## I - Sudoku Solver

**Spoiler:** Casi me rindo con este ejercicio :(

**1. Reducción del Problema:**

* **Descripción en palabras:**

El nombre del problema lo dice resolver un sudoku, nos hay alguna configuración de un sudoku y hay que resolverlo o más formalmente:

Dado un tablero de Sudoku parcialmente lleno (9×9), se debe completarlo de forma que en cada fila, columna y subcuadro de 3×3 aparezcan todos los números del 1 al 9 sin repetir.

* **Descripción en lenguaje matemático:**

Dado un arreglo A que ∈ {1,....9}9×9 U {.}, encontrar una función

f: {1,....9}x {1,....9} => {1,....9} tal que para todo i,j A[i][j] = f(i,j), si A[i][j] es diferente de “.” y se cumpla que para todo i {f(i,j): j:1,...9} = {1,..9}, para todo j, {f(i,j): i= 1,...9 = {1..9}, y en cada subcuadro 3x3 los valores también sean {1,..9} sin repeticiones

**2. Reducción de la Solución**

* **Descripción en palabras:**

Usamos backtracking para probar recursivamente todos los números posibles (1 a 9) en cada celda vacía. Si al colocar un número se mantiene la validez del tablero (es decir, no se repite en su fila, columna ni subcuadro), se continúa. Si no hay opciones válidas, se retrocede (backtrack) y se prueba otro número hasta encontrar la solución completa

.

* D**escripción en lenguaje matemático:**

Sea S: {1,....9} x {1,....9} => {1,....9} una función parcial que extiende A (tablero original). Usamos alguna función recursiva digamos Solve(S) que, para la siguiente celda vacía (i,j), intenta asignar un valor v ∈ {1..9} tal que v no pertenece a la filai v que no pertenece a la columna j, y v no pertenezca al subcuadro i,j. si se encuentra tal v, se define S(i,j) := v se llama recursivamente a solver(S). si no existe tal v, se deshace la asignación (backtrack). el algoritmo termina cuando todas las celdas están llenas respetando las restricciones.

**3. Código Realizado y Análisis**

* Enlace(s) Código: <https://vjudge.net/solution/61083806/Y32NQkmABAjdVAheD4Ah>

**4. Casos de Prueba**

**Caso 1**: simple casi completo

**Entrada:**

1

534678912

672195348

198342567

859761423

426853791

713924856

961537284

287419635

34528617.

**Salida**

Case 1:

534678912

672195348

198342567

859761423

426853791

713924856

961537284

287419635

345286179

**Justificación:**

Solo falta un número en la última fila. Este caso permite validar si el algoritmo puede identificar rápidamente la única posición faltante y completarla sin explorar demasiado.  
  
**caso 2**: con algunas celda vacías

1

53..7....

6..195...

.98....6.

8...6...3

4..8.3..1

7...2...6

.6....28.

...419..5

....8..79

**Salida**

Case 1:

534678912

672195348

198342567

859761423

426853791

713924856

961537284

287419635

345286179

**Justificación:**

Este es un ejemplo clásico de dificultad media, tomado de muchos tutoriales. Verifica que el algoritmo maneja múltiples espacios vacíos sin entrar en combinaciones explosivas, siguiendo una lógica ordenada.

**Caso 3**: Complejo mínima cantidad de pista

**Entrada**

1

.......12

4........

..3......

...7.....

.....5.9.

..2......

6.....1..

.....8...

..1......

**Salida**

Case 1:

576983412

419526378

823417956

358769241

164235897

792841635

647352189

235198764

981674523

**Justificación:**

Este caso contiene muy pocas pistas iniciales, por lo que requiere una gran cantidad de exploración y retrocesos. Sirve para comprobar la eficiencia del backtracking y que el algoritmo no falla con la complejidad combinatoria.

**5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)**

La verdad para este ejercicio hice como más de 100 intentos fallidos(En el vjudge no se refleja eso, eso lo explicare en el proceso de depuración), entonces solo pondré los intentos más cercanos antes de la solución correcta

**Casos incorrectos:**

<https://vjudge.net/solution/61033393/7Fib0fZNV5MAZub2JVsp>

<https://vjudge.net/solution/61033437/cY53HDtSUWOgYx7uvnQA>

<https://vjudge.net/solution/61033462/gPiNUeyZri9U41xNv1kl>

* **Descripción del error**: Pues, el error principal es el **time limit exceeded**, desde el primer intento, osea funcionaba pero se tardaba muchísimo.
* **Proceso de depuración**: Probe muchas cosas incluso tuve que utilizar IA(no para resolver el problema solo que me diga qué temas debería saber para resolverlo, aunque al final sí le pedí que lo resolviera pero igual daba el mismo problema, en la parte LLM lo pondré) para que me ayudara pero ni así porque seguía el mismo problema, lo intente en c++ aunque lo hacia mas rapido tampoco era suficiente, hubo un momento donde la página de vjudge se puso lenta asi que busque cómo medir el tiempo de vscode ahi probe una y otra y otra vez por varios días creo que fueron 3 o 4 dias (no es que tomo ese tiempo si no son los días que los intente) hasta que por fin lo logre aqui explico cual era el problema:

En los primero el backtracking prueba todas las celdas en orden fijo y cada vez recorre fila, columna y bloque (O(27) por intento) sin poda temprana.

Entonces ahora lo que intente fue: usar matrices booleanas para chequeos O(1), así elige siempre la celda con menos opciones, que actualiza incrementalmente sus candidatos y aborta en cuanto detecta un vacío, reduciendo drásticamente ramas exploradas y verificaciones por nodo.

* **Solución corregida:** <https://vjudge.net/solution/61043974/dstrSZ0jmqE5EL6w38Kg>

**6. Preguntas de Aprendizaje**

* **Temas aplicados**:
* Uso del time: (solo para ver el tiempo ejecucion)
* Recursividad
* Backtracking: Exploración recursiva de soluciones, pero con detención inmediata al detectar una inconsistencia
* Representación por restricciones: como las matrices booleanas
* Heurística MRV (Minimum Remaining Values)
* Ordenación adaptativa
* **Dificultad de la implementación**: Pues fue el ejercicio que más tiempo me tomó aunque tiempo no es igual que dificultad en este caso sí lo es.(para mi, quizas los otros compañeros el ejercicio más difícil fue otro), y lo mas costo fue implementar el Backtracking
* Recursos utilizados:

<https://www.wscubetech.com/resources/dsa/backtracking-algorithm>

<https://docs.python.org/3/library/time.html>

<https://www.sciencedirect.com/topics/computer-science/boolean-matrix>

**7. Feedback LLM**

* Envío código LLM: <https://vjudge.net/solution/61083859/C6fFXrF3Xxz4dzVNDSaF>
* **Comparación de su códigos**:

El de LLM recorre el tablero en orden fijo y, para cada celda vacía, llama a es\_valido que a su vez recorre fila, columna y bloque 3×3 en bucles de longitud 9 (unos 27 pasos por intento), sin anticipar si esa asignación hará inviable otra celda; en cambio, el mio mantiene tres matrices booleanas para filas, columnas y cajas que permiten comprobar disponibilidad en O(1), construye objetos Celda con su lista de candidatos y aplica la heurística MRV (elige siempre la celda con menos opciones) combinada con forward‑checking: tras asignar un dígito recalcula sólo los dominios afectados y aborta de inmediato si detecta un dominio vacío. Así, en lugar de explorar casi 9^m ramas con chequeos lineales, poda muy pronto y reduce drásticamente tanto las ramas exploradas como el coste de cada prueba.

* **Feedback de LLM**: Pida al LLM que evalúe y corrija su código:

