CSP to solve Sudoku

Abdullah Elsayed Ahmed 7459
Mohammad Ashraf Hamdy 7508
Samah Abdelaziz Draz 7889

April 27, 2024

Contents

1 Introduction

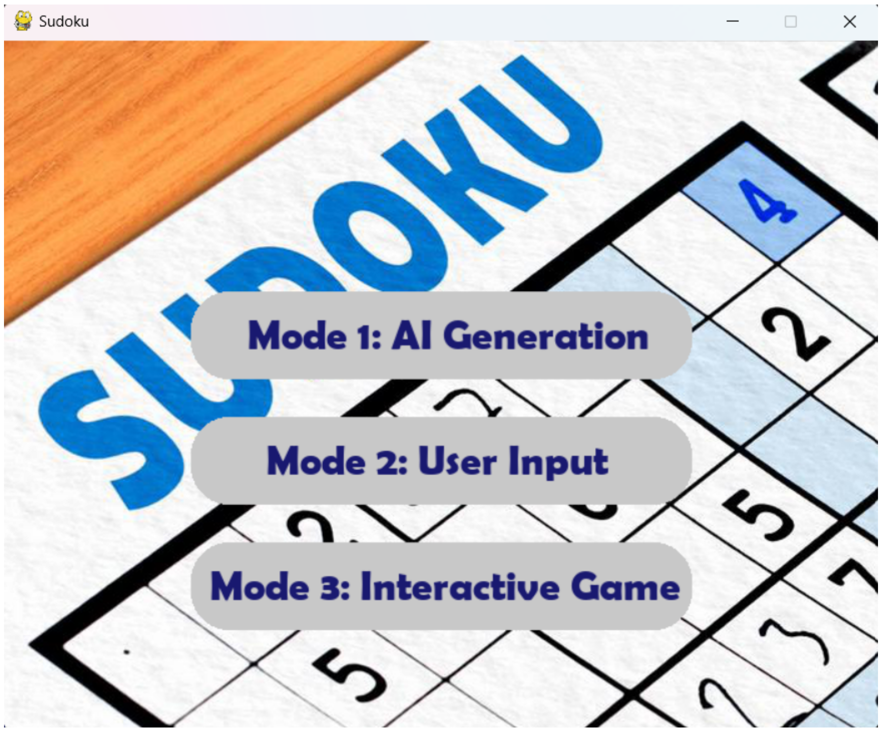


Figure 1: Sudoku game GUI

Sudoku can be formulated as a Constraint Satisfaction Problem (CSP) with the following components:

- **Variables**: Each cell in the Sudoku grid represents a variable that needs to be assigned a value from the domain $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Hence, a standard Sudoku puzzle has 81 variables.
- **Domains**: The domain of each variable is restricted to the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ because each cell can only hold one digit.
- **Constraints**: The constraints in Sudoku are that no two cells in the same row, column, or block can have the same value. This constraint ensures that each digit appears exactly once in each row, column, and block.
- **Solution**: A solution to the Sudoku CSP is an assignment of values to all variables (cells) such that the constraints are satisfied.

To solve Sudoku as a CSP, various algorithms can be applied, such as backtracking search with constraint propagation techniques like forward checking or maintaining arc consistency. These algorithms systematically explore the search space of possible assignments until a valid solution is found.

2 Data structure

Array

Used to represent Sudoku board.

Hast table

Used to represent the domain of each cell.

Domain

Used to represent a single cell domain.

Queue

Used to represent the arcs.

3 Algorithms

Backtracking

- Time complexity: $O(d^n)$
- Explanation: The time complexity of backtracking in the context of Sudoku solving is typically represented as $O(d^n)$, where d is the depth of the search tree (the number of possibilities for each decision), and n is the number of empty cells or variables that need to be filled. This exponential complexity can be mitigated by using heuristics like Minimum Remaining Values (MRV), Least Constraining Value (LCV), and constraint propagation techniques such as forward checking or maintaining arc consistency.

Arc Consistency

- Time complexity (AC-3): $O(ed^3)$
- Explanation: Applying arc consistency to a Constraint Satisfaction Problem (CSP) involves checking and reducing the domains of variables based on constraints. The time complexity of AC-3 algorithms is $O(ed^3)$, where e is the number of edges in the constraint graph and d is the maximum domain size. The cubic factor arises from the need to check and update domains iteratively.

LCV (Least Constraining Value)

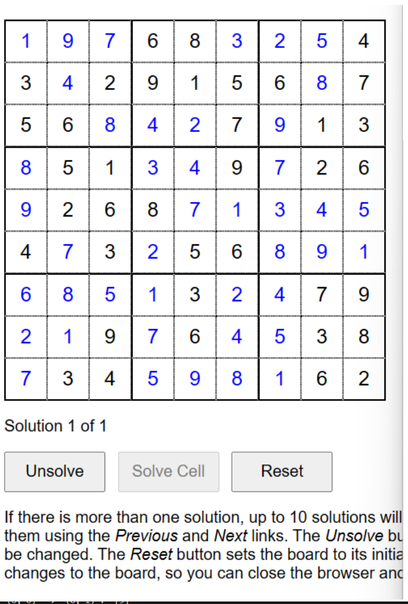
LCV is a variable ordering heuristic in CSPs that prioritizes assigning values to variables with the least impact on neighboring variables' domains. It helps reduce the branching factor by making informed decisions during variable assignment.

MRV (Minimum Remaining Values)

MRV is a variable ordering heuristic that prioritizes variables with the fewest remaining values in their domains. It aims to handle more constrained variables early in the search, reducing the search space and potentially pruning the search tree faster.

4 Results

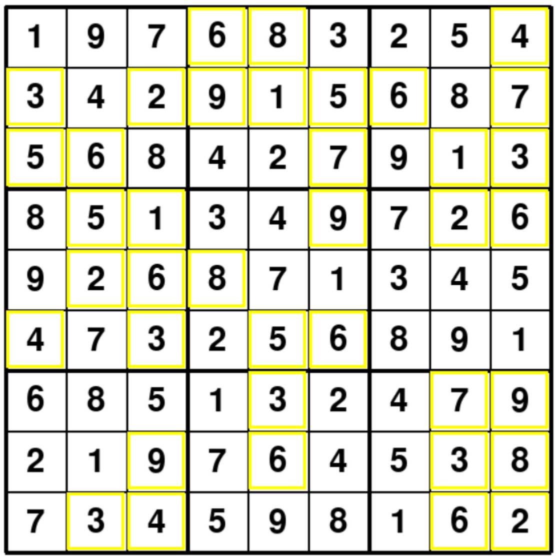
Easy



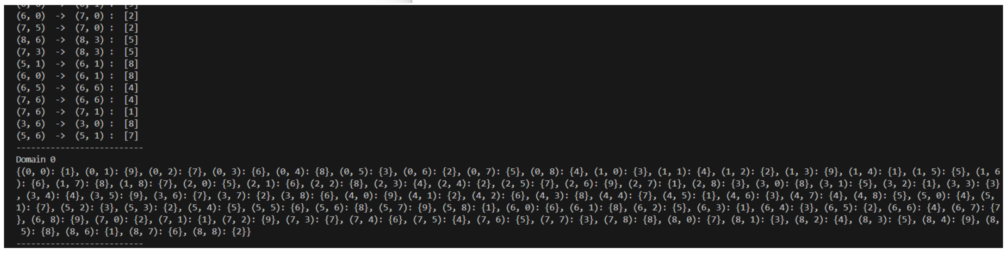
1	9	7	6	8	3	2	5	4
3	4	2	9	1	5	6	8	7
5	6	8	4	2	7	9	1	3
8	5	1	3	4	9	7	2	6
9	2	6	8	7	1	3	4	5
4	7	3	2	5	6	8	9	1
6	8	5	1	3	2	4	7	9
2	1	9	7	6	4	5	3	8
7	3	4	5	9	8	1	6	2

Solution 1 of 1

If there is more than one solution, up to 10 solutions will
them using the *Previous* and *Next* links. The *Unsolve* bu
be changed. The *Reset* button sets the board to its initia
changes to the board, so you can close the browser and



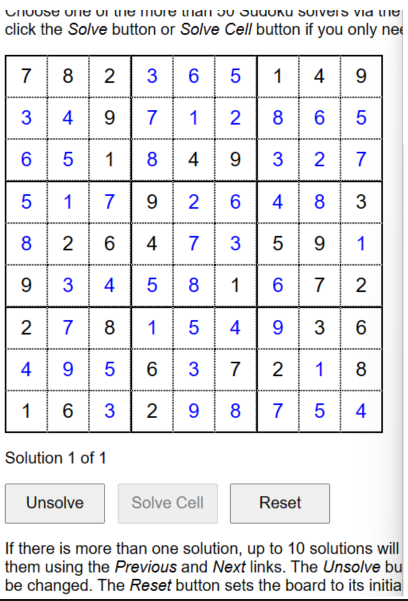
1	9	7	6	8	3	2	5	4
3	4	2	9	1	5	6	8	7
5	6	8	4	2	7	9	1	3
8	5	1	3	4	9	7	2	6
9	2	6	8	7	1	3	4	5
4	7	3	2	5	6	8	9	1
6	8	5	1	3	2	4	7	9
2	1	9	7	6	4	5	3	8
7	3	4	5	9	8	1	6	2



```
(6, 0)  ->  (7, 0) :  [2]
(7, 5)  ->  (7, 0) :  [2]
(8, 6)  ->  (8, 3) :  [5]
(7, 3)  ->  (8, 3) :  [5]
(5, 1)  ->  (6, 1) :  [8]
(6, 0)  ->  (6, 1) :  [8]
(6, 5)  ->  (6, 6) :  [4]
(7, 6)  ->  (6, 6) :  [4]
(7, 6)  ->  (7, 1) :  [1]
(3, 6)  ->  (3, 0) :  [8]
(5, 6)  ->  (5, 1) :  [7]
--------------------------
Domain 0
{(0, 0): {1}, (0, 1): {9}, (0, 2): {7}, (0, 3): {6}, (0, 4): {8}, (0, 5): {3}, (0, 6): {2}, (0, 7): {5}, (0, 8): {4}, (1, 0): {3}, (1, 1): {4}, (1, 2): {2}, (1, 3): {9}, (1, 4): {1}, (1, 5): {5}, (1, 6
): {6}, (1, 7): {8}, (1, 8): {7}, (2, 0): {5}, (2, 1): {6}, (2, 2): {8}, (2, 3): {4}, (2, 4): {2}, (2, 5): {7}, (2, 6): {9}, (2, 7): {1}, (2, 8): {3}, (3, 0): {8}, (3, 1): {5}, (3, 2): {1}, (3, 3): {3}
, (3, 4): {4}, (3, 5): {9}, (3, 6): {7}, (3, 7): {2}, (3, 8): {6}, (4, 0): {9}, (4, 1): {2}, (4, 2): {6}, (4, 3): {8}, (4, 4): {7}, (4, 5): {1}, (4, 6): {3}, (4, 7): {4}, (4, 8): {5}, (5, 0): {4}, (5,
1): {7}, (5, 2): {3}, (5, 3): {2}, (5, 4): {5}, (5, 5): {6}, (5, 6): {8}, (5, 7): {9}, (5, 8): {1}, (6, 0): {6}, (6, 1): {8}, (6, 2): {5}, (6, 3): {1}, (6, 4): {3}, (6, 5): {2}, (6, 6): {4}, (6, 7): {7
}, (6, 8): {9}, (7, 0): {2}, (7, 1): {1}, (7, 2): {9}, (7, 3): {7}, (7, 4): {6}, (7, 5): {4}, (7, 6): {5}, (7, 7): {3}, (7, 8): {8}, (8, 0): {7}, (8, 1): {3}, (8, 2): {4}, (8, 3): {5}, (8, 4): {9}, (8,
 5): {8}, (8, 6): {1}, (8, 7): {6}, (8, 8): {2}}
--------------------------
```

- Time: 0.0033524036407470703

Medium

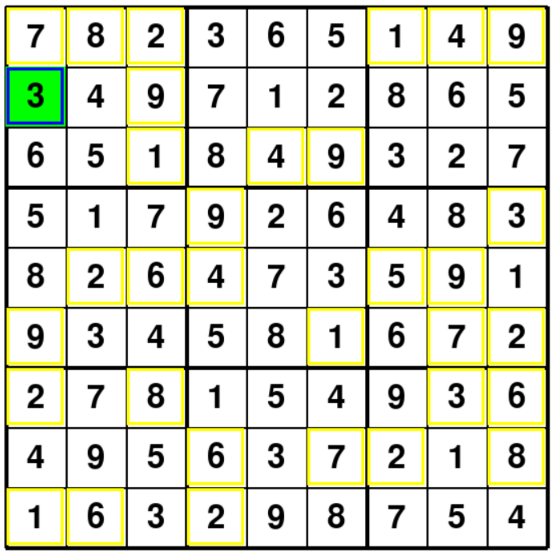


Choose one of the more than 50 Sudoku solvers via the
click the *Solve* button or *Solve Cell* button if you only nee

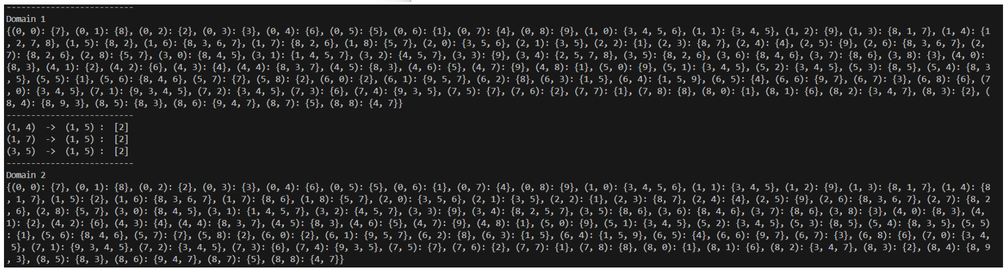
7	8	2	3	6	5	1	4	9
3	4	9	7	1	2	8	6	5
6	5	1	8	4	9	3	2	7
5	1	7	9	2	6	4	8	3
8	2	6	4	7	3	5	9	1
9	3	4	5	8	1	6	7	2
2	7	8	1	5	4	9	3	6
4	9	5	6	3	7	2	1	8
1	6	3	2	9	8	7	5	4

Solution 1 of 1

If there is more than one solution, up to 10 solutions will
them using the *Previous* and *Next* links. The *Unsolve* bu
be changed. The *Reset* button sets the board to its initia



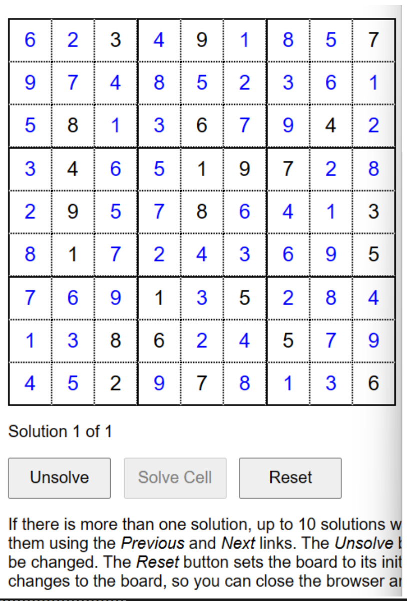
7	8	2	3	6	5	1	4	9
3	4	9	7	1	2	8	6	5
6	5	1	8	4	9	3	2	7
5	1	7	9	2	6	4	8	3
8	2	6	4	7	3	5	9	1
9	3	4	5	8	1	6	7	2
2	7	8	1	5	4	9	3	6
4	9	5	6	3	7	2	1	8
1	6	3	2	9	8	7	5	4



```
---------------------------
Domain 1
{(0, 0): {7}, (0, 1): {8}, (0, 2): {2}, (0, 3): {3}, (0, 4): {6}, (0, 5): {5}, (0, 6): {1}, (0, 7): {4}, (0, 8): {9}, (1, 0): {3, 4, 5, 6}, (1, 1): {3, 4, 5}, (1, 2): {9}, (1, 3): {8, 1, 7}, (1, 4): {1
, 2, 7, 8}, (1, 5): {8, 2}, (1, 6): {8, 3, 6, 7}, (1, 7): {8, 2, 6}, (1, 8): {5, 7}, (2, 0): {3, 5, 6}, (2, 1): {3, 5}, (2, 2): {1}, (2, 3): {8, 7}, (2, 4): {4}, (2, 5): {9}, (2, 6): {8, 3, 6, 7}, (2,
7): {8, 2, 6}, (2, 8): {5, 7}, (3, 0): {8, 4, 5}, (3, 1): {1, 4, 5, 7}, (3, 2): {4, 5, 7}, (3, 3): {9}, (3, 4): {2, 5, 7, 8}, (3, 5): {8, 2, 6}, (3, 6): {8, 4, 6}, (3, 7): {8, 6}, (3, 8): {3}, (4, 0):
{8, 3}, (4, 1): {2}, (4, 2): {6}, (4, 3): {4}, (4, 4): {8, 3, 7}, (4, 5): {8, 3}, (4, 6): {5}, (4, 7): {9}, (4, 8): {1}, (5, 0): {9}, (5, 1): {3, 4, 5}, (5, 2): {3, 4, 5}, (5, 3): {8, 5}, (5, 4): {8, 3
, 5}, (5, 5): {1}, (5, 6): {8, 4, 6}, (5, 7): {7}, (5, 8): {2}, (6, 0): {2}, (6, 1): {9, 5, 7}, (6, 2): {8}, (6, 3): {1, 5}, (6, 4): {1, 5, 9}, (6, 5): {4}, (6, 6): {9, 7}, (6, 7): {3}, (6, 8): {6}, (7
, 0): {3, 4, 5}, (7, 1): {9, 3, 4, 5}, (7, 2): {3, 4, 5}, (7, 3): {6}, (7, 4): {9, 3, 5}, (7, 5): {7}, (7, 6): {2}, (7, 7): {1}, (7, 8): {8}, (8, 0): {1}, (8, 1): {6}, (8, 2): {3, 4, 7}, (8, 3): {2}, (
8, 4): {8, 9, 3}, (8, 5): {8, 3}, (8, 6): {9, 4, 7}, (8, 7): {5}, (8, 8): {4, 7}}
---------------------------
(1, 4)  ->  (1, 5) :  [2]
(1, 7)  ->  (1, 5) :  [2]
(3, 5)  ->  (1, 5) :  [2]
---------------------------
Domain 2
{(0, 0): {7}, (0, 1): {8}, (0, 2): {2}, (0, 3): {3}, (0, 4): {6}, (0, 5): {5}, (0, 6): {1}, (0, 7): {4}, (0, 8): {9}, (1, 0): {3, 4, 5, 6}, (1, 1): {3, 4, 5}, (1, 2): {9}, (1, 3): {8, 1, 7}, (1, 4): {8
, 1, 7}, (1, 5): {2}, (1, 6): {8, 3, 6, 7}, (1, 7): {8, 6}, (1, 8): {5, 7}, (2, 0): {3, 5, 6}, (2, 1): {3, 5}, (2, 2): {1}, (2, 3): {8, 7}, (2, 4): {4}, (2, 5): {9}, (2, 6): {8, 3, 6, 7}, (2, 7): {8, 2
, 6}, (2, 8): {5, 7}, (3, 0): {8, 4, 5}, (3, 1): {1, 4, 5, 7}, (3, 2): {4, 5, 7}, (3, 3): {9}, (3, 4): {8, 2, 5, 7}, (3, 5): {8, 6}, (3, 6): {8, 4, 6}, (3, 7): {8, 6}, (3, 8): {3}, (4, 0): {8, 3}, (4,
1): {2}, (4, 2): {6}, (4, 3): {4}, (4, 4): {8, 3, 7}, (4, 5): {8, 3}, (4, 6): {5}, (4, 7): {9}, (4, 8): {1}, (5, 0): {9}, (5, 1): {3, 4, 5}, (5, 2): {3, 4, 5}, (5, 3): {8, 5}, (5, 4): {8, 3, 5}, (5, 5)
: {1}, (5, 6): {8, 4, 6}, (5, 7): {7}, (5, 8): {2}, (6, 0): {2}, (6, 1): {9, 5, 7}, (6, 2): {8}, (6, 3): {1, 5}, (6, 4): {1, 5, 9}, (6, 5): {4}, (6, 6): {9, 7}, (6, 7): {3}, (6, 8): {6}, (7, 0): {3, 4,
 5}, (7, 1): {9, 3, 4, 5}, (7, 2): {3, 4, 5}, (7, 3): {6}, (7, 4): {9, 3, 5}, (7, 5): {7}, (7, 6): {2}, (7, 7): {1}, (7, 8): {8}, (8, 0): {1}, (8, 1): {6}, (8, 2): {3, 4, 7}, (8, 3): {2}, (8, 4): {8, 9
, 3}, (8, 5): {8, 3}, (8, 6): {9, 4, 7}, (8, 7): {5}, (8, 8): {4, 7}}
```

- Time: 0.006730318069458008

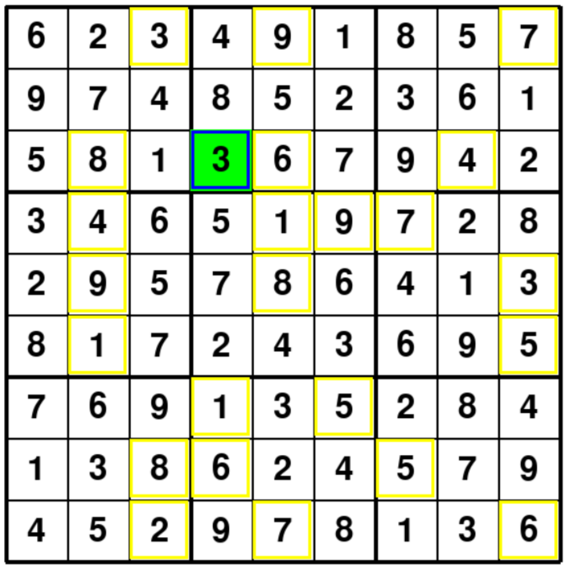
Hard



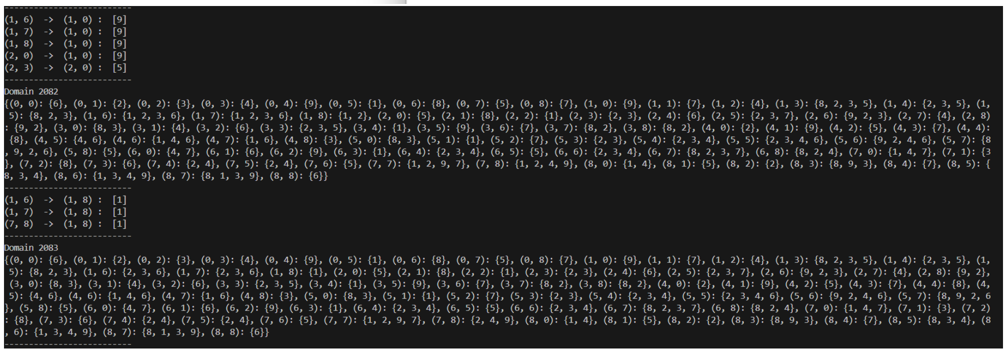
6	2	3	4	9	1	8	5	7
9	7	4	8	5	2	3	6	1
5	8	1	3	6	7	9	4	2
3	4	6	5	1	9	7	2	8
2	9	5	7	8	6	4	1	3
8	1	7	2	4	3	6	9	5
7	6	9	1	3	5	2	8	4
1	3	8	6	2	4	5	7	9
4	5	2	9	7	8	1	3	6

Solution 1 of 1

If there is more than one solution, up to 10 solutions w
them using the *Previous* and *Next* links. The *Unsolve* b
be changed. The *Reset* button sets the board to its init
changes to the board, so you can close the browser ar



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



```
-------------------------
(1, 6)  ->  (1, 0) :  [9]
(1, 7)  ->  (1, 0) :  [9]
(1, 8)  ->  (1, 0) :  [9]
(2, 0)  ->  (1, 0) :  [9]
(2, 3)  ->  (2, 0) :  [5]
-------------------------
Domain 2082
{(0, 0): {6}, (0, 1): {2}, (0, 2): {3}, (0, 3): {4}, (0, 4): {9}, (0, 5): {1}, (0, 6): {8}, (0, 7): {5}, (0, 8): {7}, (1, 0): {9}, (1, 1): {7}, (1, 2): {4}, (1, 3): {8, 2, 3, 5}, (1, 4): {2, 3, 5}, (1,
 5): {8, 2, 3}, (1, 6): {1, 2, 3, 6}, (1, 7): {1, 2, 3, 6}, (1, 8): {1, 2}, (2, 0): {5}, (2, 1): {8}, (2, 2): {1}, (2, 3): {2, 3}, (2, 4): {6}, (2, 5): {2, 3, 7}, (2, 6): {9, 2, 3}, (2, 7): {4}, (2, 8)
: {9, 2}, (3, 0): {8, 3}, (3, 1): {4}, (3, 2): {6}, (3, 3): {2, 3, 5}, (3, 4): {1}, (3, 5): {9}, (3, 6): {7}, (3, 7): {8, 2}, (3, 8): {8, 2}, (4, 0): {2}, (4, 1): {9}, (4, 2): {5}, (4, 3): {7}, (4, 4):
 {8}, (4, 5): {4, 6}, (4, 6): {1, 4, 6}, (4, 7): {1, 6}, (4, 8): {3}, (5, 0): {8, 3}, (5, 1): {1}, (5, 2): {7}, (5, 3): {2, 3}, (5, 4): {2, 3, 4}, (5, 5): {2, 3, 4, 6}, (5, 6): {9, 2, 4, 6}, (5, 7): {8
, 9, 2, 6}, (5, 8): {5}, (6, 0): {4, 7}, (6, 1): {6}, (6, 2): {9}, (6, 3): {1}, (6, 4): {2, 3, 4}, (6, 5): {5}, (6, 6): {2, 3, 4}, (6, 7): {8, 2, 3, 7}, (6, 8): {8, 2, 4}, (7, 0): {1, 4, 7}, (7, 1): {3
}, (7, 2): {8}, (7, 3): {6}, (7, 4): {2, 4}, (7, 5): {2, 4}, (7, 6): {5}, (7, 7): {1, 2, 9, 7}, (7, 8): {1, 2, 4, 9}, (8, 0): {1, 4}, (8, 1): {5}, (8, 2): {2}, (8, 3): {8, 9, 3}, (8, 4): {7}, (8, 5): {
8, 3, 4}, (8, 6): {1, 3, 4, 9}, (8, 7): {8, 1, 3, 9}, (8, 8): {6}}
-------------------------
(1, 6)  ->  (1, 8) :  [1]
(1, 7)  ->  (1, 8) :  [1]
(7, 8)  ->  (1, 8) :  [1]
-------------------------
Domain 2083
{(0, 0): {6}, (0, 1): {2}, (0, 2): {3}, (0, 3): {4}, (0, 4): {9}, (0, 5): {1}, (0, 6): {8}, (0, 7): {5}, (0, 8): {7}, (1, 0): {9}, (1, 1): {7}, (1, 2): {4}, (1, 3): {8, 2, 3, 5}, (1, 4): {2, 3, 5}, (1,
 5): {8, 2, 3}, (1, 6): {2, 3, 6}, (1, 7): {2, 3, 6}, (1, 8): {1}, (2, 0): {5}, (2, 1): {8}, (2, 2): {1}, (2, 3): {2, 3}, (2, 4): {6}, (2, 5): {2, 3, 7}, (2, 6): {9, 2, 3}, (2, 7): {4}, (2, 8): {9, 2},
 (3, 0): {8, 3}, (3, 1): {4}, (3, 2): {6}, (3, 3): {2, 3, 5}, (3, 4): {1}, (3, 5): {9}, (3, 6): {7}, (3, 7): {8, 2}, (3, 8): {8, 2}, (4, 0): {2}, (4, 1): {9}, (4, 2): {5}, (4, 3): {7}, (4, 4): {8}, (4,
 5): {4, 6}, (4, 6): {1, 4, 6}, (4, 7): {1, 6}, (4, 8): {3}, (5, 0): {8, 3}, (5, 1): {1}, (5, 2): {7}, (5, 3): {2, 3}, (5, 4): {2, 3, 4}, (5, 5): {2, 3, 4, 6}, (5, 6): {9, 2, 4, 6}, (5, 7): {8, 9, 2, 6
}, (5, 8): {5}, (6, 0): {4, 7}, (6, 1): {6}, (6, 2): {9}, (6, 3): {1}, (6, 4): {2, 3, 4}, (6, 5): {5}, (6, 6): {2, 3, 4}, (6, 7): {8, 2, 3, 7}, (6, 8): {8, 2, 4}, (7, 0): {1, 4, 7}, (7, 1): {3}, (7, 2)
: {8}, (7, 3): {6}, (7, 4): {2, 4}, (7, 5): {2, 4}, (7, 6): {5}, (7, 7): {1, 2, 9, 7}, (7, 8): {2, 4, 9}, (8, 0): {1, 4}, (8, 1): {5}, (8, 2): {2}, (8, 3): {8, 9, 3}, (8, 4): {7}, (8, 5): {8, 3, 4}, (8
, 6): {1, 3, 4, 9}, (8, 7): {8, 1, 3, 9}, (8, 8): {6}}
-------------------------
```

- Time: 0.0128