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