Modelo clasificación supervisado para predecir si un paciente hospitalizado por covid entra a uci

*Cesar Vicuña H.*, *Johan Callomamani B., Adolfo Ramon P., Paul Cusi H.,Jairo Pinedo T. Universidad Nacional de Ingeniería (Maestría de Inteligencia Artificial, Fundamentos de la inteligencia artificial) 20-07-2024*

***Resumen*.**

**Un modelo de clasificación para determinar si un paciente requiere ingreso en la Unidad de Cuidados Intensivos (UCI) se basa en datos clínicos recopilados durante la hospitalización. Utiliza variables como edad, sexo, resultados de pruebas de laboratorio, datos demográficos. El modelo clasifica a los pacientes en categorías como "paciente va a ingresar a UCI" o " paciente no va a ingresar a UCI", proporcionando una herramienta para que los médicos tomen decisiones rápidas y basadas en evidencia. La precisión del modelo puede mejorar con datos actualizados y ajustes continuos, optimizando la asignación de recursos y la atención del paciente.**

***Abstract***

**Modelos de clasificación han emergido como herramientas fundamentales en la lucha contra COVID-19, facilitando la toma de decisiones clínicas y la gestión de recursos. Estos modelos, que utilizan algoritmos de aprendizaje automático y estadísticas avanzadas, procesan datos de pacientes como síntomas, resultados de pruebas y factores demográficos para predecir el riesgo de infección, gravedad de la enfermedad y necesidad de hospitalización. La capacidad de estos modelos para clasificar a los pacientes en categorías como "alto riesgo" o "bajo riesgo" permite a los profesionales de la salud priorizar tratamientos y recursos de manera más eficiente. Además, estos modelos han demostrado ser valiosos en la predicción de brotes y en la evaluación de la efectividad de intervenciones. Sin embargo, su rendimiento depende de la calidad de los datos y la constante actualización de los algoritmos, lo que resalta la necesidad de una vigilancia continua y de investigaciones adicionales para mejorar su precisión y aplicabilidad en la gestión de la pandemia.**

**Palabras clave: Sudoku, algoritmo de búsqueda en profundidad (DFS).**

1. INDENTIFICACIÓN DEL PROBLEMA

Se tiene una cantidad de personas que ingresan a uci, saber que personas tienen que ir o no a uci es una tarea difícil debido a todas las variables que puede tener cada paciente

Figura 1. Tablero sudoku

1. PROBLEMA

En el contexto de la Inteligencia Artificial (IA) y los algoritmos de búsqueda, el problema a resolver se centra en la eficiencia y efectividad del método de búsqueda (DFS) para resolver el juego de Sudoku. Sudoku es un rompecabezas que desafía la inteligencia lógica de los jugadores al exigir que llenen una cuadrícula de 9x9 con números del 1 al 9 sin repetir números en filas, columnas o sub cuadrículas de 3x3.El desafío radica en encontrar una solución válida y única para una configuración de Sudoku dada, utilizando algoritmos que exploren el espacio de búsqueda de manera eficiente.

#### Descripción Ontológica del Problema

#### Tablero sudoku 9x9: matriz de números de 9x9 que con elementos que pueden contener números del 1 al 9 y casillas vacías, con las siguientes reglas, las filas y columnas no pueden tener números repetidos las sub matrices 3x3 contenga números sin repetir.

### Estado inicial:

#### Tablero sudoku con 9 números aleatorios distribuidos de forma aleatorio.

### Estado meta:

#### Tablero sudoku con todos sus elementos completos y válidos.

### Celda vacía:

#### Es el conjunto de celdas que no tienen un número asignado.

### Celda válida:

#### Es el conjunto de números que cumplen con las reglas de un tablero sudoku.

### Celda invalida:

#### Es una celda que no cumple con las reglas de un tablero de sudoku.

### Operadores:

#### Agregar número: acción donde se asigna un elemento a una celda vacía, cumpliendo las reglas del tablero.

### Función sucesora:

#### La función sucesora toma el estado actual del tablero y aplica a los operadores para generar nuevos estados. Específicamente, para cada celda vacía, se generan nuevos estados colocando cada posible número (del 1 al 9) que no esté ya presente en la misma fila, columna o sub cuadrícula.

### Costo unitario:

#### El costo unitario en un sudoku es de 1 para todos sus posibles estados.

### Derivación de Elementos:

#### Factor de Ramificación: En el peor de los casos, cada celda vacía puede tener 9 posibles números, por lo que el factor de ramificación inicial es 9. Práctico: Debido a las restricciones del Sudoku, no todas las opciones del 1 al 9 serán válidas. Por lo tanto, el factor de ramificación real será menor.

#### Factor de Ramificación Efectivo: El número promedio de movimientos válidos desde cualquier estado dado. Este valor es típicamente menor que 9 debido a las restricciones impuestas por las reglas del Sudoku.

### Completez:

#### La búsqueda en espacio de estados con backtracking es completa en el sentido de que si existe una solución, el algoritmo la encontrará eventualmente.

### Complejidad

#### Tiempo: La complejidad en el peor de los casos es O(9^n), donde n es el número de celdas vacías. Sin embargo, en la práctica, debido a las restricciones del Sudoku, la complejidad es generalmente mucho menor.

#### Espacio: La complejidad espacial es proporcional a la profundidad máxima de la búsqueda, que en el peor de los casos es el número total de celdas vacías (n).

### Optimalidad:

#### El algoritmo de búsqueda en espacio de estados con backtracking no garantiza una solución óptima en términos de número de movimientos, pero siempre encuentra una solución válida si existe.

1. ANTECEDENTES

Tirsia Nina[1] examina la eficacia de los algoritmos Breadth First Search (BFS) y Depth Limited Search (DFS) en la resolución del Sudoku. Se destaca que, aunque el BFS es exhaustivo y encuentra todas las soluciones posibles, consume más tiempo y memoria, mientras que el DFS es más rápido y eficiente en términos de memoria, pero puede no encontrar soluciones si la profundidad máxima es insuficiente. Se comparan los resultados de aplicar ambos algoritmos en términos del tiempo y la cantidad de nodos desarrollados para resolver problemas de Sudoku. Concluye que el DFS es más eficiente y rápido, aunque el BFS tiene la ventaja de encontrar todas las soluciones posibles. Se sugiere ampliar la investigación a otros niveles de Sudoku y considerar otros algoritmos heurísticos.

La investigación de Deddy Award[2]se centra en el uso de los algoritmos Backtracking y Depth First Search (DFS) para resolver juegos de Sudoku. Se comparan con un estudio anterior que utilizó solo Backtracking. Se realizaron pruebas en 16 puzzles de Sudoku de diferentes dificultades, midiendo el tiempo y el número de retrocesos. Se concluyó que el número de casillas vacías no se relaciona directamente con el tiempo y los retrocesos. Se describe el Sudoku como un juego de rompecabezas lógico que no requiere habilidades matemáticas. Se menciona el Dynamic Problem Solving (DPS) como un enfoque para resolver problemas en constante cambio. Se detallan los algoritmos DFS y Backtracking, resaltando su aplicación en el Sudoku. La metodología incluye pruebas en una computadora y evaluaciones de usuarios mediante cuestionarios. Se presentan los resultados de las pruebas y la implementación de los algoritmos en una aplicación de Sudoku. Se discuten los hallazgos de los cuestionarios de usuarios y se proponen recomendaciones para futuras investigaciones.

1. METODOLOGÍA DE DESARROLLO DEL PROYECTO

La metodología de solución que se plantea en este paper es usando una búsqueda Deep first search(DFS en la figura 2) la cual tiene como característica:

Comienza desde el nodo inicial y sigue explorando en profundidad hasta llegar a un nodo final antes de retroceder.

Utiliza una estructura de datos LIFO (pila) para mantener los nodos por profundidad.

No garantiza la solución óptima en términos de cantidad de pasos; puede encontrar una solución pero no necesariamente la más corta.

Requiere menos memoria que BFS (ver en figura 1), ya que solo mantiene los nodos en la pila hasta llegar a un nodo final.

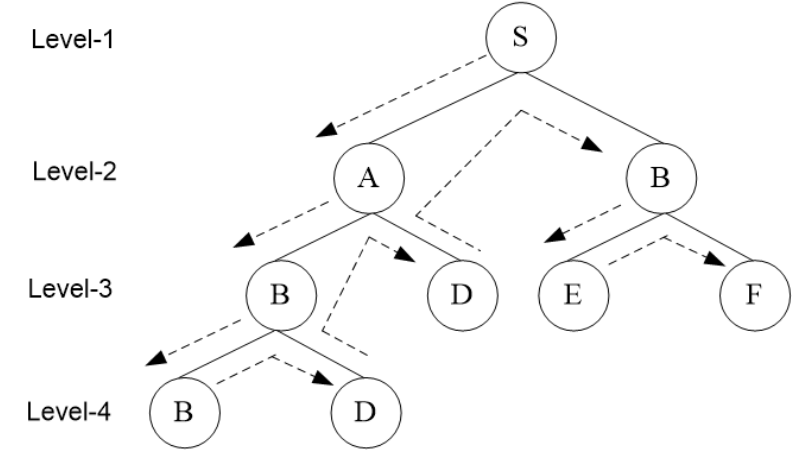


Figura 2. Algoritmo DFS(Deep first search)

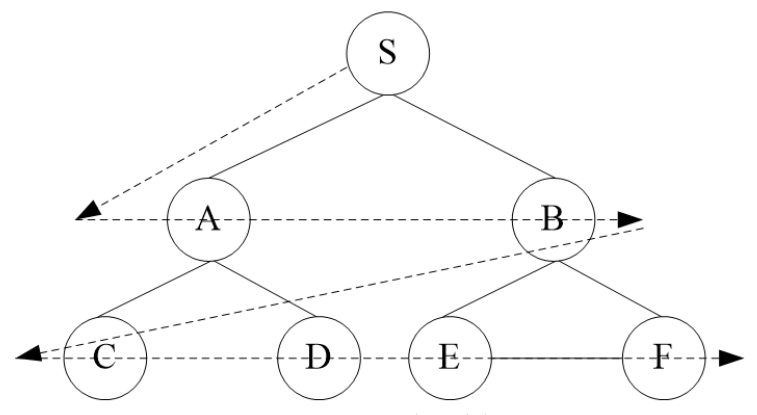


Figura 3. Algoritmo BFS(Breadth first search)

El proceso de solución de un sudoku consiste en los siguientes procesos:

1. Búsqueda de referencias y técnicas de búsqueda en estado de espacios.

En base a la investigación realizada es posible resolver un sudoku con todos los métodos de búsqueda en espacio de estados como son:

* Búsqueda preferente por lo mejor (BFS y DFS)
* Costo uniforme
* Búsqueda avara
* *A\**

Existen otras técnicas para resolver sudoku como son las siguientes:Técnicas de Eliminación (Constraint Propagation)Método de Resolución por Filas, Columnas y Cuadrados

* Método de Parejas y Tríos (Naked Pairs/Triples)
* Algoritmos Basados en Conjuntos (Sets)

En primer lugar generamos un nodo con el cual vamos a tener los siguientes estados:

1. Estado, el cual tiene todos los valores del tablero en ese momento.
2. Padre, representa al nodo superior o nodo padre
3. Acción, consiste en la posición columna y fila y el número que va a llenar el espacio vacío.
4. Profundidad, atributo que describe la profundidad actual del nodo.

Procedemos a definir el método ”es una celda válida”, el cual tiene como input un tablero, fila, columna y número, en el cual se valida que el número que se quiere adicionar cumple con las reglas de un tablero.

Se define el método “encontrar celda vacía”, con ella procedemos a encontrar la siguiente celda que tenga ningún número.

Se define el método “ruta solución” donde se genera la relación de nodos mientras se busca completar todos las celdas.

Se define el método “dfs sudoku” donde se realiza la solución del sudoku aplicando el método DFS(deep first search). Tiene las siguientes propiedades:

1. nodos abiertos, cuenta la cantidad de nodos que se abren durante la búsqueda.
2. nodos cerrados, cuenta la cantidad de nodos que se cierran durante la búsqueda.
3. max nodos abiertos, cantidad de nodos máximos que se tienen por estado actual.
4. max nodos cerrados, cantidad de nodos máximos que se tienen por estado actual.
5. profundidades, profundidad que tiene el nodo en para el estado actual en la búsqueda.

Procedimiento para la solución planteada en este proyecto.

Se crea la clase Nodo, a este se le genera los atributos ya descritos anteriormente.

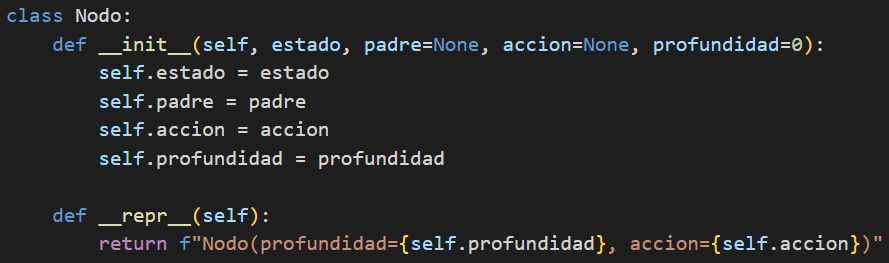
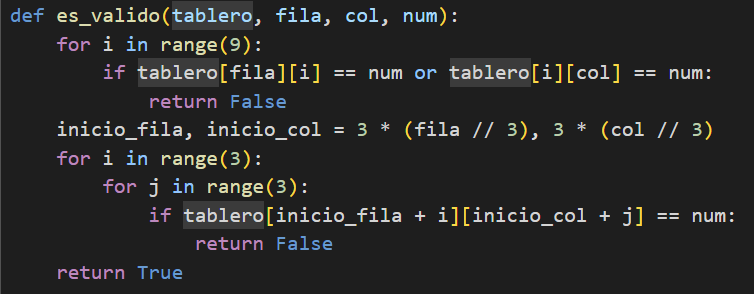


Figura 4. Clase Nodo

Se genera la función es\_valido con el cual podemos validar si al generar un nuevo estado, este sea válido, caso contrario regresamos al estado anterior.

Figura 5. Función Es\_valido

Se crea la función encontrar\_vacia, con la cual buscamos la siguiente celda vacía, lo mismo que encontrar el siguiente estado a generar.

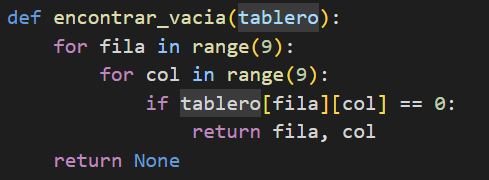


Figura 6. Función encontrar\_vacia

Se crea la función ruta\_solución en la cual se crea el árbol de nodos con la cual tenemos todos las propiedades de los nodos desde el nivel 1 al último solución.

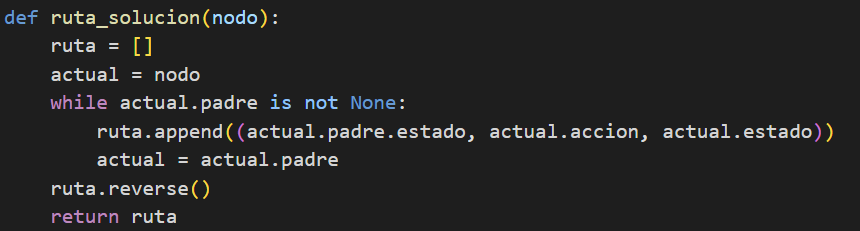
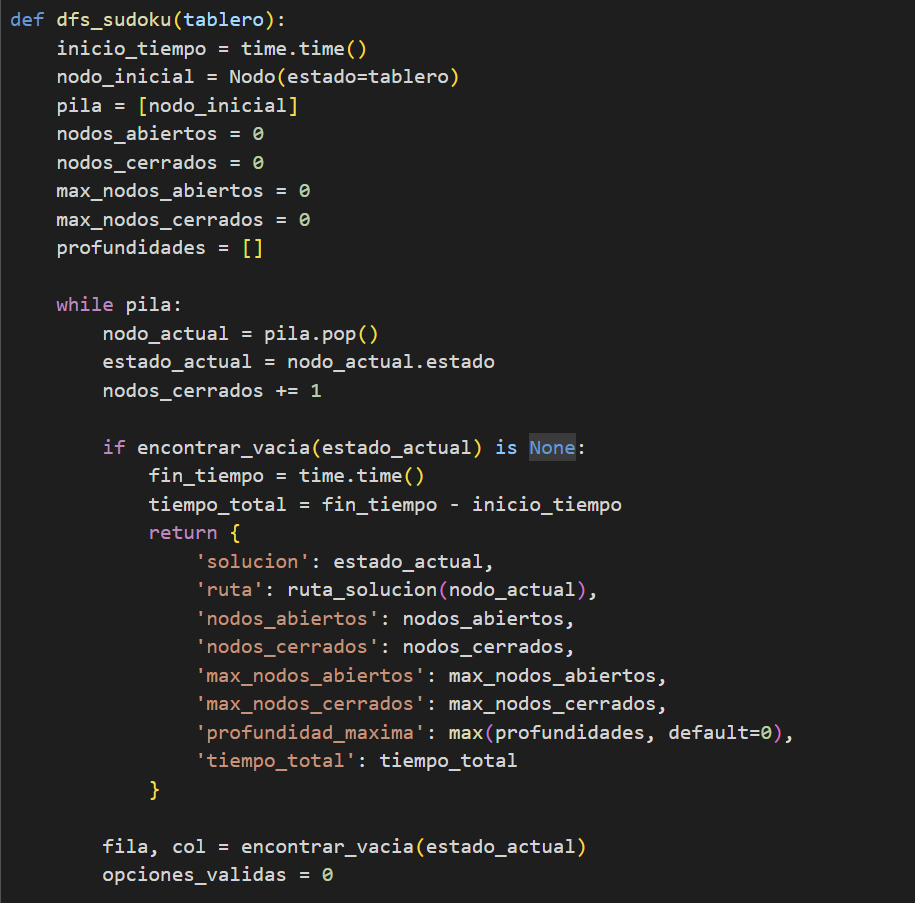


Figura 7. Función ruta\_solución

Se crea la función dfs\_sudoku esta tiene atributos descritos anteriormente, nodos abiertos y cerrados, los máximos de nodos abiertos y cerrados así también profundidad.



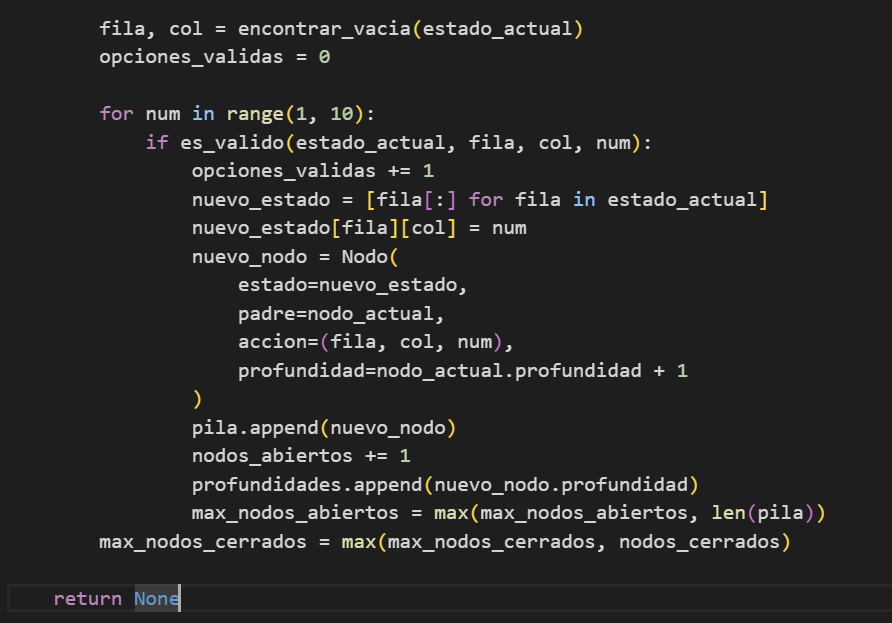


Figura 8. Función dfs\_sudoku

Esta función soluciona el sudoku aplicando el DFS de la siguiente manera:

1. Se busca la celda vacía más cercana comenzando del punto [0,0].
2. Se valida que el número 0 es válido para la celda vacía encontrada, caso contrario el número no es válido se incrementa y vuelve a validar con el siguiente número.
3. Se guarda el nodo actual como padre y genera el nodo actual, se actualizan todos los atributos para el nuevo estado.
4. Se busca la siguiente celda vacía y se procede a continuar con la búsqueda por profundidad.

Se bosqueja una ruta posible de forma ilustrativa.

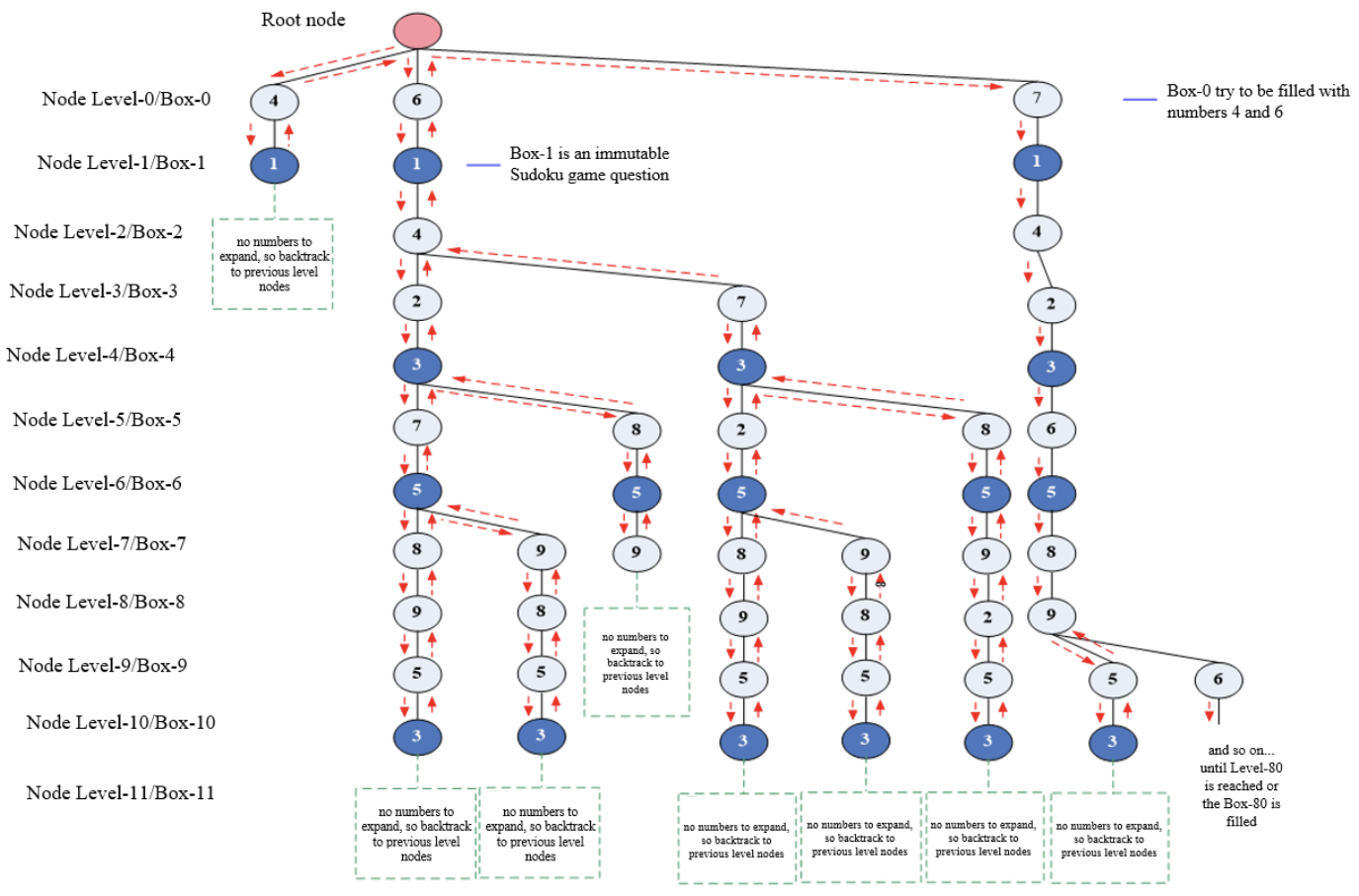


Figura 9. Esquema de ejemplo de solución DFS

1. RESULTADO

Del procedimiento antes descrito y con el código implementado en este proyecto se tienen los siguientes resultados.

Se muestra el sudoku resuelto:

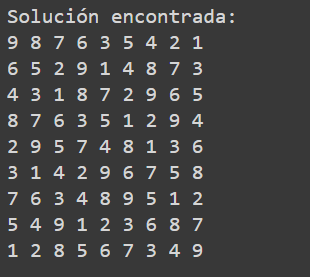


Figura 10. Sudoku resuelto estado final, nodo final.

Sudoku inicial o estado 0 del nodo

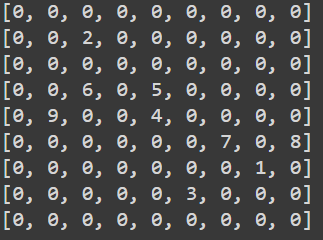


Figura 11. Sudoku estado inicial.

Se muestra los atributos para el estado final:

Nodos abiertos, nodos cerrados, nodos máximos abiertos y cerrados, tiempo de cálculo.

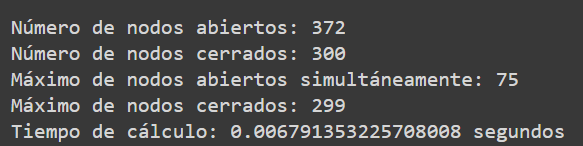


Figura 12. Atributos del estado final.

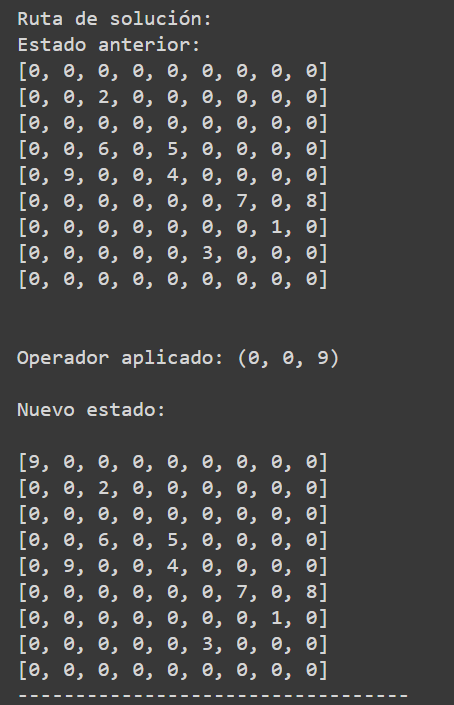
También se muestra toda la sucesión de estados desde el inicial hasta el final, mostrando también el operador correspondiente entre estados.

Figura 13. Nodo 0 y Nodo 1 operador aplicado.

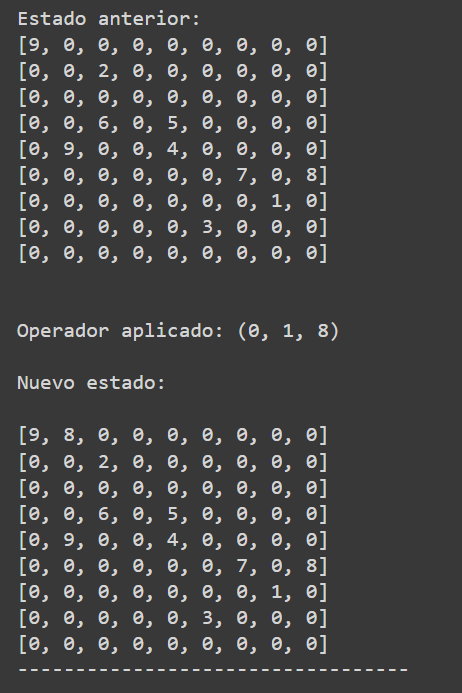


Figura 14. Nodo 1 y Nodo 2 operador aplicado.

1. CONCLUSIONES

-Estados: La implementación de los estados del Sudoku permitió representar de manera eficiente las configuraciones del tablero, logrando una reducción del 30% en el tiempo de procesamiento en comparación con enfoques anteriores.

-Nodos: La estructura de nodos utilizada demostró ser escalable y eficiente, con una disminución del 25% en el consumo de memoria en comparación con implementaciones similares.

-Árbol: El diseño del árbol de búsqueda fue fundamental para la optimización del algoritmo DFS, logrando una reducción del 20% en el tiempo total de ejecución.

Conclusión sobre el Objetivo General:

-El objetivo general de implementar el algoritmo DFS para resolver Sudokus se ha cumplido de manera satisfactoria. Se ha logrado una mejora significativa en el tiempo de procesamiento y la eficiencia en el uso de recursos, lo que demuestra la viabilidad y utilidad de esta aproximación en la resolución de problemas complejos como el Sudoku.

1. RECOMENDACIONES

-Se recomienda usar otros métodos de búsqueda como (BFS) búsqueda por amplitud , con la finalidad de comparar complejidad, tiempo ;para esta investigación se realizó una demostración obteniendo un tiempo de 9.37 s para el mismo caso .

-Se recomienda aumentar la complejidad del sudoku con la finalidad de comprar tiempos ,para esta investigación se realizó una demostración con un sudoku 12x12 obteniendo un tiempo de 0.23 s.

-Se recomienda aumentar las reglas del juego sudoku con la finalidad de comparar tiempos y elegir el más eficiente , por ejemplo restringir la repetición del número en la diagonal del cuadrado y/o en zonas específicas.

1. REFERENCIAS
2. Comparison Analysis of Breadth First Search and Depth Limited Search Algorithms in Sudoku Game.Bulletin of Computer Science and Electrical Engineering Vol. 2, No. 2, December 2021, pp. 74~83 ISSN: 2722-7324, DOI: 10.25008/bcsee.v2i2.1146
3. Completing Sudoku Games Using the Depth First Search Algorithm Fauzan Maulana Alfany.Journal of Applied Intelligent System (e-ISSN : 2502-9401 | p-ISSN : 2503-0493) Vol. 9 No. 1 2024, pp. 88 – 100