

**Fidel Morales Briones A01198630**  
**Tarea 4 backtracking 22 de septiembre**

Pseudocódigo:

```
Unset
isValid(row, column, grid, n):
    # revisar si ya existe el numero en su fila
    for i = 0 until i < 9:
        if grid[row][i] == n:
            return false

    # revisar si ya existe el numero en su columna
    for i = 0 until i < 9:
        if grid[i][column] == n:
            return false

    # obtener la primera fila y columna del cuadro del numero actual
    startRow = row - row mod 3
    starCol = column - column mod 3

    # revisar si existe el numero en su cuadro
    for i = 0 until i < 3:
        for j = 0 until j < 3:
            if grid[startRow + i][starCol + j] == n:
                return false

solveSudoku(row, column, grid):
    # caso base sudoku resuelto
    if column == 9 and row == 8:
        return true

    # revisar si la columna ya esta fuera de alcance para ir a la
    siguiente fila
    if column == 9:
        column = 0
        row++

    # revisar si ya existe un numero en la celda actual
    if grid[row][column] != 0:
        return solveSudoku(row, column + 1, grid)

    # revisar numeros para el indice actual
    for i = 1 until i < 10:
        if isValid(row, column, grid, i):
            grid[row][column] = i

            if solveSudoku(row, column + 1, grid):
```

```
return true
```

```
# si no se encuentra la solucion regresar el valor a 0  
grid[row][column] = 0
```

Análisis de complejidad:

C/C++

```
bool isValid(int row, int column, vector<vector<int>> &grid, int n) {  
    for (int i = 0; i < 9; i++) { 0(9)  
        if (grid[row][i] == n) {  
            return false;  
        }  
    }  
  
    for (int i = 0; i < 9; i++) { 0(9)  
        if (grid[i][column] == n) { 0(1)  
            return false; 0(1)  
        }  
    }  
  
    int startRow = row - row % 3; 0(1)  
    int starCol = column - column % 3; 0(1)  
  
    for (int i = 0; i < 3; i++) { 0(3)  
        for (int j = 0; j < 3; j++) { 0(3x3)  
            if (grid[startRow + i][starCol + j] == n) { 0(1)  
                return false; 0(1)  
            }  
        }  
    }  
  
    return true; 0(1)  
}
```

C/C++

```
bool solveSudoku(int row, int column, vector<vector<int>> &grid) {  
  
    if (column == 9 && row == 8) { 0(1)  
        return true; 0(1)  
    }  
}
```

```

    if (column == 9) { 0(1)
        column = 0; 0(1)
        row++; 0(1)
    }

    if (grid[row][column] != 0) { 0(1)
        return solveSudoku(row, column + 1, grid); 0(9^(9x9))
    }

    for (int i = 1; i < 10; i++) { 0(9)
        if (isValid(row, column, grid, i)) { 0(9)
            grid[row][column] = i; 0(1)

            if (solveSudoku(row, column + 1, grid)) { 0(9^(9x9))
                return true; 0(1)
            }

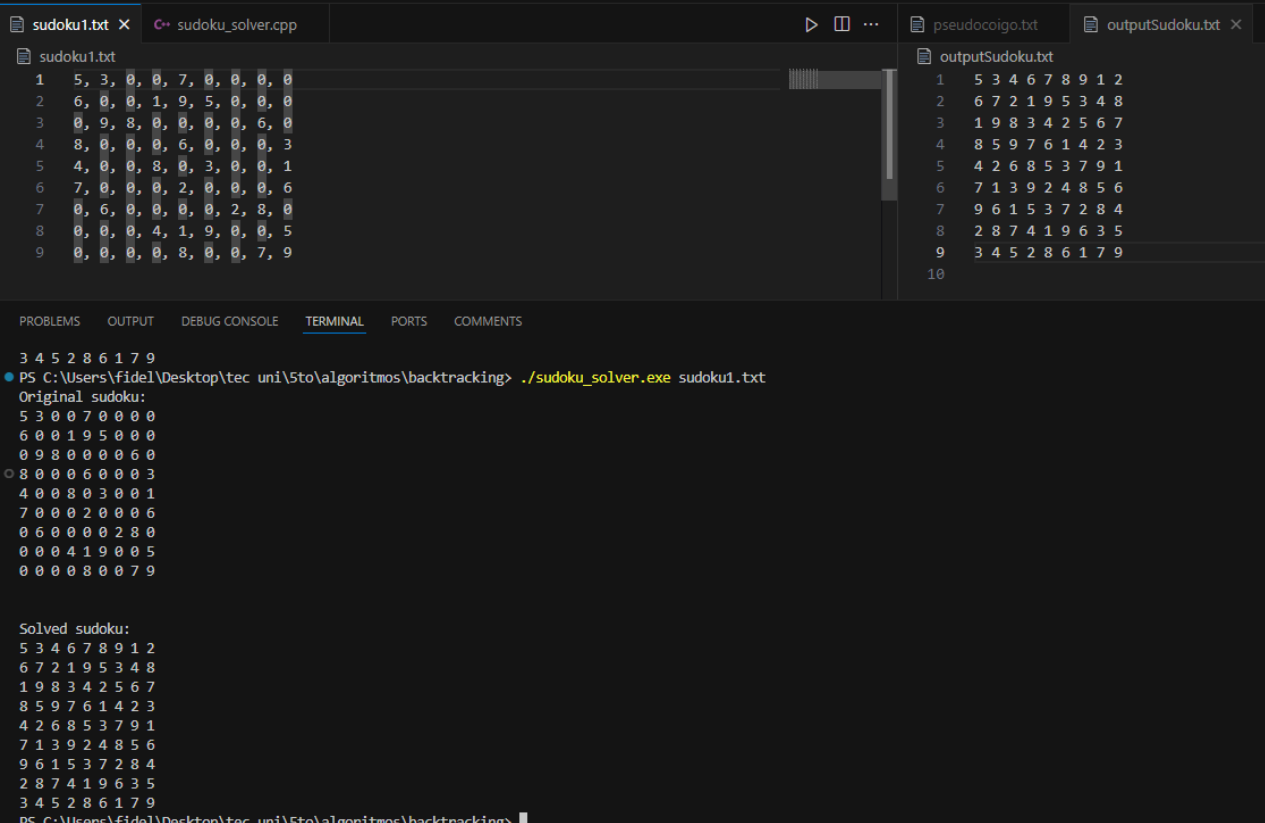
            grid[row][column] = 0; 0(1)
        }
    }

    return false; 0(1)
}

```

$O(9^{(9 \times 9)})$ , El primer nueve viene de los números que se deben de probar desde 1 hasta 9 en cada celda, y  $9 \times 9$  representa todas las celdas de la cuadrícula en donde se debe de probar cada número desde 1 a 9. En este caso como es una función que utiliza recursión y backtracking la manera en la que se obtuvo la complejidad es más con lógica que con algún cálculo matemático como lo es con el teorema maestro o sustitución en divide y vencerás.

## Caso de prueba:

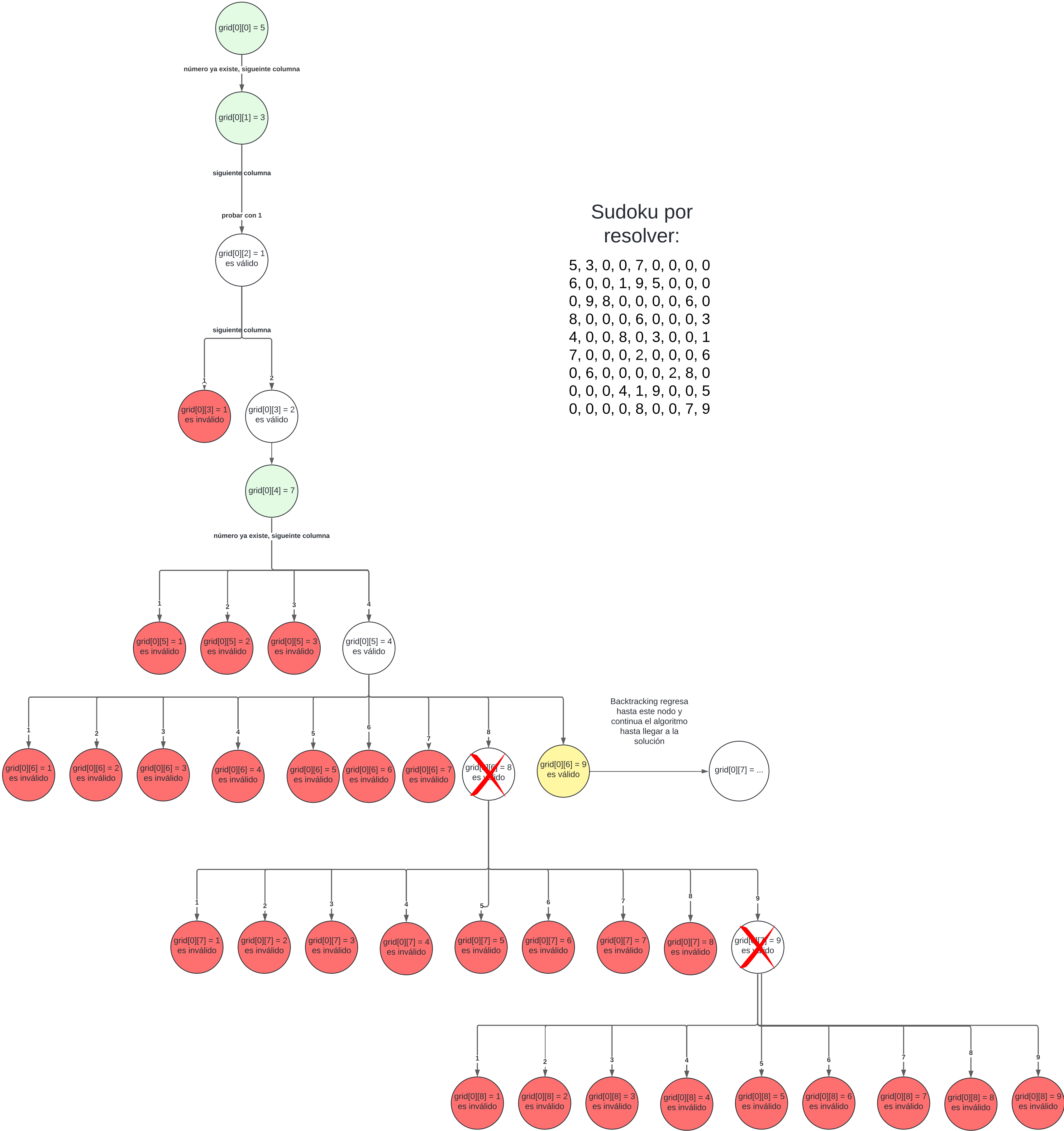


```
sudoku1.txt
1 5, 3, 0, 0, 7, 0, 0, 0, 0
2 6, 0, 0, 1, 9, 5, 0, 0, 0
3 0, 9, 8, 0, 0, 0, 0, 6, 0
4 8, 0, 0, 0, 6, 0, 0, 0, 3
5 4, 0, 0, 8, 0, 3, 0, 0, 1
6 7, 0, 0, 0, 2, 0, 0, 0, 6
7 0, 6, 0, 0, 0, 0, 2, 8, 0
8 0, 0, 0, 4, 1, 9, 0, 0, 5
9 0, 0, 0, 0, 8, 0, 0, 7, 9

outputSudoku.txt
1 5 3 4 6 7 8 9 1 2
2 6 7 2 1 9 5 3 4 8
3 1 9 8 3 4 2 5 6 7
4 8 5 9 7 6 1 4 2 3
5 4 2 6 8 5 3 7 9 1
6 7 1 3 9 2 4 8 5 6
7 9 6 1 5 3 7 2 8 4
8 2 8 7 4 1 9 6 3 5
9 3 4 5 2 8 6 1 7 9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\Fidel\Desktop\tec uni\5to\algoritmos\backtracking> ./sudoku_solver.exe sudoku1.txt
Original sudoku:
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9

Solved sudoku:
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
PS C:\Users\Fidel\Desktop\tec uni\5to\algoritmos\backtracking>
```



## Sudoku por resolver:

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

Ya no hay posibles soluciones, hacer backtracking

5, 3, 1, 2, 7, 4, 8, 9, X  
6, 0, 0, 1, 9, 5, 0, 0, 0  
0, 9, 8, 0, 0, 0, 0, 6, 0  
8, 0, 0, 0, 6, 0, 0, 0, 3  
4, 0, 0, 8, 0, 3, 0, 0, 1  
7, 0, 0, 0, 2, 0, 0, 0, 6  
0, 6, 0, 0, 0, 0, 2, 8, 0  
0, 0, 0, 4, 1, 9, 0, 0, 5  
0, 0, 0, 0, 8, 0, 0, 7, 9

**Bibliografía:**

GeeksforGeeks. (2024, 30 julio). *Algorithm to Solve Sudoku | Sudoku Solver*. GeeksforGeeks.

<https://www.geeksforgeeks.org/sudoku-backtracking-7/>

Inside code. (2022, 3 abril). *Let's make a sudoku solver in 5 minutes (Backtracking) - Inside*

*code* [Video]. YouTube. [https://www.youtube.com/watch?v=eAFcj\\_2quWI](https://www.youtube.com/watch?v=eAFcj_2quWI)