ALEXANDRE DIAMANTE

FUNDAMENTOS DE ESTRUTURAS DE DADOS 4 – ESTRUTURAS DE DADOS AVANÇADAS E ANÁLISE DE DADOS

2 - Análise de dados estruturados

Trabalho apresentado para Atividade Prática na disciplina ESTRUTURA DE DADOS, do curso de CIÊNCIA DE DADOS, turno EAD da Instituição de Ensino Ampli Anhanguera.

Umuarama-PR 2024

import networkx as nx

from networkx.algorithms import community

class User:

def \_\_init\_\_(self, user\_id, data):

self.user\_id = user\_id

self.data = data

# Métodos getter e setter

def get\_user\_id(self):

return self.user\_id

def get\_data(self):

return self.data

def set\_data(self, new\_data):

self.data = new\_data

class SocialNetwork:

def \_\_init\_\_(self):

# Grafo que armazenará os usuários e conexões

self.graph = nx.Graph()

def add\_user(self, user\_id, user\_data):

# Adiciona um nó (usuário) no grafo

if user\_id not in self.graph:

self.graph.add\_node(user\_id, data=user\_data)

else:

print(f"Usuário {user\_id} já existe na rede.")

def remove\_user(self, user\_id):

# Remove um nó (usuário) e suas conexões

if user\_id in self.graph:

self.graph.remove\_node(user\_id)

else:

print(f"Usuário {user\_id} não encontrado na rede.")

def connect\_users(self, user1\_id, user2\_id):

# Estabelece uma conexão (aresta) entre dois usuários

if user1\_id in self.graph and user2\_id in self.graph:

self.graph.add\_edge(user1\_id, user2\_id)

else:

print(f"Um ou ambos os usuários {user1\_id} e {user2\_id} não estão na rede.")

def disconnect\_users(self, user1\_id, user2\_id):

# Remove uma conexão (aresta) entre dois usuários

if self.graph.has\_edge(user1\_id, user2\_id):

self.graph.remove\_edge(user1\_id, user2\_id)

else:

print(f"Conexão entre {user1\_id} e {user2\_id} não existe.")

def find\_communities(self):

# Aplica o algoritmo de detecção de comunidades usando o método de Girvan-Newman

communities\_generator = community.girvan\_newman(self.graph)

return next(communities\_generator)

def user\_centralities(self, method='degree'):

# Calcula centralidade dos usuários usando diferentes métodos

if method == 'degree':

return nx.degree\_centrality(self.graph)

elif method == 'betweenness':

return nx.betweenness\_centrality(self.graph)

elif method == 'closeness':

return nx.closeness\_centrality(self.graph)

elif method == 'eigenvector':

return nx.eigenvector\_centrality(self.graph)

else:

raise ValueError(f"Método {method} de centralidade não suportado.")

def analyze\_subgraph(self, user\_ids):

# Cria um subgrafo a partir de um conjunto de usuários

subgraph = self.graph.subgraph(user\_ids)

return subgraph

# Parte 2: Script de Teste

# Instanciando a rede social

rede\_social = SocialNetwork()

# Adicionando usuários

rede\_social.add\_user("u1", {"nome": "Alice", "interesses": ["música", "tecnologia"]})

rede\_social.add\_user("u2", {"nome": "Bob", "interesses": ["esportes", "tecnologia"]})

rede\_social.add\_user("u3", {"nome": "Charlie", "interesses": ["arte", "música"]})

rede\_social.add\_user("u4", {"nome": "David", "interesses": ["tecnologia", "cinema"]})

# Conectando usuários

rede\_social.connect\_users("u1", "u2")

rede\_social.connect\_users("u2", "u3")

rede\_social.connect\_users("u3", "u4")

rede\_social.connect\_users("u1", "u4")

# Calculando a centralidade de grau

centralidade = rede\_social.user\_centralities(method='degree')

print("Centralidade 'degree' de grau:", centralidade)

centralidade = rede\_social.user\_centralities(method='betweenness')

print("Centralidade 'betweenness' de grau:", centralidade)

centralidade = rede\_social.user\_centralities(method='closeness')

print("Centralidade 'closeness' de grau:", centralidade)

centralidade = rede\_social.user\_centralities(method='eigenvector')

print("Centralidade 'eigenvector' de grau:", centralidade)

# Detectando comunidades

comunidades = rede\_social.find\_communities()

print("Comunidades detectadas:", list(comunidades))

# Analisando um subgrafo

subgrafo = rede\_social.analyze\_subgraph(["u1", "u2", "u3"])

print("Subgrafo nós:", subgrafo.nodes)

print("Subgrafo arestas:", subgrafo.edges)

# Removendo um usuário

rede\_social.remove\_user("u4")

# Desconectando usuários

rede\_social.disconnect\_users("u2", "u3")

Resultado:  
  
Centralidade 'degree' de grau: {'u1': 0.6666666666666666, 'u2': 0.6666666666666666, 'u3': 0.6666666666666666, 'u4': 0.6666666666666666}

Centralidade 'betweenness' de grau: {'u1': 0.16666666666666666, 'u2': 0.16666666666666666, 'u3': 0.16666666666666666, 'u4': 0.16666666666666666}

Centralidade 'closeness' de grau: {'u1': 0.75, 'u2': 0.75, 'u3': 0.75, 'u4': 0.75}

Centralidade 'eigenvector' de grau: {'u1': 0.5, 'u2': 0.5, 'u3': 0.5, 'u4': 0.5}

Comunidades detectadas: [{'u4', 'u1'}, {'u2', 'u3'}]

Subgrafo nós: ['u1', 'u2', 'u3']

Subgrafo arestas: [('u1', 'u2'), ('u2', 'u3')]

Publicado no Git: <https://github.com/alexandrediamante/cienciadedados.git>