Ejercicio de Grafos en Python para Grupos de Estudiantes

Objetivo del Ejercicio

Crear un sistema de recomendación de amistades para una red social universitaria utilizando grafos.

Descripción del Problema

Implementar un grafo que represente las relaciones de amistad entre estudiantes y desarrollar algoritmos para recomendar nuevas amistades basándose en amigos en común.

División de Tareas por Grupo (Estudiantes)

Estructura del Grafo

- Implementar la clase Grafo con listas de adyacencia
- Métodos: agregar_vertice(), agregar_arista(), obtener_vecinos()

Carga de Datos y Visualización

- Leer datos de estudiantes y amistades desde archivos CSV
- Implementar visualización básica del grafo

Algoritmo de Recomendaciones

- Implementar algoritmo de recomendación por amigos en común
- Calcular coeficientes de similitud

Interfaz y Análisis

- Crear interfaz de usuario simple
- Generar estadísticas y métricas del grafo

Código Base para Comenzar

python

```
import csv
import matplotlib.pyplot as plt
import networkx as nx
```

```
from collections import defaultdict, deque


# ============================================
# ESTRUCTURA DEL GRAFO
# ============================================


class Grafo:
    def __init__(self):
        self.adj_list = defaultdict(dict)
        self.estudiantes = {}


    def agregar_estudiante(self, id_estudiante, nombre, carrera):
        """Agrega un estudiante al grafo"""
        self.estudiantes[id_estudiante] = {
            'nombre': nombre,
            'carrera': carrera
        }
        if id_estudiante not in self.adj_list:
            self.adj_list[id_estudiante] = {}


    def agregar_amistad(self, id1, id2):
        """Agrega una relación de amistad entre dos estudiantes"""
        if id1 in self.estudiantes and id2 in self.estudiantes:
            self.adj_list[id1][id2] = 1  # Peso 1 para amistad
            self.adj_list[id2][id1] = 1
        else:
```

```
        print(f"Error: Uno o ambos estudiantes no existen")

    def obtener_amigos(self, id_estudiante):
        """Retorna la lista de amigos de un estudiante"""
        return list(self.adj_list[id_estudiante].keys())

    def son_amigos(self, id1, id2):
        """Verifica si dos estudiantes son amigos"""
        return id2 in self.adj_list[id1]

    def __str__(self):
        result = "Grafo de Amistades:\n"
        for estudiante in self.adj_list:
            amigos = self.obtener_amigos(estudiante)
            result += f"{self.estudiantes[estudiante]['nombre']}:
{[self.estudiantes[amigo]['nombre'] for amigo in amigos]}\n"
        return result

# ============================================
# CARGA DE DATOS Y VISUALIZACIÓN
# ============================================

def cargar_datos(grafo, archivo_estudiantes, archivo_amistades):
    """Carga estudiantes y amistades desde archivos CSV"""

    # Cargar estudiantes
```

```
try:
    with open(archivo_estudiantes, 'r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            grafo.agregar_estudiante(
                row['id'],
                row['nombre'],
                row['carrera']
            )
    print(f"Estudiantes cargados: {len(grafo.estudiantes)}")
except FileNotFoundError:
    print("Creando datos de ejemplo...")
    crear_datos_ejemplo(grafo)

# Cargar amistades
try:
    with open(archivo_amistades, 'r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            grafo.agregar_amistad(row['id1'], row['id2'])
    print("Amistades cargadas exitosamente")
except FileNotFoundError:
    print("Usando amistades de ejemplo")

def crear_datos_ejemplo(grafo):
    """Crea datos de ejemplo si no hay archivos"""
```

```
    estudiantes = [
        ('1', 'Ana García', 'Ingeniería'),
        ('2', 'Luis Martínez', 'Medicina'),
        ('3', 'María López', 'Derecho'),
        ('4', 'Carlos Rodríguez', 'Ingeniería'),
        ('5', 'Elena Torres', 'Psicología'),
        ('6', 'Pedro Sánchez', 'Medicina'),
        ('7', 'Sofía Ramírez', 'Derecho'),
        ('8', 'Miguel Fernández', 'Ingeniería')
    ]

    for id, nombre, carrera in estudiantes:
        grafo.agregar_estudiante(id, nombre, carrera)

    # Crear algunas amistades
    amistades = [('1','2'), ('1','3'), ('2','4'), ('3','5'),
                 ('4','6'), ('5','7'), ('6','8'), ('7','8')]

    for id1, id2 in amistades:
        grafo.agregar_amistad(id1, id2)

def visualizar_grafo(grafo):
    """Visualiza el grafo usando networkx y matplotlib"""
    G = nx.Graph()

    # Agregar nodos
```

```
for id_est, info in grafo.estudiantes.items():
    G.add_node(info['nombre'], carrera=info['carrera'])

# Agregar aristas
for id1 in grafo.adj_list:
    for id2 in grafo.adj_list[id1]:
        nombre1 = grafo.estudiantes[id1]['nombre']
        nombre2 = grafo.estudiantes[id2]['nombre']
        G.add_edge(nombre1, nombre2)

# Configurar visualización
plt.figure(figsize=(12, 8))

# Colores por carrera
carreras = list(set(info['carrera'] for info in grafo.estudiantes.values()))
colores = ['red', 'blue', 'green', 'orange', 'purple']
color_map = {}

for i, carrera in enumerate(carreras):
    color_map[carrera] = colores[i % len(colores)]

node_colors = [color_map[grafo.estudiantes[id]['carrera']]
               for id in grafo.estudiantes]

# Dibujar grafo
pos = nx.spring_layout(G, k=1, iterations=50)
```

```
    nx.draw(G, pos, with_labels=True, node_color=node_colors,
            node_size=800, font_size=8, font_weight='bold')

    # Leyenda
    for carrera, color in color_map.items():
        plt.plot([], [], 'o', color=color, label=carrera)
    plt.legend()

    plt.title("Red de Amistades Universitarias")
    plt.show()

# ============================================
# ALGORITMO DE RECOMENDACIONES
# ============================================

def recomendar_amistades(grafo, id_estudiante, max_recomendaciones=5):
    """
    Recomienda amistades basándose en amigos en común
    y misma carrera
    """
    if id_estudiante not in grafo.estudiantes:
        return []

    recomendaciones = {}
    amigos_actuales = set(grafo.obtener_amigos(id_estudiante))
    carrera_estudiante = grafo.estudiantes[id_estudiante]['carrera']
```

```
  for posible_amigo in grafo.estudiantes:
    # No recomendar a sí mismo ni a amigos actuales
    if posible_amigo == id_estudiante or posible_amigo in amigos_actuales:
      continue

    # Calcular amigos en común
    amigos_posible = set(grafo.obtener_amigos(posible_amigo))
    amigos_comunes = amigos_actuales.intersection(amigos_posible)

    # Calcular puntaje
    puntaje = len(amigos_comunes) * 2  # Peso para amigos en común

    # Bonus por misma carrera
    if grafo.estudiantes[posible_amigo]['carrera'] == carrera_estudiante:
      puntaje += 1

    if puntaje > 0:
      recomendaciones[posible_amigo] = {
        'puntaje': puntaje,
        'amigos_comunes': len(amigos_comunes),
        'misma_carrera': grafo.estudiantes[posible_amigo]['carrera'] ==
carrera_estudiante
      }

  # Ordenar por puntaje y retornar las mejores recomendaciones
```

```
    recomendaciones_ordenadas = sorted(
        recomendaciones.items(),
        key=lambda x: x[1]['puntaje'],
        reverse=True
    )[:max_recomendaciones]

    return recomendaciones_ordenadas

def camino_mas_corto(grafo, id_inicio, id_fin):
    """Encuentra el camino más corto entre dos estudiantes usando BFS"""
    if id_inicio not in grafo.estudiantes or id_fin not in grafo.estudiantes:
        return None

    if id_inicio == id_fin:
        return [id_inicio]

    visitado = set()
    cola = deque([(id_inicio, [id_inicio])])

    while cola:
        actual, camino = cola.popleft()

        if actual == id_fin:
            return [grafo.estudiantes[id]['nombre'] for id in camino]

        visitado.add(actual)
```

```
    for vecino in grafo.obtener_amigos(actual):
        if vecino not in visitado:
            cola.append((vecino, camino + [vecino]))
            visitado.add(vecino)

  return None  # No hay camino

# ============================================
# INTERFAZ Y ANÁLISIS
# ============================================

def mostrar_estadisticas(grafo):
  """Muestra estadísticas básicas del grafo"""
  print("\n" + "="*50)
  print("ESTADÍSTICAS DE LA RED")
  print("="*50)

  num_estudiantes = len(grafo.estudiantes)
  num_amistades = sum(len(amigos) for amigos in grafo.adj_list.values()) // 2

  print(f"Total de estudiantes: {num_estudiantes}")
  print(f"Total de amistades: {num_amistades}")
  print(f"Promedio de amigos por estudiante:
{num_amistades*2/num_estudiantes:.2f}")
```

```
    # Estudiantes por carrera
    carreras = {}
    for info in grafo.estudiantes.values():
        carrera = info['carrera']
        carreras[carrera] = carreras.get(carrera, 0) + 1

    print("\nEstudiantes por carrera:")
    for carrera, cantidad in carreras.items():
        print(f"  {carrera}: {cantidad} estudiantes")

    # Estudiantes más populares (con más amigos)
    popularidad = []
    for id_est in grafo.estudiantes:
        num_amigos = len(grafo.obtener_amigos(id_est))
        popularidad.append((grafo.estudiantes[id_est]['nombre'], num_amigos))

    popularidad.sort(key=lambda x: x[1], reverse=True)
    print(f"\nEstudiantes más populares:")
    for nombre, amigos in popularidad[:3]:
        print(f"  {nombre}: {amigos} amigos")

def interfaz_principal(grafo):
    """Interfaz de usuario simple para interactuar con el sistema"""
    while True:
        print("\n" + "="*50)
        print("SISTEMA DE RECOMENDACIÓN DE AMISTADES")
```

```
print("="*50)
print("1. Ver todos los estudiantes")
print("2. Ver amistades de un estudiante")
print("3. Recomendar amistades")
print("4. Encontrar camino entre estudiantes")
print("5. Ver estadísticas")
print("6. Visualizar grafo")
print("7. Salir")

opcion = input("\nSelecciona una opción (1-7): ").strip()

if opcion == '1':
    print("\nLISTA DE ESTUDIANTES:")
    for id_est, info in grafo.estudiantes.items():
        print(f"ID: {id_est} - {info['nombre']} ({info['carrera']})")

elif opcion == '2':
    id_est = input("ID del estudiante: ").strip()
    if id_est in grafo.estudiantes:
        amigos = grafo.obtener_amigos(id_est)
        print(f"\nAmigos de {grafo.estudiantes[id_est]['nombre']}:")
        for amigo_id in amigos:
            info_amigo = grafo.estudiantes[amigo_id]
            print(f"  - {info_amigo['nombre']} ({info_amigo['carrera']})")
    else:
        print("Estudiante no encontrado")
```

```
    elif opcion == '3':
        id_est = input("ID del estudiante para recomendaciones: ").strip()
        if id_est in grafo.estudiantes:
            recomendaciones = recomendar_amistades(grafo, id_est)
            if recomendaciones:
                print(f"\nRecomendaciones para {grafo.estudiantes[id_est]['nombre']}:")
                for i, (id_rec, info) in enumerate(recomendaciones, 1):
                    estudiante = grafo.estudiantes[id_rec]
                    print(f"{i}. {estudiante['nombre']} ({estudiante['carrera']})")
                    print(f"   Puntaje: {info['puntaje']} | Amigos en común:
{info['amigos_comunes']}")
            else:
                print("No hay recomendaciones disponibles")
        else:
            print("Estudiante no encontrado")

    elif opcion == '4':
        id1 = input("ID del primer estudiante: ").strip()
        id2 = input("ID del segundo estudiante: ").strip()
        camino = camino_mas_corto(grafo, id1, id2)
        if camino:
            print(f"\nCamino más corto:")
            print(" -> ".join(camino))
        else:
            print("No existe camino entre estos estudiantes")
```

```
    elif opcion == '5':
        mostrar_estadisticas(grafo)

    elif opcion == '6':
        print("Generando visualización...")
        visualizar_grafo(grafo)

    elif opcion == '7':
        print("¡Hasta luego!")
        break

    else:
        print("Opción no válida")

# ============================================
# EJECUCIÓN PRINCIPAL
# ============================================

def main():
    # Crear grafo
    red_universitaria = Grafo()

    # Cargar datos
    cargar_datos(red_universitaria, 'estudiantes.csv', 'amistades.csv')
```

```
    # Mostrar información inicial
    print(red_universitaria)

    # Ejecutar interfaz
    interfaz_principal(red_universitaria)

if __name__ == "__main__":
    main()
```

Archivos de Datos de Ejemplo

estudiantes:

```
id,nombre,carrera
1,Ana García,Ingeniería
2,Luis Martínez,Medicina
3,María López,Derecho
4,Carlos Rodríguez,Ingeniería
5,Elena Torres,Psicología
6,Pedro Sánchez,Medicina
7,Sofía Ramírez,Derecho
8,Miguel Fernández,Ingeniería
```

amistades:

```
id1,id2
1,2
1,3
2,4
3,5
```

4,6

5,7

6,8

7,8

1,4

2,6

Tareas Específicas por Estudiante

Mejoras:

- Implementar eliminación de estudiantes y amistades
- Agregar pesos a las amistades (mejores amigos)
- Implementar búsqueda en profundidad (DFS)
- Crear generador de datos aleatorios para pruebas
- Mejorar la visualización con diferentes layouts
- Exportar el grafo a formato GEXF para Gephi
- Implementar algoritmo de comunidades (Louvain)
- Agregar recomendaciones por intereses comunes
- Calcular centralidad de los nodos
- Crear interfaz web con Streamlit
- Generar reportes PDF con estadísticas
- Implementar persistencia de datos con JSON

Métricas de Evaluación

- Correcta implementación de los algoritmos
- Calidad del código (comentarios, estructura)
- Funcionalidad completa del sistema
- Creatividad en las mejoras adicionales