

AED2 - Algoritmos e Estr. de Dados II

Aula 14: Introdução a Grafos

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timóteo
Dep. Engenharia de Computação

Fevereiro de 2026

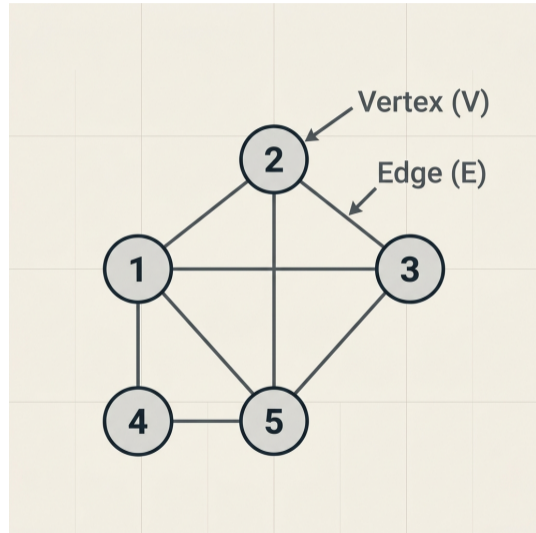
- 1 O Mundo Conectado
- 2 Terminologia
- 3 Representações
- 4 Implementação em Java
- 5 Lendo Grafos da Entrada
- 6 Propriedades Importantes
- 7 Exercícios

Grafos Estão em Todo Lugar

Um **Grafo** $G = (V, E)$ modela **relacionamentos** entre entidades.

- V = conjunto de **Vértices** (nós, entidades)
- E = conjunto de **Arestas** (conexões, relações)

Domínio	Vértice	Aresta
Redes Sociais	Pessoa	Amizade
Mapas	Cidade	Estrada
Internet	Roteador	Cabo
Web	Página	Hyperlink
Compiladores	Variável	Dependência



Conceitos Básicos

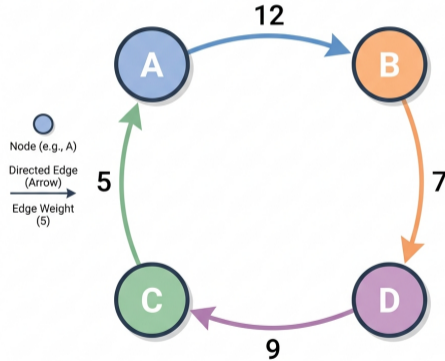
- **Grau** (*degree*): número de arestas de um vértice. Em dígrafos: grau de *entrada* e *saída*.
- **Caminho**: sequência de vértices v_1, v_2, \dots, v_k onde existe aresta entre consecutivos.
- **Caminho Simples**: sem vértices repetidos.
- **Ciclo**: caminho que começa e termina no mesmo vértice.
- **Grafo Conexo**: existe caminho entre qualquer par de vértices.
- **Componente Conexo**: subgrafo maximal conexo.

Tipos de Grafo

- **Não-Direcionado**: arestas bidirecionais ($u - v$). Ex: amizades.
- **Direcionado (Dígrafo)**: arestas unidirecionais ($u \rightarrow v$). Ex: seguir no Twitter.
- **Ponderado**: arestas com peso/custo. Ex: distância em km.
- **DAG**: dígrafo sem ciclos. Ex: dependências de tarefas.
- **Árvore**: grafo conexo e acíclico com $|E| = |V| - 1$.

DIRECTED WEIGHTED GRAPH (DIGRAPH)

A network with directed edges and assigned numeric weights.



Definições Adicionais

- **Grafo simples:** sem laços e sem arestas paralelas
- **Grafo completo K_n :** todo par de vértices está conectado; $|E| = \frac{n(n-1)}{2}$
- **Bipartido:** vértices divisíveis em 2 grupos A e B , arestas só conectam de A para B .

Densidade

- **Grafo esparso:** $|E| \ll |V|^2$
A maioria dos problemas reais.
- **Grafo denso:** $|E| \approx |V|^2$
Raramente encontrados na natureza, comuns em teoria dos jogos.

Grafo Esperso ($|E| \ll |V|^2$)

Exemplo: Mapa rodoviário com $V = 10.000$ cidades.

Cada cidade conecta em média a 4 outras
 $\Rightarrow |E| \approx 20.000$.

Max. possível:	≈ 50 milhões
Real:	≈ 20.000
Matriz:	400 MB
Lista:	120 KB

Grafo Denso ($|E| \approx |V|^2$)

Exemplo: Torneio de xadrez com $V = 16$ jogadores.

Todo par joga entre si $\Rightarrow |E| = 120$.

Max. possível:	120
Real:	120 (100%)
Matriz:	1 KB
Lista:	mesma ordem

Resumo: Esparso vs Denso

	Esparso	Denso
Exemplo real	Mapa, Redes Sociais, Internet	Torneio, Grafos de Conhecimento, K
$ E $ típico	$O(V)$	$O(V^2)$
Representação preferida	Lista de Adjacência	Matriz de Adjacência ou Lista

Conclusão Prática

Na grande maioria dos problemas do mundo real e competições de programação, os grafos são **esparso**s. Por isso, a **Lista de Adjacência** é o padrão da indústria.

Opção 1: Matriz de Adjacência

Matriz bidimensional $M[V \times V]$ onde:

- $M[i][j] = 1$ (ou peso) se aresta existe
- $M[i][j] = 0$ (ou ∞) caso contrário

Vantagem Consulta $M[u][v]$ em $O(1)$

Desvantagem Memória $O(V^2)$

Desvantagem Listar vizinhos custa $O(V)$

Use quando: $V \leq 2000$ ou grafo for extremamente denso.

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	0
2	1	0	0	0	1
3	0	1	0	0	0
4	0	0	1	0	0

Opção 2: Lista de Adjacência (Padrão Ouro)

Vetor de listas: $adj[u]$ contém a lista de todos os vizinhos do vértice u .

Vantagem Memória $O(V + E)$ (ótimo p/

Vantagem Listar vizinhos custa apenas $O($

Desvantagem Consulta se (u, v) existe custa

Use para: 99% dos problemas reais e de juiz online.

Exemplo do mesmo grafo

$adj[0] = [1, 2]$

$adj[1] = [0, 3]$

$adj[2] = [0, 4]$

$adj[3] = [1]$

$adj[4] = [2]$

Matriz de Adjacência — Com Pesos (Grafo Ponderado)

Em grafos **ponderados**, $M[i][j]$ armazena o **peso** da aresta, não apenas 0/1.

Exemplo: 4 cidades, distâncias em km:

- SP → RJ: 430 km
- SP → BH: 586 km
- RJ → BH: 434 km
- RJ → VIT: 520 km

	SP	RJ	BH	VIT
SP	0	430	586	∞
RJ	430	0	434	520
BH	586	434	0	∞
VIT	∞	520	∞	0

∞ vs 0

Nunca use 0 para “sem aresta” em grafos ponderados — 0 pode ser um peso válido!
Use ∞ (ou `Integer.MAX_VALUE`) para indicar ausência de aresta.

Lista equivalente

```
adj[SP] = [(RJ,430), (BH,586)]  
adj[RJ] = [(SP,430), (BH,434), (VIT,520)]  
adj[BH] = [(SP,586), (RJ,434)]  
adj[VIT] = [(RJ,520)]
```

Navegabilidade: O Que Posso Perguntar ao Grafo?

Navegabilidade: quais perguntas são respondíveis e em qual custo?

Pergunta	Matriz	Lista
Existe aresta (u, v) ?	$O(1)$	$O(\text{grau})$
Quais vizinhos de u ?	$O(V)$	$O(\text{grau})$
Quantas arestas?	$O(V^2)$	$O(V + E)$
Grau de u ?	$O(V)$	$O(\text{grau})$
Caminho de u a v ?	BFS/DFS $O(V + E)$	
Menor caminho?	Dijkstra $O((V + E) \log V)$	

Regra prática

Se o algoritmo precisa **iterar sobre vizinhos** (BFS, DFS, Dijkstra), prefira Lista. Se precisa **checar existência** de aresta específica frequentemente, Matriz pode valer.

O que NÃO dá pra perguntar diretamente

- Existe caminho entre u e v ?
→ precisa de BFS/DFS
- Qual o caminho mais curto?
→ precisa de BFS (sem peso) ou Dijkstra (com peso)
- Quais componentes conexos?
→ precisa de varredura
- O grafo tem ciclo?
→ precisa de DFS

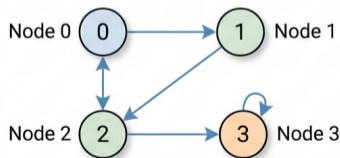
Resumo

A **representação** não resolve problemas — ela apenas organiza os dados. Os **algoritmos** (BFS, DFS, Dijkstra) é que respondem as perguntas.

Comparação das Representações

Operação	Matriz Adj.	Lista Adj.	Lista de Arestas
Memória	$O(V^2)$	$O(V + E)$	$O(E)$
Verificar aresta (u, v)	$O(1)$	$O(\text{grau})$	$O(E)$
Listar vizinhos de u	$O(V)$	$O(\text{grau})$	$O(E)$
Iterar todas as arestas	$O(V^2)$	$O(V + E)$	$O(E)$
Adicionar aresta	$O(1)$	$O(1)$	$O(1)$
Ideal para	Grafos densos	Grafos esparsos	Kruskal / Bellman-Ford

Example Graph



Adjacency Matrix

Adjacency List

Lista de Adjacência em Java — Não Ponderado

```
import java.util.*;

public class Grafo {
    private int V;
    private List<List<Integer>> adj;

    public Grafo(int V) {
        this.V = V;
        adj = new ArrayList<>();
        for (int i = 0; i < V; i++)
            adj.add(new ArrayList<>());
    }

    // Aresta nao-direcionada: adiciona nos dois sentidos
    public void addAresta(int u, int v) {
        adj.get(u).add(v);
        adj.get(v).add(u); // remova esta linha para grafo direcionado
    }

    public List<Integer> vizinhos(int u) { return adj.get(u); }

    public static void main(String[] args) {
```

```
class Aresta {
    int destino, peso;
    Aresta(int d, int p) { destino = d; peso = p; }
}

public class GrafoPonderado {
    private List<List<Aresta>> adj;

    public GrafoPonderado(int V) {
        adj = new ArrayList<>();
        for (int i = 0; i < V; i++) adj.add(new ArrayList<>());
    }

    public void addAresta(int u, int v, int peso) {
        adj.get(u).add(new Aresta(v, peso));
        adj.get(v).add(new Aresta(u, peso)); // bidirecional
    }
}
```

Quando usar grafo ponderado?

Algoritmos que usam pesos

Lista de Arestas (Edge List)

Às vezes precisamos apenas de uma **lista plana de todas as arestas** — sem saber quem são os vizinhos de cada vértice.

```
class Aresta implements Comparable<Aresta> {
    int u, v, peso;
    Aresta(int u, int v, int p) {
        this.u=u; this.v=v; this.peso=p;
    }
    public int compareTo(Aresta o) {
        return this.peso - o.peso; // ordena
            por peso
    }
}

List<Aresta> arestas = new ArrayList<>();
arestas.add(new Aresta(0, 1, 4));
arestas.add(new Aresta(0, 2, 2));
Collections.sort(arestas); // ordena por peso
```

Algoritmos que usam Edge List

- **Kruskal** (MST): ordena arestas por peso
- **Bellman-Ford**: itera $V - 1$ vezes sobre todas as arestas

Memória

Apenas $O(E)$. Mais compacto que lista de adjacência para operações globais sobre arestas.

Formato típico de entrada:

```
5 6          <- V vertices, E
    arestas
0 1          <- cada aresta u v
0 2
1 3
1 4
2 3
3 4
```

Template Java completo:

```
Scanner sc = new Scanner(System.in);
int V = sc.nextInt(), E = sc.nextInt();

List<List<Integer>> adj = new ArrayList
    <>();
for (int i = 0; i < V; i++)
    adj.add(new ArrayList<>());
```

```
1 int u = sc.nextInt() - 1;
2 int v = sc.nextInt() - 1;
```

Indexação 1-based

Muitos problemas usam vértices de 1 a N.
Lembre-se de subtrair 1:

Grafo ponderado

```
1 int u = sc.nextInt();
2 int v = sc.nextInt();
3 int w = sc.nextInt();
4 adj.get(u).add(new Aresta(v,w));
```

Dica de desempenho

Handshaking Lemma

Em qualquer grafo não-direcionado:

$$\sum_{v \in V} \text{grau}(v) = 2|E|$$

Todo grau conta cada aresta duas vezes (uma por extremo). Consequência: o número de vértices com grau ímpar é sempre **par**.

Árvore

Um grafo conexo com $|V|$ vértices é uma **árvore** se e somente se $|E| = |V| - 1$ (e portanto não tem ciclos).

Grafo Bipartido

Um grafo é bipartido \Leftrightarrow não possui ciclo de comprimento **ímpar**.

Verificação: tente colorir com 2 cores via BFS/DFS. Se encontrar conflito, não é bipartido.

Grafos esparsos vs densos

- **Esparso:** $|E| = O(V)$
 \Rightarrow use Lista de Adjacência
- **Denso:** $|E| = O(V^2)$
 \Rightarrow Matriz pode ser razoável

Na prática: quase sempre esparso.

1. **[Fácil]** Dado o grafo com arestas: $\{(0, 1), (0, 2), (1, 3), (2, 3), (3, 4)\}$, desenhe: (a) a lista de adjacência; (b) a matriz de adjacência. Quantos componentes conexos existem?
2. **[Fácil]** Prove que para qualquer grafo simples não-direcionado com V vértices, o grau máximo de qualquer vértice é $V - 1$. Qual a relação disso com o Handshaking Lemma?
3. **[Médio]** Implemente a classe Grafo em Java suportando: (a) adição de aresta ponderada; (b) listagem de vizinhos; (c) impressão da matriz de adjacência. Teste com $V = 4$ e arestas $\{(0, 1, 3), (0, 2, 1), (1, 3, 4), (2, 3, 2)\}$.
4. **[Difícil]** Um grafo tem $V = 5$ vértices. Qual é o número máximo de arestas em um grafo: (a) simples não-direcionado? (b) simples direcionado? (c) bipartido com partições de tamanho 2 e 3?

O que aprendemos

- Grafos $G = (V, E)$ modelam relacionamentos universais
- Tipos: direcionado, não-dir., ponderado, DAG, bipartido
- **Matriz de Adj.:** $O(1)$ por consulta, $O(V^2)$ memória
- **Lista de Adj.:** $O(V + E)$ memória — padrão ouro
- **Edge List:** compacto para algoritmos globais (Kruskal)
- Handshaking Lemma: $\sum \text{grau} = 2|E|$

Próximas aulas

- Aula 15: BFS — menor caminho
- Aula 16: DFS — ciclos e topologia
- Aula 17: Dijkstra — pesos positivos

Regra de ouro

“Grafos esparsos? Lista de Adj. Grafos densos ou $V \leq 2000$? Matriz. Kruskal/Bellman? Edge List.”