**Respostas — Exercícios sobre Grafos**

**Exercício 1 — representação (grafo não direcionado)**

Suponha uma turma com **5 alunos**: **A, B, C, D, E**. Uma possível rede de amizade (arestas não direcionadas):

**Arestas**:

* (A, B), (A, C), (B, C), (B, D), (C, E)

**Representações**:

* **Lista de adjacência**

**A**: **B, C**

**B**: **A, C, D**

**C**: **A, B, E**

**D**: **B**

**E**: **C**

* **Matriz de adjacência** (linhas/colunas na ordem **A,B,C,D,E** — 1 = existe aresta)

Uma imagem contendo objeto, relógio, traçado

O conteúdo gerado por IA pode estar incorreto. **A B C D E**

**A** [ 0 1 1 0 0 ]

**B** [ 1 0 1 1 0 ]

**C** [ 1 1 0 0 1 ]

**D** [ 0 1 0 0 0 ]

**E** [ 0 0 1 0 0 ]

* **Descrição curta do grafo:**

**A está ligada** a **B e C**; **B e C** formam um **triângulo com A; B** também é amigo de D; **C** é amigo de E. É um **grafo simples, não-direcionado**.

**Exercício 2 — matriz de adjacência do grafo dado**

**Vértices** = {A, B, C, D}  
**Arestas** = {(A, B), (A, C), (B, D), (C, D)}

**Ordem de vértices**: A, B, C, D.

**Matriz de adjacência** (**linha** = origem, **coluna** = destino). Como é não-direcionado, a matriz é simétrica.

**A B C D**

**A** [ 0 1 1 0 ]

**B** [ 1 0 0 1 ]

**C** [ 1 0 0 1 ]

**D** [ 0 1 1 0 ]

**Explicação célere**:

* (**A,B**) → posições (A,B) e (B,A) = 1
* (**A,C**) → (A,C) e (C,A) = 1
* (**B,D**) → (B,D) e (D,B) = 1
* (**C,D**) → (C,D) e (D,C) = 1  
  **diagonais** = 0 (**sem laços**).

**Exercício 3 — é grafo completo?**

**Dados**: V = {1,2,3,4} e E = {(1,2), (2,3), (3,4), (4,1), (1,3)}

Para um **grafo completo** com 4 vértices (K4​ ) o número de arestas deve ser n(n − 1) / 2 = 4 \* 3 / 2 ​= 6 **arestas**, e cada par distinto de vértices deve estar ligado.

Aqui temos **5 arestas**, logo **não é completo**.

Faltante: o **par (2,4)** **não** aparece em E. Portanto não existe aresta entre 2 e 4 — isso impede que seja K4.

**Conclusão**: **Não é grafo completo**. (**Justificativa**: número de arestas insuficiente e **ausência da aresta (2,4)**.)

**Exercício 4 (desafio) — implementação com lista de adjacência**

**Implementação em Python**

**# Representação simples de grafo não-direcionado usando dicionário de listas**

class Graph:

def \_\_init\_\_(self):

self.adj = {} # {vertex: [neighbors]}

def add\_vertex(self, v):

if v not in self.adj:

self.adj[v] = []

def add\_edge(self, u, v):

# adiciona vértices se não existirem

if u not in self.adj:

self.add\_vertex(u)

if v not in self.adj:

self.add\_vertex(v)

# grafo não-direcionado: adicionar em ambas listas (evitar duplicata)

if v not in self.adj[u]:

self.adj[u].append(v)

if u not in self.adj[v]:

self.adj[v].append(u)

def neighbors(self, v):

return list(self.adj.get(v, []))

def \_\_str\_\_(self):

s = []

for v in sorted(self.adj):

s.append(f"{v}: {', '.join(map(str, self.adj[v]))}")

return "\n".join(s)

**# Exemplo de uso**

if \_\_name\_\_ == "\_\_main\_\_":

g = Graph()

g.add\_edge('A','B')

g.add\_edge('A','C')

g.add\_edge('B','D')

print(g)

**Complexidade:**

* Inserir vértice: **O(1)** amortizado.
* Inserir aresta: **O(1)** para dicionário + possível custo para verificar/evitar duplicatas (no código acima a verificação if v not in self.adj[u] é O(deg(u))). Para grandes grafos convém permitir duplicatas ou usar set em vez de list.

**Implementação em Java**

import java.util.\*;

public class Graph {

private final Map<String, List<String>> adj = new HashMap<>();

public void addVertex(String v) {

adj.putIfAbsent(v, new ArrayList<>());

}

public void addEdge(String u, String v) {

addVertex(u);

addVertex(v);

List<String> nu = adj.get(u);

List<String> nv = adj.get(v);

if (!nu.contains(v)) nu.add(v);

if (!nv.contains(u)) nv.add(u);

}

public List<String> neighbors(String v) {

return adj.getOrDefault(v, Collections.emptyList());

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

List<String> keys = new ArrayList<>(adj.keySet());

Collections.sort(keys);

for (String k : keys) {

sb.append(k).append(": ").append(String.join(", ", adj.get(k))).append("\n");

}

return sb.toString();

}

**// Exemplo**

public static void main(String[] args) {

Graph g = new Graph();

g.addEdge("A", "B");

g.addEdge("A", "C");

g.addEdge("B", "D");

System.out.println(g);

}

}