SUDOKU SOLVER

Leonardo Iurada - Computational Intelligence

→ A company, every day we need a technician

- → A company, every day we need a technician
- → Freelance available days:

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Alice	X			X			X
Bob	X				X		
Chuck	X		X		X		X
Dan		X		X			
Eve			X			X	
Frank		X		X		X	

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Alice	X			X			X
Bob	X				X		
Chuck	X		X		X		X
Dan		X		X			
Eve			X			X	
Frank		X		X		X	

- → Constraints:
 - Only 1 person per
 day

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Alice	X			X			X
Bob	X				X		
Chuck	X		X		X		X
Dan		X		X			
Eve			X			X	
Frank		X		X		X	

- Only 1 person per day
- Rows picked as they are (no slicing)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Alice	X			X			X
Bob	X				X		
Chuck	X		X		X		X
Dan		X		X			
Eve			X			X	
Frank		X		X		X	

- Only 1 person per day
- Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Alice	X			X			X
Bob	X				X		
Chuck	X		X		X		X
Dan		X		X			
Eve			X			X	
Frank		X		X		X	

$$\mathcal{O}\left(\sum_{k=1}^{6} {6 \choose k}\right) \to 63$$

- ♦ Only 1 person per day
- Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Alice	X			X			X
Bob	X				X		
Chuck	X		X		X		X
Dan		X		X			
Eve			X			X	
Frank		X		X		X	

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	0	1	0	1	0	0	0
Ε	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	0	1	0	1	0	0	0
Ε	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	0	1	0	1	0	0	0
Ε	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	a		0	1	0	a	0
E	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	3	0	0	1	0	0
С	1	Э	1	0	1	0	1
D	9	4	0	1	0	a	0
E	0	9	1	0	0	1	0
F	0	1	0	1	0	1	0

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	0	0	1	0	0	1
1	9	0	0	1	0	0
1	Э	1	0	1	0	1
9	4	0	1	0	9	0
0	9	1	0	0	1	0
0	1	0	1	0	1	0
	Mon 1 1 1 0 0	Mon Tue 1	1 0 0 1 0 1	1 0 0 1 1 0 0 0 1 0 1 0 0 0 1 0	1 0 0 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0	1 0 0 1 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Τι	ıe	Wed	Т	hu	Fri	Sat	Sun	
À			J	Ø		1	Ø	Ø	1	
В	1)	0		0	1	0	0	
С	1		9	1		0	1	0	1	
D	a		7	0		1	0	Q	0	
E	0)	1		0	0	1	0	
F	Ø			g		1	9	1	0	
_										

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Wed	Fri	Sat	Sun
В	1	0	1	0	0
С	1	1	1	0	1
Ε	0	1	0	1	0

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

				1	
	Mon	Wed	Fri	Sat	Sun
В	1	0	1	0	0
С	1	1	1	0	1
Ε	0	1	0	1	0

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

				1	
	Mon	Wed	Fri	Sat	Sun
В	1	0	1	0	0
С	1	1	1	0	1
Г	0	1	О		a
L	0	'	0		

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	We	d	Fri	Sat	Sun		
В	1	0		1	0	0		
^						1		
C		l l		I	Ψ			
Г	0	-				0		
L	0			9		8		

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Fri	Sun
В	1	1	0

Solution: [D,E,B]

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Fri	Sun
В	1	1	0

Solution: [D,E,B] is not a solution!

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

Backtrack!

	Mon	Fri	Sun
В	1	1	0

Solution: [D,E,\begin{aligned} is not a solution!

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

				1	
	Mon	Wed	Fri	Sat	Sun
В	1	0	1	0	0
С	1	1	1	0	1
Ε	0	1	0	1	0

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

D.	ckti	rack!			<u> </u>			
Ba	CKC	Mon	Wed	Fri	Sat	Sun		
	В	1	0	1	0	0		
	С	1	1	1	0	1		
	Ε	0	1	0	1	0		

Solution: [D, [☑]

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	0	1	0	1	0	0	0
Ε	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

Solution: [D]

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	0	1	0	1	0	0	0
Ε	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

Solution: [∅]

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Α	1	0	0	1	0	0	1
В	1	0	0	0	1	0	0
С	1	0	1	0	1	0	1
D	0	1	0	1	0	0	0
Ε	0	0	1	0	0	1	0
F	0	1	0	1	0	1	0

Solution: [F]

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
٨	1				0		1
Т			0		0		
В	1	Ð	0	0	1	ð	0
С	1	Э	1	0	1	ð	1
D	0	1					0
ע	0		Ø		0	2	И
	0					1	0
L	U	O		p	0	1	
						4	0
Г	0	U	0		О	1	- U

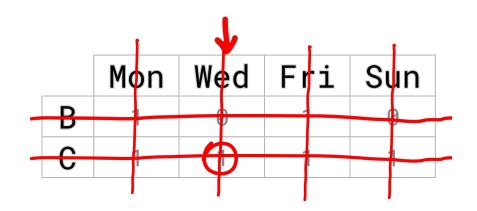
Solution: [F]

- → Constraints:
 - Only 1 person per
 day
 - Rows picked as they are (no slicing)
- → Solution:
 - ♠ A set of rows that cover all days

		1		
	Mon	Wed	Fri	Sun
В	1	0	1	0
C	1	1	1	1

Solution: [F]

- → Constraints:
 - Only 1 person per day
 - Rows picked as they are (no slicing)
- → Solution:
 - A set of rows that cover all days



Solution: [F,C] is a solution!

Sudoku, finally...

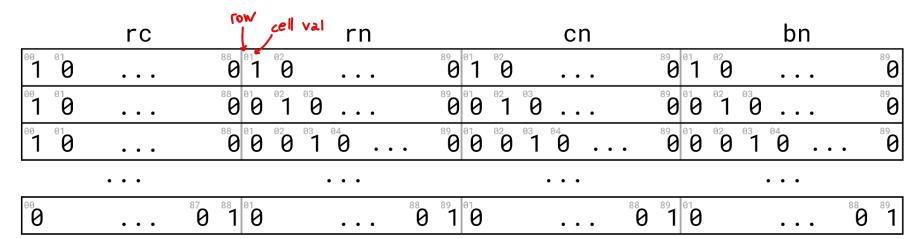
- → Sudoku as a graph, adjacency matrix:
 - ◆ Easy representation of constraints

	rc	rn	cn	bn
1 0		⁸⁸ 0 1 02	⁸⁹ 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	⁸⁹ 0 1 0 0
1 0		88 0 0 1 0 02 03 0	89 0 0 1 0	890 0 1 0 890
1 0		88 0 0 0 0 0 1 0	. 89 0 0 0 1 0	89 0 0 0 1 0 89
		• • •	• • •	
00		87 0 88 1 91	80 89 1 0	0 1 0 0 1

- → Sudoku as a graph, adjacency matrix:
 - ♦ Easy representation of constraints

row	حما /	rc		rn	cn	bn
00	010		88 0 1 02)	90 1 02	³⁹ 0 1 02
99	01		88 0 0 02	⁸³ 0 ⁸⁹	90 0 1 0	³⁹ 0 0 1 0 0
00	010		88 0 01 02 02	1 0 8	90 0 0 0 1 0	39 0 0 0 1 0 0 0 89 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
		• • •		• • •	• • •	• • •
9			870 88 1 010	8	⁹ 1 0 8 8	³³ 1 0 8 ³¹ 1

- → Sudoku as a graph, adjacency matrix:
 - ◆ Easy representation of constraints



- → Sudoku as a graph, adjacency matrix:
 - ♦ Easy representation of constraints

	rc		rn	cal cell val cn	bn
1 0		88 0 1 02		890 1 0	89 1 9 89
1 0		88 0 0 0 1	³ 0	89 0 1 0 1 0	89 0 1 0 89
1 0		88 0 0 0 0 0	1 0	890 0 0 0 1 0	89 0 0 0 1 0 89
					• • •
8		870 88 1 010	88	⁸⁹ 1 0	0 1 0 8 1

- → Sudoku as a graph, adjacency matrix:
 - ◆ Easy representation of constraints

	rc	rn	cn	bn العرب
1 0		⁸⁸ 0 1 02	⁸⁹ 0 1 02	890 1 0 89
1 0		88 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	890 0 1 0	89 0 0 1 0 89
1 0		88 0 0 0 0 1 0	890 0 0 0 1 0	. 89 0 0 0 1 0 89
	• • •		• • •	• • •
00		⁸⁷ 0 ⁸⁸ 1 ⁹¹ 0 ⁸⁸	91 0	88 89 1 0 88 89

Sudoku representation

- → Sudoku as a graph, adjacency matrix:
 - ♦ Easy representation of constraints

rov	/ /	دما	rc	r	w L	cell	val	r	n		•	ام: ا		cell	val	С	n			bex	cell	val	b	n			
0	1	010		880	⁰¹	02					89	9	1	⁰² 0					89	91	02		•				89
0	1	010		88	010	02	93				89	9 8	0	02	8 3				89	010	02	8 3	•				89
0	1	010		88	010	02	03	04	•		89	9 0.	0	02	03	04	•	•	89	010	02	03	⁰⁴ 0	•	• •	•	89
			• • •						,							• • •	•						• •	•			
0	0			⁸⁷ 0 ⁸⁸ 1	010					8	0 9		0					88	89	010			•			88	89

Sudoku representation

→ Better to use an Adjacency List

```
possible values = {
    ('rc', (0,0)) : \{ (0,0,1), (0,0,2), (0,0,3), \dots, (0,0,8), (0,0,9) \},
    ('rc', (0,1)) : \{ (0,1,1), (0,1,2), (0,1,3), \dots, (0,1,8), (0,1,9) \},
    ('rc', (8,8)) : \{ (8,8,1), (8,8,2), (8,8,3), \dots, (8,8,8), (8,8,9) \},
    (rn', (0,1)) : \{ (0,0,1), (0,1,1), (0,2,1), \dots, (0,7,1), (0,8,1) \},
    ('rn', (0,9)) : \{ (0,0,9), (0,1,9), (0,2,9), \dots, (0,7,9), (0,8,9) \},
    ('rn', (1,1)) : \{ (1,0,1), (1,1,1), (1,2,1), \dots, (1,7,1), (1,8,1) \},
    ('rn', (1,9)) : \{ (1,0,9), (1,1,9), (1,2,9), \dots, (1,7,9), (1,8,9) \},
    (\text{'rn'}, (8,9)) : \{ (8,0,9), (8,1,9), (8,2,9), \dots, (8,7,9), (8,8,9) \},
    (cn', (0,1)) : \{ (0,0,1), (1,0,1), (2,0,1), \dots, (7,0,1), (8,0,1) \},
    (cn', (1,1)) : \{ (0,1,1), (1,1,1), (2,1,1), \dots, (7,1,1), (8,1,1) \},
    ('cn', (8,9)) : \{ (0,8,9), (1,8,9), (2,8,9), \dots, (7,8,9), (8,8,9) \},
    (bn', (0,1)) : \{ (0,0,1), (0,1,1), (0,2,1), \dots, (2,1,1), (2,2,1) \},
    (bn', (0,9)) : \{ (0,0,9), (0,1,9), (0,2,9), \dots, (2,1,9), (2,2,9) \},
    ('bn', (1,1)) : \{ (0,3,1), (0,4,1), (0,5,1), \dots, (2,4,1), (2,5,1) \},
    ('bn', (1,9)) : \{ (0,3,9), (0,4,9), (0,5,9), \dots, (2,4,9), (2,5,9) \},
    (bn', (8,9)) : \{ (6,6,9), (6,7,9), (6,8,9), \dots, (8,7,9), (8,8,9) \},
```

Sudoku representation

→ Better to use an Adjacency List

```
# Links used to navigate the graph when we insert/delete values in cells links = {
    (0,0,1) : [ ('rc', (0,0)), ('rn', (0,1)), ('cn', (0,1)), ('bn', (0,1)) ], (0,0,2) : [ ('rc', (0,0)), ('rn', (0,2)), ('cn', (0,2)), ('bn', (0,2)) ], ...
    (8,8,8) : [ ('rc', (8,8)), ('rn', (8,8)), ('cn', (8,8)), ('bn', (8,8)) ], (8,8,9) : [ ('rc', (8,8)), ('rn', (8,9)), ('cn', (8,9)), ('bn', (8,9)) ]
```

```
possible values = {
    (\text{'rc'}, (0,0)) : \{ (0,0,1), (0,0,2), (0,0,3), \dots, (0,0,8), (0,0,9) \},
    (\text{'rc'}, (0,1)) : \{ (0,1,1), (0,1,2), (0,1,3), \dots, (0,1,8), (0,1,9) \},
    ('rc', (8,8)) : \{ (8,8,1), (8,8,2), (8,8,3), \dots, (8,8,8), (8,8,9) \},
    (rn', (0,1)) : \{ (0,0,1), (0,1,1), (0,2,1), \dots, (0,7,1), (0,8,1) \},
    ('rn', (0,9)) : { (0,0,9), (0,1,9), (0,2,9), ..., (0,7,9), (0,8,9) },
    (rn', (1,1)) : \{ (1,0,1), (1,1,1), (1,2,1), \dots, (1,7,1), (1,8,1) \},
    ('rn', (1,9)) : { (1,0,9), (1,1,9), (1,2,9), ..., (1,7,9), (1,8,9) },
    (\text{'rn'}, (8,9)) : \{ (8,0,9), (8,1,9), (8,2,9), \dots, (8,7,9), (8,8,9) \},
    (cn', (0,1)) : \{ (0,0,1), (1,0,1), (2,0,1), \dots, (7,0,1), (8,0,1) \},
    (cn', (1,1)) : \{ (0,1,1), (1,1,1), (2,1,1), \dots, (7,1,1), (8,1,1) \},
    ('cn', (8,9)) : \{ (0,8,9), (1,8,9), (2,8,9), \dots, (7,8,9), (8,8,9) \},
    (bn', (0,1)) : \{ (0,0,1), (0,1,1), (0,2,1), \dots, (2,1,1), (2,2,1) \},
    (bn', (0,9)) : \{ (0,0,9), (0,1,9), (0,2,9), \dots, (2,1,9), (2,2,9) \},
    ('bn', (1,1)) : \{ (0,3,1), (0,4,1), (0,5,1), \dots, (2,4,1), (2,5,1) \},
    ('bn', (1,9)) : \{ (0,3,9), (0,4,9), (0,5,9), \dots, (2,4,9), (2,5,9) \},
    (bn', (8,9)) : \{ (6,6,9), (6,7,9), (6,8,9), \dots, (8,7,9), (8,8,9) \},
```

```
# Initialization (transform the 2D array into a graph via adjacency lists)
links = {}
for row, col, n in product(range(9), range(9), range(1, 9+1)):
    box = (row//3)*3 + (col//3)
    links[(row, col, n)] = [('rc', (row, col)),
                               ('rn', (row, n)),
                               ('cn', (col, n)),
                               ('bn', (box, n))]
possible_values = ([('rc', x) \text{ for } x \text{ in product}(range(9), range(9))] +
                     [('rn', x) \text{ for } x \text{ in product(range(9), range(1, 9+1))}] +
                     [('cn', x) \text{ for } x \text{ in product(range(9), range(1, 9+1))}] +
                     [('bn', x) \text{ for } x \text{ in product(range(9), range(1, 9+1))}])
possible_values = { k : set() for k in possible_values }
for i, row in links.items(): # i is the key, row is the corresp value of the dict
    for j in row:
         possible values[j].add(i)
for r, row in enumerate(sudoku):
    for c, n in enumerate(row):
        if n:
             trim(possible_values, links, (r,c,n))
```

```
# Initialization (transform the 2D array into a graph via adjacency lists)
        links = {}
        for row, col, n in product(range(9), range(9), range(1, 9+1)):
# Start searching for a solution
num_nodes = [1]
solution = search(possible_values, links, [], num_nodes)
        for r, row in enumerate(sudoku):
           for c, n in enumerate(row):
               if n:
                  trim(possible_values, links, (r,c,n))
```

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

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

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

8
1 2 1 2 3 4 5 6 4 6 6 9 8 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7
1 2 1 2 3 4 5 6 4 6 6 9 8 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7
4 5 6 7 4 5 6 4 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
1 4 5 6 7 4 5 6 4 3 9 1 3 2 4 5 6 4 5 6 8 9 1 3 2 4 5 6 6 6 8 9 8 9 1 3 2 4 5 8 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 4 6 4 4 1 2 1 2 4 4 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 4 5 9 8 9
1 4 5 6 7 4 5 6 4 3 9 1 3 2 4 5 6 4 5 6 8 9 1 3 2 4 5 6 6 6 8 9 8 9 1 3 2 4 5 8 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 4 6 4 4 1 2 1 2 4 4 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 4 5 9 8 9
4 5 6 7 4 5 6 8 9 4 8 2 4 5 8 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 4 5 7 8 9
1 2 3 6 9 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 2 3 1 2 3 2 7 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
1 2 3 6 9 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 2 3 1 2 3 2 7 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
4 6 5 4 6 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 1 2 3 2 1 2 3 2 1 2 3 2 1 2 3 2 1 2 3 3 4 5 7 8 9
9
9
6 8 9 8 9 4 5 7 8 9 2 4 5 7 8 9 2 4 6 4 6 4 6 8 7 9 7 9 7 9 7 9 9 9 9 9 9 9 9 9 9 9 9
9 8 9 8 9 2 2 2 2 4 6 4 6 4 6 8 5 6 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
9 8 9 8 9 2 2 2 2 4 6 4 6 4 6 8 5 6 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
4 6 4 6 4 6 5 6 8 8 9 8 9 8 9 8 9 8 9 8 9 8 9
7 9 8 7 9 8 8 9 8 8 9 8 9 8 9 8 9
7 9 8 7 9 8 8 9 8 9 8 9 2 3 2 3 2 3 2 3 2 3 3 3 6 8 7 9 7 9 7 9 9 9 9 9 2 3 2 3 2 3 2 3 3 2 3 4 6 4 6 4 6 4 6 4 6 1 1
4 5 4 1 4 7 9 7 4 9 9 6 8 2 3 2 3 2 3 2 3 2 3 3 1 2 4 6 4 6 8 5 6 4 6 3 1 1
7 9 7 9 9 9 9 1 2 3 2 3 4 6 4 6 8 5 6 4 6 1
7 9 7 9 9 9 9 1 2 3 2 3 4 6 4 6 8 5 6 4 6 1
4 6 4 6 8 5 6 4 6 1
4 6 4 6 8 5 6 4 6 1
7 7 9 9 9 7
23 2 23 1 2 3 1 2 3 2 2 3
5 6 9 5 6 6 4 5 5
7 7 7 8 7 8 8 7 7 7

```
def search(possible values, links, solution, num nodes):
    if not possible values:
       return solution
   else:
       col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
           solution.append(r)
           trimmed_columns = trim(possible_values, links, r)
           s = search(possible_values, links, solution, num_nodes)
           if s:
               return s
           if expand(possible_values, links, r, trimmed_columns):
               num_nodes[0] += 1
           solution.pop()
```

			1	2			2			2	3	1	2	3	1	2	3	1		3				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
								9	7			7								9	7		9	7		9
1	2		1	2					Г			1	2		1	2		1						1		
4	5		4				3			6			5		4				5		4	5		4	5	
		9										7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
										8			_			8						8				
1	2	3					2		Г	2	3		2	3				1				2		1	2	
4		6		5		4		6						6		7				6	4			4		6
		9		_				9		8	9		8			-			8	9		8	9			9
1	2	3	1	2	3		2			2	3											2		1	2	
		6			6			6	ı				4			5			7							6
		9		8				9		8	9					_						8	9			9
	2			2			2						2			2									2	
4		6	4		6	4		6		1				6			6		5	6		3		4	5	6
7		9		8		7		9					8			8	9		8	9						9
	2	3		2	3					2	3		2	3		2	3			3						
4	5		4				1		4						4				5			6			8	
7									7		9	7					9			9						
	2	3		2	3								2	3		2	3			3					2	3
4		6	4		6		8			5				6	4		6	(1				
7										_		7					9	•		9		_		7		9
	2	3					2			2	3	1	2	3	1	2	3					2			2	3
	5	6		9			5	6						6			6		4			5			5	
7				_		7			7	8		7	8			8		•	Г		7			7		

- → Either 3 or 9
- → Both could lead to a (different) solution

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

_			_			_						_			_		_				_					_
			1	2			2		ı	2	3	1	2	3	1	2	3	1		3				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
	_							9	7			7								9	7		9	7		9
1	2		1	2								1	2		1	2		1						1		
4	5		4				3		ı	6			5		4				5		4	5		4	5	
		9					_		ı	•		7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
				•					ı	8						8			_			8				
1	2	3				_	2		Н	2	3	_	2	3				1				2		1	2	=
4	_	6		5		4	_	6	ı	_	_		_	6		7		-		6	4	_		4	_	6
1		9		_				9	ı	8	9		8			•			8	9	`	8	9			9
1	2	3	1	2	3		2	_	Н	2	3		Ť		\vdash		-		_	_	\vdash	2	_	1	2	_
1	-	6	1	-	6		-	6	ı	-	_		4			5			7			-		-	-	6
		9		8	·			9	ı	8	9		•			_			•			8	9			9
	2	_	\vdash	2			2	_	Н	Ť	_		2		Н	2					Н	_	_	_	2	_
4	-	6	4	-	6	4	-	6	ı	1			-	6		-	6		5	6		3		4	5	6
7		9	~	8	۰	7		9	ı	-			8	۰		8	9		8	9		_		7	,	9
ŕ	2	3	-	2	2	<u></u>	_	,	Н	2	2	_	2	2	Н	2	3	-	Ů	-	Н			_	_	_
١.	2 5	3	١.	2	3		4		١.	2	3		2	3	١.	2	3		5	3		6			0	
4	5		4				1		4		_	_			4		_		5	_		6			8	
7	_	_	-	_	_				7		9	7	_	_	-	_	9			9	_				_	_
١.	2	3	١.	2	3		_		ı	_			2	3	١.	2	3	/	7	3		_			2	3
4		6	4		6		8		ı	5		_		6	4		6		3	_		1		_		_
7			\vdash						╙			7					9	Ш		9				7		9
	2	3		_			2		ı	2	3	1	2	3	1	2	3		_			2			2	3
	5	6		9			5	6						6			6		4			5			5	
7						7			7	8		7	8			8					7			7		

```
• • •
  def trim(possible_values, links, r):
      trimmed_columns = []
      for j in links[r]:
          for i in possible_values[j]:
               for k in links[i]:
                   if k != j:
                       possible_values[k].remove(i)
          trimmed_columns.append(possible_values.pop(j))
      return trimmed_columns
```

_		_	_			_		_	_			_			_		_	_		_	_			_		_
			1	2			2			2	3	1	2	3	1	2	3	1		7				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
	_							9	7			7								9	7		9	7		9
1	2		1	2								1	2		1	2		1						1		
4	5		4				3			6			5		4				5		4	5		4	5	
		9					_			•		7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
				-						8			_			8			_			8				
1	2	3					2		Г	2	3		2	3				1				2		1	2	
4		6		5		4		6						6		7				6	4			4		6
		9		_				9		8	9		8			•			8	9		8	9			9
1	2	3	1	2	3		2		Н	2	3				Т							2		1	2	
		6			6			6					4			5			7							6
		9		8				9		8	9		•			_			•			8	9			9
	2			2			2						2			2									2	
4		6	4		6	4		6		1				6			6		5	6		3		4	5	6
7		9		8		7		9		_			8			8	9		8	9		_				9
	2	3		2	3				Г	2	3		2	3		2	3			1						
4	5		4				1		4						4				5	′		6			8	
7							_		7		9	7					9			9		_			_	
	2	8		2	y								2	1		2	1		_	7					2	3
4		6	4		6		8			5				6	4		6	1	3	Ú		1				•
7							_			_		7					9		_	9		_		7		9
	2	3					2		Г	2	3	1	2	3	1	2	3					2			2	2
	5	6		9			5	6	ı					6			6		4			5			5	
7				_		7			7	8		7	8			8			•		7			7		
															_											

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

_			_			_		_	_						_		_	_		_	_			_		_
			1	2			2			2	3	1	2	3	1	2	3	1		7				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
								9	7			7								9	7		9	7		9
1	2		1	2								1	2		1	2		1						1		
4	5		4				3			6			5		4				5		4	5		4	5	
		9					_			_		7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
				•						8			_			8			_			8				
1	2	3				_	2		Н	2	3	_	2	3				1				2		1	2	_
4	_	6		5		4	_	6	ı	_	-		_	6		7		ĺ		6	4	_		4	_	6
1		9		_				9		8	9		8	-		•			8	9	•	8	9			9
1	2	3	1	2	3		2	÷	Н	2	3		-						-	-		2	÷	1	2	Ť
1	-	6	1	-	6		-	6		-	•		4			5			7			-		-	-	6
		9		8	·			9		8	9		•			_			•			8	9			9
-	2	_	\vdash	2			2	-	Н	_	-	_	2		\vdash	2						Ť	Ť		2	-
4	-	6	4	-	6	4	-	6		1			-	6		-	6		5	6		3		4	5	6
7		9	~	8	۰	7		9		-			8	۰		8	9		8	9		_		7	,	9
ŕ	_	3	-	2	3	<u> </u>		,	⊢	2	3	_	2	3	Н	2	_	Н	Ů	_	_		_	-		_
4	2 5	3	١.	2	3		4		١.	2	3		2	3	١.	2	3		_	1		6			0	
	5		4				1		4		_	_			4				5	_		6			8	
7	_	_	-	_					7		9	7	_		-	_	9	-		9					_	_
١.	2	6	١.	2	Z		_			_			2	7	١.	2	1	/	7	7		_			2	7
4		6	4		6		8			5				6	4		6		2)		1				
7									┕			7					9	L	_	9				7		9
	2	3		_			2			2	3	1	2	3	1	2	3		_			2			2	2
	5	6		9			5	6						6			6		4			5			5	
7						7			7	8		7	8			8					7			7		

```
def expand(possible_values, links, r, trimmed_columns):
    expanded = False
    for j in reversed(links[r]):
        possible_values[j] = trimmed_columns.pop()
        for i in possible_values[j]:
            for k in links[i]:
                if k != j:
                    possible_values[k].add(i)
                    expanded = True
    return expanded
```

_			_			_		_	_			_			_		_	_								_
			1	2			2			2	3	1	2	3	1	2	3	1		3				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
								9	7			7								9	7		9	7		9
1	2		1	2								1	2		1	2		1						1		
4	5		4				3			6			5		4				5		4	5		4	5	
		9					_			_		7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
				•						8			_		1	8			_		'	8				
1	2	3	-	_	_		2	_	Н	2	3		2	3	\vdash	-	_	1	_	_		2		1	2	\neg
4	_	6		_		4	_	6		_	3		_	6		7		-		6	4	_		4	_	6
*		9		5		*		9		8	9		8	0		•			8	9	*	8	9	*		
-	_	_		_	_		_	9	-				8		-				8	9	_		9	_	_	9
1	2	3	1	2	3		2			2	3		_			_			_			2		1	2	_
		6			6			6					4			5			7							6
		9		8				9		8	9											8	9			9
	2			2			2			_			2			2						_			2	
4		6	4		6	4		6		1				6			6		5	6		3		4	5	6
7		9		8		7		9					8			8	9		8	9						9
	2	3		2	3					2	3		2	3		2	3			3						
4	5		4				1		4						4				5			6			8	
7									7		9	7					9			9						
	2	3		2	3								2	3		2	3			3					2	3
4		6	4		6		8		ı	5				6	4		6	1	2			1				
7							_		ı	_		7					9		_	9		_		7		9
	2	3					2		Г	2	3	1	2	3	1	2	3					2			2	3
	5	6		9			5	6	ı					6			6		4			5			5	-
7	-	•		9		7	-	-	7	8		7	8	•		8	-		-		7	-		7	-	
						Ľ			Ĺ	_			_		_	_										

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

_			_			_						_			_		_				_					_
			1	2			2		ı	2	3	1	2	3	1	2	3	1		3				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
	_							9	7			7								9	7		9	7		9
1	2		1	2								1	2		1	2		1						1		
4	5		4				3		ı	6			5		4				5		4	5		4	5	
		9					_		ı	•		7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
				•					ı	8						8			_			8				
1	2	3				_	2		Н	2	3	_	2	3				1				2		1	2	-
4	_	6		5		4	_	6	ı	_	_		_	6		7		_		6	4	_		4	_	6
1		9		_				9	ı	8	9		8			•			8	9	`	8	9			9
1	2	3	1	2	3		2	_	Н	2	3		Ť		\vdash		-		_	_	\vdash	2	_	1	2	_
1	-	6	1	-	6		-	6	ı	-	_		4			5			7			-		-	-	6
		9		8	·			9	ı	8	9		•			_			•			8	9			9
	2	_	\vdash	2			2	_	Н	Ť	_		2		Н	2					Н	_	_	_	2	_
4	-	6	4	-	6	4	-	6	ı	1			-	6		-	6		5	6		3		4	5	6
7		9	~	8	۰	7		9	ı	-			8	۰		8	9		8	9		_		7	,	9
ŕ	2	3	-	2	2	<u></u>	_	,	Н	2	2	_	2	2	Н	2	3	-	Ů	-	Н			_	_	_
١.	2 5	3	١.	2	3		4		١.	2	3		2	3	١.	2	3		5	3		6			0	
4	5		4				1		4		_	_			4		_		5	_		6			8	
7	_	_	-	_	_				7		9	7	_	_	-	_	9			9	_				_	_
١.	2	3	١.	2	3		_		ı	_			2	3	١.	2	3	/	7	3		_			2	3
4		6	4		6		8		ı	5		_		6	4		6		3	_		1		_		_
7			\vdash						╙			7					9	Ш		9				7		9
	2	3		_			2		ı	2	3	1	2	3	1	2	3		_			2			2	3
	5	6		9			5	6						6			6		4			5			5	
7						7			7	8		7	8			8					7			7		

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

8
1 2 1 2 3 4 5 6 4 6 6 9 8 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7
1 2 1 2 3 4 5 6 4 6 6 9 8 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7 9 7
4 5 6 7 4 5 6 4 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
1 4 5 6 7 4 5 6 4 3 9 1 3 2 4 5 6 4 5 6 8 9 1 3 2 4 5 6 6 6 8 9 8 9 1 3 2 4 5 8 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 4 6 4 4 1 2 1 2 4 4 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 4 5 9 8 9
1 4 5 6 7 4 5 6 4 3 9 1 3 2 4 5 6 4 5 6 8 9 1 3 2 4 5 6 6 6 8 9 8 9 1 3 2 4 5 8 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 4 6 4 4 1 2 1 2 4 4 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 4 5 9 8 9
4 5 6 7 4 5 6 8 9 4 8 2 4 5 8 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 4 5 7 8 9
1 2 3 6 9 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 2 3 1 2 3 2 7 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
1 2 3 6 9 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 1 2 3 1 2 3 2 6 9 8 9 8 7 8 9 8 9 8 2 3 1 2 3 2 7 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
4 6 5 4 6 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 1 2 3 2 1 2 3 2 1 2 3 2 1 2 3 2 1 2 3 3 4 5 7 8 9
9
9
6 8 9 8 9 4 5 7 8 9 2 4 5 7 8 9 2 4 6 4 6 4 6 8 7 9 7 9 7 9 7 9 9 9 9 9 9 9 9 9 9 9 9
9 8 9 8 9 2 2 2 2 4 6 4 6 4 6 8 5 6 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
9 8 9 8 9 2 2 2 2 4 6 4 6 4 6 8 5 6 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8
4 6 4 6 4 6 5 6 8 8 9 8 9 8 9 8 9 8 9 8 9 8 9
7 9 8 7 9 8 8 9 8 8 9 8 9 8 9 8 9
7 9 8 7 9 8 8 9 8 9 8 9 2 3 2 3 2 3 2 3 2 3 3 3 6 8 7 9 7 9 7 9 9 9 9 9 2 3 2 3 2 3 2 3 3 2 3 4 6 4 6 4 6 4 6 4 6 1 1
4 5 4 1 4 7 9 7 4 9 9 6 8 2 3 2 3 2 3 2 3 2 3 3 1 2 4 6 4 6 8 5 6 4 6 3 1 1
7 9 7 9 9 9 9 1 2 3 4 6 4 6 8 5 6 4 6 1
7 9 7 9 9 9 9 1 2 3 4 6 4 6 8 5 6 4 6 1
4 6 4 6 8 5 6 4 6 1
4 6 4 6 8 5 6 4 6 1
7 7 9 9 9 7
23 2 23 1 2 3 1 2 3 2 2 3
5 6 9 5 6 6 4 5 5
7 7 7 8 7 8 8 7 7 7

```
• • •
def search(possible values, links, solution, num nodes):
    if not possible values:
        return solution
    else:
        col = min(possible_values, key=lambda c: len(possible_values[c]))
        for r in list(possible_values[col]):
            solution.append(r)
            trimmed_columns = trim(possible_values, links, r)
            s = search(possible_values, links, solution, num_nodes)
            if s:
               return s
            if expand(possible_values, links, r, trimmed_columns):
                num_nodes[0] += 1
            solution.pop()
```

_			_				_					_					_	_			_				_	_
			1	2			2		ı	2	3	1	2	3	1	2	3	1		3				1		3
	8		4		6	4	5	6	4				5		4				5	6	4	5		4	5	6
	_							9	7			7								9	7		9	7		9
1	2		1	2								1	2		1	2		1						1		
4	5		4				3		ı	6			5		4				5		4	5		4	5	
		9					_		ı	•		7	8			8			8	9	7	8	9	7		9
1											3				1		3							1		3
4	5	6		7		4	5	6	4				9		4				2		4	5		4	5	6
				•					ı	8			_			8			_			8				
1	2	3				_	2		Н	2	3		2	3				1				2		1	2	\neg
4	_	6		5		4	_	6	ı	_	_		_	6		7		-		6	4	_		4	_	6
		9		_				9	ı	8	9		8			•			8	9	`	8	9			9
1	2	3	1	2	3		2	_	Н	2	3		Ť		\vdash		-		_	_	\vdash	2	_	1	2	_
1	-	6	1	-	6		-	6	ı	-	,		4			5			7			-		•	-	6
		9		8	۰			9	ı	8	9		7			3			•			8	9			9
	2		\vdash	2			2	,	Н	_	-		2		H	2	_					-	,		2	_
4	-	6	4	-	6	4	-	6	ı	1			_	6		_	6		5	6		3		4	5	6
7		9	"	8	٥	7		9	ı	-			8	U		8	9		8	9		3		*	3	9
ŕ	_	_	-		_	_	_	9	⊢	_	_	_		_	H		_	-	٥	-	H			_	_	9
	2	3	١.	2	3		4		١.	2	3		2	3	١.	2	3		_	3		_			_	
4	5		4				1		4		_	_			4		_		5	_		6			8	
7			_						7		9	7			_		9			9	_					
	2	3		2	3		_			_			2	3		2	3	1	`	3		_			2	3
4		6	4		6		8		ı	5				6	4		6	١,	7)			1				
7			_						L			7					9	•	_	9				7		9
	2	3		_			2			2	3	1	2	3	1	2	3		_			2			2	3
	5	6		9			5	6						6			6		4			5			5	
7						7			7	8		7	8			8					7			7		

Sudoku Demo

8	1		7		3			
	4	3	6		1	8	7	
X	7	Х		9	8	2		
	5	4	3	8	7			
	3			4	5	7	8	
			1	=		5	3	4
5	2	1		7		3	6	8
4	6	8	5	3	2	9	1	7
3	9		8	1	6	4		

Thanks for your attention!

Leonardo Iurada - Computational Intelligence