Tarea 4: BackTracking

Algoritmos Avanzados

David Míreles Gutiérrez A00836010

Monterrey, N.L. 9/22/2024

PseudoCódigo

Python en mi opinión es muy cercano a pseudocodigo una vez que lo dominas, si no le parece bien puede tomar mis comentarios como pseudocódigo.

```python
def validAddition(board, row, col, num):
    #Check Row
    for x in range(9):
        if board[row][x] == num:
       #Num Already Used
            return False
    #Check Column
    for x in range(9):
        if board[x][col] == num:
       #Num Already Used
            return False

    #Check 3x3 By Getting Start Position of the GRID

    startRow = row - row % 3
    startCol = col - col % 3
    for i in range(3):
        for j in range(3):
            if board[i + startRow][j + startCol] == num:
             #Num Already Used
                return False
    #Valid Position
    return True

def solveSudoku(board):
    #Set Initial Varaibles
    row = -1
    col = -1
    empty = False

    #Found First 0 in Board
    for i in range(N):
        for j in range(N):
            if board[i][j] == 0:
                row, col = i, j
                empty = True
                break
        if empty:
            break
    #IF NOT FOUND A 0 RETURN TRUE, SUDOKU RESOLVED
    if not empty:
        return True
```

```
    #TRY EVERY COMBINATION FROM 1 TO 9 TO INSERT IN SUDOKU
    for num in range(1, 10):
        #CHECK IF ITS VALID NUMEBER AND SET IT IN THE BOARD
        if validAddition(board, row, col, num):
            board[row][col] = num

            #TRY TO SOLVE SUDOKU WITH THAT NUMBER AND JUMP TO THE NEXT EMPTY
PLACE
            if solveSudoku(board):
                #SUDOKU SOLVED
                return True
            #BACKTRACKED SUDOKU WASNT SOLVED WITH THE NUMBER SET PREVIOUSLY
GO BACK AND TRY AGAIN
            board[row][col] = 0
    #UNABLE TO SET ANY NUMBER FROM 1 TO 9 IN THAT CELL. IMPOSSIBLE TO SOLVE
SUDOKU
    return False
```

Análisis de Complejidad

Python

```python
def validAddition(board, row, col, num): #O(N^2)
    for x in range(N):  # O(N)
        if board[row][x] == num:  # O(1)
            return False  # O(1)

    for x in range(N):  # O(N)
        if board[x][col] == num:  # O(1)
            return False  # O(1)

    startRow = row - row % 3  # O(1)
    startCol = col - col % 3  # O(1)
    for i in range(3):  # O(3) (siempre 3 iteraciones)
        for j in range(3):  # O(3^2) (siempre 3 iteraciones)
            if board[i + startRow][j + startCol] == num:  # O(1)
                return False  # O(1)

    return True  # O(1)

def solveSudoku(board): #O
    row = -1  # O(1)
    col = -1  # O(1)
```

```
    empty = False  # O(1)

#WORST CASE O(N^2) HAS TO CHECK ALL THE BOARD
    for i in range(N):  # O(N)
        for j in range(N):  # O(N^2)
            if board[i][j] == 0:  # O(1)
                row, col = i, j  # O(1)
                empty = True  # O(1)
                break  # O(1)
        if empty:  # O(1)
            break  # O(1)

    if not empty:  # O(1)
        return True  # O(1)

#WORST CASE O(9^N)
    for num in range(1, 10):  # O(1) (siempre 9 iteraciones)
        if validAddition(board, row, col, num):  # O(N)
            board[row][col] = num  # O(1)

            if solveSudoku(board):  # Llamada recursiva, complejidad O(9^N)
                return True  # O(1)

            board[row][col] = 0  # O(1)

    return False  # O(1)
```

La complejidad del Algoritmo es de O(9^N) dado por la adición de la función de validAddition dado que si no tuvieras la función validAddition, el algoritmo tendría que intentar todas las combinaciones posibles de números del 1 al 9 en todas las 81 celdas del tablero.

Esto daría una complejidad en el peor de los casos de O(9^81), que puede expresarse como O(9^N×9), donde N=9 es la longitud del tablero y 9 es el número de posibilidades (números del 1 al 9) para cada celda.

En lugar de probar todas las combinaciones posibles de números en todas las celdas, la función validAddition descarta muchas combinaciones inválidas, haciendo que el algoritmo sea mucho más eficiente en la práctica y por esta razón O(9^N) dado que en el peor caso la recursión ocupa ocurrir en toda una línea y si no es posible se rompe. Esto significa que la recursión no necesita explorar todas las posibles combinaciones en el tablero completo, sino solo las ramas válidas del árbol de búsqueda. Si en algún momento no es posible encontrar un

número válido, la recursión se detiene, lo que evita continuar con combinaciones infructuosas.

Ejemplo de Árbol de Recursión

PARA:
[5, 3, 0, 0, 7, 0, 0, 0, 0]
[6, 0, 0, 1, 9, 5, 0, 0, 0]
[0, 9, 8, 0, 0, 0, 0, 6, 0]
[8, 0, 0, 0, 6, 0, 0, 0, 3]
[4, 0, 0, 8, 0, 3, 0, 0, 1]
[7, 0, 0, 0, 2, 0, 0, 0, 6]
[0, 6, 0, 0, 0, 0, 2, 8, 0]
[0, 0, 0, 4, 1, 9, 0, 0, 5]
[0, 0, 0, 0, 8, 0, 0, 7, 9]