Professor
Edson Pedro FERLIN

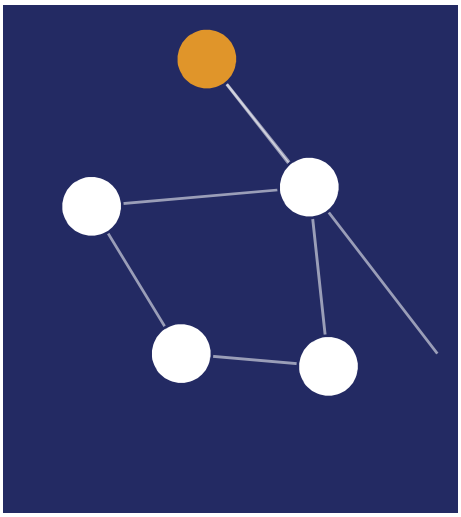
Estruturas de Dados

Grafos

Prof. Edson Pedro Ferlin

Professor
Edson Pedro FERLIN

Estruturas de Dados



AULA 7 - GRAFOS

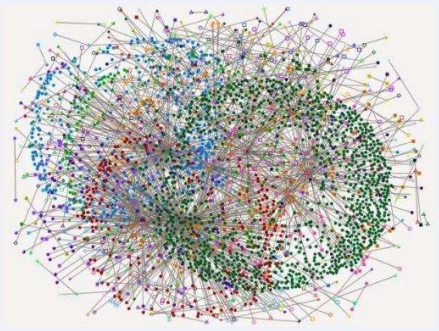
Grafos

Conceitos · Representações · Percursos · Aplicações

Onde encontramos grafos?

Os grafos aparecem em praticamente todas as áreas da Computação:

- Redes sociais
- GPS
- Redes de computadores
- Inteligência Artificial
- Jogos
- Bancos de Dados
- Internet
- Logística



O que é um grafo?

Um **grafo** é uma estrutura composta por vértices e arestas.

Vértices (nós)
Os pontos ou entidades do grafo

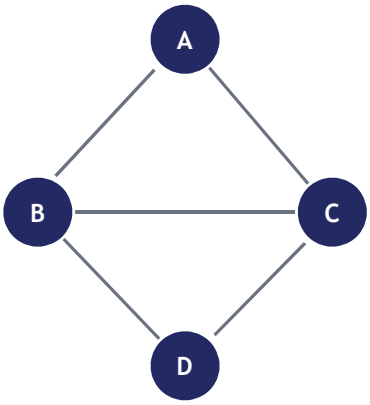
Arestas (ligações)
As conexões entre os vértices

Matematicamente

$$G = (V, E)$$

V conjunto de vértices
E conjunto de arestas

Exemplo de grafo



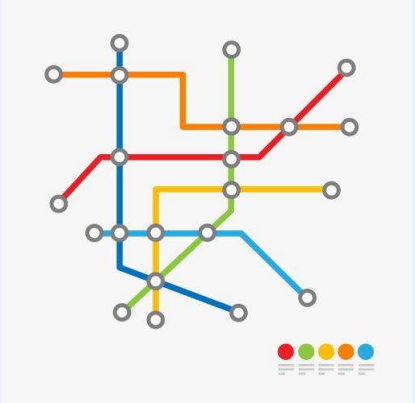
Vértices

A, B, C, D

Arestas

(A,B) (A,C) (B,C) (B,D) (C,D)

Um exemplo do mundo real



Rede de transporte

Cada estação, pessoa ou roteador representa um **vértice**, enquanto as conexões representam **arestas**.

Terminologia

Vértice
Ponto / nó do grafo

Aresta
Ligação entre dois vértices

Grau
Número de arestas de um vértice

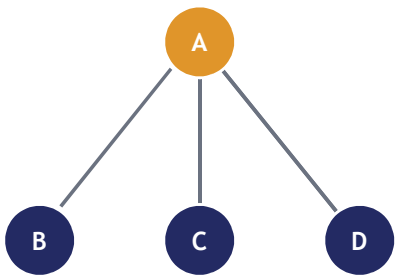
Caminho
Sequência de vértices conectados

Ciclo
Caminho que retorna à origem

Peso
Valor associado a uma aresta

Vizinho
Vértice diretamente conectado

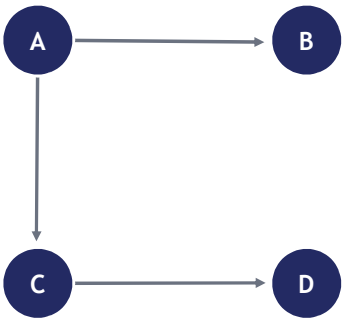
Grau de um vértice



O grau é o número de arestas incidentes a um vértice.

```
grau(A) = 3
```

Grafos direcionados

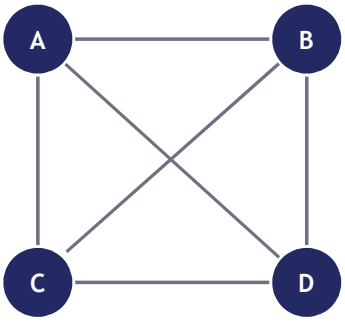


As arestas possuem direção — a ligação vai de um vértice para outro.

EXEMPLOS

- Instagram (seguir)
- Twitter / X
- Fluxogramas

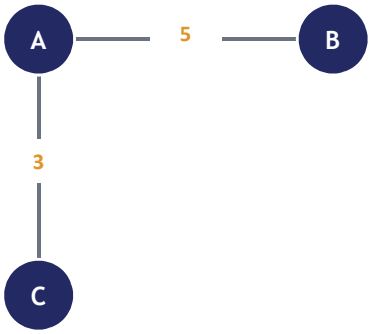
Grafos não direcionados



As arestas não possuem direção — a ligação é mútua entre os vértices.

A — B

Grafos ponderados



Cada aresta recebe um peso. Os pesos podem representar:

- **Distância**
- **Custo**
- **Tempo**
- **Largura de banda**

Como representar um grafo?

Existem duas formas principais de armazenar um grafo na memória:

1

Matriz de Adjacência

Uma tabela V×V onde cada célula indica se existe aresta entre dois vértices.

2

Lista de Adjacência

Cada vértice guarda uma lista dos seus vizinhos diretos.

Matriz de adjacência

	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	1
D	0	1	1	0

Como ler

1 → existe aresta
0 → não existe aresta
Ocupa espaço V^2 — ideal para grafos densos.

Lista de adjacência

A → B → C

B → A → C → D

C → A → B → D

D → B → C

Vantagem

Cada vértice aponta apenas para os seus vizinhos diretos — economiza memória em grafos esparsos.

Matriz × Lista de adjacência

Característica	Matriz	Lista
Memória	**Alta**	**Baixa**
Inserção	**Fácil**	**Fácil**
Busca de vizinhos	**Média**	**Muito rápida**
Grafos esparsos	**Não recomendado**	**Recomendado**

Criando um grafo em Python

```python
grafo = {
    "A": ["B", "C"],
    "B": ["A", "C", "D"],
    "C": ["A", "B", "D"],
    "D": ["B", "C"]
}
print(grafo)
```

IDEIA

Usamos um dicionário: a chave é o vértice e o valor é a lista de vizinhos.

Obtendo os vizinhos

```
print(grafo["A"])
```

RESULTADO

```
['B', 'C']
```


Adicionando um vértice

```
grafo["E"] = []
```

OBSERVAÇÃO

Criamos o vértice E com uma lista de vizinhos vazia.

Adicionando uma aresta

```
grafo["E"].append("A")
grafo["A"].append("E")
```

NÃO DIRECIONADO

Atualizamos os dois lados para registrar a ligação nos dois sentidos.

Percorrendo o grafo

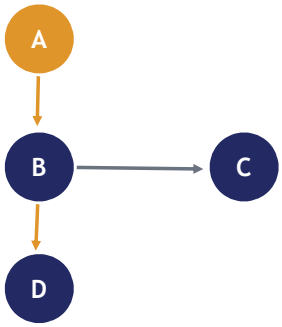
```
for v in grafo:
    print(v, "->", grafo[v])
```

O QUE FAZ

Itera sobre cada vértice e imprime a sua lista de vizinhos.

Busca em profundidade (DFS)

A DFS percorre um caminho até o fim antes de retroceder (backtracking).



Ordem de visita

A → B → D → C

Código — DFS (recursiva)

```python
def dfs(grafo, vertice, visitados=None):
    if visitados is None:
        visitados = set()
    visitados.add(vertice)
    print(vertice)
    for vizinho in grafo[vertice]:
        if vizinho not in visitados:
            dfs(grafo, vizinho, visitados)
```

Executando a DFS

```
dfs(grafo, "A")
```

SAÍDA POSSÍVEL

A
B
C
D

Busca em largura (BFS)

A BFS visita todos os vizinhos de um nível antes de avançar para o próximo.

Nível 0 A

Nível 1 B C

Nível 2 D

Percurso em ondas

A partir da origem, explora-se camada por camada — como ondas que se expandem no grafo.

Código — BFS (iterativa)

```python
from collections import deque

def bfs(grafo, inicio):
    visitados = set()
    fila = deque([inicio])
    while fila:
        v = fila.popleft()
        if v not in visitados:
            print(v)
            visitados.add(v)
            fila.extend(grafo[v])
```


Executando a BFS

```python
bfs(grafo, "A")
```

ESTRUTURA

A BFS usa uma fila (deque): o primeiro vértice a entrar é o primeiro a ser visitado.

DFS × BFS

DFS	BFS
Pilha	Fila
Recursiva	Iterativa
Explora profundidade	Explora níveis

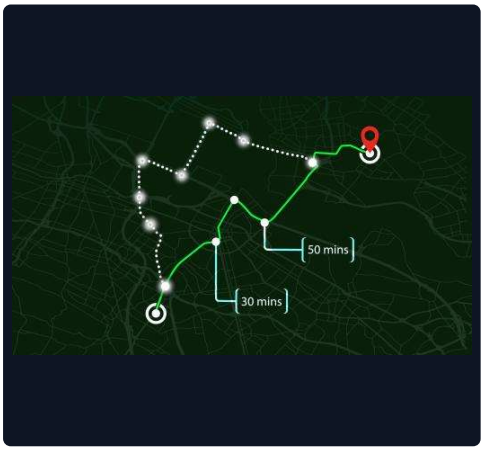
Complexidade dos percursos

Ambos os algoritmos visitam cada vértice e cada aresta uma vez:

DFS	BFS
$O(V + E)$	$O(V + E)$
V = vértices · E = arestas	V = vértices · E = arestas

Aplicações dos grafos

- GPS e navegação
- Redes sociais
- Sistemas de recomendação
- Roteamento de redes
- Dependências de software



Atividade - Grafo ponderado em Python

```python
grafo = {
    "A": {"B": 5, "C": 3},
    "B": {"A": 5, "D": 8},
    "C": {"A": 3, "D": 2},
    "D": {"B": 8, "C": 2}
}
```

IDEIA

Um dicionário aninhado: cada vizinho aponta para o peso da aresta.

Acessando o peso de uma aresta

```
print(grafo["A"]["B"])
```

RESULTADO

5

Biblioteca NetworkX

```
import networkx as nx

G = nx.Graph()
G.add_edge("A", "B")
G.add_edge("A", "C")
G.add_edge("B", "D")
```

NETWORKX

Biblioteca de Python para criar, manipular e analisar grafos com poucas linhas.

Desenhando o grafo

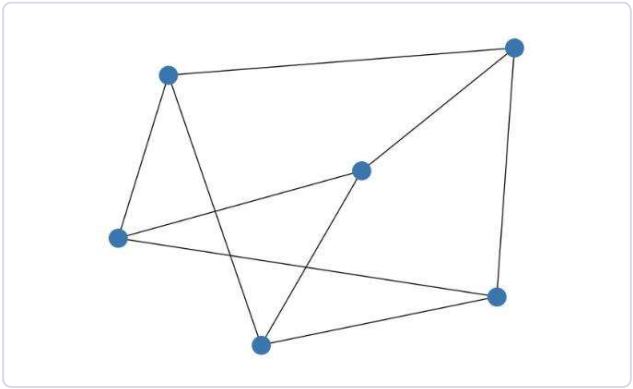
```python
import matplotlib.pyplot as plt
import networkx as nx

nx.draw(G,
        with_labels=True,
        node_size=1200)
plt.show()
```

VISUALIZAÇÃO

O matplotlib renderiza o grafo criado no NetworkX em uma figura.

Resultado esperado

O grafo desenhado pelo NetworkX, com vértices e arestas dispostos automaticamente:



Estruturas de dados – visão geral

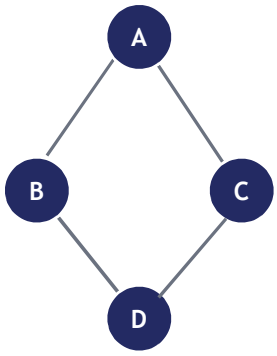
Estrutura	Organização
Vetor	Linear
Lista	Linear
Pilha	LIFO
Fila	FIFO
Árvore	Hierárquica
Grafo	**Rede**

Exercícios

Exercício 1 — Lista de adjacência

1

Construa o grafo abaixo utilizando lista de adjacência:



Dica

Represente cada vértice com a lista dos seus vizinhos, como fizemos com o dicionário em Python.

Exercício 2 — DFS e BFS

2

Implemente os percursos para o grafo do exercício anterior:

DFS

Busca em profundidade — use recursão e um conjunto de visitados.

BFS

Busca em largura — use uma fila (deque) de vértices.

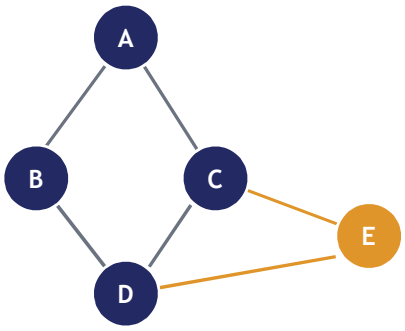
Professor
Edson Pedro FERLIN

Estruturas de Dados

Exercício 3 — Novo vértice

3

Adicione o vértice E ao grafo e conecte-o aos vértices C e D.




```
grafo["E"] = []
grafo["E"].append("C")
grafo["E"].append("D")
```


Professor
Edson Pedro FERLIN

Estruturas de Dados

Exercício 4 — Cálculos

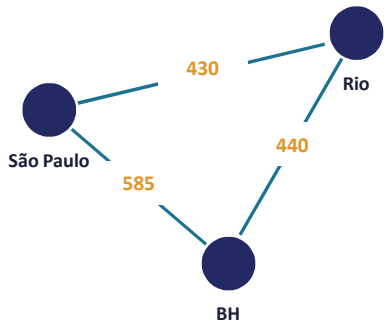
4

Para o grafo construído, calcule:

1. **Grau** dos vértices
2. **Nº de arestas** total do grafo
3. **Nº de vértices** total do grafo

Exercício desafio — Mapa de cidades

Modele um mapa de cidades como um grafo ponderado:



- Cada cidade → um vértice
- Cada estrada → uma aresta
- Distâncias → pesos das arestas

Sistema de Rotas entre Cidades

Desenvolva uma aplicação em Python que represente um mapa rodoviário utilizando grafos.

REQUISITOS

- Criar um grafo ponderado (vértice = cidade, aresta = estrada com distância)
- Permitir inserir novas cidades e conexões
- Exibir a lista de adjacência do grafo
- Implementar os percursos DFS e BFS
- Exibir nº de vértices, arestas e o grau de cada cidade

DESAFIO EXTRA

Com o **NetworkX**, implemente o **menor caminho** entre duas cidades (algoritmo de **Dijkstra**) e compare com um percurso simples.

Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.com.br

(YOUTUBE) ProfEdsonPedroFerlin