International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:4, 2013

Making Data Structures and Algorithms more Understandable by Programming Sudoku the Human Way

Roelien Goede

Abstract**—**Data Structures and Algorithms is a module in most Computer Science or Information Technology curricula. It is one of the modules most students identify as being difficult. This paper demonstrates how programming a solution for Sudoku can make abstract concepts more concrete. The paper relates concepts of a typical Data Structures and Algorithms module to a step by step solution for Sudoku in a human type as opposed to a computer oriented solution.

Keywords**—**Data Structures, Algorithms, Sudoku, Object Oriented Programming, Programming Teaching, Education.

I. Introduction

DATA Structures and Algorithms is a crucial part of any curriculum in Computer Science or Information Technology. However, often students indicate this module as more difficult than their other modules. The content of a typical module in Data Structures includes: Object Oriented Programming (OOP) concepts such as class design, polymorphism, abstract data types and more; data structures such as arrays, linked lists, queues and more; algorithm design such as sorting and merging algorithms; and algorithm analysis such as asymptotic analysis.

The aim of this paper is to demonstrate how these different concepts of data structures can be mastered by the students while writing a program to solve the traditional game of Sudoku.

This work is suitable for students who are in their second year of programming and are skilled in basic programming. It has been used as assignment in a 15 academic week semester together with parts of the textbook [3]. Addition short programming assignments could be given to ensure mastering of specific topics in the textbook.

The paper starts with a short description of the game of Sudoku in section II. Section III is the main part of the paper that demonstrates how Sudoku can be used to teach data structures. Conclusions and future work is presented in sections IV and V respectively.

The argument presented by this paper is that programming students should write programs to solve Sudoku using "human" methods to improve their algorithm development skills and their understanding of data structures.

R. Goede is with the North-West University, Vanderbijlpark, 1900, Republic of South Africa (phone: +27-16-9103276; fax: +27-86-5839764; e-mail: roelien.goede@nwu.ac.za).

II. Sudoku

Sudoku is a numbers based puzzle where the player must complete a grid of numbers using the numbers 1 to 9. The history of the development of the puzzle may be found at [1]. There is a simple rule: the numbers 1 to 9 may only appear once in each row, column and indicated 3x3 block on the grid. Fig. 1 represents a Sudoku puzzle and its solution from [1]. For reference purposes Fig. 2 presents row, column and block numbers.

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

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

Fig. 1 A Sudoku puzzle and its solution [1]

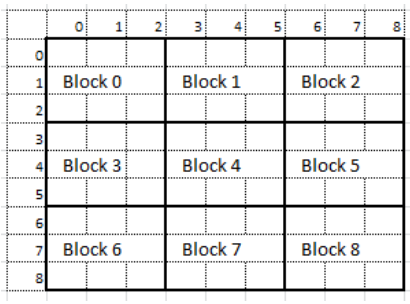

Fig. 2 A Sudoku grid with identification numbers

When using computers, the puzzle can be solved using different methods but the most obvious solution is the brute force method where available numbers are allocated to cells until a dead-end is reached where no more allocations can be made, it then moves back and makes a different allocation (full description in [2]).

Humans solve Sudoku in a much more fun and intuitive way. People set up rules or strategies for themselves to find the values of a specific cell. These strategies can be very complex to explain, but easy to do. One such example of a strategy is to view the blocks in groups. Blocks 0, 3 and 6 can be viewed as a block-column. Any specific number – say 8,

(from Fig. 1) must then be present somewhere in column 0 and somewhere in column 1 and somewhere in column 2, but only once in each block. When this strategy is applied, one notices that there is already an 8 in columns 0 and 2. There must be one in column 1 and it must be in block 6, since blocks 0 and 3 already contains an 8. The 8 must be either in row 7 or 8 in block 6. Since there is an 8 in block 7, row 8, the 8 should be in row 7 of block 6. This is an example of how complicated we can make the strategies. This strategy can be extremely difficult to program.

Exactly the same can be achieved with a much simpler strategy: Each column should have one instance of each number. When column 2 is checked, it is clear that there may not be an 8 in block 0 or block 3 as they already contain 8's. Row 8 cannot be an 8 so there is only one cell in the column where 8 can be possible and that is row 7. This second rule is much simpler than the first strategy but the first might be quicker to check for some people.

This paper aims to demonstrate how programming concepts can be used to programme these human methods in order to help students develop their data structures and algorithm skills.

III. Using Sudoku to Teach Data Structures and Algorithms

The aim of this section is to present detail on how the human way of solving Sudoku can be used to facilitate mastering of OOP concepts and algorithm design. In learning OOP, the student focuses so much on theory that many students fail to develop the skills to develop good algorithms.

When programming Sudoku using human methods as opposed to computer methods such as brute force, students develop the skill to identify how their brain works in general terms and how to "teach" a computer to perform the same activity in the form of a program.

The program based on a "human" solution can then be analysed by students using formal analysis methods and students can compare it with more "computer oriented" solutions in order to better understand why we use different strategies when solving some problems as humans than we use when programming computers.

The following sections present the detail (Java implementation) of the proposed Sudoku assignment. The discussion starts with an explanation of the initial data structures used by the solution.

A. The Cell Class

The first data structure to be used in this solution is a class. Each cell in the Sudoku grid becomes an object of the cell class. This demonstrates the relationship between classes and its related objects to the students. The overall strategy is to use a nine position array in each cell to indicate possible values for that cell. Initially all 9 digits 1-9 are possible, but as allocations are made some of these values are eliminated. The Java class can be given initially as:

```
public class SudokuCell
{
intcell_row, cell_col, cell_value;
int [] possible_values;
}
```

A constructor is then used to assign 0 to the first 3 variables and the numbers 1 to 9 to the array. Accessor and mutator methods are also added to this class.

B. The Puzzle Grid

In a separate class, which can be understood as the puzzle solving class, a two dimensional (9x9) array is used containing the objects of the cell class. In Java terms this can be initialised as:

```
public void clearBoard() {
for (int row = 0; row<9; row++)  {
for (int col = 0; col <9; col ++)
board[row][col]= new SudokuCell();
        }
 }
}
```

The data structure board [][] is used as a class variable within scope of the entire class.

The total solution strategy is kept separate from the single cell class as these are two separate entities.

C. The Basic Strategy: Assignment 1

The basic strategy of this solution is to update the possible values array in each cell and to make an allocation when only one value is possible for that cell. After any allocation was made, the possible values of all the affected cells are updated again. An allocation of a value in a cell affects the possible values of all the cells in the same row, column and block as the allocated cell. The cell class contains a mutator to set a specific value in the possible values array equal to zero.

This process of updating the possible values of the cells after an allocation is a good exercise in nested loop programming for the students. The given grid (left side of Fig. 1) is seen as specific allocations to be made. Fig. 3 contains a depiction of the cell in row1, column 1 after the initial allocation was made.

possible_values index	0	1	2	3	4	5	6	7	8
possible_values values	0	2	0	4	0	0	7	0	0
cell_row	0								
cell_col	0								
cell_value	0								

Fig. 3 Variable values for cell (1,1)

The fact that Java starts array indices at 0 rather than 1 complicates the implementation, but is a good learning experience for the students.

The main strategy is to write a method that counts the possible values in the array that are not equal to 0 (this method is in the cell class). From the solution class the method is called for each cell in a nested loop with an exit condition if one is found. If a cell is found with only one possible value the allocation is made and the affected cells need to be updated.

The implementation up to this point can be a good first assignment for the students. They can be provided with a number of puzzles that are solvable by this strategy – these are easy puzzles but the student feels a true sense of achievement by creating this program. This method has been used in ten successive semester courses and almost all the students in the Data Structures module (who completed introductory programming) are successful in this first assignment.

Students quickly discover that they need to add some sort of variable value presentation to their code in order to find logical errors. They add a method to the cell class that displays all the values for the variables on screen. At this point most students develop a more graphical display for their solution process.

Up to this point the following concepts of data structures and algorithms have been applied:

- Class and object design
- Reusability of classes
- One and two dimensional arrays
- Variable tracking / debugging

It is possible to handle some of the arrays with linked lists – One could give the exercise to replace the *possible_values* array with a linked list. Students then develop linked list access methods that traverse the linked list. This leads to a good understanding of the difference between direct and sequential access to elements! It has to be said that one should provide students with additional assignments to illustrate the advantages of linked lists over arrays.

At this point in the semester the Java solution should be explained to the students who did not succeed in this first assignment. Each assignment is evaluated in terms of code but the student also needs to explain the functioning of the code to ensure honesty and fairness

D.Assignment 2

After completing assignment 1 (setting up the data structures and updating the possible values for the initial setup) the students have an understanding of the classes involved. Many students struggle to develop the loop that manages the solution. A loop is needed as each allocation influence the state of the puzzle. The loop should terminate if no changes were made to the puzzle or it is solved.

After assignment 1 most students realise that their solution can only solve the most elementary Sudoku puzzles and they start thinking how it can be improved. This is where the success of this assignment is most evident. Students are seen working on printed puzzles, they are trying to figure out what they do to solve the problem. From this point some of the students work without any guidance. Students are given time to develop their own ideas, but are also given fixed assignments in order to practice certain theoretical concepts.

Assignment 2 involves traversing the rows, columns and blocks and search for numbers what can only be allocated to one cell in a specific block, row or column. This strategy was explained in section II of this paper. From a programming strategy, it involves nested loops where each row, column and block is traversed. The *possible_values* array of each cell in a row (column or block) is checked for each number from 1 to 9 to find numbers that only appear once in the row (column or block). Most students do this for one number (1-9) at a time, but one may use and array and do it for all 9 numbers at once.

The most challenging part is to decide how to traverse a specific block. There are two main strategies. One may compute the applicable cell indices or one could create a table with all the indices of the cells in a specific block. The advantage of the second strategy is twofold: First, a three dimensional array is used to store the indices (a 2D table for each block's row and column values). Students have a fear of higher dimensional arrays since they struggle to visualise them and are presented here with an example of such an array. The second advantage is that the code to traverse the block is very similar to that of traversing the rows or columns since it is not cluttered with index computations. This illustrates the advantage of good consistent programming style to the students.

Students need to be supplied with puzzles that are solvable with this strategy alone, and once again the feeling of achievement is noticeable when they succeed.

E. Assignments 3

When one solves Sudoku there is a special rule necessary to eliminate possible values which form the basis for Assignment 3. It is as follows: if all the possible values of a specific number within a block appear in the same row or column, that number cannot appear anywhere else in that row or column (i.e. in other blocks). In order to simplify the implementation of this rule an additional field is added to the cell class (given in Section III (A)) for the block number of the specific cell. At this point most of the students improved their skills in such way that they are motivated to take control of the entire solution. Students who still need guidance receive assignments 4.

F. Assignment 4

Most people who solve Sudoku puzzles regularly, use the "pairs" rule to make the final allocations. The "pairs" rule is about finding pairs of cells that has the same two values as only possible values. The pairs have to be in the same row, column or block. If such a pair exists, the two numbers making up the pair cannot be anywhere else in the specific row, column or block.

If a student is able to develop an algorithm for this rule, it can be said that that student mastered the skill of developing algorithms that are using multidimensional arrays, classes and objects. At this point most Sudoku puzzles can be solved by the program.

G.Assignment 5

However, some most challenging puzzles needs the solver to guess a value (choose between 2 alternatives) and see if it leads a legal solution. Students need to develop a "test solution" method. It is at this point where students start to enquire about the feasibility of brute force solutions since the algorithm required to test an allocation is a basic building block of the brute force method.

H.Assignment 6: Polymorphism

After assignment 5 the Sudoku solver is complete, now the students could be asked to rework the entire solution to function with alphabetical characters instead of digits. At this point they have mastered the theory of abstract methods and classes. The textbook of Bruno Preiss [3] is recommended reading in this regard. This assignment develops insight in re-usability of code for different data types by using polymorphism.

I. Assignment 7: Runtime Analysis

At the end of the semester, students are used to analyse the running time of algorithms as it is used frequently in their textbook [3]. In assignment 7 they have to analyse the running time of their solution compared to a brute force solution for Sudoku. At his point they develop a full understanding that some problems are solved in a total different manner when using a computer than what we as humans use. Most students in computer science also take a module covering linear programming, and they realises that instead of all the programming they did, one can view Sudoku as an Integer Programming problem [4]. Solving the puzzle is not the main focus of the assignment; developing of algorithm for something one is used to doing by hand is.

IV. Conclusion

In the first meeting session of the semester, students are surprised when a Sudoku puzzle is handed out to them for completion then and there. They are excited about the prospect of having to program the solution themselves. The problem of Sudoku is complex enough that students soon realise that it might be more difficult than they expected and that the complexity is on a much higher level than the assignments of their first year of programming.

Most first year programming assignments focus on laying a good foundation in terms of programming language constructs but are too simple to develop algorithm design skills. Students often feel after their first year that they have a lot of programming "tools" but they have not yet used it to create a solution to a problem. Quite often, they do not know where to start when confronted with a more challenging problem.

This paper presents an assignment for students in data structures and algorithms that use the many of OOP concepts and data structures to solve Sudoku puzzles. The assignment is to program a solution that is intuitive to the human mind and not the typical brute force approach associated with computers. In doing so, the students gain experience in analysing their cognitive processes and relaying these to computers in the form of algorithm development. The assignment is divided in smaller assignments. The final assignment is used to demonstrate to the students that some problems like Sudoku can be addressed on methods suitable for computers that are different from human cognitive problem solving.

V. Future research

Informal feedback received of students later in their lives has always been positive: they relate how this assignment gave them confidence. As future research a formal study will be done under alumni to better understand the role of this assignment in forming them as programmers.

References

[1] Anonymous, Sudoku, *Wikipedia*, accessed on 31/01/2013 at http://en.wikipedia.org/wiki/Sudoku.

[2] K. van der Bok, M. Taouil, P. Afratis & I. Sourdis, "The TU Delft Sudoku solver on FPGA," *Field-Programmable Technology, 2009. FPT2009. International Conference on* , vol., no., pp.526-529, 9-11 Dec.

[3] B.R. Preiss, Data structures and algorithms with object-oriented design patterns in Java. Wiley: New York, NY, 2000.

[4] Bartlett AC and Langville AN. An Integer Programming Model for the Sudoku accessed on 31/01/2013 at http://langvillea.people.cofc.edu/Sudoku/sudoku2.pdf?