

**DEFINING ALGORITHMS FOR PRODUCING, SOLVING AND CHECKING A SUDOKU INTERFACE WITH EFFICIENT COMPLEXITIES**

Matriculation Number – 40542237

Module Number - SET08122

Author Email: [40542237@live.napier.ac.uk](mailto:40542237@live.napier.ac.uk)

Pages of interest – 3 (3, 4, 5)

Table of Contents

1. **Introduction3**
2. **Implementations with Examples3**
3. **Features and Plan4**

**3.1 Language of choice : Java4**

**3.2 Parallel Streaming4**

**3.3 Tree Architecture (Data Structure)5**

1. **Conclusion6**
2. **Appendix15**
3. **References24**
4. Introduction

A Sudoku game is a board/grid based game usually consisting of a 9x9 architecture, thus having 81 cells. This paper will outline the three main requirements of the game and how to produce a solution to said needs. The first is to generate a random Sudoku grid that is almost always different. The second requirement is to allow the player to play the game by intentionally leaving out some cells. (The number of cells appearing on the screen will determine the difficulty of the puzzle). The third will deal with checking if the values are valid and if the puzzle has been solved. The additional requirements include undo-redo functionalities, game records and irregularly shaped or sized grids.

The purpose of this research is to differentiate a variety of approaches to solve certain problems and compare them in fields of time and space complexity and to determine which way is the best to solve the Sudoku problem. This research will also deal with algorithms and data structures require to achieve the determined goal as stated above. This will include additional software and data architecture so that the Sudoku application will run with the highest efficiency with minimal time and space wastage. The paper will also deal with the language comparison and will state why the author had chosen Java-Maven as the development environment for the project.

1. Implementations with Examples

During the research, most findings failed to meet all the requirements of the current project, thus multiple bits of software designs from multiple projects online have to be combined to reach a working solution for the coursework. Some of the inspiration for the solution is of the Interface and some of the algorithms used in said projects.

The solution to the first requirement which is to generate a random Sudoku puzzle is influenced by the article on StackOverflow. The simple way of doing this is to take a random existing valid Sudoku board and shuffle it up. (User: Doc Brown, Sep 2, 2011). With a simple skim of this answer for creating unique Sudoku Grid on the website, one could state that a list (array) of random numbers can be a seed to produce the whole puzzle. The trick is to move three adjacent columns of the next row forward or backwards for the first three rows. Then switch up the order of the each three-columns in the sub-grid so that a duplicate is not made on the 4th row and continue with the algorithm. Such method is used in commercial Sudoku Apps including [▻Sudoku (PeopleFun CG, LLC).](https://apps.apple.com/us/app/sudoku/id366247306) The next step is to shuffle the columns and the rows, so that guessing the number pattern is basically near impossible. While doing this shuffle, it is easier to divide the number of rows in 3s and use the quotient and incrementing the for loop which will have the complexity of O (n/x) (where x = rows/3) and then access via currentRow + 2 or +3. Such procedure must be enwrapped in try/catch to prevent the application from crashing when producing irregular grids. Full Algorithm is explained in detail at [zinhtun.github.algo.txt](https://github.com/zin-lin/Sudoku/blob/master/algo.txt). (Appendix A).

For the second step of the project, the interactive UI had been influenced by the C++ application written on GitHub ([@flyingpeakock](https://github.com/flyingpeakock/Console_sudoku), Jun 12, 2021) (see Appendix B). The Change in color of the console and UTF – 8 is supported in IntelliJ, which is a convenience since the project is to be developed in Java.

The Third requirement consists of row and column checking using Tree datatype which will be discussed in details in section 3.3.

1. Features and Plan
   1. Language of choice: Java

Java is chosen as the development language for the project and Maven is chosen as the standard packaging for the application. This allows one to use all the necessary packages in the engineering environment in the Object Oriented Way. Most projects during the research seem to manipulate memory using pointers based raw C or C extension (C++). For this project Java is chosen due to the versatility of OOP using Object Referencing which does not need any copy (not even the pointers, but reference counting is used) while passing through methods. Referencing is thus CPU expensive but still help a lot in fields of developer productivity and is very useful in building complex engineering systems; one like this. Another advantage of using Java is the fact that Java has Automatic Garbage Collection that runs on a separate thread (M. Schoeberl, 2006), providing developers with memory-leak free applications, unlike C/C++ where memory management is mandatory (Appendix C). It also seemed that some C++ Console Support for Visual Studio 2022 lacks extended ASCII character, while Java can package encoding or IntelliJ Console will deal with UTF 8 support.

* 1. Parallel Streaming

Another major advantage of Java and JVM (JAVA VIRTUAL MACHINE) is that this allows one to process the application as per power of hyper-threading of the CPU that is utilised. The only drawback of using parallel streaming is the data can be corrupted. All necessary processes are running parallel, thus changing the data of a variable during the loop can cause serious damage to one’s application. This method also produces process terminations in random orders. (Michael Müller, 2016). Parallel Streaming also is not thread safe, thus might skip some processes, ending up with null values in arrays if an insertion should happen within the loop. The saving grace though is the existence of CopyOnWriteArrayList <Template>, which is part of the Collections Framework and with such data-structures, safe parallel streaming is possible and thus is used in randomising the initial row of the Sudoku grid with faster write access.

* 1. Tree Architecture

A binary search tree is a very useful resource in building very complex systems without memory duplications and faster access time, promoting both time and space complexities. This is the architecture that is designed with author’s prior knowledge from the module: Programming Fundamental. A tree divides the insertion into larger and smaller numbers thus have the time complexity of O log (n). A tree can also only have unique values, since a node cannot be inserted twice. It is either left (smaller) or right (larger). A three-nodes-tree can exist, but would not favor the situation of the Sudoku Grid Check. A grid check can be done by adding numbers that are already present in a row and a column as such if the number is already present in the not available list for the row, it doesn’t need another insertion for the column saving a lot of space. This could also be utilized for the undo and redo functionalities of the Sudoku Application.

1. Conclusion

To summarise, the use of Object-Oriented Programming Language like Java favours developer productivity by a huge margin when compared to pre-existing Sudoku projects/repositories in lower-level procedural programming languages. The increase in productivity results in focus of more interesting features like the use of efficient Trees and Nodes architectures and Parallel Streaming and algorithms for producing random Sudoku grid. All in all, with the research done and the project planned out with the components discussed, the Sudoku Application Supporting different levels of difficulties, sizes and shapes will be ready in a few months’ time.

1. Appendices

Appendix A

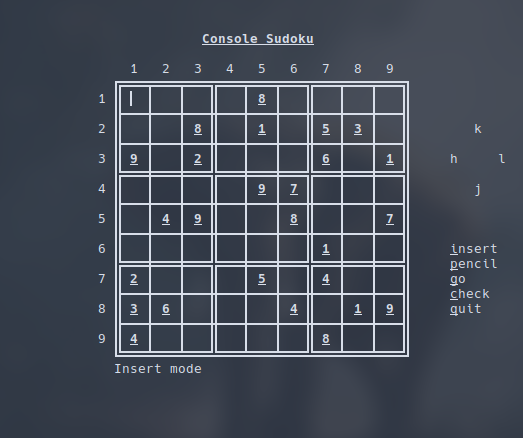
Text is copied from algo.txt at [zinhtun.github.algo.txt](https://github.com/zin-lin/Sudoku/blob/master/algo.txt).

# First Generation

6 5 7 4 3 1 2 8 9 #let’s say this is what we start off with  
  
4 3 1 2 8 9 6 5 7  
  
2 8 9 6 5 7 4 3 1  
  
  
7 6 5 1 4 3 9 2 8  
  
1 4 3 9 2 8 7 6 5  
  
9 2 8 7 6 5 1 4 3  
  
  
5 7 6 3 1 4 8 9 2  
  
3 1 4 8 9 2 5 7 6  
  
8 9 2 5 7 6 3 1 4  
  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
  
3 1 4 2 8 9 7 6 5  
  
8 9 2 6 5 7 1 4 3  
  
5 7 6 4 3 1 9 2 8  
  
  
6 5 7 1 4 3 8 9 2  
  
2 8 9 7 6 5 3 1 4  
  
4 3 1 9 2 8 5 7 6  
  
  
7 6 5 3 1 4 2 8 9  
  
1 4 3 8 9 2 6 5 7  
  
9 2 8 5 7 6 4 3 1  
  
-----------------------------------------------------  
  
// Algorithm step - 1  
// Switch x, y  
  
// Switch Columns  
// 1st quadrant 0 with 2  
// 3rd quadrant 6 with 8  
  
// Switch Rows  
/\*  
  
for +1 rows switch 0 - 2  
for +2 rows switch 1 - 2  
for +0 nae switch.  
  
x ++;  
\*/  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Appendix B

@flyingpeakock’s Design



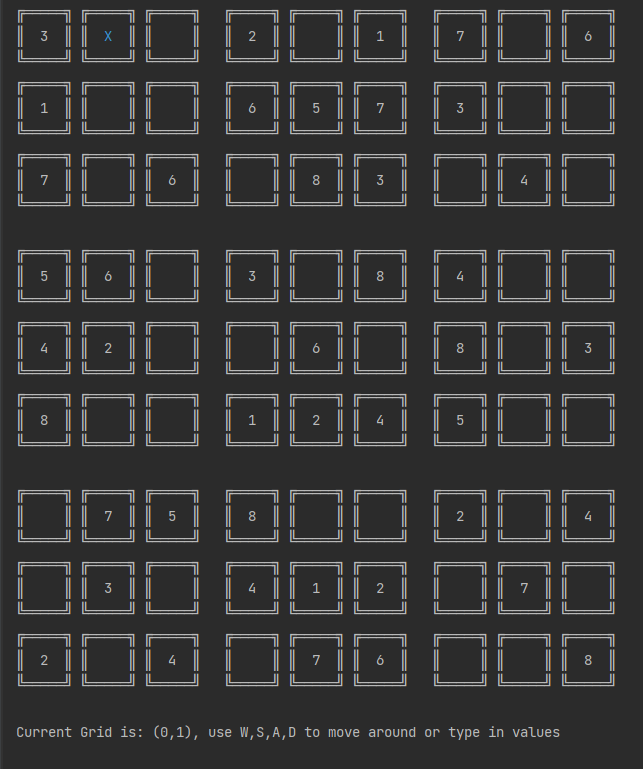


Figure: Author’s Version of the Application

Appendix C

Java vs C/C++: Memory Manipulation/Management