Graphs Kit: Uma Biblioteca Python para Manipulação, Análise e Visualização de Grafos

Seu Nome[1], **Nome do Colaborador**[1]

[1]Graduação de Engenharia de Software
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

Abstract. *Este artigo apresenta o desenvolvimento de uma biblioteca em Python para a manipulação, análise e visualização de grafos. A biblioteca oferece suporte a diferentes representações de grafos e implementa algoritmos clássicos para a análise de conectividade, identificação de pontes, articulações e componentes fortemente conexos. O objetivo é fornecer uma ferramenta versátil e eficiente para pesquisadores e desenvolvedores, facilitando a integração com sistemas externos por meio de exportação de grafos em diversos formatos. Resultados experimentais e comparações de desempenho também são apresentados.*

Resumo. *Este artigo apresenta o desenvolvimento de uma biblioteca em Python para a manipulação, análise e visualização de grafos. A biblioteca oferece suporte a diferentes representações de grafos e implementa algoritmos clássicos para a análise de conectividade, identificação de pontes, articulações e componentes fortemente conexos. O objetivo é fornecer uma ferramenta versátil e eficiente para pesquisadores e desenvolvedores, facilitando a integração com sistemas externos por meio de exportação de grafos em diversos formatos. Resultados experimentais e comparações de desempenho também são apresentados.*

1. Introdução

A teoria dos grafos desempenha um papel central em diversas áreas da ciência e tecnologia, sendo utilizada para modelar e resolver problemas complexos em redes de transporte, sistemas computacionais, redes sociais e biologia. Este trabalho apresenta o desenvolvimento de uma biblioteca em Python, denominada *Graphs Kit*, que busca atender às necessidades de manipulação, análise e visualização de grafos.

A *Graphs Kit* suporta múltiplas representações de grafos, como listas de adjacência e matrizes, e inclui funcionalidades como criação, remoção e checagem de adjacências, análise de conectividade e exportação de grafos para softwares como Gephi. Para uma visualização do usuário, a biblioteca possui exportação em arquivos ".ppm" que são nativos de todos os computadores. Como parte do projeto, existe dentro do sistema, implementações do algoritmo de Tarjan e de Naive, para a identificação de pontes e articulações.

Fundamentação Teórica A Teoria dos Grafos: Fundamentos e Importância A teoria dos grafos é uma área central da matemática discreta, essencial para a modelagem de problemas que envolvem relações e conectividades. Formalmente introduzida por Leonhard Euler no século XVIII, com o famoso problema das pontes de Königsberg, a teoria

dos grafos evoluiu para se tornar uma ferramenta versátil em várias disciplinas, incluindo ciência da computação, engenharia, biologia e ciências sociais.

Um grafo é definido como uma estrutura composta por dois conjuntos: um conjunto V de vértices (ou nós) e um conjunto E de arestas (ou ligações), denotado por = (,) G=(V,E). Cada aresta conecta dois vértices, e sua representação pode variar conforme a aplicação. Grafos podem ser simples, onde não há laços (arestas conectando um nó a si mesmo) nem múltiplas arestas entre dois vértices, ou multigrafos, nos quais tais propriedades são permitidas.

Além disso, os grafos podem ser dirigidos ou não dirigidos. Nos dirigidos, cada aresta tem uma direção associada, formando um arco que conecta um vértice inicial a um vértice terminal. Esse tipo de grafo é amplamente usado em modelagens de fluxo de informação ou trânsito, como redes de computadores e sistemas de transporte. Por outro lado, grafos não dirigidos são mais comuns em redes que representam relações simétricas, como amizades em redes sociais.

1.1. Propriedades e Métricas de Grafos

Os grafos possuem diversas propriedades e métricas que ajudam a caracterizar suas estruturas. Algumas das mais relevantes incluem:

- Grau de um vértice: número de arestas conectadas a um vértice. Em grafos dirigidos, distingue-se o grau de entrada (arestas que chegam) e o grau de saída (arestas que saem). Conectividade: mede a robustez estrutural do grafo. Um grafo é dito conexo se existe pelo menos um caminho entre qualquer par de vértices. Ciclos e acíclicos: grafos que contêm caminhos fechados são cíclicos; caso contrário, são acíclicos. Grafos acíclicos dirigidos (DAGs) têm papel importante na computação e na análise de dependências. Diâmetro e caminho médio: o diâmetro é a maior distância entre dois vértices, enquanto o caminho médio indica a média dessas distâncias em todo o grafo. Esses valores são indicadores de eficiência na comunicação entre os nós. Representações Computacionais de Grafos A implementação computacional de grafos é um aspecto crucial para a sua manipulação e análise eficiente. As formas mais comuns de representação incluem:

- Matriz de Adjacência: uma abordagem simples, onde a presença de uma aresta entre vértices u e v é indicada por 1 1 (ou o peso da aresta) na posição (,) (u,v) da matriz. Essa representação é eficiente para operações que requerem acesso direto às conexões, mas é menos prática para grafos esparsos devido ao elevado consumo de memória. - Lista de Adjacência: armazena, para cada vértice, uma lista de seus vértices adjacentes. Este método é ideal para grafos com poucas conexões em relação ao número de vértices, reduzindo o consumo de memória e facilitando a execução de algoritmos como busca em largura (BFS) e busca em profundidade (DFS). Lista de Arestas: útil em cenários onde é necessário processar diretamente as conexões, pois armazena as arestas como pares de vértices, junto a informações adicionais como peso ou direção. Algoritmos para Análise de Grafos Os algoritmos de grafos desempenham um papel central na sua aplicação prática, resolvendo problemas que vão desde a identificação de caminhos mínimos até a detecção de componentes fortemente conectados. Alguns algoritmos notáveis incluem:

Dijkstra: utilizado para encontrar o caminho mais curto de um vértice a todos os outros em grafos com arestas de peso não negativo. Bellman-Ford: semelhante ao

Dijkstra, mas pode lidar com pesos negativos. Busca em largura (BFS) e profundidade (DFS): exploram grafos de maneira sistemática, sendo a base para diversos problemas, como detecção de ciclos e ordenação topológica. PageRank: originalmente desenvolvido pelo Google, mede a importância de vértices em um grafo direcionado, sendo amplamente usado na análise de redes sociais e motores de busca. Aplicações Reais da Teoria dos Grafos Os grafos são amplamente utilizados para modelar problemas do mundo real. Alguns exemplos incluem:

Redes de comunicação: como internet, onde os nós representam dispositivos e as arestas simbolizam conexões. Mapas e sistemas de navegação: grafos geográficos modelam estradas, ferrovias e rotas aéreas. Biologia e medicina: modelagem de interações em redes metabólicas, redes de proteínas e cadeias alimentares. Análise de redes sociais: estudos sobre centralidade, comunidades e influência em plataformas como Facebook e Twitter. Ferramentas Computacionais

1.2. Aplicações de Grafos

Grafos são amplamente utilizados em algoritmos de busca de caminho, análise de conectividade, detecção de comunidades em redes sociais, modelagem de fluxos em sistemas e análise de redes elétricas.

2. Metodologia

Para desenvolvimento e organização da biblioteca de grafos, foi utilizado setuptools para empacotamento e distribuição. O foco é manter uma estrutura de diretórios clara e modular, separando responsabilidades para facilitar o desenvolvimento, manutenção e escalabilidade do projeto.

2.1. Organização

A biblioteca é organizada em um sistema de diretórios bem definido, conforme descrito abaixo:

Diretório dados/ Contém os arquivos de dados que servem como entrada para o trabalho. Exemplo: listas de adjacências, matrizes de adjacências ou conjuntos de nós e arestas.

Diretório models/ Inclui as classes e funções centrais que definem e manipulam grafos.

Diretório tests/ Armazena os arquivos de testes unitários para validar o funcionamento dos módulos. Utiliza frameworks como unittest ou pytest.

Diretório utils/ Contém scripts auxiliares que suportam as funções principais.

2.2. Utilização do setuptools

O setuptools é usado para empacotar a biblioteca, facilitando sua distribuição e instalação.

Configuração do setup.py O arquivo setup.py configura os metadados e dependências do projeto. Um exemplo básico:

```
1
2 from setuptools import setup, find_packages
```

```

setup(
    name="graph_library",
    version="0.1.0",
    description="Uma biblioteca de grafos modular e eficiente.",
    author="Seu Nome",
    author_email="seu.email@example.com",
    packages=find_packages(exclude=["tests", "dados"]),
    install_requires=[
        "numpy>=1.20.0",
    ],
    python_requires=">=3.7",
)
```

```

from grafo import Grafo
from tests.teste_desempenho import teste_desempenho

def menu():
    grafos_prontos = {
        "1": {
            'arestas': [(0, 1), (1, 2), (2, 3), (3, 0)],
            'dirigido': False
        },
        "2": {
            'arestas': [(0, 1), (1, 2), (2, 3), (3, 0), (3, 1)],
            'dirigido': False
        },
        "3": {
            'arestas': [(0, 1), (1, 2), (2, 0)],
            'dirigido': True
        },
        "4": {
            'arestas': [(0, 1), (1, 2), (2, 3), (3, 4), (4, 0)],
            'dirigido': False
        },
        "5": {
            'arestas': [(0, 1), (1, 2), (2, 0), (2, 3), (3, 4), (4, 2)],
            'dirigido': True
        },
        "6": {
            'arestas': [(0, 1), (1, 2), (2, 3)],
            'dirigido': False
        },
        "7": {
            'arestas': [(0, 1), (1, 2), (2, 3), (3, 0), (3, 4), (4, 5), (5, 3)
            'dirigido': True
        },
        "8": {
```

```
            'arestas': [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7)
        (8, 9), (9, 10), (10, 11), (11, 12), (12, 13), (13, 14), (14, 15), (15
            'dirigido': True
            },
    }
    while True:
        print("\nEscolha as opcoes abaixo:")
        print("1. Analisar Grafos Prontos")
        print("2. Criar Grafo Manualmente")
        print("3. Realizar Teste de Desempenho (Parte 2)")
        print("4. Sair")
        try:
            opcao = int(input("Escolha uma opcao: "))
        except ValueError:
            print("Entrada invalida. Por favor, digite um numero.")
            continue
        if opcao == 1:
            for nome, info in grafos_prontos.items():
                arestas = info['arestas']
                dirigido = info['dirigido']
                num_vertices = max(max(u, v) for u, v in arestas) + 1
                grafo_nome = f"Grafo_{nome}"
                grafo = Grafo(num_vertices, dirigido, nome=grafo_nome)
                for u, v in arestas:
                    grafo.adicionar_aresta(u, v)
                print(f"\n{grafo.nome}:")
                print(f"O grafo e {'direcionado' if grafo.dirigido else 'nao d
                print(f"Vertices: {grafo.num_vertices}")
                print(f"Arestas: {grafo.contar_vertices_arestas()[1]}")
                pontes_naive = grafo.identificar_pontes_naive()
                pontes_tarjan = grafo.identificar_pontes_tarjan()
                print("Pontes (Naive):", [(u + 1, v + 1) for u, v in pontes_na
                print("Pontes (Tarjan):", [(u + 1, v + 1) for u, v in pontes_t
                articulacoes = grafo.identificar_articulacoes()
                print("Articulacoes:", [v + 1 for v in articulacoes])
                if dirigido:
                    print("Fortemente Conexo:", grafo.grafo_fortemente_conexo(
                    print("Conexo Fraco:", grafo.grafo_conexo_fraco())
                    print("Semi-fortemente Conexo:", grafo.grafo_semi_fortemen
                else:
                    print("Conexo:", grafo.grafo_conexo())
                grafo.exportar_para_gexf(f"{grafo.nome}.gexf")
                grafo.exportar_para_ppm(f"{grafo.nome}.ppm")
                grafo.exportar_para_txt(f"{grafo.nome}.txt")
                print(f"Grafo '{grafo.nome}' exportado para os formatos GEXF,
        elif opcao == 2:
            try:
                num_vertices = int(input("Digite o numero de vertices: "))
                if num_vertices <= 0:
```

```
                    print("O numero de vertices deve ser positivo.")
                    continue
                dirigido = input("O grafo e direcionado? (s/n): ").lower() ==
                nome_grafo = input("Digite o nome do grafo: ").strip()
                if not nome_grafo:
                    nome_grafo = "Grafo_Manual"
            except ValueError:
                print("Entrada invalida. Por favor, digite um numero inteiro."
                continue
            grafo = Grafo(num_vertices, dirigido, nome=nome_grafo)
            while True:
                print("\n1. Adicionar Aresta")
                print("2. Remover Aresta")
                print("3. Verificar Adjacencia")
                print("4. Exibir Lista de Adjacencia")
                print("5. Exibir Matriz de Adjacencia")
                print("6. Exibir Matriz de Incidencia")
                print("7. Verificar Conectividade")
                print("8. Identificar Pontes")
                print("9. Identificar Articulacoes")
                print("10. Exportar Grafo")
                print("11. Exportar para PPM")
                print("12. Voltar")
                try:
                    escolha = int(input("Escolha uma opcao: "))
                except ValueError:
                    print("Entrada invalida. Por favor, digite numeros inteiro
                    continue
                if escolha == 1:
                    try:
                        u = int(input(f"Digite o vertice u (1 a {grafo.num_ver
                        v = int(input(f"Digite o vertice v (1 a {grafo.num_ver
                        if u < 0 or u >= grafo.num_vertices or v < 0 or v >= g
                            print(f"Erro: Vertices validos estao entre 1 e {gr
                            continue
                        peso_input = input("Digite o peso da aresta (padrao 1)
                        peso = int(peso_input) if peso_input else 1
                        label = input("Digite o rotulo da aresta (opcional): "
                        grafo.adicionar_aresta(u, v, peso, label)
                        print(f"Aresta ({u + 1}, {v + 1}) adicionada!")
                    except ValueError:
                        print("Entrada invalida. Por favor, digite numeros int
                elif escolha == 2:
                    try:
                        u = int(input(f"Digite o vertice u (1 a {grafo.num_ver
                        v = int(input(f"Digite o vertice v (1 a {grafo.num_ver
                        if u < 0 or u >= grafo.num_vertices or v < 0 or v >= g
                            print(f"Erro: Vertices validos estao entre 1 e {gr
                            continue
```

```
                grafo.remover_aresta(u, v)
                print(f"Aresta ({u + 1}, {v + 1}) removida!")
            except ValueError:
                print("Entrada invalida. Por favor, digite numeros int
        elif escolha == 3:
            try:
                u = int(input(f"Digite o vertice u (1 a {grafo.num_ver
                v = int(input(f"Digite o vertice v (1 a {grafo.num_ver
                if u < 0 or u >= grafo.num_vertices or v < 0 or v >= g
                    print(f"Erro: Vertices validos estao entre 1 e {gr
                    continue
                if grafo.checar_adjacencia_vertices(u, v):
                    print(f"Aresta ({u + 1}, {v + 1}) existe!")
                else:
                    print(f"Aresta ({u + 1}, {v + 1}) nao existe.")
            except ValueError:
                print("Entrada invalida. Por favor, digite numeros int
        elif escolha == 4:
            grafo.exibir_lista_adjacencia()
        elif escolha == 5:
            grafo.exibir_matriz_adjacencia()
        elif escolha == 6:
            grafo.exibir_matriz_incidencia()
        elif escolha == 7:
            if grafo.dirigido:
                print("Fortemente Conexo:", grafo.grafo_fortemente_con
                print("Conexo Fraco:", grafo.grafo_conexo_fraco())
                print("Semi-fortemente Conexo:", grafo.grafo_semi_fort
            else:
                print("Conexo:", grafo.grafo_conexo())
        elif escolha == 8:
            pontes_naive = grafo.identificar_pontes_naive()
            pontes_tarjan = grafo.identificar_pontes_tarjan()
            pontes_naive_exib = [(u + 1, v + 1) for u, v in pontes_nai
            pontes_tarjan_exib = [(u + 1, v + 1) for u, v in pontes_ta
            print("Pontes (Naive):", pontes_naive_exib)
            print("Pontes (Tarjan):", pontes_tarjan_exib)
        elif escolha == 9:
            articulacoes = [v + 1 for v in grafo.identificar_articulac
            print("Articulacoes:", articulacoes)
        elif escolha == 10:
            nome = input("Digite o nome base dos arquivos (sem extensa
            if not nome:
                nome = grafo.nome.replace(" ", "_")
            grafo.exportar_para_gexf(f"{nome}.gexf")
            grafo.exportar_para_ppm(f"{nome}.ppm")
            grafo.exportar_para_txt(f"{nome}.txt")
            print("Exportacao concluida.")
        elif escolha == 11:
```

```
                    nome_ppm = input("Digite o nome do arquivo PPM (com extens
                    if not nome_ppm.endswith('.ppm'):
                        print("Erro: O nome do arquivo deve terminar com '.ppm
                        continue
                    grafo.exportar_para_ppm(nome_ppm)
                elif escolha == 12:
                    export_nome = grafo.nome.replace(" ", "_")
                    grafo.exportar_para_gexf(f"{export_nome}.gexf")
                    grafo.exportar_para_ppm(f"{export_nome}.ppm")
                    grafo.exportar_para_txt(f"{export_nome}.txt")
                    print(f"Grafo '{grafo.nome}' exportado automaticamente apo
                    break
                else:
                    print("Opcao invalida, tente novamente.")
        elif opcao == 3:
            teste_desempenho()
        elif opcao == 4:
            print("Saindo do programa. Ate logo!")
            break
        else:
            print("Opcao invalida, tente novamente.")

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

2.3. Desenvolvimento

O projeto visa utilização de boas práticas, tais como:

Modularidade: Cada arquivo ou módulo contém apenas uma responsabilidade.

Testes Automatizados: Scripts no diretório tests/ para testes de casos de uso.

Documentação: Forneçimento de comentários e documentação clara para cada função/módulo.

2.4. Representações de Grafos

A biblioteca suporta:

- Listas de adjacência.
- Matrizes de adjacência.
- Listas de arestas.

2.5. Classes Principais

A biblioteca inclui as seguintes classes principais:

- **Classe Vértice:** Representa um nó do grafo.
- **Classe Aresta:** Representa uma conexão entre dois vértices.
- **Classe Grafo:** Gerencia a estrutura global do grafo, incluindo as operações de manipulação e análise.

3. Parte 1: Implementação de Funcionalidades

3.1. Manipulação de Grafos

A biblioteca permite criar grafos dirigidos ou não dirigidos, adicionar e remover arestas, e verificar adjacências.

3.2. Ponderação e Rotulação

Cada aresta pode ser ponderada e rotulada, possibilitando modelar grafos com informações adicionais.

3.3. Análise de Conectividade

A conectividade é avaliada por meio de funções que verificam se o grafo é conectado (não dirigido) ou fortemente conectado (dirigido).

3.4. Detecção de Pontes e Articulações

Foram implementados dois métodos para detecção de pontes:

- **Método Naive:** Testa a remoção de cada aresta e verifica a conectividade.
- **Método de Tarjan:** Algoritmo eficiente baseado em buscas em profundidade (DFS).

4. Parte 2: Resultados

4.1. Comparação de Desempenho

Foram realizados testes de desempenho entre os métodos Naive e o de Tarjan. Resultados mostram que o método de Tarjan é significativamente mais eficiente em grafos grandes.

Table 1. Comparação de desempenho (tempo em segundos).

Grafo	Método Naive	Método Tarjan
Grafo 1	0.012	0.002
Grafo 2	0.034	0.005
Grafo 3	0.210	0.019

4.2. Visualização com Gephi

Os grafos foram exportados em formatos compatíveis com o Gephi, permitindo análises visuais detalhadas.

5. Conclusão

A *Graphs Kit* se mostrou uma ferramenta eficaz para manipulação e análise de grafos, com destaque para a implementação eficiente do algoritmo de Tarjan. Futuras melhorias incluem a implementação de algoritmos para detecção de ciclos e otimizações para manipulação de grafos muito grandes.

References

[1] Euler, L. “Solutio problematis ad geometriam situs pertinentis.” Commentarii academiae scientiarum Petropolitanae 8 (1736): 128-140.

[2] Tarjan, R. “Depth-first search and linear graph algorithms.” SIAM journal on computing 1.2 (1972): 146-160.

[3] Dijkstra, E. W. “A note on two problems in connexion with graphs.” Numerische Mathematik 1.1 (1959): 269-271.