Noviembre 2024

Keisi Gómez

PROYECTO SUDOKU

Informe del proyecto sudoku
Técnicas Algorítmicas

UNIVERSIDAD DEL CARIBE

Informe: Resolución del Sudoku utilizando Backtracking

Justificación de la Técnica Seleccionada

Para resolver el problema del Sudoku, se utilizó el método de **Backtracking**, una técnica de exploración que emplea prueba y error para encontrar soluciones válidas. Este enfoque es ideal para problemas que tienen restricciones estrictas, como las que plantea el Sudoku: cada número debe ser único en una fila, columna y subcuadrícula.

Ventajas del Enfoque

1. **Ajuste Natural al Problema**: El Backtracking analiza celda por celda, retrocediendo cuando una elección no cumple con las restricciones. Esto facilita la resolución de problemas altamente estructurados como el Sudoku.
2. **Facilidad de Implementación**: Su implementación es directa y se puede mejorar con heurísticas, como priorizar celdas con menos opciones posibles.
3. **Reducción de Complejidad Práctica**: Aunque teóricamente es intensivo, el uso de heurísticas disminuye significativamente la cantidad de intentos fallidos.

Descripción del Algoritmo

El algoritmo implementado sigue los siguientes pasos:

1. **Identificación de Celdas Vacías**: Se identifican las celdas sin valor asignado. Estas se ordenan según la cantidad de opciones válidas disponibles para reducir el número de intentos fallidos.
2. **Prueba y Error**: Para cada celda vacía, se prueban los números del 1 al 9 y se verifica si cumplen las restricciones.

3. **Retroceso (Backtracking)**: Si un número no lleva a una solución válida, se deshace la asignación y se prueba con el siguiente número.
4. **Solución Final**: Cuando todas las celdas se llenan correctamente, se completa el Sudoku.

Visualización del Proceso

- Cada paso se grafica en tiempo real.
- Las asignaciones exitosas se resaltan en **azul**.
- Los errores no se muestran ya que el código al poner los errores se llegaba a trabar entonces para que este optimizado de manera correcta decidí solo poner el proceso.

Análisis de Complejidad

1. **Complejidad Temporal**:
    - En el peor caso, se exploran todas las combinaciones posibles de números (O(9^n), donde *n* es el número de celdas vacías). Sin embargo, las heurísticas disminuyen este número significativamente.
2. **Complejidad Espacial**:
    - La pila de recursión ocupa un espacio de O(n), siendo *n* el número de celdas vacías.

Resultados Obtenidos

1. **Sudoku Resuelto**: El algoritmo resolvió el Sudoku dado, mostrando paso a paso cada intento y los retrocesos necesarios.

2. **Tiempo de Ejecución**: En pruebas iniciales, el tiempo promedio de ejecución fue de 10 **segundos**, dependiendo del tablero y el hardware utilizado.

Comparación con Otras Técnicas

1. **Programación Dinámica**: No es adecuada para el Sudoku, ya que no hay subproblemas repetitivos claros que se puedan reutilizar.
2. **Divide y Vencerás**: Aunque podría dividir el tablero en subregiones, los subproblemas no son independientes y dependen de decisiones globales.

Por estas razones, el **Backtracking** es la técnica más adecuada para resolver el problema del Sudoku.

Instrucciones para Ejecutar el Código

1. **Requisitos Previos**:
    - Instalar Python 3.8 o superior.
    - Instalar la biblioteca matplotlib ejecutando:
2. **Guardar el Código**:
    - Copia el siguiente código y guárdalo en un archivo llamado sudoku_solver.py:

```
3. import time
4. import matplotlib
5. matplotlib.use("TkAgg")  # Asegura el uso del backend interactivo
6. import matplotlib.pyplot as plt
7.
8. # Función para encontrar celdas vacías en el tablero y ordenar por
   cantidad de opciones válidas
9. def encontrar_vacio_menor_opciones(tablero):
10.     vacios = []
11.     for i in range(9):
12.         for j in range(9):
13.             if tablero[i][j] == 0:
```

```
                opciones = sum(1 for num in range(1, 10) if
    es_valido(tablero, num, (i, j)))
                vacios.append((opciones, i, j))
    vacios.sort()
    return vacios[0][1:] if vacios else None

# Función para verificar si un número es válido en una posición
    específica
def es_valido(tablero, num, pos):
    fila, col = pos
    if num in tablero[fila]: return False
    if num in [tablero[i][col] for i in range(9)]: return False
    inicio_fila, inicio_col = (fila // 3) * 3, (col // 3) * 3
    for i in range(inicio_fila, inicio_fila + 3):
        for j in range(inicio_col, inicio_col + 3):
            if tablero[i][j] == num: return False
    return True

# Función para graficar el tablero y mostrarlo en tiempo real
def graficar_tablero_en_tiempo_real(tablero, proceso, intento=None,
    error=False):
    plt.clf()  # Limpia la figura antes de redibujar
    plt.title(f"Proceso {proceso}", fontsize=16, color="green" if not
    error else "red")

    for i in range(10):
        lw = 2 if i % 3 == 0 else 0.5
        plt.plot([i, i], [0, 9], color="black", linewidth=lw)
        plt.plot([0, 9], [i, i], color="black", linewidth=lw)

    for i in range(9):
        for j in range(9):
            if tablero[i][j] != 0:
                color = "black"  # Números originales en negro
                if intento and (i, j) == intento and error:
                    color = "red"  # Marcar intentos erróneos en rojo
                elif intento and (i, j) == intento:
                    color = "blue"  # Números colocados por el
    algoritmo en azul
                plt.text(j + 0.5, 8.5 - i, str(tablero[i][j]),
    fontsize=16, ha='center', va='center', color=color)

    plt.axis("off")
    plt.pause(1)  # Pausa para que el gráfico se actualice en tiempo
    real
```

```
52.
53.# Algoritmo de backtracking para resolver el Sudoku y graficar cada
   paso
54.def resolver_sudoku_paso_a_paso(tablero, proceso=1):
55.    vacio = encontrar_vacio_menor_opciones(tablero)
56.    if not vacio:
57.        graficar_tablero_en_tiempo_real(tablero, proceso)
58.        return True  # Sudoku resuelto
59.    fila, col = vacio
60.
61.    for num in range(1, 10):
62.        if es_valido(tablero, num, (fila, col)):
63.            tablero[fila][col] = num
64.            graficar_tablero_en_tiempo_real(tablero, proceso,
   intento=(fila, col))  # Mostrar tablero en tiempo real
65.            if resolver_sudoku_paso_a_paso(tablero, proceso + 1):
66.                return True
67.            tablero[fila][col] = 0  # Deshacer asignación
68.        else:
69.            # Mostrar intento erróneo
70.            graficar_tablero_en_tiempo_real(tablero, proceso,
   intento=(fila, col), error=True)
71.    return False
72.
73.# Tablero de ejemplo
74.tablero = [
75.    [5, 3, 0, 0, 7, 0, 0, 0, 0],
76.    [6, 0, 0, 1, 9, 5, 0, 0, 0],
77.    [0, 9, 8, 0, 0, 0, 0, 6, 0],
78.    [8, 0, 0, 0, 6, 0, 0, 0, 3],
79.    [4, 0, 0, 8, 0, 3, 0, 0, 1],
80.    [7, 0, 0, 0, 2, 0, 0, 0, 6],
81.    [0, 6, 0, 0, 0, 0, 2, 8, 0],
82.    [0, 0, 0, 4, 1, 9, 0, 0, 5],
83.    [0, 0, 0, 0, 8, 0, 0, 7, 9]
84.]
85.
86.# Configurar visualización interactiva
87.plt.ion()
88.plt.figure(figsize=(6, 6))
89.
90.# Ejecutar el algoritmo
91.print("Resolviendo Sudoku paso a paso:")
92.inicio = time.time()
93.resolver_sudoku_paso_a_paso(tablero)
```

```
fin = time.time()

print(f"\nTiempo de ejecución: {fin - inicio:.4f} segundos")
plt.ioff()  # Desactivar el modo interactivo al finalizar
plt.show()
```

3. **Ejecutar el Programa**:
    - Usa este comando en la terminal:
4. **Resultados**:
    - Observa la solución del Sudoku en tiempo real.