

Grafos

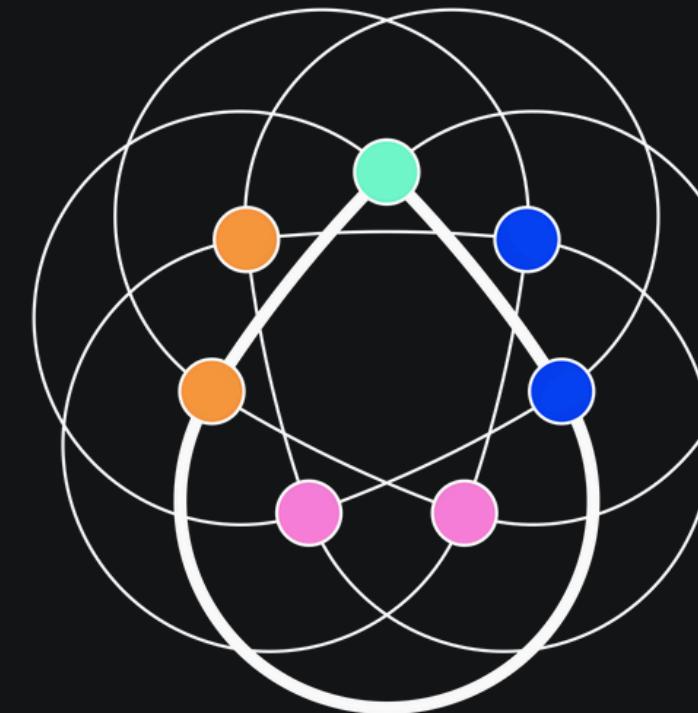
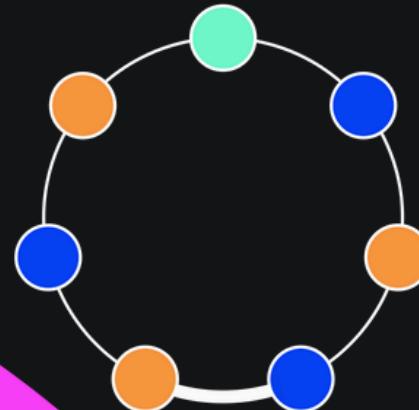
Conceitos, Representações, DFS, BFS, Dijkstra

ByteBusters
Alicia da Silva Feliciano
Gabriel Freitas Santos
Giovanna Maria Alves Evangelista

Grafos

Introdução

- Grafos: Estruturas compostas por vértices (nós) e arestas (conexões).
- Podem ser direcionados ou não direcionados e ponderados ou não ponderados.
- Usados para modelar relações e conexões entre objetos.



Aplicação

- Redes sociais: Conexões entre usuários.
- Roteamento: Cálculo de rotas em GPS e redes de comunicação.
- Circuitos elétricos: Modelagem de conexões entre componentes.
- Biologia: Representação de redes neurais e genéticas.
- Inteligência artificial: Processamento de dados e aprendizado de máquina.

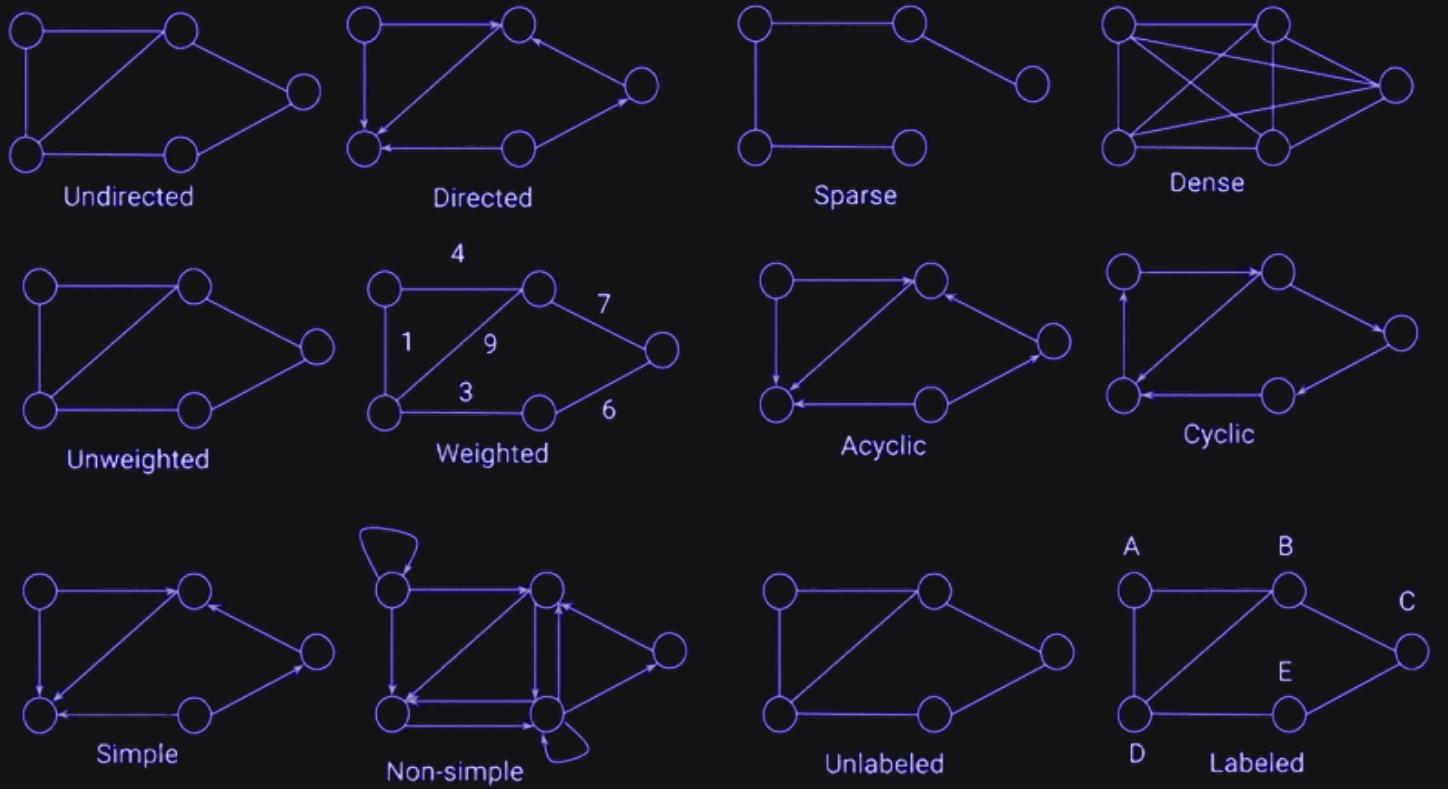
Grafos

Conceitos básicos

- Vértices (Nodos): Representam os objetos ou entidades no grafo, como pessoas, cidades ou computadores.
- Areias: São as conexões entre os vértices, que representam a relação entre as entidades.

Tipos

- Direcionado: Areias com direção (exemplo: Twitter, onde o "seguir" não é necessariamente recíproco).
- Não Direcionado: Areias sem direção (exemplo: Facebook, amizade bidirecional).
- Ponderado: Areias com valores (exemplo: mapa de rotas, com distâncias).
- Não Ponderado: Areias sem valores (exemplo: rede social sem intensidade de conexão).
- Cíclico: Contém ciclos (exemplo: rede de tráfego).
- Acíclico: Não contém ciclos (exemplo: árvores).



Grafos

Representações de grafos

Matriz de adjacência

- Tabela onde as células indicam conexões entre vértices.
- Usa $O(V^2)$ de espaço, mesmo para poucos vínculos.
- Melhor para grafos densos (muitas conexões).

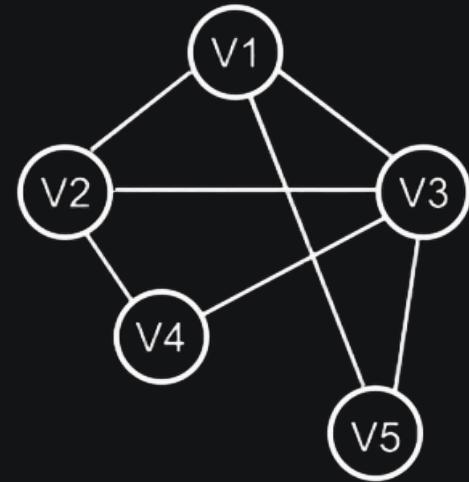
$V1 \rightarrow V2, V3, V5$

$V2 \rightarrow V1, V3, V4$

$V3 \rightarrow V1, V2, V4, V5$

$V4 \rightarrow V2, V3$

$V5 \rightarrow V1, V3$



	V1	V2	V3	V4	V5
V1	0	1	1	0	1
V2	1	0	1	1	0
V3	1	1	0	1	1
V4	0	1	1	0	0
V5	1	0	1	0	0

Lista de adjacência

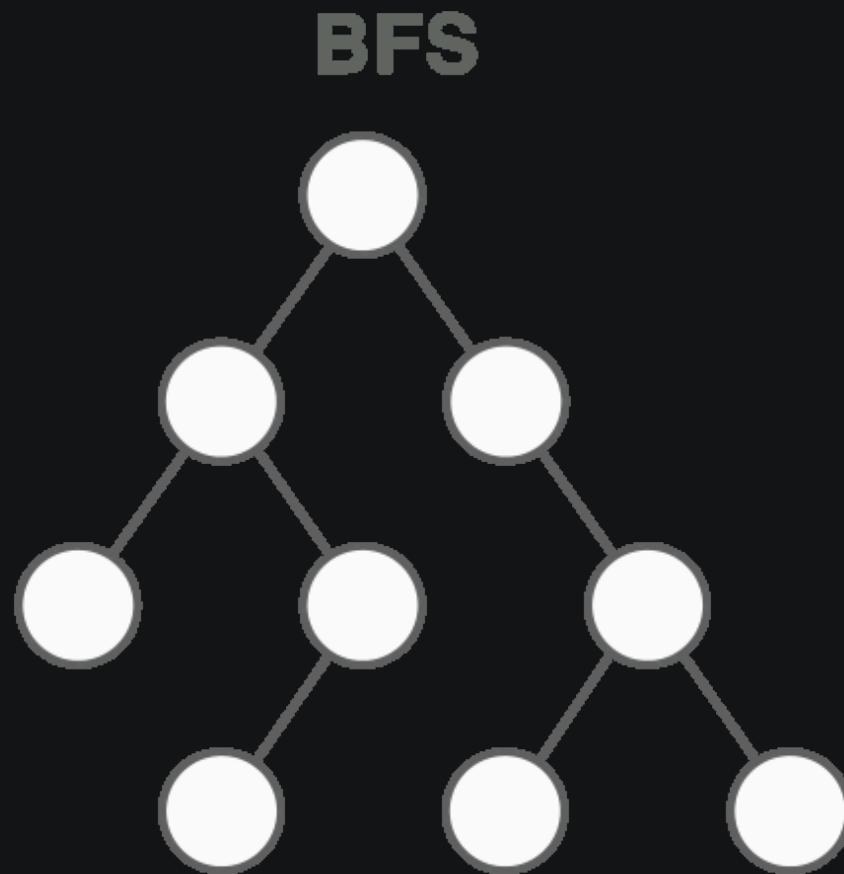
- Cada vértice aponta para seus vizinhos diretamente.
- Usa menos espaço, eficiente para grafos esparsos (poucas conexões).
- Melhor para buscas como DFS e BFS.

Grafos

Busca em Largura (BFS)

Conceito

- Exploração nível por nível, visitando primeiro os vizinhos mais próximos.
- Utiliza uma fila (FIFO – Primeiro a entrar, primeiro a sair) para gerenciar os vértices.



Aplicações

- Menor caminho em grafos não ponderados.
- Verificação de conectividade entre vértices.
- Navegação em redes sociais (sugestões de amigos).
- Resolução de quebra-cabeças (como o Cubo Mágico).

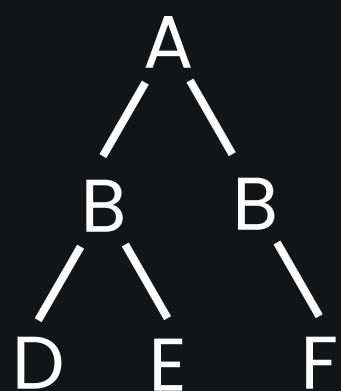
Grafos

Busca em Largura (BFS)

Passo a Passo da Execução

1. Inicia com um vértice e o adiciona à fila.
2. Remove o primeiro da fila, marca como visitado e processa.
3. Adiciona todos os vizinhos não visitados à fila.
4. Repete até visitar todos os vértices conectados.

Grafo Representado



Pseudocódigo

ALGORITMO BFS(GRAFO, INICIO)

 CRIAR CONJUNTO VISITADO

 CRIAR FILA VAZIA

 ENFILEIRAR INICIO NA FILA

 ENQUANTO FILA NÃO ESTIVER VAZIA FAÇA

 REMOVER PRIMEIRO ELEMENTO DA FILA → VERTICE

 SE VERTICE NÃO ESTIVER EM VISITADO ENTÃO

 IMPRIMIR VERTICE

 ADICIONAR VERTICE EM VISITADO

 ENFILEIRAR TODOS OS VIZINHOS NÃO VISITADOS DO VERTICE

FIM ALGORITMO

Exemplo de uso

```
GRAFO = {  
  'A': ['B', 'C'],  
  'B': ['A', 'D', 'E'],  
  'C': ['A', 'F'],  
  'D': ['B'],  
  'E': ['B', 'F'],  
  'F': ['C', 'E']  
}
```

BFS(GRAFO, 'A')

Passo a passo da saída

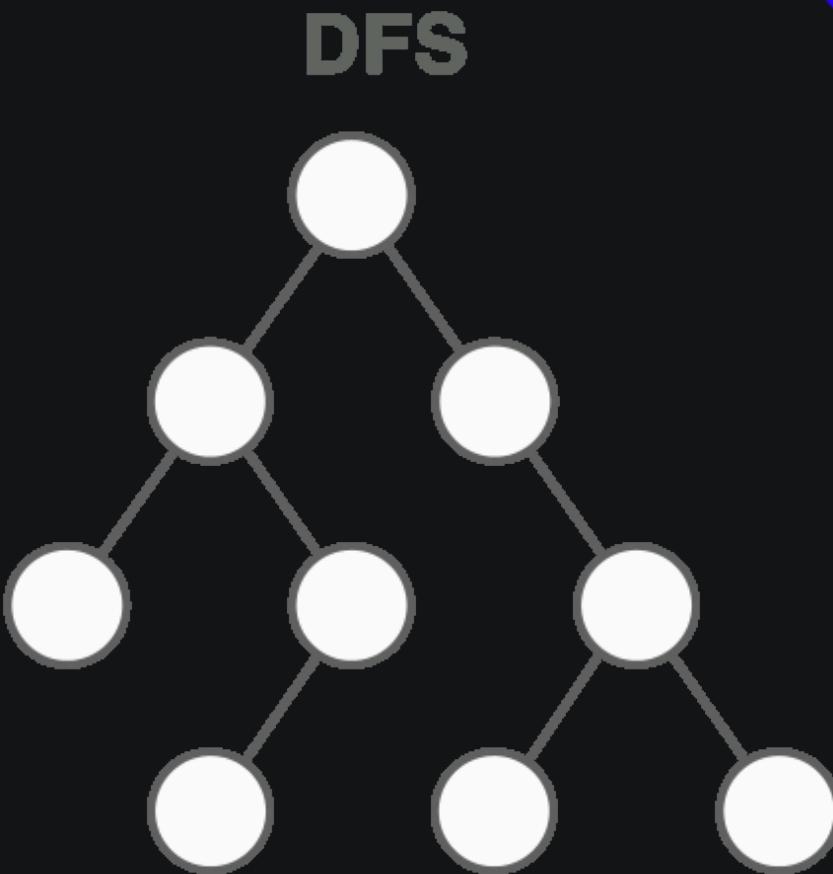
1. Começa em A → Fila: [B, C] → Visitado: {A}
2. Processa B → Fila: [C, D, E] → Visitado: {A, B}
3. Processa C → Fila: [D, E, F] → Visitado: {A, B, C}
4. Processa D → Fila: [E, F] → Visitado: {A, B, C, D}
5. Processa E → Fila: [F] → Visitado: {A, B, C, D, E}
6. Processa F → Fila: [] → Visitado: {A, B, C, D, E, F}

Grafos

Busca em Profundidade (DFS)

Conceito

- Explora os caminhos ao máximo antes de voltar, seguindo um ramo até o fim antes de retornar.
- Utiliza uma pilha (pode ser implementada com recursão ou estrutura explícita).



Aplicações

- Detecção de ciclos em grafos direcionados e não direcionados.
- Ordenação topológica de tarefas com dependências (exemplo: compilação de código).
- Busca de componentes conectados em grafos.
- Resolução de labirintos e jogos baseados em caminhos.

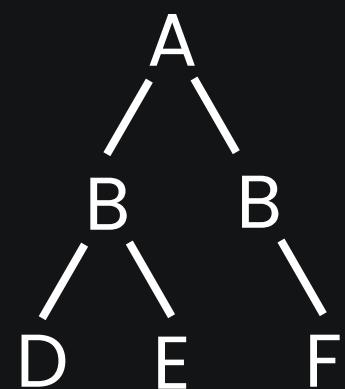
Grafos

Busca em Largura (BFS)

Passo a Passo da Execução

1. Inicia com um vértice e o adiciona à pilha.
2. Remove do topo, marca como visitado e processa.
3. Adiciona o primeiro vizinho não visitado à pilha e continua.
4. Se não houver vizinhos disponíveis, volta ao anterior.
5. Repete até visitar todos os vértices.

Grafo Representado



Pseudocódigo

```
ALGORITMO DFS(GRAFO, VERTICE, VISITADO)
    SE VERTICE NÃO ESTIVER EM VISITADO ENTÃO
        IMPRIMIR VERTICE
        ADICIONAR VERTICE EM VISITADO
        PARA CADA VIZINHO EM GRAFO[VERTICE] FAÇA
            DFS(GRAFO, VIZINHO, VISITADO)

    FIM ALGORITMO
```

Exemplo de uso

```
GRAFO = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
```

```
DFS(GRAFO, 'A', CONJUNTO VAZIO)
```

Passo a passo da saída

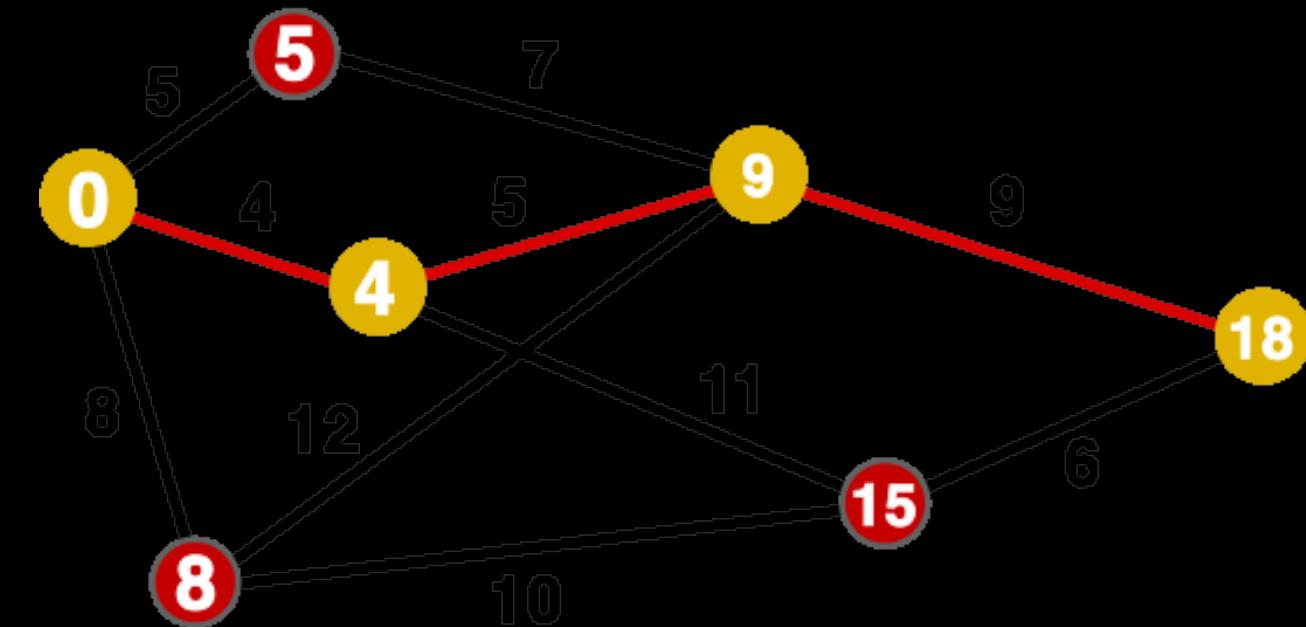
1. A → Visitado: {A} → DFS para B.
2. B → Visitado: {A, B} → DFS para D.
3. D → Visitado: {A, B, D} → Retorna B.
4. Em B, chama DFS para E.
5. E → Visitado: {A, B, D, E} → DFS para F.
6. F → Visitado: {A, B, D, E, F} → DFS para C.
7. C → Visitado: {A, B, D, E, F, C}.

Grafos

Algoritmo de Dijkstra

Conceito

- Resolve o problema do menor caminho em grafos ponderados (arestas com peso).
- Utiliza uma fila de prioridade (heap) para sempre expandir o vértice com a menor distância conhecida.



Aplicações

- Navegação GPS e mapas (Google Maps, Waze).
- Roteamento de redes (protocolos como OSPF).
- Logística e transporte (planejamento de rotas).
- Jogos e IA (caminho mais curto para NPCs).

Grafos

Algoritmo de Dijkstra

Passo a Passo da Execução

1. Inicializa as distâncias de todos os vértices como infinito, exceto o inicial (distância 0).
2. Adiciona o vértice inicial na fila de prioridade (heap).
3. Remove o vértice com menor distância da fila, processa seus vizinhos e atualiza a menor distância encontrada.
4. Se uma menor distância for encontrada, atualiza o valor e adiciona o vizinho na fila.
5. Repete até visitar todos os vértices acessíveis.

Pseudocódigo

```
ALGORITMO DIJKSTRA(GRAFO, INICIO)
    PARA CADA VERTICE V EM GRAFO FAÇA
        DISTANCIA[V] ← ∞
    DISTANCIA[INICIO] ← 0
    FILA ← [(0, INICIO)] // Fila de prioridade

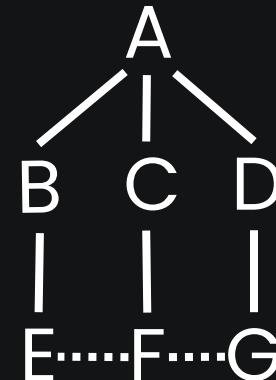
    ENQUANTO FILA NÃO ESTIVER VAZIA FAÇA
        (DIST, VERTICE) ← REMOVER MENOR DE FILA
        PARA CADA (VIZINHO, PESO) EM GRAFO[VERTICE] FAÇA
            NOVA_DISTANCIA ← DISTANCIA[VERTICE] + PESO
            SE NOVA_DISTANCIA < DISTANCIA[VIZINHO] ENTÃO
                DISTANCIA[VIZINHO] ← NOVA_DISTANCIA
                ADICIONAR (NOVA_DISTANCIA, VIZINHO) NA FILA

    RETORNAR DISTANCIA
FIM ALGORITMO
```

Grafos

Algoritmo de Dijkstra

Grafo Representado



Passo a Passo da Saída (Dijkstra a partir de A)

- Começa em A → Distâncias: {A: 0, B: ∞ , C: ∞ , D: ∞ , E: ∞ , F: ∞ , G: ∞ }
- Visita B (custo 4), C (custo 2) e D (custo 5)
- Visita C e encontra F (custo 5)
- Visita B e encontra E (custo 5)
- Visita F e encontra G (custo 6)
- Visita E e G → Fim da execução.

Pesos das Arestas

```
GRAFO = {  
    'A': [('B', 4), ('C', 2), ('D', 5)],  
    'B': [('A', 4), ('E', 1)],  
    'C': [('A', 2), ('F', 3)],  
    'D': [('A', 5), ('G', 2)],  
    'E': [('B', 1), ('F', 1)],  
    'F': [('C', 3), ('E', 1), ('G', 1)],  
    'G': [('D', 2), ('F', 1)]  
}
```

Resultado Final (Menor Distância de A para os Outros Vértices)

A → A = 0
A → B = 4
A → C = 2
A → D = 5
A → E = 5
A → F = 5
A → G = 6

Grafos

Comparativo

Algoritmo	Tipo de Grafo	Complexidade	Estrutura de Dados	Tipo de Exploração	Uso de Arestas	Ciclos	Limitações
BFS	Direcionados e não direcionados	$O(V + E)$	Fila	Explora o grafo nível por nível (camada por camada).	Não considera os pesos das arestas.	Não detecta ciclos (explora nivelmente)	Não encontra o caminho mais curto em grafos ponderados
DFS	Direcionados e não direcionados	$O(V + E)$	Pilha (recursão)	Explora o grafo o mais profundo possível antes de voltar.	Não considera os pesos das arestas.	Detecta ciclos (importante em grafos não direcionados)	Pode não encontrar o caminho mais curto
Dijkstra	Funciona apenas em grafos direcionados ou não direcionados com pesos positivos	$O((V + E) \log V)$	Fila de prioridade	Explora o grafo com base no menor custo acumulado até o momento.	Considera os pesos das arestas (exclusivo para grafos ponderados).	Não detecta ciclos, apenas busca o caminho mínimo.	Não funciona corretamente com arestas de peso negativo



Obrigado!