

Kakkuro Game Solver - Primera entrega - Proyecto Análisis de Algoritmos 2023-30

Anderson J. Alvarado^a, Santiago Rey^a

^a*Pontificia Universidad Javeriana, Bogotá, Colombia*

Abstract

Este documento presenta un algoritmo de validación para el juego Kakkuro, que verifica si un tablero cumple con las restricciones de suma en las celdas sombreadas.

Keywords: Kakkuro, Juego de rompecabezas, Validación de tablero, Restricciones de suma, Algoritmo de validación, Tablero de juego, Solución de rompecabezas, Matriz de juego, Algoritmo de verificación, Reglas de Kakkuro.

Índice

1. Introducción	3
2. Análisis del Problema	3
2.1. Definiciones Formales	3
2.2. Reglas y Restricciones	3
2.2.1. Reglas del Juego	3
2.2.2. Restricciones del Juego	4
3. Diseño del Problema	4
3.1. Entrada del Problema	4
3.2. Salida del Problema	5
3.3. Ejemplo de Entrada y Salida	5
3.3.1. Entrada de Ejemplo	5
3.3.2. Entrada de Ejemplo	5

*Este documento presenta el algoritmo de validación para el juego Kakkuro, que verifica si un tablero cumple con las restricciones de suma en las celdas sombreadas.

Email addresses: andersonjalvarado@javeriana.edu.co (Anderson J. Alvarado),
screyb@javeriana.edu.co (Santiago Rey)

<i>ÍNDICE</i>	2
3.4. Consideraciones Adicionales	6
4. Algoritmo de Solución Jugador	6
5. Análisis de Complejidad	12
5.1. Procedimiento ‘read_matrices’	12
5.2. Procedimiento ‘is_valid_kakkuro’	13
5.3. Procedimiento ‘validate_sum_down’	13
5.4. Procedimiento ‘validate_sum_right’	13
5.5. Procedimiento ‘validate_entry’	14
5.6. Procedimiento ‘show_random_matrix’	14
5.7. Complejidad General del Algoritmo	14
6. Algoritmo de Solución Máquina	15
7. Heurística	20
7.1. Menor Dominio Primero	20
8. Implementación en Python	20
9. Pantallas de la Aplicación Kakkuro Python	20
9.1. Pantalla Principal	21
9.2. Tablero Seleccionado	22
9.2.1. Solucionar Tablero	23
9.3. Tablero Solucionado	24
9.4. Kakkuro No Válido	25
9.5. Kakkuro Válido	25
10. Conclusiones	26

1. Introducción

El juego Kakkuro es un desafío de lógica y números que implica llenar un tablero con dígitos de acuerdo con ciertas reglas y restricciones. En este juego, se proporciona un tablero de celdas sombreadas y celdas vacías, junto con restricciones de suma para grupos de celdas sombreadas. El objetivo es llenar las celdas vacías con dígitos de manera que se cumplan todas las restricciones de suma. En este artículo, se presenta un algoritmo para validar si un tablero de juego Kakkuro cumple con todas las restricciones de suma.

2. Análisis del Problema

2.1. Definiciones Formales

Antes de proponer soluciones algorítmicas, es importante establecer definiciones formales relacionadas con el problema:

Definición 1. Un **Kakkuro** es un juego de lógica y números que implica llenar un tablero con dígitos de acuerdo con ciertas reglas y restricciones.

Definición 2. Una **Celda Sombreada** es una celda en el tablero de juego que debe contener un dígito y está sombreada para indicar una restricción.

Definición 3. Una **Celda Vacía** es una celda en el tablero de juego que debe contener un dígito del 1 al 9.

Definición 4. Una **Suma Válida en una Celda** significa que la suma de los dígitos en una celda debe ser igual a una restricción específica.

Definición 5. Una **Suma Válida en un Grupo de Celdas Sombreadas** significa que la suma de los dígitos en un grupo de celdas sombreadas debe ser igual a una restricción específica para ese grupo.

2.2. Reglas y Restricciones

El juego de Kakkuro tiene reglas y restricciones específicas que deben cumplirse:

2.2.1. Reglas del Juego

1. El tablero de juego consta de celdas sombreadas y celdas vacías.
2. Cada celda sombreada debe contener un dígito.
3. La suma de los dígitos en cada celda sombreada debe cumplir con una restricción de suma específica.
4. Los dígitos en una misma celda no pueden repetirse.
5. Los dígitos en una misma fila no pueden repetirse.
6. Los dígitos en una misma columna no pueden repetirse.

2.2.2. Restricciones del Juego

1. Cada grupo de celdas sombreadas tiene una restricción de suma única.
2. Las restricciones de suma pueden variar en valor.
3. Las restricciones de suma se aplican solo a las celdas sombreadas dentro del grupo.
4. La suma de los dígitos en un grupo de celdas sombreadas debe ser igual a la restricción correspondiente.

3. Diseño del Problema

El problema consiste en desarrollar un algoritmo que valide un tablero de juego Kakkuro, un tipo de rompecabezas numérico en el que se deben colocar números en celdas de una cuadrícula. El tablero de juego Kakkuro tiene las siguientes características:

- Es una matriz cuadrada de tamaño $n \times n$, donde n es un número entero positivo.
- Cada celda de la matriz puede contener uno de los siguientes valores:
 - Un número entero del 1 al 9.
 - Una celda vacía representada como 0.
 - Una celda sombreada con una restricción de suma.
- Las restricciones de suma se representan como fracciones en las celdas sombreadas. Por ejemplo, una celda sombreada puede contener la restricción $2/3$, lo que significa que la suma de los números en las celdas adyacentes hacia abajo debe ser igual a 2, y la suma de los números en las celdas adyacentes hacia la derecha debe ser igual a 3.

El objetivo del algoritmo es determinar si el tablero de juego Kakkuro cumple con todas las restricciones de suma en las celdas sombreadas. Si todas las restricciones se cumplen, el tablero se considera válido; de lo contrario, se considera inválido.

3.1. Entrada del Problema

La entrada del problema consiste en un tablero de juego Kakkuro representado como una matriz bidimensional $n \times n$. Cada celda de la matriz contiene uno de los siguientes valores:

- Un número entero del 1 al 9.

- La representación de una celda vacía como 0.
- Una cadena que representa una restricción de suma en el formato x/y , donde x y y son números enteros positivos que indican la suma requerida en las direcciones hacia abajo y hacia la derecha, respectivamente. Si no hay restricción en una celda sombreada, se representa como 0.

3.2. Salida del Problema

La salida del problema es un valor booleano que indica si el tablero de juego Kakkuro es válido o no. El algoritmo debe devolver Verdadero si todas las restricciones de suma en las celdas sombreadas se cumplen; de lo contrario, debe devolver Falso.

3.3. Ejemplo de Entrada y Salida

3.3.1. Entrada de Ejemplo

Consideremos un tablero de juego Kakkuro de tamaño 4×4 con las siguientes celdas:

4	4		
0	0	-1	-1
0	1	16	-1
0	2	-1	-1
0	3	13	-1
1	0	-1	9
1	2	4	6
2	0	-1	12
3	0	-1	-1
3	1	-1	6

Figura 1: Kakkuro Válido

3.3.2. Entrada de Ejemplo

Para el tablero de ejemplo anterior, el algoritmo debe devolver Verdadero, ya que todas las restricciones de suma en las celdas sombreadas se cumplen. Una posible solución válida es la siguiente:

$$\begin{bmatrix} X & 16/ & X & 13/ \\ /9 & 9 & 4/6 & 6 \\ /12 & 7 & 3 & 2 \\ X & /6 & 1 & 5 \end{bmatrix}$$

3.4. Consideraciones Adicionales

- El juego Kakuro también admite matrices que no sean cuadradas, pero para esta implementación solo se tuvieron en cuenta matrices cuadradas para realizar los tableros del juego.
- Asegúrese de implementar el algoritmo de manera eficiente, ya que el tamaño del tablero de juego Kakkuro puede ser grande.
- El algoritmo debe manejar correctamente casos en los que haya múltiples celdas sombreadas con restricciones de suma en el mismo grupo.

4. Algoritmo de Solución Jugador

El algoritmo de solución tiene como objetivo solucionar un tablero de juego Kakuro para asegurarse de que cumple con todas las restricciones de suma en las celdas sombreadas. El algoritmo recibe como entrada el tablero de juego y una matriz de restricciones que indica las sumas válidas para cada grupo de celdas sombreadas en el tablero. Luego, verifica si todas las sumas en los grupos de celdas sombreadas son válidas de acuerdo con las restricciones proporcionadas. Si todas las sumas son válidas, el tablero se considera válido y el algoritmo devuelve Verdadero. En caso contrario, devuelve Falso.

Algoritmo 1 Procedimiento para validar un tablero Kakkuro

Require: Tablero de juego *board*

Ensure: Verdadero si el tablero cumple con las restricciones, Falso de lo contrario

```

0: procedure ISVALIDKAKKURO(board)
0:   for cada fila i en board do
0:     for cada celda j en i do
0:       cellValue  $\leftarrow$  Valor de la celda en la posición (i, j) en board
0:       if cellValue  $\neq$  Null y es un dígito then
0:         Separar cellValue en partes n y m
0:         n  $\leftarrow$  Convertir n a entero si es un dígito, de lo contrario Null
0:         m  $\leftarrow$  Convertir m a entero si es un dígito, de lo contrario Null
0:         if n  $\neq$  Null y no se cumple ValidateSumDown(board, i, j, n) then
0:           return Falso
0:         end if
0:         if m  $\neq$  Null y no se cumple ValidateSumRight(board, i, j, m) then
0:           return Falso
0:         end if
0:       end if
0:     end for
0:   end for
0:   return Verdadero
0: end procedure

```

Explicación del Procedimiento ‘IsValidKakkuro’: Este procedimiento toma un tablero de juego Kakkuro como entrada y verifica si cumple con las restricciones de suma en las celdas sombreadas. Itera a través de cada celda del tablero y, si encuentra una celda sombreada con una restricción de suma (/"), separa la restricción en dos valores, *n* y *m*. Luego, verifica si se cumple la restricción de suma hacia abajo (*n*) y hacia la derecha (*m*) utilizando los procedimientos *ValidateSumDown* y *ValidateSumRight* respectivamente.

Algoritmo 2 Procedimiento para mostrar un tablero Kakkuro aleatorio

```

0: procedure SHOWRANDOMMATRIX
0:   Acceder a la variable global random_matrices
0:   Obtener el tamaño seleccionado selected_size desde self.size_var
0:   if selected_size no es nulo then
0:     Obtener la matriz aleatoria random_matrix correspondiente a
       selected_size desde random_matrices
0:     if self.entry_matrix existe then
0:       Eliminar las entradas y etiquetas anteriores
0:     end if
0:     if self.entry_matrix no existe then
0:       Crear las entradas y etiquetas según los valores de random_matrix
0:     end if
0:     if self.submit_button existe then
0:       Eliminar el botón de verificación anterior
0:     end if
0:     Crear un botón de verificación de tablero con texto "Verificar Tableroz co-
       mando self.validate_board
0:     if self.restart_button existe then
0:       Eliminar el botón de reinicio anterior
0:     end if
0:     Crear un botón de reinicio de juego con texto Reiniciar Juegoz comando
       self.restart_game
0:   else
0:     Mostrar un mensaje de advertencia si no se seleccionó un tamaño de matriz
0:   end if
0: end procedure=0

```

Explicación del Procedimiento ‘ShowRandomMatrix’: Este procedimiento muestra un tablero Kakkuro aleatorio en la interfaz de usuario. Accede a *random_matrices* para llevar un registro de las matrices aleatorias generadas. Cuando se selecciona un tamaño válido, se muestra el tablero, eliminando el tablero anterior si existe. Luego, crea las entradas y etiquetas según la matriz aleatoria *random_matrix*. Se actualiza el botón de verificación del tablero y se añade un botón de reinicio del juego. Si no se selecciona un tamaño válido, se muestra una advertencia para que el usuario elija un tamaño de matriz.

Algoritmo 3 Procedimiento para leer matrices desde un archivo de texto

Require: Nombre de archivo *filename*

Ensure: Diccionario de matrices *matrices*

```

0: procedure READMATRICES(filename)
0:   matrices  $\leftarrow \{\}$ 
0:   currentId  $\leftarrow$  Null
0:   currentMatrix  $\leftarrow []$ 
0:   Abrir el archivo filename para lectura
0:   Leer todas las líneas del archivo y almacenarlas en lines
0:   for cada línea line en lines do
0:     line  $\leftarrow$  Eliminar espacios en blanco de line
0:     if line es un número entero then
0:       if currentId no es Null then
0:         Agregar currentMatrix al diccionario matrices con clave currentId
0:       end if
0:       currentId  $\leftarrow$  Convertir line a entero
0:       currentMatrix  $\leftarrow []$ 
0:     else
0:       rowValues  $\leftarrow []$ 
0:       Separar line en celdas por comas y agregarlas a rowValues
0:       Agregar rowValues a currentMatrix
0:     end if
0:   end for
0:   if currentId no es Null then
0:     Agregar currentMatrix al diccionario matrices con clave currentId
0:   end if
0:   Cerrar el archivo
0:   return matrices
0: end procedure

```

Explicación del Procedimiento ‘ReadMatrices’: Este procedimiento toma el nombre de un archivo como entrada y lee las matrices contenidas en el archivo. Cada matriz comienza con un número entero que representa su identificador. Las matrices se almacenan en un diccionario donde la clave es el identificador y el valor es la matriz en forma de lista de listas. El procedimiento recorre el archivo línea por línea, detecta los identificadores y las filas de las matrices, y las almacena en el diccionario.

Algoritmo 4 Procedimiento para validar la suma hacia abajo en un tablero Kakkuro

Require: Tablero de juego *board*, Fila de inicio *row*, Columna de inicio *col*, Valor de suma esperado *n*

Ensure: Verdadero si la suma hacia abajo se cumple, Falso de lo contrario

```

0: procedure VALIDATESUMDOWN(board, row, col, n)
0:   currentSum  $\leftarrow$  0
0:   Conjunto de valores únicos seenValues  $\leftarrow$  {}
0:   for cada fila i desde row + 1 hasta el final de la columna col en board do
0:     cellValue  $\leftarrow$  Valor de la celda en la posición (i, col) en board
0:     cellValue2  $\leftarrow$  Valor de la celda en la posición (i, col) en randomMatrix
0:     if cellValue  $\neq$  Null y es un dígito then
0:       value  $\leftarrow$  Convertir cellValue a entero
0:       if value está en seenValues then
0:         Mostrar error: "Valores repetidos en la misma columna o fila."
0:         return Falso
0:       end if
0:       currentSum  $\leftarrow$  currentSum + value
0:       Agregar value a seenValues
0:     else if cellValue2  $\neq$  Null y contiene /.º "X" then
0:       Romper el bucle (detener la suma hacia abajo)
0:     else
0:       return Falso
0:     end if
0:   end for
0:   return currentSum == n
0: end procedure

```

Explicación del Procedimiento ‘ValidateSumDown’: Este procedimiento toma un tablero de juego Kakkuro, una posición de inicio (fila y columna), y un valor de suma esperado *n*. Luego, verifica si la suma hacia abajo desde la posición de inicio cumple con la restricción de suma *n*. Recorre las celdas hacia abajo desde la posición de inicio y suma sus valores. Si encuentra un valor repetido en la misma columna o fila, muestra un error y devuelve Falso. Si encuentra una celda /.º "X", detiene la suma hacia abajo. Si la suma es igual a *n*, devuelve Verdadero; de lo contrario, devuelve Falso.

Algoritmo 5 Procedimiento para validar la suma hacia la derecha en un tablero Kakkuro

Require: Tablero de juego *board*, Fila de inicio *row*, Columna de inicio *col*, Valor de suma esperado *m*

Ensure: Verdadero si la suma hacia la derecha se cumple, Falso de lo contrario

```

0: procedure VALIDATESUMRIGHT(board, row, col, m)
0:   currentSum  $\leftarrow$  0
0:   Conjunto de valores únicos seenValues  $\leftarrow$  {}
0:   for cada columna j desde col + 1 hasta el final de la fila row en board do
0:     cellValue  $\leftarrow$  Valor de la celda en la posición (row, j) en board
0:     cellValue2  $\leftarrow$  Valor de la celda en la posición (row, j) en randomMatrix
0:     if cellValue  $\neq$  Null y es un dígito then
0:       value  $\leftarrow$  Convertir cellValue a entero
0:       if value está en seenValues then
0:         Mostrar error: "Valores repetidos en la misma columna o fila."
0:         return Falso
0:       end if
0:       currentSum  $\leftarrow$  currentSum + value
0:       Agregar value a seenValues
0:     else if cellValue2  $\neq$  Null y contiene /.o "X" then
0:       Romper el bucle (detener la suma hacia la derecha)
0:     else
0:       return Falso
0:     end if
0:   end for
0:   return currentSum == m
0: end procedure

```

Explicación del Procedimiento ‘ValidateSumRight’: Este procedimiento toma un tablero de juego Kakkuro, una posición de inicio (fila y columna), y un valor de suma esperado *m*. Luego, verifica si la suma hacia la derecha desde la posición de inicio cumple con la restricción de suma *m*. Recorre las celdas hacia la derecha desde la posición de inicio y suma sus valores. Si encuentra un valor repetido en la misma columna o fila, muestra un error y devuelve Falso. Si encuentra una celda /.^o "X", detiene la suma hacia la derecha. Si la suma es igual a *m*, devuelve Verdadero; de lo contrario, devuelve Falso.

Algoritmo 6 Procedimiento para validar la entrada en una celda de la matriz

Require: Nuevo valor ingresado *newValue*

Ensure: Verdadero si la entrada es válida, Falso de lo contrario

```

0: procedure VALIDATEENTRY(newValue)
0:   if newValue es una cadena vacía then
0:     return Verdadero
0:   end if
0:   Convertir newValue a entero y almacenar en value
0:   if value está en el rango de 1 a 9 then
0:     return Verdadero
0:   else
0:     return Falso
0:   end if
0:   return Falso
0: end procedure

```

Explicación del Procedimiento ‘ValidateEntry’: Este procedimiento toma un nuevo valor ingresado en una celda de la matriz y verifica si es una entrada válida. Si el valor es una cadena vacía, se considera válido (permite celdas vacías). Luego, intenta convertir el valor a un entero y verifica si está en el rango de 1 a 9. Si cumple con estas condiciones, se considera una entrada válida y devuelve Verdadero; de lo contrario, devuelve Falso.

5. Análisis de Complejidad

El algoritmo para validar un tablero de juego Kakkuro consta de varios procedimientos que trabajan juntos para determinar si todas las restricciones de suma en las celdas sombreadas se cumplen. La complejidad del algoritmo depende de varios factores, incluido el tamaño del tablero, la cantidad de celdas sombreadas y las restricciones presentes. A continuación, se presenta un análisis de complejidad general que considera estos factores.

5.1. Procedimiento ‘read_matrices’

El procedimiento ‘read_matrices’ se encarga de leer las matrices desde un archivo de texto y convertirlas en un diccionario de matrices. Su complejidad está dominada por la lectura de las líneas del archivo y la creación de las matrices. Supongamos que hay m matrices en el archivo, cada una de tamaño $n \times n$.

- Lectura de líneas del archivo: $\mathcal{O}(m)$
- Creación de matrices: $\mathcal{O}(m \cdot n^2)$

Por lo tanto, la complejidad total de este procedimiento es $\mathcal{O}(m \cdot n^2)$.

5.2. Procedimiento ‘is_valid_kakkuro’

El procedimiento ‘is_valid_kakkuro’ verifica si un tablero de juego Kakkuro cumple con todas las restricciones de suma en las celdas sombreadas. Supongamos que el tablero tiene un tamaño de $n \times n$ y que hay k celdas sombreadas con restricciones.

- Iteración a través de cada celda del tablero: $\mathcal{O}(n^2)$
- Verificación de restricciones de suma: $\mathcal{O}(k)$ en el peor caso

Por lo tanto, la complejidad total de este procedimiento es $\mathcal{O}(n^2 \cdot k)$.

5.3. Procedimiento ‘validate_sum_down’

El procedimiento ‘validate_sum_down’ verifica la suma hacia abajo desde una celda de inicio en un tablero Kakkuro. Supongamos que la suma se realiza en una columna de altura h .

- Iteración a través de las celdas en la columna: $\mathcal{O}(h)$
- Verificación de valores y sumas: $\mathcal{O}(1)$ por celda

Por lo tanto, la complejidad total de este procedimiento es $\mathcal{O}(h)$.

5.4. Procedimiento ‘validate_sum_right’

El procedimiento ‘validate_sum_right’ verifica la suma hacia la derecha desde una celda de inicio en un tablero Kakkuro. Supongamos que la suma se realiza en una fila de longitud w .

- Iteración a través de las celdas en la fila: $\mathcal{O}(w)$
- Verificación de valores y sumas: $\mathcal{O}(1)$ por celda

Por lo tanto, la complejidad total de este procedimiento es $\mathcal{O}(w)$.

5.5. Procedimiento ‘validate_entry’

El procedimiento ‘validate_entry’ verifica si un nuevo valor ingresado en una celda de la matriz es válido. Su complejidad está dominada por la conversión del valor a entero y la verificación del rango.

- Conversión a entero y verificación de rango: $\mathcal{O}(1)$

Por lo tanto, la complejidad total de este procedimiento es $\mathcal{O}(1)$.

5.6. Procedimiento ‘show_random_matrix’

El procedimiento ‘show_random_matrix’ muestra un tablero Kakkuro aleatorio en la interfaz de usuario. Su complejidad está dominada por la lectura y procesamiento de la matriz aleatoria desde un archivo, así como la creación o actualización de las entradas y etiquetas en la interfaz de usuario.

- Lectura y procesamiento de la matriz aleatoria: $\mathcal{O}(n^2)$
- Creación o actualización de elementos de la interfaz de usuario: $\mathcal{O}(size^2)$

Por lo tanto, la complejidad total de este procedimiento es $\mathcal{O}(n^2)$ en el peor caso, donde n es el tamaño seleccionado para el tablero.

5.7. Complejidad General del Algoritmo

La complejidad general del algoritmo para validar un tablero de juego Kakkuro depende de varios factores, incluido el tamaño del tablero ($n \times n$), la cantidad de celdas sombreadas con restricciones (k) y las restricciones presentes en el tablero. En el peor caso, la complejidad total puede ser $\mathcal{O}(n^2 \cdot k)$, pero puede variar según las características del tablero.

6. Algoritmo de Solución Máquina

El algoritmo de solución tiene como objetivo solucionar un tablero de juego Kakuro de forma exhaustiva teniendo en cuenta algunas heurísticas para mejorar la eficiencia del algoritmo de solución.

Los algoritmos utilizados son:

Algoritmo 7 get_cell_options

```

0: function GET_CELL_OPTIONS(empty_cells)
0:   cell_options  $\leftarrow$  {cell: list(range(1, 10)) for cell in empty_cells}
0:   for cell in cell_options do
0:     value_one_solution  $\leftarrow$  False
0:     row, col  $\leftarrow$  cell
0:     if col > 0 and col < len(self.random_matrix[row]) - 1 then
0:       row_left_value  $\leftarrow$  GET_CELL_VALUE(row, col-1)
0:       row_right_value  $\leftarrow$  GET_CELL_VALUE(row, col+1)
0:       if row_left_value is not None and row_right_value is not None then
0:         value  $\leftarrow$  row_left_value.split('
0:       ')
0:       cell_options[cell]  $\leftarrow$  [int(value[1])]
0:       value_one_solution  $\leftarrow$  True
0:     end if
0:     else if col == len(self.random_matrix[row]) - 1 then
0:       row_left_value  $\leftarrow$  GET_CELL_VALUE(row, col-1)
0:       if row_left_value is not None then
0:         value  $\leftarrow$  row_left_value.split('
0:       ')
0:       cell_options[cell]  $\leftarrow$  [int(value[1])]
0:       value_one_solution  $\leftarrow$  True
0:     end if
0:     end if
0:     if row > 0 and row < len(self.random_matrix) - 1 then
0:       col_up_value  $\leftarrow$  GET_CELL_VALUE(row-1, col)
0:       col_down_value  $\leftarrow$  GET_CELL_VALUE(row+1, col)
0:       if col_up_value is not None and col_down_value is not None then
0:         value  $\leftarrow$  col_up_value.split('
0:       ')
0:       cell_options[cell]  $\leftarrow$  [int(value[0])]
0:       value_one_solution  $\leftarrow$  True
0:     end if
0:     else if row == len(self.random_matrix) - 1 then
0:       col_up_value  $\leftarrow$  GET_CELL_VALUE(row-1, col)
0:       if col_up_value is not None then
0:         value  $\leftarrow$  col_up_value.split('
0:       ')
0:       cell_options[cell]  $\leftarrow$  [int(value[0])]
0:       value_one_solution  $\leftarrow$  True
0:     end if
0:   end if
0:

```

Algoritmo 8 get_cell_options Continuación

```

0: function GET_CELL_OPTIONS(empty_cells)
0:   sum_row  $\leftarrow$  []
0:   sum_col  $\leftarrow$  []
0:   if value_one_solution is False then
0:     for i in range(col, -1, -1) do
0:       value_sum  $\leftarrow$  GET_CELL_VALUE(row, i)
0:       if value_sum is not None then
0:         break
0:       end if
0:     end for
0:     value_sum  $\leftarrow$  value_sum.split(
0:       ')
0:     if int(value_sum[1]) < 10 then
0:       sum_row  $\leftarrow$  list(range(1, int(value_sum[1])))
0:     end if
0:     {COLUMNAS}
0:     for i in range(row, -1, -1) do
0:       value_sum  $\leftarrow$  GET_CELL_VALUE(i, col)
0:       if value_sum is not None then
0:         break
0:       end if
0:     end for
0:     value_sum  $\leftarrow$  value_sum.split(
0:       ')
0:     if int(value_sum[0]) < 10 then
0:       sum_col  $\leftarrow$  list(range(1, int(value_sum[0])))
0:     end if
0:     if sum_row and sum_col then
0:       cell_options[cell]  $\leftarrow$  list(set(sum_row) & set(sum_col))
0:     else if sum_row then
0:       cell_options[cell]  $\leftarrow$  sum_row
0:     else if sum_col then
0:       cell_options[cell]  $\leftarrow$  sum_col
0:     end if
0:   end if
0:   return cell_options
0: end function=0

```

Explicación del Procedimiento ‘get_cell_options’: Este procedimiento toma es una heurística en la que se calculan el número posible de soluciones para una celda vacía, Priorizando las celdas en las que solo tienen una solución y en las que la suma a su derecha o abajo del número bloqueado es menor a 9.

Algoritmo 9 solve_kakkuro

```

0: function SOLVE_KAKKURO(self)
0:   empty_cells  $\leftarrow$  []
0:   for i in range(len(self.random_matrix)) do
0:     for j in range(len(self.random_matrix[i])) do
0:       if self.random_matrix[i][j] == '0' then
0:         empty_cells.append((i, j))
0:       end if
0:     end for
0:   end for
0:   cell_options  $\leftarrow$  GET_CELL_OPTIONS(empty_cells)
0:   print('cell_opt', cell_options)
0:   cell_options  $\leftarrow$  DELETE_CELL_OPTIONS(cell_options)
0:   print('cell_opt clean', cell_options)
0:   Sort empty_cells by length(cell_options[cell]) {Ordenar las celdas vacías
    por el número de opciones posibles}
0:   print('empty sort', empty_cells)
0:   print('Iterar:', *[cell_options[cell] for cell in empty_cells])
0:   found_solution  $\leftarrow$  False
0:   for combination in ITERTOOLS.PRODUCT(*[cell_options[cell] for cell in em-
    pty_cells]) do
0:     temp_matrix  $\leftarrow$  copy(self.random_matrix)
0:     for index, (row, col) in ENUMERATE(empty_cells) do
0:       temp_matrix[row][col]  $\leftarrow$  str(combination[index]) {Probar cada combi-
        nación de números en el tablero}
0:     end for
0:     if IS_VALID_KAKKURO(temp_matrix, self.random_matrix) then
0:       return temp_matrix
0:     end if
0:   end for
0:   return None
0: end function=0

```

Explicación del Procedimiento ‘solve_kakkuro’:

Este procedimiento ‘solve_kakkuro’ es la función principal para resolver el juego de Kakuro. Comienza identificando todas las celdas vacías en la cuadrícula del rompecabezas, que están representadas por ‘0’ en la ‘random_matrix’. Estas celdas vacías se almacenan en la lista ‘empty_cells’.

A continuación, la función llama a ‘get_cell_options’ para generar un diccionario, ‘cell_options’, donde las claves son las celdas vacías (representadas como tuplas de índices de fila y columna) y los valores son listas de posibles números que se pueden colocar en esas celdas.

Luego, la función llama a ‘delete_cell_options’ para refinar el diccionario ‘cell_options’ eliminando cualquier número que no sea una opción válida para una celda basándose en las reglas de Kakuro. El diccionario ‘cell_options’ refinado se imprime luego en la consola.

La lista ‘empty_cells’ se ordena en función del número de opciones disponibles para cada celda. Esto se hace para priorizar las celdas con menos opciones, lo que puede ayudar a acelerar la búsqueda de una solución.

La función entra en un bucle donde prueba todas las combinaciones posibles de números en las celdas vacías. Esto se hace utilizando la función ‘itertools.product’, que genera el producto cartesiano de las listas de opciones para cada celda. Para cada combinación, se hace una copia temporal de la cuadrícula del rompecabezas y los números de la combinación se colocan en las celdas vacías.

La función verifica si la combinación es una solución válida al rompecabezas llamando a ‘is_valid_kakkuro’. Si la combinación es válida, la función establece ‘found_solution’ en ‘True’ y devuelve la cuadrícula temporal como la solución. Si no se encuentra una solución válida después de probar todas las combinaciones, la función devuelve ‘None’.

7. Heurística

Para poder mejorar el algoritmo, se utilizó la siguiente heurística:

7.1. *Menor Dominio Primero*

Se basa en el principio de que las celdas con menos posibles valores son las más probables de ser correctas.

Esta heurística consiste en seleccionar la variable con el menor dominio es decir, la menor cantidad de valores posibles que puede tomar, para ser asignada primero. En el juego de Kakuro, esta estrategia puede ayudar a reducir el espacio de búsqueda y facilitar la resolución del juego al priorizar las variables con menos opciones.

Para este caso se tienen dos restricciones en las funciones `get_cell_options` y `delete_cell_options`, las cuales buscan disminuir las opciones posibles que se pueden probar al buscar la combinación correcta que solucione el Kakuro.

8. Implementación en Python

Para la implementación se utiliza la biblioteca ‘tkinter’ para crear una interfaz gráfica que permite seleccionar el tamaño de la matriz y llenar las celdas del tablero. Luego, verifica si el tablero ingresado es válido según las restricciones de suma de Kakkuro.

El código adjuntado en el archivo *Proyecto.py* incluye explicaciones detalladas de cada parte del programa para facilitar su comprensión.

9. Pantallas de la Aplicación Kakkuro Python

En esta sección, se presentan algunas capturas de pantalla de la aplicación Kakkuro junto con explicaciones breves de cada pantalla.

9.1. Pantalla Principal



Figura 2: Pantalla Principal de la Aplicación Kakkuro

En la Pantalla Principal, los usuarios pueden seleccionar el tamaño del tablero Kakkuro que desean resolver. Esta pantalla es el punto de partida para comenzar el juego.

9.2. Tablero Seleccionado

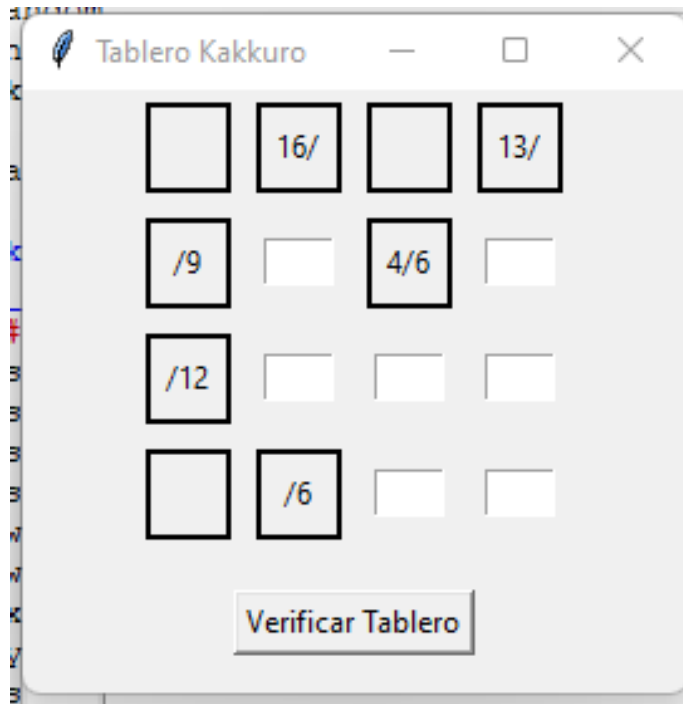


Figura 3: Tablero Seleccionado con Tamaño 4x4

Después de seleccionar un tamaño de tablero, se muestra el Tablero Seleccionado. Aquí, los usuarios verán el tablero Kakkuro generado aleatoriamente, con algunas celdas que contienen valores iniciales.

9.2.1. Solucionar Tablero

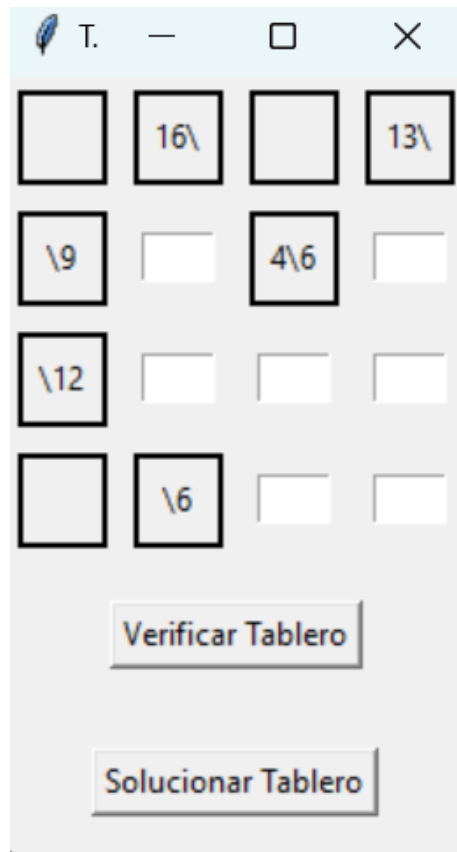
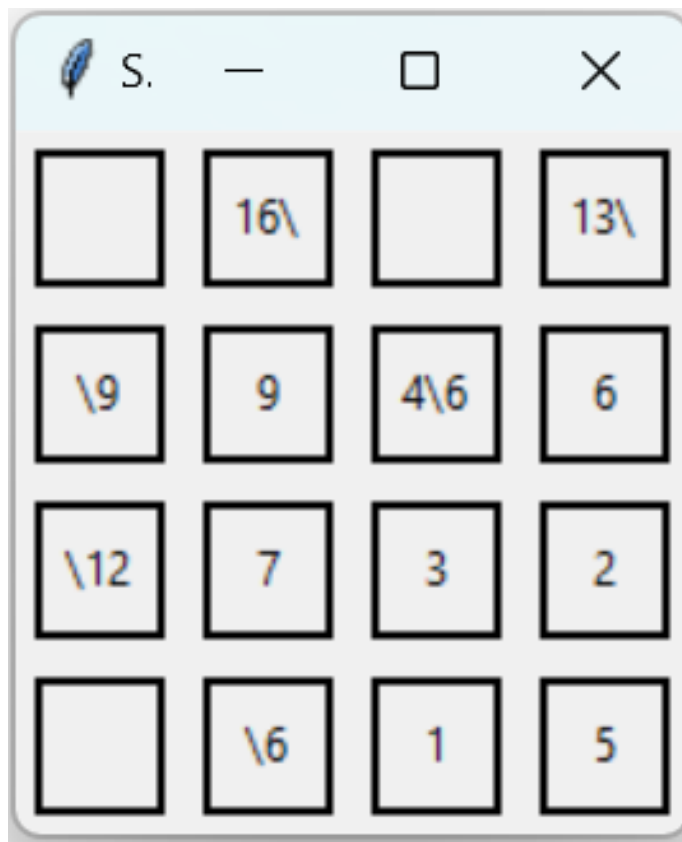


Figura 4: Opción Solucionar Tablero

La opción Tablero solucionado busca las soluciones para un tablero nxn de manera exhaustiva implementado algunas heurísticas para mejorar la eficiencia.

9.3. Tablero Solucionado



	16\		13\
\9	9	4\6	6
\12	7	3	2
	\6	1	5

Figura 5: Tablero Solucionado

Cuando el algoritmo encuentra una solución inmediatamente se muestra la solución en el tablero.

9.4. Kakkuro No Válido

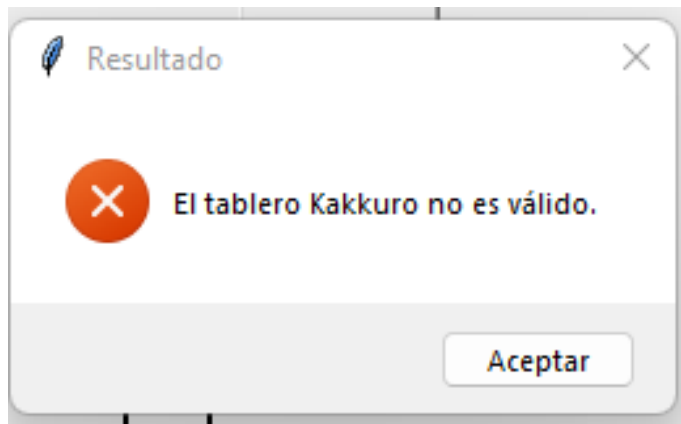


Figura 6: Kakkuro No Válido

Si el usuario intenta validar un tablero que no cumple con las reglas del juego, aparecerá la pantalla Kakkuro No Válido. En esta pantalla, se muestra un mensaje de error o advertencia, indicando que el tablero no es válido.

9.5. Kakkuro Válido

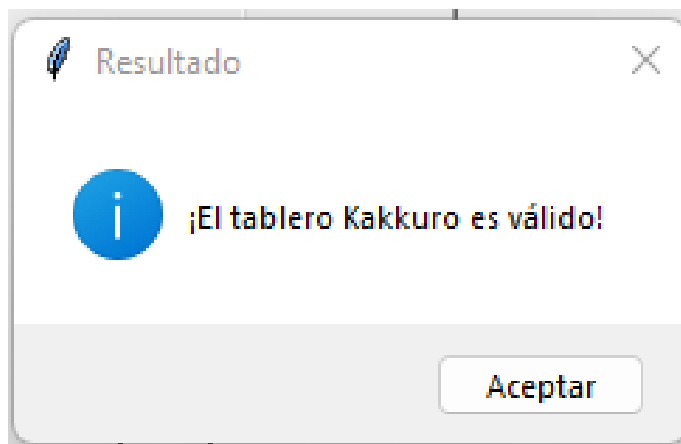


Figura 7: Kakkuro Válido

Cuando el usuario ha resuelto con éxito un tablero Kakkuro válido, se muestra la pantalla Kakkuro Válido. En esta pantalla, se puede mostrar un mensaje de felicitación y confirmación de que el tablero es válido.

10. Conclusiones

El algoritmo desarrollado para validar un tablero de juego Kakkuro es una herramienta eficaz para garantizar que todas las restricciones de suma en las celdas sombreadas se cumplan correctamente. A lo largo de este análisis y diseño, se han observado varios aspectos clave:

- El procedimiento ‘read_matrices’ permite la lectura eficiente de matrices desde un archivo de texto, lo que facilita la entrada de datos para su validación.
- El procedimiento ‘is_valid_kakkuro’ es esencial para verificar la validez global del tablero, y su complejidad depende del tamaño del tablero y la cantidad de celdas sombreadas.
- Los procedimientos ‘validate_sum_down’ y ‘validate_sum_right’ son fundamentales para la verificación de las restricciones de suma en las celdas sombreadas y tienen una complejidad lineal en función de la altura y la longitud de las sumas, respectivamente.
- El procedimiento ‘validate_entry’ proporciona una forma eficiente de validar las entradas de las celdas de la matriz.
- Se ha agregado una nueva funcionalidad que permite resolver automáticamente el tablero utilizando una heurística de menor dominio primero. Esta adición amplía la utilidad del algoritmo al proporcionar una capacidad de resolución para tableros completos basados en el enfoque de exploración de posibilidades con menor dominio primero.

En general, el algoritmo es capaz de manejar tableros de diferentes tamaños y con diversas restricciones de suma. Su complejidad computacional es adecuada para su aplicación en tableros de tamaño razonable y proporciona resultados rápidos y precisos, incluso al resolver el tablero automáticamente.

Además, el ejemplo proporcionado demuestra la capacidad del algoritmo para validar un tablero Kakkuro y encontrar una posible solución válida.

En resumen, el algoritmo de validación de tableros Kakkuro es una herramienta valiosa para garantizar la integridad y precisión de estos desafiantes juegos de lógica. Su diseño modular, su eficiencia en la validación y ahora su capacidad de resolución automática lo hacen adecuado para su uso en aplicaciones de resolución de rompecabezas y verificación de tableros Kakkuro.