ALGORITMO PARA COMPROBAR SI LA OPERACIÓN DE UN CONJUNTO ES UN CUADRO LATINO O UN GRUPO

Matemáticas Discretas 2: 2023-1 | Universidad Nacional De Colombia

Juan Carlos Garavito Higuera

El presente reto se nos muestra para facilitar el análisis de tablas de tablas de Cayley, analizando y evaluando las distintas propiedades de cuadro latino y grupo dictadas según la definición de las mismas, es por esto que se realiza el algoritmo a manera de código aplicable para optimizar el proceso.

```
import numpy as np                  #Libreria de manejo matemático y
matrices multidimencionales.
import itertools                    #Libreria de manejo de combinaciones
y permutaciones.
```

DESARROLLO DE LA SOLUCIÓN DEL PROBLEMA.

El acercamiento realizado consiste en un programa de tipo cascada en el que al realizar una prueba esta falla cancela las demás pruebas y salta a la siguiente evaluación para ayudar con la complejidad y tiempos de ejecución. En este orden de ideas lo primero que debemos tener en cuenta en la definición de grupos son las propiedades que estos cargan consigo, podemos verlas en este orden (El cual está previsto de menor a mayor requerimiento computacional):

- La operación debe ser cerrada.
- Deber haber la existencia del elemento neutro.
- Debe haber la existencia de elementos inversos para los elementos del conjunto.
- La operación debe ser asociativa.

Como se mencionó antes y por la naturaleza del programa, si no se cumple alguna de estas propiedades podemos descartar que la matriz sea un grupo, y podemos empezar a analizar si la matriz es un cuadrado latino.

```
Operación cerrada > Elemento neutro > Elementos inversos >
Asociatividad
```

Este orden tiene sentido, ya que de igual manera, no podemos verificar la existencia de elementos inversos sin haber definido el elemento neutro.

Comenzamos por ingresar el conjunto, y su matriz asociada a comprobar.

```
Conjunto =np.array(['e','g1','g2','g3','g4','g5'])  # Definimos un
arreglo con los elementos que representan el conjunto
Matriz = np.array([                                 # Definimos una
matriz con los elementos dentro de la tabla de cayley
            [ 'e','g1','g2','g3','g4','g5'],
            ['g1', 'e','g3','g4','g5','g2'],
            ['g2','g3', 'e','g5','g1','g4'],
```

```
            ['g3','g4','g5', 'e','g2','g1'],
            ['g4','g5','g1','g2', 'e','g3'],
            ['g5','g2','g4','g1','g3','e']
        ])
```

Declaramos las banderas donde se guardan las propiedades de nuestra matriz, todas inicializadas en falso.

```
# Se crean variables bandera que sirven para determinar y analizar los
resultados de las pruebas.

EsCerrada=False
HayNeutro=False
HayInversos=False
EsAsociativa=False
```

El primer paso es comprobar si la operación es cerrada. Para esto, simplemente verificamos que cada elemento de la matriz, esté dentro del conjunto. De esto se encarga la función VerificarCerrada().

```
def VerificarCerrada(Conjunto, Matriz):                    #Se toman
de parámetros el conjunto y matriz instanciados anteriormente.
    tamMatriz = len(Matriz)
    for i in range(tamMatriz):
        for j in range(tamMatriz):                          #Se accede
a la matriz dada.
            if (Matriz[i][j] not in Conjunto):              #Revisa si
los elementos de la matriz no están dentro del conjunto base.
                return False                                #Retorna
falso (booleano) si no es cerrada.
    return True                                             #Retorna
verdadero (booleano) si es cerrada.
```

Para el elemento neutro, basta con verificar que exista un elemento cuya fila y columna correspondiente, coincida con la definición original del conjunto. De esto se encarga la función CalcularNeutro().

```
def CalcularNeutro(Conjunto,Matriz):

    Neutrofila=[False,None]
    Neutrocolumna=[False,None]
    TamMatriz = len(Matriz)                                 #Es el tamaño de
la matriz
    MatTranspuesta = Matriz.transpose()                     #Entrega la
matriz transpuesta según la original.

    for i in range(TamMatriz):
        if(np.array_equal(Matriz[i],Conjunto)):
#Revisa si la fila de la matriz es equivalente al conjunto inicial
dado.
            Neutrofila[0]=True
```

```
            Neutrofila[1]=Conjunto[i]
#Guarda el posible elemento neutro de las filas.

        if(np.array_equal(MatTranspuesta[i],Conjunto)):
#Revisa si la fila de la matriz transpuesta (columna original)...
            Neutrocolumna[0]=True                                   #es
equivalente al conjunto inicial dado.
            Neutrocolumna[1]=Conjunto[i]
#Guarda el posible elemento neutro de las columnas.

        if (Neutrofila[0] and Neutrocolumna[0] and
(Neutrofila[1]==Neutrocolumna[1])):
            return True, Neutrofila[1]                              #Si
encuentra el elemento neutro en fila y columna lo retorna.

    return False, None
#Retorna falso (booleano) y none si no no encuentra el elemento
neutro.
```

Para los elementos inversos, buscamos por cada elemento, otro que multiplicado con el inicial por izquierda nos de el identidad, y verificamos que esto sea conmutativo.

Buscamos el elemento neutro de la matriz, si la operación es cerrada. De lo contrario, no.

```
EsCerrada = VerificarCerrada(Conjunto, Matriz)
if (EsCerrada):
#Si la matriz es cerrada cambiar la bandera de HayNeutro a True
    HayNeutro, Neutro = CalcularNeutro(Conjunto,Matriz)             #Y
guarda el elemento Neutro en la variable del mismo nombre.
```

BuscarInverso() es la función que nos permite revisar los inversos dentro de la tabla de Cayley. Esta función realiza su tarea bajo la definición del elemento inverso diciendo que si existe, la operación del mismo con el elemento original debe dar el elemento neutro.

```
#Busca el inverso de cada elemento del conjunto.
def BuscarInverso(NumElemento,Conjunto,Matriz,Neutro):
    MatTranspuesta = Matriz.transpose()
#Transpone la Matriz base
    InversoFila=None
    InversoColumna=None
    if (Neutro in Matriz[NumElemento]):
#Busca si el Elemento neutro está en el índice dado en las filas
        InversoFila=Conjunto[np.where(Matriz[NumElemento]==Neutro)]
#np.where busca en la matriz el elemento neutro y luego trae...
    if (InversoFila):
#cuál sería ese elemento.
        if (Neutro in MatTranspuesta[NumElemento]):
```

```
InversoColumna=Conjunto[np.where(MatTranspuesta[NumElemento]==Neutro)]
#Busca si el Elemento neutro está en el índice dado en las columnas
        if(InversoFila==InversoColumna):     #Revisa si son iguales
los inversos encontrados en la posición puesto que los inversos son
asociativos.
            return True
    return False      #Si no hay algún elemento neutro entonces el
inverso no puede existir por la definición de si mismo.

#Verifica la unicidad de los elementos inversos para cualquier
elemento dentro de la operación.
def VerificarInversos(Conjunto,Matriz,Neutro):
    for numelemento in range(len(Conjunto)):
        if (not BuscarInverso(numelemento,Conjunto,Matriz,Neutro)):
            return False
    return True
```

Verificamos que cada elemento dentro de la tabla tenga un inverso.

```
if (HayNeutro):
    HayInversos=VerificarInversos(Conjunto,Matriz,Neutro) #Si
encuentra un Neutro ejecuta la función HayInversos.
```

Para la asociatividad, verificamos si todas las combinaciones de operaciones de tres elementos posibles del conjunto al analizarlas independientemente del orden de ejecución de la operación dan el mismo resultado. Si se encuenta una operación de tres elementos en la que no se presenta la asociatividad la matriz devuelve un False (booleano) indicando el no cumplimiento de las propiedades de un grupo.

```
def IsAsociative(Conjunto,Matriz):
    n = len(Conjunto)
    lista=[]
    for i in range(n):
      lista.append(i)
    result = []
    for i in range(n-2):
        for j in range(i+1, n-1):
            for k in range(j+1, n):
                result.append([lista[i], lista[j], lista[k]])
# encuentra los 3 elementos

    ban=True
    posibilidades=result
    for i in range(len(posibilidades)):
      resultado1=np.where(Conjunto==Matriz[posibilidades[i][1]]
[posibilidades[i][2]])[0][0]       # ""     ""      ""       ""
      resultado2=np.where(Conjunto==Matriz[posibilidades[i][0]]
[posibilidades[i][1]])[0][0]       # calcula las dos operaciones
```

```
      if(Matriz[(posibilidades[i][0])][resultado1]!=Matriz[resultado2]
[(posibilidades[i][2])]):   # revisa si no son equivalentes
        ban=False
# retorna Falso (booleano) en caso de no ser Asociativa
        i=len(posibilidades)-1
    return ban
# retorna Verdadero (booleano) en caso de ser Asociativa
```

Verificamos que la operación sea asociativa, si cada elemento tiene inverso. De lo contrario, pasamos a verificar si la tabla es un cuadrado latino.

```
if (HayInversos):                          #Si encuentra los inversos
ejecuta la función EsAsociativa.
    EsAsociativa = IsAsociative(Conjunto,Matriz)
```

Si estas tres comprobaciones se cumplen, podemos decir que la matriz inicial es un grupo (Y a su vez, un cuadrado latino). Si no, procedemos a verificar si la matriz es un cuadro latino o no.

```
if (EsCerrada and HayNeutro and HayInversos and EsAsociativa):
    print("El conjunto más la operación definida por la tabla de
Cayley ingresadas son un grupo, y un cuadrado latino.")
else:
    print("El conjunto más la operación definida por la tabla de
Cayley ingresadas NO son un grupo")
    EsGrupo=False
```

Si identificamos en el paso anterior que la matriz NO es un grupo, procedemos a verificar si es un cuadrado latino o no. De esto se encarga la función LatinSquareBool() que recibe únicamente la matriz correspondiente y retorna un dato de tipo booleano confirmando o desmintiendo si es un cuadrado latino.

```
# Proceso para verificar si es un cuadrado latino o no
def LatinSquareBool(Matriz):
  LatinSquare=True
  result = []
  cuenta=0
  for item in Matriz[0]:
      if item not in result:
          result.append(item)                      #añade los
elementos que no encuentre en la fila
      else:
        cuenta+=1
  if cuenta>=1:
    LatinSquare=False                              #si hay algún
elemento repetido nos indica que no es un cuadrado latino
  else:
```

```
    permutations = list(itertools.permutations(result))      #Crea las
permutaciones posibles de lo que se guardó en la lista
    filas=np.shape(Matriz)[0]                                #tamaño
de la fila
    Trans=np.transpose(Matriz)
#transpone la matriz
    permutaciones= list(map(list,permutations))              #Añade
las permutaciones a una lista.
    for i in range(filas):

      if Matriz[i].tolist() not in permutaciones or Trans[i].tolist()
not in permutaciones:
        LatinSquare=False                    #revisa si las filas o
columnas no están en las posibles permutaciones,

  return LatinSquare                         #si esto ocurre, no es
un cuadrado latino.
```

Procedemos a la muestra de resultados, de acuerdo con los resultados analizados en los pasos anteriores, para que sean interpletables por cualquier persona.

```
#Analisis y muestra de resultados.

if(LatinSquareBool(Matriz)):
    print("La matriz ingresada es un cuadrado latino")
else:
  print("La matriz ingresada no es un cuadrado latino")
```

Bibliografía

https://marcelgoh.ca/2018/10/06/cayley-tables.html

Clifford, Alfred Hoblitzelle; Preston, Gordon Bamford (1961). The algebraic theory of semigroups. Vol. I. Mathematical Surveys, No. 7. Providence, R.I.: American Mathematical Society. ISBN 978-0-8218-0272-4. MR 0132791. (pp. 7–9)

https://gist.github.com/jfinkels/c33681e7f7b54421ea02