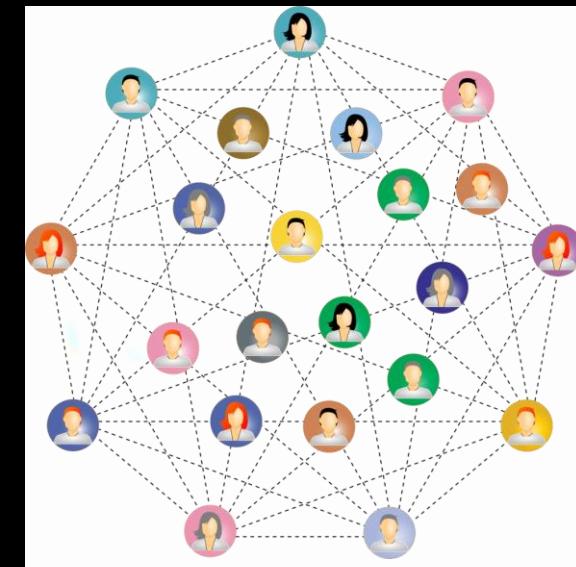


# ALGORITMOS BFS Y DFS APLICADOS A REDES SOCIALES

*ESTRUCTURA DE DATOS Y  
ALGORITMOS II*

*8 DE JUNIO DE 2025*

- RICHARD TIPANTIZA
- JAIRO ANGULO
- TAMARA BENAVIDEZ



# OBJETIVOS

- Implementar **BFS** y **DFS** para analizar conexiones entre perfiles en redes sociales mediante grafos.
- Diseñar un modelo orientado a objetos (POO) para representar grafos, mediante clases como Grafo, Nodo y Artista.

# MOTIVACIÓN

Las redes sociales requieren algoritmos eficientes para:

- Recomendar conexiones relevantes (BFS para rutas cortas).
- Explorar comunidades o grupos afines (DFS para análisis profundo).

# DEFINICIÓN DEL PROBLEMA

## CASO DE ESTUDIO

Grafo de una red social.

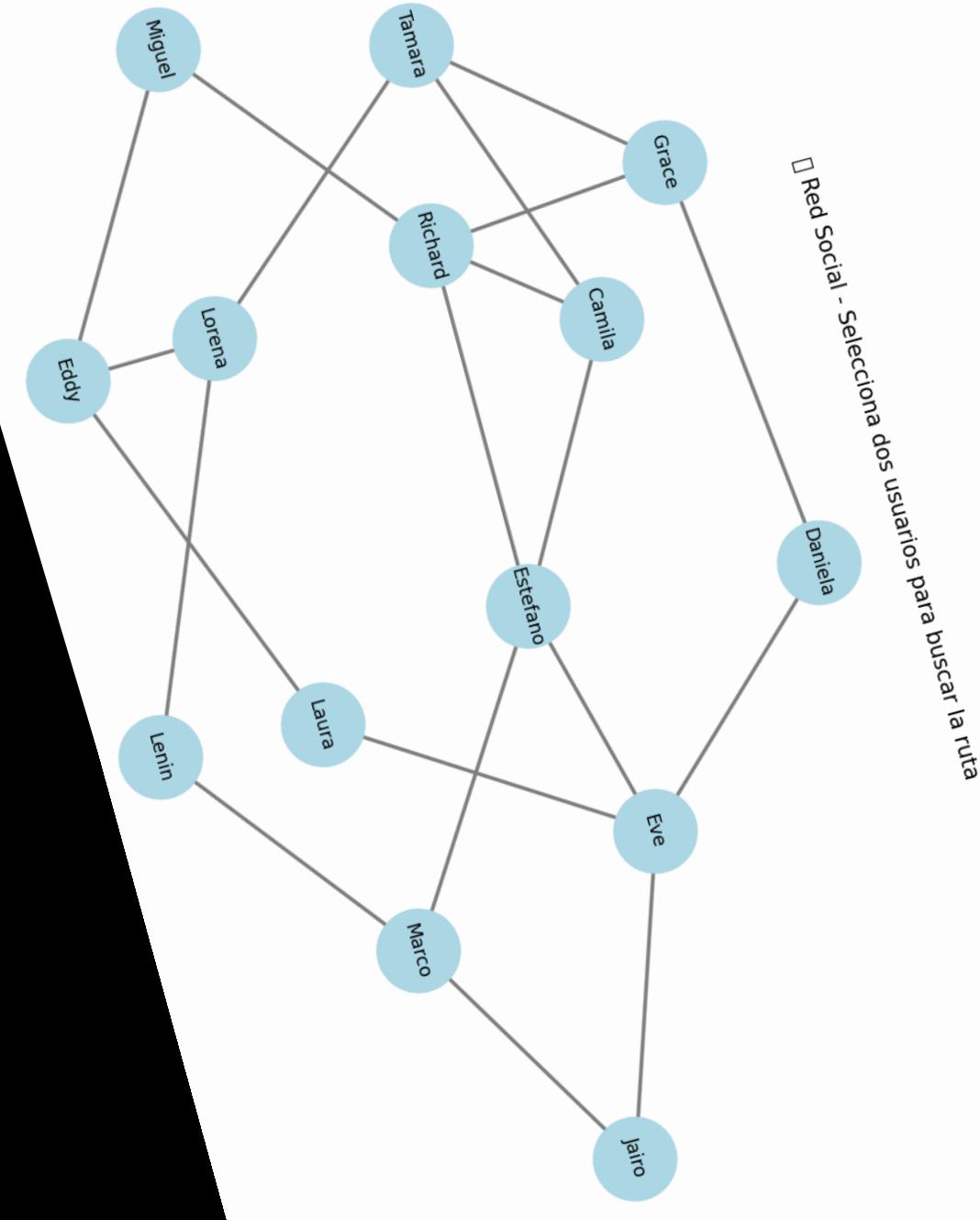
### GRAFO

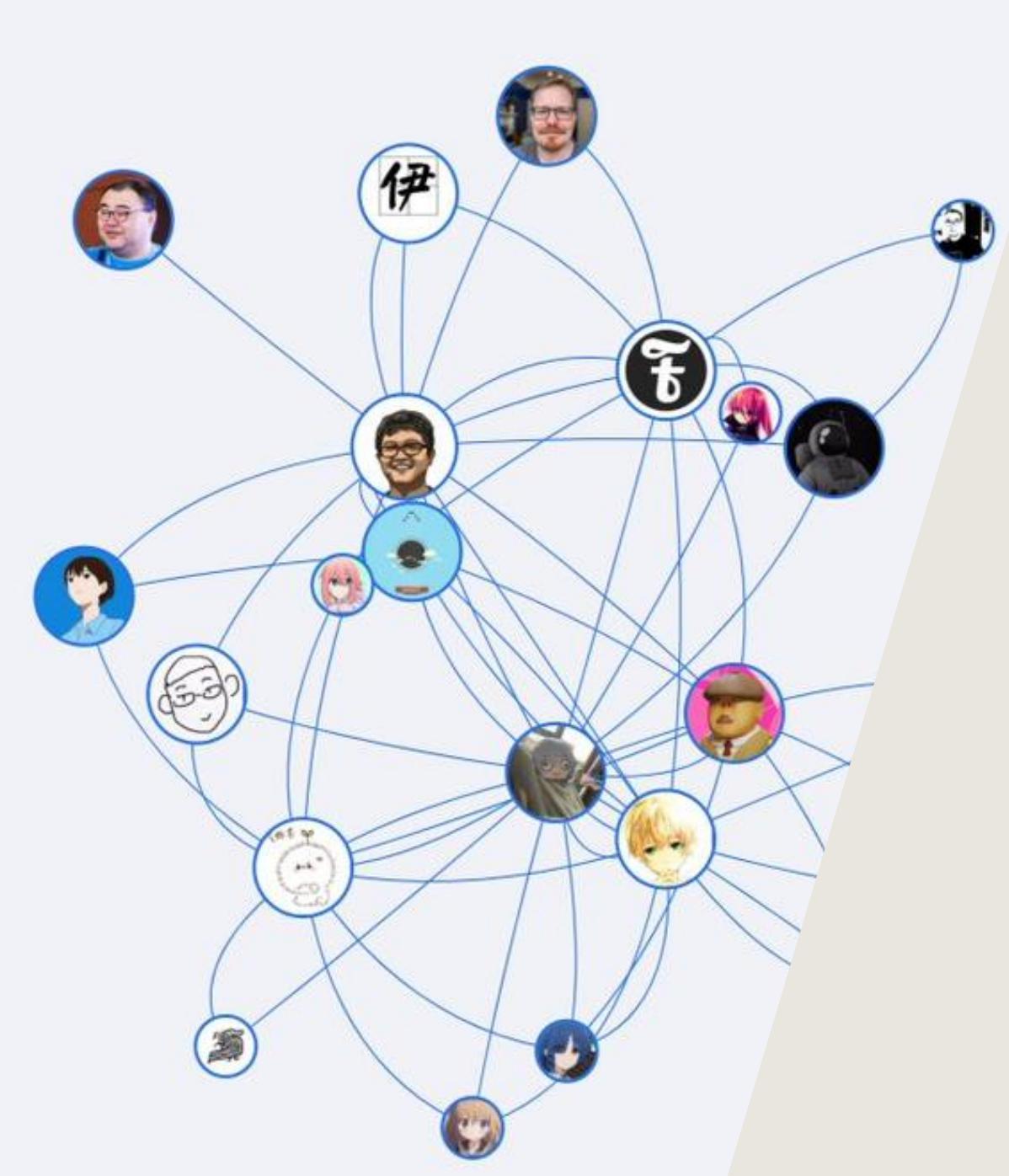
Perfiles (Nodos): - Grace - Daniela - Eve - Jairo - Marco - Lenin - Lorena - Eddy - Laura - Miguel - Estefano - Camila - Richard - Tamara.

Aristas: Conexiones entre personas.

### ALGORITMOS

- BFS → Encuentra la ruta más corta entre usuarios.
- DFS → Explora conexiones profundas.





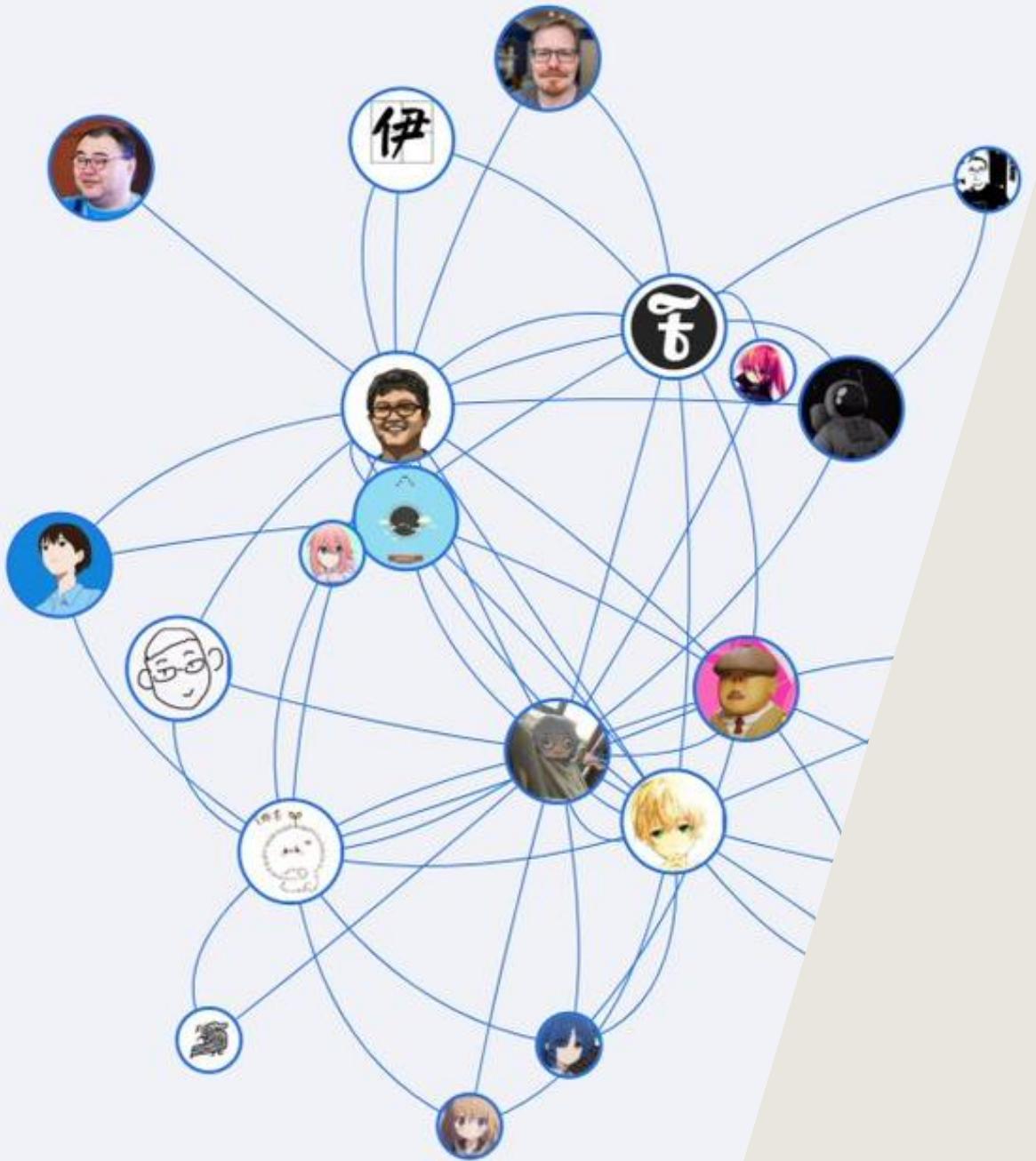
# DISEÑO DE CLASES

Clase Nodo:

```
class Nodo:  
    Tabnine | Edit | Test | Explain | Document  
    def __init__(self, nombre, tipo=None):  
        self.nombre = nombre  
        self.tipo = tipo # entrada, salida, edificio, etc.  
  
    Tabnine | Edit | Test | Explain | Document  
    def __repr__(self):  
        return self.nombre
```

Clase Arista:

```
class Arista:  
    Tabnine | Edit | Test | Explain | Document  
    def __init__(self, nodo1, nodo2, peso=1):  
        self.nodo1 = nodo1  
        self.nodo2 = nodo2  
        self.peso = peso
```



# DISEÑO DE CLASES

Clase Grafo:

```
class Grafo:  
    Tabnine | Edit | Test | Explain | Document  
    def __init__(self):  
        self.nodos = {}  
        self.aristas = []  
        self.adj_list = {}  
  
    Tabnine | Edit | Test | Explain | Document  
    def agregar_nodo(self, nodo):  
        if nodo.nombre not in self.nodos:  
            self.nodos[nodo.nombre] = nodo  
            self.adj_list[nodo] = []  
  
    Tabnine | Edit | Test | Explain | Document  
    def agregar_arista(self, nodo1, nodo2, peso=1):  
        if nodo1.nombre in self.nodos and nodo2.nombre in self.nodos:  
            arista = Arista(nodo1, nodo2, peso)  
            self.aristas.append(arista)  
            self.adj_list[nodo1].append(nodo2)  
            self.adj_list[nodo2].append(nodo1)  
        else:  
            raise ValueError("Uno o ambos nodos no existen")
```



# DISEÑO DE CLASES

BFS:

```
def bfs(self, inicio, fin):
    visitados = {nodo: False for nodo in self.nodos.values()}
    cola = deque([(inicio, [inicio])])
    visitados[inicio] = True

    while cola:
        actual, camino = cola.popleft()
        if actual == fin:
            return camino
        for vecino in self.adj_list[actual]:
            if not visitados[vecino]:
                visitados[vecino] = True
                cola.append((vecino, camino + [vecino]))
    return None
```



```

def dfs(self, inicio, destino):
    print("\nIniciando DFS (recursivo)...")
    searched = {nodo: False for nodo in self.nodos.values()}
    parents = {}
    componentR = []

    def _dfs(nodo, parent=None):
        componentR.append(nodo)
        searched[nodo] = True
        parents[nodo] = parent

        print(f"Estado de búsqueda en nodo {nodo}:")
        print({n.nombre: v for n, v in searched.items()})
        print(f"Vecinos de {nodo}: {self.adj_list[nodo]}")
        print()

        if nodo == destino:
            return True

        for vecino in self.adj_list[nodo]:
            if not searched[vecino]:
                if _dfs(vecino, nodo):
                    return True
                print(f"Finaliza {vecino}")
                print(f"Vuelve a {nodo}")
                print()

    return False

```

# DISEÑO DE CLASES

DFS:

```

encontrado = _dfs(inicio)

if encontrado:
    # Reconstrucción del camino usando el diccionario de padres
    camino = []
    actual = destino
    while actual is not None:
        camino.append(actual)
        actual = parents.get(actual)
    camino.reverse()
    print("✅ Ruta encontrada con DFS:")
    print(" → ".join(n.nombre for n in camino))
    return camino
else:
    print("❌ No se encontró ruta con DFS")
    return None

```

# IMPLEMENTACIÓN

## VISUALIZACIÓN CON NETWORKX

```
def dibujar_ruta(grafo, ruta, titulo):
    G = nx.Graph()
    for nodo in grafo.nodos.values():
        G.add_node(nodo.nombre)
    for arista in grafo.aristas:
        G.add_edge(arista.nodo1.nombre, arista.nodo2.nombre)

    pos = nx.kamada_kawai_layout(G)

    fig, ax = plt.subplots(figsize=(14, 10))

    nombres_ruta = [n.nombre for n in ruta]
    colores_nodos = ['red' if nodo in nombres_ruta else 'lightblue' for nodo in G.nodes()]
    aristas_ruta = list(zip(nombres_ruta[:-1], nombres_ruta[1:]))
    colores_aristas = ['red' if (u, v) in aristas_ruta or (v, u) in aristas_ruta else 'gray' for u, v in G.edges()]

    nx.draw(G, pos, with_labels=True, node_color=colores_nodos, edge_color=colores_aristas,
            node_size=2000, font_size=10, width=2, ax=ax)

    # Título informativo con enfoque de red social y tipo de algoritmo
    if "BFS" in titulo:
        plt.title("◆ BFS - Ruta más corta entre dos usuarios en la red social")
    elif "DFS" in titulo:
        plt.title("◆ DFS - Ruta profunda entre dos usuarios en la red social")
    else:
        plt.title(f"◆ Ruta encontrada - {titulo}")

    plt.show()
```

## INTERFAZ INTERACTIVA

```
def ejecutar_simulacion():
    grafo = construir_grafo_demo()

    print("● Haz clic en el nodo de inicio y luego en el de destino...")
    seleccionados = dibujar_grafo_interactivo(grafo)

    if len(seleccionados) != 2:
        print("✗ Debes seleccionar exactamente 2 nodos")
        return

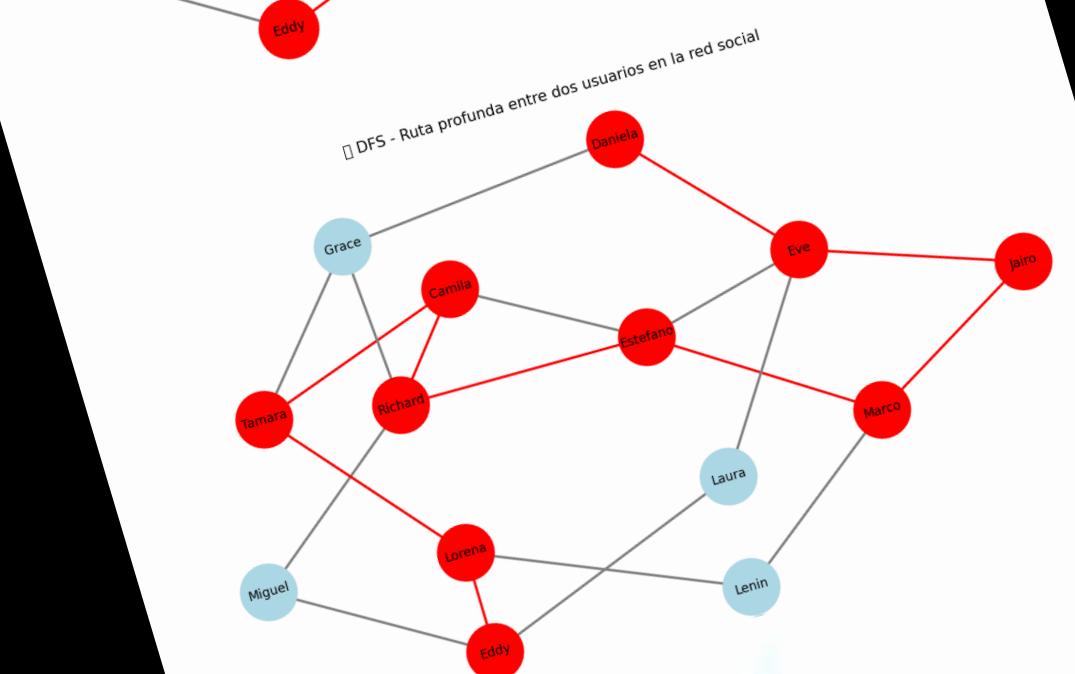
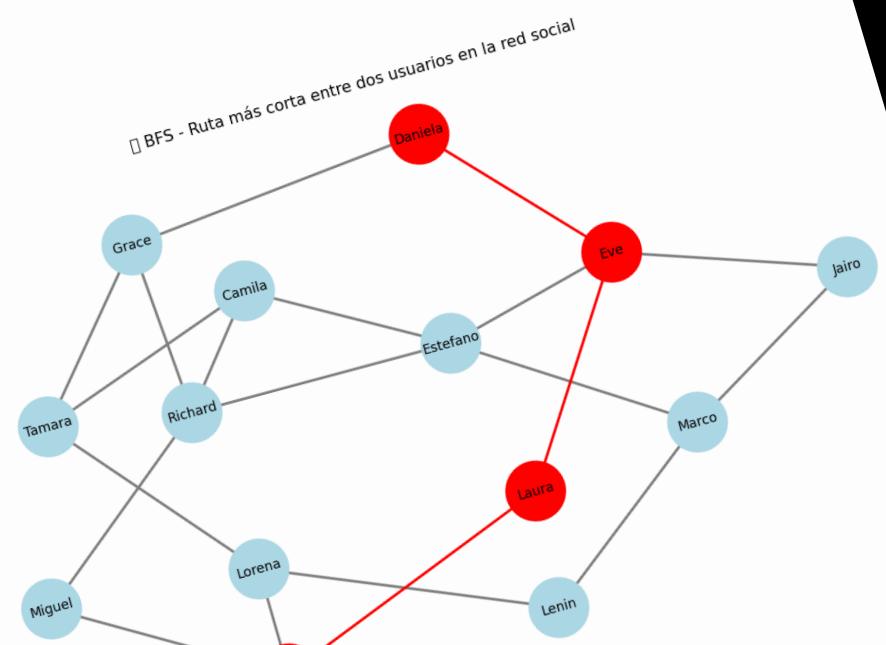
    inicio, fin = seleccionados
    nodo_inicio = grafo.nodos[inicio]
    nodo_fin = grafo.nodos[fin]

    print(f"\nRuta desde {inicio} hasta {fin}:")

    print("\n◆ Ejecutando BFS (Ruta más corta)...")
    ruta_bfs = grafo.bfs(nodo_inicio, nodo_fin)
    if ruta_bfs:
        print(" → ".join(n.nombre for n in ruta_bfs))
        dibujar_ruta(grafo, ruta_bfs, "BFS - Ruta más corta")
    else:
        print("No se encontró ruta con BFS")

    print("\n◆ Ejecutando DFS (Ruta más profunda)...")
    ruta_dfs = grafo.dfs(nodo_inicio, nodo_fin)
    if ruta_dfs:
        print(" → ".join(n.nombre for n in ruta_dfs))
        dibujar_ruta(grafo, ruta_dfs, "DFS - Ruta más profunda")
    else:
        print("No se encontró ruta con DFS")
```

# RESULTADOS BFS VS DFS



Ejemplo: De Daniela a Eddy

BFS Ruta más corta: Daniela → Eve → Laura → Eddy

DFS Ruta más profunda: Daniela → Eve → Jairo → Marco → Estefano → Richard → Camila → Tamara → Lorena → Eddy

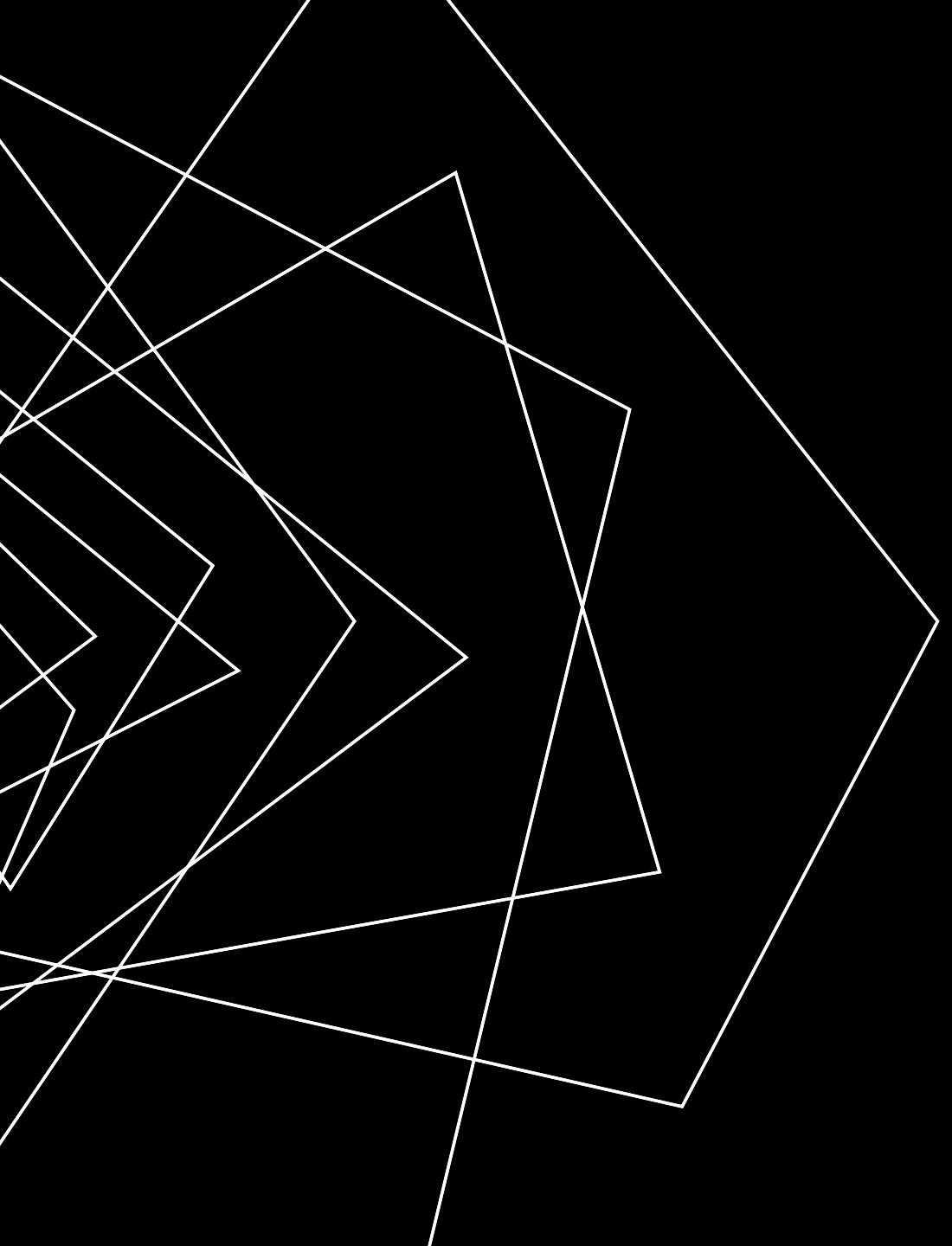
# CONCLUSIONES

## APRENDIZAJES:

- POO permite modelar redes sociales de forma modular.
- BFS es ideal para conexiones rápidas; DFS para análisis estructural.

## DESAFÍOS:

- Optimizar la recursividad en DFS para grafos grandes.



GRACIAS POR SU  
ATENCIÓN