

Generador de Kakuros

Eduard Torres Chaves, y Juan José Solano Quesada,

Abstract—This document discusses the study of the order of functions of pruning, backtracking and the functions in charge of permuting a list. It begins with the study of the large backtracking O and backtracking itself, then follows the study of the or the functions in charge of pruning and then the function in charge of performing permutations.

Index Terms—Backtracking, thread, poda.

I. INTRODUCCIÓN

Un kakuro es un juego matemático que consiste en un conjunto de casillas blancas y negras, en las cuales se deben rellenar las casillas blancas en el tablero con números del 1 al 9 tal que la suma de los valores en las casilla sea igual a la clave que se encuentra a la izquierda para las filas y por encima de las columnas.

Entre sus reglas se distinguen que no se puede repetir el valor de una casilla en la misma fila y columna hasta que haya un espacio negro y, por ende, la cantidad máxima de celdas consecutivas es 9.

Los problemas matemáticos que representan estos enigmas pueden resolverse utilizando técnicas de matriz matemática y backtracking.

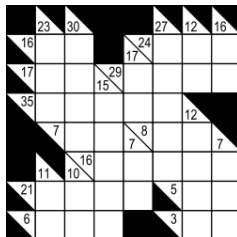


Fig. 1. Ejemplo de kakuro sin resolver

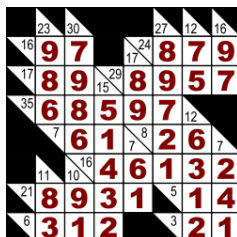


Fig. 2. Ejemplo de kakuro resuelto

II. ANÁLISIS

A. Permutaciones

En la figura 3, se muestra la función utilizada para calcular las permutaciones de una lista de tamaño n , al estudiar la función y las características del mismo, se llega a la conclusión de que posee un orden de $O(n!)$, ya que en la definición fundamental de una permutación es la igual al orden del factorial de n o $n!$, este representa el número de formas distintas de ordenar n objetos de manera diferente (sin repetir ordenes). El contenido de la lista se vuelve indiferente ya que, no importa el valor de los mismo a la hora de correr el algoritmo, pero si afecta el número de los mismo.

```
def swap(a, i, j):
    a[i], a[j] = a[j], a[i]

def permute(a, i, n):
    if i == n:
        print(a)
        return a
    for j in range(i, n+1):
        swap(a, i, j)
        permute(a, i+1, n)
        swap(a, i, j) # backtrack
```

Fig. 3. Función de permutaciones

B. Poda

Una de las partes más importantes del algoritmo de backtracking es su poda, pues reduce al mínimo la búsqueda de posibles soluciones al problema, lo que permite que la resolución del problema se realiza con mayor rapidez y eficiencia. En este caso la poda consiste en una estructura que contiene las posibles soluciones para que los números de una cantidad determinada de celdas sumen el valor de la clave determinada.

```
def obtenerPosibilidades(goalValue, emptyCells):
    possibilities = {
        3: { 2: [[1,2]]},
        4: { 2: [[1,3]]},
        5: { 2: [[1,4],[2,3]]},
        6: { 2: [[1,5],[2,4]], 3: [[1,2,3]]},
        7: { 2: [[1,6],[2,5],[3,4]], 3: [[1,2,4]]},
```

Fig. 4. Parte de la función de poda

En la figura 4 se puede ver que la cantidad de combinaciones crece conforme el valor de la clave se incrementa, no obstante, cuando la clave se encuentre entre 21 y 24 alcanza el valor máximo de posibles combinaciones, con diferentes valores de casillas consecutivas, las cuales varían en 23 posibles combinaciones. El límite que llega a alcanzar el valor de la clave es de 45, el cual posee una única combinación posible, que es la sumatoria del 1 al 9.

C. Experimentos

Utilizando distintas dimensiones y múltiples pruebas se obtuvo un resultado en el promedio que tarda resolviendo los kakuros de distintos tamaños y se obtuvo los siguientes resultados:

10x10	1.995
11x11	2.296
12x12	2.500
13x13	3.306
14x14	4.104
15x15	5.003
16x16	5.499
17x17	7.208
18x18	8.005
19x19	10.307
20x20	11.411

Fig. 8. Tamaños utilizados y tiempo en segundos obtenidos

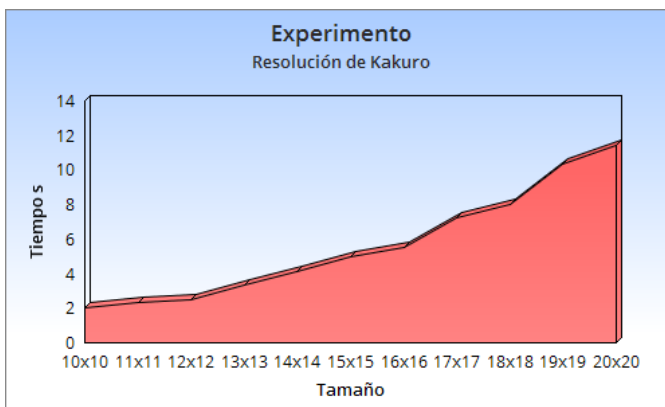


Fig. 9. Representación gráfica