Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

# Grafos (parte 2)


https://devsuperior.com.br



## Representações de grafos em memória

https://devsuperior.com.br

Prof. Dr. Nelio Alves

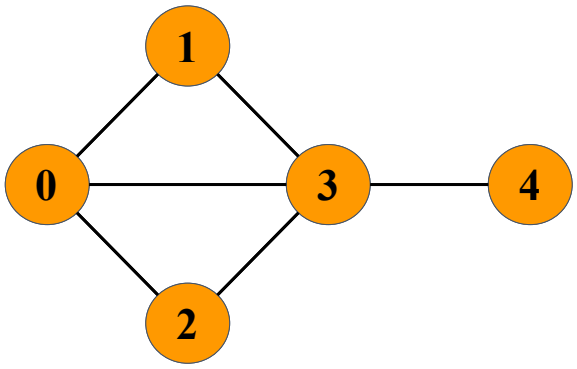
## Representações de grafos

Como escolher uma estrutura de dados para representar um grafo?

A escolha da estrutura terá impacto no desempenho dos algoritmos!

Há algumas opções:

- Lista de arestas
- Matriz de adjacência
- Lista de adjacência

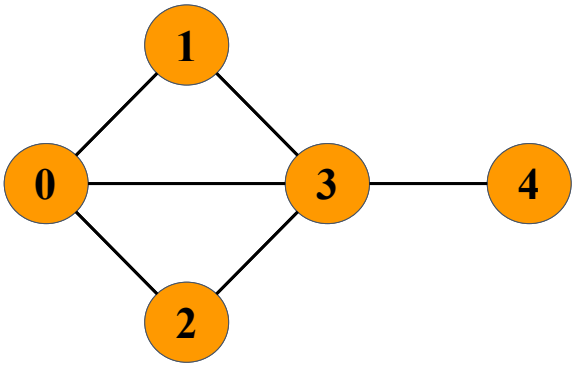





## Lista de arestas

Lista de tamanho N = |A|, em que cada posição possui uma aresta (i, j).

{(0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 4)}



## Lista de arestas - Características

- Representação super simples
- Geralmente é a forma na qual a entrada é dada
- Propriedades
  - Custo de representação em memória: O(|A|) = O(n)
  - Custo para consultar aresta (i, j) = O(n)
  - Custo para verificar grau de vértice = O(n)

- Útil quando precisamos percorrer somente as arestas, mas deixar a desejar nas consultas.

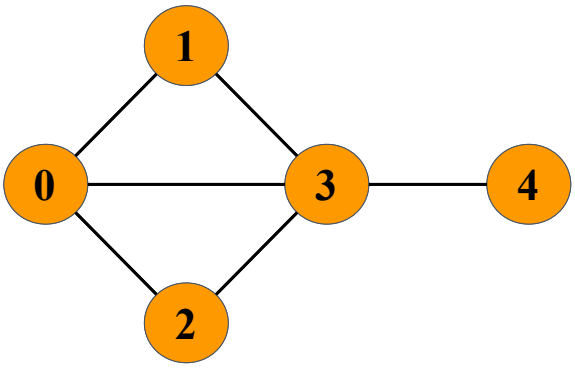


## Matriz de adjacência

Criar matriz NxN, onde N = |V|, seguindo a seguinte regra:

$A_{ij} = \begin{cases} 1 \text{ se houver aresta } (i, j) \\ 0 \text{ caso contrário} \end{cases}$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 | 0 |

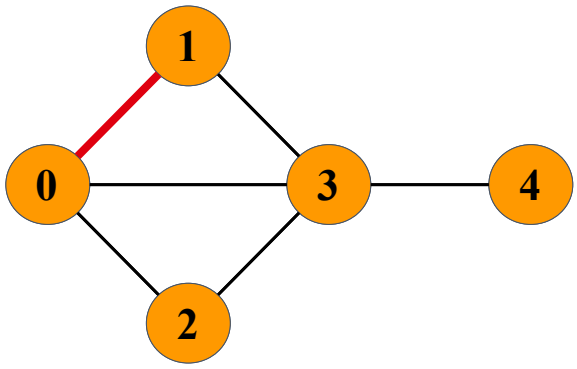


## Matriz de adjacência

Para grafos não-direcionados, a matriz será simétrica.

$A_{ij} = \begin{cases} 1 \text{ se houver aresta } (i, j) \\ 0 \text{ caso contrário} \end{cases}$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 | 0 |



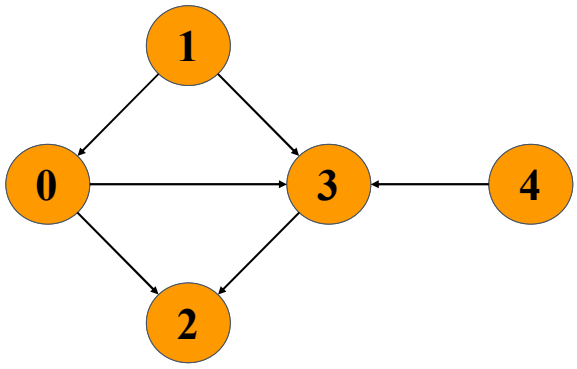



## Matriz de adjacência

Para grafos direcionados, uma aresta (i, j) indica que ela é divergente de i e convergente a j, isto é i → j.

$A_{ij} = \begin{cases} 1 \text{ se houver aresta } (i, j) \\ 0 \text{ caso contrário} \end{cases}$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 |



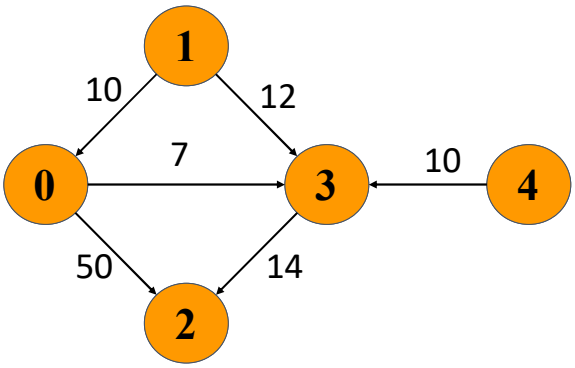

Matriz assimétrica

## Matriz de adjacência

Para grafos direcionados ponderados, uma aresta (i, j, k) indica que ela é divergente de i e convergente a j, isto é i → j, com peso k.

$A_{ij} = \begin{cases} k \text{ se houver aresta } (i, j) \\ -1 \text{ caso contrário} \end{cases}$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | -1 | -1 | 50 | 7 | -1 |
| 1 | 10 | -1 | -1 | 12 | -1 |
| 2 | -1 | -1 | -1 | -1 | -1 |
| 3 | -1 | -1 | 14 | -1 | -1 |
| 4 | -1 | -1 | -1 | 10 | -1 |






## Matriz de adjacência - Características

- Representação mais simples
- Propriedades
  - Custo de representação em memória: $O(|V|^2) = O(n^2)$
  - Custo para consultar aresta (i, j) = O(1)

- É uma **boa** estrutura quando precisamos:
  - representar **grafos densos**
  - **consultar arestas** rapidamente (tempo constante)
  - **inserir/remover arestas** rapidamente (tempo constante)

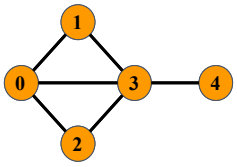
- É **ruim** quando precisamos:
  - representar **grafos esparsos**
  - **percorrer toda a matriz** $O(|V|^2) = O(n^2)$

## Matriz de adjacência - Implementação

**1)** Implementar uma classe Graph, utilizando uma matriz de adjacência para representar um grafo não direcionado e não ponderado. A classe deve conter as seguintes operações:

- constructor(numVertices)
  - inicializa a estrutura de representação com tamanho numVertices
- addEdge(v1, v2)
  - adiciona aresta entre os vértices v1 e v2
- removeEdge(v1, v2)
  - remove aresta entre os vértices v1 e v2
- printGraph()
  - mostra no console o estado atual da matriz
- degree(v)
  - retorna o grau do vértice v
- listByDegree()
  - mostra no console os vértices agrupados em função de seus graus

Exemplo




```
grau=0:{}
grau=1:{4}
grau=2:{1, 2}
grau=3:{0}
grau=4:{3}
```

11

## Matriz de adjacência - Implementação

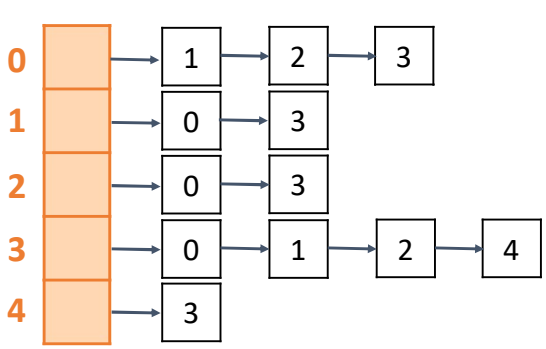
**2)** Implementar uma classe Graph, utilizando uma matriz de adjacência para representar um grafo direcionado e ponderado. A classe deve conter as seguintes operações:

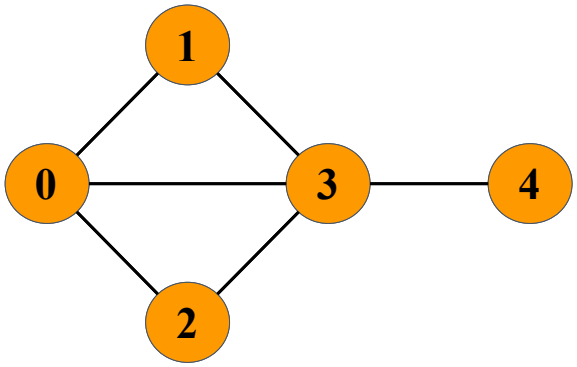
- constructor(numVertices)
  - inicializa a estrutura de representação com tamanho numVertices
- addEdge(v1, v2, w)
  - adiciona aresta entre os vértices v1 e v2 com peso w
- removeEdge(v1, v2)
  - remove aresta entre os vértices v1 e v2
- printGraph()
  - mostra no console o estado atual da matriz
- lowestWeight()
  - retorna a aresta de menor peso
- neighbours(v)
  - retorna os vizinhos do vértice v

12

## Listas de adjacências

Conjunto de |V| = n listas, onde cada lista L(v) contém os vértices adjacentes ao vértice v.



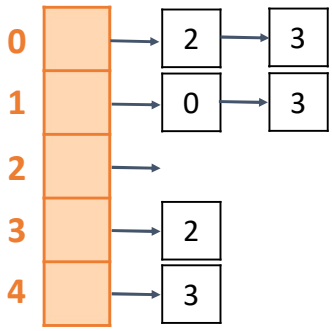



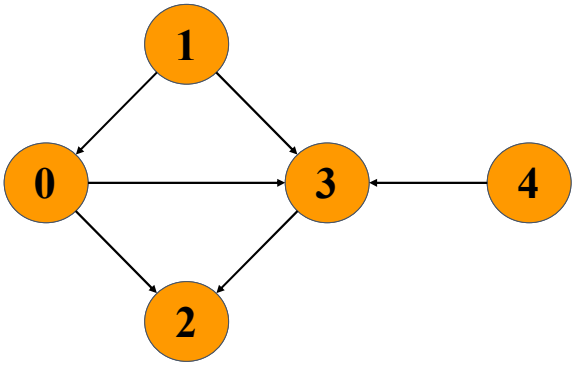

Em grafos não direcionados adicionamos duas vezes uma mesma aresta.



## Listas de adjacências

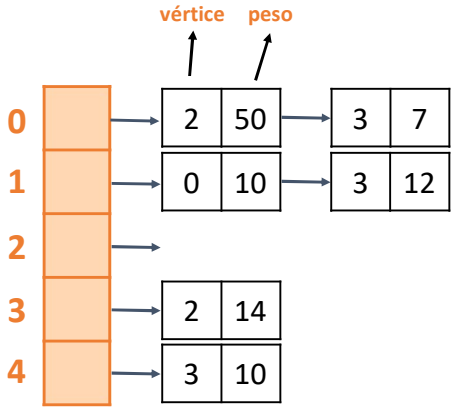
Para grafos direcionados, precisamos adicionar apenas uma vez cada aresta.

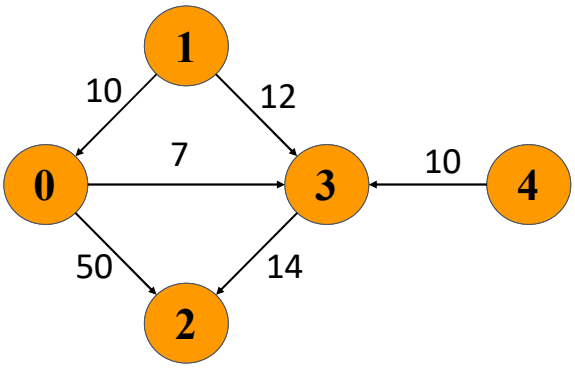


## Listas de adjacências

Para grafos direcionados ponderados, cada nó da lista é representado pelo **vértice adjacente** e pelo **peso** da aresta.






## Listas de adjacências - Características

- Representação compacta, utilizada na maioria das vezes
- Propriedades
  - Custo de representação em memória: O(|V| + |A|) = O(n + m)
  - Custo para consultar aresta (i, j) = O($d_i$), sendo $d_i$ o grau de i

- É uma **boa** estrutura quando:
  - representamos **grafos esparsos**
    - |A| muito menor que |V²|
  - **percorrer todo o grafo** O(|V| + |A|)
  - determinar **grau** de um vértice rapidamente

- É **ruim** quando precisamos:
  - verificar se um vértice i é **adjacente** a outro vértice j, O(n)

## Qual a melhor estrutura?

| Tarefa | Melhor estrutura |
|---|---|
| Determinar se aresta (x, y) existe | Matriz de adjacências |
| Determinar o grau de um vértice | Listas de adjacências |
| Representar grafos esparsos | Listas de adjacências |
| Representar grafos densos | Matriz de adjacências |
| Inserção/remoção de arestas | Matriz: O(1)<br>Listas: O(di) |
| Percorrer todo o grafo | Listas: O(\|V\| + \|A\|)<br>Matriz: O(\|V\|²) |
| Melhor na maioria dos problemas | Listas de adjacências |

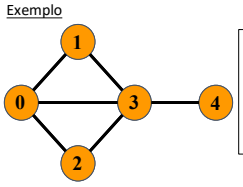


## Listas de adjacências - Implementação

**1)** Implementar uma classe Graph, utilizando uma lista de adjacências para representar um grafo não direcionado e não ponderado. A classe deve conter as seguintes operações:

- constructor(numVertices)
  - inicializa a estrutura de representação com tamanho numVertices
- addEdge(v1, v2)
  - adiciona aresta entre os vértices v1 e v2
- removeEdge(v1, v2)
  - remove aresta entre os vértices v1 e v2
- printGraph()
  - mostra no console o estado atual da matriz
- degree(v)
  - retorna o grau do vértice v
- listByDegree()
  - mostra no console os vértices agrupados em função de seus graus

Exemplo




```
grau=0:{}
grau=1:{4}
grau=2:{1, 2}
grau=3:{0}
grau=4:{3}
```

## Listas de adjacências - Implementação

**2)** Implementar uma classe Graph, utilizando uma lista de adjacências para representar um grafo direcionado e ponderado. A classe deve conter as seguintes operações:

- constructor(numVertices)
  - inicializa a estrutura de representação com tamanho numVertices
- addEdge(v1, v2, w)
  - adiciona aresta entre os vértices v1 e v2 com peso w
- removeEdge(v1, v2)
  - remove aresta entre os vértices v1 e v2
- printGraph()
  - mostra no console o estado atual da matriz
- lowestWeight()
  - retorna a aresta de menor peso
- neighbours(v)
  - retorna os vizinhos do vértice v



## Exercícios de Fixação

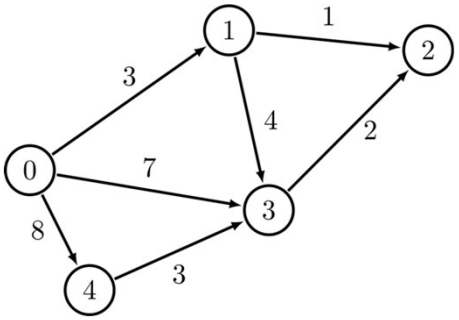
Dada a seguinte lista de arestas:

$$A = \{ (0, 1), (0, 2), (0, 4), (1, 2), (1, 3), (2, 3), (2, 4)\}$$

- Desenhe o grafo não-direcionado correspondente
- Desenhe a representação em matriz de adjacência
- Desenhe a representação em lista de adjacência

## Exercícios de Fixação

Dado o seguinte grafo direcionado ponderado




- Represente-o com uma matriz de adjacência
- Represente-o com uma lista de adjacências