En la resolución del Sudoku utilizando **programación dinámica**, se aprovechó la capacidad de esta técnica para almacenar y reutilizar soluciones parciales, lo que reduce cálculos redundantes y optimiza la búsqueda de soluciones. Al emplear una estructura tridimensional (`dp`), se representaron todas las posibles opciones para cada celda, y estas se actualizaron dinámicamente a medida que se resolvía el tablero. Este enfoque resulta especialmente útil porque el Sudoku es un problema basado en restricciones, y la programación dinámica permite manejar estas restricciones de manera eficiente.

En términos de **complejidad computacional**, aunque la solución todavía enfrenta el crecimiento exponencial inherente al problema en su peor caso, la reducción del espacio de búsqueda al elegir las celdas con menos opciones válidas mejora considerablemente el rendimiento en promedio. La técnica también presenta ventajas en términos de **facilidad de implementación**: al usar estructuras como `dp`, se puede adaptar fácilmente el algoritmo a variantes del Sudoku o incluso a otros problemas similares. Este enfoque combina claridad, eficiencia y modularidad, siendo una elección sólida para resolver el problema en un marco analítico y escalable.

Notacion Big-O:

El algoritmo utiliza una combinación de:

1. **Búsqueda de celdas vacías**: Recorre el tablero de tamaño N×NN \times N×N (en el caso del Sudoku, N=9), lo que tiene un costo de O(N2).
2. **Evaluación de posibilidades por celda**: Para cada celda vacía, se prueban hasta 9 números posibles. Cada validación para filas, columnas y subcuadros tiene un costo de O(N).
3. **Propagación de restricciones y retroceso**: En el peor caso, se exploran todas las combinaciones posibles de celdas, lo que lleva a O(9N2), ya que en cada celda puede haber hasta 9 opciones posibles.

Sin embargo, **en promedio**, gracias a las restricciones dinámicas y la selección de celdas con menos opciones, el espacio de búsqueda se reduce significativamente, haciendo que la complejidad efectiva sea mucho menor que el peor caso.

- **Complejidad temporal aproximada**: O(9N), ya que no todas las celdas necesitan todas las posibilidades.

Complejidad Espacial:

- La estructura `dp` requiere O(N3) espacio, ya que almacena una lista de 9 posibles valores para cada celda en el tablero N×N .

Resultados:

El uso de programación dinámica resultó efectivo para resolver el Sudoku, con las siguientes ventajas:

- **Eficiencia**: La reducción del espacio de búsqueda mejora el rendimiento frente a métodos ingenuos de fuerza bruta.
- **Escalabilidad**: La implementación puede adaptarse a tableros más grandes o variaciones del Sudoku con cambios mínimos.
- **Tiempo de ejecución**: El tiempo medido demuestra que el algoritmo es rápido para casos típicos, logrando una solución casi instantánea.