

# Rally 2 - Sudoku

## Implementación partiendo del Tutorial de Peter Norvig.

```
In [ ]: def cross(A, B):
    "Cross product of elements in A and elements in B."
    return [a+b for a in A for b in B]

digits = '123456789'
rows = 'ABCDEFGHI'
cols = digits
squares = cross(rows, cols)
# Define lista de unidades como columnas, filas y bloques en ese orden
unitlist = ([cross(rows, c) for c in cols] +
             [cross(r, cols) for r in rows] +
             [cross(rs, cs) for rs in ('CAB','DEF','GHI') for cs in ('123','456','789')])
units = dict((s, [u for u in unitlist if s in u]) for s in squares)
peers = dict((s, set(sum(units[s],[]))-set([s])) for s in squares)
```

```
In [ ]: def eliminate(values, s, d):
    """Eliminate d from values[s]; propagate when values or places <= 2.
    Return values, except return False if a contradiction is detected."""
    if d not in values[s]:
        return values ## Already eliminated
    values[s] = values[s].replace(d, '')
    ## (1) If a square s is reduced to one value d2, then eliminate d2 from the peers.
    if len(values[s]) == 0:
        return False ## Contradiction: removed last value
    elif len(values[s]) == 1:
        d2 = values[s]
        if not all(eliminate(values, s2, d2) for s2 in peers[s]):
            return False
    ## (2) If a unit u is reduced to only one place for a value d, then put it there.
    for u in units[s]:
        dplaces = [s for s in u if d in values[s]]
        if len(dplaces) == 0:
            return False ## Contradiction: no place for this value
        elif len(dplaces) == 1:
            # d can only be in one place in unit; assign it there
            if not assign(values, dplaces[0], d):
                return False
    return values

def assign(values, s, d):
    """Eliminate all the other values (except d) from values[s] and propagate.
    Return values, except return False if a contradiction is detected."""
    other_values = values[s].replace(d, '')
    if all(eliminate(values, s, d2) for d2 in other_values):
        return values
    else:
        return False
```

```
In [ ]: # NOTA: grid hace referencia al grid inicial, values hace referencia al grid que va cambiando
def grid_values(grid):
    "Convert grid into a dict of {square: char} with '0' or '.' for empties."
```

```
import time

easy = ".....4...692.539..2684.24..5.38..38.12..16.2..47.3856..741.163...9....
medium = "7.49..5..5.2.31....3.....99.7.1.....3.4.2.....5.8.13.....7....72.9.8..5
hard = " .4.....3....17...75...31.....74....9315876....93.....93...25...26....1..
evil = " ...4..5..5...2.....91.46.56...3..7..7...3..8..5...46.75.38.....1..9..2..
worlds_hardest = "8.....36.....7..9.2...5...7.....457.....1...3...1...68..85

def solve_and_measure(sudoku:str, difficulty: str):
    print(f"Solving {difficulty} sudoku...")
    start = time.perf_counter()
    solve(sudoku)
    end = time.perf_counter()
    print(f"Solved {difficulty} sudoku in {end - start}s")
```

```

solve_and_measure(easy, "easy")
solve_and_measure(medium, "medium")
solve_and_measure(hard, "hard")
solve_and_measure(evil, "evil")
solve_and_measure(worlds_hardest, "world's hardest")

```

```

Solving easy sudoku...
Solved easy sudoku in 0.0021083999999973457s
Solving medium sudoku...
Solved medium sudoku in 0.0018649999983608723s
Solving hard sudoku...
Solved hard sudoku in 0.0021851999990758486s
Solving evil sudoku...
Solved evil sudoku in 0.002300000000104774s
Solving world's hardest sudoku...
Solved world's hardest sudoku in 0.017375800001900643s

```

```

In [ ]: ultimo = """
. . . | . . 6 | . . .
. 5 9 | . . . | . . 8
2 . . | . . 8 | . . .
-----+-----+-----
. 4 5 | . . . | . . .
. . 3 | . . . | . . .
. . 6 | . . 3 | . 5 4
-----+-----+-----
. . . | 3 2 5 | . . 6
. . . | . . . | . . .
. . . | . . . | . . .
"""

solve_and_measure(ultimo, "last")

```

```

Solving last sudoku...
Solved last sudoku in 30.657734700002038s

```

## Mejoras potenciales para el algoritmo

Peter Norvig establece 2 reglas a seguir como heurísticas para mejorar su código:

1. If a square  $s$  is reduced to one value  $d$ , then eliminate  $d$  from the peers.
2. If a unit  $u$  is reduced to only one place for a value  $d$ , then put it there.

La regla n. 2 también se le llama la regla de "Hidden Singles".

Hay otras reglas que existen al resolver Sudokus que podemos implementar para mejorar la eficiencia del algoritmo y 2 muy efectivas son Hidden Pairs (3) y Naked Pairs (4).

Los Hidden Pairs son cuando se tiene el mismo par de 2 dígitos como candidatos en exactamente 2 celdas dentro de la misma unidad. Cuando se presenta este caso, se pueden eliminar los otros candidatos de esas celdas. Esto puede llevar a que se revelen algunos Hidden Singles.

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

Otro muy útil son los Naked Pairs. Estos son cuando se tiene el mismo par de dígitos como candidatos en exactamente 2 celdas dentro de la misma unidad **sin otros candidatos en esas celdas**. En ese caso, se pueden eliminar ese par de dígitos de todas las otras celdas de su unidad. Esto igual hace que se revelen otros patrones útiles.

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

→

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

→
  

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

Con estas nuevas reglas se pueden agregar nuevas heurísticas:

- Si se tiene el mismo par de 2 dígitos como candidatos en exactamente 2 celdas dentro de la misma unidad, elimina los candidatos que no formen parte de ese par en sus celdas.
- Si se tiene el mismo par de 2 dígitos como los únicos candidatos en exactamente 2 celdas dentro de la misma unidad, elimina ese par de dígitos como candidatos de todas las otras celdas en la unidad.