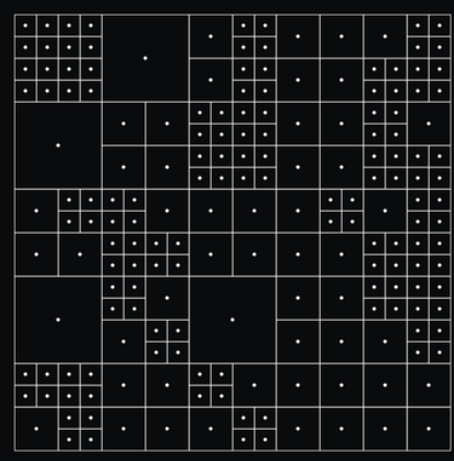


TÉCNICAS ALGORÍTMICAS: SUDOKU



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

```
// Sidebar.jsx

import { useRef, useState } from "react";

import "./styles.css";

const menuItems = [
  {
    name: "Home",
    icon: "home",
  },
]
```

J O N A T A N D A V I D M A C D O N A L R O D A S
M A T R Í C U L A : 2 3 0 3 0 0 7 7 8



INTRODUCCIÓN

Como parte del proyecto final de Técnicas Algorítmicas, presento la resolución del Sudoku mediante un algoritmo eficiente, utilizando una de las técnicas algorítmicas avanzadas propuestas: programación dinámica, divide y vencerás, o algoritmos voraces. El objetivo principal de este proyecto es determinar cuál de estas técnicas ofrece la solución más óptima para abordar el problema del Sudoku, considerando tanto la complejidad computacional como la facilidad de implementación en el contexto específico de este desafío.

El proyecto tiene como objetivo no solo desarrollar una solución algorítmica, sino también evaluar el rendimiento de las técnicas seleccionadas. Esto incluye un análisis detallado de la complejidad temporal y espacial de los algoritmos implementados, así como la medición del tiempo de ejecución para la resolución de un Sudoku determinado. Para ello, se debe elegir la técnica que mejor se adapte a la naturaleza del problema, justificando la elección en función de su eficiencia, la claridad en su implementación y su capacidad para manejar el crecimiento exponencial de las soluciones posibles en el Sudoku. Además, se deben comparar brevemente las otras técnicas disponibles y explicar por qué no fueron seleccionadas, en términos de su viabilidad para resolver el problema de forma eficiente.

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

SUDOKU

El Sudoku es básicamente un juego, es como un rompecabezas numérico que consiste en una cuadrícula de 9x9 celdas dividida en 9 subcuadrículas de 3x3. El objetivo del juego es llenar la cuadrícula con números del 1 al 9 de manera que en cada fila, cada columna y cada subcuadrícula de 3x3 no se repitan números. El Sudoku viene con algunas celdas ya llenas, y el desafío radica en completar el tablero respetando estas restricciones. Lo que se busca con el proyecto, es que mi programa resuelva este sudoku, y que el usuario (yo) pueda observar el proceso.

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

TÉCNICAS ALGORÍTMICAS

Para resolver el sudoku, voy a aplicar las 3 técnicas algorítmicas propuestas (algoritmos voraces, divide y vencerás, y programación dinámica), analizándolas y eligiendo la técnica más óptima.

La **programación dinámica**, aunque no es la técnica más intuitiva para resolver un Sudoku, podría aplicarse dividiendo el problema en subproblemas más pequeños, como el cálculo de posibles valores para celdas específicas en función de las restricciones de filas, columnas y subcuadrículas. Cada solución parcial podría ser almacenada en una tabla para evitar recalcular los mismos resultados y optimizar la búsqueda. Por otro lado, la técnica de **divide y vencerás** podría aplicarse dividiendo el Sudoku en subcuadrículas o filas, resolviendo cada una de forma independiente y luego combinando los resultados, aunque la interdependencia entre las partes del puzzle podría complicar su implementación. De igual manera, los **algoritmos voraces** podrían ser útiles en un enfoque de búsqueda, donde en cada paso se elige el número más prometedor para cada celda, tomando decisiones basadas en las restricciones más inmediatas y avanzando hacia la solución de manera iterativa.

Analizando estas 3 técnicas, he elegido como principal la técnica de divide y vencerás, la cual me resulta muy óptima e indicada para resolver un sudoku

TÉCNICA ELEGIDA

Elegí la técnica de **divide y vencerás** para resolver el Sudoku debido a su enfoque eficiente y estructurado al abordar problemas complejos. Esta técnica permite dividir el Sudoku en subproblemas más pequeños y manejables, lo que facilita la resolución de cada parte de manera más directa. En el contexto del Sudoku, se puede aplicar dividiendo el tablero en subcuadrículas, filas o columnas y luego trabajar de forma independiente sobre estos componentes.

La ventaja principal de esta técnica radica en su capacidad para reducir la complejidad del problema al permitir que cada parte sea resuelta por separado antes de combinar las soluciones. Esto no solo hace que el algoritmo sea más claro y fácil de implementar, sino que también facilita la paralelización de ciertos pasos. La recursividad que emplea divide y vencerás es clave para garantizar que cada subdivisión del Sudoku se resuelva sin perder de vista la interrelación entre los diferentes elementos del tablero. Esto mejora la eficiencia en la resolución y permite que el algoritmo se enfoque en resolver gradualmente el problema sin tener que volver a revisar las partes previamente resueltas. En resumen, la aplicabilidad de divide y vencerás en el Sudoku se debe a su capacidad para descomponer un problema grande en subproblemas más sencillos, optimizando así la solución y mejorando la claridad del proceso.

Código implementado en VS Code:

```
divide-venceras.py > ...
1  # JONATAN DAVID MACDONAL RODAS - 230300778
2  # DIVIDE Y VENCERÁS
3
4  import matplotlib.pyplot as plt
5  from matplotlib.widgets import Button
6  import numpy as np
7  import time
8
9  # Función para imprimir el Sudoku
10 def imprimir_sudoku(sudoku):
11     for fila in sudoku:
12         print(" ".join(str(celda) if celda != 0 else '.' for celda in fila))
13     print()
14
15 # Verificar si es seguro colocar un número en una celda
16 def es_seguro(sudoku, fila, col, num):
17     # Revisar la fila
18     if num in sudoku[fila]:
19         return False
20     # Revisar la columna
21     if num in [sudoku[f][col] for f in range(9)]:
22         return False
23     # Revisar la subcuadrícula de 3x3
24     inicio_fila, inicio_col = 3 * (fila // 3), 3 * (col // 3)
25     for i in range(inicio_fila, inicio_fila + 3):
26         for j in range(inicio_col, inicio_col + 3):
27             if sudoku[i][j] == num:
28                 return False
29     return True
30
```

```

31 # Resolver Sudoku con divide y vencerás (backtracking global) y visualización paso a paso
32 def resolver_sudoku_divide_vencerás(sudoku, resaltados, ax, fig):
33     for fila in range(9):
34         for col in range(9):
35             if sudoku[fila][col] == 0: # Celda vacía
36                 for num in range(1, 10): # Probar números del 1 al 9
37                     if es_seguro(sudoku, fila, col, num):
38                         sudoku[fila][col] = num
39                         resaltados[fila][col] = True
40                         actualizar_tablero(sudoku, ax, fig, resaltados)
41                         if resolver_sudoku_divide_vencerás(sudoku, resaltados, ax, fig):
42                             return True
43                         sudoku[fila][col] = 0 # Retroceder
44                         actualizar_tablero(sudoku, ax, fig, resaltados)
45                 return False
46     return True
47
48 # Actualizar el tablero en pantalla
49 def actualizar_tablero(sudoku, ax, fig, resaltados):
50     ax.clear()
51
52     # Fondo azul claro para el Sudoku
53     ax.set_facecolor('#a3d1ff')
54
55     # Dibujar las celdas del tablero
56     for i in range(9):
57         for j in range(9):
58             ax.add_patch(plt.Rectangle((j, 8-i), 1, 1, color='#a3d1ff', ec='none'))
59

```

```

60 # Dibujar las líneas del tablero
61 for i in range(10):
62     lw = 2 if i % 3 == 0 else 0.5 # Líneas más gruesas para las subcuadrículas
63     ax.axhline(i, color="#333333", lw=lw) # Las líneas horizontales
64     ax.axvline(i, color="#333333", lw=lw) # Las líneas verticales
65
66 # Rellenar los números
67 for i in range(9):
68     for j in range(9):
69         if sudoku[i][j] != 0:
70             color = "blue" if resaltados[i][j] else "black"
71             ax.text(j + 0.5, 8 - i + 0.5, str(sudoku[i][j]),
72                    ha="center", va="center", fontsize=16, color=color)
73
74 ax.set_xlim(0, 9)
75 ax.set_ylim(0, 9)
76 ax.axis("off")
77 fig.canvas.draw()
78 plt.pause(0.05) # Aumentar o reducir la pausa según sea necesario
79

```

```

80 # Función que se activa con el botón "Resolver Sudoku"
81 def on_resolver_click(event):
82     resaltados = np.zeros((9, 9), dtype=bool) # Mantener los números resaltados
83     print("Sudoku inicial:")
84     imprimir_sudoku(sudoku_inicial)
85
86     inicio = time.time()
87     resolver_sudoku_divide_vencerás(sudoku_inicial, resaltados, ax, fig)
88     fin = time.time()
89
90     print("Sudoku resuelto:")
91     imprimir_sudoku(sudoku_inicial)
92     print(f"Tiempo de ejecución: {fin - inicio:.2f} segundos")
93
94 # Tablero inicial del Sudoku
95 sudoku_inicial = [
96     [5, 3, 0, 0, 7, 0, 0, 0, 0],
97     [6, 0, 0, 1, 9, 5, 0, 0, 0],
98     [0, 9, 8, 0, 0, 0, 0, 6, 0],
99     [8, 0, 0, 0, 6, 0, 0, 0, 3],
100    [4, 0, 0, 8, 0, 3, 0, 0, 1],
101    [7, 0, 0, 0, 2, 0, 0, 0, 6],
102    [0, 6, 0, 0, 0, 0, 2, 8, 0],
103    [0, 0, 0, 4, 1, 9, 0, 0, 5],
104    [0, 0, 0, 0, 8, 0, 0, 7, 9]
105 ]

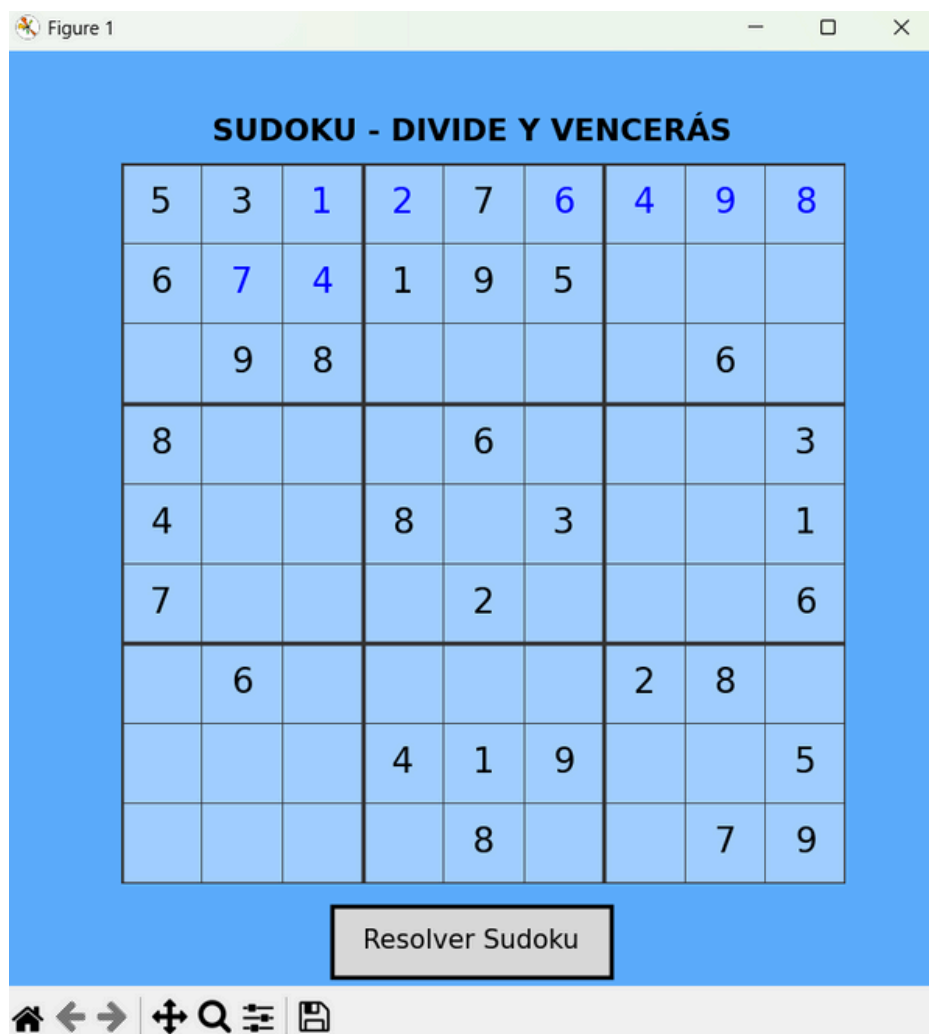
```

```

105 ]
106
107 # Crear la figura y el tablero inicial
108 fig, ax = plt.subplots(figsize=(6, 6))
109
110 # Cambiar el color de fondo de la interfaz
111 fig.patch.set_facecolor('#5DAEFE')
112
113 # Añadir el texto con tu nombre y número de matrícula (ajustado)
114 fig.text(0.5, 1.05, 'JONATAN DAVID MACDONAL RODAS - 230300778', ha='center', va='center', fontsize=10, color='black')
115
116 # Título (mover hacia abajo para no solaparse con el texto)
117 fig.suptitle("SUDOKU - DIVIDE Y VENCERÁS", fontsize=14, fontweight='bold', y=0.93)
118
119 resaltados = np.zeros((9, 9), dtype=bool)
120 actualizar_tablero(sudoku_inicial, ax, fig, resaltados)
121
122 # Crear el botón "Resolver Sudoku"
123 ax_button = plt.axes([0.35, 0.01, 0.3, 0.075]) # Botón más grande: ajustar ancho y posición
124 btn_resolver = Button(ax_button, 'Resolver Sudoku')
125
126 # Ajustar estilo del botón
127 btn_resolver.label.set_fontsize(12) # Reducir tamaño de fuente
128 btn_resolver.ax.set_facecolor('#a3d1ff') # Fondo azul claro
129 btn_resolver.label.set_color('black') # Cambiar color de texto a negro
130 btn_resolver.ax.patch.set_edgecolor('black')
131 btn_resolver.ax.patch.set_linewidth(2)
132
133 btn_resolver.on_clicked(on_resolver_click)
134
135 # Mostrar la interfaz
136 plt.show()

```

Output (interfaz del sudoku como ventana externa)



La aplicación del divide y vencerás en mi código puede verse en la función `resolver_sudoku_divide_vencerás`. Esta función recursiva representa la esencia de dividir el problema del Sudoku en subproblemas más pequeños, que son las celdas vacías del tablero. A medida que el algoritmo avanza, prueba números del 1 al 9 para cada celda vacía, y divide el problema al abordar cada celda de manera independiente, verificando si un número es seguro para colocar en la posición dada.

El proceso de dividir el problema en partes más pequeñas y resolver cada parte de forma recursiva se refleja cuando el algoritmo intenta asignar un número a una celda, realiza una comprobación de seguridad (la función `es_seguro`), y luego resuelve el siguiente subproblema: la siguiente celda vacía. Si encuentra que no es posible continuar con una asignación, retrocede (backtracking) y vuelve a intentar con otro número, lo cual también refleja una estrategia de divide y vencerás, ya que está descomponiendo el problema de manera que puede volver a abordar subproblemas más pequeños de forma independiente.

```
31 # Resolver Sudoku con divide y vencerás (backtracking global) y visualización paso a paso
32 def resolver_sudoku_divide_vencerás(sudoku, resaltados, ax, fig):
33     for fila in range(9):
34         for col in range(9):
35             if sudoku[fila][col] == 0: # Celda vacía
36                 for num in range(1, 10): # Probar números del 1 al 9
37                     if es_seguro(sudoku, fila, col, num):
38                         sudoku[fila][col] = num
39                         resaltados[fila][col] = True
40                         actualizar_tablero(sudoku, ax, fig, resaltados)
41                     if resolver_sudoku_divide_vencerás(sudoku, resaltados, ax, fig):
42                         return True
43                     sudoku[fila][col] = 0 # Retroceder
44                     actualizar_tablero(sudoku, ax, fig, resaltados)
45         return False
46     return True
```

¿ POR QUÉ NO OTRA TÉCNICA ?

Una de las técnicas con las que tuve un debate fue principalmente algoritmos voraces, ya que es útil cuando se quiere priorizar la rapidez (tiempo). A pesar de que divide y vencerás suele llevarse un tiempo de entre 10 y 20-25 minutos, no utilicé **algoritmos voraces** para resolver el Sudoku debido a que, aunque esta técnica es generalmente eficaz cuando se busca una solución rápida y sencilla, no se adapta bien a la complejidad y las restricciones de un Sudoku. Los algoritmos voraces funcionan tomando decisiones locales óptimas en cada paso, lo que puede ser eficiente en muchos problemas. Sin embargo, en el caso del Sudoku, las decisiones locales pueden no llevar a una solución global válida, ya que existe una interdependencia entre las celdas del tablero, y tomar una decisión rápida en una celda puede no ser compatible con las restricciones en otras partes del tablero.

Además, dado que un Sudoku requiere explorar todas las posibilidades y asegurar que todas las reglas (filas, columnas y subcuadrículas) se cumplan simultáneamente, un enfoque voraz no es lo suficientemente robusto para garantizar una solución correcta en todos los casos. En contraste, la técnica de divide y vencerás, mediante el uso de backtracking, ofrece una forma más controlada y exhaustiva de abordar el problema, dividiendo el Sudoku en subproblemas que se resuelven recursivamente, lo que permite explorar todas las posibles configuraciones y retroceder cuando sea necesario.

Output (código algoritmos voráces)

```
PS C:\Users\macdo\OneDrive\Documentos\CURSO PYTHON> & C:/Users/macdo/AppData/Local/
ON/voraces.py"
Sudoku inicial:
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

Sudoku resuelto:
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

Tiempo de ejecución: 86.88 segundos
```

Por otro lado, aunque la programación dinámica también podría parecer una opción válida, decidí no usarla porque, aunque es útil para optimizar problemas que involucran subproblemas solapados, no encaja bien con la naturaleza del Sudoku. La programación dinámica se basa en almacenar soluciones parciales para evitar cálculos repetitivos, pero en el caso del Sudoku, la interrelación entre filas, columnas y subcuadrículas hace que no sea fácil dividir el problema en subproblemas independientes y memorizar soluciones parciales de manera eficiente. Además, la programación dinámica no facilita el proceso de retroceder en caso de encontrar una solución incorrecta, lo que es una característica esencial en la resolución de Sudoku.

CONCLUSIÓN

Como conclusión de este proyecto, puedo decir que ha sido un proyecto muy interesante e igual complejo, donde se aprecia el trabajo y lo que se ha visto alrededor del semestre. Al verme a la tarea de resolver un Sudoku utilizando técnicas algorítmicas avanzadas, me di cuenta de la complejidad que involucra encontrar la mejor opción para este tipo de problemas. Aunque las tres técnicas (algoritmos voraces, programación dinámica y divide y vencerás) ofrecen ventajas en ciertos contextos, fue un reto significativo enlazarlas de manera efectiva con la estructura y restricciones del Sudoku. Particularmente, fue complicado aplicar las técnicas de algoritmos voraces y programación dinámica, ya que no se adaptan tan bien a la naturaleza interdependiente del problema, lo que me llevó a elegir finalmente la técnica de divide y vencerás, combinada con backtracking, como la más adecuada.

CÓMO EJECUTAR LOS CÓDIGOS

Para ejecutar mis códigos en un entorno como el de VS Code, es necesario instalar algunas librerías para que se pueda ver la interfaz al ejecutarlos.

Requerimientos:

1. matplotlib: Para la visualización gráfica del Sudoku.
2. numpy: Para manipulación de matrices, útil para la representación del tablero de Sudoku.

Primero, se debe verificar que está instalado correctamente Python en la computadora, a través de ejecutar **python --version** en la terminal

Para instalar matplotlib, primero se debe instalar pip (si no se tiene instalado). La primera opción sería agregarlo al PATH de la computadora, o de igual forma copiar el código que está en <https://bootstrap.pypa.io/get-pip.py> y / o descarga el archivo get-pip.py. También se puede instalar a través de la ejecución de python **get-pip.py** en la terminal.

Una vez instalado pip, ya se puede instalar las librerías, llendo a la terminal (ya sea en VS Code o fuera del programa) y ejecutando **pip install matplotlib numpy**

De esta manera, ya se tiene instaladas todas las librerías y ya se puede ejecutar los códigos en el entorno de VS Code