

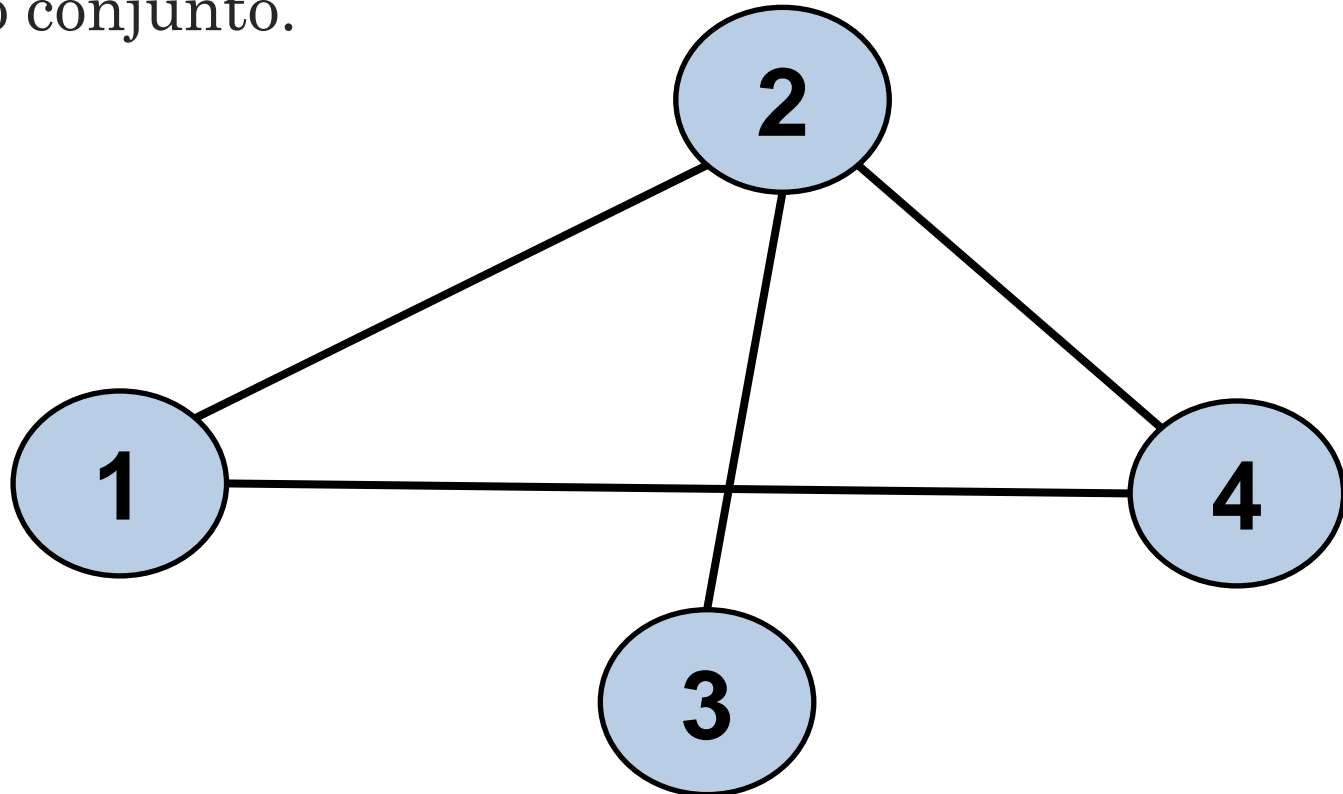
# **ESTRUTURA DE DADOS: GRAFOS**

Definições e conceitos

Prof. Jean Nunes

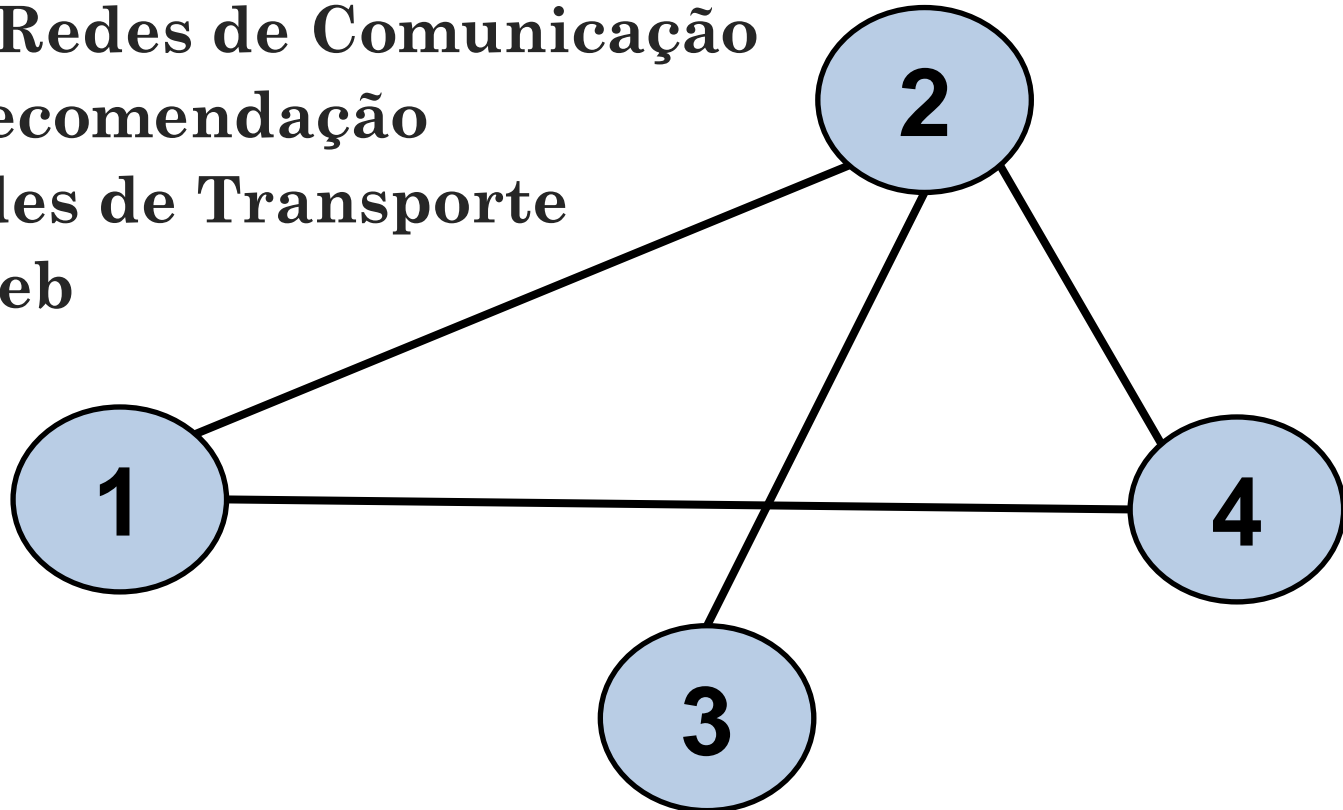
# Definição

- Como representar um conjunto de objetos e as suas relações?
  - Diversos tipos de aplicações necessitam disso
  - É um modelo matemático que representa as relações entre objetos de um determinado conjunto.



# Definição

- Grafos em computação
  - Forma de solucionar problemas computáveis
  - Buscam o desenvolvimento de algoritmos mais eficientes
    - **Redes Sociais**
    - **Roteamento e Redes de Comunicação**
    - **Sistemas de Recomendação**
    - **Análise de Redes de Transporte**
    - **Pesquisa na Web**



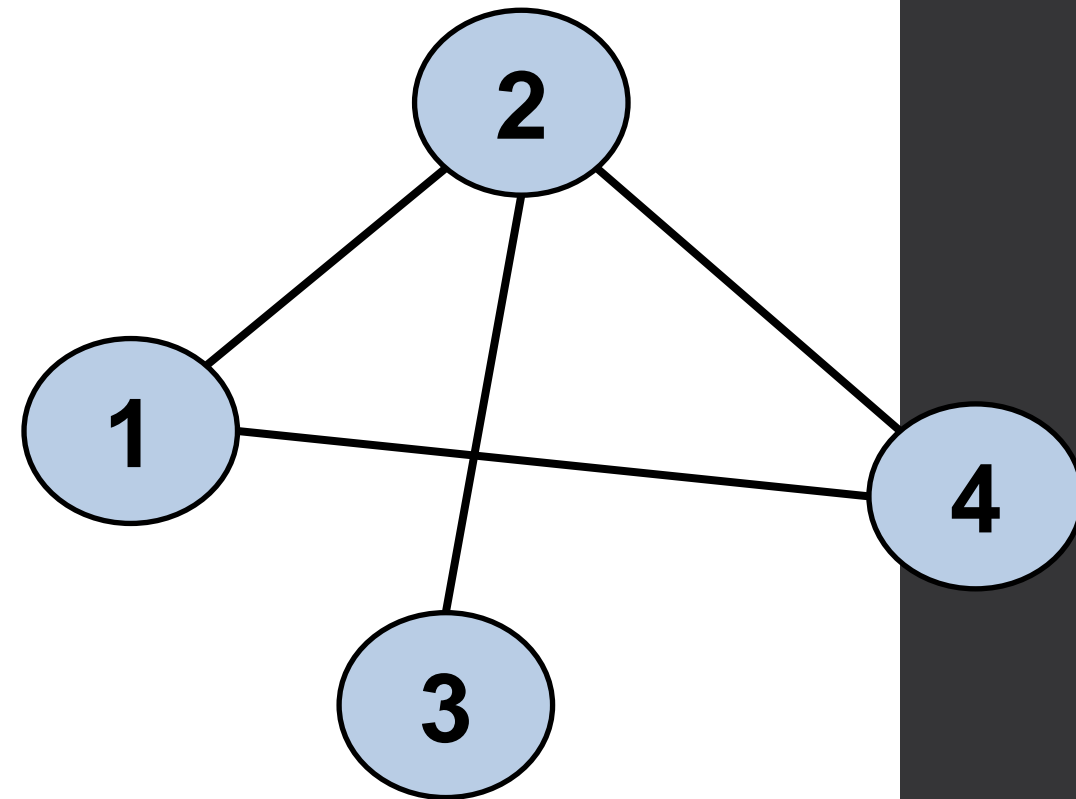
# Definição

- Um grafo  $G(V,A)$  é definido por dois conjuntos
  - Conjunto  $V$  de vértices (não vazio)
    - Itens representados em um grafo;
  - Conjunto  $A$  de arestas
    - Utilizadas para conectar pares de vértices, usando um critério previamente estabelecido.

$G(V,A)$

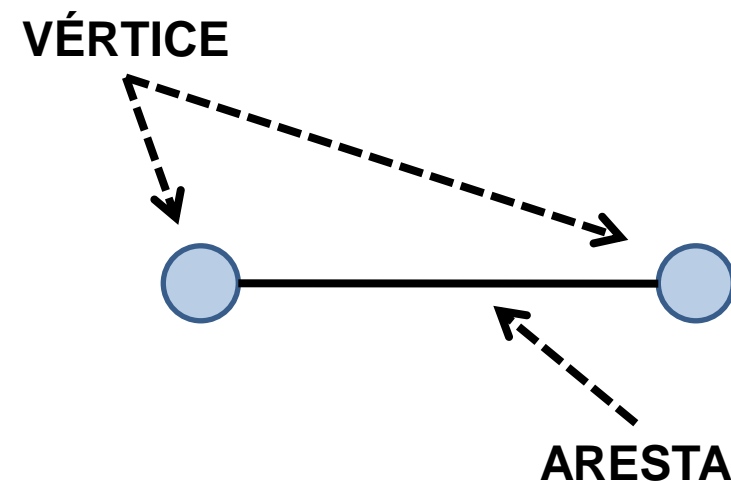
$V = \{1,2,3,4\}$

$A = \{\{1,2\},\{1,4\},\{2,3\},\{2,4\}\}$



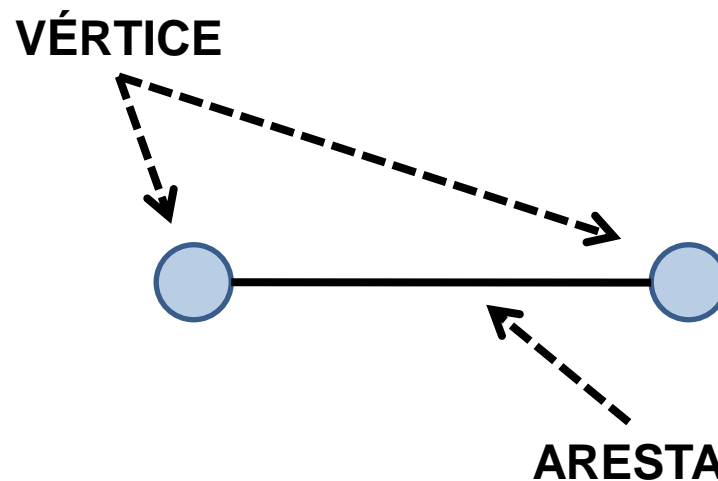
# Conceitos básicos

- Vértice é cada um dos itens representados no grafo.
  - O seu significado depende da natureza do problema modelado
    - Pessoas, uma tarefa em um projeto, lugares em um mapa, etc.



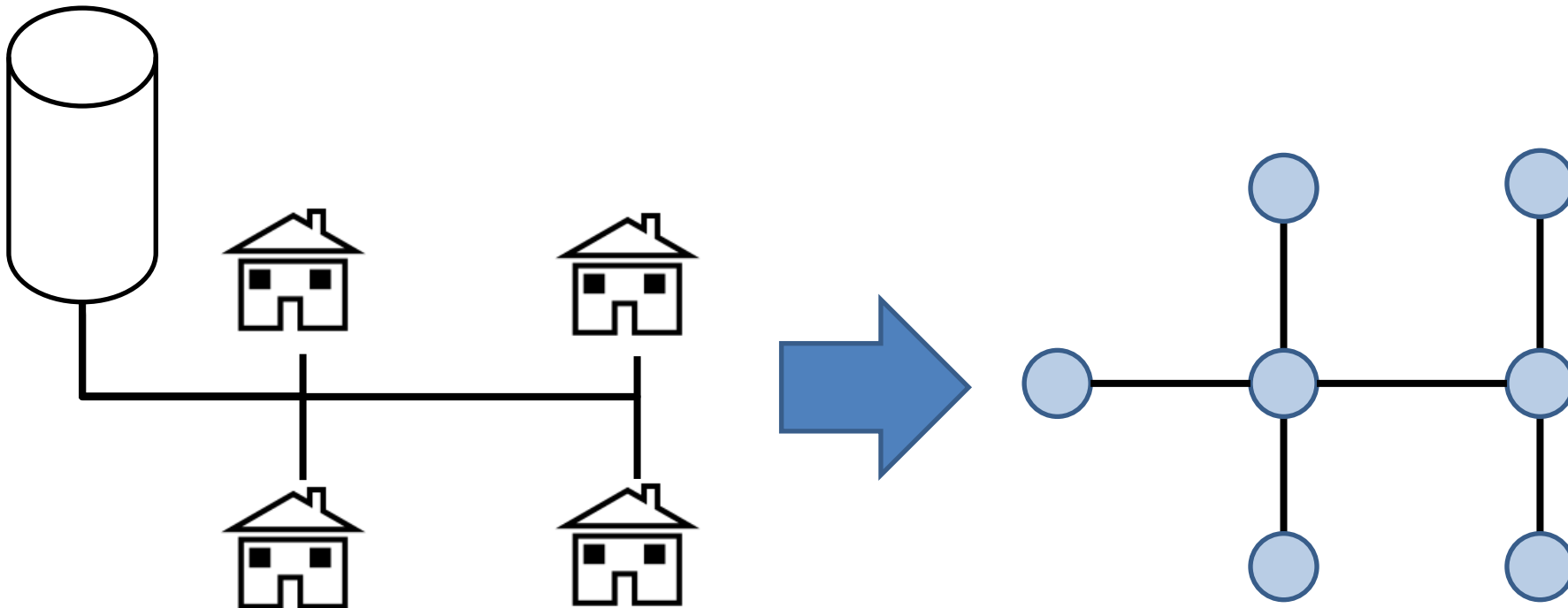
# Conceitos básicos

- Aresta (ou arco) liga dois vértices
  - Diz qual a relação entre eles
  - Dois vértices são **adjacentes** se existir uma aresta ligando eles.
    - Pessoas (parentesco entre elas ou amizade), tarefas de um projeto (pré-requisito entre as tarefas), lugares de um mapa (estradas que existem ligando os lugares), etc.



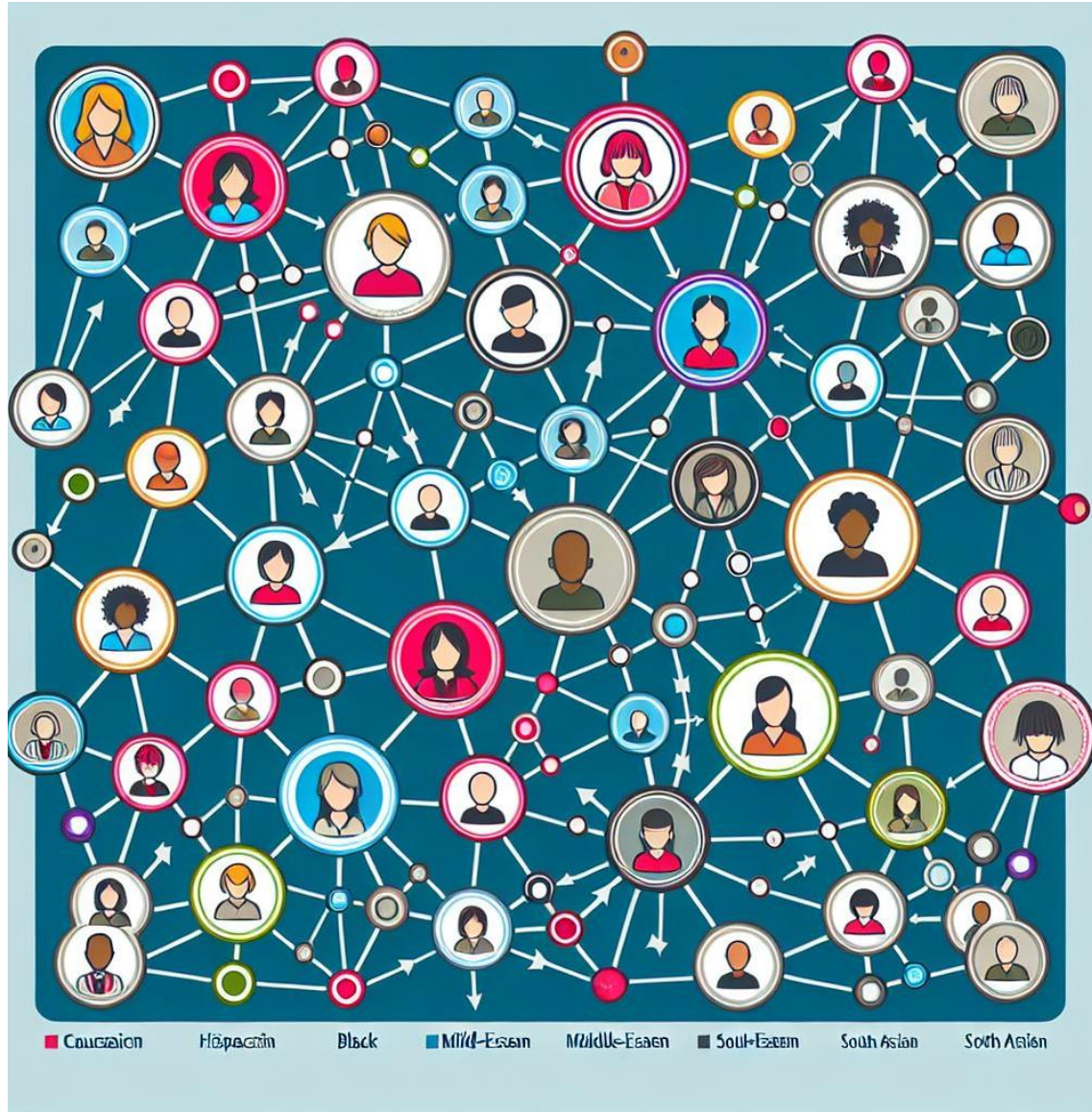
# Conceitos básicos

- Praticamente qualquer objeto pode ser representado como um grafo.
  - Exemplo: sistema de distribuição de água



# Conceitos básicos

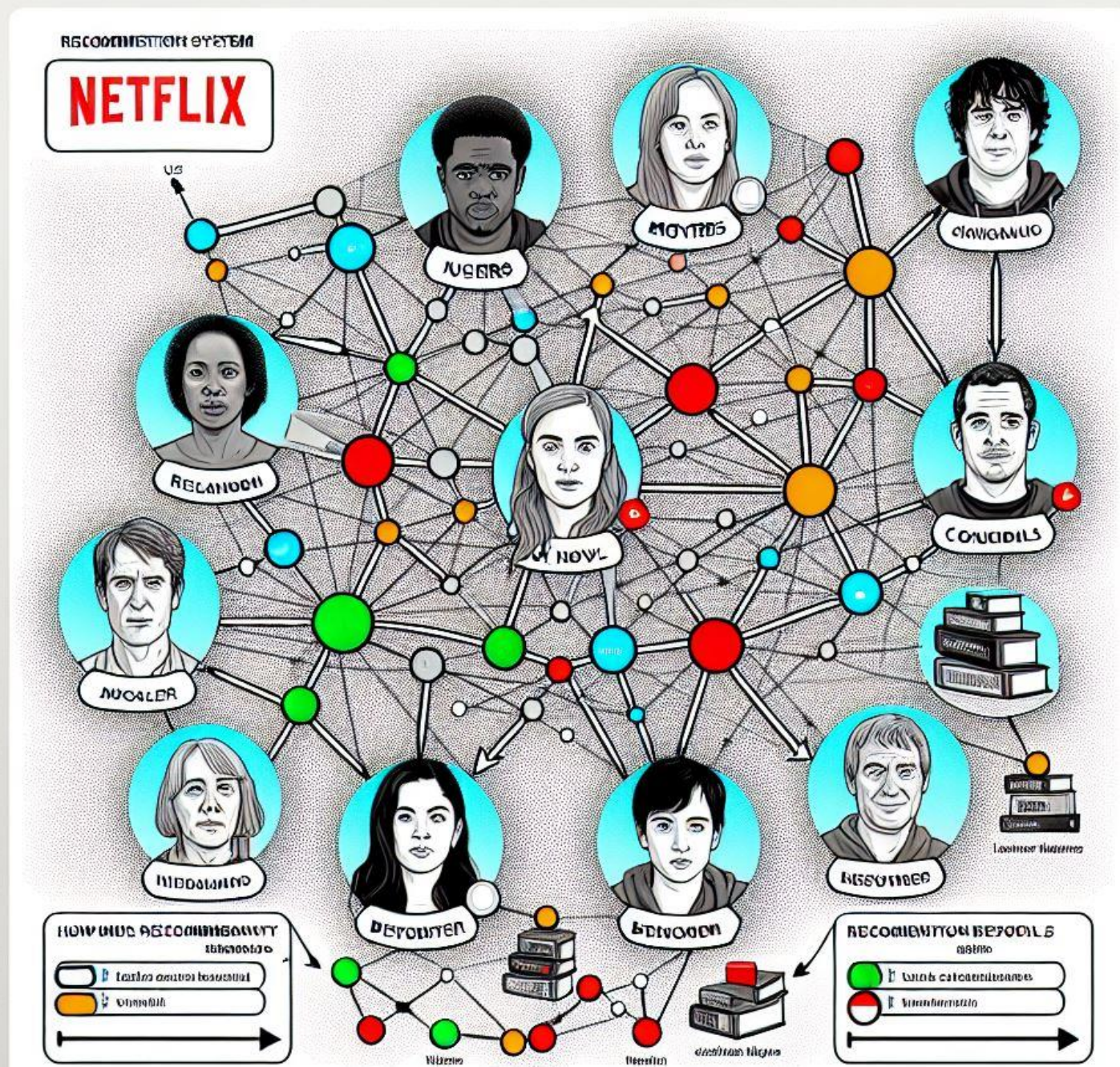
- Praticamente qualquer objeto pode ser representado como um grafo
  - Exemplo: rede social





# Conceitos básicos

- Praticamente qualquer objeto pode ser representado como um grafo
  - Exemplo: sistema de recomendação

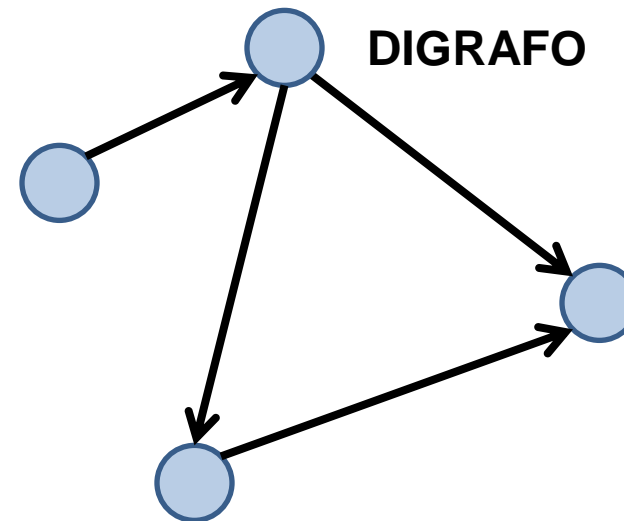
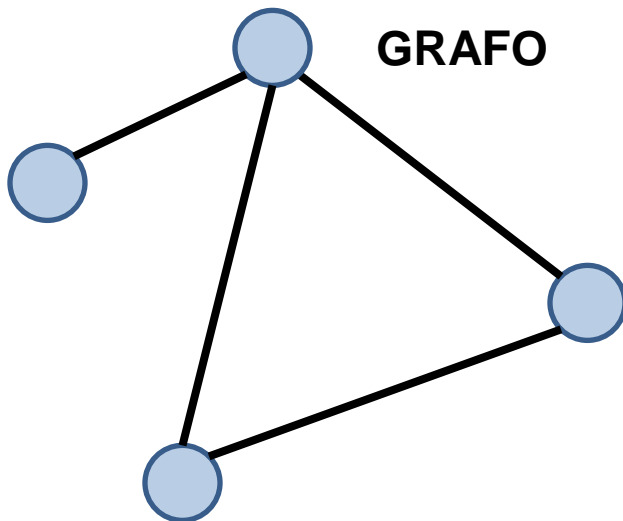




- Praticamente qualquer objeto pode ser representado como um grafo
  - Exemplo: análise de sistema de transporte

# Conceitos básicos

- As arestas podem ou não ter direção
  - Existe uma orientação quanto ao sentido da aresta
    - Em um grafo direcionado ou **digrafo**, se uma aresta liga os vértices **A** a **B**, isso significa que podemos ir de **A** para **B**, mas não o contrário



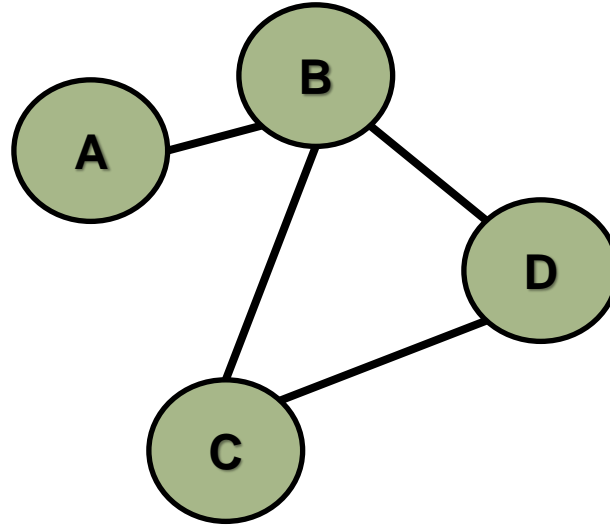
# Conceitos básicos

- Grau

- Indica o número de arestas que conectam um vértice do grafo a outros vértices
  - número de vizinhos que aquele vértice possui no grafo (que chegam ou partem dele)
- No caso dos dígrafos, temos dois tipos de grau:
  - **grau de entrada:** número de arestas que chegam ao vértice;
  - **grau de saída:** número de arestas que partem do vértice.

# Conceitos básicos

GRAFO



## Grau

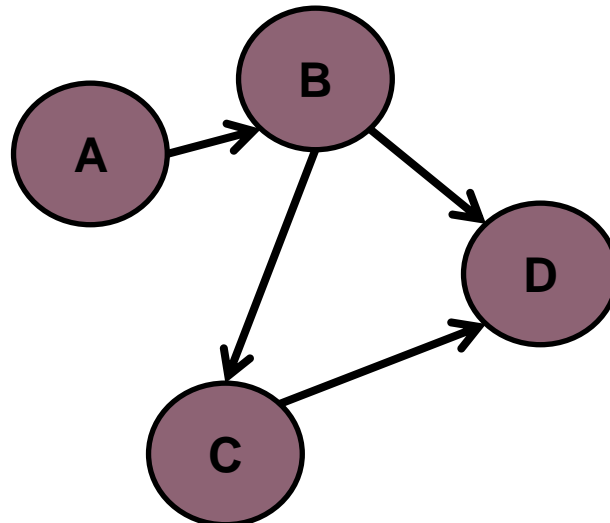
$$G(A) = 1$$

$$G(B) = 3$$

$$G(C) = 2$$

$$G(D) = 2$$

DIGRAFO



## Grau Entrada

$$G(A) = 0$$

$$G(B) = 1$$

$$G(C) = 1$$

$$G(D) = 2$$

## Grau Saída

$$G(A) = 1$$

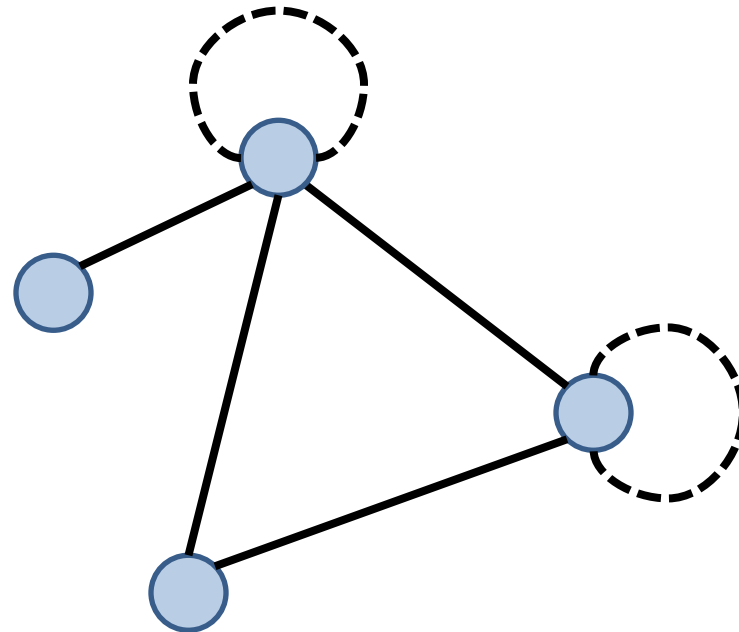
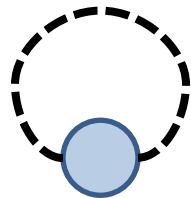
$$G(B) = 2$$

$$G(C) = 1$$

$$G(D) = 0$$

# Conceitos básicos

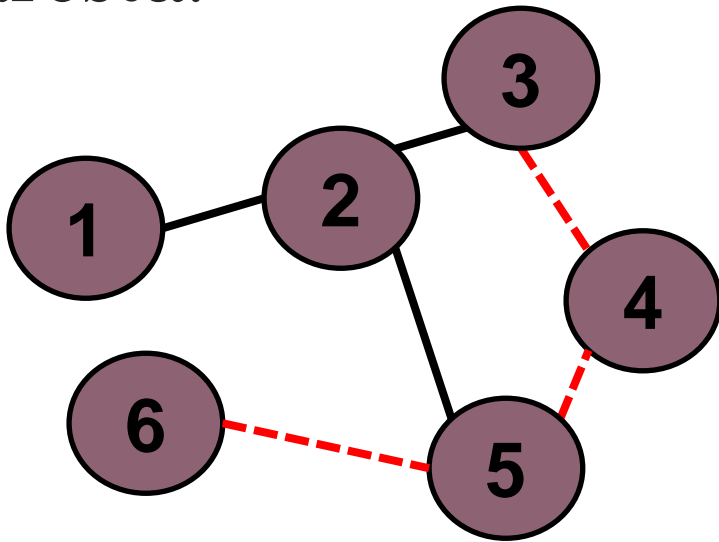
- Laço
  - Uma aresta é chamada de laço se seu vértice de partida é o mesmo que o de chegada
    - A aresta conecta o vértice a ele mesmo



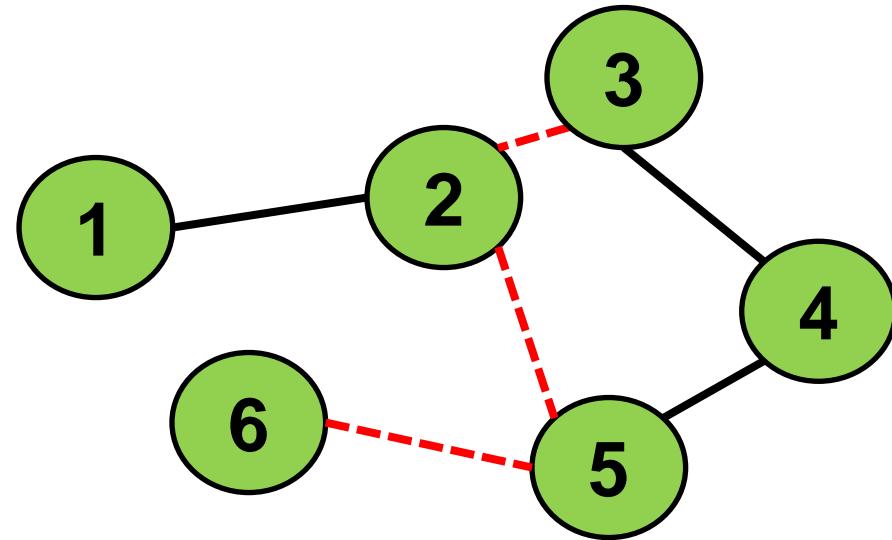
# Conceitos básicos

- Caminho

- Um caminho entre dois vértices é uma sequência de vértices onde cada vértice está conectado ao vértice seguinte por meio de uma aresta.



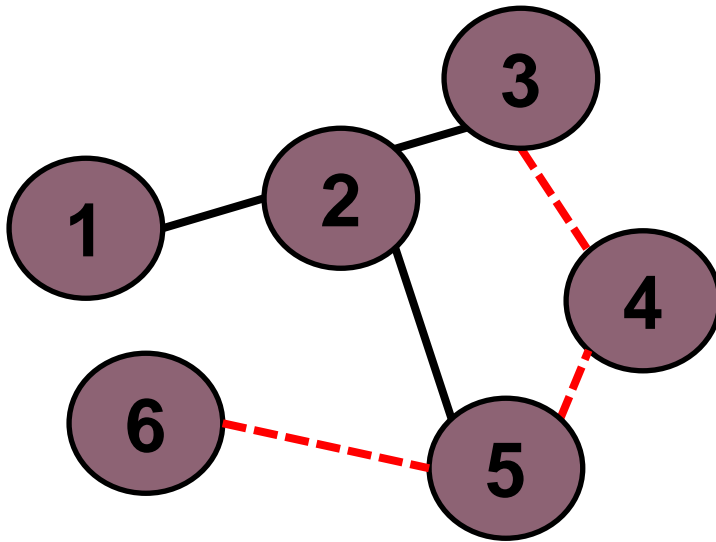
**CAMINHO: 3-4-5-6**



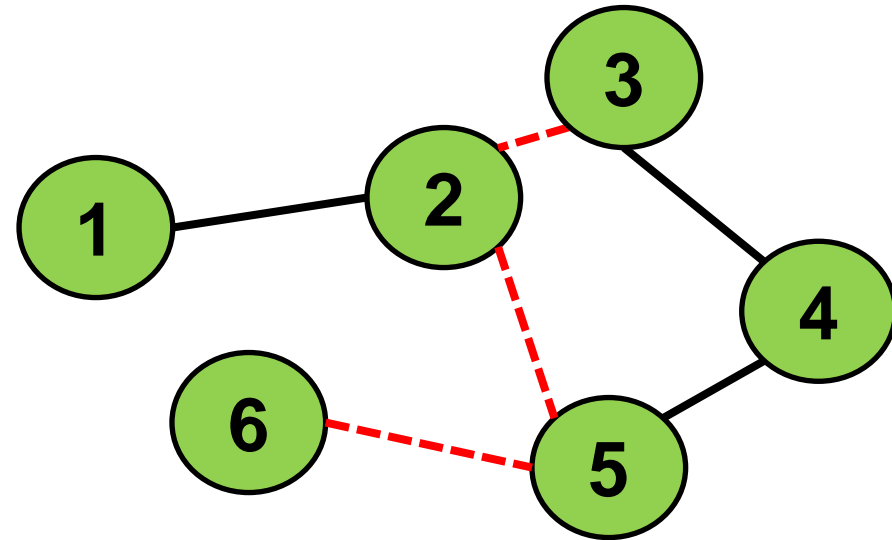
**CAMINHO: 3-2-5-6**

# Conceitos básicos

- Caminho
  - **Comprimento do caminho:** número de vértices que precisamos percorrer de um vértice até o outro



**CAMINHO: 3-4-5-6**



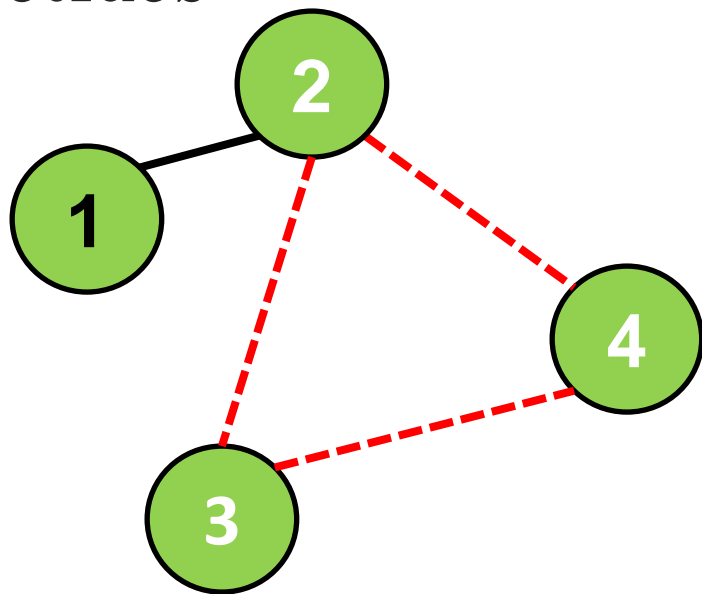
**CAMINHO: 3-2-5-6**



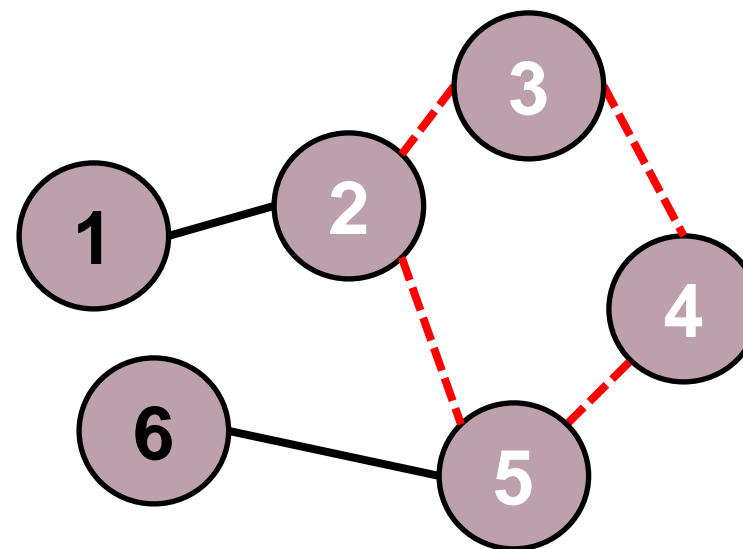
# Conceitos básicos

- **Ciclo**

- Caminho onde o vértice inicial e o final são o mesmo vértice.
- Note que um ciclo é um caminho fechado sem vértices repetidos



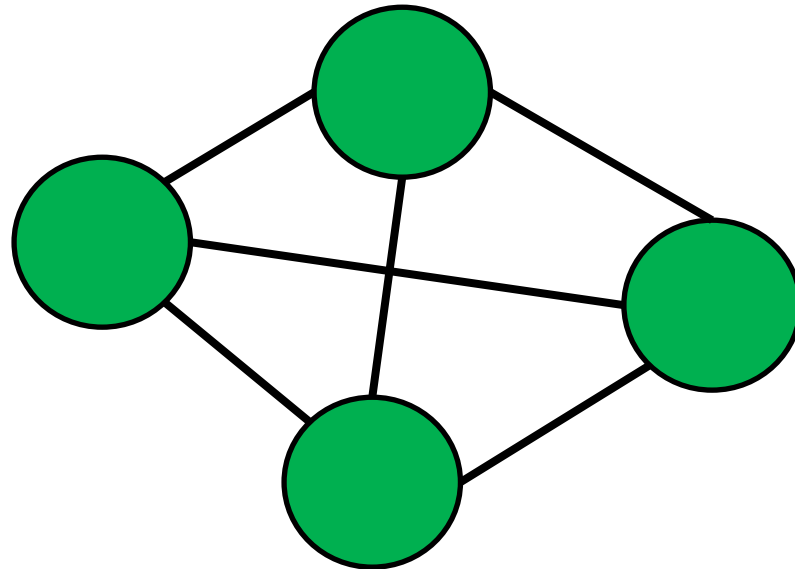
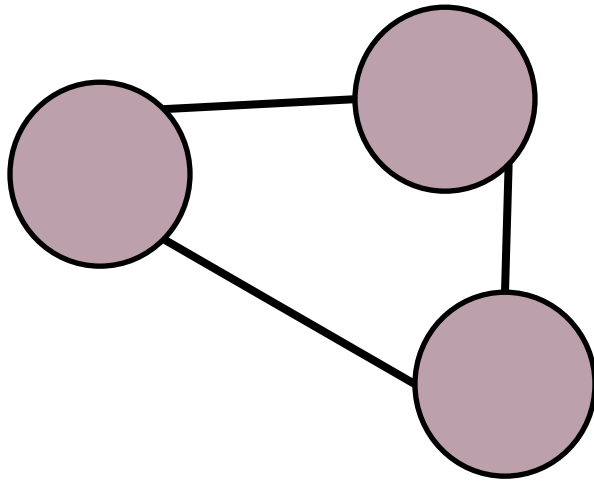
CICLO: 2-3-4



CICLO: 2-3-4-5

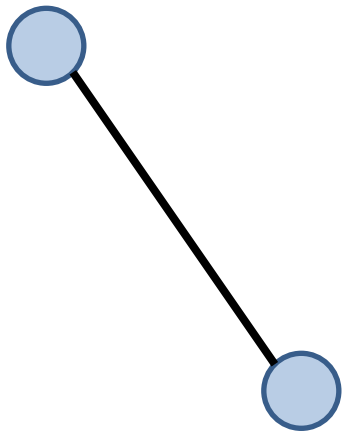
# Tipos de Grafos

- Grafo completo
  - Grafo simples onde cada vértice se conecta a todos os outros vértices do grafo.

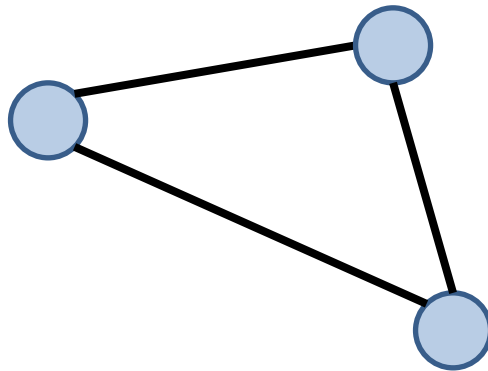


# Tipos de Grafos

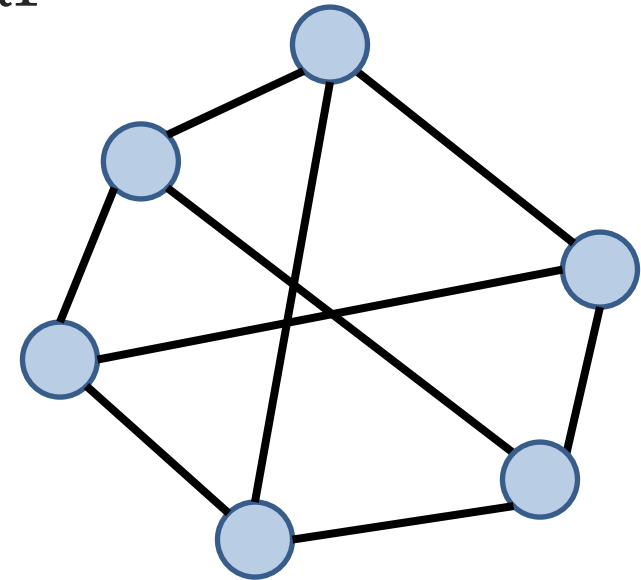
- Grafo regular
  - Grafo onde todos os seus vértices possuem o mesmo grau (número de arestas ligadas a ele)
  - Todo grafo completo é também regular



**Grau = 1**



**Grau = 2**

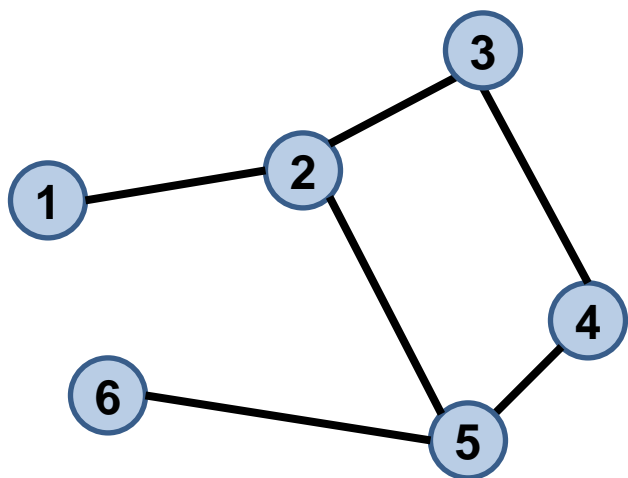


**Grau = 3**

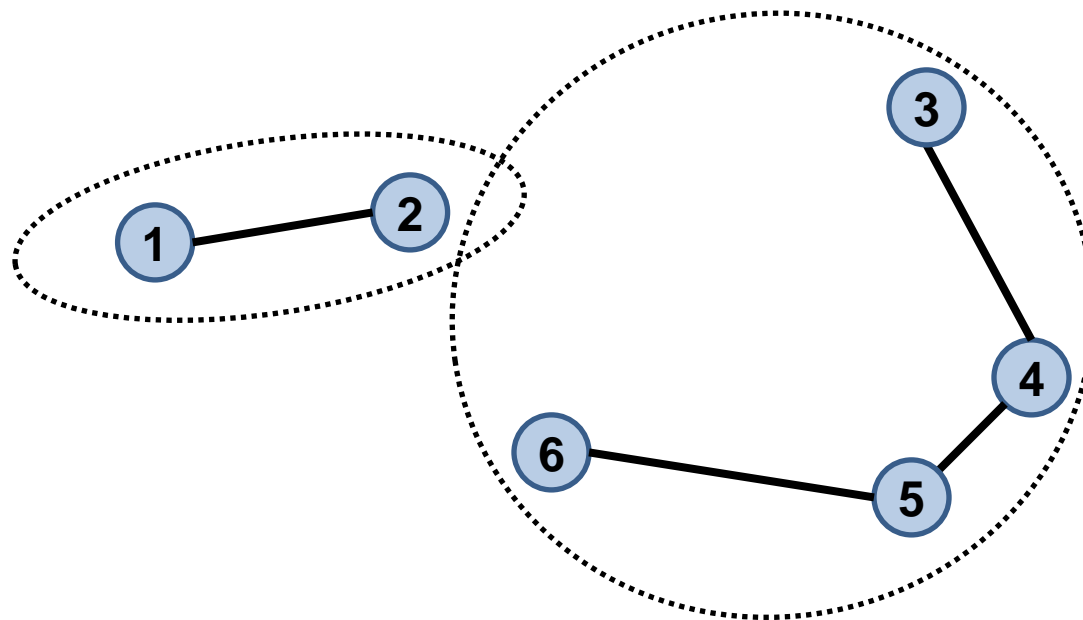
# Tipos de Grafos

- Subgrafo

- $G_s(V_s, A_s)$  é um subgrafo de  $G(V, A)$  se o conjunto de vértices  $V_s$  for um subconjunto de  $V$ ,  $V_s \subseteq V$ , e se o conjunto de arestas  $A_s$  for um subconjunto de  $A$ ,  $A_s \subseteq A$ .



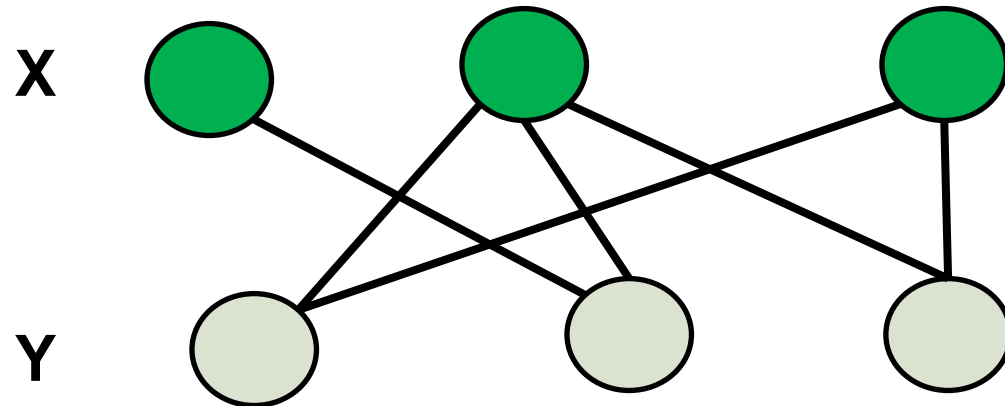
GRAFO



SUBGRAFOS

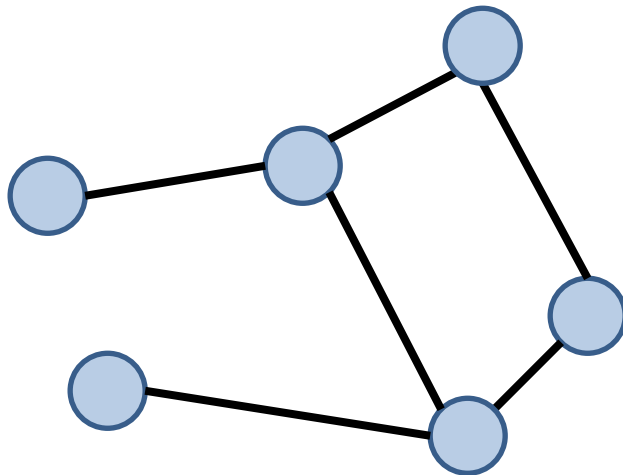
# Tipos de Grafos

- Grafo bipartido
  - Um grafo  $G(V,A)$  onde o seu conjunto de vértices pode ser divididos em dois subconjuntos  $X$  e  $Y$  sem intersecção.
  - As arestas conectam apenas os vértices que estão em subconjuntos diferentes

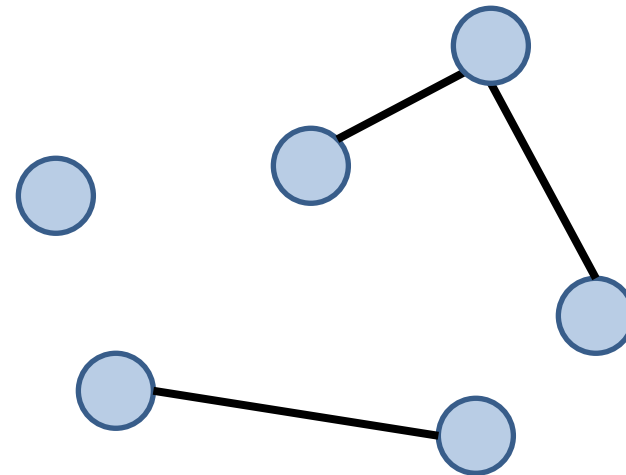


# Tipos de Grafos

- Grafo conexo e desconexo
  - **Grafo conexo**: existe um caminho ligando quaisquer dois vértices.
  - Quando isso não acontece, temos um **grafo desconexo**



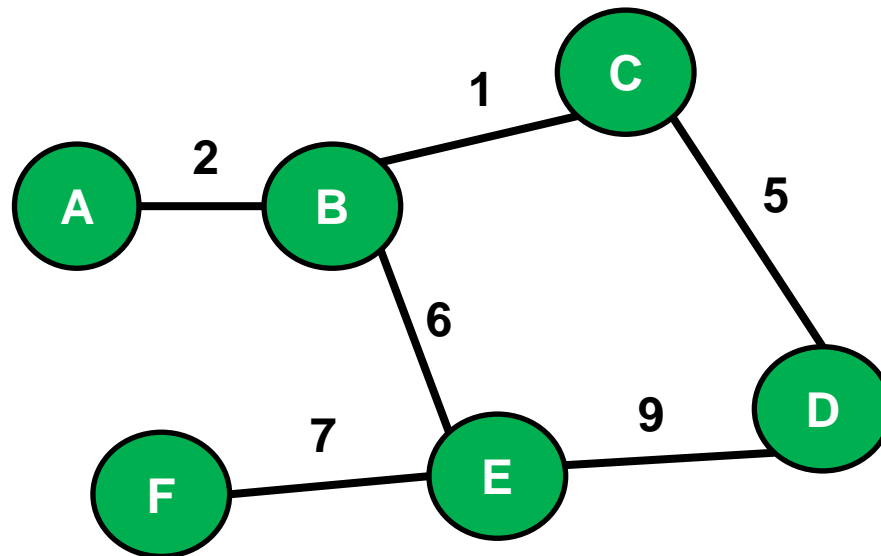
GRAFO CONEXO



GRAFO DESCONEXO

# Tipos de Grafos

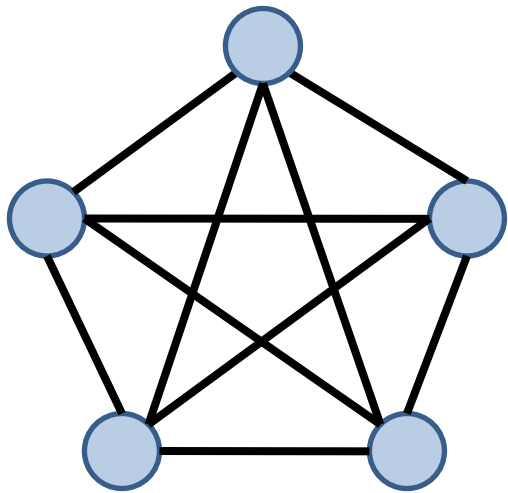
- Grafo ponderado
  - É um grafo que possui **pesos** (valor numérico) associados a cada uma de suas arestas.



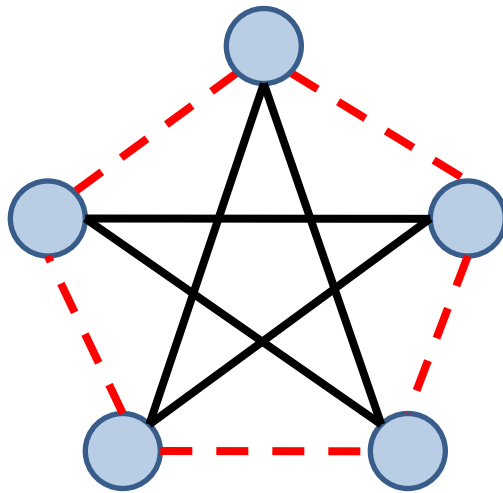
# Tipos de Grafos

- Grafo Hamiltoniano

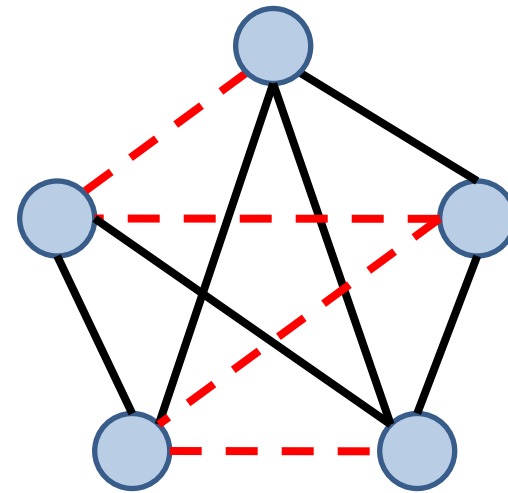
- Grafo que possui um caminho que visita todos os seus vértices apenas uma vez.
- Pode ser um ciclo



**GRAFO HAMILTONIANO**



**CICLO HAMILTONIANO**



**CAMINHO  
HAMILTONIANO**





# **ESTRUTURA DE DADOS: GRAFOS**

Implementação

Prof. Jean Nunes

# Tipos de representação

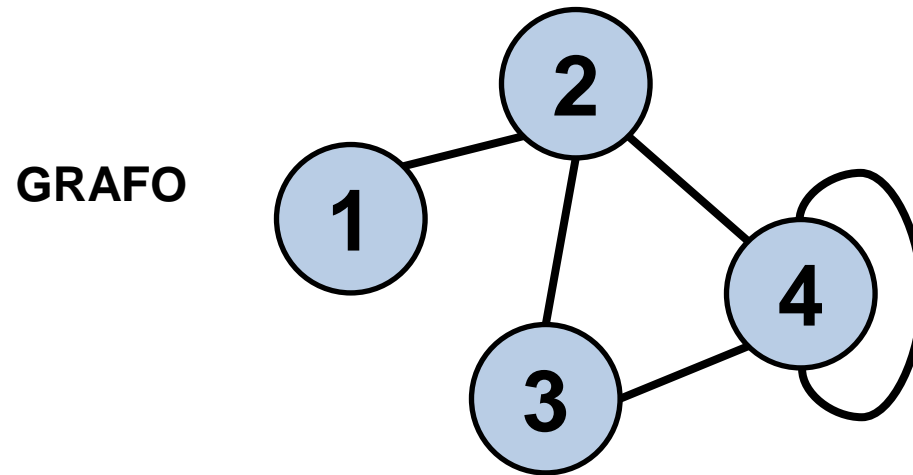
- Como representar um grafo no computador?
  - Existem duas abordagens muito utilizadas:
    - Matriz de Adjacência
    - Lista de Adjacência
- Qual a representação que deve ser utilizada?
  - Depende da aplicação!

# Tipos de representação

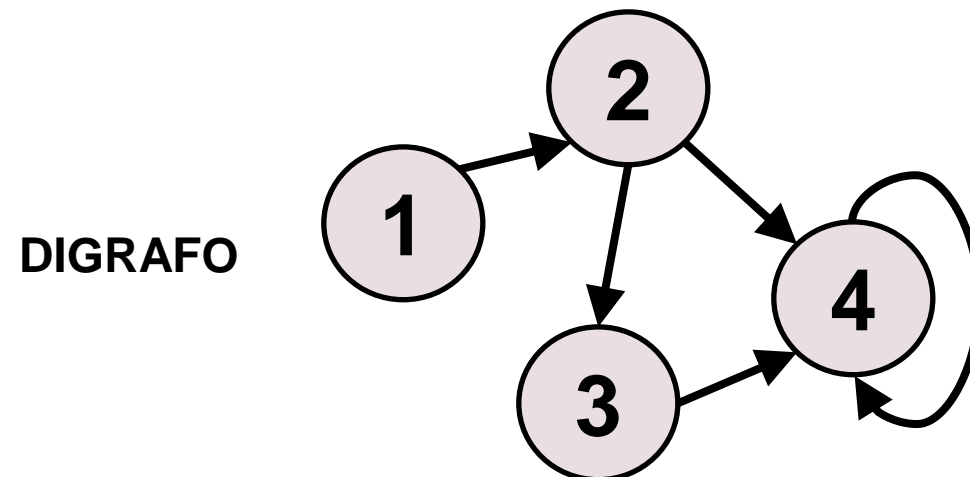
- Matriz de adjacência
  - Utiliza uma matriz  $N \times N$  para armazenar o grafo, onde  $N$  é o número de vértices
    - Alto custo computacional,  $O(N^2)$
  - Uma aresta é representada por uma marca na posição  $(i, j)$  da matriz
    - Aresta liga o vértice  $i$  ao  $j$

# Tipos de representação

- Matriz de adjacência



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



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

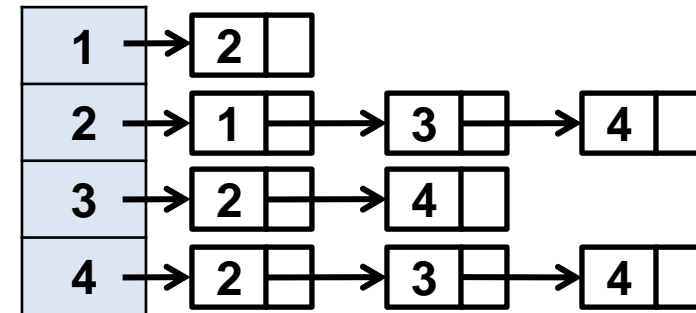
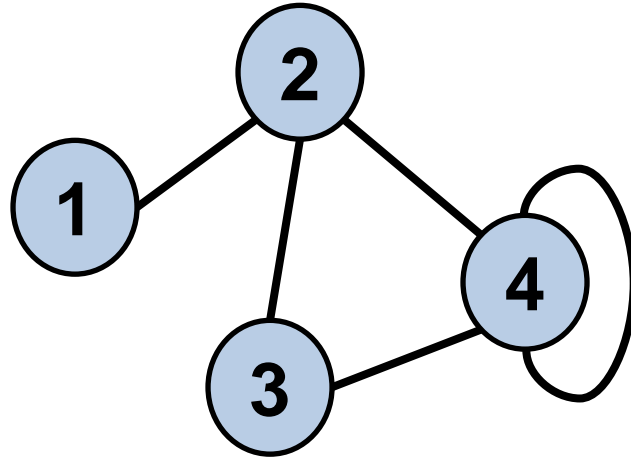
# Tipos de representação

- Lista de adjacência
  - Utiliza uma lista para descrever as relações entre os vértices.
    - Um grafo contendo  $N$  vértices utiliza um *array* de ponteiros de tamanho  $N$  para armazenar os vértices do grafo
    - Para cada vértice é criada uma lista de arestas, onde cada posição da lista armazena o índice do vértice a qual aquele vértice se conecta

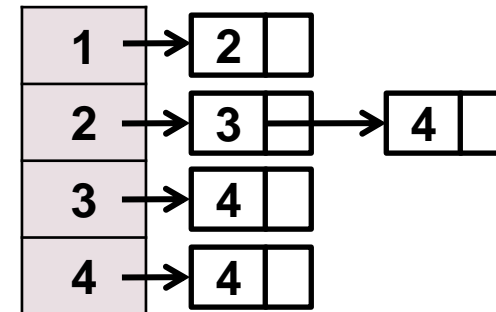
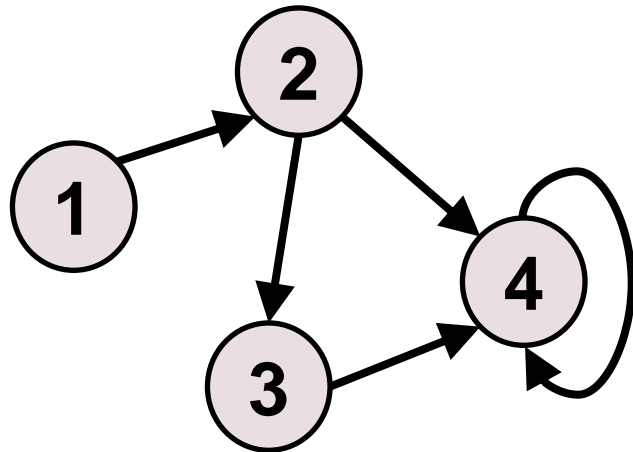
# Tipos de representação

- Lista de adjacência

GRAFO



DIGRAFO



# Tipos de representação

- Qual representação utilizar?
  - Lista de adjacência é mais indicada para um grafo que possui muitos vértices mas poucas arestas ligando esses vértices.
  - A medida que o número de arestas cresce e não havendo nenhuma outra informação associada a aresta (por exemplo, seu peso), o uso de uma matriz de adjacência se torna mais eficiente

# Grafo: Implementando em C

- Vamos usar uma **lista de adjacência**
  - Lista de arestas: lista dinâmica encadeada

```
1 //Arquivo Grafo.h
2 typedef struct grafo Grafo;
3 typedef struct vertice Vertice;
```

```
6 // Grafo.c
7 // Estrutura para representar um
8 // vertice na lista de adjacência
9 struct vertice {
10     int destino;
11     float peso;
12     struct vertice* prox;
13 };
14
15 struct listaArestas{
16     struct vertice* head;
17 };
```

```
18 // Grafo.c
19 // Definicao do tipo Grafo
20 struct grafo{
21     int eh_ponderado;
22     int nro_vertices;
23     int grau_max;
24     struct listaArestas* arestas;
25     int* grau;
26 };
```

Tamanho das  
listas

Array de  
listas

Qtd de elementos  
em cada lista



# Grafo: Implementando em C

- Criando um grafo

```
4 //Arquivo Grafo.h
5 Grafo* cria_Grafo(int nro_vertices,
6                   int grau_max,
7                   int eh_ponderado);

7 //main.c
8 Grafo* gr = cria_Grafo(10, 7, 0);
```

# Grafo: Implementando em C

- Criando um grafo

```
39 // Grafo.c
40 Grafo* cria_Grafo(int nro_vertices, int grau_max, int eh_ponderado){
41     Grafo *gr;
42     gr = (Grafo*) malloc(sizeof(struct grafo));
43     if(gr != NULL){
44         int i;
45         gr->nro_vertices = nro_vertices;
46         gr->grau_max = grau_max;
47         gr->eh_ponderado = eh_ponderado;
48         gr->grau = (int*) calloc(nro_vertices, sizeof(int));
49
50         gr->arestas = (struct listaArestas*) malloc(nro_vertices * sizeof(struct listaArestas));
51         for(i=0; i<nro_vertices; i++)
52             gr->arestas[i].head = NULL;
53     }
54     return gr;
55 }
```

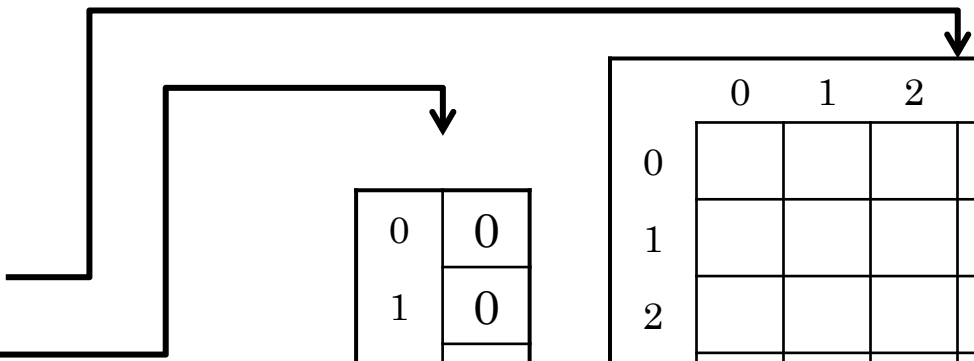
Cria matriz  
arestas

{

# Grafo: Implementando em C

```
18 // Grafo.c
19 // Definicao do tipo Grafo
20 struct grafo{
21     int eh_ponderado;
22     int nro_vertices;
23     int grau_max;
24     struct listaArestas* arestas;
25     int* grau;
26 };

7 //main.c
8 Grafo* gr = cria_Grafo(10, 7, 0);
```



0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							

Cria um grafo de 10 vértices.

Cada vértice se conecta com até outros 7 vértices

- Matriz 10x7 para as arestas
- Vetor “grau” guarda o número de conexões de cada um dos 10 vértices

# Grafo: Implementando em C

- Liberando o grafo

```
79 void libera_Grafo(Grafo* gr){
80     if(gr != NULL){
81         int i;
82         for(i=0; i<gr->nro_vertices; i++){
83             Vertice* temp = gr->arestas[i].head;
84             while (temp != NULL) {
85                 Vertice* proximo = temp->prox;
86                 free(temp);
87                 temp = proximo;
88             }
89         }
90         free(gr->arestas);
91         free(gr->grau);
92         free(gr);
93     }
94 }
```

Libera matriz arestas  
representada pelas  
listas

Libera vetor de  
vértices, vetor de  
grau e grafo

# Grafo: Implementando em C

- Inserindo uma aresta

```
11 //Arquivo Grafo.h
12 int insereAresta(Grafo* gr, int orig, int dest, int eh_digrafo, float peso);
```

```
10 // main.c
11 insereAresta(gr, 0, 1, 0, 3);
12 insereAresta(gr, 1, 3, 0, 2);
13 insereAresta(gr, 3, 2, 0, 4);
14 insereAresta(gr, 6, 1, 0, 1);
```

# Grafo: Implementando em C

- Inserindo uma aresta

```
57 // Grafo.c
58 int insereAresta(Grafo* gr, int orig, int dest, int eh_digrafo, float peso){
59     if(gr == NULL)
60         return 0;
61     if(orig < 0 || orig >= gr->nro_vertices)
62         return 0;
63     if(dest < 0 || dest >= gr->nro_vertices)
64         return 0;
65
66     Vertice* novoVert = criarVertice(dest);
67     novoVert->prox = gr->arestas[orig].head;
68     if(gr->eh_ponderado)
69         novoVert->peso = peso;
70     gr->arestas[orig].head = novoVert;
71
72     gr->grau[orig]++;
73
74     if(eh_digrafo == 0)
75         insereAresta(gr, dest, orig, 1, peso);
76     return 1;
77 }
```

Verifica se vértices existem

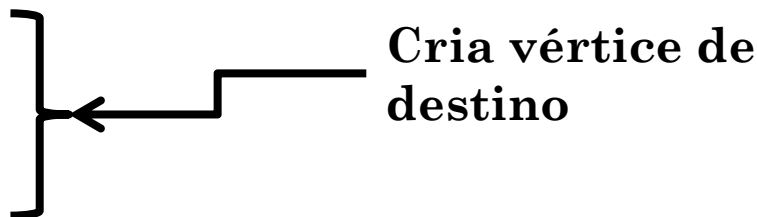
Insere no início da lista

Insere outra aresta se NÃO for dígrafo

# Grafo: Implementando em C

- Inserindo uma aresta

```
29 // Função para criar um novo vertice na lista de adjacência
30 Vertice* criarVertice(int destino) {
31     Vertice* novoVertice = (Vertice*) malloc(sizeof(struct vertice));
32     if(novoVertice != NULL){
33         novoVertice->destino = destino;
34         novoVertice->prox = NULL;
35     }
36     return novoVertice;
37 }
```



Cria vértice de destino

# Grafo: Implementando em C

```
insereAresta (gr, 0, 1, 0, 0) ;
```

**Antes da inserção**

		0	1	2	3	4	5	6
0	0							
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							
7	0							
8	0							
9	0							

**Após a inserção**

		0	1	2	3	4	5	6
0	1							
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							
7	0							
8	0							
9	0							



# Grafo: Implementando em C

```
insereAresta (gr, 0, 1, 0, 0) ;  
insereAresta (gr, 1, 3, 0, 0) ;
```

Antes da inserção

		0	1	2	3	4	5	6
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	0							
3	0							
4	0							
5	0							
6	0							
7	0							
8	0							
9	0							

Após a inserção

		0	1	2	3	4	5	6
0	1	0	0	0	0	0	0	0
1	2	3	0	0	0	0	0	0
2	0							
3	1	1						
4	0							
5	0							
6	0							
7	0							
8	0							
9	0							

# Grafo: Implementando em C

```
insereAresta (gr, 0, 1, 0, 0) ;  
insereAresta (gr, 1, 3, 0, 0) ;  
insereAresta (gr, 3, 2, 0, 0) ;
```

Antes da inserção

	0	1	2	3	4	5	6
0	1						
1	2						
2	0						
3	1						
4	0						
5	0						
6	0						
7	0						
8	0						
9	0						

Após a inserção

	0	1	2	3	4	5	6
0	1						
1	3	0					
2	3						
3	2	1					
4	0						
5	0						
6	0						
7	0						
8	0						
9	0						

# Grafo: Implementando em C

```
insereAresta(gr,0,1,0,0);  
insereAresta(gr,1,3,0,0);  
insereAresta(gr,3,2,0,0);  
insereAresta(gr,6,1,0,0);
```

**Antes da inserção**

		0	1	2	3	4	5	6
0	1	1						
1	2	3	0					
2	1	3						
3	2	2	1					
4	0							
5	0							
6	0							
7	0							
8	0							
9	0							

**Após a inserção**

		0	1	2	3	4	5	6
0	1	1						
1	3	6	3	0				
2	1	3						
3	2	2	1					
4	0							
5	0							
6	1	1						
7	0							
8	0							
9	0							

# Referências

