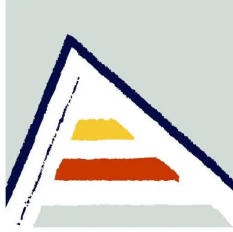


**SISTEMAS INTELIGENTES  
UNIVERSIDAD DE ALICANTE  
CURSO 2023/2024**



Universitat d'Alacant  
Universidad de Alicante

**PRÁCTICA 1**

**Satisfacción de restricciones**

**PROFESOR**

Lucas Martínez Bernabéu, [lucas.martinez@ua.es](mailto:lucas.martinez@ua.es)

**AUTORES**

Javier Sala Pérez, [jsp67@gcloud.ua.es](mailto:jsp67@gcloud.ua.es)

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Descripción de Algoritmos</b>	<b>2</b>
2.1. Forward Checking recursivo	2
2.1.1. Pseudocódigo	2
2.1.1.1. Explicación/Análisis	3
2.1.2. Análisis de mi implementación	3
2.1.2.1. Gestión de variables	3
2.1.2.1.1. Recorrer tablero y crear variables	4
2.1.2.1.2. Añadir restricciones a variables	6
2.1.2.1.3. Añadir dominios a las variables	7
2.1.2.2. Forward Checking	9
2.1.2.2.1. Forward Checking	9
2.1.2.2.2. Forward	10
2.1.2.2.3. Restaura	11
<b>3. Problema pequeño 2x2</b>	<b>12</b>
3.1. Grafo	13
3.2. Traza	13
<b>4. Pruebas y experimentación</b>	<b>14</b>
4.1. Análisis y gráfico de las pruebas	17
<b>5. Bibliografía</b>	<b>17</b>
<b>6. Anexo</b>	<b>18</b>

# 1. Introducción

En esta primera práctica de SI vamos a desarrollar la implementación de dos algoritmos de búsqueda, generando con estos los valores para un crucigrama que se ajusten a las restricciones especificadas.

Se nos ha proporcionado una plantilla con la parte del tablero ya implementada, la cual utiliza las librerías de pygame, para representar el entorno gráfico.

En mi práctica he implementado la creación de las variables para luego aplicar el algoritmo también implementado de Forward Checking.

## 2. Descripción de Algoritmos

### 2.1. Forward Checking recursivo

Primeramente el algoritmo Forward Checking recursivo, es uno de los algoritmo utilizado en la resolución de problemas de satisfacción de restricciones conocidos algoritmo CSP (Constraint Satisfaction Problems), estos se aplican como hemos visto en teoría sobre un conjunto de variables definidas sobre dominios finitos y conjunto de restricciones definidas sobre subconjuntos de dichas variables

#### 2.1.1. Pseudocódigo

```
función FC(i variable): booleano
    para cada a ∈ factibles[i] hacer
        Xi ← a
        si i=N solución retorna CIERTO
        sino
            si forward (i,a)
                si FC(i+1) retorna CIERTO
            Restaurar (i)
    retorna FALSO
función forward(i variable, a valor): booleano
    para toda j=i+1 hasta N hacer
        Vacío ← CIERTO
        para cada b ∈ factibles[j] hacer
            si (a,b) ∈ Rij vacío ←FALSO
            sino eliminar b de factible[j]
                Añadir b a podado[j]
        si vacío retorna FALSO
    retorna CIERTO
```

```

procedimiento restaura(i variable)
    para toda j=i+1 hasta N hacer
        para todo b ∈ podado[j] hacer
            si Xi responsable filtrado b
                Eliminar b de podado[j]
                Añadir b a factible[j]

```

#### 2.1.1.1. Explicación/Análisis

Vamos a analizar el pseudocódigo, este consiste de 3 funciones que vamos a analizar:

FC → Esta será un booleano que devolverá si se ha cumplido el algoritmo o no, además recibe por parámetro las variables y un índice por el cual a analizar las variables

- Una vez dentro vamos a iterar con los valores de las variables
- Asignaremos a cada valor de las variables a #Ej asignamos AL a la variable h1
- Si i=N (todas las variables con valor) es cierto y solución encontrada
- Si no llamamos al forward para que asigne valor y aplique restricciones
- Si en la rama actual no se cumplen restricciones, volvemos atrás(restauramos la anterior)
- Por último si no se ha encontrado solución retorna falso

Forward → Esta es la encargada de ver si con el valor asignado a la variable esta cumple con las restricciones

- Itera desde i+1 hasta N
- Crea una variable Vacío = True que devolverá al final
- Por último itera sobre los valores de b que son válidos en j, si la restricción entre 'a' y 'b' para las variables 'i' y 'j' es válida y pondrá Vacío Falso, de lo contrario eliminará b de los dominios y la añadirá a podados.
- Retorna vacío, que retornara true o false en función de que haya valores factibles.

Restaura → Encargada de volver al estado anterior

- Itera sobre cada valor desde i+1 hasta N
- Por último para cada valor de b de la lista de podados de la variable j, si Xi es responsable elimina b de podados y añade b al dominio de vuelta.

#### 2.1.2. Análisis de mi implementación

##### 2.1.2.1. Gestión de variables

Esta la hago desde la función `recorreTablero` a la cual le mando por referencia el tablero, voy a dividir la explicación en tres partes, la primera será recorrer el tablero tanto horizontal como vertical para crear las variables, seguiremos con la creación de los restricciones y por último las dominios .

### 2.1.2.1.1. Recorrer tablero y crear variables

En esta parte crearemos las variables con dominio y restricción vacío, voy a mostrar como lo hago para las variables horizontales, para las verticales sería exactamente lo mismo pero con orientación vertical.

Para la explicación más sencilla con el código voy a copiar mi código con una herramienta que simula cómo se vería en python mostraré comentado todos los pasos que realiza este, mostraré con azul claro algún comentario que crea necesario

```
def recorreTablero(tablero):
    variables = []
    restricciones = []
    #Contador para las variables verticales, las horizontales llevan el
    #orden de nº de variables en cambio vert. empiezan por 1 otra vez
    contador_var_v = 1

    # Recorrer horizontalmente recorriendo filas y columnas
    for fila in range(tablero.getAlto()):
        palabra = [] # Lista de letras de la palabra
        for col in range(tablero.getAncho()):
            celda = tablero.getCelda(fila, col)

            # Sumamos letra a palabra comprobando bloqueo o fin tablero
            if celda != '*' and col != tablero.getAncho() - 1:
                palabra.append(celda)

            # Tenemos tres maneras de crear variable, la primera será al tener una
            # palabra ya formada y llegar a *(bloqueo)
            # Llegamos a un bloqueo y tenemos palabra
            elif celda == '*':
                if palabra:
                    # +1 para incluir la celda actual
                    nombre = 'h' + str(len(variables) + 1)
                    # +1 para incluir la celda actual
                    longitud = len(palabra)
                    orientacion = "horizontal"
                    dominio = []
                    restricciones = []
                    var = Variable(nombre, fila, col-len(palabra),
                                  longitud, orientacion, dominio,
                                  restricciones)

                    #Creamos la variable
                    variables.append(var)
                #borramos el contenido de la palabra
                palabra = []
```

La segunda manera será al llegar al final del ancho de tablero, lo que hago es que si no tenemos ni letras, ni está vacío, ni hay bloquero, (no hay nada por que hemos llegado el final del tablero), creamos variable

```
# Final tablero y tenemos palabra
elif palabra:
    # +1 para incluir la celda actual
    nombre = 'h' + str(len(variables) + 1)
    # +1 para incluir la celda actual
    longitud = len(palabra)+1
    orientacion = "horizontal"
    dominio = []
    restricciones = []
    var = Variable(nombre, fila, col-len(palabra),
                   longitud, orientacion, dominio,
restricciones)
    variables.append(var)
    palabra = []
```

Y por último tenemos un caso en el que la variable anterior es un bloqueo y esto me daba errores de variables que se sumaba al tamaño primeramente el \* ej. \*LO -> long=3, para esto cree este paso.

```
# crear variable si la celda anterior *
elif (tablero.getCelda(fila, col-1) == "*"):
    # +1 para incluir la celda actual
    nombre = 'h' + str(len(variables) + 1)
    # +1 para incluir la celda actual
    longitud = len(palabra)+1
    orientacion = "horizontal"
    dominio = []
    restricciones = []
    var = Variable(nombre, fila, col-len(palabra),
                   longitud, orientacion, dominio,
restricciones)
    variables.append(var)
    palabra = []
```

Seguimos ahora con las verticales que es exactamente los mismo, a excepción de que recorreremos el ancho del tablero (nº columnas) y luego su largo (filas) esto es al contrario que en horizontal, además en variable la orientación será= vertical y su nombre será nombre:

```
'v'+contador_var_v
# Recorrer verticalmente
for col in range(tablero.getAncho()):
    palabra = []
    for fila in range(tablero.getAlto()):
```

### 2.1.2.1.2. Añadir restricciones a variables

Pasamos a la parte de las restricciones, aquí lo tengo organizado en pasos siguiendo los pasos que mi profesor me recomendó a través de una tutoría.

Primeramente recorremos las variables verticales y luego las horizontales para comprobar a posterior si se cruzan

```
# 1º recorremos verticales y comprobamos si variable v
for var_v in variables:
    if var_v.getOrientacion() == "vertical":
        # 2º recorremos horizontales y comprobamos si variable h
        for var_h in variables:
            if var_h.getOrientacion() == "horizontal":
```

Aquí vamos a comprobar cuatro cosas:

- Fila máxima vertical  $\geq$  Fila horizontal
- Fila mínima vertical  $\leq$  Fila horizontal
- Columna máxima horizontal  $\geq$  Columna vertical
- Columna mínima horizontal  $\leq$  Columna vertical

```
# fMv $\geq$ fh Y fmv  $\leq$  fh Y cMh  $\geq$  cv Y cMh  $\leq$  cv
# Por tanto: fmv  $\leq$  fh  $\leq$  fMv(fv+lv) Y cv  $\leq$  ch  $\leq$  cMh(ch+lh)
if (var_v.getFila()  $\leq$  var_h.getFila()  $\leq$ 
    var_v.getFila() + var_v.getLongitud() and
    var_h.getColumna()  $\leq$  var_v.getColumna()  $\leq$ 
    var_h.getColumna() + var_h.getLongitud()):
```

```
# 4ª celada sera filaV-filaH y columnaV-columnaH
fila_cruce_h = var_h.getFila() - var_v.getFila()
columna_cruce_v = var_v.getColumna() - var_h.getColumna()
```

Las restricciones las quise crear como el ejemplo, dudo que fuera mi mejor idea ya lo luego me trajo problemas pero sigue el formato de las trazas.

```
restriccion_h= ("(mi_celda: " + str(columna_cruce_v) + ",
    var: " + str(var_v.nombre) + ", su_celda:"
    + str(fila_cruce_h)+")")
restriccion_v = ("(mi_celda: " + str(fila_cruce_h) + ",
    var: " + str(var_h.nombre)+", su_celda: "
    + str(columna_cruce_v)+")")
```

Añadimos tanto a la variable horizontal como a la vertical su restricción

```
var_h.addRestriccion(restriccion_h)
var_v.addRestriccion(restriccion_v)
```

### 2.1.2.1.3. Añadir dominios a las variables

En la creación de dominios, vamos a buscar en el almacén, donde están las palabras importadas del .txt (función ya creada en el proyecto base), y las vamos a añadir a las variables siguiendo unas normas:

- El dominio tendrá la misma longitud que el tamaño de la variable
- Si la variable tiene las celdas vacías, su dominio será todas aquellas palabras de su longitud
- Si la variable tiene algunas celdas con letras y otras vacías, su dominio serán todas aquellas palabras que cuadren ej \_o\_a Dominio:[Hola,Hoja,Cola...]
- Si la variable tiene letras en todas las celdas su dominio sólo puede ser ese, si no está en el diccionario, no tendrá dominio.

En mi programa entendí mal algunas de estas condiciones y tuve que reescribir porque funcionaba distinto.

Primero he inicializado cuatro variables, 3 serán para hacer controles en distintos if y la otra será para y sumándole a ésta los distintos dominios que después añadiré

```
empty = False
semi = False
control = False
letras_p = ""
```

Vamos a recorrer todas las variables, dentro todas las palabras del “diccionario” y aquí dentro vamos a recorrer las celdas para a posterior comparar

```
for var in variables:
    for palabra in almacen:
        if palabra.tam == var.getLongitud():
            for pal in palabra.getLista():
                letras_p = ""
                empty = False
                semi = False
                control = False
                # recorro todas las celdas de la palabra
                for i in range(len(pal)):

                    # Solo meto todos los dominios (de su tamaño) si la palabra
                    # tiene las celdas vacias
                    # Si está completa, su dominio solo tendrá esa palabra (si
                    # existe)
                    # si está semicompleta cuadramos las letras que se pueda
```

Estos if extensos, están basados basados en unas consultas a ChatGPT, la idea del control si la elegí yo, control solo sirve para controlar si ha entrado a empty, ya que tenía diversos fallos. Si están todas las celdas vacías empty=true

```
if (tablero.getCelda(var.getFila() + (i if
    var.getOrientacion() == "vertical" else 0),
    var.getColumna() + (i if var.getOrientacion() ==
    "horizontal" else 0)) == VACIA) and control == False:
    empty = True
```



```

        else:
            control = True
            empt = False
Obtenemos el contenido de la celda y lo guardamos en letras_p, para el contenido de
la celda la parte de la orientación ayuda CharGPT
        # guardamos todas las letras de las celdas de la palabra
        celda = tablero.getCelda(var.getFila() + (i if
            var.getOrientacion() == "vertical" else 0),
            var.getColumna() + (i if var.getOrientacion() ==
                "horizontal" else 0))
        letras_p += celda
Palabras completas, pal (palabra diccionario) == letras_p (suma celdas)
        # si letras_p == pal y
        if len(letras_p) == len(pal) and len(pal) ==
            var.getLongitud() and celda != '-' and letras_p != '-' and
            letras_p == pal:
            # print('Letras celda', letras_p)
            var.addDominio(letras_p)
            letras_p = ""
Palabras semicompletas, comprobamos tamaños validos, si la palabra del diccionario
tiene tamaño == a el tamaño de letras_p y la long de la variable, además que no esten
ni todas las celdas llenas ni vacías
        if len(letras_p) == len(pal) and len(pal) == var.getLongitud()
            and empt == False and semi == False:
            # 'y', pal, 'y', var.nombre)
Una vez esto, ya compruebo que o es la misma letra que letras_p o es celda vacía, si
es asi será pala semí y la añadiremos, si no no será y no se añade a su dominio.
        for j in range(len(pal)):

            if letras_p[j] == pal[j] or letras_p[j] == '-':
                semi = True
            else:
                semi = False
                break
Por ultimo añadir al dominio las palabras vacías o completas
        if empt:
            var.addDominio(pal)
            letras_p = ""

        if semi:
            var.addDominio(pal)
            letras_p = ""

return variables

```

### 2.1.2.2. Forward Checking

Como hemos visto en el pseudocódigo, el algoritmo consta de tres métodos FC que asigna valor y comprobará si es solución, forward que comprueba si es valido ese valor y restaura si el valor no ha sido válido para volver al anterior estado.

#### 2.1.2.2.1. Forward Checking

Recibirá por parámetro tres valores el primero i que será para tener un índice sobre las variables a iterar y un segundo que serán las variables, además yo he añadido uno más para crear un índice para mostrar iteraciones por pantalla (no es necesario para el algoritmo).

El funcionamiento de este es muy similar al pseudocódigo.

```
def forwardChecking(i, variables, count):
    control = False
    dominios_temp = []
    count += 1
    Si encuentra solución la muestra por pantalla y devuelve true
    # si i=N solución retorna CIERTO
    if (i == len(variables)):
        control = True
        print("Solución encontrada")
        print("Solución: ")
        for var in variables:
            print (var.getNombre(), ":", var.getValor())
        return control
    # para cada a ∈ factibles[i] hacer
    var = variables[i]
    for a in var.getDominio():
        #  $X_i \leftarrow a$ 
        variables[i].setValor(a)
    Copia temporal de dominios para su posterior restauración
    for var in variables:
        dominios_temp.append(copy.deepcopy(var.getDominio()))
    # si forward (i,a)
    Llamamos a forward con el índice el valor seleccionado y las variables
    if forward(i, a, variables):
        # si FC(i+1) retorna CIERTO
        if (forwardChecking(i+1, variables, count)):
            return TRUE
```

A restaura le pasamos la lista de dominios\_temp que hemos creado antes

```
#Restaura
```

```
if(restaura(i, variables, dominios_temp)) :
```

El siguiente return me trajo muchos quebraderos de cabeza, es el encargado de si hay solucion devolver el true, y parar el programa, si lo dejamos sin el return como lo tenía, itera infinitamente hasta que no haya más posibilidades, encontrado así todas las soluciones y todos las no soluciones.

```
return forwardChecking(i, variables, count+1)
```

```
else:
```

```
control = False
```

```
return control
```

#### 2.1.2.2.2. Forward

Esta función será la encargada con el valor añadido de eliminar dominios del resto de variables siguiendo las restricciones previamente creadas, si no queda ningún dominio vacío devolverá true, en caso contrario false.

```
def forward(i, a, variables):  
    var_act = variables[i]  
    vacio = TRUE  
    # para toda j=i+1 hasta N hacer
```

Primeramente tengo un bucle que elimina de todos los dominios el valor asignado

```
for var in variables:  
    for dominio in var.getDominio():  
        if dominio == a:  
            var.getDominio().remove(dominio)  
            var.getPodados().append(dominio)  
            vacio = FALSE
```

#Aquí recorremos las restricciones de la variable seleccionada para eliminar los dominios que no coincidan

```
for restriccion in var_act.getRestricciones():  
    #vamos directos a las variables que tienen restricciones con la variable i
```

Tanto esta función como las dos de las celdas que vienen ahora, las obtuve con ChatGPT, al guardar las restricciones de la manera que dije anteriormente, necesito sacar el nombre de la variable para así obtener la variable, al sacar el nombre lo único que tengo es como un string con el nombre, por eso luego recorro todas la variables para encontrar una con ese nombre y asignarle ahora si la variable

```
variable_rest = restriccion[restriccion.find(  
    "var:") + 5:restriccion.find("su_celda:") - 2]
```

```

    for vars in variables:
        if vars.getNombre() == variable_rest:
            var_rest = vars

```

Con la variable ahora tuve que hacer algo similar para las celdas, ya que tenía problemas con que se salían del dominio.

```

    # celda de la variable al cruzarse
    celda = (int(restriccion[restriccion.find(
        "su_celda:") + 10:restriccion.find(")"])))
    celda2 = int(restriccion[restriccion.find(
        "mi_celda:") + 10:restriccion.find("var:") - 2])

```

Comprobamos letra a letra si encajan los cruces, si no, lo eliminamos

```

    for dominio in var_rest.getDominio() + var_act.getDominio():
        # si no cuadra la primera letra borramos
        if celda < len(dominio) and celda2 < len(dominio):
            if dominio[celda] != '#' and dominio[celda2] != '#' and
                dominio[celda] != a[celda2] and dominio in
                var_rest.getDominio():
                # Eliminamos b de variable[j]
                var_rest.getDominio().remove(dominio)
                # Añadir b a podado[j]
                var_rest.getPodados().append(dominio)
                vacio = FALSE

```

Por último borramos la lista de dominios de aquellas que tengan un valor asignado.

```

    variables[i].getDominio().clear()
    variables[i].getDominio().append(a)

```

Devolveremos true si ha salido bien, false si hay dominios vacíos y por tanto no hay solución por esa rama

```

    if vacio == TRUE:
        return FALSE

```

return TRUE

### 2.1.2.2.3. Restaura

Es el último método del algoritmo, es el encargado de volver al estado anterior en caso de no encontrar solución en el forward, recibe por parámetro el índice de variables, las variables, y la copia de dominios que hemos hecho en el FC.

```

def restaura(i, variables, dominios temp):
    # Recupera el valor de la variable actual
    valor_actual = variables[i].getValor()
    Bucle que recupera los dominios del estado anterior de todas las
    variables

```

```

for j, var in enumerate(variables):
    if j < len(dominios_temp):
        var.setDominio(dominios_temp[j])

Comprobamos si hay dominios vacíos, si hay dominios vacíos, significa
que no hay solución, si no hay debemos seguir buscando.
#comprobamos si hay dominios vacíos
for var in variables:
    if not var.getDominio():
        #print("No hay solución")
        return FALSE

Por último borramos el valor que hemos visto que no es solución de
la variable a la que se lo habíamos asignado
variables[i].getDominio().remove(valor_actual)

return TRUE

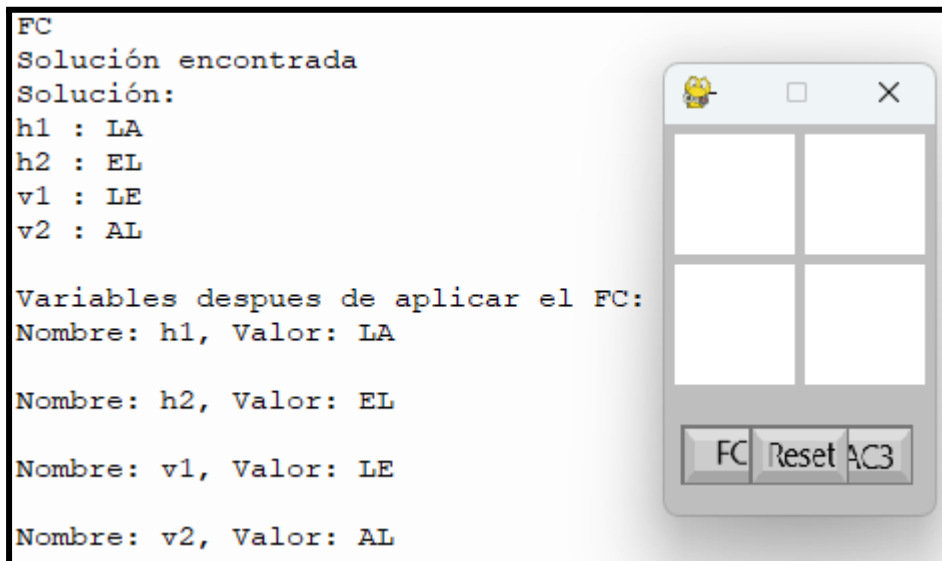
```

### 3. Problema pequeño 2x2

He seleccionado el problema que se nos muestra en el moodle con un diccionario más corto, este problema tiene un máximo de 28 iteraciones y 2 soluciones válidas, la primera solución es en la iteración 10.

El problema consiste en un 2x2 vacío que tiene como diccionario: LO, LA, EL, LE, AL.

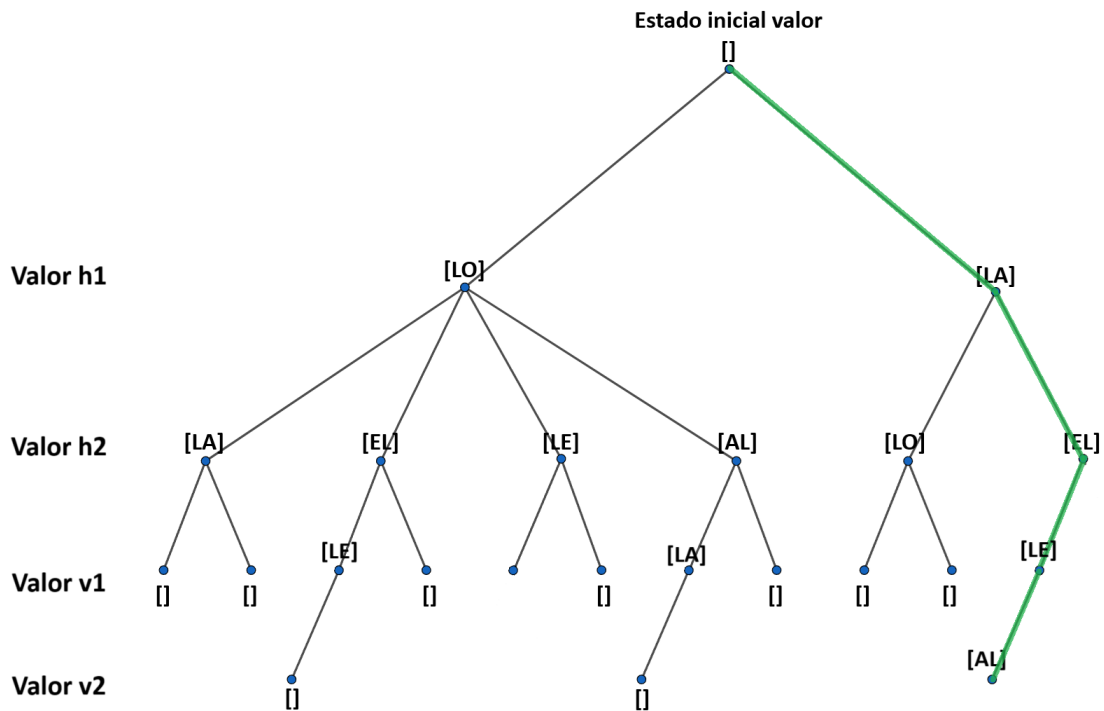
Al ejecutar el programa nos muestra la solución directamente:



### 3.1. Grafo

Grafo de la solución  $\rightarrow$  LA, EL, LE, AL.

El estado inicial será valor [] teniendo dominio [LO, LA, EL, LE, AL]



### 3.2. Traza

Para encontrar la solución vamos a necesitar 6 iteraciones, empezaremos con la iteración 0.

Iteración 0:

# h1: [ LO, LA, EL, LE, AL]

## h2: [ LO, LA, EL, LE, AL]

v1: [ LO, LA, EL, LE, AL]

v2: [ LO, LA, EL, LE, AL]

Iteración 1:

# h1→ Valor: [ LO ]

## h2: [ LA, EL, LE, AL]

v1: [ LA, LE]

v2: ☐

Asignamos LO a h1.

Tenemos un dominio vacío, por tanto restauramos al estado anterior.

Restauramos:

h1: [ LA, EL, LE, AL]

## h2: [ LO, LA, EL, LE, AL]

v1: [ LO, LA, EL, LE, AL]

v2: [ LO, LA, EL, LE, AL]

Iteración 2:

Asignamos LA a h1.

h1 → Valor: [LA]  
h2: [ LO, EL, LE, AL]  
v1: [ LO, LE]  
v2: [ AL]

Iteración 3:

h1 → Valor: [LA]  
h2 → Valor: [LO]  
v1: []  
v2: []

Asignamos LO a h2.

Tenemos dominios vacíos, por tanto restauramos al estado anterior

Restauramos:

h1 → Valor: [LA]  
h2: [ EL, LE, AL]  
v1: [ LO, EL, LE, AL]  
v2: [ LO, EL, LE, AL]

Iteración 4:

h1 → Valor: [LA]  
h2 → Valor: [EL]  
v1: [ LE ]  
v2: [ AL]

Asignamos EL a h2.

Iteración 5:

h1 → Valor: [LA]  
h2 → Valor: [EL]  
v1 → Valor: [LE]  
v2: [ AL]

Asignamos LE a v1

Iteración 6:

h1 → Valor: [LA]  
h2 → Valor: [EL]  
v1 → Valor: [LE]  
v2 → Valor: [AL]

Asignamos AL a v2

## 4. Pruebas y experimentación

He hecho diversas pruebas con distintos diccionarios, estos los he [descargado de github de un repositorio](#) y provienen de la rae, he tenido problemas con diccionarios muy grandes, y es que el programa me da un “crash”.

De la misma manera si esta el bucle iterando mucho manda el siguiente error:

```
File "C:\Users\javi9\AppData\Local\Programs\Thonny\lib\logging\ init .py", line 424, in u
sesTime
    return self._fmt.find(self.asctime_search) >= 0
RecursionError: maximum recursion depth exceeded while calling a Python object
```

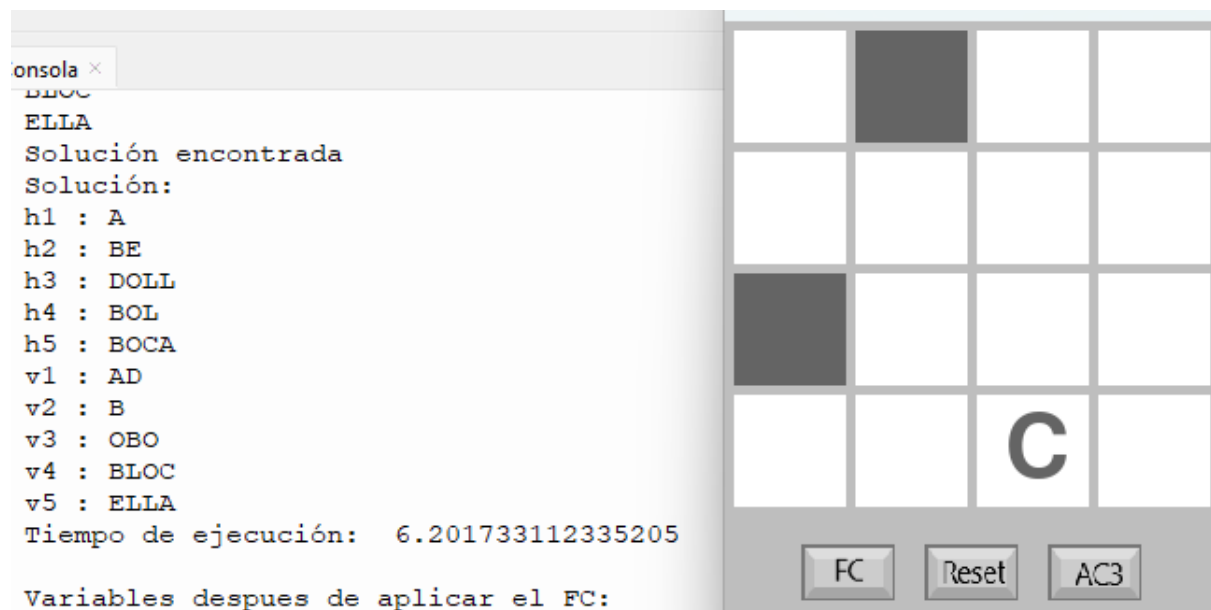
Que significa que hemos alcanzado el límite máximo de recursión de Python, esto creo que fue en un diccionario de tamaño de palabra 17.

Ahora vamos con diversas pruebas, he añadido un temporizador a las pruebas, para medir así su ejecución:

Empezamos con un 3x3 con celdas vacías y un diccionario de palabras de tamaño 3 con 1266 palabras,añado que “ABA” si es la primera palabra, pero “BAR” es la número 73, por tanto se ha tenido que restaurar varias veces, el tiempo ha sido de 3,2 seg, con 919 iteraciones.

```
Solución encontrada
Solución:
h1 : ABA
h2 : BAR
h3 : CHA
v1 : ABC
v2 : BAH
v3 : ARA
Tiempo de ejecución: 3.2053256034851074
```

De la misma manera he probado con un 4x4 con el mismo tipo de diccionario pero con palabras de tamaño 1,2,3 y 4 , ha tenido 1060 iteraciones y ha tardado 6.2seg



onsola x

```
BLOC
ELLA
Solución encontrada
Solución:
h1 : A
h2 : BE
h3 : DOLL
h4 : BOL
h5 : BOCA
v1 : AD
v2 : B
v3 : OBO
v4 : BLOC
v5 : ELLA
Tiempo de ejecución: 6.201733112335205

Variables despues de aplicar el FC:
```


FC Reset AC3

Volví al 3x3 limitando la letra de la esquina inferior derecha a una “Z” y con 2191 iteraciones tardó alrededor 8 seg.

```
Solución encontrada
Solución:
h1 : ABC
h2 : ROA
h3 : TAZ
v1 : ART
v2 : BOA
v3 : CAZ
Tiempo de ejecución: 8.267979383468628
```



También comprobé un 7x7 con una casilla bloqueada otra no sucesivamente, hay 24 casillas pero al no poder repetirse la solución no se iba a encontrar nunca, el tiempo de ejecución fué después de varias alrededor de 2,15 seg lo que con 643 iteraciones

He conseguido saltarme el límite de recursión con un 5x5 utilizando además vscode (con thonny me daba error de memoria) utilizando el diccionario de palabras de tamaño 5 con 25378 palabras ejecutarlo.

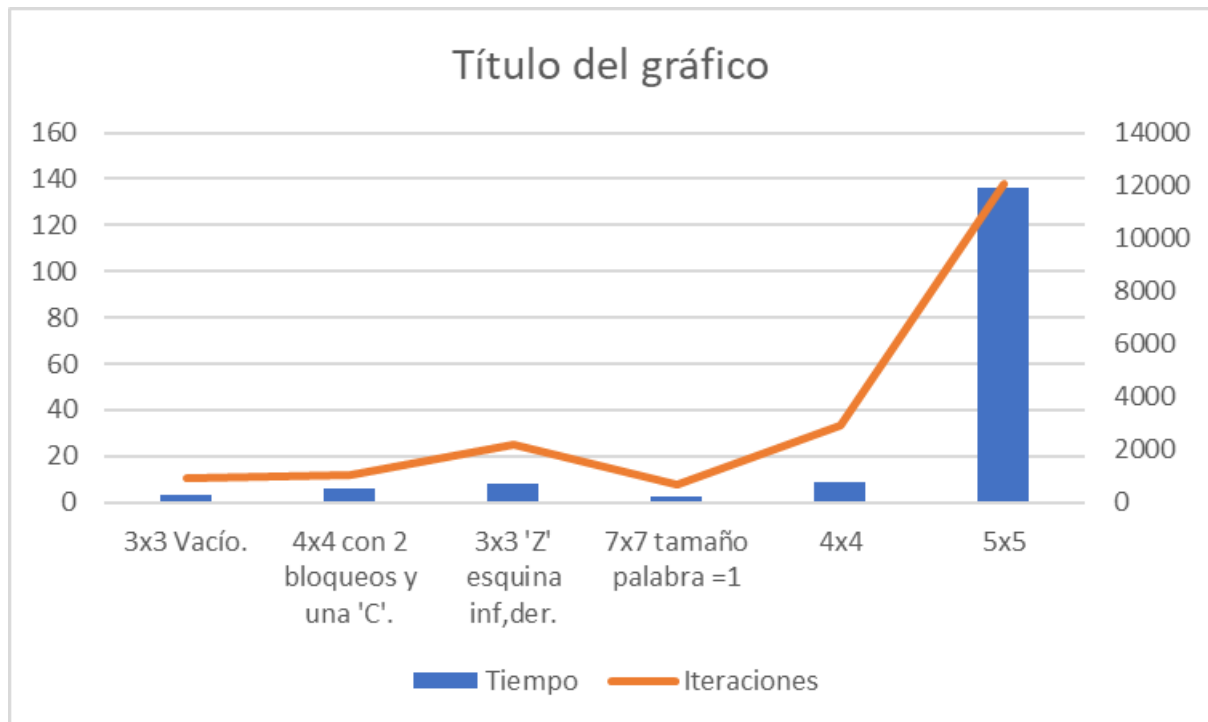
El resultado después de 12087 iteraciones ha sido:

```
Solución encontrada
Solución:
h1 : ABACA
h2 : MARES
h3 : ABACO
h4 : BEBAN
h5 : ALALA
v1 : AMABA
v2 : BABEL
v3 : ARABA
v4 : CECAL
v5 : ASONA
Tiempo de ejecución: 136.08881044387817
```

Y ya por último he hecho uno de la misma manera pero con 4x4 este tiene un diccionario de 7804 y su resultado ha sido con 2922 iteraciones:

```
Solución encontrada
Solución:
h1 : ABAA
h2 : DIEM
h3 : ACTA
h4 : SIAN
v1 : ADAS
v2 : BICI
v3 : AETA
v4 : AMAN
Tiempo de ejecución: 9.044774293899536
```

## 4.1. Análisis y gráfico de las pruebas



Como podemos ver en el gráfico que muestra en las barras azules el tiempo y en la línea naranja las iteraciones, podemos ver como el tiempo va en función de las iteraciones, y el tamaño de la palabra a la hora de comprobar no interviene mucho.

He calculado además que a excepción del 4x4 con bloqueos y la C y el 5x5 la iteración de palabra tarda alrededor de 0,0035 segundos.

## 5. Bibliografía

Tema 3.2 de los materiales de clase

Diccionarios de palabras utilizados para las pruebas:

<https://github.com/JorgeDuenasLerin/diccionario-espanol-txt/tree/master>

Satisfacción de Restricciones - Javier Béjar -UPC

<https://www.cs.upc.edu/~bejar/ia/transpas/teoria/2-BH5-CSP.pdf>

## 6. Anexo

He utilizado tanto stackoverflow como chatgpt para la resolución de dudas, como he mencionado en el código mostrado anteriormente como por ejemplo distintas partes en el forward como las búsquedas dentro de las variables, la utilización de la librería copy y su uso, el uso de las librerías time y sys para la parte de las pruebas.

Muchas de mis dudas también han sido resueltas por mi profesor de prácticas en distintas tutorías.