Ton Duc Thang University
Faculty of Information Technology

Combinatorics and Graphs

Module code: 502042
Assignment # 1

March 4, 2019

0 Objective

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

1 Sudoku problem

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:

- Numbers in rows are not repeated
- Numbers in columns are not repeated
- Numbers in 3×3 blocks are not repeated
- Order of the numbers when filling is not important

On the other hand, 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.

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

(a) The Puzzle

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

(b) The Solution

A sample Sudoku puzzle and Solution

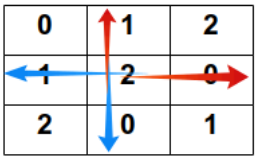
2 Build in Sudoku puzzle

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

2.1 Latin Squares for Solution Generation

To quickly and effectively generate a solution, we use the 12 unique 3×3 Latin Squares.

- Select nine 3×3 Latin Squares, with replacement. You could define a **Block object** to describe for each block of sudoku
- 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.



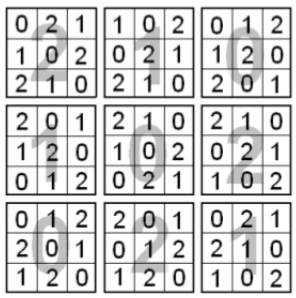
- Select another 3 × 3 Latin Square and match each cell with the corresponding block in the Sudoku grid. The same as the previous step, you must guaratee that do not any duplicated block in all blocks except outside block as Fig (a).
- Each cell now has a pair of numbers. Treat these pairs as base 3 numbers, and convert to base 10, adding 1 (Fig (b)). For example, with the block following:

0	2	1
1	0	2
2	1	0

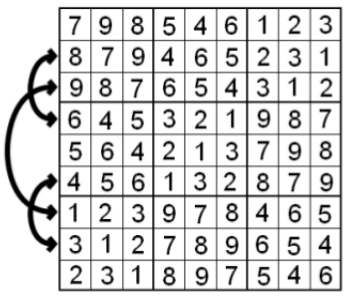
You will obtain the first row as below:
$2 \times 3 + 0 + 1 = 7$
$2 \times 3 + 2 + 1 = 9$
$2 \times 3 + 1 + 1 = 8$
the second row:
$2 \times 3 + 1 + 1 = 8$
$2 \times 3 + 0 + 1 = 7$
$2 \times 3 + 2 + 1 = 9$
and the third row:
$2 \times 3 + 2 + 1 = 9$
$2 \times 3 + 1 + 1 = 8$
$2 \times 3 + 0 + 1 = 7$
Finally, you will have the result as following:

7	9	8
8	7	9
9	8	7

- 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 Fig (c)



(a) Selection of Latin Squares



(b) Conversion to Base 10

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

(c) Sudoku Solution

2.2 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.

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

3 Instructions

3.1 Requirements for your program

- The main class (containing the main function) **must be named as SudokuGenerator** for generating Sudoku and **SudokuSolver** for Sudoku solution
- 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.
  The format for *puzzle1.txt* and *solution1.txt* as following:
  - *solution1.txt*

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

  - *puzzle1.txt*

```
7 8 0 0 0 0 0 0 0
1 0 2 0 7 8 5 4 6
0 0 0 0 2 3 9 8 0
9 0 0 0 0 0 0 2 1
0 1 3 7 0 9 0 5 4
6 0 0 0 3 0 0 0 8
8 9 7 5 4 0 0 0 2
3 0 0 8 0 7 0 6 5
0 5 6 3 1 2 0 0 0
```

These blanks will present by "0"

- *Notes:* Student will be recieved zero if:
  - Program that **do not compile**.
  - Program that **work only on your machine**.
  - Program that **do not run by command line arguments and your program name is different with requirement**
  - Your code is the same with another student.

3.2 Submission

- Submit your project through website of course: *sakai.it.tdt.edu.vn*
- **Deadline**: **April 23, 2019**

4 References

1. Khee-Meng Koh, Chuan Chong Chen, [1992], Principles and Techniques in Combinatorics, World Scientific Publishing Company.
2. John Harris, Jeffry L. Hirst, Michael Mossinghoff, [2008], Combinatorics and Graph Theory (2nd Edition), Springer.
3. C. Vasudev, [2007], Combinatorics and Graph Theory, New Age International Pvt Ltd Publishers.
4. Miklos Bona, [2011], A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory (3rd Edition), World Scientific Publishing Company.
5. Alan Tucker, [2012], Applied Combinatorics (6th Edition), Wiley.
6. Sudoku: Bagging a Difficulty Metric & Building Up Puzzles