# Lab 1

Rodrigo Cayazaya Marín

Rol: 201773538-4

1. (5 puntos) Explique qué procesos debe realizar antes de aplicar cualquiera de las técnicas solicitadas para este laboratorio. Detalle el efecto en el problema a resolver tras aplicar estos procesos.

Primero transformo el sudoku en una matriz tipo lista:

```
# sudoku = open("Sudoku01.txt","r")
sudoku = open("SudokuE.txt","r")
matriz = []
for linea in sudoku:
    for valor in linea:
        if (valor != " " and valor != "\n"):
            matriz.append(valor)
```

Luego inicializo los arcos, dominios, nodos y restricciones:

```
arcos = []
restricciones = {}
nodos = ['0','1','3','4','5','6','9','10','11','12','13','15']
dominios = {
    '0': [1, 2, 3, 4],
    '1': [1, 2, 3, 4],
    '2': [1, 2, 3, 4],
    '3': [1, 2, 3, 4],
    '4': [1, 2, 3, 4],
    '5': [1, 2, 3, 4],
    '6': [1, 2, 3, 4],
    '7': [1, 2, 3, 4],
    '8': [1, 2, 3, 4],
    '9': [1, 2, 3, 4],
    '10': [1, 2, 3, 4],
    '11': [1, 2, 3, 4],
    '12': [1, 2, 3, 4],
    '13': [1, 2, 3, 4],
    '14': [1, 2, 3, 4],
    '15': [1, 2, 3, 4]
}
```

Luego reviso por filas para agregar los arcos y dominios (nodo consistencia), también elimino los dominios de las pistas:

```python
def filas(matriz,arcos,dominios):
    variableFila1 = []
    variableFila2 = []
    variableFila3 = []
    variableFila4 = []

    pistaFila1 = []
    pistaFila2 = []
    pistaFila3 = []
    pistaFila4 = []

    for i in range(len(matriz)):
        valor = matriz[i]
        fila = (i)//4
        if(valor == '0'):
            if(fila == 0):
                variableFila1.append(i)
            elif(fila == 1):
                variableFila2.append(i)
            elif(fila == 2):
                variableFila3.append(i)
            else:
                variableFila4.append(i)
        else:
            del dominios[str(i)]
            if(fila == 0):
                pistaFila1.append(valor)
            elif(fila == 1):
                pistaFila2.append(valor)
            elif(fila == 2):
                pistaFila3.append(valor)
            else:
                pistaFila4.append(valor)

    agregar_arco(variableFila1,variableFila2,variableFila3,variableFila4,arcos)
    nodo_consistencia(pistaFila1,pistaFila2,pistaFila3,pistaFila4,dominios,range(4),range(4,8),range(8,12),range(12,16)
```

Agregar arcos

```
def agregar_arco(lista1,lista2,lista3,lista4,arcos):
    for i1 in range(len(lista1)):
        for i11 in range(i1+1,len(lista1)):
            if((str(lista1[i1]),str(lista1[i11])) not in arcos):
                arcos.append((str(lista1[i1]),str(lista1[i11])))
                arcos.append((str(lista1[i11]),str(lista1[i1])))

    for i1 in range(len(lista2)):
        for i11 in range(i1+1,len(lista2)):
            if((str(lista2[i1]),str(lista2[i11])) not in arcos):
                arcos.append((str(lista2[i1]),str(lista2[i11])))
                arcos.append((str(lista2[i11]),str(lista2[i1])))

    for i1 in range(len(lista3)):
        for i11 in range(i1+1,len(lista3)):
            if((str(lista3[i1]),str(lista3[i11])) not in arcos):
                arcos.append((str(lista3[i1]),str(lista3[i11])))
                arcos.append((str(lista3[i11]),str(lista3[i1])))

    for i1 in range(len(lista4)):
        for i11 in range(i1+1,len(lista4)):
            if((str(lista4[i1]),str(lista4[i11])) not in arcos):
                arcos.append((str(lista4[i1]),str(lista4[i11])))
                arcos.append((str(lista4[i11]),str(lista4[i1])))
```

Nodo consistencia:

```
def nodo_consistencia(lista1,lista2,lista3,lista4,dominios,l1,l2,l3,l4):
    if(lista1 != None):
        for i1 in l1:
            if(str(i1) in dominios):
                for pista in lista1:
                    if(int(pista) in dominios[str(i1)]):
                        lista_updated = dominios[str(i1)]
                        lista_updated.remove(int(pista))
                        dominios[str(i1)] = lista_updated
    if(lista2 != None):
        for i1 in l2:
            if(str(i1) in dominios):
                for pista in lista2:
                    if(int(pista) in dominios[str(i1)]):
                        lista_updated = dominios[str(i1)]
                        lista_updated.remove(int(pista))
                        dominios[str(i1)] = lista_updated
    if(lista3 != None):
        for i1 in l3:
            if(str(i1) in dominios):
                for pista in lista3:
                    if(int(pista) in dominios[str(i1)]):
                        lista_updated = dominios[str(i1)]
                        lista_updated.remove(int(pista))
                        dominios[str(i1)] = lista_updated
    if(lista4 != None):
        for i1 in l4:
            if(str(i1) in dominios):
                for pista in lista4:
                    if(int(pista) in dominios[str(i1)]):
                        lista_updated = dominios[str(i1)]
                        lista_updated.remove(int(pista))
                        dominios[str(i1)] = lista_updated
```

Se repite lo mismo, pero para las columnas:

```python
def columnas(matriz,arcos):
    variableColumna1 = []
    variableColumna2 = []
    variableColumna3 = []
    variableColumna4 = []

    pistaColumna1 = []
    pistaColumna2 = []
    pistaColumna3 = []
    pistaColumna4 = []

    for i in range(len(matriz)):
        valor = matriz[i]
        columna = (i)%4
        if(valor == '0'):
            if(columna == 0):
                variableColumna1.append(i)
            elif(columna == 1):
                variableColumna2.append(i)
            elif(columna == 2):
                variableColumna3.append(i)
            else:
                variableColumna4.append(i)
        else:
            if(columna == 0):
                pistaColumna1.append(valor)
            elif(columna == 1):
                pistaColumna2.append(valor)
            elif(columna == 2):
                pistaColumna3.append(valor)
            else:
                pistaColumna4.append(valor)

    agregar_arco(variableColumna1,variableColumna2,variableColumna3,variableColumna4,arcos)
    nodo_consistencia(pistaColumna1,pistaColumna2,pistaColumna3,pistaColumna4,dominios,[0,4,8,12],[1,5,9,13],[2,6,10,1
```

Y para las submatrices:

```python
def submatrices(matriz,arcos):
    variableSubMatriz1 = []
    variableSubMatriz2 = []
    variableSubMatriz3 = []
    variableSubMatriz4 = []

    pistaSubMatriz1 = []
    pistaSubMatriz2 = []
    pistaSubMatriz3 = []
    pistaSubMatriz4 = []

    for i in range(len(matriz)):
        valor = matriz[i]
        subMatriz = 2*(i//8) + (i%4)//2
        if(valor == '0'):
            if(subMatriz == 0):
                variableSubMatriz1.append(i)
            elif(subMatriz == 1):
                variableSubMatriz2.append(i)
            elif(subMatriz == 2):
                variableSubMatriz3.append(i)
            else:
                variableSubMatriz4.append(i)
        else:
            if(subMatriz == 0):
                pistaSubMatriz1.append(valor)
            elif(subMatriz == 1):
                pistaSubMatriz2.append(valor)
            elif(subMatriz == 2):
                pistaSubMatriz3.append(valor)
            else:
                pistaSubMatriz4.append(valor)
    nodo_consistencia(pistaSubMatriz1,pistaSubMatriz2,pistaSubMatriz3,pistaSubMatriz4,dominios,[0,1,4,5],[2,3,
    agregar_arco(variableSubMatriz1,variableSubMatriz2,variableSubMatriz3,variableSubMatriz4,arcos)
```

Finalmente se agregan las restricciones en base a los arcos:

```
def definir_restricciones(restricciones,arcos):
    for arco in arcos:
        restricciones[arco] = lambda a, b: a != b
```

Ya que tengo los arcos, nodos, restricciones y dominios definidos, puedo aplicar los algoritmos sin preocuparme de las restricciones unarias.

2. (10 puntos) Explique el efecto de aplicar AC-3 al problema asignado

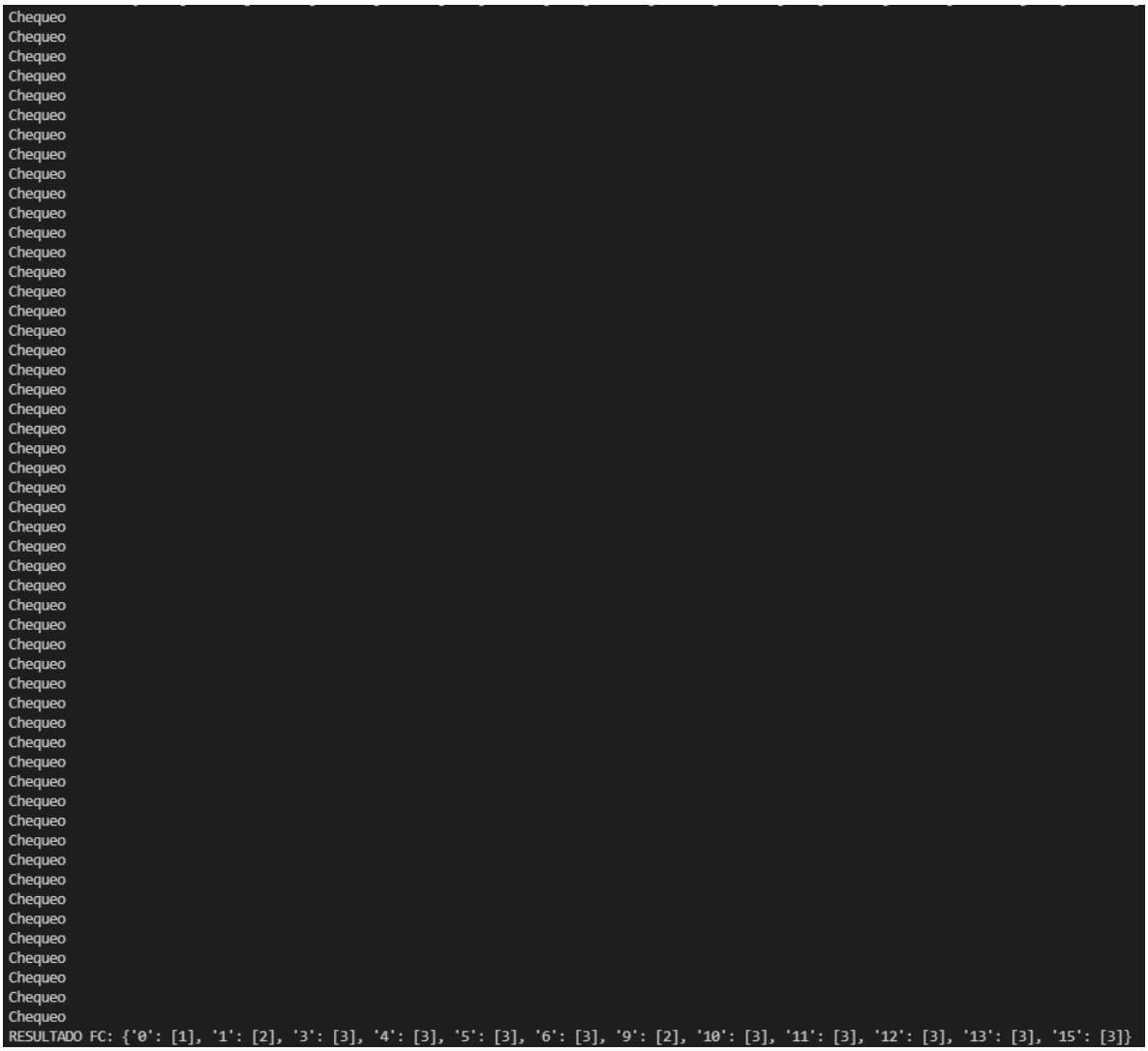
Debido a que la cantidad de evidencia es pequeña (2 y 4), el dominio no se restringe lo suficiente para modificarse al realizar arco consistencia, es por ello que el dominio no se ve afectado, ya que siempre existirá un valor de soporte entre los dominios de los nodos.

```
Dominio Original: {'0': [1, 2, 3], '1': [1, 2, 3], '3': [1, 3], '4': [1, 3], '5': [1, 3, 4], '6': [1, 3], '9': [1, 2, 3], '10': [1, 3], '11': [1, 3], '12': [1, 3], '13': [1, 3], '15': [1, 3, 4]}
RESULTADO AC3: {'0': [1, 2, 3], '1': [1, 2, 3], '3': [1, 3], '4': [1, 3], '5': [1, 3, 4], '6': [1, 3], '9': [1, 2, 3], '10': [1, 3], '11': [1, 3], '12': [1, 3], '13': [1, 3], '15': [1, 3, 4]}
```

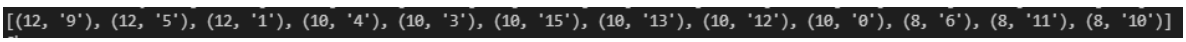
3. (20 puntos) Compare el desempeño de Forward Checking (FC) y Graph-based Backjumping (GBJ) para encontrar una solución al problema asignado

- FC
  Instanciaciones: 12
  Chequeos: 52
  Saltos: 0

```
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
Instanciación
RESULTADO FC: {'0': [1], '1': [2], '3': [3], '4': [3], '5': [3], '6': [3], '9': [2], '10': [3], '11': [3], '12': [3], '13': [3], '15': [3]}
```




4. (30 puntos) Evalúe el efecto de aplicar una heurística de ordenamiento de variables en el desempeño de FC y GBJ (para encontrar una solución), dando prioridad a las variables más conectadas del problema.




El orden sería: x9,x5,x1,x4,x3,x15,x13,x12,x0,x6,x11,x10

5. (35 puntos) Evalúe el efecto de aplicar una heurística de ordenamiento dinámico de variables en el desempeño de FC y GBJ (para encontrar una solución), dando prioridad a aquellas variables cuyos dominios tengan menos elementos.