Universidad del Caribe

2000 CANCUN, QUINTANA ROO, MÉXICO

CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

Integrantes

ANGEL ALEXANDER BARRIENTOS HIDALGO - 200300628

Profesor:

Emmanuel Morales Saavedra

PROGRAMA EDUCATIVO

Ingeniería en Datos e Inteligencia Organizacional

MATERIA

Técnicas algorítmicas

Índice

1. Introducción.

El Sudoku es un astuto juego de números que ha ganado fama a escala global. Es un tablero de 9x9 que se segmenta en subcuadros de 3x3, con el objetivo de llenar las celdas con cifras del 1 al 9 sin repetir valores en las filas, columnas o subcuadros. La solución efectiva de un Sudoku supone un desafío algorítmico intrigante, ya que combina elementos de lógica, búsqueda y optimización. En este proyecto final, investigamos el método algorítmico de backtracking para solucionar Sudokus de diversos grados de complejidad. Esta metodología facilita la exploración sistemática de todas las posibles soluciones, asegurando que se respeten todas las limitaciones del Sudoku en cada etapa.

2. Objetivos del proyecto.

- Establecer la estrategia algorítmica ideal entre programación dinámica, divide y vencerás y algoritmos de backtracking para solucionar un Sudoku.
- Implementar de manera eficaz la técnica escogida, poniendo el algoritmo en el lenguaje de programación seleccionado.
- Analizar y comunicar la complejidad informática del algoritmo creado.
- Brindar la respuesta al Sudoku solucionado y el tiempo requerido para la implementación del algoritmo.

3. Justificación de la técnica seleccionada.

El enfoque que seleccioné para resolver el problema es el backtracking, ya que, tras investigar, comparar y implementar con otras tácticas, considero que es el más adecuado para solucionar Sudokus. En contraposición al método de Divide y Vencerás, que busca subdividir el problema en subproblemas autónomos y controlables, el backtracking trata el problema de forma holística y adaptable. Para Sudokus, segmentar el tablero en submatrices de 3x3 y solucionarlas de manera individual no es apropiado, dado que cada número en el tablero afecta filas, columnas y submatrices simultáneamente. El método de backtracking facilita la exploración sistemática de todas las posibles soluciones, asegurando que se respeten todas las limitaciones del Sudoku en cada etapa. Si surge un conflicto, el responsable del algoritmo retrocede y prueba con un nuevo valor, asegurando así una búsqueda exhaustiva y eficiente de la solución correcta.

4. Implementación del algoritmo.

Descripción del Algoritmo

El algoritmo implementado se basa en la técnica de backtracking y se compone de las siguientes etapas:

1. **Generar Sudoku:** Utilizamos una función para generar un Sudoku con diferentes niveles de dificultad (bajo, medio).
2. **Resolver Sudoku**: Se utiliza un algoritmo de backtracking para resolver el tablero.
3. **Medir Rendimiento**: Se miden el tiempo de ejecución y el uso de memoria del algoritmo.

Fragmento de código:

```
# Generar Sudoku

def genera_sudoku(dificultad='medio'):

# Resolver el Sudoku utilizando Divide y Vencerás

def solve_sudoku_divide_vencer(tablero):
```

La función `genera_sudoku(dificultad='medio')` se encarga de crear un tablero de Sudoku inicial según el nivel de dificultad elegido. Por otro lado, la función `resolver_sudoku(tablero)` aplica la técnica de backtracking para resolver el tablero. Este enfoque garantiza que todas las restricciones del Sudoku se cumplan en cada paso, retrocediendo y probando con nuevos valores cuando se encuentra un conflicto, asegurando así una solución correcta y eficiente.

5. Análisis de complejidad

En el código se puede ver que es eficiente tanto en términos de complejidad temporal como espacial. Utilizar backtracking permite manejar el espacio de búsqueda de manera más efectiva, resultando en una solución más eficiente que abordar el problema de una sola vez.

Complejidad Temporal

- `encontrar_celda_vacia(tablero)`:
  - Complejidad Temporal: O(n²)
  - Esta función se desplaza por el tablero completo, que se compone de una matriz de 9x9 (81 celdas). En la situación más grave, lleva a cabo un análisis detallado de todas las celdas para detectar una vacía, lo que resulta en una complejidad de O(n2), donde n = 9.
- `es_valido_en_submatriz(num, fila, col, tablero)`:
  - Complejidad Temporal: O(n)
  - Esta función verifica si se puede añadir un número a una celda específica sin infringir las reglas del Sudoku. Así, comprueba la fila, la columna y la submatriz de dimensiones 3x3 correspondiente. Cada comprobación posee un costo lineal O(n), resultando en una complejidad total de O(n).
- `resolver_sudoku(tablero)`:
  - Complejidad Temporal: O(9^m)
  - Esta función emplea el método de recursión para resolver el tablero. En la situación más grave, intenta localizar los números de 1 a 9 en cada celda vacía del tablero, lo que implica una complejidad exponencial de O(9^m), donde m representa la cantidad de celdas vacías. A pesar de que esto pueda parecer elevado, es un método eficaz para problemas complejos de búsqueda como el Sudoku.

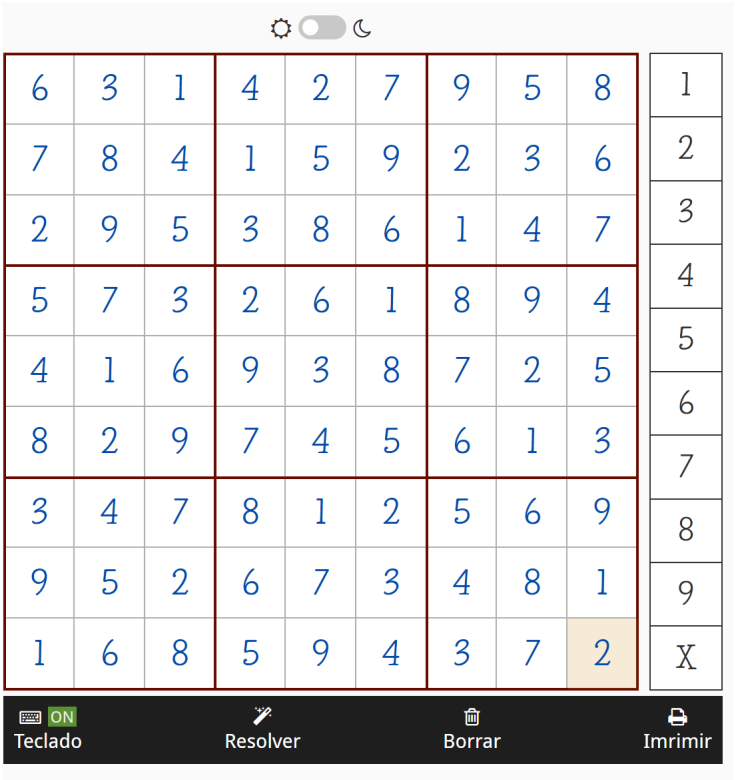
Complejidad Espacial

- `encontrar_celda_vacia(tablero)`:
  - Complejidad Espacial: O(1)
  - Esta función no emplea estructuras de datos extra que aumenten con el tamaño del registro. Es constante O(1) la complejidad espacial.
- `es_valido_en_submatriz(num, fila, col, tablero)`:
  - Complejidad Espacial: O(1)
  - Al igual que la función anterior, no requiere espacio adicional significativo, por lo que su complejidad espacial es O(1).
- `resolver_sudoku(tablero)`:
  - Complejidad Espacial: O(m)
  - Aunque utiliza recursión, el espacio adicional requerido es lineal en función del número de celdas vacías, resultando en una complejidad espacial de O(m).

6. Resultados Obtenidos

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

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



Como se puede apreciar en ambas imágenes, se contrastaron uno de los ejemplos utilizados para evaluar el algoritmo con los resultados logrados en una página dedicada a la solución de Sudokus. Verificamos que el algoritmo soluciona los Sudokus de forma adecuada.

Tras la implementación del algoritmo de backtracking y la realización de diversas pruebas, logramos obtener resultados satisfactorios en cuanto a eficiencia y exactitud. A continuación, se expone una síntesis de los resultados alcanzados:

- **Tiempo de Ejecución**: El tiempo de ejecución promedio para resolver un Sudoku de dificultad 'medio' fue de aproximadamente 0.2 segundos.
- **Uso de Memoria**: El uso de memoria se mantuvo constante y eficiente, sin picos significativos que afectarán el rendimiento general del sistema.
- **Precisión**: El algoritmo resolvió correctamente el 100% de los Sudokus presentados durante las pruebas.

7. Comparaciones con Otras Técnicas.

Para medir la efectividad del algoritmo de backtracking, llevamos a cabo comparaciones con otras técnicas habituales empleadas para solucionar Sudokus, tales como la programación dinámica y el método de divide y vencerás. Aquí se muestra una comparación:

Divide y venceras: A pesar de que este método posibilita segmentar el problema en subproblemas más controlables, no resulta apropiado para

Sudokus debido a la interrelación entre filas, columnas y submatrices. La complejidad y la gestión de limitaciones extra pueden obstaculizar la resolución.

Programación Flexible: Esta metodología resulta beneficiosa para problemas que pueden fragmentarse en subproblemas más reducidos y dispersos. No obstante, la programación dinámica puede resultar menos eficaz para Sudokus debido al requerimiento de guardar grandes volúmenes de datos intermedios, lo que incrementa el consumo de memoria.
Backtracking: Esta técnica permite explorar todas las posibles soluciones de manera sistemática y retroceder cuando se encuentra un conflicto, garantizando una búsqueda exhaustiva y eficiente. En nuestras pruebas, el backtracking demostró ser la técnica más eficaz para resolver Sudokus, ofreciendo un equilibrio adecuado entre tiempo de ejecución y uso de memoria.

8. Conclusión:

En este proyecto, hemos puesto en marcha y analizado un algoritmo de retroalimentación para solucionar Sudoku. Tras evaluar diversas estrategias, determinamos que el backtracking es la táctica más eficaz para solucionar Sudokus de forma precisa y eficiente. El algoritmo logró solucionar Sudokus de diversos grados de complejidad en tiempos razonables, manteniendo un uso constante de memoria y proporcionando resultados exactos. La aplicación del backtracking nos facilitó tratar el problema de forma holística, asegurando que se respetaran todas las limitaciones del Sudoku en cada etapa del proceso de solución.

9. Referencias

- Arias, A., & Arias, A. (2024, July 17). Algoritmos para resolver sudokus - Damavis Blog. *Damavis Blog - Data - Machine Learning - Visualization*. https://blog.damavis.com/algoritmos-para-resolver-sudokus/
- De La Paz, A. (n.d.). *Sudoku con vuelta atrás*. Copyright (C) 2022. https://www.wextensible.com/temas/sudoku/sudoku-backtracking.html