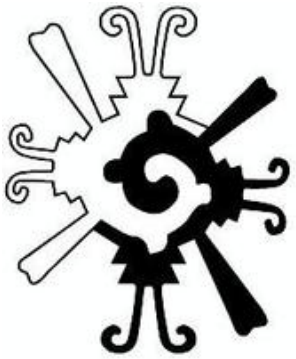


Universidad del Caribe

2000 CANCUN, QUINTANA ROO, MEXICO

CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

Ingeniería en Datos e Inteligencia Organizacional

Programación orientada a objetos.

Congreso de Ingenierías.

Profesor:

Emmanuel Morales Saavedra

Presenta:

190300607 Miranda Avila Karina

Elección de Técnica: Algoritmos Voraces
Entre las técnicas propuestas (programación dinámica, divide y vencerás y algoritmos voraces), los algoritmos voraces son la elección más adecuada para resolver el problema de Sudoku. Esto se debe a las siguientes razones:

Complejidad Computacional: Resolver un Sudoku es un problema NP-completo. Utilizar un algoritmo voraz (basado en backtracking) es más intuitivo y efectivo para buscar soluciones al explorar posibles valores en celdas vacías. Aunque en el peor de los casos se revisen todas las configuraciones posibles, las optimizaciones (como determinar valores válidos basados en restricciones) reducen significativamente el espacio de búsqueda en la práctica.

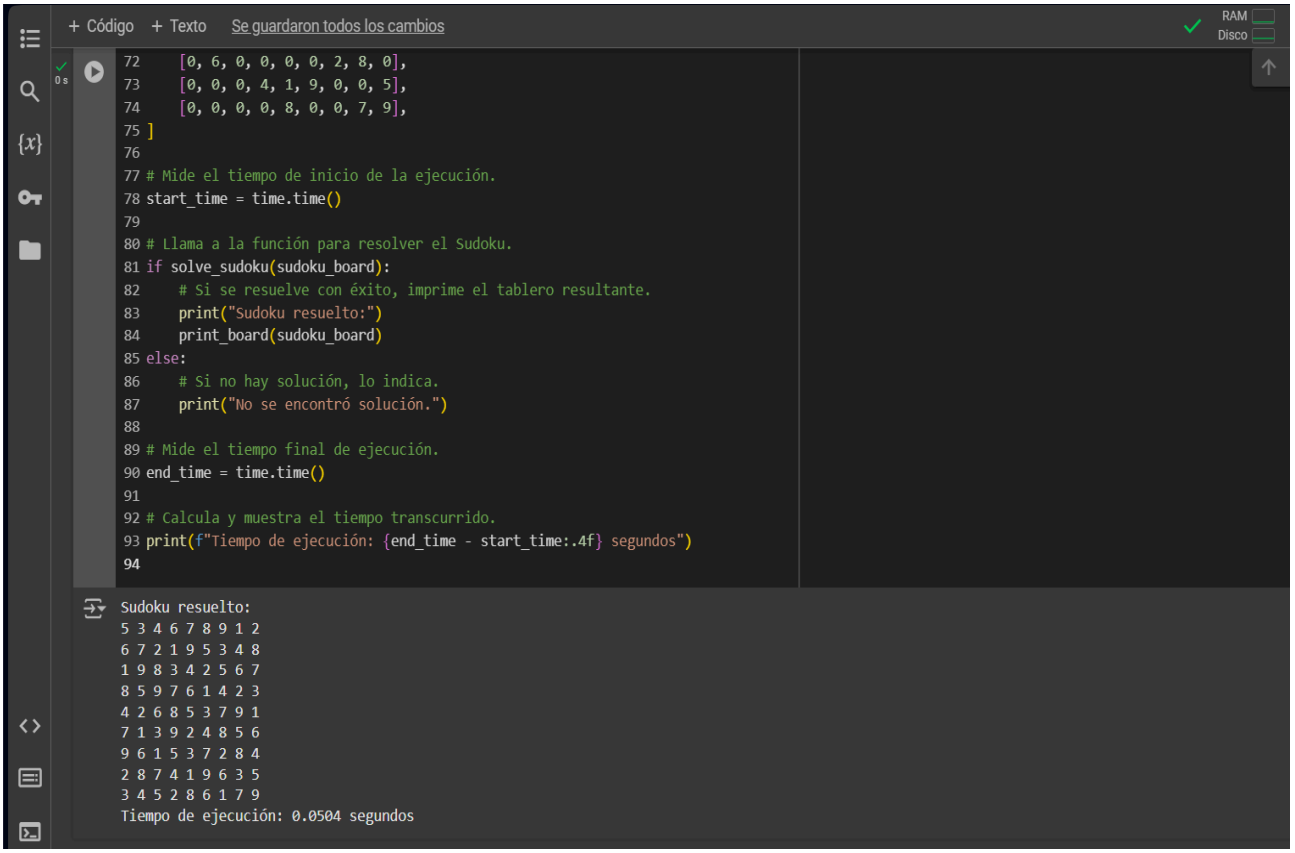
Facilidad de Implementación: Los algoritmos voraces con backtracking se adaptan bien al problema porque permiten buscar soluciones válidas probando alternativas y retrocediendo si una opción no funciona, aprovechando la estructura de restricciones del Sudoku (filas, columnas y subcuadrantes).

Resolución de Sudoku utilizando Backtracking (Algoritmo Voraz)
Justificación de la Técnica Seleccionada:
Se eligió un enfoque basado en algoritmos voraces con backtracking debido a las características del problema de Sudoku:

Restricciones Naturales: El Sudoku presenta restricciones en filas, columnas que permiten podar significativamente el espacio de búsqueda. Esto hace que el backtracking, al explorar decisiones y retroceder en caso de error, sea una opción natural y eficiente.

Facilidad de Implementación: La naturaleza recursiva del algoritmo permite construir soluciones de manera intuitiva al llenar celdas vacías, probando posibles valores válidos y retrocediendo si es necesario.

Otras técnicas como divide y vencerás o programación dinámica no se adaptan bien al problema. Aunque estas técnicas son útiles para problemas estructurados o acumulativos, no ofrecen ventajas significativas para un problema como el Sudoku, que requiere explorar configuraciones completas basadas en restricciones.


+ Código
+ Texto
Se guardaron todos los cambios
RAM
Disco
0 s
[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],
]
# Mide el tiempo de inicio de la ejecución.
start_time = time.time()
# Llama a la función para resolver el Sudoku.
if solve_sudoku(sudoku_board):
# Si se resuelve con éxito, imprime el tablero resultante.
print("Sudoku resuelto:")
print_board(sudoku_board)
else:
# Si no hay solución, lo indica.
print("No se encontró solución.")
# Mide el tiempo final de ejecución.
end_time = time.time()
# Calcula y muestra el tiempo transcurrido.
print(f"Tiempo de ejecución: {end_time - start_time:.4f} segundos")
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: 0.0504 segundos