MEMORIA PRACTICA PROGRAMACION 3

CURSO 2005-2006

ALUMNO VICTOR M. DEL CANTO GODINO

CENTRO ASOCIADO PALENCIA

INDICE :

1.1.1TIPO DE PROBLEMA

El Sudoku es un juego popular que consiste en rellenar una matriz de 9x9 casillas con digitos del 1 al 9. Cada matriz esta parcialmente vacia, salvo algunas valores ya rellenos de manera que:
Cada digito aparece una y solo una vez en cada fila
Cda digito aparece una y solo una vez en cada columna
Cada digito aparece una y solo una vez en cada uno de los nueve cuadrados o cajas de 3x3 en los que se puede subdividr la matriz inicial.
Ademas y es algo que no parece tenerse muy claro es que cada Sudoku es unico, es decir lo que se plantea debe tener una unica solucion y si tiene varias es incorrecto.

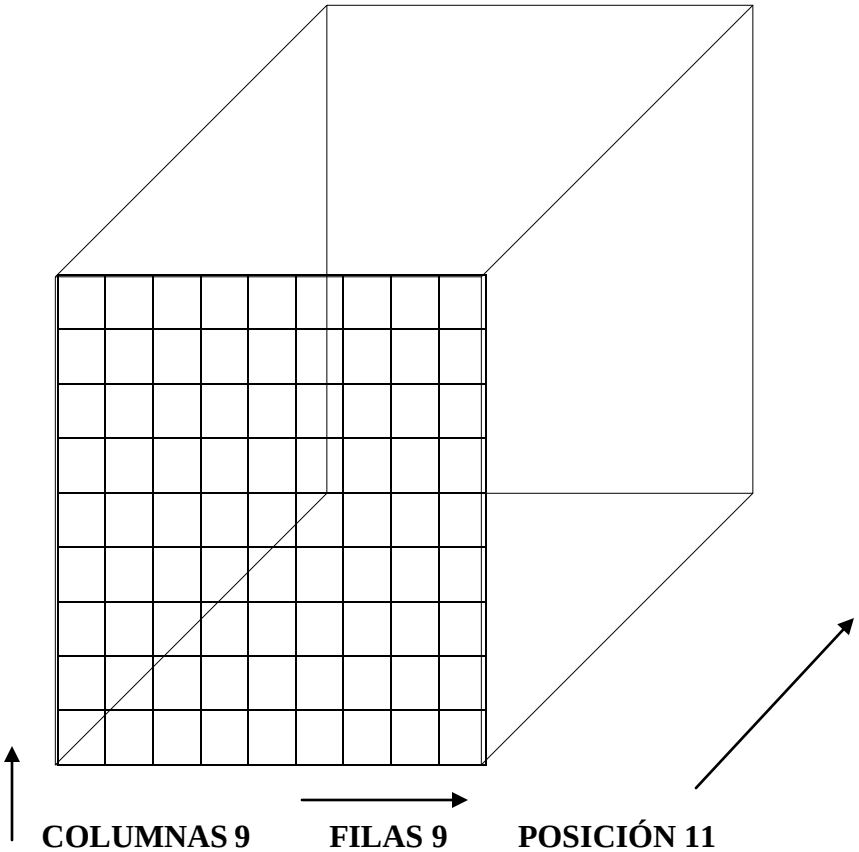
1.1.2- ESQUEMA ALGORITMICO UTILIZADO

Puesto que consiste en elegir un valor del 1 al 9 en cada celda y ademas tenemos que restringirnos a las reglas del Sudoku la opcion que parece mas adecuada es escoger un algoritmo de exploracion de grafos con vuelta atrás. Esto es cogemos un numero,lo ponemos en una celda, otro en otra,etc.. cumpliendo que el primer numero que colocamos en la primera celda invalida que ese numero se coloque en cualquier celda de la fila, columna y caja 3x3 (en adelante caja) a las que pertenece la celda hallada.
Ademas hay que tener en cuenta que la solucion esta parcialmente hallada, es decir tenemos soluciones en celdas que restringiran a sus filas, columnas y cajas asociadas. Esto nos servira como poda.

Para la vuelta atrás haremos:
Busco en la matriz de candidatos la celda no resuelta con menor numero de candidatos. En caso de igualdad la primera .Me devolvera (fila, columna, digito-el primero de los candidatos-

Si me devuelve 0,0,0. Caso trivial. No quedan candidatos. Celdas resueltas. Sudoku resuelto
Si me devuelve 0,0,10 Caso trivial. No quedan candidatos. Celdas sin resolver. Vuelta atrás
Caso no trivial :
Si no asumo el candidato como cierto y hago los ajustes en el Sudoku
Si ajustes incorrectos vuelvo a elegir otro candidato mayor en la misma celda
Si ajustes correctos llamada a si misma pasando nueva matriz

Para los ajustes creo una matriz de candidatos de tres dimensiones



COLUMNAS 9 FILAS 9 POSICIÓN 11

Evidentemente la filas y columnas se corresponden con las del Sudoku y las posiciones son

0- numero de candidatos posibles en esa celda. Es la suma de 1 hasta 9

1-9 cada uno de los digitos posibles en esa celda. El valor de posición [i] sera 0 o 1 dependiendo de si el digito i es o no posible en la celda

10 Se pone un 1 si esta celda ya ha sido resuelta para no trabajar mas con ella

Se trata de poner 0 o 1 en las posiciones de las celdas según sea posible el candidato en esa celda. Se usa la posición 0 para encontrar los solos , y la posición 10 para marcarlos como ya tratados.

A partir de ahí se buscan solos se ajustan los candidatos de las celdas afectadas y se devuelve la matriz una vez marcada la celda como hecha.

1.2-ESTRATEGIAS LOCALES

En principio podemos considerar que el algoritmo general colocara en la celda un valor entre 1 y 9, y a continuacion verifica la exactitud, es decir que ese valor no esta ni en la fila ni en la columna ni en la caja adjunta a la celda.Si es correcto vamos a l a siguiente celda y repetimos el proceso. Si no ponemos el siguiente numero en la celda primera y volvemos a verificar.

Asi hasta que lleguemos a la ultima celda. Fijemonos por un momento en el cuadrado vacio, que no es un Sudoku . Tenemos 81 celda con 9 candidatos para cada una. Eso nos da unas 9^81 combinaciones posibles, lo cual es intratable. El Sudoku tiene una serie de celdas ya resueltas pero

como se vera en el apartado anterior siguen siendo muchas combinaciones. Como tenemos una serie de restricciones aprovechemoslas. Si un digito solo puede aparecer una vez en la fila, columna y caja adjunta a su celda, no tenemos porque tener ese digito como candidato posible en el resto de celdas de esa fila, columna y caja.

La primera poda que podemos hacer es establecer una matriz de tres dimensiones en la cual las dos primera representa el sudoku, es decir filas x columnas y la tercera dimension es la de los candidatos validos en cada celda . Asi una celda resuelta tiene 1 candidato, el resuelto, pero ese queda invalidado para las demas celdas de la fila, columa y caja adjuntas.

La segunda poda que podemos hacer es evidente. Es posible que al restringirse los candidatos por aquellas celdas ya resueltas, existan celdas que solo tengan un candidato o que exista un unico digito para toda una fila, o columna o caja en una celda . Evidentemente esa celda queda resuelta. Y esto lleva nuevas eliminaciones de candidatos. Existen otras estrategias como las parejas solas, parejas escondidas, filas o columnas restringidas a cajas o al reves, trios desnudos. Realmente el Sudoku es un puzle logico. He resuelto unos cuantos y solo una vez he tenido que usar la prueba y error para resolver la seis ultimas celdas de uno en un Sudoku Harakiri (combinacion de 8 Sudokus).

El uso como poda de los solos y solos escondidos podria ser discutible en un ejercicio como este que trata de implementar un algoritmo concreto, pero me ha parecido razonable puesto que en los candidatos solos no hay combinacion posible, es la solucion. Los solos escondidos es mas discutible pero se pueden eliminar del codigo poniendo las adecuadas lineas como comentario (//), que ya señalare en el codigo.

1.3-COSTE TEORICO

Como he señalado anteriormente partimos de que un cuadrado vacio tiene 9^81 combinaciones posibles. Con uno resuelto tenemos 8^20 x9^60

Inicial	con 1 numero
9 9 9 9 9 9 9 9 9	x 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9	8 8 8 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9	8 8 8 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9	8 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9	8 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9	8 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9	8 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9	8 9 9 9 9 9 9 9 9

9 9 9 9 9 9 9 9 9 8 9 9 9 9 9 9 9 9 9

A partir de aqui la cosa cambia. El nmumero de combinaciones no depende solo de los resueltos es decir de los datos sino tambien de su colocacion . Por ejemplo colocando el segundo resuelto

7^7 x 8^24 x 9^48 7^2 x 8^36 x 9^41

x 7 7 8 8 8 8 8 8 x 8 8 7 8 8 8 8 8
7 x 7 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9
7 7 7 9 9 9 9 9 9 8 8 8 8 9 9 9 9 9
8 8 9 9 9 9 9 9 9 7 8 8 x 8 8 8 8 8
8 8 9 9 9 9 9 9 9 8 9 9 8 8 8 9 9 9
8 8 9 9 9 9 9 9 9 8 9 9 8 8 8 9 9 9
8 8 9 9 9 9 9 9 9 8 9 9 8 9 9 9 9 9
8 8 9 9 9 9 9 9 9 8 9 9 8 9 9 9 9 9
8 8 9 9 9 9 9 9 9 8 9 9 8 9 9 9 9 9

El de la derecha tiene mas combinaciones

Otro caso curioso que se da es la coincidencia de digitos en los resueltos. Si cogemos el 2º cuadrado con dos resultados y suponemos que son iguales (pueden serlo porque no coinciden en fila columna o caja) tenemos que

x 8 8 8 8 8 8 8 8
8 8 8 8 9 9 9 9 9
8 8 8 8 9 9 9 9 9
8 8 8 x 8 8 8 8 8
8 9 9 8 8 8 9 9 9
8 9 9 8 8 8 9 9 9
8 9 9 8 9 9 9 9 9
8 9 9 8 9 9 9 9 9
8 9 9 8 9 9 9 9 9

tiene 8^38 x 9^41 que son mas que el anterior.

De esta forma lo unico que yo puedo decir con mis conocimentos es que la cota maxima para cualquier Sudoku esta en 9^81 combinaciones posibles.

No puedo hablar de calcular una complejidad en funcion de los datos porque no depende solo de los datos que nos dan sino tambien de su posicion y de si son iguales o no.
Con un Sudoku cualquiera el hecho de tener n celdas resueltas no implica nada. Ese mismo Sudoku con otras n celdas resueltas diferentes cambia el coste totalmente.
Sin embargo si no tenemos en cuenta ningun tipo de poda, teniendo como posibles en cada celda sin resolver los 9 digitos, podemos calcular una cota maxima. 9^(81-n) siendo n el numero de casillas resueltas.
Entrare mas profundamente en esta cuestion en el apartado posterior.

2.EJEMPLO DE EJECUCION

Se toma como datos de origen el sudoku proporcionado en el enunciado debidamente corregido (en la fila 5 columna 9 se corresponde un 4, no un 8).El resultado es mediante traza:
AJUSTE DE CANDIDATOS SOLOS

El candidato 1 era unico en la fila 5 columna 5

El candidato 6 era unico en la fila 3 columna 5

El candidato 4 era unico en la fila 7 columna 5

AJUSTE POR ALGORITMO VUELTA ATRAS

Candidato 1 seleccionado en fila 2 columna 4 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Candidato 4 seleccionado en fila 2 columna 4 con algoritmo de vuelta atras.

El candidato 4 era unico en la fila 2 columna 4

El candidato 9 era unico en la fila 6 columna 4

El candidato 4 era unico en la fila 4 columna 6

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 0 0 0 0 5 0 0 2 0

2 0 0 3 4 9 0 8 0 0

3 0 4 0 0 6 7 0 0 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 0 3 0 2 0 0

9 0 7 0 0 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 1 seleccionado en fila 1 columna 4 con algoritmo de vuelta atras.

El candidato 1 era unico en la fila 1 columna 4

El candidato 3 era unico en la fila 1 columna 6

El candidato 2 era unico en la fila 2 columna 6

El candidato 8 era unico en la fila 3 columna 4

El candidato 2 era unico en la fila 3 columna 3

El candidato 5 era unico en la fila 3 columna 1

El candidato 1 era unico en la fila 3 columna 7

El candidato 3 era unico en la fila 3 columna 8

El candidato 6 era unico en la fila 9 columna 4

El candidato 7 era unico en la fila 8 columna 4

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 0 0 0 1 5 3 0 2 0

2 0 0 3 4 9 2 8 0 0

3 5 4 2 8 6 7 1 3 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 7 3 0 2 0 0

9 0 7 0 6 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 6 seleccionado en fila 1 columna 9 con algoritmo de vuelta atras.

El candidato 6 era unico en la fila 1 columna 9

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 0 0 0 1 5 3 0 2 6

2 0 0 3 4 9 2 8 0 0

3 5 4 2 8 6 7 1 3 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 7 3 0 2 0 0

9 0 7 0 6 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 7 seleccionado en fila 1 columna 1 con algoritmo de vuelta atras.

El candidato 7 era unico en la fila 1 columna 1

El candidato 4 era unico en la fila 1 columna 7

El candidato 6 era unico en la fila 2 columna 1

El candidato 1 era unico en la fila 2 columna 2

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 7 0 0 1 5 3 4 2 6

2 6 1 3 4 9 2 8 0 0

3 5 4 2 8 6 7 1 3 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 7 3 0 2 0 0

9 0 7 0 6 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 8 seleccionado en fila 1 columna 2 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Candidato 9 seleccionado en fila 1 columna 2 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Error en los calculos o el SUDOKU. Vuelta atras a la situacion anterior resuelta

Candidato 8 seleccionado en fila 1 columna 1 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Error en los calculos o el SUDOKU. Vuelta atras a la situacion anterior resuelta

Candidato 7 seleccionado en fila 1 columna 9 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Error en los calculos o el SUDOKU. Vuelta atras a la situacion anterior resuelta

Candidato 8 seleccionado en fila 1 columna 4 con algoritmo de vuelta atras.

El candidato 8 era unico en la fila 1 columna 4

El candidato 1 era unico en la fila 3 columna 4

El candidato 3 era unico en la fila 1 columna 6

El candidato 2 era unico en la fila 2 columna 6

El candidato 5 era unico en la fila 3 columna 7

El candidato 3 era unico en la fila 3 columna 8

El candidato 6 era unico en la fila 9 columna 4

El candidato 7 era unico en la fila 8 columna 4

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 0 0 0 8 5 3 0 2 0

2 0 0 3 4 9 2 8 0 0

3 0 4 0 1 6 7 5 3 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 7 3 0 2 0 0

9 0 7 0 6 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 6 seleccionado en fila 1 columna 1 con algoritmo de vuelta atras.

El candidato 6 era unico en la fila 1 columna 1

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 6 0 0 8 5 3 0 2 0

2 0 0 3 4 9 2 8 0 0

3 0 4 0 1 6 7 5 3 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 7 3 0 2 0 0

9 0 7 0 6 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 1 seleccionado en fila 1 columna 2 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Candidato 9 seleccionado en fila 1 columna 2 con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.

Error en los calculos o el SUDOKU. Vuelta atras a la situacion anterior resuelta

Candidato 7 seleccionado en fila 1 columna 1 con algoritmo de vuelta atras.

El candidato 7 era unico en la fila 1 columna 1

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 7 0 0 8 5 3 0 2 0

2 0 0 3 4 9 2 8 0 0

3 0 4 0 1 6 7 5 3 9

4 0 0 1 5 2 4 0 0 0

5 9 0 0 3 1 8 0 0 4

6 0 0 0 9 7 6 3 0 0

7 1 0 0 2 4 0 0 6 0

8 0 0 5 7 3 0 2 0 0

9 0 7 0 6 8 0 0 0 0

Ahora se procede a recalcular los candidatos .

Candidato 6 seleccionado en fila 1 columna 3 con algoritmo de vuelta atras.

El candidato 6 era unico en la fila 1 columna 3

El candidato 1 era unico en la fila 1 columna 9

El candidato 9 era unico en la fila 1 columna 2

El candidato 4 era unico en la fila 1 columna 7

El candidato 5 era unico en la fila 2 columna 1

El candidato 1 era unico en la fila 2 columna 2

El candidato 7 era unico en la fila 2 columna 8

El candidato 6 era unico en la fila 2 columna 9

El candidato 5 era unico en la fila 5 columna 8

El candidato 8 era unico en la fila 8 columna 9

El candidato 7 era unico en la fila 4 columna 9

El candidato 6 era unico en la fila 5 columna 7

El candidato 9 era unico en la fila 4 columna 7

El candidato 8 era unico en la fila 4 columna 8

El candidato 2 era unico en la fila 5 columna 2

El candidato 7 era unico en la fila 5 columna 3

El candidato 1 era unico en la fila 6 columna 8

El candidato 2 era unico en la fila 6 columna 9

El candidato 7 era unico en la fila 7 columna 7

El candidato 6 era unico en la fila 8 columna 2

El candidato 3 era unico en la fila 4 columna 2

El candidato 6 era unico en la fila 4 columna 1

El candidato 8 era unico en la fila 7 columna 2

El candidato 5 era unico en la fila 6 columna 2

El candidato 9 era unico en la fila 7 columna 3

El candidato 5 era unico en la fila 7 columna 6

El candidato 3 era unico en la fila 7 columna 9

El candidato 4 era unico en la fila 8 columna 1

El candidato 8 era unico en la fila 6 columna 1

El candidato 2 era unico en la fila 3 columna 1

El candidato 8 era unico en la fila 3 columna 3

El candidato 4 era unico en la fila 6 columna 3

El candidato 9 era unico en la fila 8 columna 8

El candidato 1 era unico en la fila 8 columna 6

El candidato 3 era unico en la fila 9 columna 1

El candidato 2 era unico en la fila 9 columna 3

El candidato 9 era unico en la fila 9 columna 6

El candidato 1 era unico en la fila 9 columna 7

El candidato 4 era unico en la fila 9 columna 8

El candidato 5 era unico en la fila 9 columna 9

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 7 9 6 8 5 3 4 2 1

2 5 1 3 4 9 2 8 7 6

3 2 4 8 1 6 7 5 3 9

4 6 3 1 5 2 4 9 8 7

5 9 2 7 3 1 8 6 5 4

6 8 5 4 9 7 6 3 1 2

7 1 8 9 2 4 5 7 6 3

8 4 6 5 7 3 1 2 9 8

9 3 7 2 6 8 9 1 4 5

Ahora se procede a recalcular los candidatos .

SUDOKU RESUELTO

F

I

L COLUMNAS

A

S 1 2 3 4 5 6 7 8 9

1 7 9 6 8 5 3 4 2 1

2 5 1 3 4 9 2 8 7 6

3 2 4 8 1 6 7 5 3 9

4 6 3 1 5 2 4 9 8 7

5 9 2 7 3 1 8 6 5 4

6 8 5 4 9 7 6 3 1 2

7 1 8 9 2 4 5 7 6 3

8 4 6 5 7 3 1 2 9 8

9 3 7 2 6 8 9 1 4 5

3.ESTUDIO DEL COSTE

Como decia anteriormente podemos hallar una cota maxima unicamente para el caso peor. Para el promedio entran en juego otros factores que cite anteriormente (colocacion, coincidencia de digitos,..) lo que hace el calculo casi imposible.
En el caso peor tenemos 81 casillas con n celdas resueltas. Evidentemente si hacemos un algoritmo de fuerza bruta o vuelta atrás sin refinamientos, la cota se halla en 9^(81-n) posible combinaciones.
La primera poda es la eleccion de candidatos validos según sudoku .La cota asi obtenida es inferior y en mucho a la anterior ,puesto que no todas las celdas no resueltas tienen ya 9 candidatos, sino menos, algunas incluso solo uno. Esto es asi puesto que las celdas resueltas nos eliminan candidatos en las celdas de la fila, columna y caja adjunta.
La segunda poda que es la resolucion de candidatos solos surge como consecuencia de la primera. Si en una celda solo es posible la combinacion en la que aparezca el unico candidato, podemo resolver dicho candidato, y lo que es mas importante, tener la posibilidad de que esa resolucion lleve a nuevas eliminaciones de candidatos, reduciendo aun mas el numero de combinaciones posibles.
En cuanto a la cota maxima, no solo es diferente para cada Sudoku, sino que para uno cualquier, distintas formas de presentarlo (para su resolucion), conlleva distintas combinaciones
En el codigo de la practica implemente un metodo logico añadido, el de los solos escondidos. Aquellos digitos que son unicos en una fila, columna o caja , aunque en su celda esten acompañados por mas candidatos, son la elccion correcta para esa celda. Este metodo lo he dejado inactivo porque junto con el de los solos me resolvia los Sudokus totalmente, incluso los que por la red se comentan mas dificiles, sin entrar en el algoritmo vuelta atrás. Se basan en busquedas en vectores y son recursivos.

4.LISTADO COMPLETO DEL CODIGO

```
import java.util.*;
import java.io.*;


public class Sudoku extends OperacionesSudoku{

//clase principal tiene el tratamiento de ficheros y las llamadas a metodos para
resolver el sudoku


      public static void main (String[] args) {

      boolean traza,test,ayuda;
      traza=false;test=false;ayuda=false;
      String fichero;
      fichero=null;
      String[]argumento;
      try{
            if (args.length==0) {
                  fichero=EntradaEstandar();//se introduce el argumento por la
entrada estandar
            argumento=args;//se pasa argumento
            }else{
```

```
            argumento=args;//se pasa argumento
            if (argumento[0].charAt(0)=='-') {//busqueda de  opciones en sintaxis
                  switch (argumento[0].charAt(1)) {
                  case 't':
                                    test=true;
                                          if (args.length==1){
                                          fichero=EntradaEstandar();//se introduce
el argumento por la entrada estandar
                                          }else{fichero=argumento[1]; }
                                    break;
                  case 'a':
                                    traza=true;
                                          if (args.length==1){
                                          fichero=EntradaEstandar();//se introduce
el argumento por la entrada estandar
                                          }else{fichero=argumento[1]; }
                                    break;
                  case 'h':
                                    ayuda=true;
                                    break;
                  default :
                                    System.out.println(" Parametro Incorrecto ");

                                    System.out.println();
                                    return;
                  }//fin del switch
            } else {
            fichero=argumento[0]; //entrada sin opciones
            }
            if (fichero==null) {
                  fichero=EntradaEstandar();//se introduce el argumento por la
entrada estandar
            }
            }
      } catch (IOException e) {System.err.println("error de entrada/salida
");return;}
            ResolverSudoku (test,traza,ayuda,fichero);
            return;
      } //fin del matodo principal




      public static String EntradaEstandar ()throws IOException  {
      try {
            BufferedReader entrada = new BufferedReader (new InputStreamReader
(System.in));// Lee de la entrada estandar
            PrintWriter salida = new PrintWriter(new BufferedWriter (new FileWriter
("xyz.txt")));// Abre fichero para copiar de la entrada estandar
            int x,aux;
            String s=" ";
            System.out.println("Si no ha redirigido el fichero teclee el sudoku a
tratar y ponga fin sin espacios");
            while ((s=entrada.readLine())!=null) {
                  salida.println(s);
            }
```

```
            salida.close();
      } catch (IOException e) { }

                  String argumento;
                  argumento="xyz.txt";
            return argumento;
      }//fin del metodo EntradaEstandar




      public static int[][] LecturaFichero(String fichero) throws
FileNotFoundException {
      int[][]MatrizInicial= new int[9][9];
      int[][]MatrizAuxiliar= new int[10][10];
      BufferedReader in = new BufferedReader(new FileReader(fichero));
      String str,celda;
      int i,j,valor,vacio;//i son filas ,j son columnas y valor es el de la
casilla, vacio sirve para evitar errores de indice de array cuando leemos las
lineas del fichero y alguna esta en blanco
      i=0;
      try {
            while ((str = in.readLine()) != null) {
                  vacio=str.length();//numero de caracteres de la linea
                  if (vacio==0) continue;//saltamos a la siguiente linea
                  char cadena[] = str.toCharArray();
                  if ((cadena[0]=='#')||(cadena[0]==' ')) {
                        continue ;
                  }else {
                        i++;
                        j=0;
                  }
                  while (j< cadena.length) {
                        if (cadena[j]=='*') {
                              valor=0;
                        } else {
                        celda=(cadena[j]+" ");
                        valor=Integer.parseInt(celda.trim());
                        }
                              j++;
                              MatrizAuxiliar[i][j]=valor;
                  }// fin del while anidado
            }// fin del while principal
            in.close();
            i=0;j=0;
            for (i=1;i<=9;i++) {
                  for (j=1;j<=9;j++) {
                  MatrizInicial[i-1][j-1]=MatrizAuxiliar[i][j];
                  }
            }
      } catch (IOException e) {System.err.println("error de entrada/salida ");}
      return MatrizInicial;
   } // fin del metodo LecturaFichero
```

```
  public static void ResolverSudoku (boolean test,boolean traza,boolean
ayuda,String fichero) {
      int[][] Matriz=new int[9][9];
      int[][][]MatrizAux = new int[9][9][11];
      boolean correcto;
      int fila,columna,hora,minuto,segundo;
      Date Costeinicio=new Date();//sirve para calcular el tiempo que tarda en
encontrar la solucion
      long inicio,actual;
            if (ayuda==true) {
            ImprimirAyuda();
            return;
            }

      try{
      Matriz=LecturaFichero(fichero);
      } catch (FileNotFoundException e) {
      System.err.println("fichero no existe o mal introducido ");//excepcion el
fichero no existe
      return;
      }catch (NumberFormatException e)  {
      System.err.println("datos fichero incorrectos ");
      return;
      }
      correcto=ComprobarSudoku(Matriz);
            if (correcto==false) {
            System.out.println(" Los datos del fichero "+fichero+" NO son correctos
segun las reglas de SUDOKU ");
            return;
            }
            if (test==true) {
            System.out.println(" Los datos del fichero "+fichero+" son correctos
segun las reglas de SUDOKU ");
            return;
            }
      inicio=Costeinicio.getTime();
      MatrizAux=EstablecerCandidatos(Matriz);
            if (traza==true){System.out.println(" AJUSTE DE CANDIDATOS SOLOS ");
      }
      MatrizAux=AjusteCandidatosSolos(MatrizAux,traza);
      //MatrizAux=AjusteSolosEscondidos(MatrizAux,traza);// QUEDA AQUI POR SI SE
QUIERE PROBAR SOLOS ESCONDIDOS(QUITAR // INICIAL)
            if (traza==true){System.out.println(" AJUSTE POR ALGORITMO VUELTA ATRAS
");   }
      MatrizAux=VueltaAtras(MatrizAux,0,0,0,traza);
      SalidaDatos(MatrizAux);
      Date Costefin=new Date();
      actual=Costefin.getTime();
      //System.out.println(" el tiempo tardado en resolver es de  "+(actual-
inicio)+"  ms. ");//muestra el tiempo empleado
      return;
  }//fin del metodo ResolverSudoku

  public static void ImprimirAyuda () {
 System.out.println();
 System.out.println("La practica se invoca usando la siguiente sintaxis:");
 System.out.println();
```

```
 System.out.println("sudoku [-t][-a][-h] [fichero] ");
 System.out.println();
 System.out.println("Opciones: ");
 System.out.println();
 System.out.println("-t:  Realiza un test de correccion a la matriz de entrada. Si
es incompleta o");
 System.out.println("     incorrecta segun las reglas del juego devuelve 1 en caso
contrario");
 System.out.println("     devuelve 0. Sirve para comprobar errores sintacticos en
la entrada.");
 System.out.println();
 System.out.println("-a:  Modo traza. Muestra toda la secuencia de valores que se
van incorporando a");
 System.out.println("     la matriz de entrada hasta completarla correctamente. Hay
que tratar que");
 System.out.println("     sea lo mas ilustrativa posible sobre el proceso del
problema por parte del");
 System.out.println("     algoritmo.");
 System.out.println( );
 System.out.println("-h:  Modo ayuda. Muestra la sintaxis y los creditos.  ");
 System.out.println( );
 System.out.println("Si no tiene argumentos, el programa muestra la matriz inicial,
y la matriz final");
 System.out.println("resuelta.");
 System.out.println( );
 System.out.println(" Practica resuelta por Victor M. del Canto Godino ");
 System.out.println( );
 System.out.println(" Alumno del Centro Asociado De Palencia. UNED Curso 2005-2006"
);
 System.out.println( );

 return;
  }//fin del metodo ImprimirAyuda


}// fin de la clase Sudoku
```

```
class OperacionesSudoku extends Util{

//clase que realiza operaciones especificas de sudoku


        static int[][] MatrizOrigen =new int[9][9];


        public static boolean ComprobarSudoku(int[][] MatrizOrigen) {

                int aux,aux1,aux2,aux3,aux4,cont;

                int[][] RangoCaja = new int[2][3];

                int[] VectorOrigen = new int[11];

                for (aux1=0;aux1<=8;aux1++) {
```

```
for (cont=0;cont<=8;cont++) {
VectorOrigen[cont+1]=MatrizOrigen[aux1][cont];//COMPRUEBA SUDOKU EN FILAS
}
if (RangoCandidato(VectorOrigen,0,9)==false) {
return false;
}
for (aux=1;aux<=9;aux++) {
if (ComparaElementos(VectorOrigen,aux)==false) {
return false;
}
}

for (cont=0;cont<=8;cont++) {
VectorOrigen[cont+1]=MatrizOrigen[cont][aux1];//COMPRUEBA SUDOKU EN COLUMNAS
}
if (RangoCandidato(VectorOrigen,0,9)==false) {
return false;
}
for (aux=1;aux<=9;aux++) {
if (ComparaElementos (VectorOrigen,aux)==false) {
return false;
}
}
}
for (aux1=0;aux1<=2;aux1++) { //COMPRUEBA SUDOKU  EN CAJAS
for (aux2=0;aux2<=2;aux2++) {
RangoCaja=CeldaEstaEnCaja(((3*(aux1+1))-1),((3*(aux2+1))-1));

for (aux3=0;aux3<=2;aux3++) { //pongo todos los valores de la caja3X3 que
corresponda en un array para comprobar
```

```
                                        for (aux4=0;aux4<=2;aux4++) {

        VectorOrigen[(3*aux3)+aux4+1]=MatrizOrigen[RangoCaja[0][aux3]][(RangoCaja[1][aux4])];
                                        }
                                }
                                if (RangoCandidato(VectorOrigen,0,9)==false) {
                                return false;
                                }
                                for (aux=1;aux<=9;aux++) {
                                        if (ComparaElementos (VectorOrigen,aux)==false) {
                                        return false;
                                        }
                                }
                        }

                }
                return true;
        } // fin del metodo ComprobarSudoku


        public static int [][][] EstablecerCandidatos(int [][] MatrizOrigen) {
                int[][][] MatrizCandidatos = new int [9][9][11];
                int fila,columna,profund;
                for (fila=0;fila<=8;fila++) {
                        for (columna=0;columna<=8;columna++) {
                                Iniciar (MatrizCandidatos[fila][columna],1); //Poner candidatos a 1
                                MatrizCandidatos[fila][columna]= Suma (MatrizCandidatos[fila][columna]);// Total de
candidatos
                        }
                }
                for (fila=0;fila<=8;fila++) {
                        for (columna=0;columna<=8;columna++) {
```

```
                                        if (MatrizOrigen[fila][columna]==0) {//No tiene solucion original

                                        MatrizCandidatos[fila][columna]= Suma (MatrizCandidatos[fila][columna]);


                                        } else {

                                        Iniciar (MatrizCandidatos[fila][columna],0); //Poner candidatos a 0

                                        MatrizCandidatos= AjusteCandidatosGeneral
(MatrizCandidatos,fila,columna,(MatrizOrigen[fila][columna]));

                                        }

                                }

                }

                return MatrizCandidatos;

        }//fin del metodo EstablecerCandidatos


        public static int[][][] AjusteCandidatosSolos(int [][][] MatrizCandidatosOrigen,boolean traza) {//encuentra y trata
los candidatos solos

                int[][][] MatrizCandidatosFin = new int [9][9][11];

                int[][][] MatrizCandidatosAux = new int [9][9][11];

                int fila,columna,posicion,aux1,aux2;

                boolean encontrado,unico;

                aux1=0;aux2=0;

                for (fila=0;fila<=8;fila++) {

                        for (columna=0;columna<=8;columna++) {

                                unico=MirarCandidatos(MatrizCandidatosOrigen[fila][columna],1) ;

                                if (unico==false) { //No tiene solucion hallada

                                continue;

                                }else {

                                        encontrado=CandidatoEncontrado (MatrizCandidatosOrigen[fila][columna]);


                                        if (encontrado==true) { //ya se habia tratado esta celda

                                        continue;

                                        } else {

                                        posicion=BuscarUnico(MatrizCandidatosOrigen[fila][columna]) ;


        MatrizCandidatosFin=AjusteCandidatosGeneral(MatrizCandidatosOrigen,fila,columna,posicion);
```

```
                                        if (traza==true) { //mostrar la secuencia de valores que se añaden
                                                System.out.println("El candidato "+posicion+" era unico en la fila "+(fila+1)+" columna "+(columna+1));
                                                System.out.println();
                                        }
                                    MatrizCandidatosFin= AjusteCandidatosSolos(MatrizCandidatosFin,traza);//llamada a si misma para que comienze el proceso desde el principio otra vez
                                    }
                                    }//fin del if-else
                            }
                    }

                    return MatrizCandidatosOrigen;//en la ultima llamada no hay candidatos solos y devuelve el resultado de la anterior
            }//fin del metodo AjusteCandidatosSolos


            public static int[][][] AjusteSolosEscondidos(int [][][] MatrizCandidatosOrigen,boolean traza) {//encuentra y trata los candidatos solos escondidos
                    int[][][] MatrizCandidatosFin = new int [9][9][11];
                    int[][] RangoCaja = new int[2][3];
                    int[] VectorAux = new int[11];
                    int fila,columna,posicion,aux,aux1,aux2,aux3,aux4,aux5,cont;
                    boolean unico,encontrado;
                    MatrizCandidatosFin=MatrizCandidatosOrigen;
                    VectorAux=Iniciar(VectorAux,0);//inicializo el vector auxiliar de datos a 0 hasta la posicion 10
                    //COMPROBACION DE SOLOS ESCONDIDOS EN FILAS
                    aux5=0;
                    for (fila=aux5;fila<=8;fila++){
                            for (posicion=1;posicion<=9;posicion++) {
                                    for (cont=1;cont<=9;cont++) {
                                    VectorAux[cont]=MatrizCandidatosOrigen[fila][cont-1][posicion];
```

```
                    }
                VectorAux=Suma(VectorAux);
                unico=MirarCandidatos(VectorAux,1) ;
                aux=BuscarUnico(VectorAux);
                        if (unico==false) {//No tiene solucion hallada
                        continue;
                        } else {
                        encontrado=CandidatoEncontrado (MatrizCandidatosOrigen[fila][aux-1]);

                        if (encontrado==true) { //ya se habia tratado esta celda
                        continue;
                        } else {


                                MatrizCandidatosFin=
AjusteCandidatosGeneral(MatrizCandidatosOrigen,fila,aux-1,posicion);
                                if (traza==true) { //mostrar la secuencia de valores que se añaden
                                        System.out.println("El candidato "+posicion+" era unico
'escondido en fila' en la fila "+(fila+1)+" columna "+(aux)+" marca : "+(MatrizCandidatosOrigen[fila][aux-1][10]));
                                        System.out.println();


                                }
        MatrizCandidatosFin=AjusteCandidatosSolos(MatrizCandidatosFin,traza);// busco si hay candidatos solos
                                MatrizCandidatosFin=
AjusteSolosEscondidos(MatrizCandidatosFin,traza);//llamada a si misma para que comienze el proceso desde el principio
otra vez
                        }
                        }
                }
        }

        //COMPROBACION DE SOLOS ESCONDIDOS EN COLUMNAS
        for (columna=0;columna<=8;columna++){
                for (posicion=1;posicion<=9;posicion++) {
```

```
                    for (cont=1;cont<=9;cont++) {
                    VectorAux[cont]=MatrizCandidatosOrigen[cont-1][columna][posicion];
                    }
                VectorAux=Suma(VectorAux);
                unico=MirarCandidatos(VectorAux,1) ;
                aux=BuscarUnico(VectorAux);
                        if (unico==false) {//No tiene solucion hallada
                        continue;
                        } else {
                        encontrado=CandidatoEncontrado (MatrizCandidatosOrigen[aux-1][columna]);

                        if (encontrado==true) { //ya se habia tratado esta celda
                        continue;
                        } else {


                                MatrizCandidatosFin=
AjusteCandidatosGeneral(MatrizCandidatosOrigen,aux-1,columna,posicion);
                                if (traza==true) { //mostrar la secuencia de valores que se añaden
                                    System.out.println("El candidato "+posicion+" era unico
'escondido en columna' en la fila "+(aux)+" columna "+(columna+1)+" marca : "+(MatrizCandidatosOrigen[aux-
1][columna][10]));
                                    System.out.println();


                                }

    MatrizCandidatosFin=AjusteCandidatosSolos(MatrizCandidatosFin,traza);// busco si hay candidatos solos
                                MatrizCandidatosFin=
AjusteSolosEscondidos(MatrizCandidatosFin,traza);//llamada a si misma para que comienze el proceso desde el principio
otra vez
                        }
                        }
                }
            }
```

```
//COMPROBACION DE SOLOS ESCONDIDOS EN CAJAS 3X3
for (aux1=0;aux1<=2;aux1++) {
    for (aux2=0;aux2<=2;aux2++) {
        for (posicion=1;posicion<=9;posicion++) {
        RangoCaja=CeldaEstaEnCaja(3*(aux1+1),3*(aux2+1));
            for (aux3=0;aux3<=2;aux3++) {     //pongo todos los valores de la caja3X3 que corresponda en un array para comprobar
                for (aux4=0;aux4<=2;aux4++) {

VectorAux[((3*aux3)+(aux4+1))]=MatrizCandidatosOrigen[(RangoCaja[0][aux3])][(RangoCaja[1][aux4])][posicion];

                }
            }
        VectorAux=Suma(VectorAux);
        unico=MirarCandidatos(VectorAux,1) ;
        aux=BuscarUnico(VectorAux);
            if (unico==false) {//No tiene solucion hallada
            continue;
            } else {
            encontrado=CandidatoEncontrado (MatrizCandidatosOrigen[(RangoCaja[0][(aux-1)/3])][(RangoCaja[1][(aux-1)%3])]);
            if (encontrado==true) { //ya se habia tratado esta celda
            continue;
            } else {

                MatrizCandidatosFin= AjusteCandidatosGeneral(MatrizCandidatosOrigen,(RangoCaja[0][(aux-1)/3]),(RangoCaja[1][(aux-1)%3]),posicion);

                if (traza==true) { //mostrar la secuencia de valores que se añaden
                    System.out.println("El candidato "+posicion+" era unico 'escondido en caja' en la fila "+((RangoCaja[0][(aux-1)/3])+1)+" columna "+((RangoCaja[1][(aux-1)%3])+1)+" marca : "+(MatrizCandidatosOrigen[(RangoCaja[0][(aux-1)/3])][(RangoCaja[1][(aux-1)%3])][10]));
                    System.out.println();
```

```
                                        }

        MatrizCandidatosFin=AjusteCandidatosSolos(MatrizCandidatosFin,traza);// busco si hay candidatos solos

                                                MatrizCandidatosFin=
AjusteSolosEscondidos(MatrizCandidatosFin,traza);//llamada a si misma para que comienze el proceso desde el principio otra vez

                                                }

                                                }

                                        }

                                }

                        }


                        return MatrizCandidatosOrigen;//en la ultima llamada no hay candidatos escondidos y devuelve el resultado de la anterior


        }//fin del metodo AjusteSolosEscondidos



        public static int[][][] AjusteCandidatosGeneral(int [][][] MatrizCandidatosOrigen,int fila,int columna,int posicion)
{
        //pone las filas columnas y cajas3X3 a 0 en la posicion donde se ha hallado el unico candidato
                int[][][] MatrizCandidatosFin = new int [9][9][11];
                int[][][] MatrizCandidatosAux = new int [9][9][11];
                int[][] RangoCaja = new int[2][3];
                int aux1,aux2;
                                        for (aux1=0;aux1<=8;aux1++) {
                                                if(CandidatoEncontrado(MatrizCandidatosOrigen[fila][aux1])==true) {
                                                        continue;
                                                } else {
                                                        MatrizCandidatosOrigen[fila][aux1][posicion]=0 ;//pongo a 0 las casillas de filas segun SUDOKU
                                                }
                                        }
                                        for (aux1=0;aux1<=8;aux1++) {
```

```
                    if(CandidatoEncontrado(MatrizCandidatosOrigen[aux1][columna])==true) {
                        continue;
                    } else {
                        MatrizCandidatosOrigen[aux1][columna][posicion]=0 ;//pongo a 0 las columnas adjuntas segun SUDOKU
                    }
                }

                RangoCaja=CeldaEstaEnCaja (fila,columna) ;
                for (aux1=0;aux1<=2;aux1++) {
                    for (aux2=0;aux2<=2;aux2++) {

    if(CandidatoEncontrado(MatrizCandidatosOrigen[RangoCaja[0][aux1]][RangoCaja[1][aux2]])==true) {
                            continue;
                        } else {

    MatrizCandidatosOrigen[RangoCaja[0][aux1]][RangoCaja[1][aux2]][posicion]=0 ;//pongo a
                            // 0 las casillas de la caja 3X3 segun SUDOKU
                        }
                    }
                }
            MatrizCandidatosOrigen[fila][columna]=Iniciar(MatrizCandidatosOrigen[fila][columna],0);//pongo a 0 los candidatos de la celda (para solos escondidos)
            MatrizCandidatosOrigen[fila][columna][posicion]=1 ;//dejo a 1 el encontrado
            MatrizCandidatosFin[fila][columna]=AjusteFinalCandidato(MatrizCandidatosOrigen[fila][columna]) ;
            MatrizCandidatosFin=ActualizaCandidatos(MatrizCandidatosOrigen);

            //Ajuste final de cuantos candidatos quedan en cada celda del SUDOKU
            return MatrizCandidatosFin;

    }// fin del metodo  AjusteCandidatosGeneral
```

```
public static int [][][] ActualizaCandidatos (int [][][] MatrizCandidatosOrigen) {
// Despues de cada ajuste actualiza la matriz de candidatos
        int fila,columna;
        for (fila=0;fila<=8;fila++) {
                for (columna=0;columna<=8;columna++) {
                MatrizCandidatosOrigen[fila][columna]=Suma (MatrizCandidatosOrigen[fila][columna]);
                }
        }
        return MatrizCandidatosOrigen;
} //fin del metodo ActualizaCandidatos


public static int[][][] VueltaAtras (int[][][]MatrizCandidatosOrigen,int fila,int columna,int posicion,boolean traza)
{
//Escoge un candidato y avanza en la resolucion, si es erroneo retrocede. Hasta que encuentre la solucion
        int[][][] MatrizAux1= new int[9][9][11];//hace los calculos de sudoku
        int[][][] MatrizAux2= new int[9][9][11];// se usa para la vuelta atras de la matriz
        int[][][] MatrizFin= new int[9][9][11];//sudoku final
        int[][][] MatrizVacia= new int[9][9][11];//para devolver en caso de error
        int[][][] MatrizAnt= new int[9][9][11];//usada para la traza
        int[]Cero={0,0,0,0,0,0,0,0,0,0,0};//vector de ceros para crear la matriz vacia
        int aux1,aux2,aux3;
        MatrizAux1=CopiaMatriz(MatrizCandidatosOrigen);
        int[] eleccion=EleccionVueltaAtras(MatrizAux1,fila,columna,posicion);
        MatrizVacia=CopiaVector(Cero);

        // CASOS TRIVIALES
        if ((eleccion[0]==0)&&(eleccion[1]==0)&&(eleccion[2]==0)) {
                                                        if (traza==true) {
                                                        System.out.println("SUDOKU RESUELTO ");
```

```
                                                    System.out.println();
                                                    }
                    MatrizFin=CopiaMatriz(MatrizCandidatosOrigen);   // HEMOS ENCONTRADO LA SOLUCION DEL SUDOKU
                } else {

                    if ((eleccion[0]==0)&&(eleccion[1]==0)&&(eleccion[2]==10)) {
                                                    if (traza==true) {
                                                    System.out.println ("Error en los calculos o el SUDOKU. Vuelta atras a la situacion anterior resuelta");
                                                    }
                            MatrizFin=CopiaMatriz(MatrizVacia);        // SE HA HECHO UN RECORRIDO ERRONEO
                    } else {

                    // CASOS NO TRIVIALES
                    aux1=eleccion[0];//fila escogida
                    aux2=eleccion[1];//columna escogida
                    aux3=eleccion[2];//posicion escogida
                    Iniciar(MatrizAux1[aux1][aux2],0);//                                    me preparo para
                    MatrizAux1[aux1][aux2][aux3]=1;//                                       comprobar si el candidato elegido
                    MatrizAux1[aux1][aux2]=Suma(MatrizAux1[aux1][aux2]);//              es valido
                    MatrizAnt=CopiaMatriz(MatrizAux1);//guardo una copia por si es correcto y se pide hacer la traza
                    MatrizAux1=AjusteCandidatosSolos(MatrizAux1,false);// resuelvo el candidato solo que acabo de establecer
                    //MatrizAux1=AjusteSolosEscondidos(MatrizAux1,false);//hago el resto de ajustes solos escondidos,etc..  // QUEDA AQUI POR SI SE QUIERE PROBAR SOLOS ESCONDIDOS(QUITAR // INICIAL)

                            // SITUACION DE PODA EL CANDIDATO ELEGIDO NO ES VALIDO, DEJA CELDAS SIN CANDIDATOS
                            if (ComprobarCandidatos(MatrizAux1)==false) {
                                                    if (traza==true) {
                                                    System.out.println("Candidato "+aux3+"
```

```
seleccionado en fila "+(aux1+1)+" columna "+(aux2+1)+" con algoritmo de vuelta atras NO ES VALIDO. Buscamos otro.
");

                                                        System.out.println();

                                                        }

                                MatrizFin=VueltaAtras(MatrizCandidatosOrigen,aux1,aux2,aux3,traza);


                                //CASOS NO TRIVIALES

                                } else {

                                                        if (traza==true) {

                                                        System.out.println("Candidato "+aux3+"
seleccionado en fila "+(aux1+1)+" columna "+(aux2+1)+" con algoritmo de vuelta atras. ");

                                                        System.out.println();

                                                        MatrizAnt=AjusteCandidatosSolos(MatrizAnt,true);

        //MatrizAnt=AjusteSolosEscondidos(MatrizAnt,true);// QUEDA AQUI POR SI SE QUIERE PROBAR SOLOS
ESCONDIDOS(QUITAR // INICIAL)

                                                        SalidaDatos(MatrizAnt);

                                                        System.out.println();

                                                        System.out.println("Ahora se procede a recalcular los
candidatos .");

                                                        }


                                MatrizAux2=VueltaAtras(MatrizAux1,0,0,0,traza);// busco hacia adelante la siguiente
solucion posible


                                        if (ComprobarCeros(MatrizAux2)==true) {

                                        //el Sudoku no acaba correctamente y hay que volver atras

                                        MatrizFin=VueltaAtras(MatrizCandidatosOrigen,aux1,aux2,aux3,traza);

                                        } else {

                                        //el Sudoku acaba correctamente y devuelve el resultado


                                        MatrizFin=CopiaMatriz(MatrizAux2);

                                        }

                                }
```

```
                    }
                }
        return MatrizFin;
        }//fin del metodo VueltaAtras



        public static int[] EleccionVueltaAtras (int[][][]MatrizCandidatosOrigen,int fila,int columna,int posicion) {
        //SITUACION DE PODA tiene que escoger un candidato (mayor de 1); de entre todas las celdas la que tenga menor,para evitar el mayor numero de busquedas
        int[] eleccion= new int [3];
        int aux1,aux2,aux3,aux4,aux5,cand;
        cand=2;
        aux3=2;
        if ((fila==0)&&(columna==0)&&(posicion==0)) {
                while ((cand+1)>aux3) {// Busco celdas no resueltas con el minimo de candidatos,o sea empezando por 2
                        for (aux1=0;aux1<=8;aux1++) {
                                for (aux2=0;aux2<=8;aux2++) {
                                 if (CandidatoEncontrado(MatrizCandidatosOrigen[aux1][aux2])==true) {// ya esta resuelta
                                 continue;
                                 } else {
                                         cand=MatrizCandidatosOrigen[aux1][aux2][0];//tiene estos candidatos. de esta forma siempre escojera una celda
                                         if (cand>aux3) {//busco la celda con menos candidatos del sudoku
                                         continue;
                                         } else {
                                                 for(aux4=posicion+1;aux4<=9;aux4++) {//escojo el menor candidato que sea mayor que posicion
                                                         if (ComprobarPosicion(MatrizCandidatosOrigen[aux1][aux2],aux4)==true) {
                                                         eleccion[0]=aux1;// fila ,
                                                         eleccion[1]=aux2;//  columna y
                                                         eleccion[2]=aux4;//  posicion a devolver con el candidato elegido
```

```
                                                    return eleccion;
                                                    }
                                                }//fin de la iteracion en las posiciones de los candidatos
                                            }
                                        }
                                    }// fin de la iteracion en las columnas
                                }// fin de la iteracion en las filas
                                aux3++;
                            }//fin del while
                            eleccion[0]=0;// fila ,
                            eleccion[1]=0;//  columna y
                            eleccion[2]=0;//  posicion a devolver con 0 porque ya esta resuelto, no hay candidatos
                            return eleccion;
                        }else {
                            for (aux5=(posicion+1);aux5<=9;aux5++) {
                                if (ComprobarPosicion(MatrizCandidatosOrigen[fila][columna],aux5)==true) {
                                eleccion[0]=fila; //fila ,
                                eleccion[1]=columna; //  columna y
                                eleccion[2]=aux5; //posicion a devolver con el candidato elegido
                                return eleccion;
                                }
                            }
                            eleccion[0]=0;// fila ,
                            eleccion[1]=0;//  columna y
                            eleccion[2]=10;//  posicion a devolver con 10 porque existe un error, no hay candidatos hay que volver
atras
                            return eleccion;
                        }
                        }//fin del metodo EleccionVueltaAtras
```

```
} // Fin de la clase OperacionesSudoku



class Util {
// clase con funciones sencillas y datos utiles

	static int xaux; //variable auxiliar



	public static int[]  Iniciar (int[] Matriz,int inicio) { //inicializa el vector base al valor de inicio
		Arrays.fill(Matriz,1,10,inicio);
		Matriz[10]=0;
		return Matriz;
	} //fin del metodo Iniciar



	public static int[]  Suma (int[] Matriz) {// pone en la celda 0 el valor de la suma de las celdas 1-9
		int sumaux; //variable auxiliar
		sumaux=0;
		for (xaux=1;xaux<10;xaux++) {
		sumaux=(sumaux+(int)Matriz[xaux]);
		}
		Matriz[0]=sumaux;
		return Matriz;
	} //fin del metodo Suma
```

```
	public static int[]  AjusteFinalCandidato (int[] Matriz) { //despues de una seleccion hace la suma y pone un 1 en la celda 10

		int[] Matrizaux=new int[11];
	//para identificarlo como hallado ya

		Matrizaux = Suma (Matriz);

		if (Matrizaux[0]==1) {

			Matrizaux[10]=1;

		}

		return Matrizaux;

	} //fin del metodo AjusteFinalCandidato




	public static boolean MirarCandidatos (int[] Matriz,int candidatos) { //devuelve cierto si tiene los candidatos que se pasan

		if (Matriz[0]==candidatos) {

			return true;

		}

		return false ;

	} //fin del metodo




	public static boolean ComprobarPosicion (int[] Matriz,int posicion) { //devuelve cierto si la posicion tiene un candidato

		if (Matriz[posicion]==1) {

			return true;

		}

		return false ;

	} //fin del metodo ComprobarPosicion




	public static boolean CandidatoEncontrado (int[] Matriz) { //devuelve cierto si el candidato esta

		// y se habia hallado con anterioridad es decir la celda 10 vale 1
```

```
		if ((Matriz[0]==1)&&(Matriz[10]==1)) {
			return true;
		}
		return false ;
	} //fin del metodo CandidatoEncontrado


	public static int BuscarUnico (int[] Matriz) { // busca donde esta el unico candidato
		if (MirarCandidatos(Matriz,1)==true) {
			for (xaux=1;xaux<10;xaux++) {
				if (ComprobarPosicion(Matriz,xaux)==true) {
				return xaux;
				}
			}
		}
		return 0;// No tiene candidato unico
	} //fin del metodo BuscarUnico


	public static boolean ComparaElementos (int[] Matriz,int elemento) { //busca si existe mas de un candidato igual a elemento
		int igual=0;
		for (xaux=1;xaux<10;xaux++) {
			if (Matriz[xaux]==elemento) {
			igual++;
			}
		}
		if (igual>1) {
			return false; //hay mas de un candidato igual y devuelve falso
			}
		return true; //hay uno o ningun candidato igual y devuelve cierto
```

```
} //fin del metodo ComparaElementos

public static boolean RangoCandidato (int[] Matriz,int minimo,int maximo) { //calcula si los candidatos estan en un rango
        int maux;
        for (xaux=1;xaux<10;xaux++) {
        maux=(int)Matriz[xaux];
                if ((maux<minimo)||(maux>maximo)) {
                return false; // Estan fuera del rango y devuelve falso
                }
        }
        return true;// Estan dentro del rango y devuelve cierto
} //fin del metodo  RangoCandidato

public static int[][] CeldaEstaEnCaja (int Fila,int Columna) { //Dada una celda establece cuales son las de su caja 3X3
        int[][] RangoCelda = new int[2][3];
        int xaux;
        xaux = (((Fila+3)/3)*((Columna+9)/3)); // Formula para identificar la caja de cada celda
        switch (xaux) {  //Se asignan los valores de fila y columna de cada caja
                case 3:   RangoCelda[0][0]=0 ;
                                RangoCelda[0][1]=1 ;
                                RangoCelda[0][2]=2 ;
                                RangoCelda[1][0]=0 ;
                                RangoCelda[1][1]=1 ;
```

```
                    RangoCelda[1][2]=2 ;
                    break;
case 4:   RangoCelda[0][0]= 0;
                    RangoCelda[0][1]= 1;
                    RangoCelda[0][2]= 2;
                    RangoCelda[1][0]= 3;
                    RangoCelda[1][1]= 4;
                    RangoCelda[1][2]= 5;
                    break;
case 5:   RangoCelda[0][0]=0 ;
                    RangoCelda[0][1]=1 ;
                    RangoCelda[0][2]=2 ;
                    RangoCelda[1][0]=6 ;
                    RangoCelda[1][1]=7 ;
                    RangoCelda[1][2]=8 ;
                    break;
case 6:   RangoCelda[0][0]= 3;
                    RangoCelda[0][1]= 4;
                    RangoCelda[0][2]= 5;
                    RangoCelda[1][0]= 0;
                    RangoCelda[1][1]= 1;
                    RangoCelda[1][2]= 2;
                    break;
case 8:   RangoCelda[0][0]= 3;
                    RangoCelda[0][1]= 4;
                    RangoCelda[0][2]= 5;
                    RangoCelda[1][0]= 3;
                    RangoCelda[1][1]= 4;
                    RangoCelda[1][2]= 5;
                    break;
```

```
            case 9:   RangoCelda[0][0]=6 ;
                                RangoCelda[0][1]=7 ;
                                RangoCelda[0][2]=8 ;
                                RangoCelda[1][0]=0 ;
                                RangoCelda[1][1]=1 ;
                                RangoCelda[1][2]=2 ;
                                break;
            case 10: RangoCelda[0][0]= 3;
                                RangoCelda[0][1]= 4;
                                RangoCelda[0][2]= 5;
                                RangoCelda[1][0]= 6;
                                RangoCelda[1][1]= 7;
                                RangoCelda[1][2]= 8;
                                break;
            case 12: RangoCelda[0][0]=6 ;
                                RangoCelda[0][1]=7 ;
                                RangoCelda[0][2]=8 ;
                                RangoCelda[1][0]=3 ;
                                RangoCelda[1][1]=4 ;
                                RangoCelda[1][2]=5 ;
                                break;
            case 15: RangoCelda[0][0]= 6;
                                RangoCelda[0][1]= 7;
                                RangoCelda[0][2]= 8;
                                RangoCelda[1][0]= 6;
                                RangoCelda[1][1]= 7;
                                RangoCelda[1][2]= 8;
                                break;
            default:            break;
    }
    return RangoCelda;
```

```
} // Fin del metodo            CeldaEstaEnCaja



public static boolean ComprobarCandidatos(int[][][]Matriz) {
//comprueba que existen candidatos, 1 o mas, en todas las celdas
int fila,columna;
        for (fila=0;fila<=8;fila++) {
                for (columna=0;columna<=8;columna++) {
                        if (Matriz[fila][columna][0]==0) {
                        return false;
                        }
                }
        }
        return true;
}//fin del metodo ComprobarCandidatos



public static boolean ComprobarCeros(int[][][]Matriz) {
//comprueba en todas las celdas que no existan candidatos
int fila,columna,posicion;
        for (fila=0;fila<=8;fila++) {
                for (columna=0;columna<=8;columna++) {
                        for (posicion=0;posicion<=10;posicion++) {
                                if (Matriz[fila][columna][posicion]!=0) {
                                return false;
                                }
                        }
                }
        }
```

```
            return true;
}//fin del metodo ComprobarCeros


public static void SalidaDatos(int[][][]Matriz) {
int fila,columna;
int[][]MatrizResultado=new int[9][9];
System.out.println();
System.out.println("F");
System.out.println("I");
System.out.println("L      COLUMNAS       ");
System.out.println("A");
System.out.println("S  1 2 3 4 5 6 7 8 9  ");
System.out.println();
MatrizResultado=ConvierteResultado(Matriz);
            for (fila=0;fila<=8;fila++) {
                        System.out.print((fila+1)+"  ");
                        for (columna=0;columna<=8;columna++) {
                        System.out.print(MatrizResultado[fila][columna]+" ");
                        }
                        System.out.println(" ");
             }
             System.out.println();

             return;
}//fin del metodo SalidaDatos

public static int[][] ConvierteResultado(int[][][]Matriz) {
//convierte una matriz de candidatos este como este en una matriz de resultados
int fila,columna;
int[][]MatrizResultado=new int[9][9];
```

```
		for(fila=0;fila<=8;fila++) {
			for (columna=0;columna<=8;columna++) {
				if (CandidatoEncontrado(Matriz[fila][columna])==true) {
				MatrizResultado[fila][columna]=BuscarUnico(Matriz[fila][columna]);//pone el resultado
en cada celda
				}else {
				MatrizResultado[fila][columna]=0; //y si no se ha obtenido aun pone un 0
				}
			}
		}
		return MatrizResultado;
	}//fin del metodo ConvierteResultado

	public static int[][][] CopiaMatriz(int[][][]Matriz) {
	int[][][] MatrizAux=new int [9][9][11];//hace una copia de una matriz
	int fila,columna;
		for(fila=0;fila<=8;fila++) {
			for (columna=0;columna<=8;columna++) {
				System.arraycopy(Matriz[fila][columna],0,MatrizAux[fila][columna],0,11);
		}
	}
	return MatrizAux;
	}//fin del metodo CopiaMatriz

	public static int[][][] CopiaVector(int[]Matriz) {
	int[][][] MatrizAux=new int [9][9][11];//hace una copia de un vector en toda la matriz
	int fila,columna;
		for(fila=0;fila<=8;fila++) {
			for (columna=0;columna<=8;columna++) {
				System.arraycopy(Matriz,0,MatrizAux[fila][columna],0,11);
		}
```

```
        }
        return MatrizAux;
        }//fin del metodo CopiaVector

} //fin de la clase Util
```