In [1]:

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from numpy import linalg as LA
from collections import defaultdict
import numpy as np
import random
from graphrole import RecursiveFeatureExtractor, RoleExtractor
import multiprocessing as mp
import itertools
from numpy import linalg as LA
from os import cpu_count
import time
```

Ejercicio 1

Se quiere convocar a una elección a la que se presentan 4 candidatos (A, B, C y D). Hay 3 votantes del jurado que tienen sus siguientes rankings individuales:

• Jurado 1: B -> C -> D -> A

• Jurado 2: C -> D -> A -> B

• Jurado 3: D -> A -> B -> C

a. ¿Quién ganaría por eliminación iterativa?

b. ¿Quién ganaría por Borda rule?

c. Suponé que estás a cargo de definir las reglas/formato de la votación, y sos un miembro corrupto que desea que si o si gane la alternativa A (te asegura favores si logra ganar la elección). Definir (si existe) un sistema de votación en el cual A resulte ganador de la elección. En caso de no existir, explicar por qué. ¿Cuál propiedad deseable de los sistemas de votación no se está cumpliendo si, efectivamente, ganara A?

a) Veo que en este caso no tiene sentido hacer el codigo ya que los candidatos B, C y D obtienen 1 punto cada uno por parte de los votantes del jurado. En este caso habria empate en la eleccion y habria que desempatar de alguna manera.

b) Programo el algoritmo de Borda Rule para encontrar al ganador.

In [2]:

```
# precondicion: todos los jurados votaron a todos los candidatos.

jurado_1 = ['B','C','D','A']
jurado_2 = ['C','D','A','B']
jurado_3 = ['D','A','B','C']

jurados = []
jurados.append(jurado_1)
jurados.append(jurado_2)
jurados.append(jurado_3)

def borda_rule(jurados, largo):
    votos_A = 0
    votos_B = 0
    votos_C = 0
    votos_D = 0

    for i in range(len(jurados)):
        for j in range(largo):
                if jurados[i][j] == 'A':
                    votos_A += ((largo - 1) - j)
                if jurados[i][j] == 'B':
                    votos_B += ((largo - 1) - j)
                if jurados[i][j] == 'C':
                    votos_C += ((largo - 1) - j)
                if jurados[i][j] == 'D':
                    votos_D += ((largo - 1) - j)

    dict_final = {}
    dict_final[votos_A] = 'A'
    dict_final[votos_B] = 'B'
    dict_final[votos_C] = 'C'
    dict_final[votos_D] = 'D'

    if  (votos_A != votos_B) and (votos_A != votos_C) and (votos_A != votos_D)\
    and (votos_B != votos_C) and (votos_B != votos_D) and (votos_D != votos_C):
        print("El ganador es el candidato", dict_final.get(max(votos_A, votos_B, votos_C, votos_D)))
    else:
        print("El criterio devuelve un empate")

borda_rule(jurados, len(jurados[0]))
```

```
El ganador es el candidato D
```

c) Para que gane A, lo que se puede hacer es un sistema de votacion de eliminaciones sucesivas pero en orden alfabeticamente inverso. En el primer paso, se compararia el candidato D con el C, donde este ultimo ganara. Luego, se comparara con B al ganador y sera B quien se lleve la victoria. Llegando asi, al ultimo paso donde se enfrentan B y A y se obtendra al candidato A como el ganador de la votacion.

La propiedad que no se estaría cumpliendo es que un sistema de votacion debe ser *Pareto-Eficiente*. Esto significa que si todos los jueces estan de acuerdo en el orden relativo de preferencia de dos alternativas, el sistema de votacion debe elegir ese mismo orden. Entonces, al estar cumpliendose en todos los casos que el candidato D esta siendo preferido antes que el candidato A, el orden se respetaria y el sistema nunca dejaria que gane ese candidato.

Ejercicio 2

Considerando el modelo de cascadas de información visto en clase, supongamos que hay una nueva tecnología que los individuos pueden optar por aceptar o rechazar. Supongamos que cada uno que acepta la tecnología recibe una ganancia positiva o negativa (sin conocerla a priori). Estos valores son aleatorios para cada nodo, y si la tecnología es “Buena”, entonces el promedio será positivo, y si la tecnología es “Mala” el promedio será negativo (esta información es conocida por los individuos). Quienes rechacen la tecnología reciben ganancia 0. En este modelo, cuando a un individuo le toca elegir si acepta o rechaza la nueva tecnología, recibe la información de las ganancias de todos los que vinieron antes.

a. Supongamos que esta nueva tecnología es, en realidad, “Mala”. ¿Cómo afecta esta nueva información (qué ganancia tuvo cada uno de los que vinieron antes) a la potencial formación de una cascada para que persista la nueva tecnología? (No es necesario dar una demostración, simplemente argumentar)

b. Supongamos que esta nueva tecnología es, en realidad, “Buena”. ¿Puede surgir una cascada de rechazo de esta nueva tecnología?

a) Al conocer que esta nueva tecnologia es "Mala" se puede determinar que cada vez que venga un individuo nuevo a adquirirla este tendera a rechazarla. Esto se debe a que la persona preferira no arriesgarse a perder dinero con la nueva tecnologia viendo que, en promedio, las ganancias de los acredores previos de la tecnologia fueron negativas. En conclusion, la tecnologia no persistira ya que cada persona antes de tomar la decision se basara en el promedio de las ganancias.

b) Si la tecnologia es "Buena" es muy probable que no se genere una cascada de rechazo. Ya que ahora depende puramente en lo que piensa el sujeto sobre las tecnologias. Lo que sucede tambien es que el aceptar esta claramente influenciado por los resultados de los demas, ya que al ver que los demas obtuvieron ganancias aceptando tenderan a repetir el comportamiento en busca de las mismas ganancias.

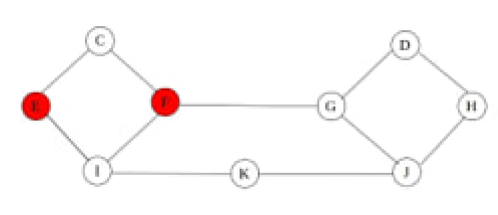
Ejercicio 3

Considerá la siguiente red, suponiendo que todos los nodos tienen inicialmente un comportamiento B. Cada nodo puede cambiar al comportamiento A si al menos la mitad de sus vecinos tiene dicho comportamiento.

a. Supongamos que los nodos E y F son early adopters del comportamiento A. Si los demás nodos siguen la regla del umbral (threshold) para adherir a este nuevo comportamiento, ¿qué nodos implementarían el comportamiento A?

b. Explicar a qué se debe que el comportamiento A no se propaga a través de toda la red en el escenario del punto (a). ¿Qué característica de la red lo impide? (responder a esta pregunta no apuntando a nodos particulares sino a presencias de ciertas características) ¿Dónde más tendría que haber otro early adopter de A sí o sí para que el comportamiento se propague a través de toda la red?

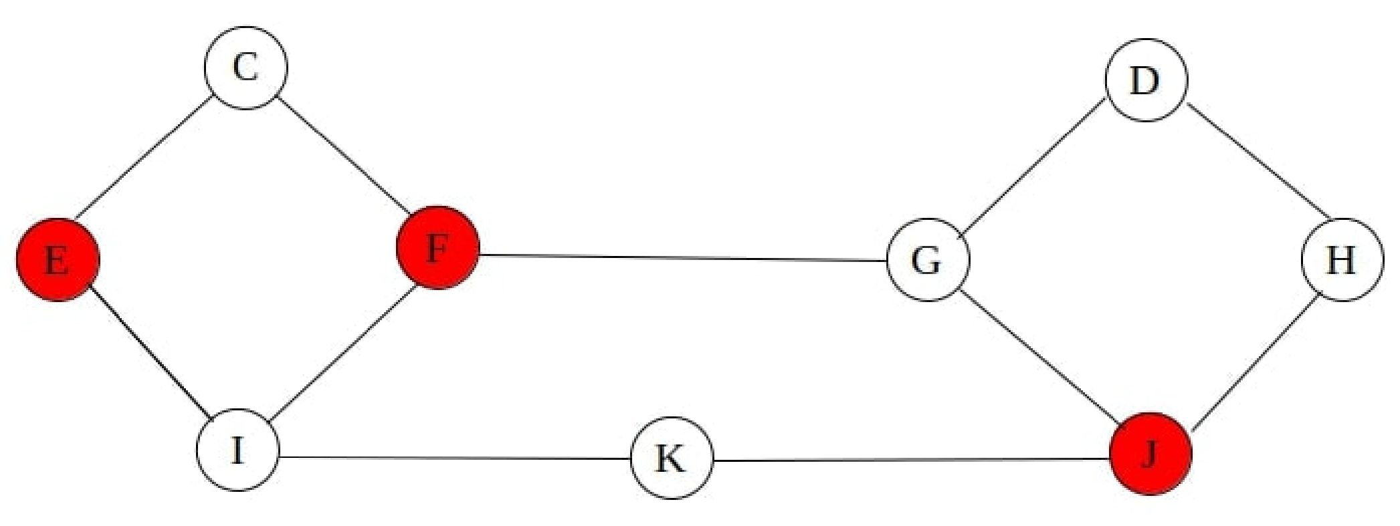
a) Si se marcan en rojo los nodos que se adhieren al comportamiento de A, la red quedaria:



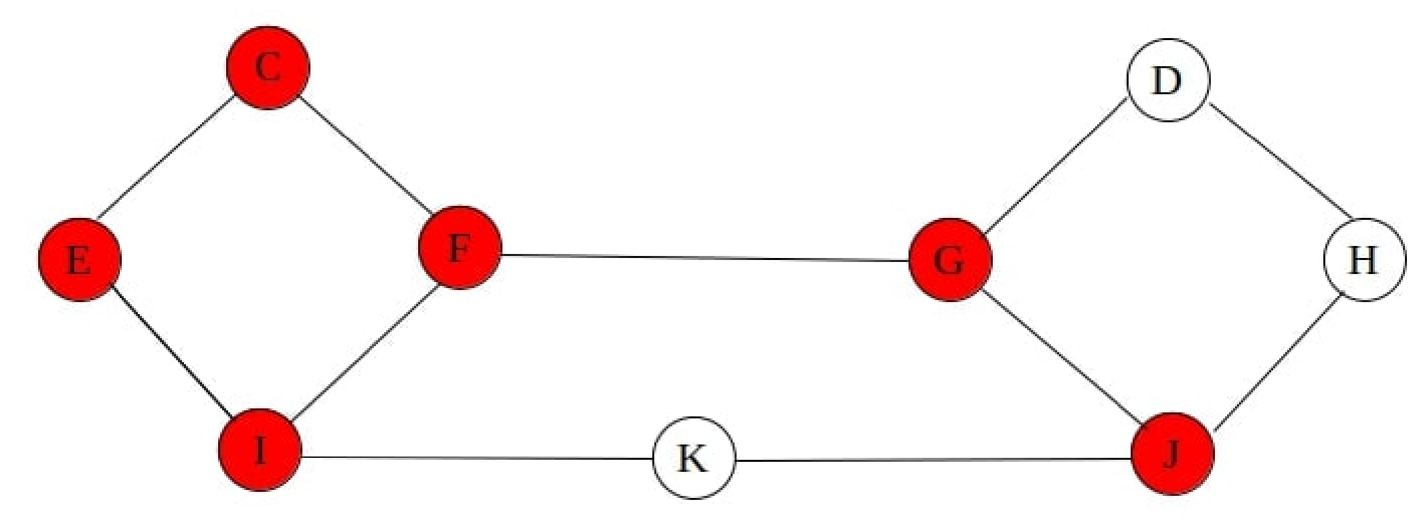
I, K y C se adhieren ya que tienen mas de la mitad de los nodos adyacentes con el comportamiento A. J y G no llegan a esa condicion por lo que se quedan con su comportamiento anterior.

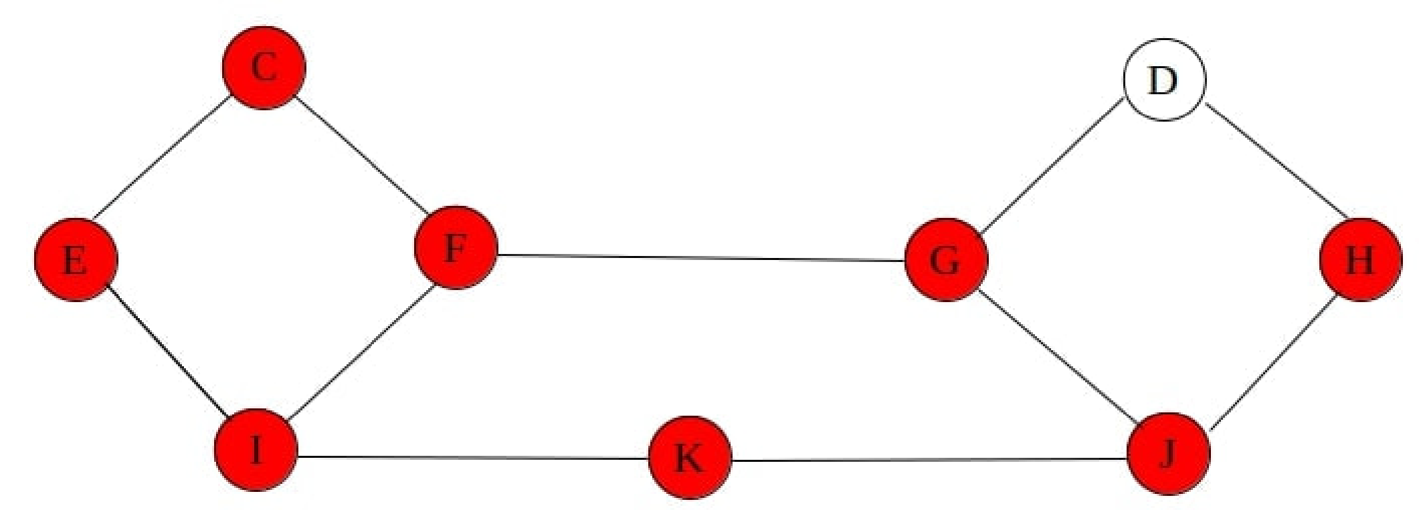
b) Lo que pasa en este caso es que a la cascada se le dificulta "salir" de la comunidad de la izquierda, que esta densamente conectada. Dicha comunidad esta conectada con la de la derecha mediante los nodos G y K (K no pertenece a ninguna de las dos comunidades) mediante lazos debiles, los cuales son muy buenos para transmitir informacion sobre un comportamiento pero son debiles para trasmitir el comportamiento en si. Por lo que la comunidad de la derecha nunca obtendra el comportamiento de la otra comunidad salvo que haya nodos internamente en ella que comienzen con él.

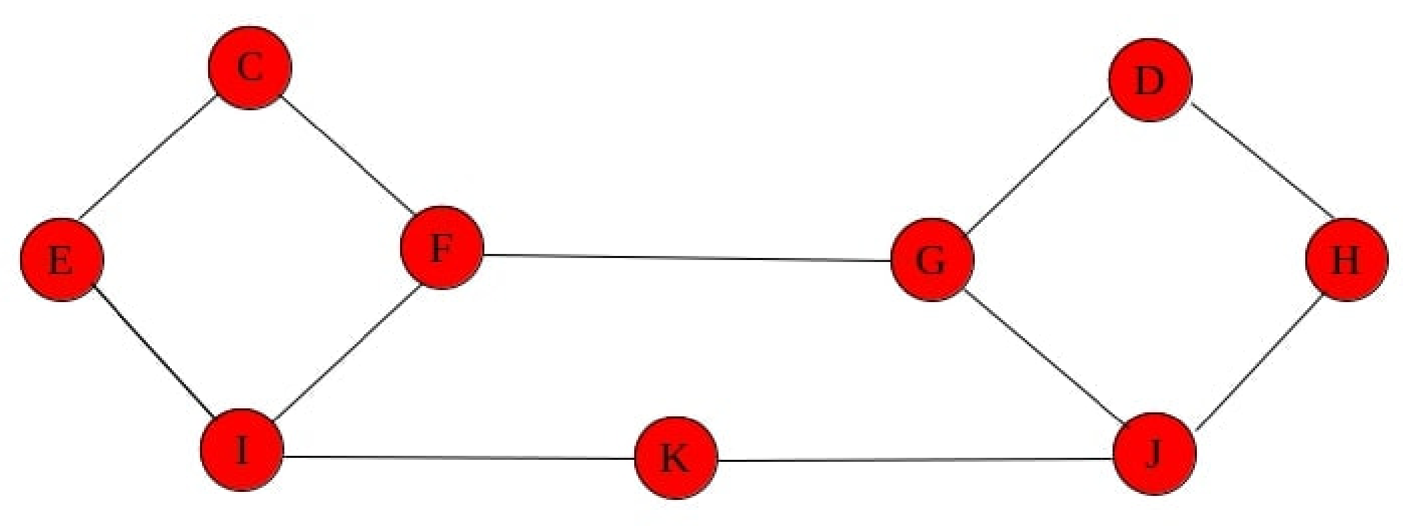
Para que el comportamiento se propague por toda la red, deben haber early adopters en la otra comunidad. Por ejemplo, si el nodo J es un early adapter el comportamiento se propagara.



Esto generara que los demas nodos que quedaron sin "contagiarse" consigan un treashold mayor o igual a 0.5 y puedan adoptar el comportamiento A. Finalmente la red quedaria asi







Ejercicio 4

Tenemos dos grafos no dirigidos G1 y G2, con la misma cantidad de vértices y aristas. G1 es un grafo aleatorio de Erdös-Rényi, mientras G2 es un grafo que cumple la ley de potencias en la distribución de los grados. Consideremos un virus que comienza en un único vértice aleatorio y se expande según el modelo SIR.

a. ¿En cuál grafo es más probable que ocurra una epidemia (i.e. se infecte al menos un 30% de la red)? Justificar brevemente la respuesta.

b. Supongamos que en vez de comenzar en un vértice aleatorio, la epidemia comenzara en el vértice de mayor grado de G1 y G2, respectivamente. ¿En cuál de los grafos es más probable que ocurra una epidemia? Justificar brevemente la respuesta.

c. ¿Cómo afecta la existencia (o no existencia) de comunidades en la expansión de la epidemia?

a) El grafo que tiene mas probabilidad de que se infecte el 30% de la red es la red aleatoria G1. Ya que las redes que cumplen con la red de potencias son mucho mas robustas y se "bancan" mas un ataque aleatorio.

c) Si en una red existen comunidades se podra decir que el nivel de contagio aumentara, ya que todos los elementos que la componen estan muy conectados entre si. Pero por otro lado, si estas comunidades no estan muy conectadas entre si y se aisla a las comunidades infectadas estas no lograran contagiar a las demas. Por lo que esto seria un aspecto positivo de la existencia de comunidades.

Ejercicio 5

Aplicar el Algoritmo REV2 al siguiente set de datos de reviews de productos de Amazon, para detectar potenciales usuarios maliciosos y otros ciertamente honestos. Por simplificación (y unificación de criterios), considerar γ1 = γ2 = 0.5. Obtener aquellos usuarios cuya justicia (fairness) es menor o igual a 0.2 (son maliciosos) y tienen al menos 5 reviews, así como la proporción de nodos que son extremadamente justos: aquellos con justicia mayor o igual a 0.9, y con al menos 10 reviews (aristas de salida).

In [3]:

```
# Obtengo el set de datos

amazon = pd.read_csv("ratings_electronics.csv")
amazon = amazon.rename(columns={'AKM1MP6P0OYPR':'user_id', '0132793040': 'product_id',\
                                '5.0': 'rating', '1365811200':'timestamp'})
amazon = amazon.drop(['timestamp'], axis=1)
```

In [4]:

```
vector_amazon = amazon.to_numpy()
```

In [5]:

```
# cambio puntaje por rango del score, debe pertenecer a [-1, 1]
for e in vector_amazon:
    e[2] = ((e[2] - 3) / 2)
```

In [6]:

```
grado_salida_producto = {}
grado_salida_usuario = {}

for i in range(len(vector_amazon)):
    if grado_salida_usuario.get(vector_amazon[i][0]):
        grado_salida_usuario[vector_amazon[i][0]] += 1
    else:
        grado_salida_usuario[vector_amazon[i][0]] = 1

    if grado_salida_producto.get(vector_amazon[i][1]):
        grado_salida_producto[vector_amazon[i][1]] += 1
    else:
        grado_salida_producto[vector_amazon[i][1]] = 1

def obtener_grado_salida(key, char):
    if char == 'u':
        return grado_salida_usuario.get(key)
    else:
        return grado_salida_producto.get(key)
```

In [7]:

```
# producto-> { key: nombre_producto, value: {int grado_salida, int gain, [] usuarios_calificadores} }
# usuario-> { key: nombre_usuario, value: {int grado_salida, int fairness, [] productos_calificados} }
# usuario_producto-> { key: usuario+producto, value: {int RUP, int score, string usuario, string producto} }

producto = {}
usuario = {}
usuario_producto = {}

for i in range(len(vector_amazon)):
    #----------------------PRODUCTO-------------------------#
    p = {'grado_salida': 0, 'gain': 1, 'usuarios_calificadores': []}

    p['grado_salida'] = obtener_grado_salida(vector_amazon[i][1], 'p')
    if producto.get(vector_amazon[i][1]):
            producto.get(vector_amazon[i][1])['usuarios_calificadores'].append(vector_amazon[i][0])
    else:
        producto[vector_amazon[i][1]] = p
        producto.get(vector_amazon[i][1])['usuarios_calificadores'].append(vector_amazon[i][0])
    #----------------------USUARIO-------------------------#
    u = {'grado_salida': 0, 'fairness': 1, 'productos_calificados': []}

    u['grado_salida'] = obtener_grado_salida(vector_amazon[i][0], 'u')
    if usuario.get(vector_amazon[i][0]):
            usuario.get(vector_amazon[i][0])['productos_calificados'].append(vector_amazon[i][1])
    else:
        usuario[vector_amazon[i][0]] = u
        usuario.get(vector_amazon[i][0])['productos_calificados'].append(vector_amazon[i][1])
    #----------------------USUARIO_PRODUCTO-------------------------#
    up = {'RUP': 1, 'score': 0, 'usuario': '', 'producto': ''}

    up['usuario']=vector_amazon[i][0]
    up['producto']=vector_amazon[i][1]
    up['score'] = vector_amazon[i][2]
    usuario_producto[vector_amazon[i][0]+vector_amazon[i][1]] = up
```

In [8]:

```
def obtener_usuarios_maliciosos():
    iteraciones = 0
    while (iteraciones < 11):
        # -----------------------------------------------------------------------------------------
        # El fu lo actualizo con (la sumatoria de Rup (variando p)) / (el grado de salida del usuario)
        # -----------------------------------------------------------------------------------------
        for u, value_u in usuario.items():
            sumatoria_rup = 0
            for p in range(len(value_u.get('productos_calificados'))):
                sumatoria_rup += usuario_producto.get(u+value_u.get('productos_calificados')[p])['RUP']
            value_u['fairness'] = sumatoria_rup / obtener_grado_salida(u, 'u')
        # ---------------------------------------------------------------------
        # El gp lo actualizo con (la sumatoria de Rup (variando u) * score) / (cantidad de usuarios que vieron ese producto)
        # ---------------------------------------------------------------------
        for p, value_p in producto.items():
            sumatoria_rup_por_score = 0
            for u in range(len(value_p.get('usuarios_calificadores'))):
                sumatoria_rup_por_score += (usuario_producto.get(value_p.get('usuarios_calificadores')[u]+p)['RUP'] * usuario_producto.get(value_p.get('usuari
            value_p['gain'] = sumatoria_rup_por_score / obtener_grado_salida(p, 'p')
        # ---------------------------------------------------------------------
        # El rup se actualiza como (0.5 * Fu + 0.5(1 - (|score - Gp|)/2))
        # ---------------------------------------------------------------------
        for up, value_up in usuario_producto.items():
            value_up['RUP'] =   ( 0.5 * usuario.get(value_up['usuario'])['fairness'] \
                              + 0.5 * (1 - (np.abs(value_up['score'] - producto.get(value_up['producto'])['gain'])/2)) )

        print("Se realizo la iteracion numero", iteraciones)
        iteraciones += 1

obtener_usuarios_maliciosos()
```

```
Se realizo la iteracion numero 0
Se realizo la iteracion numero 1
Se realizo la iteracion numero 2
Se realizo la iteracion numero 3
Se realizo la iteracion numero 4
Se realizo la iteracion numero 5
Se realizo la iteracion numero 6
Se realizo la iteracion numero 7
Se realizo la iteracion numero 8
Se realizo la iteracion numero 9
Se realizo la iteracion numero 10
```

In [10]:

```
usuarios_maliciosos = []
usuarios_buenitos = []

for u, value_u in usuario.items():
    if (value_u['fairness'] <= 0.2) and (value_u['grado_salida'] >= 5):
        usuarios_maliciosos.append(u)

for u, value_u in usuario.items():
    if value_u['fairness'] > 0.9 and (value_u['grado_salida'] >= 10):
        usuarios_buenitos.append(u)

print("Los usuarios maliciosos son:", usuarios_maliciosos)
print("El porcentaje de usuarios extremadamente justos es:", len(usuarios_buenitos) / len(usuario) * 100)
```

```
Los usuarios maliciosos son: ['A2G4GBZBU0191J']
El porcentaje de usuarios extremadamente justos es: 0.009876963968835441
```