

# Modelos y algoritmos II

- Consignas TP -

## Importante

Puede utilizar contenedores genéricos Stack<T> y Queue<T> de C#, pero debe limitarse solo a las operaciones básicas: Peek, Count, iteración con foreach, Pop, Push, Enqueue, Dequeue.

Pueden usarse arrays estáticos, List<T>, LinkedList<T> y Dictionary<K,T> de C# con todos sus métodos y propiedades.

**No lo deje para último momento.  
Si necesita ayuda, pídale.**

## Criterio de evaluación

1. Debe tener 3 de las 5 consignas implementados correctamente (no perfectamente) para poder aprobar.
2. Debe tener cada punto implementado perfectamente para conseguir el 10.
3. 1 punto va ser la prolijidad y la optimización del código desarrollado.
4. Si se copia, le queda un 1.

## Proyecto base

Se provee un proyecto base para que usted enfoque y dedique su tiempo a los algoritmos y estructuras, y no así en detalles estéticos u operativos irrelevantes a la materia.

[https://drive.google.com/file/d/1DxJflbM\\_cH-SCGGRNULfkSdZTxrh4HbZ/view?usp=sharing](https://drive.google.com/file/d/1DxJflbM_cH-SCGGRNULfkSdZTxrh4HbZ/view?usp=sharing)

Para el ejercicio se provee un proyecto con assets e scripts incompletos en la carpeta.

## Ejercicio 1: Creador y resolutor de sudokus

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

### Introducción

El sudoku es un rompecabezas lógico de combinación de números.

Normalmente es una grilla de 9x9 celdas subdividida en secciones de 3x3 que se encuentra parcialmente resuelta y el juego consiste en rellenarla con el resto de los números.

Los números van del 1 al 9.

Se deben cumplir tres condiciones para que el sudoku sea **válido**:

- Cada columna no debe tener números repetidos
- Cada fila no debe tener números repetidos
- Cada sección (de 3x3) no debe tener números repetidos.

Y para **completarlo** se debe cumplir la última

- No debe haber celdas vacías.

Ejemplo del sudoku anterior resuelto:

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

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

Mire la posición del 7 marcada violeta: No hay números repetidos en su fila, columna ni sección.

Ejemplos

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

Paso 756/1197 - MEM: 7.49MB

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

Paso 556/556 - MEM: 7.19MB

## Guía del proyecto base (Sudoku)

Hay incluida una clase Cell que **no debe modificar su código**. Se le puede marcar como “fija” (locked) que hará su fondo gris, marcar como inválida (invalid) la pondrá roja, y mediante un número (number) se le asignará un número. Al asignar a number el valor “Cell.EMPTY” (constante igual a 0) se interpretará como celda vacía y no mostrará ningún número.

La clase Sudoku es la que debe modificar y donde se enfocará el grueso de la lógica del ejercicio.

## CONSIGNAS

### 1) Implementación de matriz extendida

[ 1 punto ]

- Implemente Matrix. **Debe usar esta clase para resolver el ejercicio.** Puede expandirla a gusto.
- Cargue los valores en la matriz de Cells“\_board” con una matriz de enteros
- Pruebe si las matrices proporcionadas de ejemplo concuerdan con el funcionamiento esperado (inválidas/válidas, inválidas completas, válidas completas)
- Consejo, implemente el filtro “get range” que retorne un rango x, y en un array/lista plana
- Pista: No intente hacer una matriz de matrices, se complicará sin sentido. Use una y construya funciones para su manejo con mayor comodidad.

### 2) Resolvedor de sudokus instantáneo

[ 3 puntos ]

- Aun siendo instantánea, debe pasar por todos los pasos para resolver el puzzle.
- Implementar una solución usando **exclusivamente funciones recursivas haciendo backtracking**
- Contar la cantidad de pasos realizados y mostrarlas en el texto “feedback”.
- Para este punto no utilice co-rutinas ni generators. La resolución debe ejecutarse instantáneamente, pero debe guardar todos sus pasos para luego reproducirlos.
- Puede utilizar los Tests.validBoards proporcionados como tableros base para sus pruebas.
- Tenga en cuenta cargar números “locked” (grises) del tablero base y no modificar nunca esos durante el procesamiento.

### 3) Resolvedor de sudokus paso por paso (con delay)

[ 1 punto - depende de consigna 2 ]

- Utilizar los pasos guardados en la función anterior.
- Utilizar stepDuration para demorar cada transición.
- Mostrar en qué paso se encuentra.

#### 4) Creador de sudokus

**[ 2 puntos - depende de consigna 2 ]**

Un creador de sudokus es esencialmente “resolver un tablero vacío” y luego ahuecarlo.

- Utilice el algoritmo anterior para resolver un tablero vacío.
- Utilice la variable “difficulty” para determinar cuántas celdas deberán quedar vacías.
- Asegúrese de usar un orden aleatorio de prueba ya que si no creará siempre el mismo sudoku si mantiene una secuencia numérica fija.

#### 5) Bonus

**[ 2 puntos - depende de consigna 2 ]**

- Agregar una variable configurable para determinar el número de secciones y con ello crear sudokus de tamaño arbitrario.  
Si en vez de “3” usamos 4, tendremos un tablero de 16x16 y los números irán del 1 al 16.