. София, Просвета, 2005.

[5] П. Азълов. Обектно ориентирано програмиране Структури от данни и STL. София, Сиела, 2008.

[6] Г. И. Борзунов. Шерстяная промишленост – обзорная информация. Москва, ЦНИИ ИТЭИЛП, **3**, 1983.

[7] С. В. Глушаков, А. В. Коваль, С. В. Смирнов. Язык программирования C++. Харьков, Фолио, 2001.

[8] К. Я. Йорджев, И. В. Статулов. Математическо моделиране и количествена оценка на първичните тъкачни сплитки. *Текстил и облекло*, **10**, (1999), 18–20.

[9] Х. Крушков. Практическо ръководство по програмиране на C++. Пловдив, Макрос, 2006.

[10] Е. Л. Романов. Практикум по программированию на C++. Санкт Петербург, БХВ, 2004.

[11] М. Тодорова. Програмиране на C++. Част I, част II, София, Сиела, 2002.

Krasimir Yordzhev
South-West University “N. Rilsky”
2700 Blagoevgrad, Bulgaria
e-mail: iordjev@swu.bg, iordjev@yahoo.com

ПРИМЕР ЗА ИЗПОЛЗУВАНЕ НА ПОБИТОВИ ОПЕРАЦИИ В ПРОГРАМИРАНЕТО

Красимир Йорджев

В работата е посочен съдържателен пример за използването на побитовите оператори при оформянето на ефективни алгоритми в програмирането. Разгледана и решена е една задача, свързана с математическото моделиране в тъкачната промишленост.