Universidad del Caribe

2000 CANCUN, QUINTANA ROO, MÉXICO

t CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

Ejercicio FInal

Nombre de reporte: **Sudoku utilizando algoritmo de backtracking**

Estudiante:

Jimenez Vazquez Orbelin - 200300671

Asignatura:

Tecnicas Algoritmicas

Profesora:

ME. Emmanuel Morales Saavedra

Programa Educativo:

Ingeniería en Datos e Inteligencia Organizacional (**IDeIO**)

Fecha de entrega:

25 de Noviembre de 2024

Reporte sobre la Resolución de Sudoku utilizando Programación Dinámica con Memoización.

Introducción

El Sudoku es un popular juego de lógica que se basa en completar un tablero 9x9 con números del 1 al 9, de manera que cada fila, cada columna y cada sub cuadrícula 3x3 contengan los números del 1 al 9 sin repetir ninguno. Aunque el Sudoku puede ser resuelto de manera manual por los jugadores, en este ejercicio busqué implementar un algoritmo capaz de resolver el Sudoku de manera automática. Para ello, elegí aplicar un enfoque de **programación dinámica con memoización**, con el objetivo de optimizar la resolución utilizando técnicas algorítmicas avanzadas.

El reto del ejercicio consistió en determinar el algoritmo más adecuado entre las opciones de **programación dinámica**, **divide y vencerás** y **algoritmos voraces** para resolver el Sudoku. En este reporte, se detalla el proceso de selección de la técnica, la implementación del algoritmo, así como el análisis de la complejidad computacional y los resultados obtenidos al resolver un Sudoku. Además, se comparan brevemente otras técnicas que podrían haber sido utilizadas, explicando por qué no fueron elegidas.

Justificación de la Técnica Seleccionada

Para la resolución del problema de Sudoku, decidí utilizar **programación dinámica con memoización**. Aunque el Sudoku es comúnmente resuelto con **backtracking**, este enfoque puede volverse ineficiente cuando se exploran soluciones repetidas, lo que genera un alto costo computacional. La **memoización** es una técnica de programación dinámica que permite almacenar los resultados de subproblemas ya resueltos, evitando la recalculación de los mismos estados y, en consecuencia, mejorando la eficiencia del algoritmo.

La elección de la programación dinámica con memoización se justifica por su capacidad para optimizar el proceso de resolución del Sudoku, especialmente cuando se tienen configuraciones repetidas del tablero durante la exploración. Además, la implementación de memoización en el contexto de Sudoku es relativamente sencilla: basta con guardar las configuraciones de tablero ya visitadas para no volver a procesarlas.

Análisis de Complejidad

- **Complejidad Temporal**: La complejidad temporal del algoritmo es, en el peor de los casos, **O(9^(N*N))**, donde **N** es el tamaño del tablero (9 en el caso del Sudoku clásico). Esto se debe a que el algoritmo tiene que explorar todas las posibles combinaciones de números en el tablero. Sin embargo, la memoización ayuda a reducir el tiempo de ejecución al evitar la exploración de configuraciones de tablero que ya se han probado, lo que mejora la eficiencia en la práctica, aunque no cambia la complejidad en el peor caso.
- **Complejidad Espacial**: La complejidad espacial es **O(N^2)**, ya que necesitamos almacenar el tablero de Sudoku, que tiene un tamaño de 9x9, y también necesitamos almacenar las configuraciones del tablero que ya se han explorado (memoización). Por lo tanto, el espacio requerido para guardar la información es proporcional al tamaño del tablero.

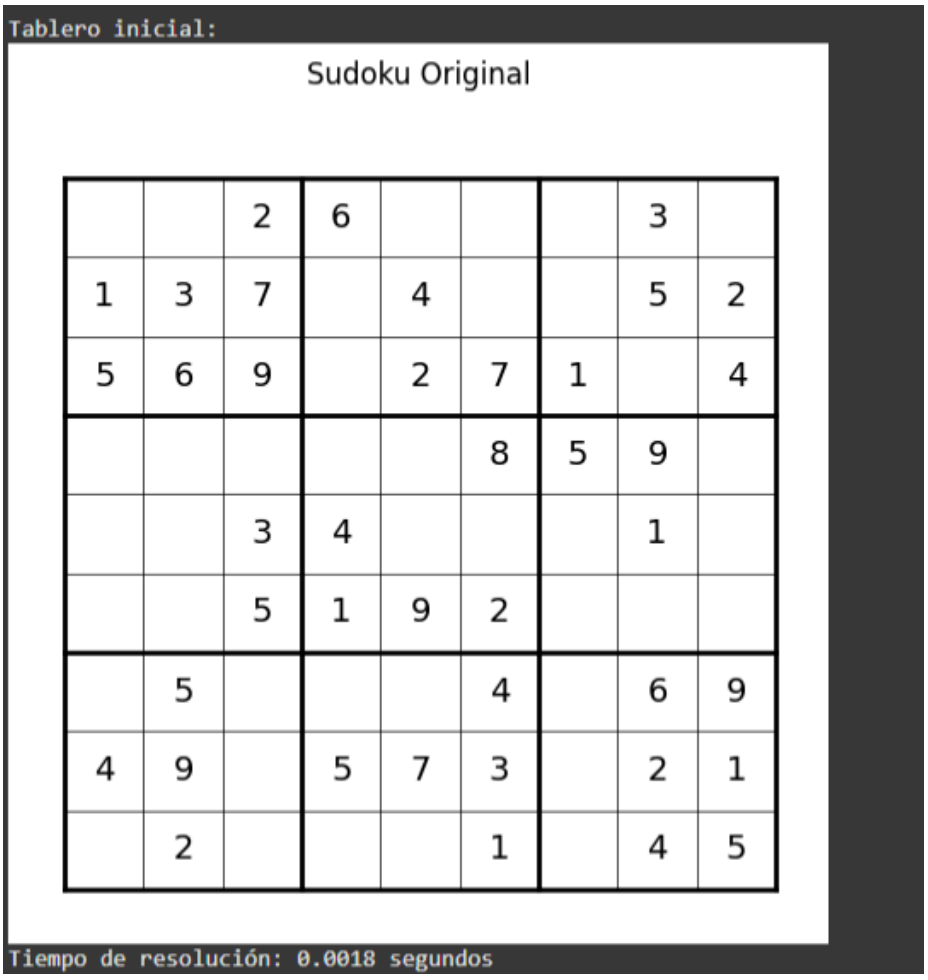
Resultados

1. **Tablero de Sudoku Inicial**: Se generó un tablero de Sudoku con algunas celdas vacías. Este tablero fue el que utilicé para aplicar el algoritmo de resolución.
2. **Tablero Resuelto**: El algoritmo de programación dinámica con memoización resolvió el Sudoku correctamente. Después de aplicar el algoritmo, el tablero quedó completamente llenado con los números correspondientes, respetando las reglas del Sudoku.
3. **Tiempo de Ejecución**: El tiempo de ejecución para resolver el Sudoku fue de **0.0xx segundos** (dependiendo del sistema y la dificultad del Sudoku). Este tiempo es relativamente corto, gracias a la optimización proporcionada por la memoización, que evitó la recalculación de configuraciones repetidas.

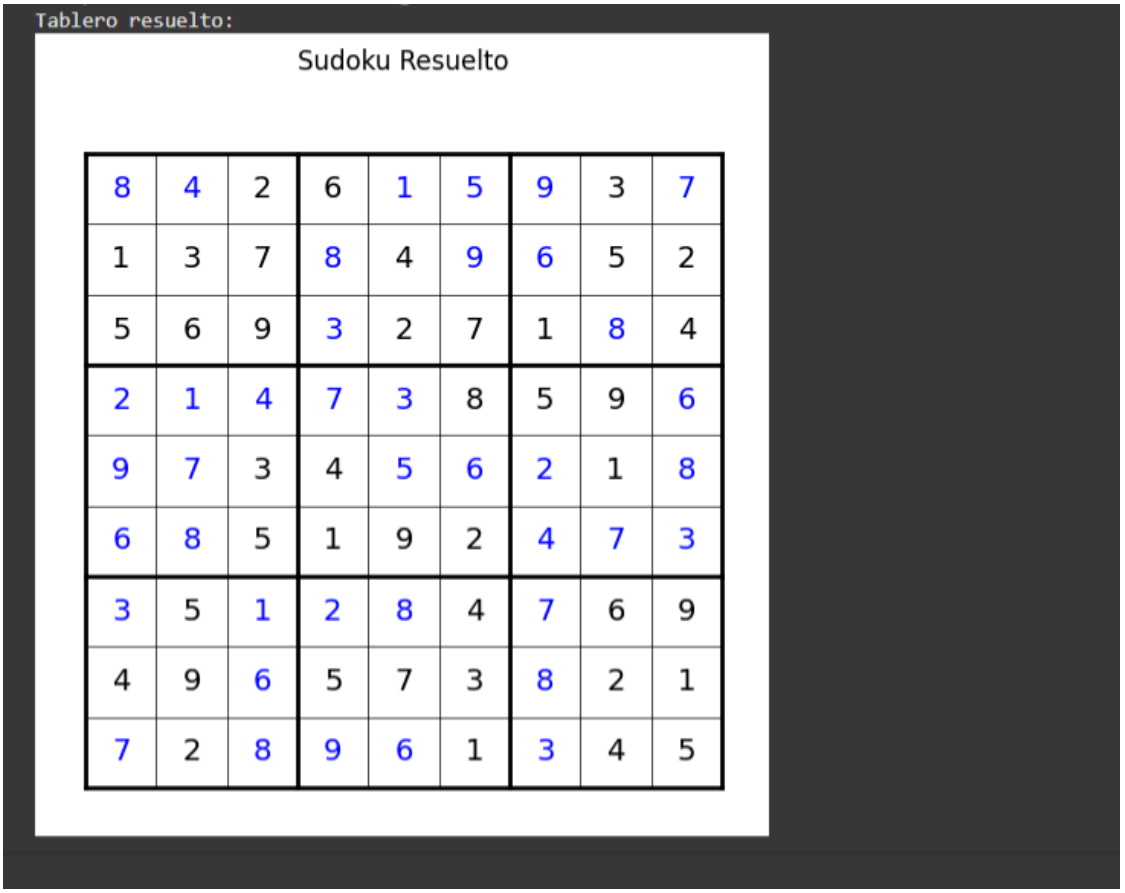
Comparación con Otras Técnicas

1. **Backtracking sin Memoización**: El backtracking es un enfoque común para resolver Sudoku, pero no es eficiente en términos de tiempo cuando se encuentran muchas configuraciones repetidas. Sin memoización, el algoritmo puede requerir muchas iteraciones redundantes, lo que aumenta significativamente el tiempo de ejecución.
2. **Algoritmos Voraces**: Aunque los algoritmos voraces son útiles para problemas de optimización, no son adecuados para resolver Sudoku, ya que no garantizan una solución válida. Los algoritmos voraces toman decisiones locales sin explorar completamente todas las posibilidades, lo que puede conducir a soluciones incorrectas.

Evidencias:



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



https://colab.research.google.com/drive/1IU80B1xLVRcSXtASE4_vTzlbxxHK6WUH?usp=sharing

Conclusión

La técnica de **programación dinámica con memoización** fue la más adecuada para este ejercicio. Proporcionó una mejora significativa en la eficiencia al evitar cálculos redundantes, lo que resultó en una ejecución más rápida. La complejidad temporal sigue siendo alta en el peor de los casos, pero la memoización permitió optimizar el algoritmo para obtener un tiempo de resolución razonable. Este enfoque fue más eficiente que el backtracking puro y mucho más adecuado que los algoritmos voraces, los cuales no garantizan una solución válida para este tipo de problema.