Sistema de Análise de Grafos – Algoritmos Clássicos

Integrantes: Diogo Lamera, Gabriel Xavier, Matheus Amaral, Max Augusto e Ronaldo Soares

Turma: Sistemas de Informação

Data: 07/12/2025

Link do repositório: https://github.com/Max-Augusto/Grafos

1. Introdução

Este trabalho tem como objetivo desenvolver um Sistema de Análise de Grafos, capaz de representar grafos ponderados não direcionados e executar os principais algoritmos clássicos estudados na disciplina: DFS**,** BFS**,** Dijkstra **e** Prim.

O projeto busca consolidar os conhecimentos teóricos através da implementação prática dos algoritmos e oferecer uma ferramenta que permita visualizar, analisar e testar diferentes estruturas de grafos.

Além disso, foram incluídas funcionalidades opcionais como leitura do grafo via arquivo .txt/.csv e visualização gráfica do grafo utilizando a biblioteca NetworkX, tornando o sistema mais completo e amigável ao usuário.

---

2. Descrição dos Algoritmos

2.1 DFS – Depth-First Search (Busca em Profundidade)

A DFS é um algoritmo de travessia de grafos que explora o mais profundamente possível antes de retroceder. É implementada utilizando recursão ou pilha.
Aplicação**:** detecção de componentes conectados, verificação de caminhos, problemas de backtracking.

2.2 BFS – Breadth-First Search (Busca em Largura)

A BFS explora primeiro todos os vizinhos imediatos antes de avançar para níveis mais profundos. Utiliza uma fila para controlar a ordem de visita.
Aplicação**:** menor caminho em grafos não ponderados, distâncias mínimas, análise de camadas.

2.3 Dijkstra – Menor Caminho em Grafo Ponderado

O algoritmo de Dijkstra encontra o menor caminho entre dois vértices em um grafo com pesos positivos. Utiliza uma fila de prioridade (heap) para sempre expandir o vértice de menor distância acumulada.

Aplicação**:** rotas, redes, logística, sistemas de navegação.

2.4 Prim – Árvore Geradora Mínima (MST)

O algoritmo de Prim encontra uma árvore geradora mínima selecionando arestas de menor peso que conectam novos vértices ao conjunto já visitado.
Aplicação**:** redes elétricas, telecomunicações, planejamento de conexões de custo mínimo.

---

3. Explicação dos Principais Trechos do Código

3.1 Representação do Grafo `self.adjacencia = defaultdict(list)`

O grafo é representado como lista de adjacência, ideal para buscas e algoritmos de caminhos mínimos.

3.2 Adicionar Aresta

```
self.adicionar_aresta(u, v, peso)
```

Como o grafo é não direcionado, adicionamos u→v e v→u.

3.3 DFS

```
def visitar(v):
    visitados.add(v)
resultado.append(v)     for vizinho, _ in
self.adjacencia[v]:
        if vizinho not in visitados:
visitar(vizinho)
```

Implementação recursiva seguindo exploração profunda.

3.4 BFS

```
visitados = set([inicio]) fila
= deque([inicio])
```

Fila controla a ordem em que os nós são visitados.

3.5 Dijkstra

```
distancias = {v: float('inf') for v in self.adjacencia} heap
= [(0, inicio)]
```

Utiliza heap para escolher a próxima aresta com menor peso acumulado.

3.6 Prim `heap = [(0, None, vertices[0])]`

A cada etapa adiciona a menor aresta que conecta um vértice novo à árvore.

3.7 Visualização do Grafo `nx.draw(G, pos, with_labels=True)`

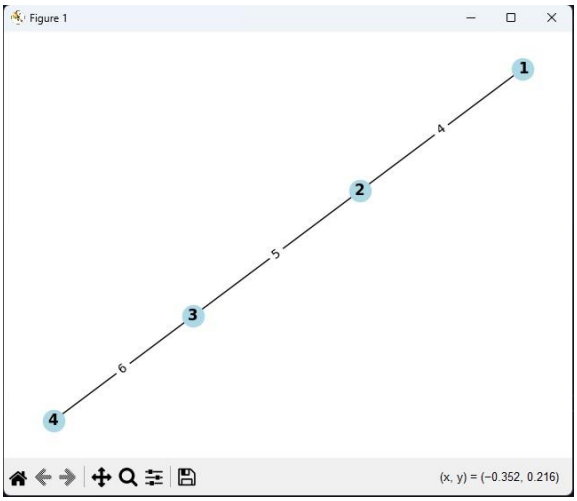
Utiliza NetworkX + Matplotlib para gerar imagem do grafo com pesos.

---

4. Execução Prática


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK  COMMENTS

PS C:\Users\Max\Desktop\projetos_python\grafos> python grafos.py
DFS: [1, 2, 3, 4]
BFS: [1, 2, 3, 4]
Dijkstra (1 -> 4): ([1, 2, 3, 4], 15)
Árvore Geradora Mínima (Prim): [(1, 2, 4), (2, 3, 5), (3, 4, 6)]
```



---

5. Resultados Obtidos

Com o arquivo `grafo.txt` contendo:

```
1 2 4
2 3 5
3 4 6
```

Os resultados foram:

DFS

```
[1, 2, 3, 4]
```

BFS

```
[1, 2, 3, 4]
```

Dijkstra (1 → 4)

Caminho: `[1, 2, 3, 4]`
Custo total: `15`

Prim – Árvore Geradora Mínima

```
(1,2,4)
(2,3,5)
(3,4,6)
```

Custo total: `15`

---

6. Conclusão

O desenvolvimento deste sistema permitiu consolidar o entendimento dos algoritmos clássicos de grafos, desde travessias até cálculo de caminhos mínimos e criação de árvores geradoras.

A implementação prática reforçou conceitos como:

- estruturação de grafos com lista de adjacência
- uso de filas, recursão e fila de prioridade
- manipulação de arquivos
- visualização e análise estrutural de grafos

Entre as maiores dificuldades estiveram o tratamento correto das estruturas de dados e a depuração dos algoritmos, especialmente o Dijkstra e o Prim. Ao final, o sistema atingiu plenamente os objetivos propostos e se mostrou funcional e eficiente.