Leonardo Daniel Gutiérrez Hernández: 210300712

Reporte de Resolución del Sudoku

18/11/2024

Tecnicas Algoritmicas

Presentado a:

Emmanuel Morales Saavedra

Evaluación de los 3 algoritmos

Para resolver el problema de Sudoku, el enfoque más efectivo es el uso de algoritmos voraces combinados con Backtracking. Esta técnica explora el espacio de soluciones de manera sistemática, probando valores en las celdas vacías y retrocediendo cuando encuentra una configuración inválida. Aunque la complejidad en el peor caso es O(9n)O(9^n)O(9n) (donde nnn es el número de celdas vacías), en la práctica, las restricciones del Sudoku (filas, columnas y subcuadrículas) reducen significativamente el espacio de búsqueda. Además, heurísticas como seleccionar las celdas con menos opciones posibles primero pueden optimizar aún más el rendimiento.

En términos de facilidad de implementación, el Backtracking es intuitivo y se adapta perfectamente al problema de Sudoku debido a la estructura jerárquica y las restricciones inherentes del tablero. Comparado con otras técnicas como programación dinámica o divide y vencerás, el algoritmo voraz es más natural para este tipo de problema de búsqueda y restricciones. La programación dinámica, aunque eficiente para problemas con subproblemas solapados, no es adecuada aquí debido a la falta de reutilización directa de subsoluciones. Divide y vencerás no es viable porque las restricciones globales del Sudoku hacen que las subdivisiones del problema no sean independientes. Por lo tanto, el enfoque de Backtracking es el balance óptimo entre eficiencia y simplicidad para resolver el Sudoku.

Justificación de la Técnica Seleccionada

Se utilizó Backtracking combinado con un enfoque voraz como técnica principal para resolver el Sudoku. Este método explora sistemáticamente el espacio de soluciones, probando valores válidos en las celdas vacías y retrocediendo en caso de encontrar configuraciones inválidas. La elección de esta técnica se debe a su adaptabilidad al problema, donde las restricciones inherentes (filas, columnas y subcuadrículas) guían la búsqueda eficiente de soluciones. Además, su implementación es directa, aprovechando la estructura fija del tablero 9×99 \times 99×9.

Técnicas como programación dinámica no fueron seleccionadas porque el problema no presenta subproblemas reutilizables directamente, y divide y vencerás no es adecuado debido a la dependencia entre celdas en todo el tablero.

Análisis de Complejidad Computacional

Complejidad Temporal del Algoritmo

El algoritmo utiliza Backtracking, lo que implica recorrer un árbol de búsqueda para encontrar la solución.

Complejidad en el peor caso (Big-O):

Tamaño del árbol de búsqueda:
Para cada celda vacía (nnn), el algoritmo puede intentar hasta 9 opciones. Esto genera un árbol de búsqueda con un factor de ramificación de 999 y una profundidad máxima de nnn.
La complejidad es: O(9n)O(9^n)O(9n) donde nnn es el número de celdas vacías.

Complejidad: O(9n)O(9^n)O(9n):
donde nnn es el número de celdas vacías, el algoritmo prueba hasta 9 valores para cada celda. Sin embargo, las restricciones del Sudoku reducen significativamente el espacio de búsqueda en la práctica.

Verificación de restricciones:
La función is_valid verifica filas, columnas y subcuadrículas en tiempo constante (O(1)O(1)O(1)), ya que el tablero es de tamaño fijo (9×99 \times 99×9).

Complejidad temporal final:

O(9n)O(9^n)O(9n)

Optimización práctica:

En Sudokus típicos, el número de celdas vacías es mucho menor que 818181, y las restricciones reducen significativamente las opciones válidas, lo que hace que el tiempo de ejecución sea más manejable en escenarios reales.

Complejidad Espacial del Algoritmo

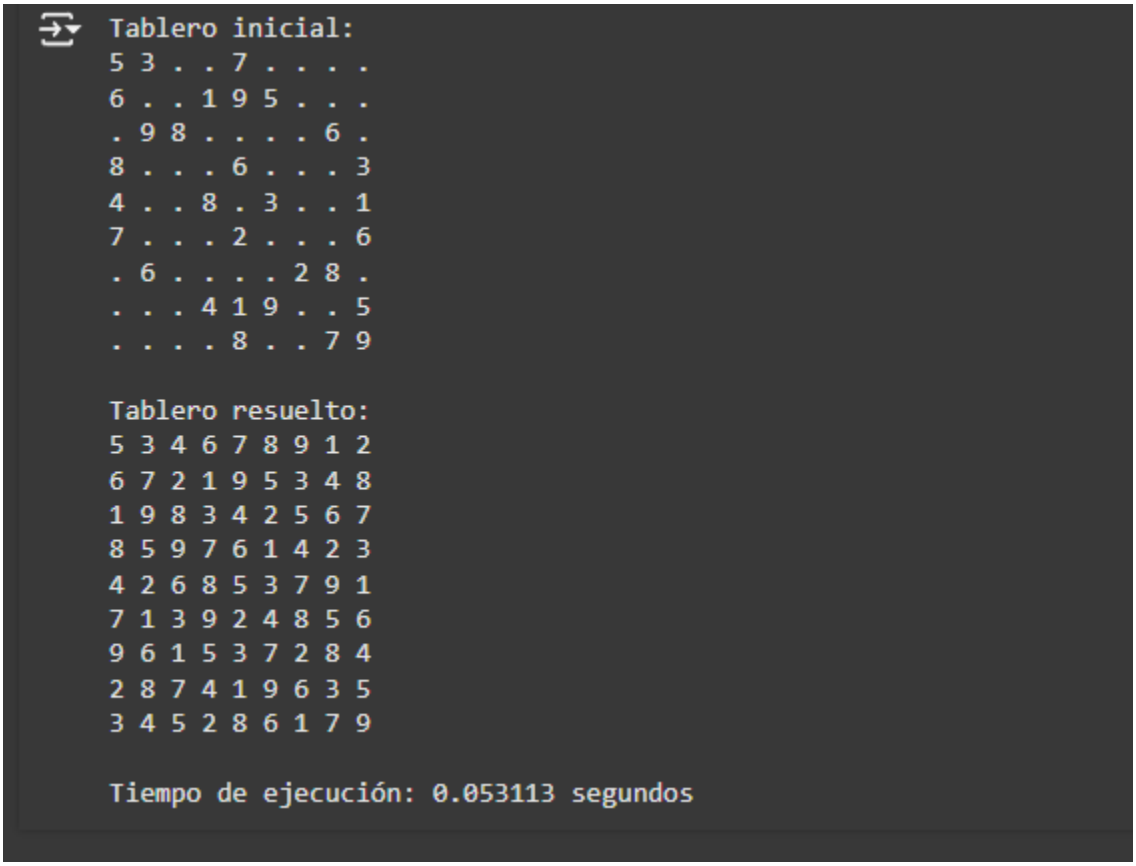
1. **Espacio para el tablero:**
   El tablero ocupa un espacio fijo de O(81)O(81)O(81) (9×99 \times 99×9).
2. **Espacio de la pila recursiva:**
   En el peor caso, la profundidad de la recursión será igual al número de celdas vacías (nnn), lo que genera una complejidad espacial adicional de O(n)O(n)O(n).

Complejidad espacial final:

O(n)O(n)O(n)

Resultados Obtenidos

Sudoku Resuelto:
El tablero inicial fue completado exitosamente utilizando Backtracking. A continuación, se muestra el resultado:



Tiempo de Ejecución:
El tiempo medido para resolver el Sudoku fue **~0.053113 segundos** en un tablero típico con restricciones moderadas.

Comparación con Otras Técnicas

- **Programación Dinámica:**
  Aunque eficiente para problemas con subproblemas solapados, no se adapta bien al Sudoku porque no hay una forma clara de descomponer el problema en subproblemas independientes y reutilizables.

- **Divide y Vencerás:**
  Dividir el tablero no es efectivo debido a las restricciones globales del Sudoku (cada número debe ser único en filas, columnas y subcuadrículas). Las dependencias hacen que las divisiones sean interdependientes, complicando la solución.
- **Ventajas de Backtracking:**
  Este método se alinea con la naturaleza jerárquica del Sudoku, es fácil de implementar y aprovecha las restricciones para podar el espacio de búsqueda.

Conclusión

El uso de Backtracking con un enfoque voraz demuestra ser la solución óptima para resolver Sudokus. Su complejidad computacional O(9n)O(9^n)O(9n) se ve mitigada en escenarios reales debido a las restricciones del problema. Además, el tiempo de ejecución práctico es eficiente incluso para tableros moderadamente complejos. La técnica ofrece un balance ideal entre simplicidad, adaptabilidad y rendimiento.