Combinatorics and Graphs

Sudoku Puzzle

Kim Hue Nguyen1

1.railykim469@gmail.com ,Ton Duc Thang University, 19 Nguyen Huu Tho, Tan Phong, 7 Disctrict, Ho Chi Minh City, Viet Nam

**Abstract :** Since its founding in 1979, Sudoku has experienced rapid popularity and global attention. We present an algorithms that create puzzles for popular Sudoku puzzles by using Latin Squares is to build a valid solution grid, and then construct a puzzle that leads to that solution from an empty grid. Using these transformations, we can convert a puzzle into more than 1 trillion, further expanding the efficiency of our generators. This article is on the implementation of a Sudoku game in Java using permutation.

**Index Terms :** sudoku puzzle, latin square, permutation

1. **Introduction**

The word Sudoku is Japanese and is composed of two parts: Su- meaning ‘number’, and Doku- meaning ‘single’. Rightfully so, as Sudoku is a puzzle where the objective is to fill a 9×9 square grid with digits numbered 1 to 9, so that each column, each row, and each of the nine 3×3 sub-grids contains all of the digits, but absolutely **no** duplicates. The puzzle starts out partially filled in i.e some of the numbers are given to you as a starting point as shown for example above. It’s quite the mentally stimulating game and is often featured in newspapers, right beside the crossword.[2]

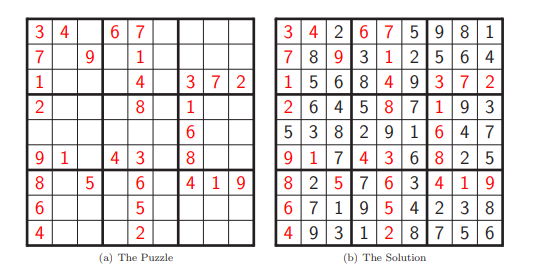
The puzzle consists of a 9×9 grid in which some of the entries of the grid have a number from 1 to 9. A traditional Sudoku puzzle 9 cells divided into 3×3 subsections called blocks. A Sudoku solution must satisfy the rules of Sudoku following:

* Every cell contains a single number between 1 and 9
* Every row contains each of the numbers 1-9 exactly once
* Every column contains each of the numbers 1-9 exactly once
* Every box contains each of the numbers 1-9 exactly once

A Sudoku puzzle is a Sudoku grid with some of the numbers filled in, called givens. The givens in a valid Sudoku puzzle must satisfy the following conditions:

* The givens do not violate any of the above rules of Sudoku
* There must be a single Sudoku solution that contains the set of givens, i.e. the set of givens must lead to a unique solution.

Completed games are always a type of Latin square with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column, or any of the nine 3×3 subregions of the 9×9 playing board.



*Figure 1: Sudoku Puzzle and Solution*

(Source: Sudoku:Bagging a Diﬃculty Metric & Building Up Puzzles)

Since its inception in 1979, Sudoku has experienced rapidly rising popularity and global interest. Sudoku puzzles are simple to understand, simple to try, though often not simple to solve. In this paper, we will build in Sudoku puzzle by using permutation.

1. **Related Work**

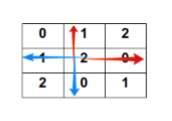
In this Assignment #1 you must build in Sudoku puzzle by using permutation and output to ﬁle including solution and puzzle for Sudoku.

* The main class (containing the main function) must be named as SudokuGenerator.java for generating Sudoku.
* Your program must take command line arguments and must follow this order of ﬁrst your’s program, puzzle ﬁle name, solution ﬁle name and the number of blanks.

1. **Approaches**

We start with a valid solution and generate a puzzle leading to that solution instead of picking random numbers to seed a blank puzzle.

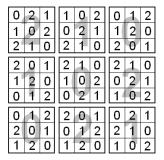
* 1. **Latin Squares for Solution Generation**
* Select nine 3×3 Latin Squares, with replacement.
* Place each of these squares into one of the blocks in a blank grid. In this step, you must guaratee that do not any duplicated number on row and column matrix as ﬁgure below.



***Figure 2****: Check Block*

(Source: Assigment#1**,** Combinatorics and Graphs**,** Ton Duc Thang University)

* Select another 3 × 3 Latin Square and match each cell with the corresponding block in the Sudoku grid. The same as the previous step, you must guaratee that do not any duplicated block in all blocks except outside block as Figure (3).



***Figure 3****: Selection of Lantin Square*

(Source: Sudoku**:**Bagging a Diﬃculty Metric & Building Up Puzzles)

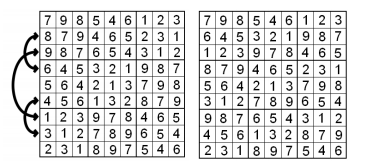
* Each cell now has a pair of numbers. Treat these pairs as base 3 numbers, and convert to base 10, adding 1 (Figure (4)). For example, with the block following:



***Figure 4****: Conversion to Base 10*

(Source: Sudoku**:**Bagging a Diﬃculty Metric & Building Up Puzzles)

* Each cell now has the numbers 1-9. However blocks contain duplicates. So that you will swap the 2nd & 4th rows, 3rd & 7th rows, and 6th & 8th rows preserving the row and column properties, and adding the desired property for blocks. Finally, we will obtain a valid solution as Figure (5)

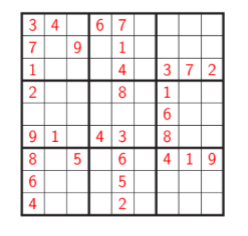


***Figure 5****: Sudoku Solution*

(Source: Sudoku**:**Bagging a Diﬃculty Metric & Building Up Puzzles)

* 1. **Digging hole**

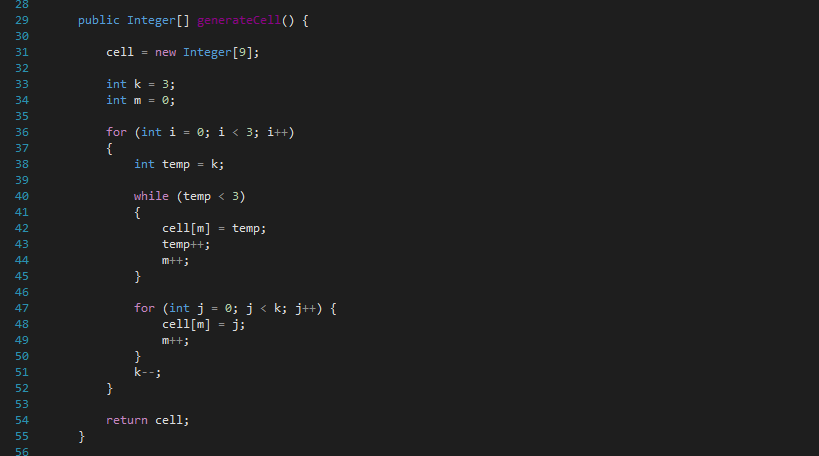
After generating solution for Sudoku puzzle, you will dig some holes to make Sudoku puzzle. The number of holes belong to argument passed from command line and you must guarantee the number of solutions for a set of givens to ensure uniqueness. For example, you will obtain Sudoku puzzle as following with the number of holes is 50.



***Figure 6****: Digging holes*

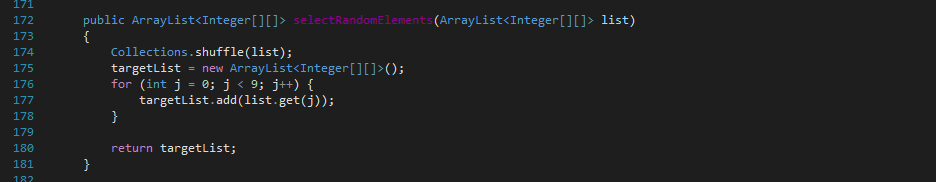
(Source: Assigment#1**,** Combinatorics and Graphs**,** Ton Duc Thang University)

1. **Experiments and Results**



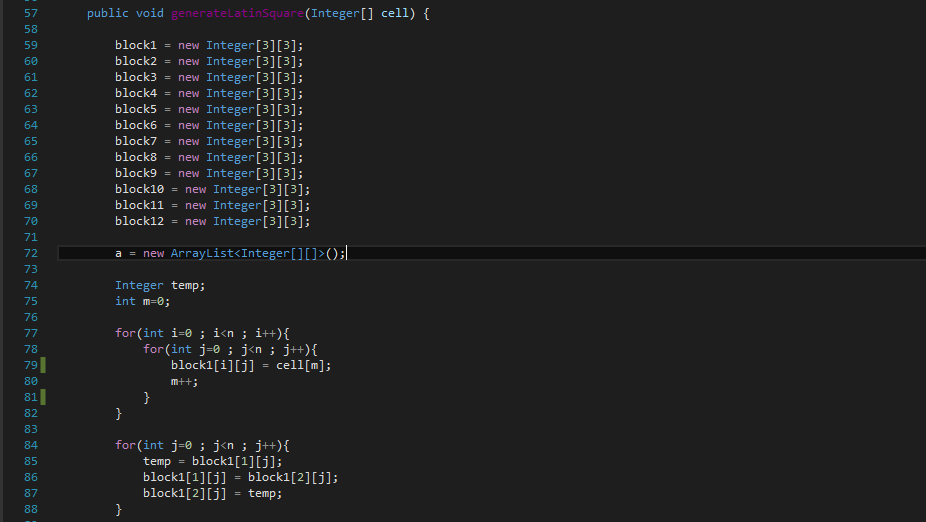
(Source: SudokuGenerator.java)

***Figure 7:*** *Generate cell*



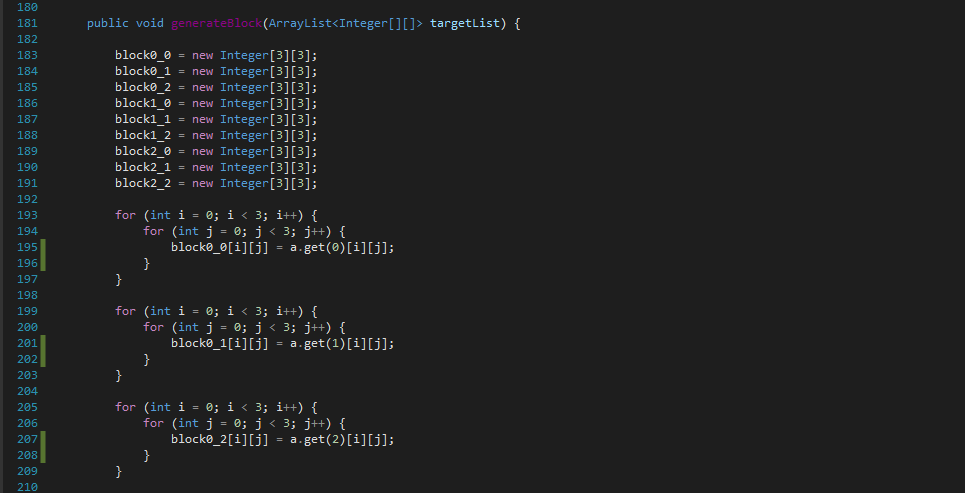
(Source: SudokuGenerator.java)

***Figure 8:*** *Select 9 random elements*



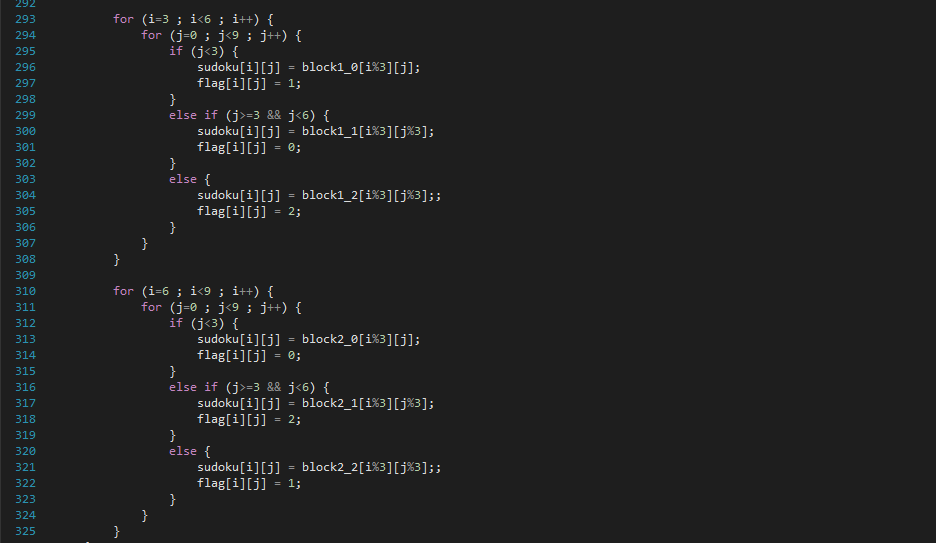
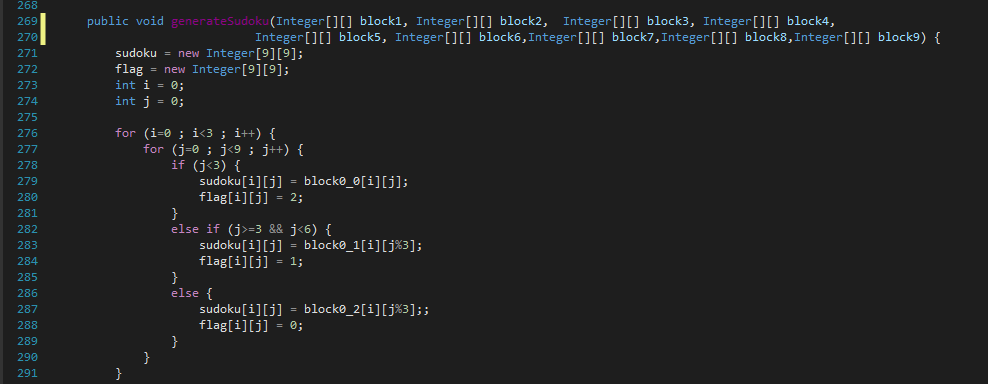
(Source: SudokuGenerator.java)

***Figure 9:*** *Generate latin square*



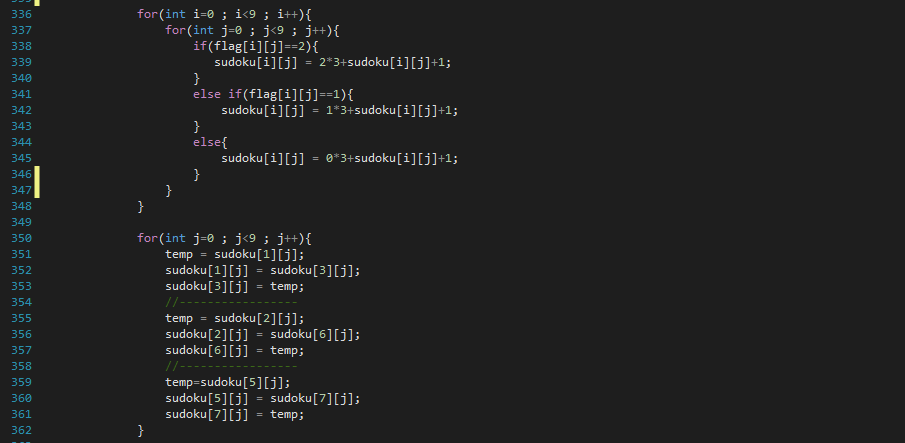
(Source: SudokuGenerator.java)

***Figure 10:*** *Generate block*



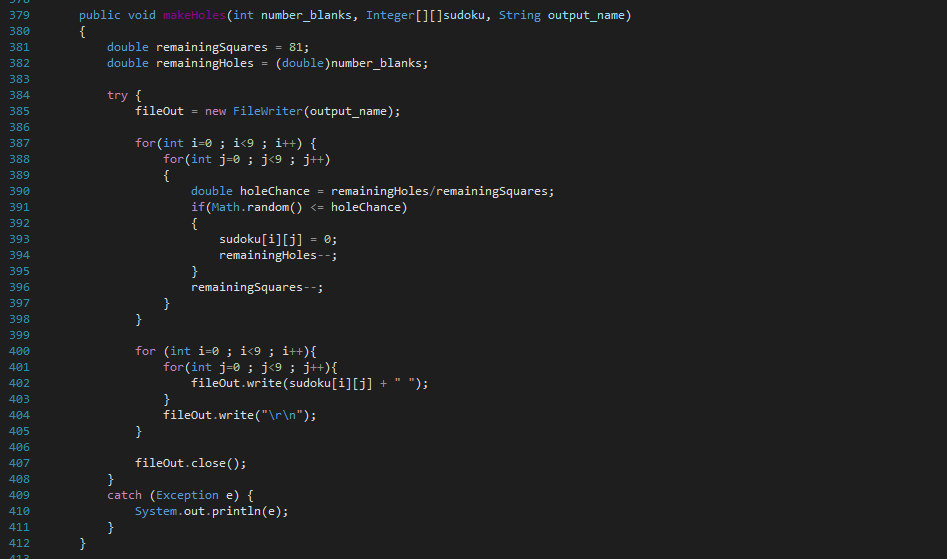
(Source: SudokuGenerator.java)

***Figure 11:*** *Generate sudoku*



(Source: SudokuGenerator.java)

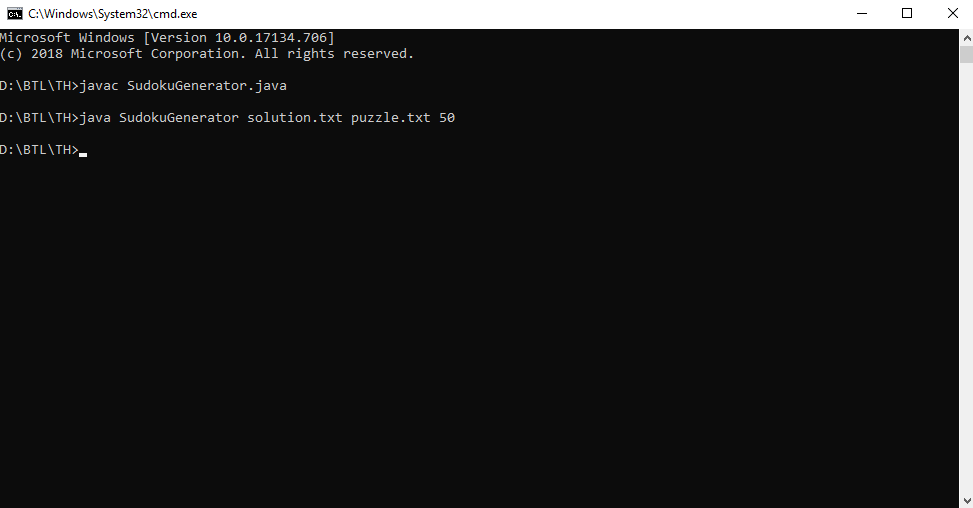
***Figure 12:*** *Convert to base 10 and swap row*



(Source: SudokuGenerator.java)

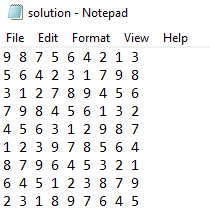
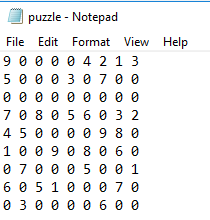
***Figure 13:*** *Digging holes*

Build :



Result :

- solution.txt - puzzle.txt

1. **Conclusion**

Sudoku puzzles are simple to understand, simple to try, though often not simple to solve. We present an algorithms that create puzzles for popular Sudoku puzzles by using Latin Squares is to build a valid solution grid, and then construct a puzzle that leads to that solution from an empty grid. Using these transformations, we can convert a puzzle into more than 1 trillion, further expanding the efficiency of our generators. This article is on the implementation of a Sudoku game in Java using permutation.

**References**

[1] Tem 2280, “Sudoku**:**Bagging a Diﬃculty Metric & Building Up Puzzles”, 2008.

[2] Tem 2280, “Sudoku**:**Bagging a Diﬃculty Metric & Building Up Puzzles”, 2008.

[3] Sudoku, Available at https://en.wikipedia.org/wiki/Sudoku

[4] Assigment#1 Combinatorics and Graphs**,** Ton Duc Thang University, 2017