Teoria dos GRAFOS

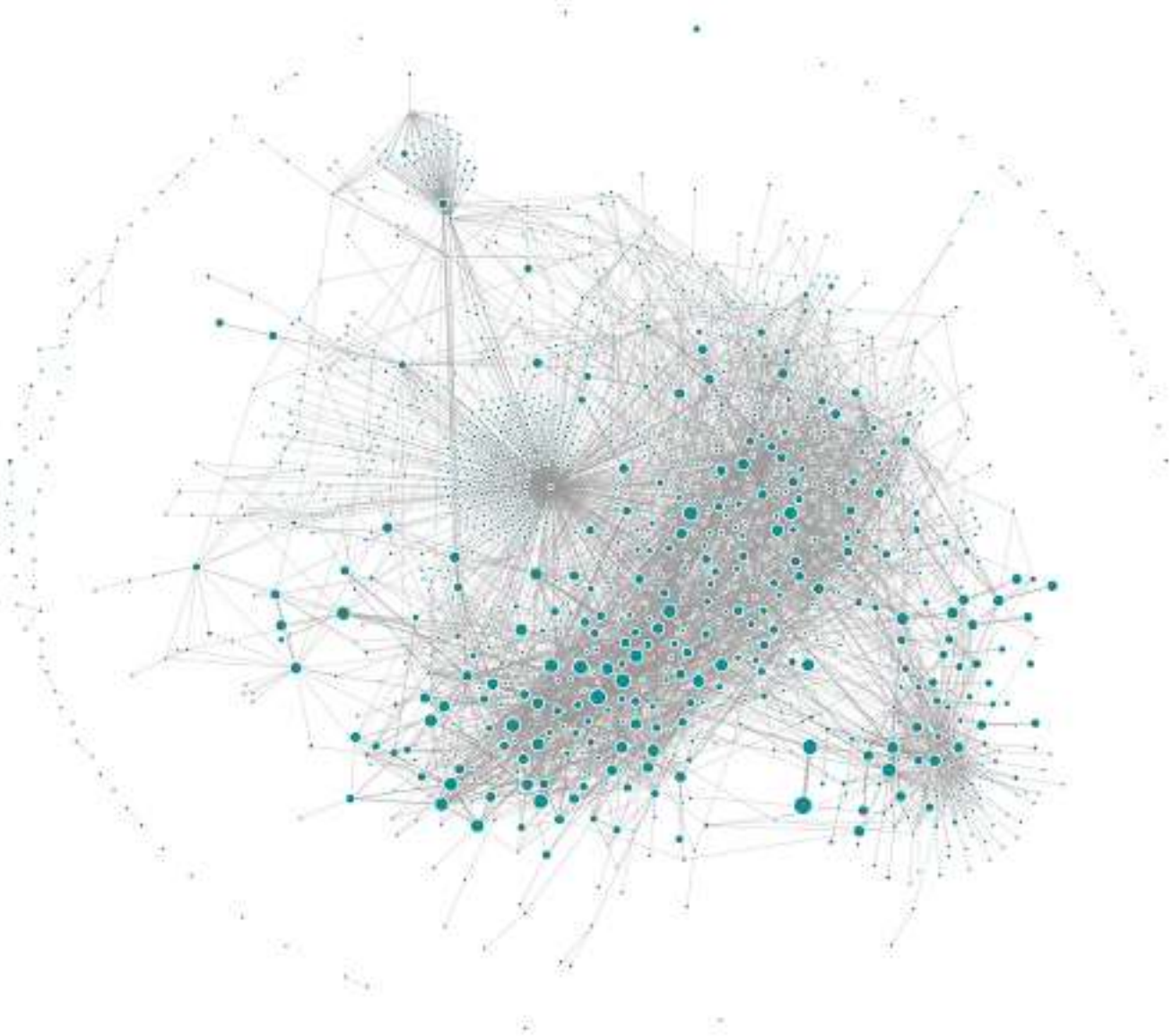
Aula 02: Representação de Grafos

Prof. José Alberto S. Torres

Representação de Grafos

- Utilizando o conceito matemático de vértices e arestas, a representação **visual dos grafos através de pontos ou círculos conectados por retas** é muito simples e funcional.
- Por meio dessa representação, consegue-se **modelar e visualizar diversas situações e problemas**, e com isso **identificar caminhos ou agrupamentos** conforme os dados que se encontrem armazenados no grafo.
- Contudo, quando o problema contém **dezenas de milhares de vértices conectados entre si**, o acompanhamento visual se torna muito difícil.

IESB
Centro Universitário



Representação de Grafos

- Isto levou ao surgimento de duas possibilidades alternativas de representação, uma por meio de **listas** e outra utilizando **matrizes de adjacência**.
- A escolha de qual alternativa adotar depende da finalidade com que se deseja trabalhar com o grafo e quais algoritmos empregar.
- O uso de **listas de adjacências** é mais recomendada nos casos em que temos **grafos mais esparsos**, apresentando uma solução mais compacta.
- Já a representação de **matrizes** é aconselhada quando temos **um grafo mais denso** em que temos o interesse em analisar a conectividade entre os vértices

IESB
Centro Universitário

Matriz de adjacências

- É uma **matriz quadrada** de **tamanho** correspondente ao **número de vértices** do grafo.
- Então, sendo N o número de vértices, a matriz de adjacência terá o tamanho N x N.

Definição da Matriz

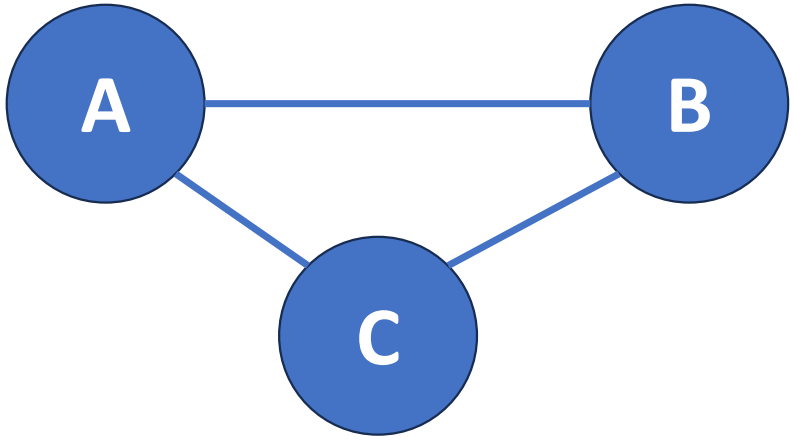
- Podemos representar uma matriz por uma letra maiúscula e seus elementos são representados por uma letra minúscula.
- Assim sendo, em uma matriz A, seus elementos podem ser indicados por $a_{i,j}$, em que o i corresponde ao índice da linha e o j ao índice da coluna.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & \dots & a_{3,n} \end{pmatrix}$$

Matriz de adjacência em **Grafo não orientado**

- Quando vamos implementar uma matriz de adjacência em um grafo não orientado, devemos **inicializá-la com todos os elementos sendo preenchidos por zero (0**), e **quando ocorrer uma conexão** entre o vértice da linha com um da coluna **somamos 1 ao valor original.**
- Quando os valores da matriz de adjacência são somente zeros ou uns, trata-se de uma **matriz booleana,** que define apenas a existência de ligação entre os vértices.

Matriz de adjacência em **Grafo não orientado**



ORIGEM

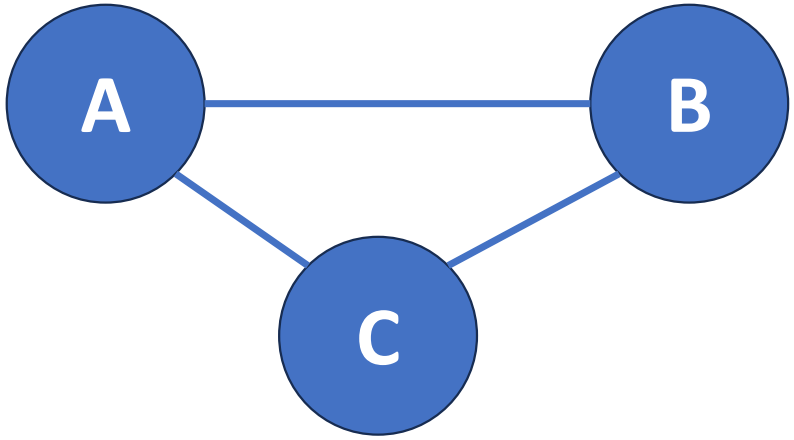
DESTINO

	A	B	C
A	0	0	0
B	0	0	0
C	0	0	0

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

$|V| = 3$

Matriz de adjacência em **Grafo não orientado**



ORIGEM

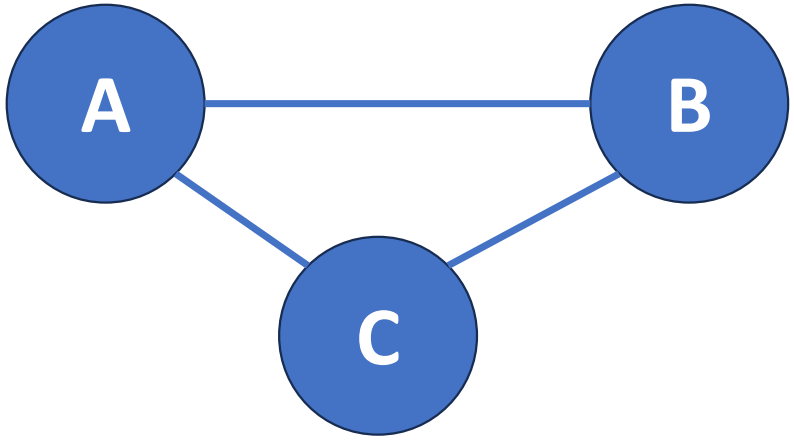
DESTINO

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

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

$|V| = 3$

Matriz de adjacência em **Grafo não orientado**



ORIGEM

DESTINO

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

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

$|V| = 3$

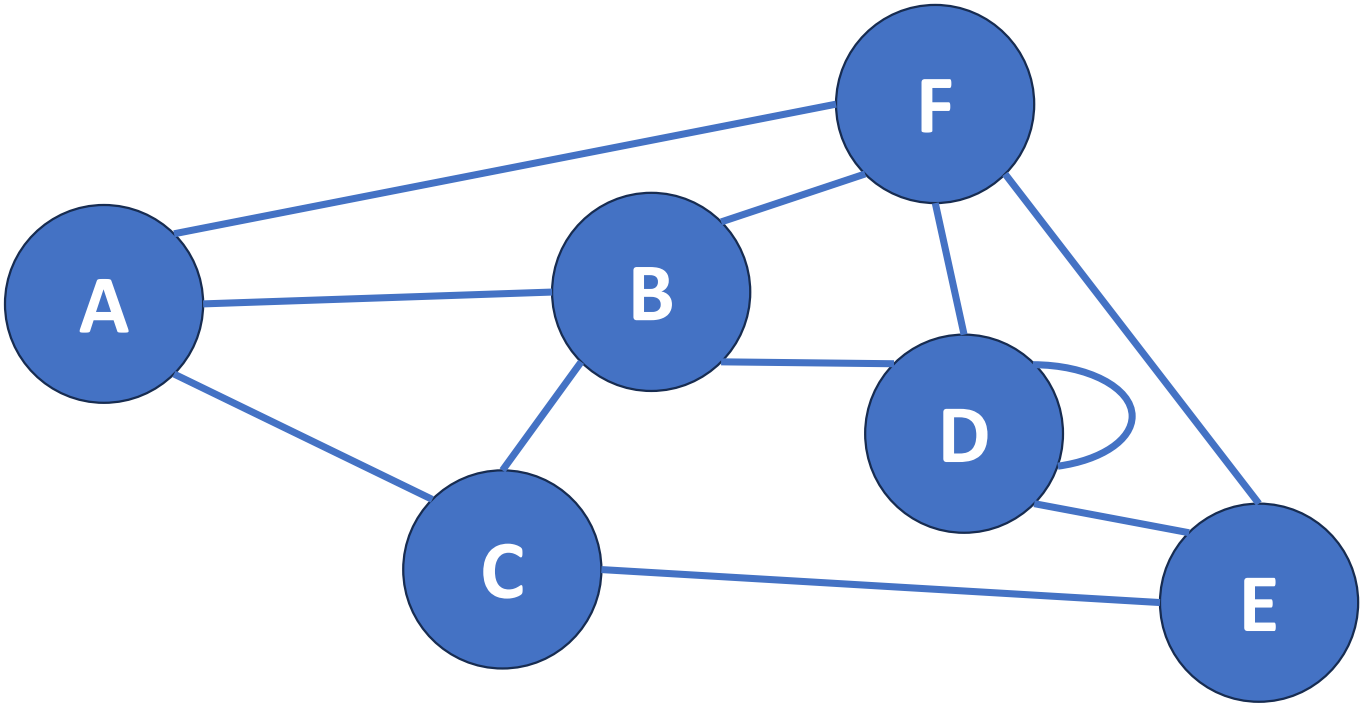
Como nosso grafo não têm laços, os elementos da diagonal da matriz de adjacências são iguais a 0

Matriz de adjacência em **Grafo não orientado**

- A matriz traz, ainda, uma característica importante da matriz de adjacência de grafos não orientados: a **simetria existente na diagonal**.
- Se acessarmos um elemento com origem no vértice O e destino no vértice D: $X_{O,D}$, e fazermos o mesmo com o inverso: $X_{D,O}$ obteremos o mesmo resultado.
- Assim, $X_{O,D} = X_{D,O}$.

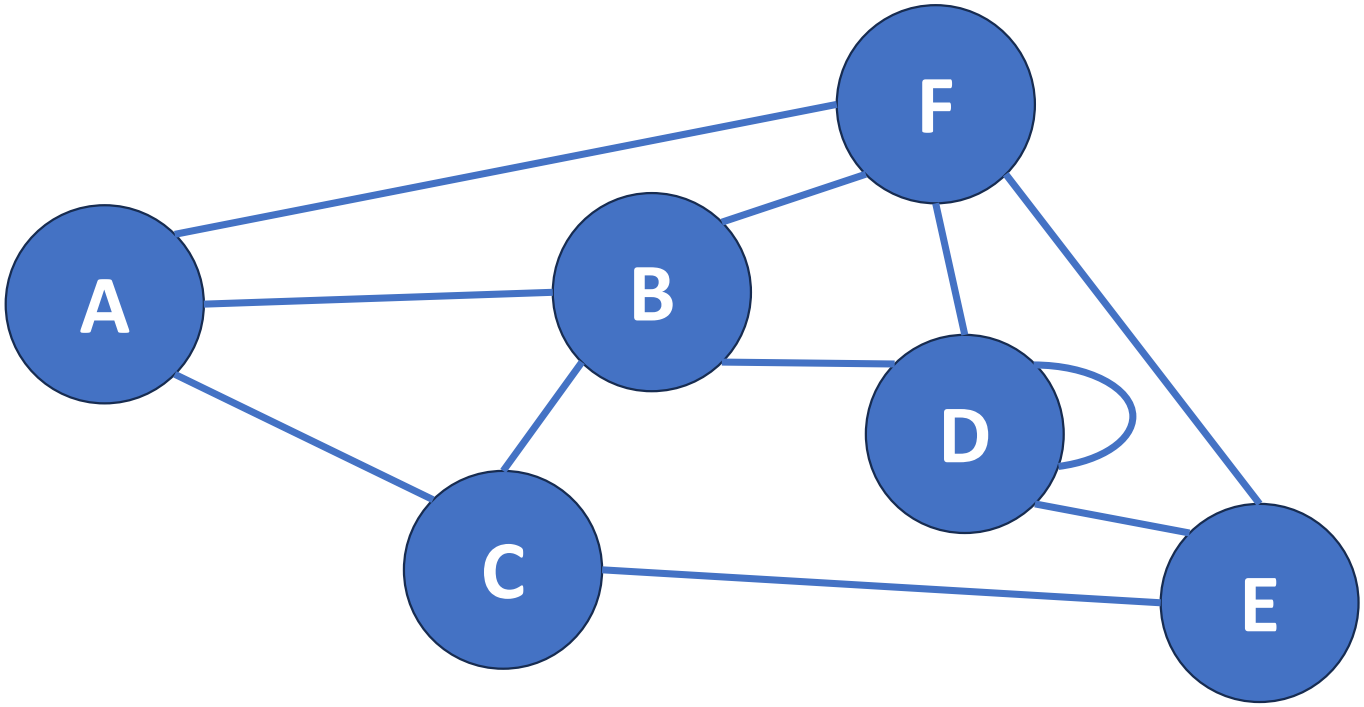
Atividade 1

- Dado o grafo não orientado **G** abaixo, com 6 vértices e 11 arestas, construa a sua matriz de adjacência correspondente.



Atividade 1

- Dado o grafo não orientado **G** abaixo, com 6 vértices e 11 arestas, construa a sua matriz de adjacência correspondente.



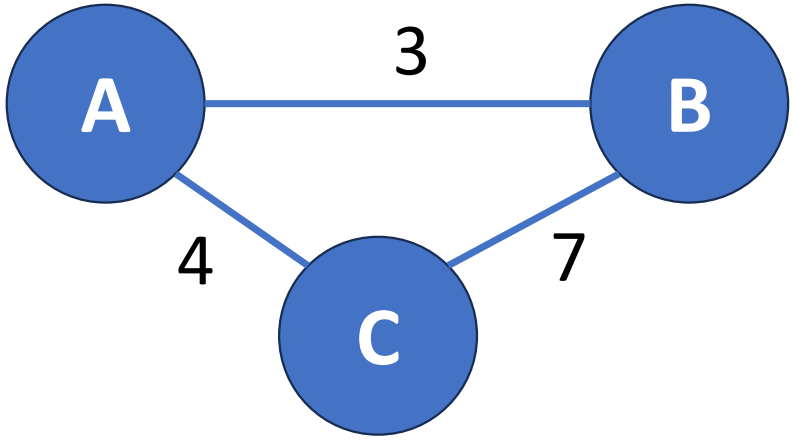
G	A	B	C	D	E	F
A	0	1	1	0	0	1
B	1	0	1	1	0	1
C	1	1	0	0	1	0
D	0	1	0	**1**	1	1
E	0	0	1	1	0	1
F	1	1	0	1	1	0

Matriz de adjacência em **Grafo ponderado**

- É possível ter grafos que possuem peso em suas arestas; nesses casos, podemos representá-los por meio das matrizes de adjacência, só que em vez de utilizar o valor 1 no elemento que possui os extremos de uma aresta, utilizamos o valor do peso.
- Ao implementar a matriz de adjacência de grafos ponderados devemos utilizar um valor de elemento distinto de zero quando se inicializar a matriz, isto porque **o valor zero pode ser um peso empregado no grafo**.
- Dessa forma, vamos deixar a matriz inicializada com todos os elementos em branco.

IESB
Centro Universitário

Matriz de adjacência em **Grafo não orientado**



ORIGEM

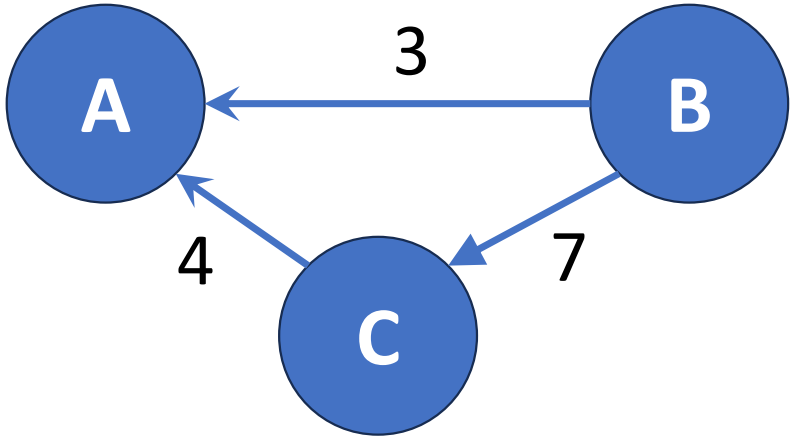
DESTINO

	A	B	C
A			
B			
C			

	A	B	C
A		3	4
B	3		7
C	4	7	

$|V| = 3$

Matriz de adjacência em **Dígrafos**



ORIGEM

DESTINO

	A	B	C
A			
B			
C			

	A	B	C
A		3	4
B			
C		7	

$|V| = 3$

Atividade 2 – Parte 1

- Vamos implementar a estrutura de armazenamento de um **grafo não direcionado não ponderado**.
- Inicialmente, precisamos criar uma interface chamada Grafo, com os seguintes métodos abstratos e seus parâmetros:
  - numero_de_vertices() – retorna o número de vértices
  - numero_de_arestas() – retorna o número de arestas
  - sequencia_de_graus() – retorna a sequência de graus
  - adicionar_aresta(u, v) – adiciona uma nova aresta
  - remover_aresta(u, v) – remove uma aresta
  - imprimir() – imprime a matriz/ lista ligada do grafo
- Onde u e v são vértices do grafo - $\{u, v\} \in V$.
- O peso é aplicado apenas quando trata-se de grafos ponderados.

Atividade 2

```python
from abc import ABC, abstractmethod

class Grafo(ABC):
    @abstractmethod
    def numero_de_vertices(self):
        pass

    @abstractmethod
    def numero_de_arestas(self):
        pass

    @abstractmethod
    def sequencia_de_graus(self):
        pass

    @abstractmethod
    def adicionar_aresta(self, u, v):
        pass

    @abstractmethod
    def remover_aresta(self, u, v):
        pass

    @abstractmethod
    def imprimir(self):
        pass
```

IESB
Centro Universitário

Atividade 2 – Parte 2

- Vamos implementar a classe GrafoDenso, que implementa a interface Grafo utilizando uma representação por matriz.
- Para isso, será preciso instanciar cada um dos métodos abstratos definidos na interface Grafo na classe **GrafoDenso**.
- Inicie criando o método para instanciar a classe (constructor).
- O grafo pode ser criado pelo número de vértices, com os rótulos sendo atribuídos em ordem numérica de 0 até a quantidade de vértices-1, ou por rótulos, onde a quantidade de vértices é a quantidade de rótulos.
- Com base nestas informações, deve-se inicializar a matriz base com valores iniciais igual a 0.

IESB
Centro Universitário

Atividade 2

```python
class GrafoDenso(Grafo):
    # Definição do grafo
    def __init__(self, num_vertices=None, labels=None):
        if labels:
            self.labels = labels
            self.num_vertices = len(labels)
            self.mapa_labels = {label: i for i, label in enumerate(labels)}
        elif num_vertices:
            self.num_vertices = num_vertices
            self.labels = [str(i) for i in range(num_vertices)]
            self.mapa_labels = {str(i): i for i in range(num_vertices)}
        else:
            print("Erro: Forneça 'num_vertices' ou uma lista de 'labels'.")
            sys.exit(1)

        # Cria a matriz de adjacência NxN preenchida com zeros
        self.matriz = [[0] * self.num_vertices for i in range(self.num_vertices)]
```

IESB
Centro Universitário

Atividade 2 – Parte 3

- Implemente os métodos:
  - numero_de_vertices() – retorna o número de vértices
  - numero_de_arestas() – retorna o número de arestas
  - sequencia_de_graus() – retorna a sequência de graus

Atividade 2 – Parte 3

```python
def numero_de_vertices(self):
    # Retorna o número total de vértices no grafo.
    return self.num_vertices

def numero_de_arestas(self):
    # Retorna o número total de arestas no grafo.
    count = 0
    for i in range(self.num_vertices):
        for j in range(i + 1, self.num_vertices):
            if self.matriz[i][j] != 0:
                count += 1
    return count

def sequencia_de_graus(self):
    # Retorna uma lista com os graus de todos os vértices.
    return sorted([sum(row) for row in self.matriz])
```

IESB
Centro Universitário

Atividade 2 – Parte 4

- Crie os métodos para adicionar e remover arestas, e imprimir o Grafo.
  - adicionar_aresta(u, v) – adiciona uma nova aresta
  - remover_aresta(u, v) – remove uma aresta
  - imprimir() – imprime a matriz
- O grafo permite apenas uma aresta entre dois vértives u,v.

IESB
Centro Universitário

Atividade 2 – Parte 4

```python
def _obter_indice(self, vertice):
    if isinstance(vertice, str) and vertice in self.mapa_labels:
        return self.mapa_labels[vertice]
    elif isinstance(vertice, int) and 0 <= vertice < self.num_vertices:
        return vertice
    else:
        raise ValueError(f"Vértice '{vertice}' é inválido.")


def adicionar_aresta(self, u, v):
    """
    Adiciona a aresta entre os vértices u e v.
    """
    try:
        idx_u = self._obter_indice(u)
        idx_v = self._obter_indice(v)

        self.matriz[idx_u][idx_v] = 1
        self.matriz[idx_v][idx_u] = 1

        print(f"Aresta adicionada entre {u} e {v}.")
    except ValueError as e:
        print(f"Erro ao adicionar aresta: {e}")
```

IESB
Centro Universitário

Atividade 2 – Parte 4

```python
def _obter_indice(self, vertice):
    if isinstance(vertice, str) and vertice in self.mapa_labels:
        return self.mapa_labels[vertice]
    elif isinstance(vertice, int) and 0 <= vertice < self.num_vertices:
        return vertice
    else:
        raise ValueError(f"Vértice '{vertice}' é inválido.")


def adicionar_aresta(self, u, v):
    """
    Adiciona a aresta entre os vértices u e v.
    """
    try:
        idx_u = self._obter_indice(u)
        idx_v = self._obter_indice(v)

        self.matriz[idx_u][idx_v] = 1
        self.matriz[idx_v][idx_u] = 1

        print(f"Aresta adicionada entre {u} e {v}.")
    except ValueError as e:
        print(f"Erro ao adicionar aresta: {e}")
```

IESB
Centro Universitário

Atividade 2 – Parte 4

```python
def remover_aresta(self, u, v):
    """
    Remove a aresta entre os vértices u e v.
    """
    try:
        idx_u = self._obter_indice(u)
        idx_v = self._obter_indice(v)

        if self.matriz[idx_u][idx_v] == 0:
            print(f"Aresta entre {u} e {v} não existe.")
            return

        # Remove a aresta
        self.matriz[idx_u][idx_v] = 0
        self.matriz[idx_v][idx_u] = 0
        print(f"Aresta removida entre {u} e {v}.")

    except ValueError as e:
        print(f"Erro ao remover aresta: {e}")
```

IESB
Centro Universitário

Atividade 2 – Parte 4

```
def imprimir(self):
    """Imprime a matriz de adjacência de forma legível."""
    print("\nMatriz de Adjacência:")
    # Imprime o cabeçalho das colunas
    header = "   " + "  ".join(self.labels)
    print(header)
    print("—" * len(header))

    # Imprime as linhas com seus respectivos rótulos
    for i, linha in enumerate(self.matriz):
        print(f"{self.labels[i]} |", "  ".join(map(str, linha)))
    print()
```

IESB
Centro Universitário

Atividade 2 – Parte 5

- Instancie um Grafo com vértices com rótulos V = {A,B,C,D,E}.
- Adicione as arestas E = {(A,B), (A,C), (C,D), (C,E), (B,D)}
- Imprima o grafo, a quantidade de vértices, a quantidade de arestas, a sequência de graus.
- Remova a aresta e = (A,C)
- Imprima novamente o grafo.

IESB
Centro Universitário

Atividade 2 – Parte 5

```python
if __name__ == "__main__":

    vertices_labels = ['A', 'B', 'C', 'D', 'E']
    g = GrafoDenso(labels=vertices_labels)

    g.adicionar_aresta('A', 'B')
    g.adicionar_aresta('A', 'C')
    g.adicionar_aresta('C', 'D')
    g.adicionar_aresta('C', 'E')
    g.adicionar_aresta('B', 'D')
    g.imprimir()
    print(f"Número de vértices: {g.numero_de_vertices()}")
    print(f"Número de arestas: {g.numero_de_arestas()}")
    print(f"Sequência de graus: {g.sequencia_de_graus()}")
    g.remover_aresta('A', 'C')
    g.imprimir()
```

IESB
Centro Universitário

Atividade 2 – Parte 5

```
Aresta adicionada entre A e B.
Aresta adicionada entre A e C.
Aresta adicionada entre C e D.
Aresta adicionada entre C e E.
Aresta adicionada entre B e D.

Matriz de Adjacência:
   A  B  C  D  E
_____________________
A | 0  1  1  0  0
B | 1  0  0  1  0
C | 1  0  0  1  1
D | 0  1  1  0  0
E | 0  0  1  0  0

Número de vértices: 5
Número de arestas: 5
Sequência de graus: [1, 2, 2, 2, 3]
Aresta removida entre A e C.

Matriz de Adjacência:
   A  B  C  D  E
_____________________
A | 0  1  0  0  0
B | 1  0  0  1  0
C | 0  0  0  1  1
D | 0  1  1  0  0
E | 0  0  1  0  0
```