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 las 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 tecnicas de matriz matemática y backtracking.

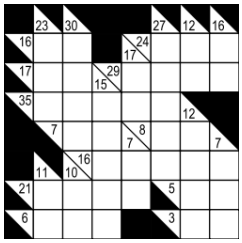

Fig. 1. Ejemplo de kakuro sin resolver

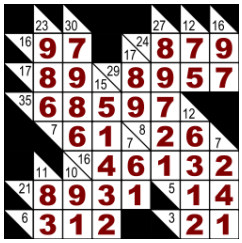

Fig. 2. Ejemplo de kakuro resuelto

E. Torres Chaves estudiante de Ingeniería en computación, Instituto Tecnológico de Costa Rica

J.J. Solano estudiante de Ingeniería en computación, Instituto Tecnológico de Costa Rica

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 caracteristicas del mismo, se llega a la comclusion de que posee un orden de $\mathcal{O}(n!)$, 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 numero 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

C. Backtracking

Al momento de analizar las funcioónes encargadas del backtraking, se debe de tomar en cuenta que el orden puede ser afectado por otras funciónes, como por ejemplo las encargadas de la poda, sin embargo, al ser estudiadas en otra sección, no se contemplara el efecto que tengan en el orden del backtracking.
El kakuro tiene la particularidad de solo tener, por numero, 9 casillas vaciás, por lo que el rango maximo de interaciones por numero a calcular es de 9. La función backtrack, figura 4, busca la solucion con un maximo de 11 veces por el parametro ”intentos” (maximo de veces que un mismo numero aparece en una lista de los posibles conjuntos) si no es que no ha encontrado una solución, al analizarla obtenemos un orden $O((n * 11 * k))$ donde k es la cantidad de veces que la recursividad necesiat evaluar por casilla, que a su vez se puede reducir a $O((n * k))$,se tiene que tener en cuenta que backtrack llama a la función meterEnMatriz, figura 5, por lo que es necesario el estudio de la función meterEnMatriz.
La función meterEnMatriz entra en un while que puede tener dos duraciones, en el mejor caso una duración de 9 (en el caso de ser una serie de 9 espacios en blanco sin ninguna intersección) o de 15 como maximo de interaciones antes de que se requiera otro grupo de solución (el cual se encarga la función backtrack), por lo que el orden de la función esta dada

por $O(15)$ al ser 15 el maximo entre (9,15).
Con lo anterior, podemos deducir que la O grande del backtracking, como un todo, seria de $O(n*k)*O(C)$, con C siendo la constante 15 de la O grande de la función meterEnMatriz, a lo que termina siendo $O(n*k)$

```
def backtrack(matrizSolu,tuplaActual,listaDeTuplas,intentos):
    if (listaDeTuplas==[]):
        return matrizSolu
    else:
        resultado=meterEnMatriz(matrizSolu,tuplaActual,obtenerPosibilidades(tuplaActual[3],tuplaActual[2]))
        if (resultado[0] and (intentos < 11)):
            backtrack(resultado[1],listaDeTuplas.pop(),listaDeTuplas,0)
        else:
            intentos+=1
            backtrack(resultado[1],tuplaActual,listaDeTuplas,intentos)
```

Fig. 4. Función de Backtrack

```
def meterEnMatriz(pmatrix,tupla,posibilidades):
    if(tupla[4]==1):#para las horizontales
        listaToques=sacarToquesHorizontales(tupla[0],tupla[1],tupla[2],pmatrix)#para ver si pega en algun lado con una vertical
        posibilidadMasFactible=escojerMejorPosibilidad(posibilidades,listaToques)
        posibilidadMasFactible=eliminarPresentes(posibilidadMasFactible,listaToques)
        contador=0
        random.shuffle(posibilidadMasFactible)
        metidos=0
        intentos=0
        while ((metidos!=len(posibilidadMasFactible))and intentos<15):
            if(pmatrix[tupla[0]][tupla[1]+contador]=="1"):
                pmatrix[tupla[0]][tupla[1]+contador]=posibilidadMasFactible[0]
                if (verificar(tupla[0],tupla[1]+contador,tupla[2],pmatrix)):
                    print("Hizo true")
                    metidos+=1
                    contador+=1
                else:
                    print("va a hacer cambio porque no es valido el num",posibilidadMasFactible[0]," en la posicion ",tupla[0]," ",tupla[1]+contador )
                    pmatrix[tupla[0]][tupla[1]+contador]="1"
                    random.shuffle(posibilidadMasFactible)
                    intentos+=1

            else:
                contador+=1
        if(intentos>=15):
            pmatrix=limpiarLoQueHizo(tupla,pmatrix,posibilidadMasFactible)
            return (False,pmatrix)
        else:
            return (True,pmatrix)
    else:
        listaToques=sacarToquesVerticales(tupla[0],tupla[1],tupla[2],pmatrix)
        posibilidadMasFactible=escojerMejorPosibilidad(posibilidades,listaToques)
        posibilidadMasFactible=eliminarPresentes(posibilidadMasFactible,listaToques)
        contador=0
        random.shuffle(posibilidadMasFactible)
        metidos=0
        intentos=0
        while ((metidos!=len(posibilidadMasFactible))and intentos<15):
            if(pmatrix[tupla[0]+contador][tupla[1]]=="1"):
                pmatrix[tupla[0]+contador][tupla[1]]=posibilidadMasFactible[0]
                #print(generarTupla(4)[tupla[1]+contador])
                if (verificar(tupla[0]+contador,tupla[1],tupla[2],pmatrix)):
                    print("Hizo true")
                    metidos+=1
                    contador+=1
                else:
                    print("va a hacer cambio porque no es valido el num",posibilidadMasFactible[0]," en la posicion ",tupla[0]," ",tupla[1]+contador )
                    pmatrix[tupla[0]+contador][tupla[1]]="1"
                    random.shuffle(posibilidadMasFactible)
                    intentos+=1

            else:
                contador+=1
        if(intentos>=15):
            pmatrix=limpiarLoQueHizo(tupla,pmatrix,posibilidadMasFactible)
            return (False,pmatrix)
        else:
            return (True,pmatrix)
```

Fig. 5. Función de meterEnMatriz

D. Generar un tablero