Rocío Guzmán Arroyo

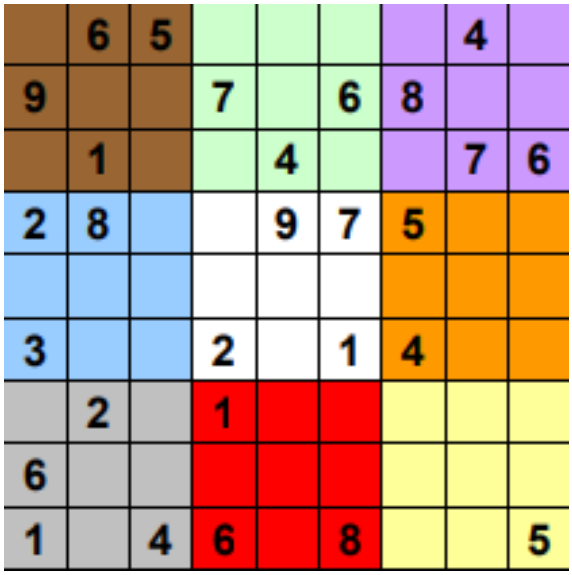
MEMORIA DE ADA

SUDOKU

ÍNDICE

INTRODUCCIÓN

Se nos pide implementar un código utilizando la técnica de vuelta atrás que resuelva un 'Sudoku'.´El juego consiste en completar una matriz de 9x9 la cual cuenta con subtableros de 3x3,los cuales están coloreados de varios colores en la siguiente imagen. Las reglas del juego nos permiten colocar números del 1 al 9 en los espacios en blanco sin que haya números repetidos en los sudtableros,ni en las filas ni en las columnas de la matriz principal.

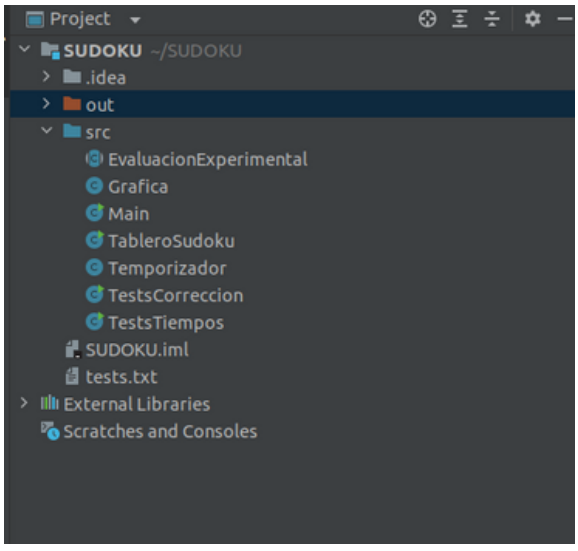


BACKTRACKING

La técnica va creando todas las posibles combinaciones de elementos, que da lugar aun árbol de exploración que nos permite obtener una solución.

CÓDIGOS SUMINISTRADOS

En el campus virtual se nos proporciona un pdf donde se explica la práctica,todos los códigos proporcionados y sus funciones y se nos indica los subprogramas que el alumno debe completar.A la derecha se puede ver todas las clases proporcionas,'TableroSuduko' es la que debemos completar,'TestCorrepción' se encarga de corregir el código que devuelve 'Test de resolverTodos correcto' , por último 'TestTiempos' devulve una gráfica de la que abraremos en el apartado 3.



PROGRAMAS

En este apartado estudiaremos el desarrollo de los subprogramas que se nos pide completar en la clase 'TableroSudoku' ,dicha clase se encarga de completar los 'Sudokus' que se proporcionan en el archivo test.txt.

estaEnFila

En primer lugar para satisfacer las reglas del juego tenemos esaEnFila que devuelve true o false si el número que vamos a introducir esta ya en la fila por lo tanto no podía introducirse.Para esto recorremos toda la fila fijando el parámetro que nos entra por valor y vemos si el número esta en la fila.

```
protected boolean estaEnFila(int fila, int valor) {
    boolean ok = false;
    for (int i = 0; i < FILAS; ++i) {
        if (celdas[fila][i] == valor) {
            ok = true;
        }
    }
    return ok;
}
```

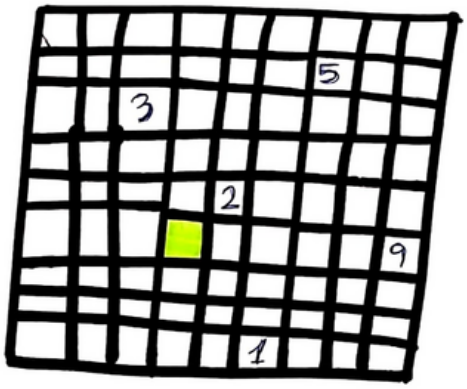
estaEnColumna

Aplicando la misma lógica que hemos aplicado en el método anterior comprobamos si el valor esta en la columna y devolvemos true o false en función del resultado.

```
protected boolean estaEnColumna(int columna, int valor) {
    boolean ok = false;
    for (int i = 0; i < COLUMNAS; ++i) {
        if (celdas[i][columna] == valor) {
            ok = true;
        }
    }
    return ok;
}
```

estaEnSubtablero

```java
protected boolean estaEnSubtablero(int fila, int columna, int valor) {
    boolean esta = false;
    int fila_comenzar = fila - (fila % 3);
    int columna_comenzar = columna - (columna % 3);
    for (int i = fila_comenzar; i < (fila_comenzar + 3) && !esta; i++) {
        for (int j = columna_comenzar; j < (columna_comenzar + 3) && !esta; j++) {
            if (celdas[i][j] == valor) {
                esta = true;
            }
        }
    }
    return esta;
}
```



Devuelve true o false si el valor que nos proporciona esta en el subtablero que le corresponde a la posición (fila,columna) que se nos proporciona, a la derecha tenemos un ejemplo.Este método es aplicable a todos los subtableros.Usamos el 3 porque sabemos que tiene dimensión 3x3 pero si tuviera otra dimensión sería igual unicamente cambiamos el 3 por otro valor.

No encontramos en las coordenadas
fila = 3
columna = 5
Si hacemos $3 - \overbrace{3 \bmod 3}^{0} = 3$
y si hacemos $5 - 5 \bmod 3 = 5 - 2 = 3$
Por lo tanto recorre del 3 a <6
y de igual modo en el segundo for.

sePuedePonerEn

Devuelve true o false si puede añadirse o no.Para poder añadirse deben aceptarse todas las condiciones anterjores.

```java
protected boolean sePuedePonerEn(int fila, int columna, int valor) {
    boolean ok = false;
    if(estaEnColumna(columna,valor) == false && estaEnFila(fila,valor) == false && estaEnSubtablero(fila,columna,valor) == false){
        ok = true;
    }
    return ok;
}
```

resolverTodos

```java
protected void resolverTodos(List<TableroSudoku> soluciones, int fila, int columna) {

    if(numeroDeLibres() == 0){
        soluciones.add( new TableroSudoku( uno: this));
    }else{
        // Tenemos que llenar el tablero hasta que este completo
        // Una vez encontramos un sitio donde podemos introducir un número del uno al 9 vamos comprobando y desaciendo

        if(estaLibre(fila,columna)){
            int num = 1;
            while(num <= 9){
                if (sePuedePonerEn(fila, columna, num)) {

                    celdas [fila][columna] = num;
                    // Añadimos un posible numero

                    int col;
                    int  fil;

                    if(columna == 8){
                        col = 0;
                        fil = fila +1;
```

```java
                    }else{
                        fil = fila;
                        col = columna +1;
                    }
                    resolverTodos(soluciones, fil, col);
                    celdas [fila][columna] = 0;

                }
                num++;

            }


        } else {

            if(columna == 8){
                columna = 0;
                fila++;
            }else{
                columna++;
            }
            resolverTodos(soluciones, fila, columna);

        }
```

En primer lugar:

Si ya no quedan celdas vacías hemos encontrado un tablero. Utilizamos el metodo "numerodelibres" si es 0 no queda más nada que rellenar:

```
protected boolean estaLibre(int fila, int columna) {
    return celdas[fila][columna] == 0;
}

// devuelve el n?mero de casillas libres en un sudoku.
2 usages
protected int numeroDeLibres() {
    int n=0;
    for (int f = 0; f < FILAS; f++)
        for (int c = 0; c < COLUMNAS; c++)
            if(estaLibre(f,c))
                n++;
    return n;
}
```

Como podemos ver

← MÉTODO QUE COMPRUEBA SI ESTA SIN RELLENAR

"SIN RELLENAR = 0"

← Las contamos recorriendo el tablero.

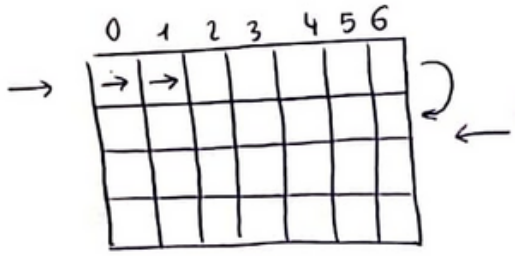
Luego según la posición en la que estemos, basandonos en la fila y columna que nos pasan, vemos si esta libre o tiene un número fijo asignado.

Como podemo introducir unicamente valores del 1 al 9 creo la variable num que inicializo a 1 y voy incrementando hasta 9.

Lo siguiente que tenemos que ver es si ese número se puede poner según las normas del sudoku, si no se puede paso al siguiente.

Si se puede lo añadimos.

Para el bracktraking primero consideramos el estudio de como nos movemos por la matriz.



¡ESTO ES UNA MATRIZ EJEMPLO, NO TIENE LA ESTRUCTURA DE UN SUDOKU PERO SE RECORRE IGUAL!

Si esto en la columna 6 la pongo a 0 para comenzar y paso a la siguiente fila.

En nuestro código si tenemos un número fijo o añadido saltamos al else y llamamos de forma recursiva a resolver todos.

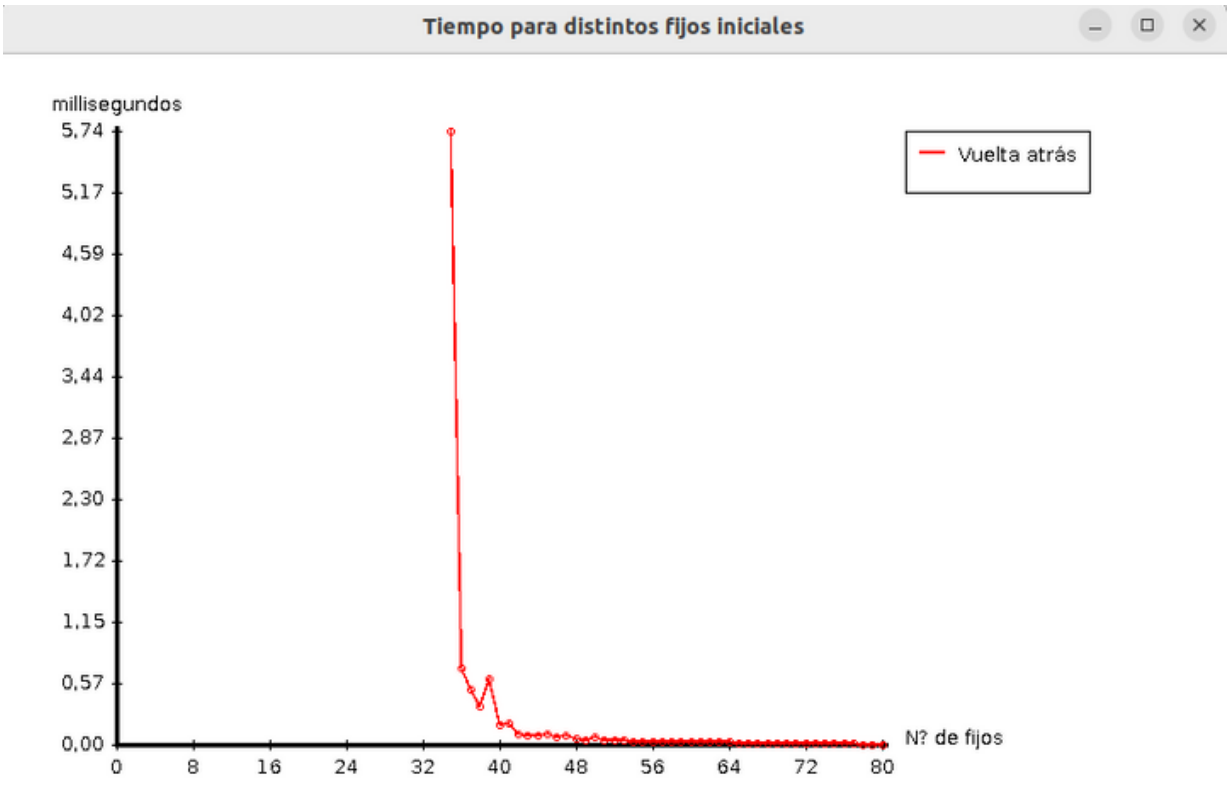
Si se puede añadir lo añadimos estudiamos el siguiente y despejamos la fila para añadir otro número, y así desglosamos un arbol de posibilidades.

Además una peculiaridad de este juego es que los sudokus tienen una única solución por lo tanto solo vamos a poder llegar a una solución final.

GRÁFICA

La gráfica siguiente es proporcionada por la clase 'TestTiempos' la cual reciba de la clase 'Temporizador' los tiempos que se van obteniendo y desarrolla la gráfica en función de las veces que el algoritmo de sobrescribe gracias al Backtraking hasta encontrar la solución correcta.

Es decir al principio el número de números fijos es menor debido a los errores y cambios que se van haciendo,a medida que se va rellenando el tablero la franja de error es menor y el número de números fijos es mayor se hace menos veces el backtraking porque ya hay menos posibilidades de equivocarse ya que hay menos posibles números que puedes poner.



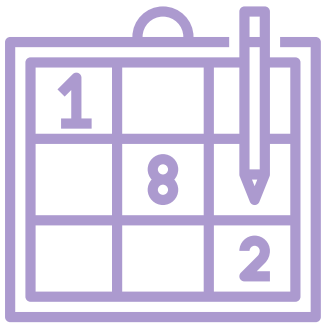
COMPLEJIDAD

En este apartado se estudiarán las complejidades temporales de los subprogramas que se nos pide completar.

En primer lugar tanto estaEnFila como estaEnColumna tiene complejidad O(n) en el infinito ya que ambas recorren con un for todos los elementos de una fila determinada o una columna.

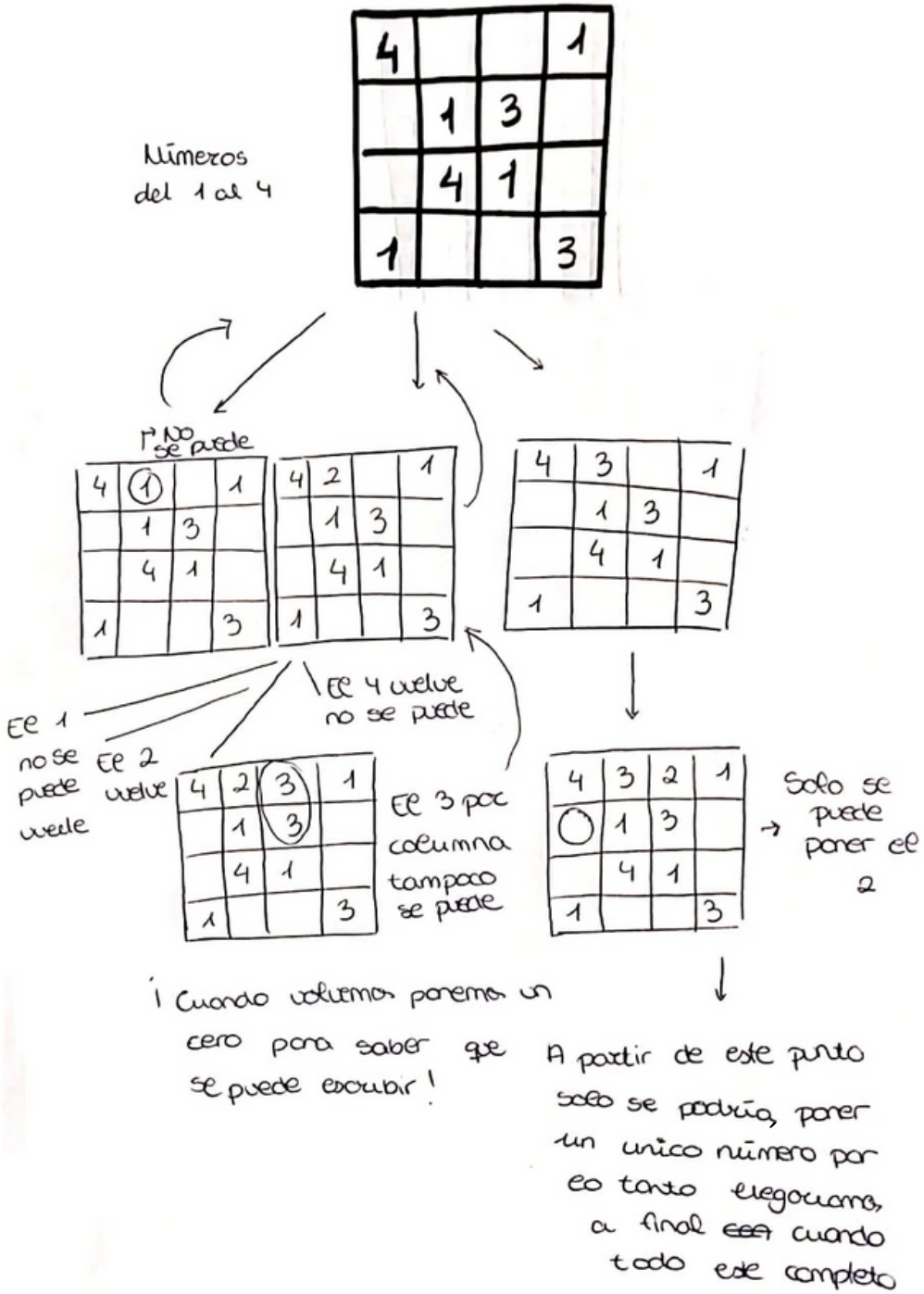
Por otro lado tenemos a estaEnSubtablero tiene complejidad $O(n^2)$ en el infinito,ya que utilizamos un doble for para recorrer un subtablero de dimensión 3x3.

Por último el subprograma 'resolverTodos' en el peor de los casos tenemos una complejidad de $O(9^{n^2})$ ya que tenemos 9 opciones, como es un problema de combinatoria pues contamos con el número de opciones a rellenar elevado al doble array que es la dimensión del tablero



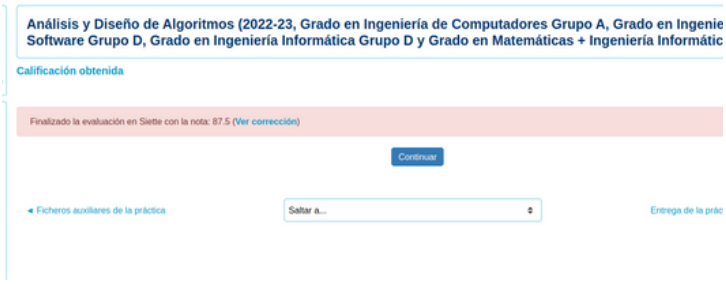
DESARROLLO

En este apartado se desarrollará un árbol auxiliar de un tablero totalmente inventado, en los Sudokus originales hay que poner 17 variables fijas,y la matriz es de 9x9 mientras menos números fijos mayor será la dificultad.
En este caso experimental se elegirá un tablero de 4x4 con 8 números fijos.



PROBLEMAS

La primera vez que introduje el código en el corrector 'Siette' la nota que salio fue un 8.75, esto se debía a que mi subprograma 'estaEnSubtablero' era demasiado extenso así que tenía que plantear otro código que simplificara todas las opciones en una.



Como primera idea cree una subprograma que plantease todos los posibles casos, como puede verse en el ejemplo siguiente.

```
//TABLERO 2

if( fila < 3 && columna < 6 && columna > 2){
        for(int i = 0;i<3;i++){
                for (int j = 3; j<6;j++){
                        if(i != fila && j!= columna){
                                if(celdas[i][j] == valor){
                                        ok = true;
                                }
                        }
                }
        }
}

//TABLERO 3

if( fila < 3 && columna < 9 && columna > 5){
        for(int i = 0;i<3;i++){
                for (int j = 6; j<9;j++){
                        if(i != fila && j!= columna){
                                if(celdas[i][j] == valor){
                                        ok = true;
                                }
                        }
                }
        }
}
```