

**ON SOME ENTERTAINING APPLICATIONS OF THE CONCEPT OF SET
IN COMPUTER SCIENCE COURSE**

Yordzhev K., Kostadinova H.

Some aspects of programming education are examined in this work. It is emphasised, based on the entertainment value, the most appropriate examples are chosen to demonstrate the different language constructions and data structures. Such an example is the demonstrated algorithm for solving the widespread nowadays “Sudoku” puzzle. This is made, because of the connection with the term set and putting it into practice in the programming. Using the so built program there are solved some combinatorial problems, connected to the Sudoku matrices.

Keywords: Education in programming, programming languages, data structures, set, Sudoku matrix, Sudoku puzzle.

INTRODUCTION

The present work is contemplated to help the lecturer in programming in his aspiration for giving an appropriate, interesting and entertaining example of the advantage to use the term set in programming. For this purpose the students have to be familiar with the basic definitions of the set theory and to be skillful at the basic operations with sets. It follows the well-known fact that to be a good programmer it is necessary (but not efficient) to be a good mathematician.

A classical example for the use of sets in programming has become the programming realization of the problem for finding prime numbers using the method Sieve of Eratosthenes [2,5,8]. How to construct a faster algorithm, solving the problem for receiving all $n \times n$ binary matrices, which contain exactly k ones in each row and each column, with the help of the set theory and the operations over sets, is shown in [10].

We will examine the entertaining and actual problem, which is very interesting for the students – algorithms for solving Sudoku. It is widespread puzzle nowadays, which presents in the entertaining pages in most of the newspapers and magazines and in entertaining web sites. Sudoku, or Su Doku, is a Japanese word (or phrase) meaning something like Number Place.

On the other side, Sudoku matrices find an interesting combinatorial application, for example in the design theory [7]. The connection between the set of all $m \times m$ permutation matrices (i.e. binary matrices which contain just one 1 in every row and every column) and the set of all $m \times m$ Sudoku matrices, is shown in [3].

As far as the authors of this study know there is not a universal formula for the number of Sudoku matrices σ_n with every natural number n . We consider it as an open problem in mathematics. When $n = 3$ in [4] is shown that there are exactly

$$\begin{aligned} \sigma_3 &= 6\,670\,903\,752\,021\,072\,936\,960 = \\ &= 9! \times 72^2 \times 2^7 \times 27\,704\,267\,971 = \\ &= 2^{20} \times 3^8 \times 5 \times 7 \times 27\,704\,267\,971 \approx \\ &\approx 6.671 \times 10^{21} \end{aligned}$$

in number Sudoku matrices.

PROBLEM FORMULATION AND ALGORITHM DESCRIPTION

Let n is a positive integer and let $m = n^2$. Let $S = (s_{ij})$ is a square $m \times m$ matrix, whose elements in this table are integers, belonging to the closed interval $[1, m]$. The matrix S is divided to n^2 , $n \times n$ square submatrices, which are not intersected and will be called blocks, with the help of $n - 1$ horizontal and $n - 1$ vertical lines (the matrix S , when $n = 3$, is shown on fig. 1).

S11	S12	S13	S14	S15	S16	S17	S18	S19
S21	S22	S23	S24	S25	S26	S27	S28	S29
S31	S32	S33	S34	S35	S36	S37	S38	S39
S41	S42	S43	S44	S45	S46	S47	S48	S49
S51	S52	S53	S54	S55	S56	S57	S58	S59
S61	S62	S63	S64	S65	S66	S67	S68	S69
S71	S72	S73	S74	S75	S76	S77	S78	S79
S81	S82	S83	S84	S85	S86	S87	S88	S89
S91	S92	S93	S94	S95	S96	S97	S98	S99

Fig. 1

Let denote by $A_{kl}, 1 \leq k, l \leq n$ the blocks in the above described matrix $S = (s_{ij})$. Then by definition if $s_{ij} \in A_{kl}$, then

$$(k - 1)n < i \leq kn$$

and

$$(l - 1) < j \leq (l - 1).$$

Let s_{ij} belong to the block A_{kl} and let i and j are known. Then it is easy to guess that k and l can be calculated with the help of the formulas

$$k = \left[\frac{i-1}{n} \right] + 1$$

and

$$l = \left[\frac{j-1}{n} \right] + 1,$$

where, as usual, we denote by $[x]$ the function: whole part of the real number x .

We say that $S = (s_{ij}), 1 \leq i, j \leq m = n^2$ is a **Sudoku matrix** if there is just one number of the set $Z_m = \{1, 2, \dots, m = n^2\}$ in every row, every column and every block.

The puzzle named Sudoku is widespread nowadays. It is given a Sudoku matrix, in which some of the elements are erased. The missing elements will be equal to 0, if we need this. The task in the puzzle is to restore the missing elements of the Sudoku matrix. It is supposed that the authors of the concrete puzzle have chosen the missing elements, so the problem has only one solution. This condition we will miss and will not reckon with it. In this work we will build our programming product to show the number of every possible solution. If the task has no solution this number has to be zero.

The most popular puzzles Sudoku are when $n = 3$, i.e. $m = 9$.

We are going to describe an algorithm for creating a computer program, which finds all solutions (if there are some) of random Sudoku. For this aim we will use the knowledge of the set theory.

We examine the sets R_i, C_j and B_{kl} , where $1 \leq i, j \leq m = n^2, 1 \leq k, l \leq n$. For every $i = 1, 2, \dots, m$, the set R_j consists of all missing numbers in the i -th row of the matrix. Analogously

we define the sets C_j , $j=1,2,\dots,m$ correspondingly for the missing numbers in the j -th column and B_{kl} , $k,l=1,2,\dots,n$ correspondingly for the missing numbers in the blocks A_{kl} of S .

When the algorithm starts working it traverses many times all of the elements $s_{ij} \in S$, such that $s_{ij} = 0$, i.e. these are the elements, which real values we have to find.

Let $s_{ij} = 0$ and let $s_{ij} \in A_{kl}$. We assume

$$P = R_i \cap C_j \cap B_{kl}.$$

Then the following three cases are possible:

- i) $P = \emptyset$ (empty set). The task has no solution in this case;
- ii) $P = \{d\}$, $d \in Z_m = \{1,2,\dots,m\}$, i.e. $|P|$: the number of the elements of P is equal to 1

(P is a set containing one element). Then the only one possibility for s_{ij} is $s_{ij} = d$, i.e. we have found the unknown value of s_{ij} in this case. After this we remove the common element d from the sets R_i , C_j and B_{kl} , and then we continue to the next zero element of the matrix S (if there is such an element);

iii) $|P| \geq 2$. Then we can not say anything about the unknown value of s_{ij} and we move on the next missing (zero) element of the matrix S .

We traverse all zero elements of the matrix S until one the following events occur:

e1) For some $i, j \in \{1,2,\dots,m\}$ is true $s_{ij} = 0$, but $P = R_i \cap C_j \cap B_{kl} = \emptyset$;

e2) All elements in S become positive;

e3) All zero elements of S are traversed, but it does not occur neither event e1, nor event e2. In other words for all the remaining zero elements in S , the above described case iii is always true.

In case that it occurs one of the events e1 or e2, then the procedure stops its work and visualizes the obtained result.

In case that event e3 occurs, then the algorithm has to continue working by using other methods, for example it has to apply the “trial and error” method. In the concrete case this method consists of the following things:

We choose random $s_{ij} \in S$, such that $s_{ij} = 0$ and let $k = \left\lceil \frac{i-1}{n} \right\rceil + 1$, $l = \left\lceil \frac{j-1}{n} \right\rceil + 1$. Let

$P = R_i \cap C_j \cap B_{kl} = \{d_1, d_2, \dots, d_t\}$. Then for every $d_r \in P$, $r=1,2,\dots,t$ we assume $s_{ij} = d_r$. Such an assuming we call a *random trial*. We count the number of all random trials, until the solution is found, in the programming realization of the algorithm. After this we solve the problem for finding the unknown elements of the Sudoku matrix, which contain one element less than the previous matrix. It is comfortable to use a recursion here. The base of the recursion, i.e. we go out of the procedure if there occurs event e1 or e2. It is absolutely sure that it will happen (i.e. there will not be an infinite cycle”), because when we do the random trials we reduce the number of the zero elements by 1.

The above described algorithm is realized in the Pascal programming language, because of the two main reasons:

1) Pascal is chosen, of the most educational institutions in the system of the secondary education, as a first programming language. This means that the basic aim of this work is realized – to help the teacher in his preparation in the process of finding interesting application of the teaching material. That is the way the presented material becomes more interesting for the pupils.

2) There are built in instruments used to work with sets in the Pascal programming language.

The above described ideas can be realized in other random programming language, for example C++. But in this case we have to look for additional instruments to work with sets – for

example the associative containers set and multiset realized in Standard Template Library (STL) [1,5]. It can be used the template class set of the system of computer algebra “Symbolic C++”, programming code is given in details in [9]. Of course can be built another class set, and specific methods of this class can be described, as a training. This is a good exercise, having in mind the fact that the cardinal number of the basic (“universal”) set is not very big. For example the “standard” puzzle Sudoku has basic set the set of the integers from 1 to 9 plus the empty set.

SOME COMBINATORIAL APPLICATION

Obviously, if for solving given Sudoku, there is no need to use the trial and error method, i.e. the problem can be solved by using set theoretical operations (0 random trials), then this Sudoku has a unique solution. Such a Sudoku is shown on figure 2. The opposite proposition is not true, as it is in the next example, shown on figure 3, where there is a unique solution and program has made 218 random trials only to receive this solution, as the number of the random trials (after the final solution is received, the program continues working if it is necessary) is 332.

								3
5		3		8	2		4	1
6	2	4	5				7	
7	8	5		9		1	6	
4		6	2			3		
	3		1	7	6			
1								4
		7	8			2		
3	4	8	6			7	1	5

Fig. 2

7			3					4
	9				1			
	1	2			4			
			9		7	4		
	5			3				
	8			2		1		7
			6					3
			7			5		
8	4						2	9

Fig. 3

Sometimes the Sudoku authors do not examine if the solution is unique. For example the Sudoku, shown on figure 4 (is published in a newspaper’s rubric “Easy Sudoku”, the name of the newspaper we will not mention) has 4 different solutions, the program does 16 random trials until the final solution is obtained, and the common number of the random trials is 18.

The Sudoku puzzle shown on figure 5 has no solution, although on a first sight there is not contradiction in the condition. The program has to make 21 random trials to receive the result that this puzzle has no solution.

		4	9		3	8		
	2		8			9		3
3		8		4			1	
7				9	4	1		
2	4		7					8
	3		2				7	5
8		7		2			3	
	5				7	2		9
		2		3	8		5	

Fig. 4

	5		9		2		4	
2		7				9		8
	4		8	7		3		
	1			2		8		9
5		8		9				3
7			5		3		1	
3		4		5				2
			2		7	4	8	
	7	2			8		3	

Fig. 5

There are even more combinatorial results with different Sudoku matrices, received by the help of the above described algorithm and some experiments.

- There are 283 576 Sudoku matrices of the kind shown on figure 6;
- There are 6 280 Sudoku matrices of the kind shown on figure 7;
- There are 680 Sudoku matrices of the kind shown on figure 8;
- There are 1 728 Sudoku matrices of the kind shown on figure 9;
- There are 22 Sudoku matrices of the kind shown on figure 10;
- There are 8 Sudoku matrices of the kind shown on figure 11;

1	2	3	*	*	*	*	*	*
4	5	6	*	*	*	*	*	*
7	8	9	*	*	*	*	*	*
*	*	*	1	2	3	*	*	*
*	*	*	4	5	6	*	*	*
*	*	*	7	8	9	*	*	*
*	*	*	*	*	*	1	2	3
*	*	*	*	*	*	4	5	6
*	*	*	*	*	*	7	8	9

Fig. 6

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	1	*	*	*	*	*	*
5	6	4	*	*	*	*	*	*
8	9	7	*	*	*	*	*	*
3	1	2	*	*	*	*	*	*
6	4	5	*	*	*	*	*	*
9	7	8	*	*	*	*	*	*

Fig. 7

1	2	3	4	*	6	7	8	9
4	5	6	*	*	*	1	2	3
7	8	*	*	*	*	*	5	6
2	*	*	*	*	*	*	*	1
*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	7
3	4	*	*	*	*	*	1	2
6	7	8	*	*	*	3	4	5
9	1	2	3	*	5	6	7	8

Fig. 8

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*

Fig. 9

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	*	*	*
5	6	7	8	9	1	*	*	*
8	9	1	2	3	4	*	*	*
3	4	5	*	*	*	*	*	*
6	7	8	*	*	*	*	*	*
9	1	2	*	*	*	*	*	*

Fig. 10

1	*	*	7	*	*	4	*	*
*	2	*	*	8	*	*	5	*
*	*	3	*	*	9	*	*	6
7	*	*	4	*	*	1	*	*
*	8	*	*	5	*	*	2	*
*	*	9	*	*	6	*	*	3
4	*	*	1	*	*	7	*	*
*	5	*	*	2	*	*	8	*
*	*	6	*	*	3	*	*	9

Fig. 11

REFERENCES

1. Collins W. J., Data structures and the standard template library. McGraw-Hill, 2002.
2. Dahl D. J., Dijkstra E. W., Hoare C. A. R., Structured Programming. Academic Press Inc., 1972.
3. Dahl G., Permutation matrices related to Sudoku. Linear Algebra and its Applications, 2009, 430, 2457–2463.
4. Felgenhauer B., Jarvis F. Enumerating Possible Sudoku Grids. Preprint available at <http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf>
5. Jensen, K., Wirth N., Pascal User Manual and Report. 3rd ed., Springer-Verlag, 1985.
6. Lischener R., STL Pocket Reference. O'Reilly Media, 2004.
7. Mo Hui-Dong, Xu Ru-Gen, Sudoku Square — a New Design in Field Experiment. Acta Agronomica Sinica, 2008, 34(9), 1489–1493.
8. Price D., UCSD Pascal A Considerate Approach. Prentice-Hall, 1983.
9. Shi T. K., Steeb W.-H., Hardy Y., Symbolic C++: An Introduction to Computer Algebra using Object-Oriented Programming, Springer, 2000.
10. Yordzhev, K., Stefanov S., On Some Applications of the Concept of Set in Computer Science Course. Mathematics and Educations in Mathematics, 2003, v.32, 249-252.