



PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS – PED2

Solucionador de Sudokus.

Paulino Esteban Bermúdez Rodríguez. – 09146352B

pbermudez30@alumno.uned.es

UNED – LAS TABLAS.

Contenido

ENUNCIADO.	3
DISEÑO DEL ALGORITMO.	3
CÓDIGO.	6
RESULTADOS DEL ALGORITMO.	8
BIBLIOGRAFÍA.	9

ENUNCIADO.

El juego del Sudoku consiste en rellenar un cubo de 9 x 9 celdas dispuestas en 9 subgrupos de 3 x 3 celdas, con números del 1 al 9, atendiendo a la restricción de que no se debe repetir el mismo número en la misma fila, columna o subgrupo de 9. Un Sudoku dispone de varias celdas con un valor inicial, de modo que debemos empezar a resolver el problema a partir de esta solución parcial sin modificar ninguna de las celdas iniciales.

La práctica constará de un programa en java que resuelva el problema aplicando el esquema de Vuelta Atrás (Backtracking) junto con una memoria de su implementación.

DISEÑO DEL ALGORITMO.

Sudoku es un rompecabezas de colocación de números donde el objetivo es llenar una cuadrícula de tamaño 'n' con números comprendidos entre 1 y 'n'. Los números deben colocarse de forma que cada fila, cada columna y cada una de las subcuadrículas contengan todos los números del 1 al 'n'.

En el caso de la presente práctica, la cuadrícula es de 9x9, las rejillas están parcialmente llenas con números ya fijados con el fin de garantizar que se pueda llegar a una solución. Se usará como caso base, el sudoku propuesto en el enunciado, sin embargo, el programa puede resolver cualquier tipo de sudoku de 9x9, siempre y cuando los datos introducidos den una solución real.

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

Ilustración 1 Sudoku Inicial.

Para realizar la creación del algoritmo, primero use las técnicas de eliminar números y buscar candidatos únicos, usando la siguiente guía como ayuda: [Técnicas de resolución de Sudokus.](#)

Para el algoritmo, se usa un 'árbol de posibilidades', donde en su nodo raíz tengo el Sudoku a resolver y cada nodo es un candidato y cada 'hoja' naciente son los posibles candidatos a la solución.

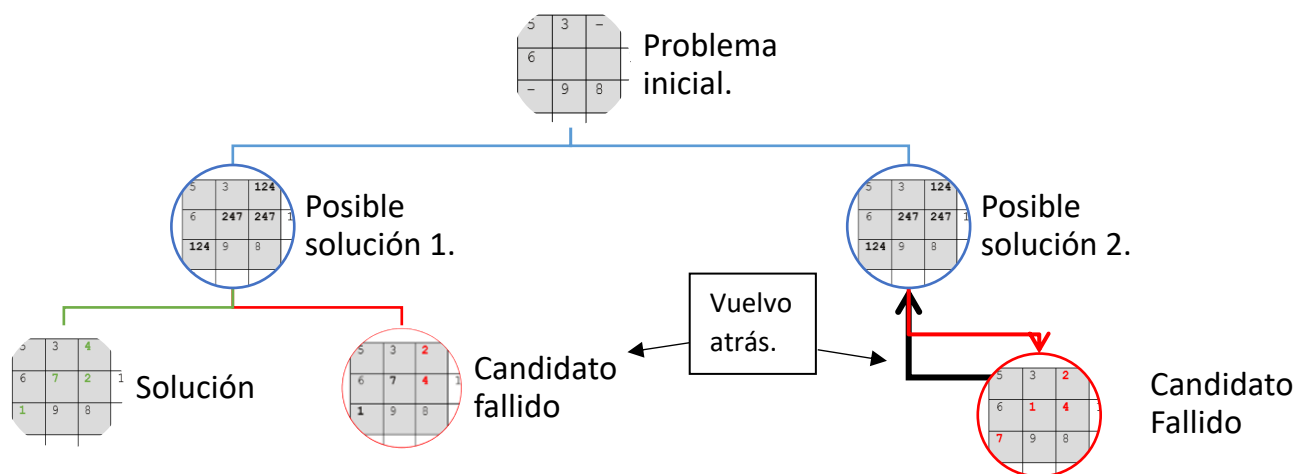


Ilustración 2 Backtracking del caso base. Usando la forma de árbol.

Se **define la meta** para verificar la solución, una vez encontrada la meta, la búsqueda termina, una cuadrícula completamente llena es válida si:

1. Cada fila tiene todos los números del 1 al 9.
2. Cada columna tiene todos los números del 1 al 9.
3. Cada subcuadrícula tiene todos los números del 1 al 9.

Restricciones para definir un candidato son:

1. Cada fila tiene números únicos del 1 al 9 o espacios con el símbolo '- '.
2. Cada columna tiene números únicos del 1 al 9 o '- '.
3. Cada subcuadrícula tiene números únicos del 1 al 9 o guiones.

La **condición de terminación**, por lo general, son:

1. No quedan espacios con guiones '- ' por llenar y el candidato aún no se califica como la solución - Subcuadrículas.
2. No hay espacios con guiones, es decir la cuadrícula ya está completamente llena.
3. El sudoku no tiene solución.


Algoritmo paso a paso:

Es la forma en que el programa 'adivina' el número solución de la casilla correspondiente:

1. Hacer una lista de todos los espacios con guiones.
2. Seleccione un lugar coloque un número, entre 1 y 9, en él y valide la cuadrícula candidata.
3. Si alguna de las restricciones falla, abandone el número candidato y repita el paso 2º con el siguiente número. De lo contrario, compruebe si el Sudoku tiene una solución válida.
4. Si encuentra una solución válida, para el algoritmo. De lo contrario, repita los pasos 2 al 4.

Demostración con un caso base, considerando como única cuadrícula el primer cuadrado del Sudoku Base (Cuadro rojo en Ilustración 3).

5	3	-
6	-	-
-	9	8



1	2	-
3	-	-
-	3	1

Ilustración 4 Sudoku 3x3 de uno de 9x9 transformado a un caso base.

Comenzamos enumerando todos los espacios con guiones. Etiquetamos cada celda con una posición vectorial (x,y) para poder identificarlas y marcamos la primera celda (esquina superior izquierda) como (1,1). Obteniendo así nuestro espacio de ubicaciones por asignar.

(1,3) (2,2) (2,3) (3,1)

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

Ilustración 3 Solución sudoku base

Ahora seleccionamos el primer lugar (1,3) para trabajar. Y realizamos las pruebas con todos los valores posibles, al ser un Sudoku de 3x3, tengo los valores del 1 al 3.

1	2	1
3	-	-
-	3	1

1	2	2
3	-	-
-	3	1

1	2	3
3	-	-
-	3	1

Ilustración 5 Prueba de valores posibles

De la misma forma lo realizamos con el resto de casillas, hasta obtener alguna condición de terminación.

1	2	3
3	1	2
2	3	1

Ilustración 6 Sudoku solución

Usando este caso base, realizamos la misma secuencia para el sudoku de 9x9, pero añadiendo una doble condición extra, debe comprobar que el número introducido en la casilla (x, y), la fila y la columna, no lo repita.

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

Ilustración 7 Sudoku base de 9x9, 4 comprueba fila y columna para que no se repita y es válido para su subcuadrícula, al igual que lo hacen los 4's de las 2da y 3ra filas.

CÓDIGO.

```
public static boolean resuelveSudoku(
    int[][] tablero, int n)
{
    int fila = -1;
    int columna = -1;
    boolean esVacia = true;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (tablero[i][j] == 0) {
                fila = i;
                columna = j;

                esVacia = false;
                break;
            }
        }
        if (!esVacia) {
            break;
        }
    }

    if (esVacia) {
        return true;
    }

    for (int num = 1; num <= n; num++) {
        if (esSeguro(tablero, fila, columna, num)) {
            tablero[fila][columna] = num;
            if (resuelveSudoku(tablero, n)) {
                return true;
            }
        }
        else {
            tablero[fila][columna] = 0;
        }
    }
    return false;
}

public static void print(
    int[][] tablero, int N)
{
    for (int r = 0; r < N; r++) {
        for (int d = 0; d < N; d++) {
            System.out.print(tablero[r][d]);
            System.out.print(" ");
        }
        System.out.print("\n");

        if ((r + 1) % (int)Math.sqrt(N) == 0) {
            System.out.print("");
        }
    }
}
```

Ilustración 8 Clase ejecutora que soluciona el Sudoku.

```
int N = tablero.length;

if (resuelveSudoku(tablero, N)) {
    print(tablero, N);
}else {
    System.out.println("Sin solución");
}
}
```

Ilustración 9 Clase Main

RESULTADOS DEL ALGORITMO.

Tablero inicial:									
5	3	0	0	7	0	0	0	0	0
6	0	0	1	9	5	0	0	0	0
0	9	8	0	0	0	0	0	6	0
8	0	0	0	6	0	0	0	0	3
4	0	0	8	0	3	0	0	0	1
7	0	0	0	2	0	0	0	0	6
0	6	0	0	0	0	2	8	0	0
0	0	0	4	1	9	0	0	0	5
0	0	0	0	8	0	0	0	7	9
Solucion:									
5	3	4	6	7	8	9	1	2	0
6	7	2	1	9	5	3	4	8	0
1	9	8	3	4	2	5	6	7	0
8	5	9	7	6	1	4	2	3	0
4	2	6	8	5	3	7	9	1	0
7	1	3	9	2	4	8	5	6	0
9	6	1	5	3	7	2	8	4	0
2	8	7	4	1	9	6	3	5	0
3	4	5	2	8	6	1	7	9	0

Ilustración 11 Solución Sudoku Modelo: Default

Tablero inicial:									
0	6	0	1	0	4	0	5	0	0
0	0	8	3	0	5	6	0	0	0
2	0	0	0	0	0	0	0	0	1
8	0	0	4	0	7	0	0	0	6
0	0	6	0	0	0	3	0	0	0
7	0	0	9	0	1	0	0	0	4
5	0	0	0	0	0	0	0	0	2
0	0	7	2	0	6	9	0	0	0
0	4	0	5	0	8	0	7	0	0
Solucion:									
9	6	3	1	7	4	2	5	8	0
1	7	8	3	2	5	6	4	9	0
2	5	4	6	8	9	7	3	1	0
8	2	1	4	3	7	5	9	6	0
4	9	6	8	5	2	3	1	7	0
7	3	5	9	6	1	8	2	4	0
5	8	9	7	1	3	4	6	2	0
3	1	7	2	4	6	9	8	5	0
6	4	2	5	9	8	1	7	3	0

Ilustración 10 Solucion Sudoku Modelo: Prueba

BIBLIOGRAFÍA.

Libro de texto base de la asignatura.

<https://www.kristanix.com/sudoku/sudoku-solving-techniques.php>

<http://www.playsudoku.biz/tecnicas-solucion-sudokus>

<https://www.baeldung.com/cs/backtracking-algorithms>