PRÁCTICA 1
SATISFACCIÓN DE RESTRICCIONES

Andrea Gómez Bobes

ÍNDICE

ALGORITMO FORWARD CHECKING

DEFINICIÓN

El algoritmo Forward Checking es un algoritmo de búsqueda para problemas de satisfacción de restricciones. Esto quiere decir que el problema es un conjunto de objetos que han de cumplir una serie de limitaciones.
En este caso el algoritmo empieza con una variable y busca valores dentro de su dominio que darle a esta variable que cumpla con las restricciones, lo que diferencia a este algoritmo de otros es que antes de instanciar el valor lo que se hace es comprobar el efecto que este valor escogido tiene en las siguientes variables que se encuentran restringidas por la variable actual. Si al comprobar este valor elegido alguna de las variables futuras se queda con dominio vacío la variable actual no se instancia con ese valor y se prueba con otro distinto.

ALGORITMO APLICADO AL CRUCIGRAMA

Para la resolución de un crucigrama el algoritmo de forward checking trabajaría de la siguiente manera.
Contamos con un grupo de variables que son el conjunto de palabras a rellenar, estas guardan su tamaño, las coordenadas de cada una de las casillas, su orientación (horizontal o vertical) y su dominio de valores posibles determinados por la longitud de la palabra y las palabras disponibles en el diccionario, las palabras del diccionario se guardan en un almacén que usaremos para rellenar los dominios.
Para almacenar este conjunto se recorre el mapa buscando los huecos y guardandolos.
Una vez guardados se empieza a implementar el algoritmo, se elige la primera variable, se selecciona un valor del dominio y se comprueba que a partir de ese valor el resto de variables sigan teniendo valores para poder rellenar el crucigrama, si hay valores entonces se instancia ese valor para la variable, si no se prueba con otro.
Si con ningún valor hay posibilidades para el resto de variables, el crucigrama no tendrá solución.

EXPLICACIÓN DEL CÓDIGO

La función empieza con las variables ya creadas en una lista, lo que hace es recorrer dicha lista e ir cogiendo las variables, de estas recorre su dominio. Por cada valor de su dominio comprueba si hay alguna otra variable que coincida en alguna coordenada con esta y si coincide comprueba por cada valor de su dominio si la letra que cruza coincide. Si la letra no coincide se borra ese valor de la variable próxima, si alguna variable se queda con dominio vacío se vuelve atrás y se prueba con otros valores del dominio.

```
def forwardChecking(almacen,COLS,FILS,tablero,res,lista):
    #Recorremos todas las variables guardadas
    inicio = time.time()
    i = 0
    while i < len(lista):
```

```
        print(f"Variable a instanciar: {i}")
        coincide = False
        hueco = lista[i]
        valido = False
        z =0
        p = 0
        dominio = hueco.getDominio()

        #Se recorren todas las opciones del dominio
        while z < len(dominio) and valido == False:
            p = 0
            posible = True
            sol = dominio[z]
            print(f"Posible valor: {sol}")
            j = i + 1
              #Se recorren las siguientes variables para comprobar el efecto que haría la variable
actual en sus dominios
            while j < len(lista) and posible == True:
                var_dor = copy.deepcopy(lista[j])
                #Recorremos las coordenadas de la siguiente variable
                for l in range(len(var_dor.getCoord())):
                    #Recorremos las coordenadas de la variable actual
                    for m in range(len(hueco.getCoord())):
                        if hueco.getCoord()[m] == var_dor.getCoord()[l]:
                            coincide = True
                            p=0
                            while p < len(var_dor.getDominio()):
                                    #Si cooinciden las coordenadas pero no coinciden los caracteres se
borra del dominio de la nueva variable
                                if var_dor.getDominio()[p][l] != sol[m]:
                                            print(f"Se borra del dominio de la variable {j} la palabra
{var_dor.getDominio()[p]}")
                                    var_dor.deleteDominio(p)
                                    print(f"Nuevo dominio de la variable {j} {var_dor.getDominio()}")
                                    p = p - 1
                                p = p + 1
                            #Si el dominio de la nueva variable se queda vacio debemos instancia otro
valor
                            if len(var_dor.getDominio()) == 0:
                                print(f"El valor {sol} no era factible, seguimos")
                                posible = False

                            #Si no se vacía guardamos el dominio editado
                            else:
                                lista[j] = var_dor
                                valido = True
                    j = j+1
            if coincide == False:
```

```
            print(f"Para esta variable no habían restricciones sin comprobar")
            valido = True


        z = z + 1

            #Una vez tenemos la solución para un hueco dejamos los caracteres en su correspondiente coordenada
        j = 0
        print(f"El valor {sol} sí es factible,se muestra en el crucigrama")
        for j in range(hueco.getTam()):
            hueco.addChar(sol[j])
            tablero.setCelda(hueco.getCoord()[j][0], hueco.getCoord()[j][1], sol[j].upper())
        x = i + 1
        y = 0
        while x  < len(lista):
            while y < len(lista[x].getDominio()):
                if lista[x].getDominio()[y] == sol:
                    lista[x].deleteDominio(y)
                    y = y -1
                y = y +1
            x = x +1
        i = i + 1
    final = time.time()
    tiempo = final - inicio;
    print(f"tiempo de ejecución: {tiempo} segundos")
```

Una vez terminado el algoritmo se muestra en el crucigrama la solución.

ALGORITMO AC3

DEFINICIÓN

El algoritmo AC3 es un algoritmo de inferencia que se usa también en problemas de satisfacción de restricciones. Su objetivo principal es reducir el dominio de las variables de manera más eficiente, eliminando así las inconsistencias de arco que se pudiesen generar.
Las inconsistencias de arco son aquellos valores de una variable que no cumplen con la restricción que viene dada de otra variable.
Esto simplifica los datos para luego trabajar con ellos usando otro algoritmo, genera tiempos de resolución bastante más cortos ya que muchas inconsistencias han sido eliminadas.
El algoritmo termina cuando ya no es posible simplificar más y puede determinar si el problema tiene o no solución según los valores proporcionados.

ALGORITMO APLICADO AL CRUCIGRAMA

En cuanto a la aplicación del algoritmo en nuestro problema, el crucigrama, se desarrollaría de la siguiente manera.
Se tiene un conjunto de variables ya previamente almacenadas al recorrer el crucigrama, estas variables proporcionan su tamaño, sus coordenadas, su orientación y su dominio de valores.
El algoritmo lo que comprueba es que cuando crucen dos palabras en una casilla ambas tengan la misma letra en esa casilla, lo que quiere decir que hayan dos palabras con esa letra justo en esa posición cada una en una orientación distinta.
Va realizando esta comprobación hasta que se simplifique del todo.
Se mostrará si el crucigrama tiene o no solución.

EXPLICACIÓN DEL CÓDIGO

La función empieza con las variables ya guardadas en una lista. Esta lista la va recorriendo y va mirando si para una variable hay otra que la cruce, si hay alguna variable que coincida en coordenadas en algún punto se mira si hay alguna palabra del dominio de esa nueva variable que no se pueda combinar con ninguna de las palabras de esta variable, si existe alguna esta no se añade al dominio por lo que se borra. Si algún dominio queda vacío el problema no tiene solución.
Una vez simplificados los dominios se llama al forward checking para que resuelva el problema pero de manera más ágil y eficaz que si no se hubiesen limpiado los dominios.

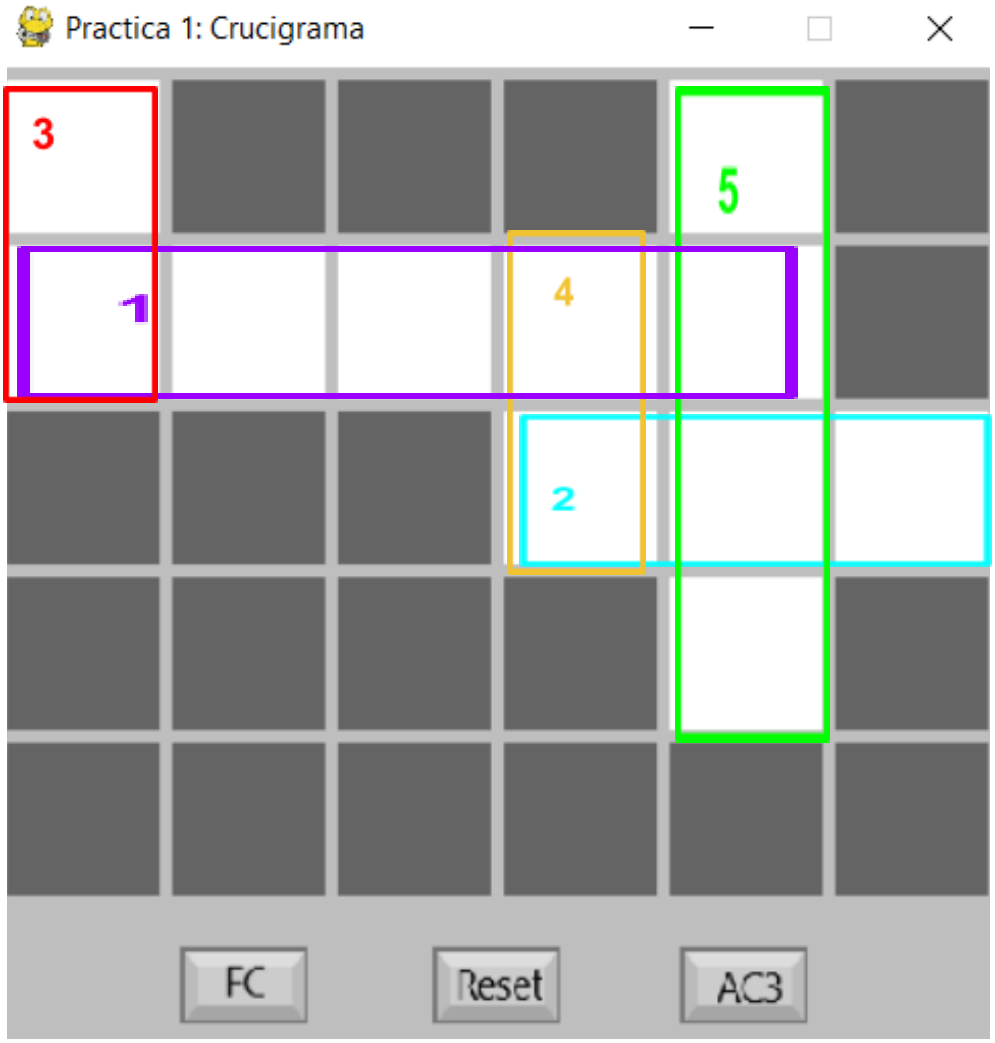
```
def ac3(almacen,COLS,FILS,tablero,res):
    lista = crearVariables(almacen,COLS,FILS,tablero,res)
    print("Dominio antes del ac3")
    for i in range(len(lista)):
            print(f"Nombre {i} Posición {lista[i].getFil()} {lista[i].getCol()} Tipo: {lista[i].getOri()}
Dominio: {lista[i].getDominio()}")
    for i in range(len(lista)):
        print(f"Variable para comprobar {i}")
        hueco = lista[i]
        dominio = hueco.getDominio()
        j = i + 1
```

```
        while j < len(lista):
            coincide = False
            n_dominio = []
            m = 0
            valido = True
            while m < len(dominio):
                sol = dominio[m]
                print(f"Posible valor {sol}")
                for l in range(len(hueco.getCoord())):
                    for p in range(len(lista[j].getCoord())):
                        if hueco.getCoord()[l]==lista[j].getCoord()[p]:
                            coincide = True
                            x = 0
                            while x < len(lista[j].getDominio()):
                                if sol[l] == lista[j].getDominio()[x][p]:
                                        print(f"Se mantiene en el dominio de la variable {j} la palabra
{lista[j].getDominio()[x]}")
                                    n_dominio.append(lista[j].getDominio()[x])
                                x = x + 1
                            if len(n_dominio) == 0:
                                valido = False
                                print(f"La palabra {sol} no es factible, se quita del dominio")
                                hueco.deleteDominio(m)
                                m = m - 1

                m = m + 1
            if coincide == True:
                if len(n_dominio) == 0:
                    res = False
                    return res
                else:
                    lista[j].setDominio(n_dominio)
                    print(f"Nuevo dominio de la variable {j} {n_dominio}")
            j =j +1


    print("Domino después del ac3")
    for i in range(len(lista)):
            print(f"Nombre {i} Posición {lista[i].getFil()} {lista[i].getCol()} Tipo: {lista[i].getOri()}
Dominio: {lista[i].getDominio()}")
    forwardChecking(almacen,COLS,FILS,tablero,res,lista)
```

PROBLEMA PEQUEÑO

EXPLICACIÓN DEL PROBLEMA

Vamos a especificar un problema de tamaño pequeño de 5 variables, dos palabras horizontales y 3 verticales que entre ellas cortan varias con varias.
Para que sea un poco más entretenido he puesto variables de distintos tamaños.
Aquí se muestra como quedaría la cuadrícula y cada variable numerada.

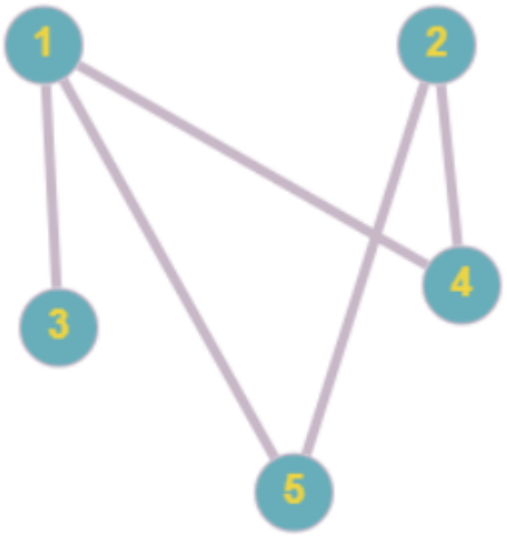


DICCIONARIO UTILIZADO

Para resolver este problema he creado un diccionario pequeño de dos palabras por cada tamaño para que sea muy reducido el campo de búsqueda.
El diccionario es el siguiente:

```
colas osera paro lala l b no re mio era
```

GRAFO DE RESTRICCIONES



Así quedaría el grafo de restricciones ya que las variables 3,4 y 5 cortan con la variable 1 y a la vez las variables 4 y 5 cortan con la variable 2.

TRAZA FORWARD CHECKING

Aquí se muestra paso a paso las decisiones que va tomando el programa según va avanzando. Lo primero que hace es guardar todas las variables que existen y rellena sus dominios con las palabras de longitud correspondiente que hayan.
Una vez hecho eso va variable por variable comprobando con el dominio de las siguientes variables y quitando o no valores de esos dominios.
Si una variable acaba con su dominio vacío se vuelve atrás y se elige otro valor por el que continuar.
En la traza se muestran todos los valores eliminados y como queda su dominio tras esa eliminación.

Variables guardadas
Variable 0 Posición 1 0 Tipo: horizontal Dominio: ['COLAS', 'OSERA']
Variable 1 Posición 2 3 Tipo: horizontal Dominio: ['MIO', 'ERA']
Variable 2 Posición 0 0 Tipo: vertical Dominio: ['NO', 'RE']
Variable 3 Posición 1 3 Tipo: vertical Dominio: ['NO', 'RE']
Variable 4 Posición 0 4 Tipo: vertical Dominio: ['PARO', 'LALA']
Variable a instanciar: 0
Posible valor: COLAS
Se borra del dominio de la variable 2 la palabra NO
Nuevo dominio de la variable 2 ['RE']
Se borra del dominio de la variable 2 la palabra RE
Nuevo dominio de la variable 2 []
El valor COLAS no era factible, seguimos

Andrea Gómez Bobes

Posible valor: OSERA
Se borra del dominio de la variable 2 la palabra RE
Nuevo dominio de la variable 2 ['NO']
Se borra del dominio de la variable 3 la palabra NO
Nuevo dominio de la variable 3 ['RE']
El valor OSERA sí es factible,se muestra en el crucigrama
Variable a instanciar: 1
Posible valor: MIO
Se borra del dominio de la variable 3 la palabra RE
Nuevo dominio de la variable 3 []
El valor MIO no era factible, seguimos
Posible valor: ERA
Se borra del dominio de la variable 4 la palabra LALA
Nuevo dominio de la variable 4 ['PARO']
El valor ERA sí es factible,se muestra en el crucigrama
Variable a instanciar: 2
Posible valor: NO
Para esta variable no habían restricciones sin comprobar
El valor NO sí es factible,se muestra en el crucigrama
Variable a instanciar: 3
Posible valor: RE
Para esta variable no habían restricciones sin comprobar
El valor RE sí es factible,se muestra en el crucigrama
Variable a instanciar: 4
Posible valor: PARO
Para esta variable no habían restricciones sin comprobar
El valor PARO sí es factible,se muestra en el crucigrama

TRAZA AC3

En esta traza se muestran los pasos que va siguiendo el algoritmo para ir vaciando los dominios de manera efectiva.
Primero se empieza guardando las variables existentes y sus posibles valores en sus dominios. Una vez tenemos eso se va comprobando cada variable con aquellas que cortan con ella y se va limpiando el dominio de aquellas palabras que no coincidan en letras con ninguna de las posibilidades de valores de la variable actual.
Se muestra en la traza aquellos valores que sí se mantienen en el dominio y como queda el dominio de cada variable finalmente.
Si algún dominio acaba vacío la palabra no es factible y se elimina del dominio.
En este caso de ejemplo se acaba limpiando del todo el dominio dejando una única combinación posible de variables ya que el diccionario cuenta con pocas palabras.

AC3
Dominio antes del ac3
Nombre 0 Posición 1 0 Tipo: horizontal Dominio: ['COLAS', 'OSERA']
Nombre 1 Posición 2 3 Tipo: horizontal Dominio: ['MIO', 'ERA']

Nombre 2 Posición 0 0 Tipo: vertical Dominio: ['NO', 'RE', 'AS']
Nombre 3 Posición 1 3 Tipo: vertical Dominio: ['NO', 'RE', 'AS']
Nombre 4 Posición 0 4 Tipo: vertical Dominio: ['PARO', 'LALA']
Variable para comprobar 0
Posible valor COLAS
La palabra COLAS no es factible, se quita del dominio
Posible valor OSERA
Se mantiene en el dominio de la variable 2 la palabra NO
Nuevo dominio de la variable 2 ['NO']
Posible valor OSERA
Se mantiene en el dominio de la variable 3 la palabra RE
Nuevo dominio de la variable 3 ['RE']
Posible valor OSERA
Se mantiene en el dominio de la variable 4 la palabra PARO
Se mantiene en el dominio de la variable 4 la palabra LALA
Nuevo dominio de la variable 4 ['PARO', 'LALA']
Variable para comprobar 1
Posible valor MIO
La palabra MIO no es factible, se quita del dominio
Posible valor ERA
Se mantiene en el dominio de la variable 3 la palabra RE
Nuevo dominio de la variable 3 ['RE']
Posible valor ERA
Se mantiene en el dominio de la variable 4 la palabra PARO
Nuevo dominio de la variable 4 ['PARO']
Variable para comprobar 2
Posible valor NO
Variable para comprobar 3
Posible valor RE
Variable para comprobar 4
Dominio después del ac3
Nombre 0 Posición 1 0 Tipo: horizontal Dominio: ['OSERA']
Nombre 1 Posición 2 3 Tipo: horizontal Dominio: ['ERA']
Nombre 2 Posición 0 0 Tipo: vertical Dominio: ['NO']
Nombre 3 Posición 1 3 Tipo: vertical Dominio: ['RE']
Nombre 4 Posición 0 4 Tipo: vertical Dominio: ['PARO']

RESULTADO PROBLEMA PEQUEÑO

Aquí se muestra como quedaría el crucigrama tras la resolución del mismo.

EXPERIMENTACIÓN

En este caso el problema tiene resultado y lo han mostrado ambos algoritmos. El diccionario creado contaba con dichas palabras que encajaban a la perfección en cada hueco y por tanto el resultado se muestra.

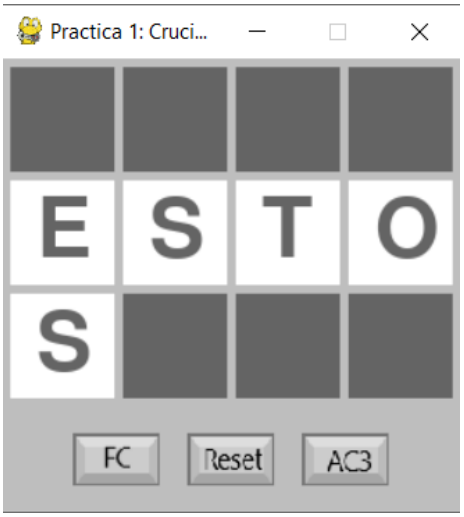
En este caso el algoritmo AC3 ya nos ha informado de que el crucigrama no tiene solución y por tanto no se llama a Forward Checking. El aviso decidí dejar el mismo ya que supuse que la solución debía ser igual pero en este caso el algoritmo AC3 ya descarta que haya solución.

ESTUDIO DE TIEMPOS

En este apartado vamos a ir modificando ciertas características del problema para ver cómo se comportan los algoritmos y la velocidad que estos llevan.

CRUCIGRAMA PEQUEÑO

En este caso creamos un crucigrama de 4x3 bastante pequeño y se tarda muy poco en resolver, no es tan notable la rapidez del algoritmo AC3 frente a la del Forward Checking.



AC3 : 0,001009 segundos
FC: 0,001034 segundos

CRUCIGRAMA PROMEDIO

En este caso creamos un crucigrama de tamaño estándar, 6x65, aquí se empieza a notar un poco más la diferencia siendo el AC3 más rápido.

AC3:0,0043 segundos
FC:0,0054 segundos

CRUCIGRAMA GRANDE

En este caso creamos un crucigrama bastante más grande de 9x8, aquí ya se ve notablemente la diferencia en velocidad cuando se usa AC3 respecto a sólo usar Forward Checking.

Andrea Gómez Bobes

AC3 :0,007 segundos
FC: 0.018 segundos

50 PALABRAS DICCIONARIO

En este caso empezamos con un diccionario de 50 palabras, se puede notar la diferencia en velocidad pero ambos van bastante rápido. En todos los casos de aumento de palabras del diccionario se solicitó a chatgpt que generase una lista de la cantidad de palabras indicadas de manera aleatoria con palabras de longitudes distintas.

AC3:0,0015 segundos
FC:0,002 segundos

100 PALABRAS DICCIONARIO

Aumentamos el número de palabras a 100, sigue siendo bastante rápido pero se puede notar que el algoritmo AC3 lo hace 3 veces más rápido.

Andrea Gómez Bobes

AC3:0,002
FC:0,006

200 PALABRAS EL DICCIONARIO

Para finalizar, usamos un diccionario de 200 palabras, aquí ya se nota que cuesta un poco más de solucionar, siendo igualmente rápido. Se observa que el algoritmo AC3 sigue siendo mucho más rápido y eficaz.

AC3: 0,003 segundos
FC:0,004 segundos

CONCLUSIÓN

En esta gráfica vemos la comparación en cuanto a tamaño, como se puede observar el algoritmo AC3 agiliza bastante el proceso cuanto más grande sea el crucigrama.

Andrea Gómez Bobes

Como podemos observar en la segunda tabla que compara los tamaños de los diccionarios en este se nota que el AC3 agiliza el proceso pero que también depende de cuántas palabras haya que encajen en el hueco y que tan fácil sea encontrarlas.

En general, se observa claramente cómo el algoritmo AC3 mejora en tiempo y esfuerzo el proceso del algoritmo Forward Checking devolviendo resultados mucho más rápidos.

LIMITACIONES

Me parece importante añadir este punto comentando en mi caso aquellas cosas que no he logrado implementar y por tanto el programa no funciona a la perfección.

Lo más importante y principal es que no he llegado a implementar un backtracking en Forward Checking una vez una nos damos cuenta de que se queda sin solución cuando sigue por ese camino, por tanto sí puede retroceder a la acción anterior y la decisión anterior pero no llega a acceder a varias elecciones atrás para enmendar el error. Es algo que me planteé realizar ya más tarde y no conseguí desarrollar de manera efectiva y por tanto no se encuentra incorporado en el código.

Adicionalmente, no conseguí poner que Forward Checking indique cuando no hay solución, no me dió tiempo a hacerlo ya que me di cuenta tarde y tenía que cambiar varias cosas del código para ello. Por tanto, Forward Checking siempre saca una solución aunque sea errónea y no exista dicha solución.

En general ha sido una práctica muy entretenida pero sí han habido detalles que no he sabido desarrollar y quería comentarlos para que se supiese que los tuve en cuenta.