

# GRAFOS E DIGRAFOS

Vanessa Braganholo  
Estruturas de Dados e Seus  
Algoritmos

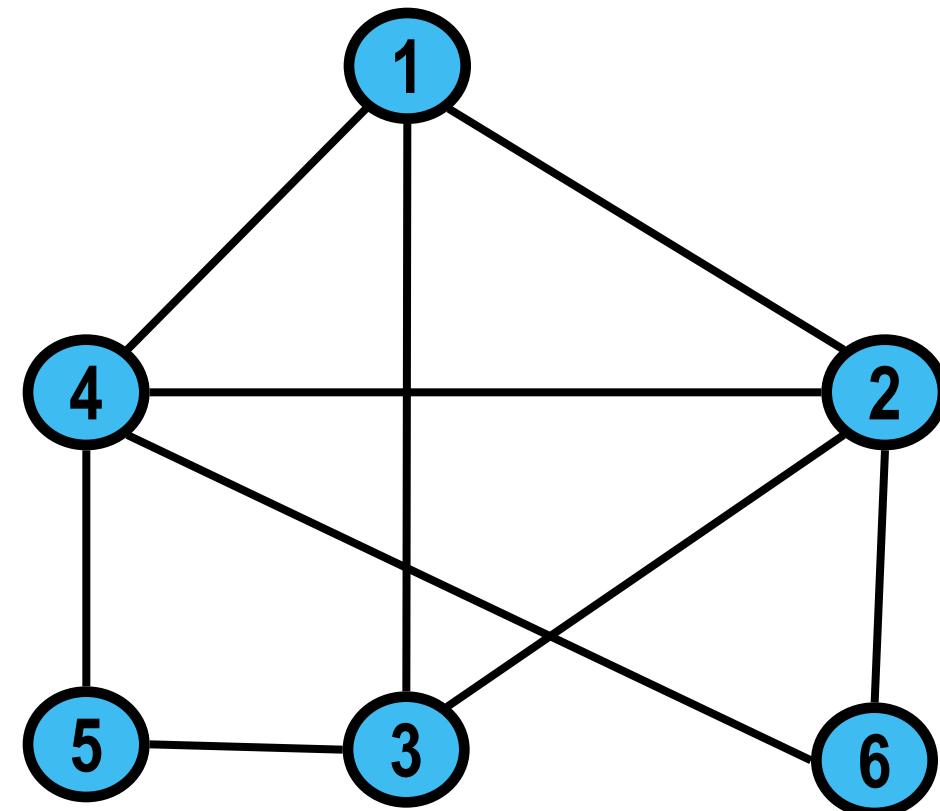
# GRAFOS

Definições

Representação

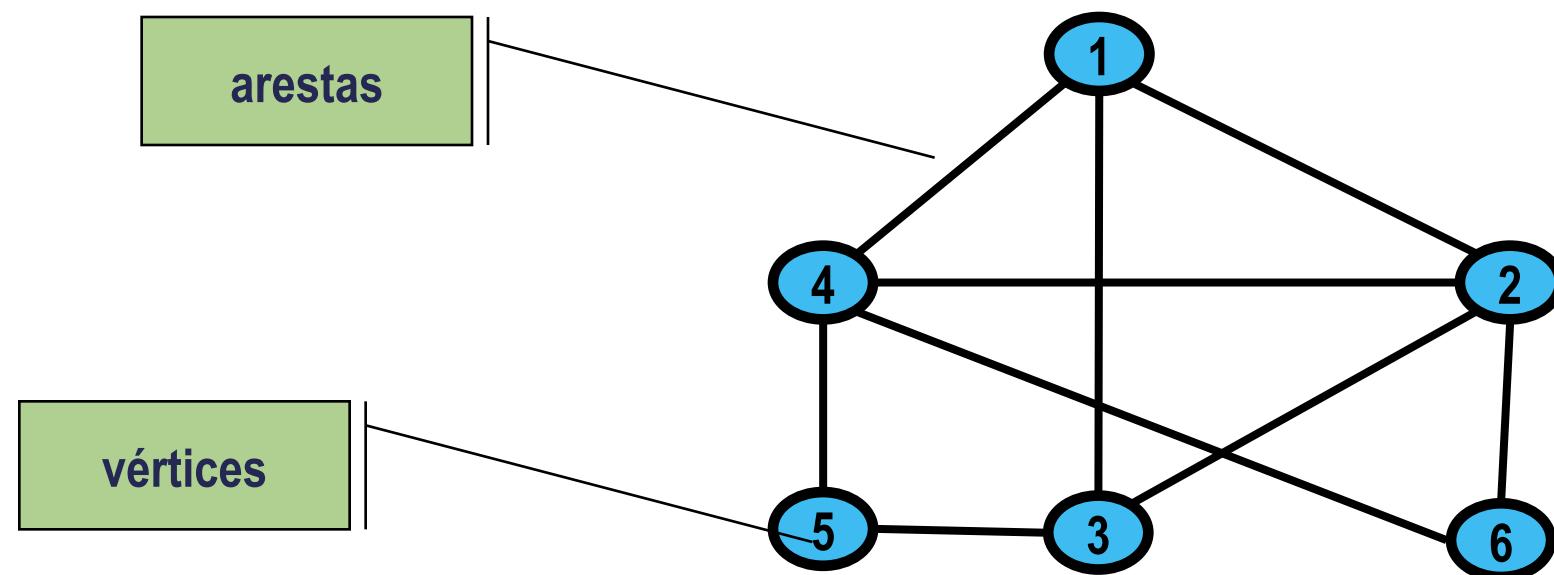
Algoritmos

- Busca
- Inserção (arestas e vértices)
- Exclusão (arestas e vértices)



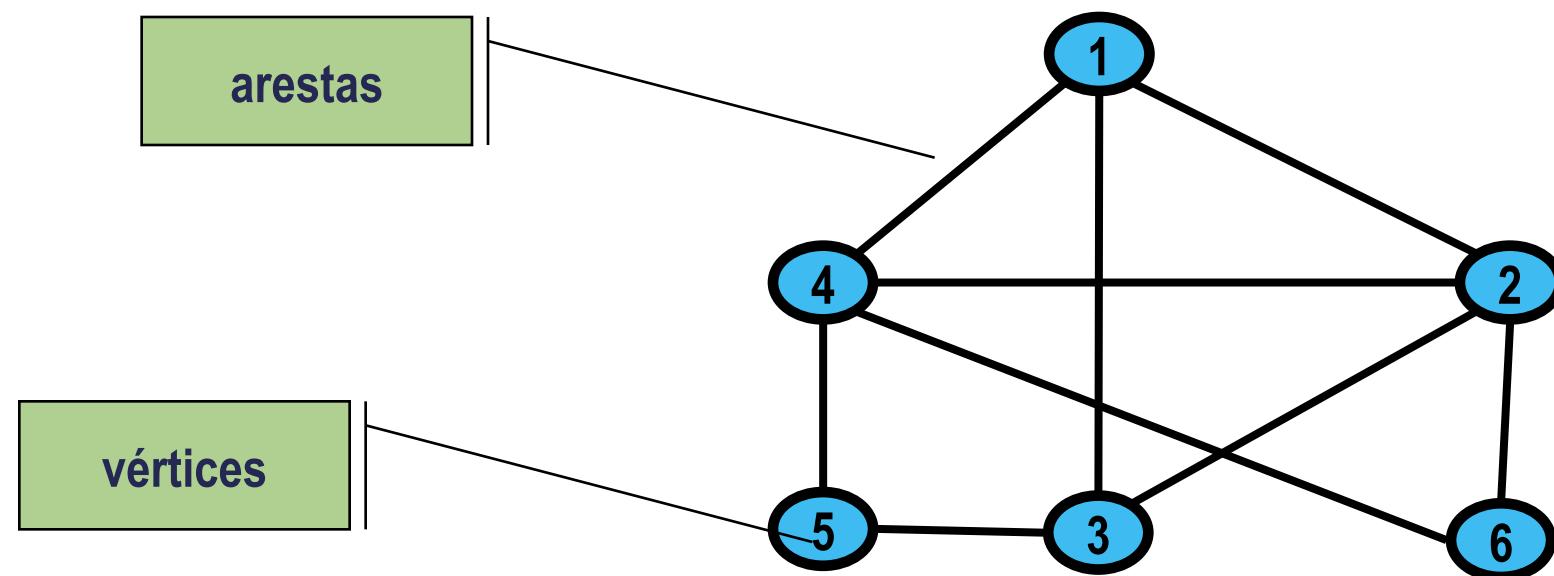
# GRAFOS

Grafos são estruturas de dados formadas por um conjunto de vértices e um conjunto de arestas.

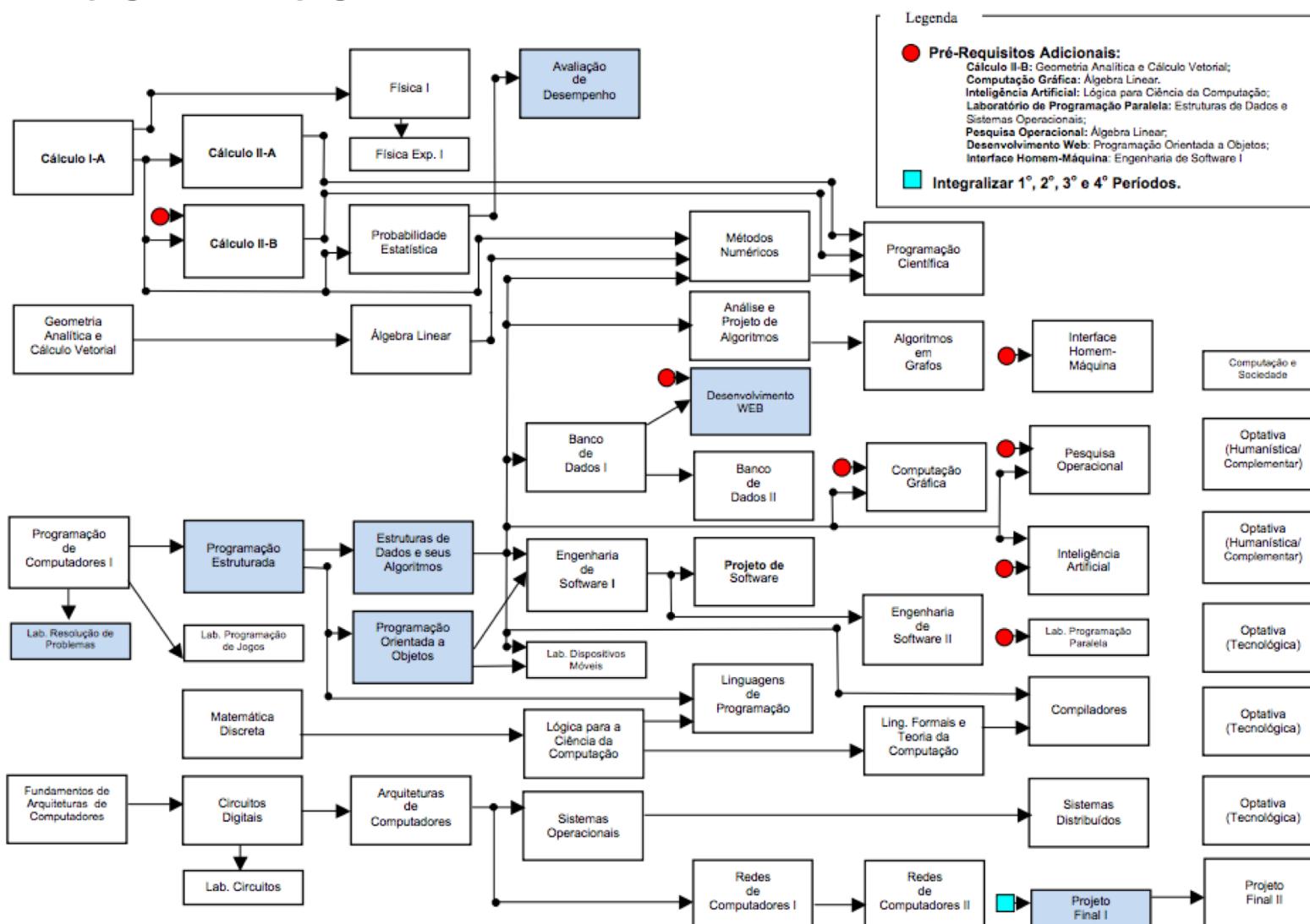


# APLICAÇÕES DE GRAFOS

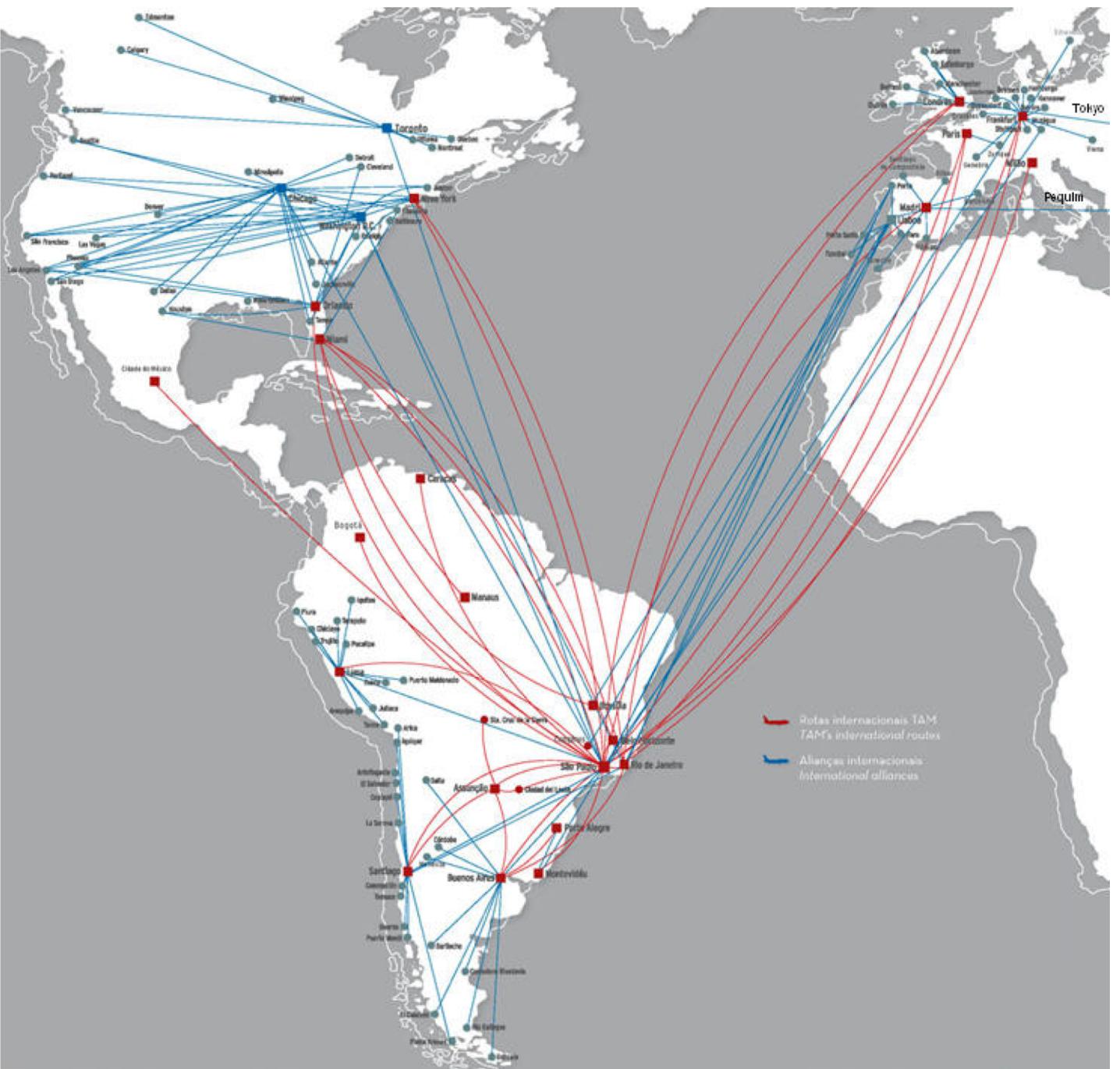
Associando-se significados aos **vértices** e às **arestas**, o grafo passa a constituir um modelo de uma situação ou informação real



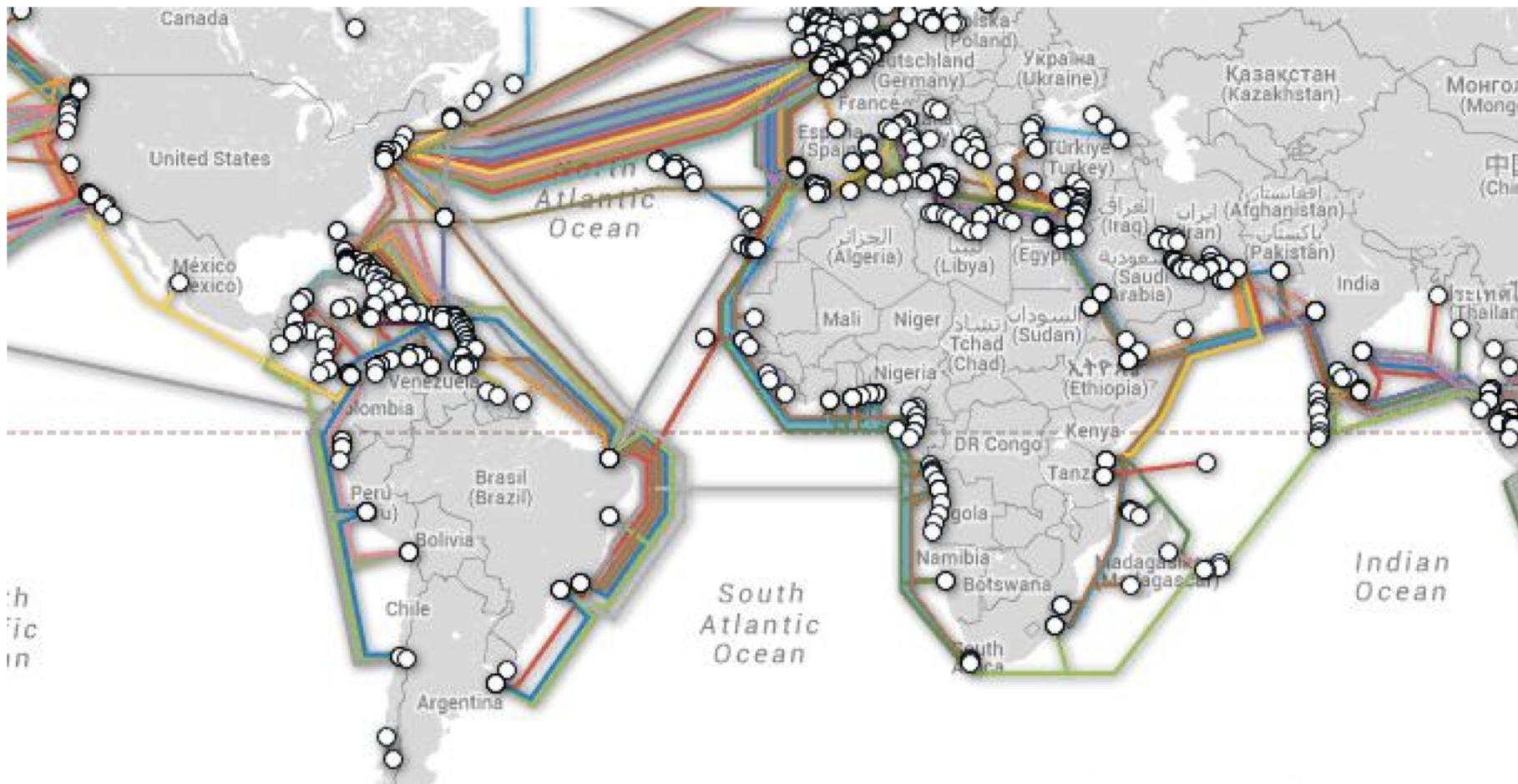
# GRADE CURRICULAR



# ROTAS DE VOOS



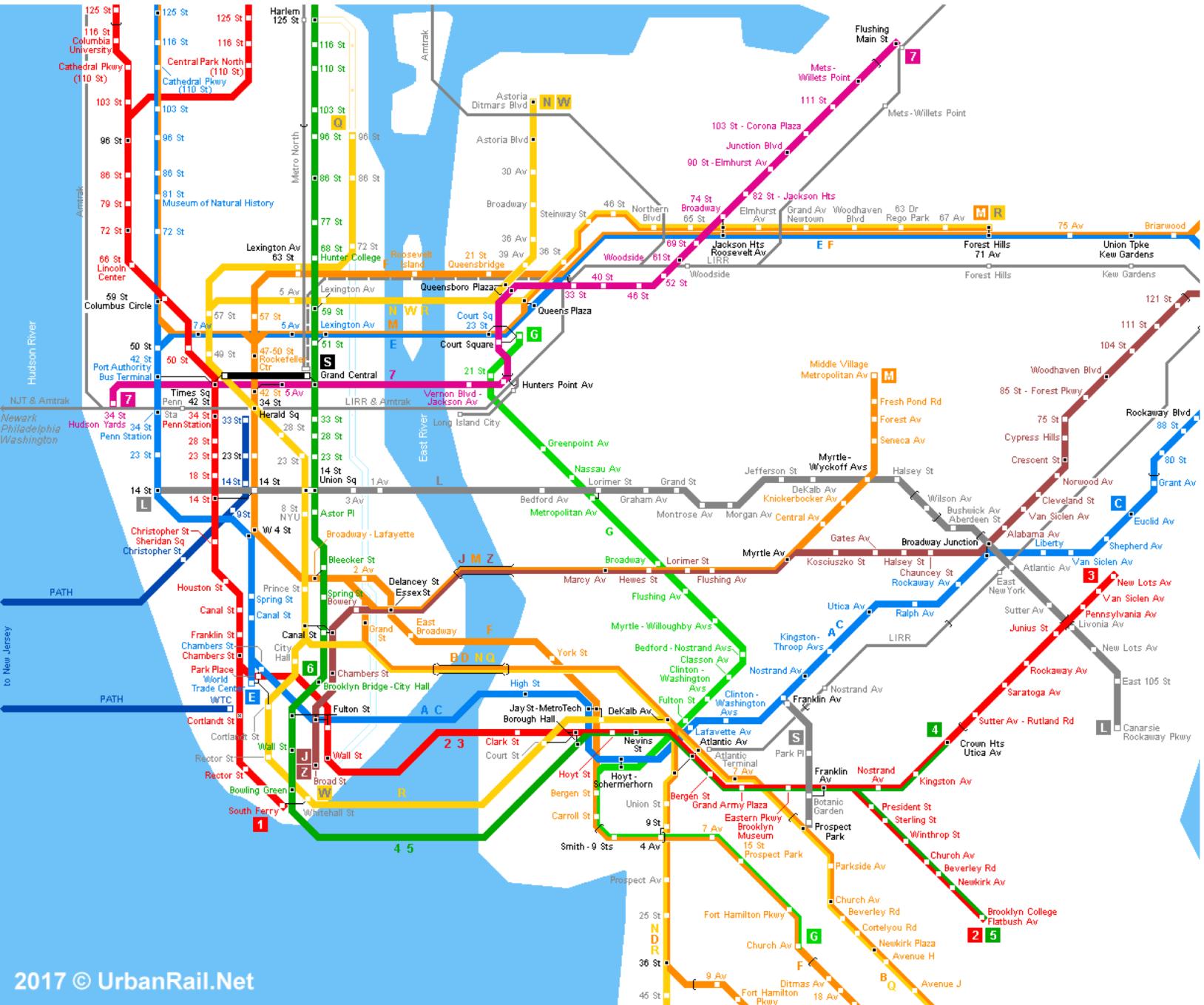
# INTERNET (MAPA DE CABOS SUBMARINOS)



# MAPA DE ESTRADAS



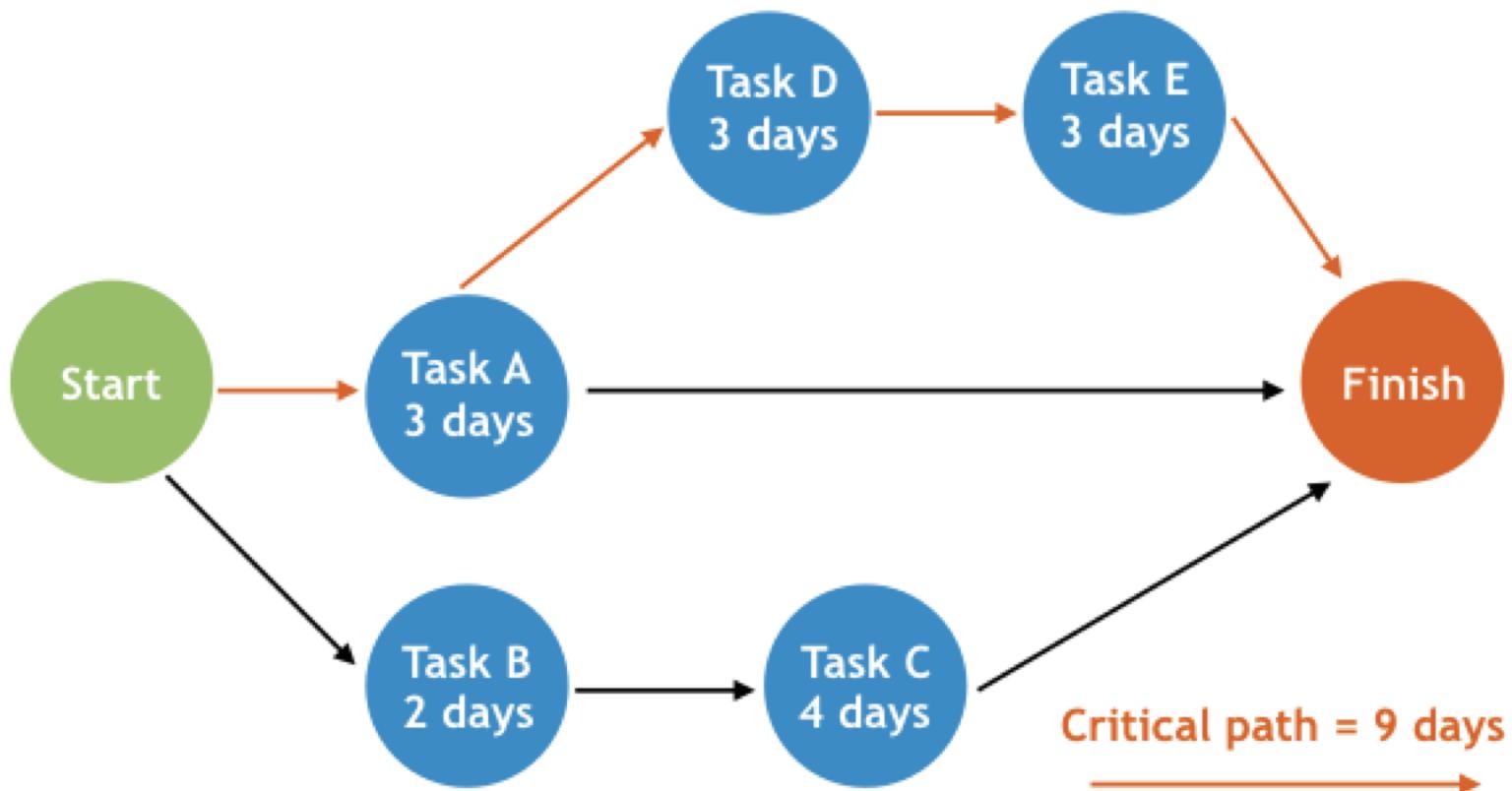
# MAPA DE METRÔ



# REDES SOCIAIS



# PROCESSOS/TAREFAS



# DEPENDÊNCIAS EM SOFTWARE

Cada vértice é um arquivo de um sistema de software

Cada arco é uma dependência: um arquivo  $v$  é construído a partir de todos os arquivos  $w$  para os quais existe um arco da forma  $(v,w)$

- O utilitário `make` do UNIX trabalha sobre grafos deste tipo

# OUTRAS APLICAÇÕES

Cada vértice é uma **página na internet**. Cada arco é um **link** que leva de uma página a outra.

Os vértices são **times de futebol** e os arcos são os **jogos** entre os times durante um campeonato.

Os vértices são as **casas de um tabuleiro de xadrez**. Há um arco de x para y se uma peça do jogo pode ir de x a y em um só **movimento**.

# **GRAFOS – DEFINIÇÕES**

---

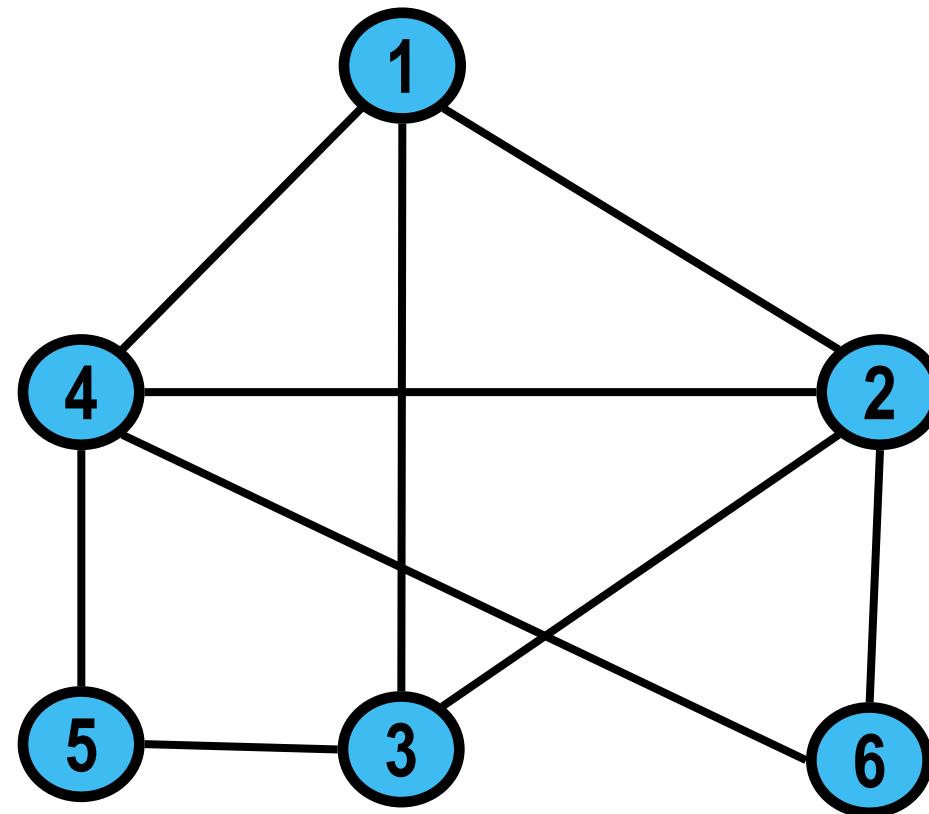
# DEFINIÇÃO FORMAL

Um grafo **G** é representado por um conjunto (não vazio) **V** de vértices e um conjunto (possivelmente vazio) **E** de arestas (edges)

$$G = (V, E)$$

$|V|$  é a quantidade de vértices de G

$|E|$  é a quantidade de arestas de G



# GRAFOS ORIENTADOS X NÃO ORIENTADOS

As arestas podem ter uma direção ou podem ser bi-direcionais

Grafos **orientados**: as arestas possuem uma direção

- Também são chamados de grafos dirigidos ou **digraphos**

Grafos **não orientados**: as arestas são bi-direcionais (se existe uma conexão  $a \rightarrow b$  então também existe uma conexão  $b \rightarrow a$ )

# GRAFO ORIENTADO (DIRIGIDO OU DIGRAFO)

Arestas possuem uma direção

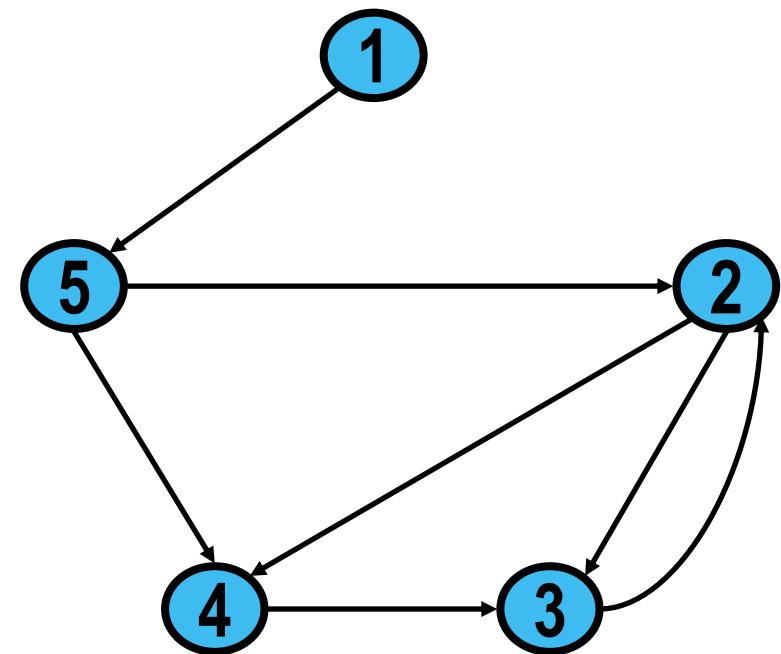
Alguns autores usam o termo **arco** para as arestas de um grafo dirigido

Exemplo:

$$G = (V, E)$$

$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1, 5), (2, 3), (2, 4), (3, 2), (4, 3), (5, 2), (5, 4)\}$$



# GRAFO ORIENTADO (DIRIGIDO OU DIGRAFO)

Arestas possuem uma direção

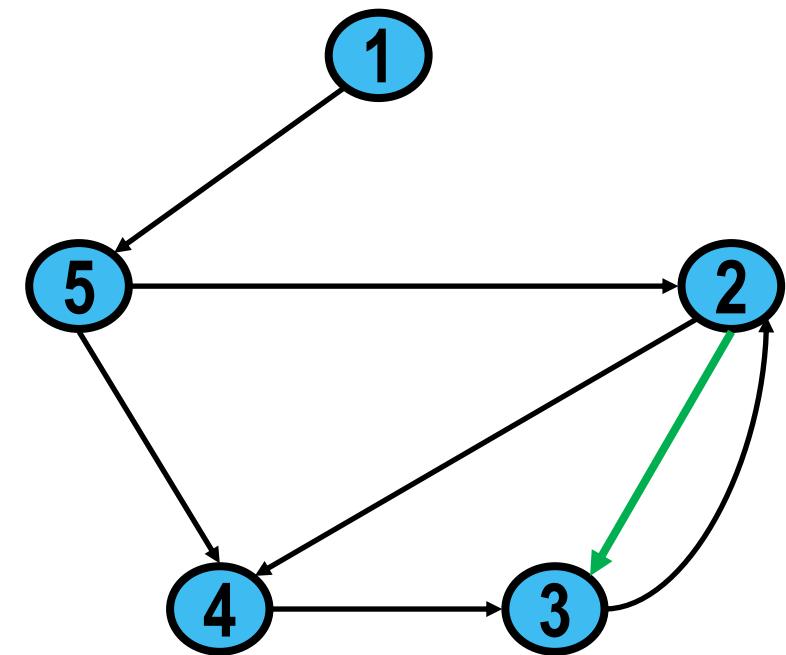
Alguns autores usam o termo **arco** para as arestas de um grafo dirigido

Exemplo:

$$G = (V, E)$$

$$V = \{1, 2, 3, 4, 5\}$$

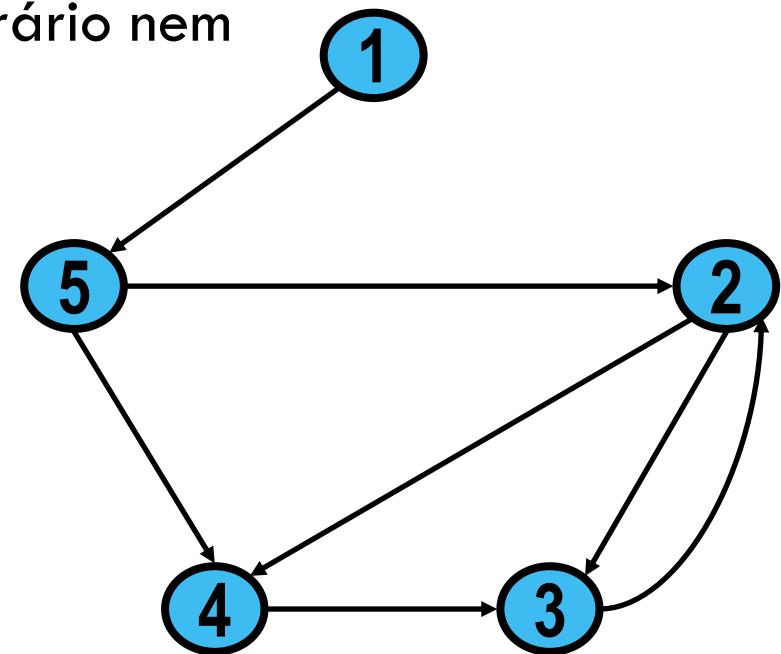
$$E = \{(1, 5), (2, 3), (2, 4), (3, 2), (4, 3), (5, 2), (5, 4)\}$$



# GRAFO ORIENTADO (DIRIGIDO OU DIGRAFO)

Exemplos:

- Malha de transporte urbano
- Pessoa A segue pessoa B no Instagram (mas o contrário nem sempre é verdadeiro)



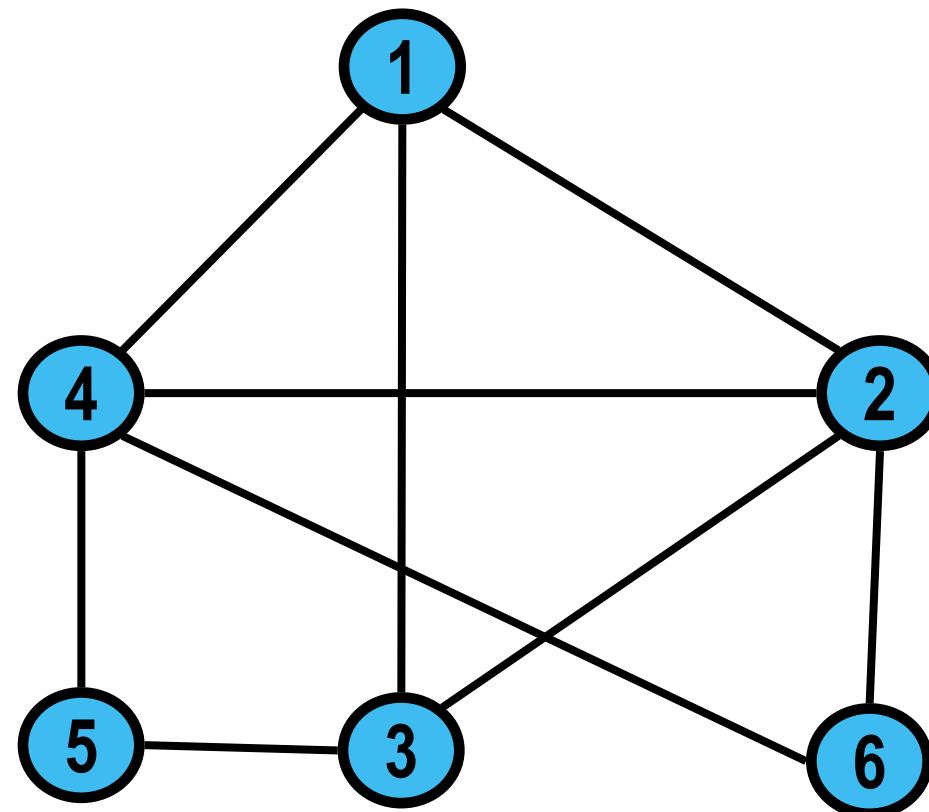
# GRAFO NÃO ORIENTADO

Arestas são bi-direcionais

$$G = (V, E)$$

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{2, 6\}, \{3, 5\}, \{4, 5\}, \{4, 6\}, \{6, 2\}\}$$



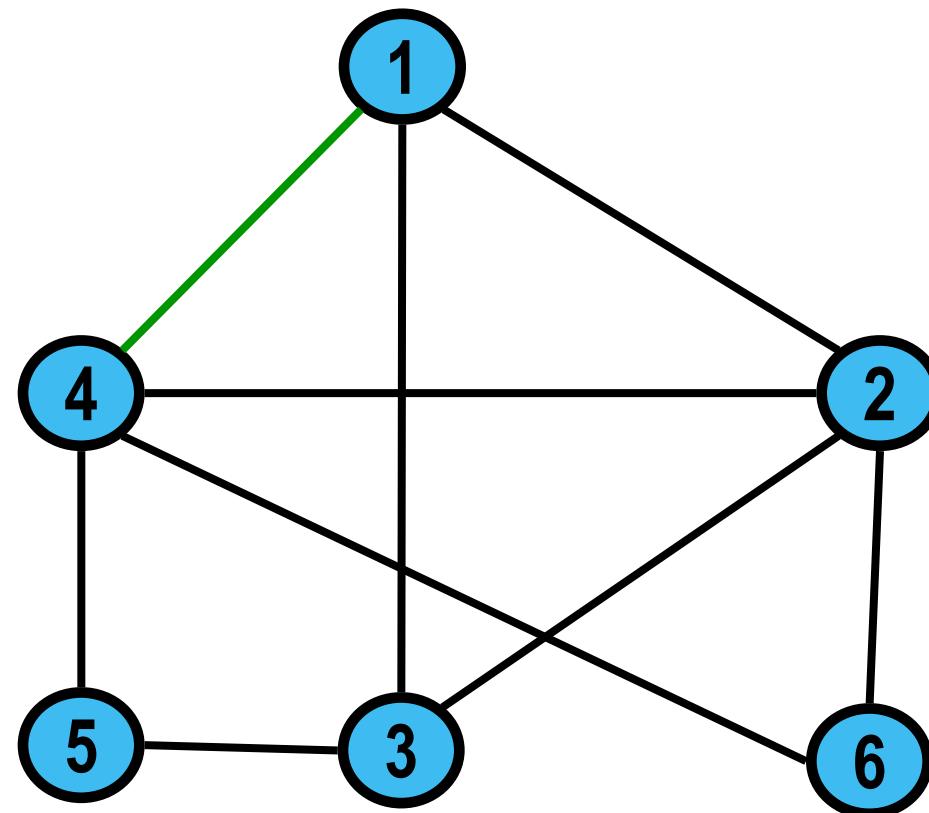
# GRAFO NÃO ORIENTADO

Arestas são bi-direcionais

$$G = (V, E)$$

$$V = \{1, 2, 3, 4, 5, 6\}$$

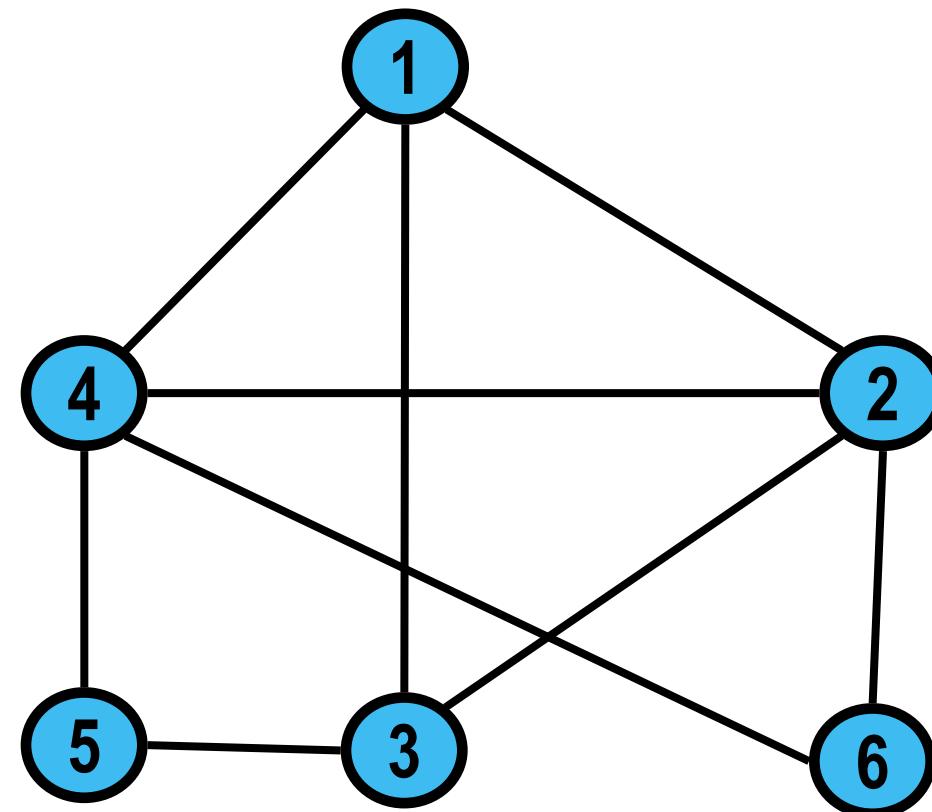
$$E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \\ \{2, 6\}, \{3, 5\}, \{4, 5\}, \{4, 6\}, \{6, 2\}\}$$



# GRAFO NÃO ORIENTADO

## Exemplo:

- Amigos no Facebook (se A é amigo de B, B também é amigo de A)
- Mapa de cabeamento de uma rede (se é possível enviar dados do ponto A ao ponto B, então também é possível enviar do ponto B ao ponto A)



# GRAFOS ORIENTADOS X NÃO ORIENTADOS

Um grafo expressa uma relação binária  $R$

Grafo não orientado

- $\{v_1, v_2\} \in G(E) \Leftrightarrow v_1 R v_2 \wedge v_2 R v_1$
- Exemplo:  $R = \text{amigo no Facebook}$

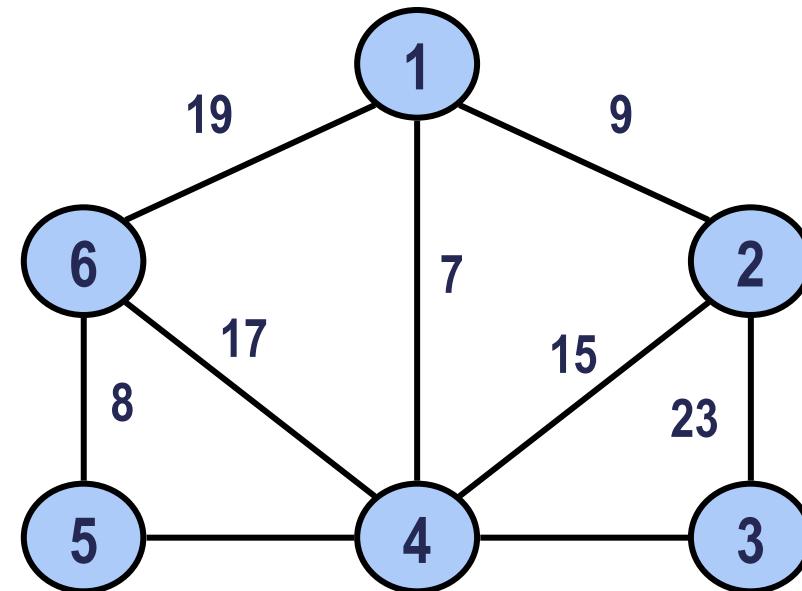
Grafo orientado

- $(v_1, v_2) \in G(E) \Leftrightarrow v_1 R v_2$
- Exemplo:  $R = \text{seguir alguém no Instagram}$

# DEFINIÇÕES E TERMINOLOGIA

Um grafo é **valorado** se possuir valores (**pesos**) associados às arestas e/ou aos vértices

Exemplo: num grafo de rotas de voo, uma aresta pode ser valorada com a **distância** entre os dois aeroportos que ela conecta

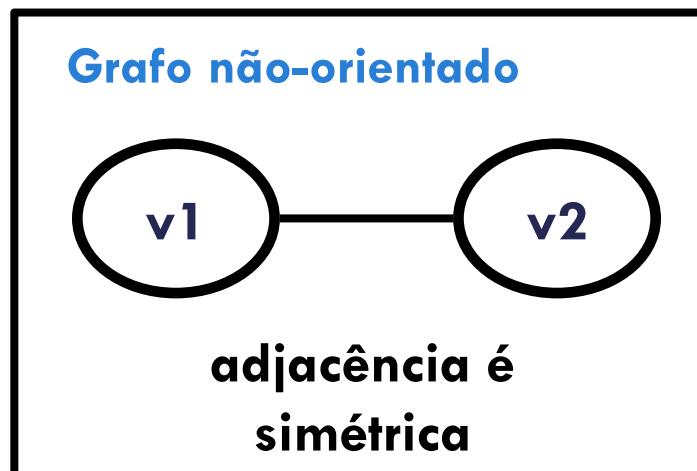


# DEFINIÇÕES E TERMINOLOGIA

Um vértice  $v_1$  é **adjacente** a um vértice  $v_2$  em  $G$ , se existe uma aresta conectando  $v_1$  a  $v_2$  em  $G$ .

Em **grafo não orientado**:  $v_1$  é adjacente a  $v_2$  se existe aresta  $\{v_1, v_2\}$  (nesse caso  $v_2$  também é adjacente a  $v_1$ )

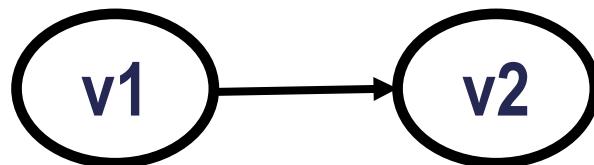
Em **grafo orientado**,  $v_1$  é adjacente a  $v_2$  se existe aresta  $(v_1, v_2)$



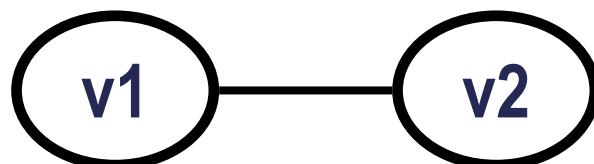
# DEFINIÇÕES E TERMINOLOGIA

Dados dois vértices adjacentes  $v_1$  e  $v_2$

Em um **grafo orientado**, uma aresta  $(v_1, v_2)$  é **incidente de** (sai de)  $v_1$  e é **incidente a** (entra em)  $v_2$

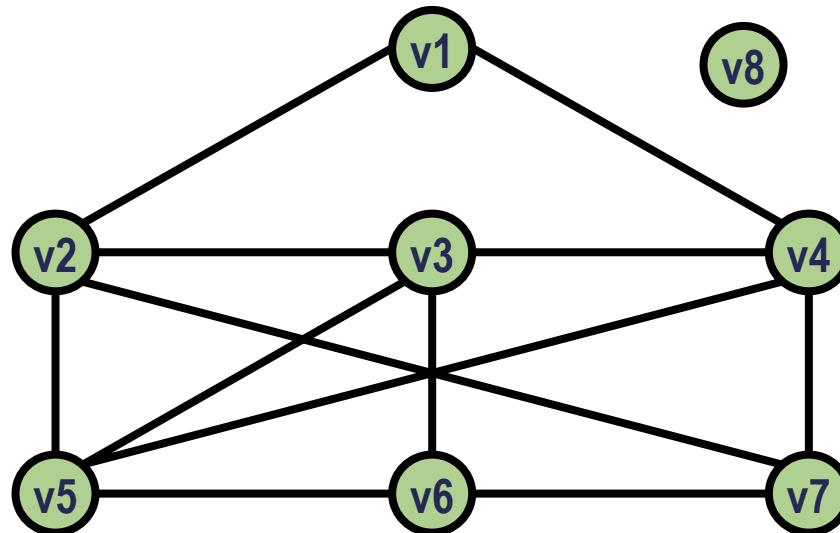


Em um **grafo não orientado**, uma aresta  $\{v_1, v_2\}$  é **incidente em**  $v_1$  e  $v_2$



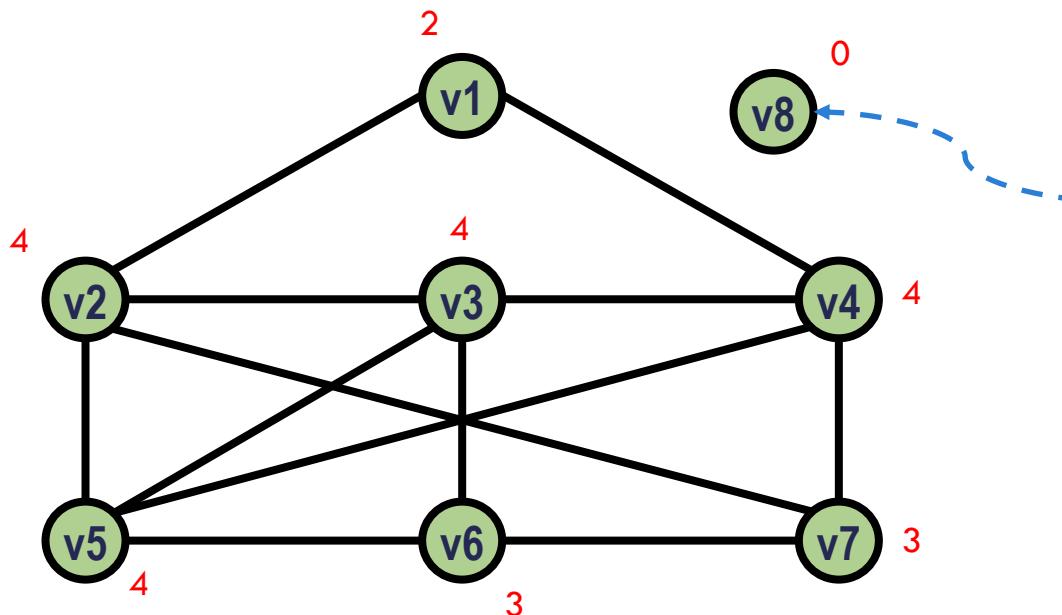
# DEFINIÇÕES E TERMINOLOGIA

O **grau** de um vértice é o número de arestas que nele/dele incidem



# DEFINIÇÕES E TERMINOLOGIA

O **grau** de um vértice é o número de arestas que nele/dele incidem



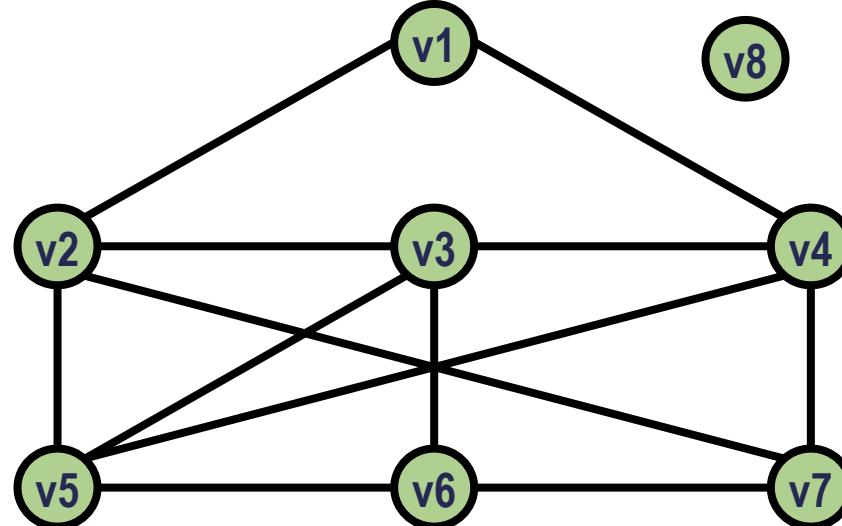
Vértices com **grau 0** são  
ditos vértices **isolados**

# DEFINIÇÕES E TERMINOLOGIA

A **ordem** de um grafo é o número de vértices que ele possui

$$G = (V, E)$$

$$\text{ordem}(G) = |V|$$



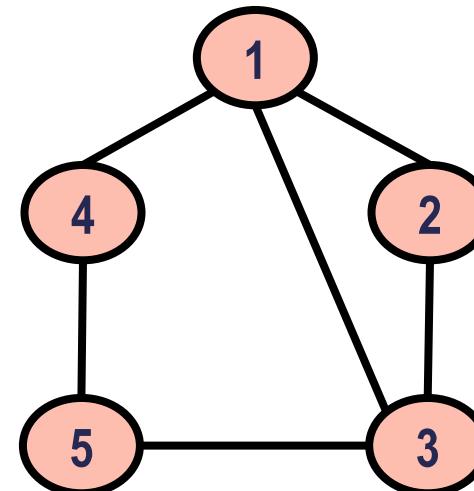
# DEFINIÇÕES E TERMINOLOGIA

O **caminho de tamanho k** entre dois vértices **v1** e **v2** é a sequencia  $\langle v_0, v_1, \dots, v_k \rangle$ , onde  $v_0 = v1$ ,  $v_k = v2$ , e  $(v_{i-1}, v_i) \in G(E)$  (ou  $\{v_{i-1}, v_i\} \in G(E)$ ) para  $i=1..k$

O caminho contém os **vértices**  $v_0, v_1, \dots, v_k$ , e as **arestas**  $(v_0, v_1), (v_1, v_2), \dots (v_{k-1}, v_k)$  (ou  $\{v_0, v_1\}, \{v_1, v_2\}, \dots \{v_{k-1}, v_k\}$  para grafos não orientados)

Exemplo:

- Caminho de tamanho 1 entre os vértices 1 e 3:
  - Vértices: {1, 3}
  - Arestas: {{1, 3}}
- Caminho de tamanho 2 entre os vértices 1 e 3:
  - Vértices: {1, 2, 3}
  - Arestas: {{1, 2}, {2, 3}}



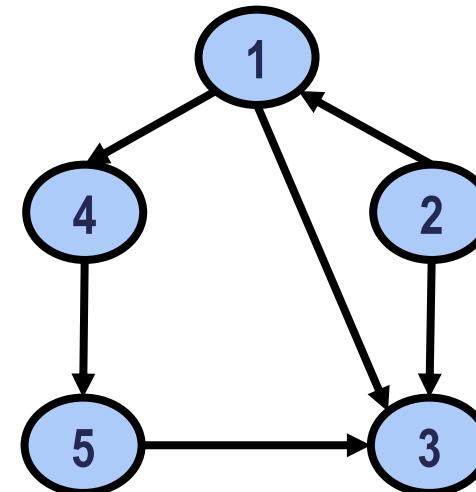
# DEFINIÇÕES E TERMINOLOGIA

Se existe um caminho entre  $v_1$  e  $v_2$ , diz-se que  $v_2$  é **alcançável** a partir de  $v_1$

O caminho é **simples** se todos os vértices no caminho são distintos

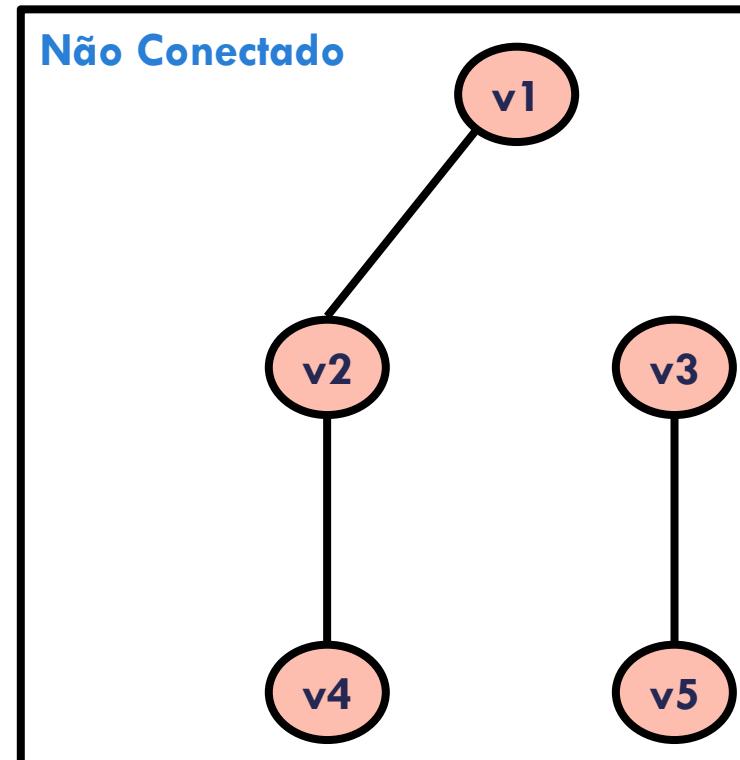
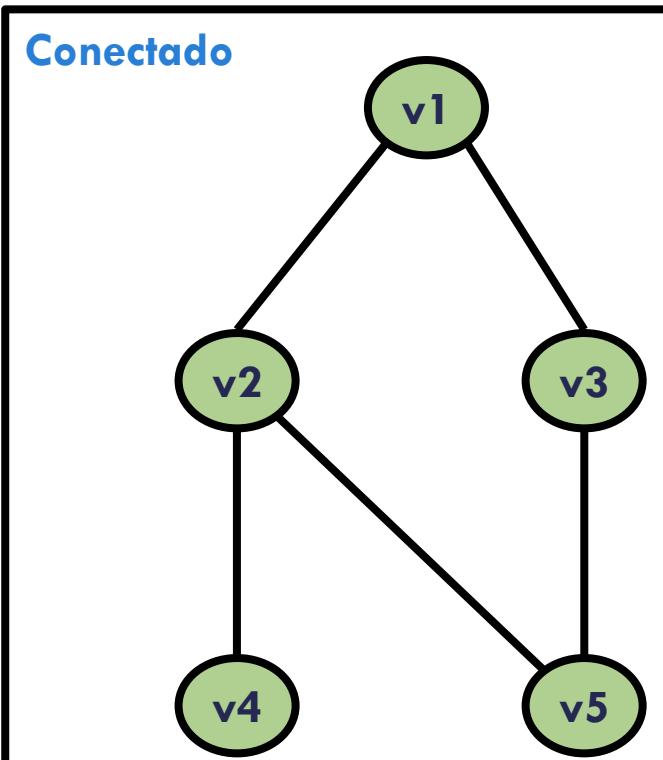
Exemplo:

- 3 é alcançável a partir de 4
- 2 não é alcançável a partir de 1



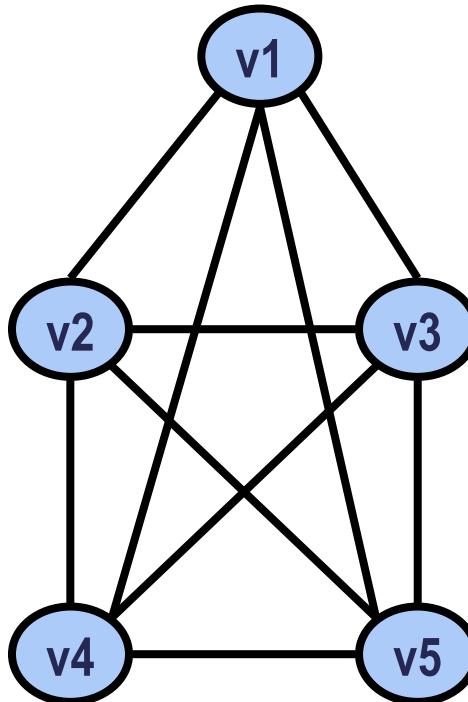
# DEFINIÇÕES E TERMINOLOGIA

Um grafo é dito **conectado** se existe um caminho ligando cada par de vértices



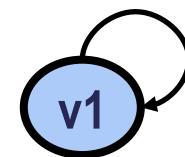
# DEFINIÇÕES E TERMINOLOGIA

Um grafo é dito **completo** se todos os seus pares de vértices forem adjacentes



# DEFINIÇÕES E TERMINOLOGIA

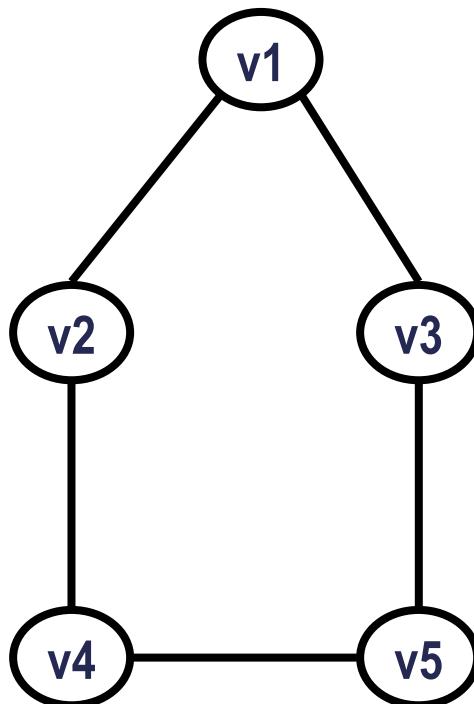
Uma aresta que tem ambas as extremidades em um mesmo vértice é chamada **laço**



# DEFINIÇÕES E TERMINOLOGIA

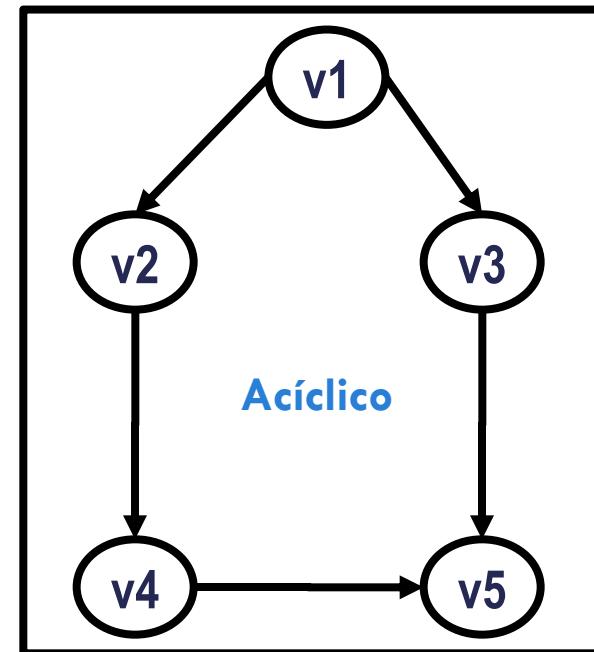
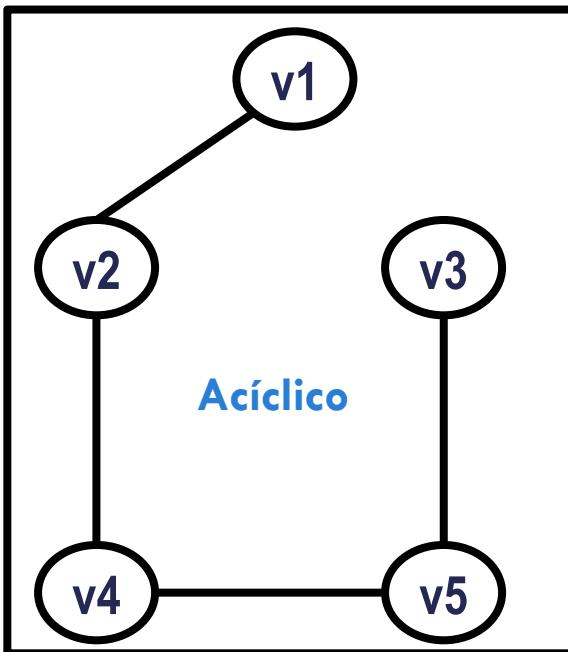
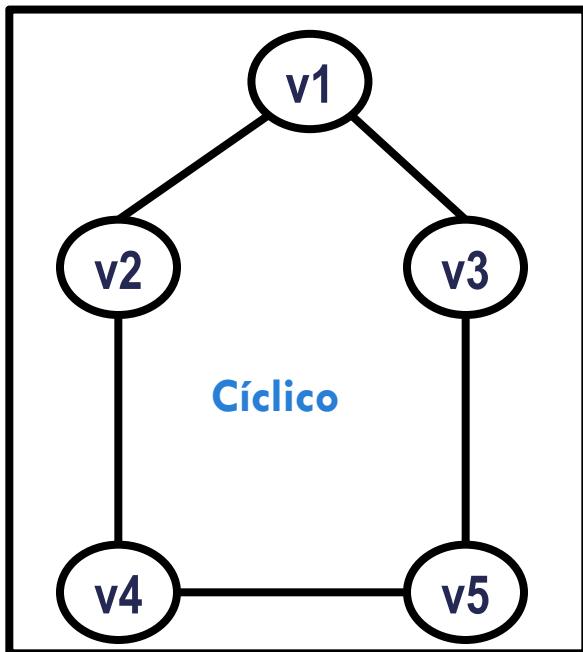
Um caminho  $\langle v_0, v_1, \dots, v_k \rangle$  forma um **ciclo** se  $v_0 = v_k$  e o caminho contém pelo menos uma aresta

Exemplo: caminho de  $v1$  a  $v1$



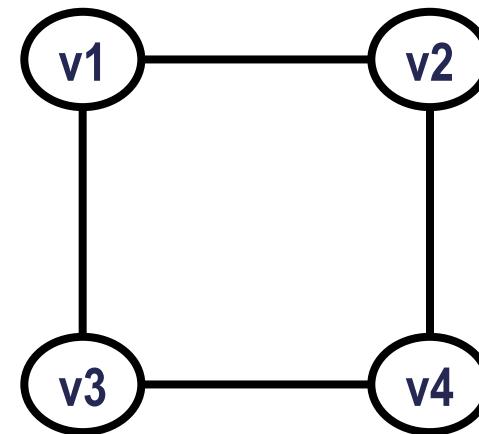
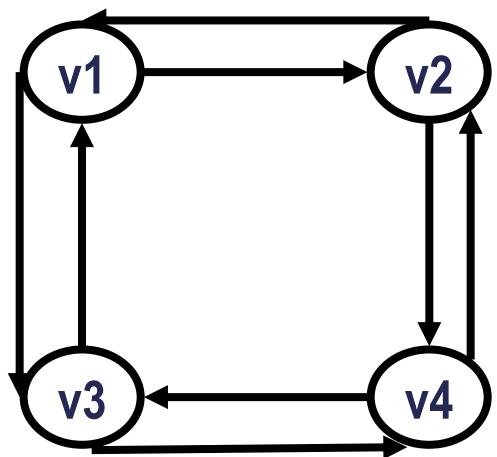
# DEFINIÇÕES E TERMINOLOGIA

Um grafo sem ciclos é dito **acíclico**



# DEFINIÇÕES E TERMINOLOGIA

Um grafo é dito **simétrico** se para cada aresta  $(v, w)$  existe uma aresta  $(w, v)$



# GRAFOS – REPRESENTAÇÕES

# REPRESENTAÇÃO FÍSICA DE GRAFOS

Matriz de adjacência

Matriz de incidência

Lista de adjacência

Lista de incidência

# REPRESENTAÇÃO FÍSICA DE GRAFOS

**Matriz de adjacência**

Matriz de incidência

Lista de adjacência

Lista de incidência

# MATRIZ DE ADJACÊNCIA

**Matriz de adjacência**  $A(n \times n)$  de um grafo  $G$  de **ordem  $n$** , é uma matriz onde cada elemento  $a_{i,j}$  é:

- **Grafos orientados:**

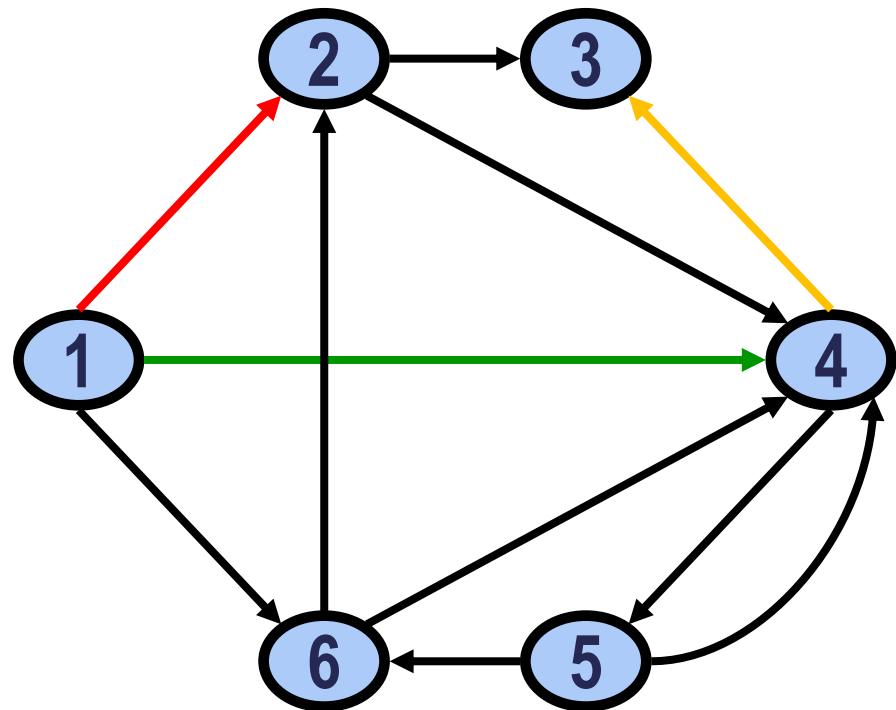
- $a_{i,j} = 1$  se  $(v_i, v_j) \in G(E)$
- $a_{i,j} = 0$  se  $(v_i, v_j) \notin G(E)$

- **Grafos não orientados:**  $a_{i,j} = a_{j,i}$

- $a_{i,j} = 1$  se  $\{v_i, v_j\} \in G(E)$
- $a_{i,j} = 0$  se  $\{v_i, v_j\} \notin G(E)$

# MATRIZ DE ADJACÊNCIA

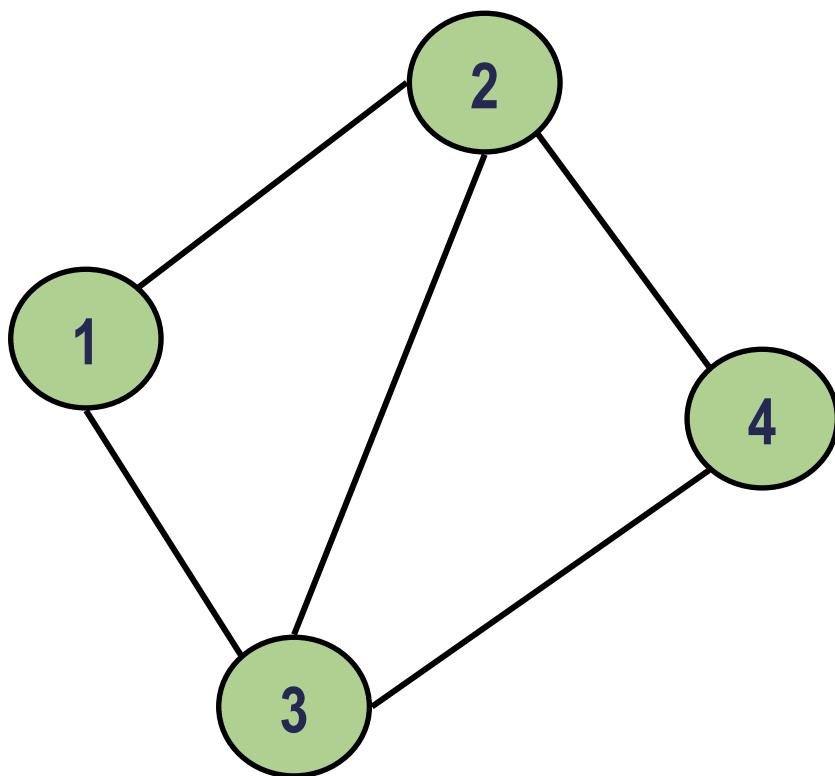
A matriz de adjacência é uma forma de representação de grafos simples, econômica e adequada para muitos problemas que envolvem apenas a estrutura do grafo



vértices

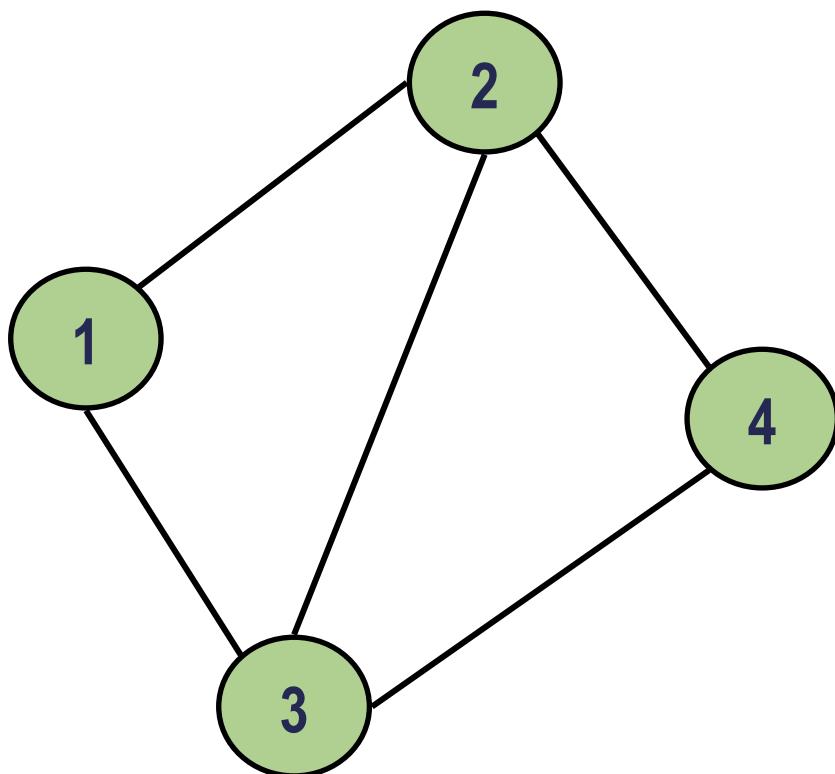
		1	2	3	4	5	6
vértices	1	0	1	0	1	0	1
	2	0	0	1	1	0	0
	3	0	0	0	0	0	0
	4	0	0	1	0	1	0
	5	0	0	0	1	0	1
	6	0	1	0	1	0	0

# MATRIZ DE ADJACÊNCIA PARA GRAFO NÃO ORIENTADO



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

# MATRIZ DE ADJACÊNCIA PARA GRAFO NÃO ORIENTADO



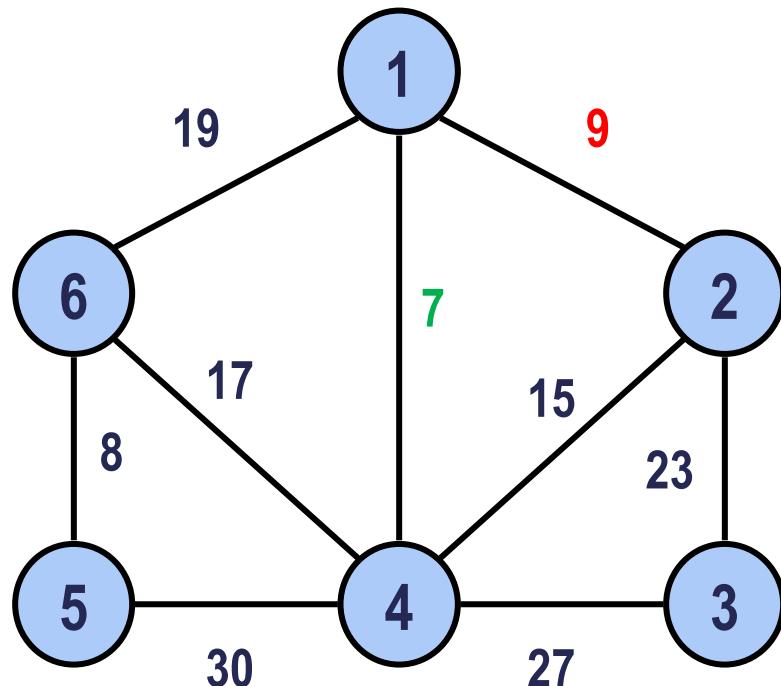
	1	2	3	4
1	0	1	1	0
2		0	1	1
3			0	1
4				0

Matriz é simétrica

# MATRIZ DE ADJACÊNCIA PARA GRAFOS VALORADOS

Valores associados às linhas podem ser representados por uma extensão simples da Matriz de Adjacência

- $a_{ij} = k$  se  $(v_i, v_j) \in G(E)$
- $a_{ij} = *$  se  $(v_i, v_j) \notin G(E)$



	1	2	3	4	5	6
1	*	9	*	7	*	19
2	9	*	23	15	*	*
3	*	23	*	27	*	*
4	7	15	27	*	30	17
5	*	*	*	30	*	8
6	19	*	*	17	8	*

# MATRIZ DE ADJACÊNCIA

Matriz binária: ocupa pouco espaço, especialmente para grafos grandes

Manipulação simples: recursos para manipular matrizes existem em qualquer linguagem de programação

Fácil determinar se  $(v_i, v_j) \in G(E)$

Fácil determinar vértices adjacentes a um determinado vértice  $v$

Quando o grafo é não orientado, a MA é simétrica (mais econômica)

Inserção de novas arestas é fácil

Inserção de novos vértices é **muito difícil**

# REPRESENTAÇÃO FÍSICA DE GRAFOS

Matriz de adjacência

**Matriz de incidência**

Lista de adjacência

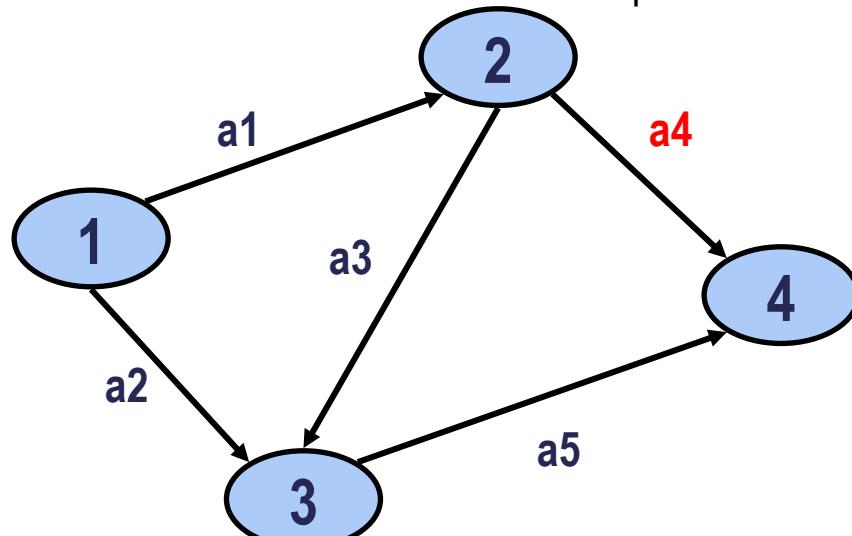
Lista de incidência

# MATRIZ DE INCIDÊNCIA

É uma matriz  $B(n \times m)$ , sendo  $n$  o número de vértices,  $m$  o número de arestas e:

- $b_{ij} = -1$  se o vértice  $i$  é a origem da aresta  $j$
- $b_{ij} = 1$  se o vértice  $i$  é o término da aresta  $j$
- $b_{ij} = 0$  se aresta  $(i,j) \notin G(E)$

Para grafos não orientados,  $b_{ij} = 1$  se a aresta  $j$  é incidente ao vértice  $i$ .



arestas					
	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
vértices	1	2	3	4	
1	-1	-1	0	0	0
2	1	0	-1	-1	0
3	0	1	1	0	-1
4	0	0	0	1	1

# REPRESENTAÇÃO FÍSICA DE GRAFOS

Matriz de adjacência

Matriz de incidência

**Lista de adjacência**

Listas de incidência

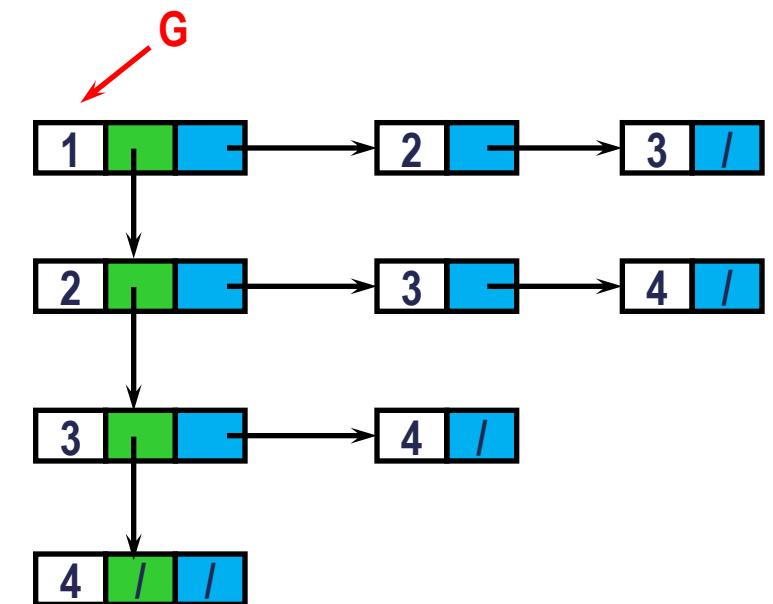
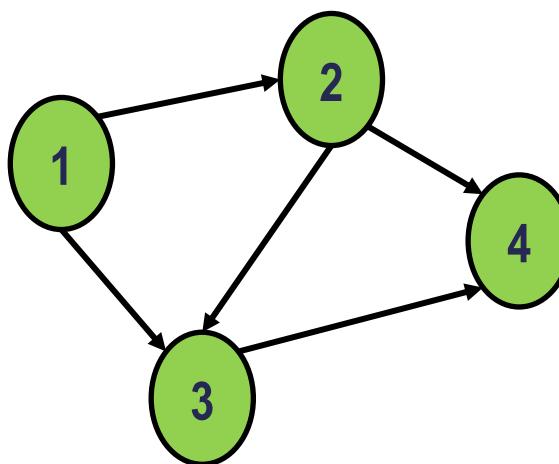
# LISTA DE ADJACÊNCIA

Para cada vértice  $v$  é representada a lista de vértices  $u$  tais que  $(v,u) \in G(E)$

Possíveis formas de armazenamento: vetores, vetores + listas encadeadas, listas encadeadas

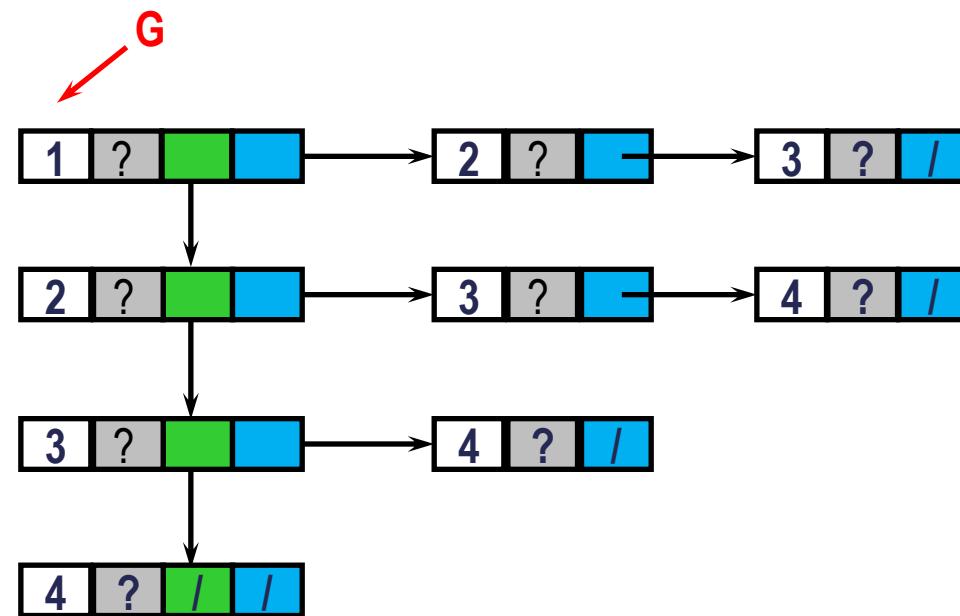
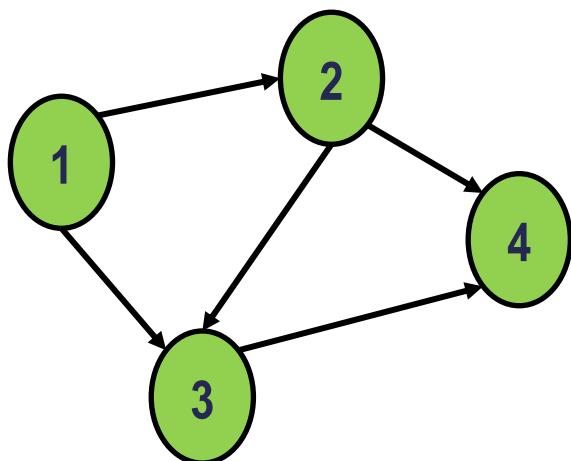
Melhor forma de representação: **listas encadeadas**

- Uso racional do espaço
- Flexibilidade



# LISTAS DE ADJACÊNCIA

Nós podemos estender para representar outras informações



# REPRESENTAÇÃO FÍSICA DE GRAFOS

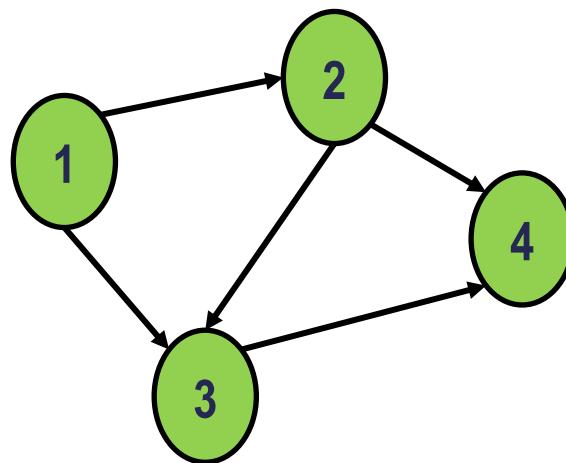
Matriz de adjacência

Matriz de incidência

Lista de adjacência

**Lista de incidência**

# LISTAS DE INCIDÊNCIA



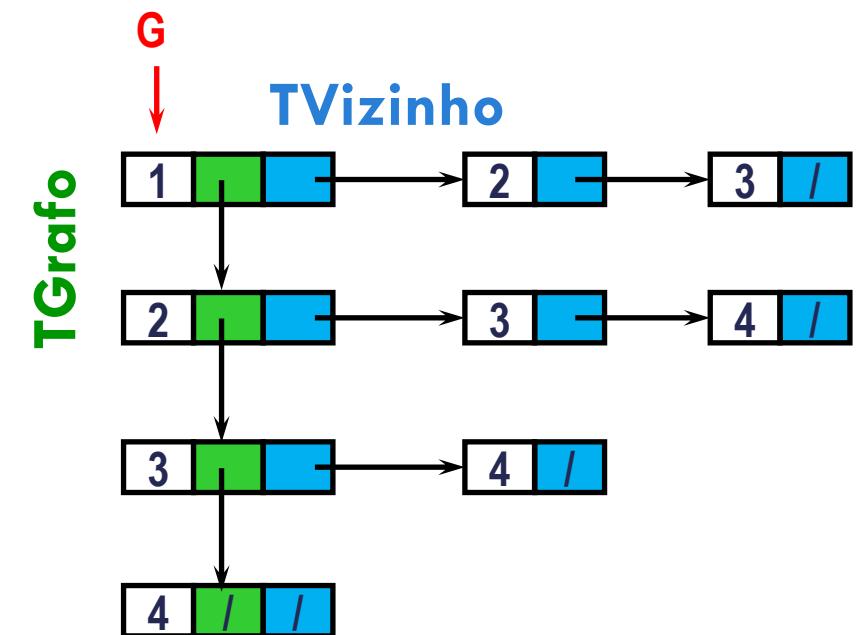
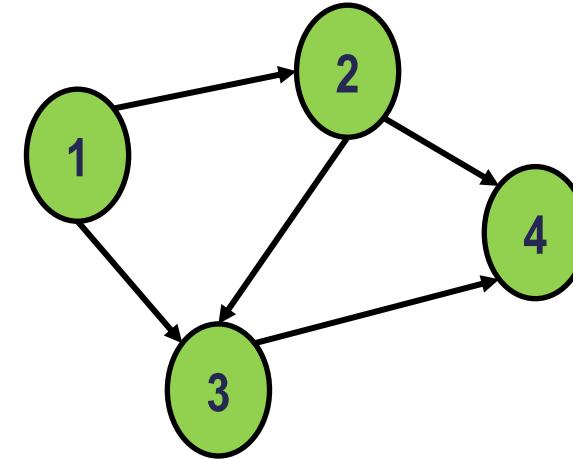
# IMPLEMENTAÇÃO

Veremos a implementação de grafos usando **lista de adjacência**

- São flexíveis para acomodar inserções e remoções, ao contrário das matrizes de adjacência e incidência
- Facilitam a identificação dos vértices do grafo, ao contrário das listas de incidência

# ESTRUTURA EM C

```
typedef struct vizinho {  
    int id_vizinho;  
    struct vizinho *prox;  
} TVizinho;  
  
typedef struct grafo{  
    int id_vertice;  
    TVizinho *prim_vizinho;  
    struct grafo *prox;  
} TGrafo;
```



# INICIALIZAÇÃO DA ESTRUTURA

```
TGrafo *inicializa() {  
    return NULL;  
}
```

# IMPRESSÃO DO GRAFO

```
void imprime(TGrafo *g) {
    while(g != NULL) {
        printf("Vértice %d\n", g->id_vertice);
        printf("Vizinhos: ");
        TVizinho *v = g->prim_vizinho;
        while(v != NULL) {
            printf("%d ", v->id_vizinho);
            v = v->prox;
        }
        printf("\n\n");
        g = g->prox;
    }
}
```

# IMPRESSÃO DO GRAFO (IMPLEMENTAÇÃO RECURSIVA)

```
void imprime_recursivo(TGrafo *g) {
    if(g != NULL) {
        printf("Vértice: %d:\n", g->id_vertice);
        printf("Vizinhos: ");
        TVizinho *v = g->prim_vizinho;
        while(v) {
            printf("%d ", v->id_vizinho);
            v = v->prox;
        }
        printf("\n\n");
        imprime_recursivo(g->prox);
    }
}
```

# LIBERAÇÃO DA ESTRUTURA

```
void libera(TGrafo *g) {
    while(g != NULL) {
        libera_vizinhos(g->prim_vizinho);
        TGrafo *temp = g;
        g = g->prox;
        free(temp);
    }
}

void libera_vizinhos(TVizinho *v) {
    while(v != NULL) {
        TVizinho *temp = v;
        v = v->prox;
        free(temp);
    }
}
```

# LIBERAÇÃO DA ESTRUTURA (IMPLEMENTAÇÃO RECURSIVA)

```
void libera_recursivo(TGrafo *g) {
    if(g != NULL) {
        libera_vizinho_recursivo(g->prim_vizinho);
        libera_recursivo(g->prox);
        free(g);
    }
}

void libera_vizinho_recursivo(TVizinho *v) {
    if(v != NULL) {
        libera_vizinho_recursivo(v->prox);
        free(v);
    }
}
```

# GRAFOS – ALGORITMOS BÁSICOS

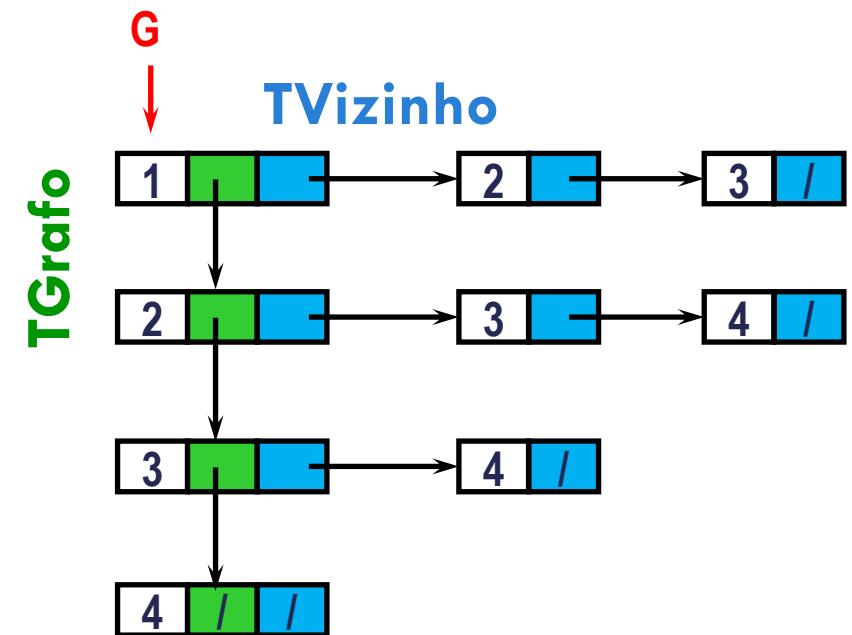
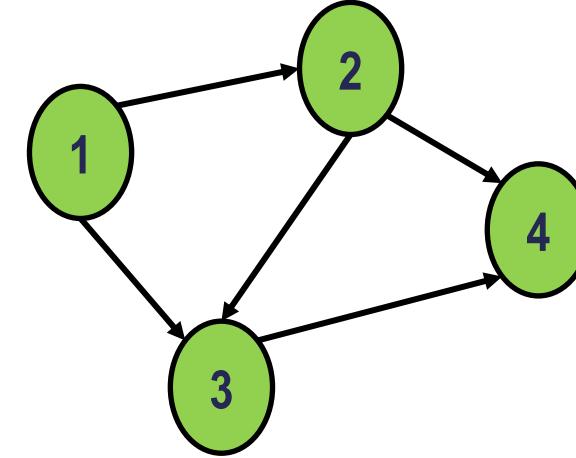
# BUSCA

## Busca por um vértice $v_1$

- Basta percorrer a lista de vértices até encontrar  $v_1$

## Busca por uma aresta $(v_1, v_2)$

- Percorrer a lista de vértices até encontrar  $v_1$
- Depois percorrer a lista de vizinhos de  $v_1$  até encontrar  $v_2$



# BUSCA POR VÉRTICE X

```
TGrafo* busca_vertice(TGrafo* g, int x) {
    while((g != NULL) && (g->id_vertice != x)) {
        g = g->prox;
    }
    return g;
}
```

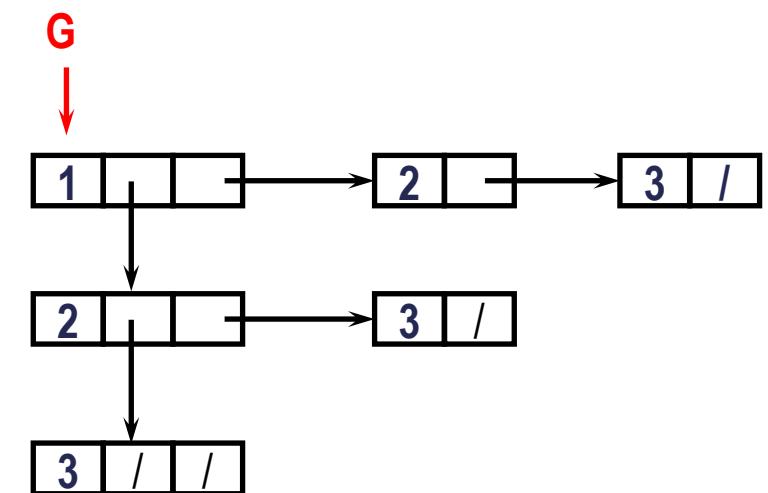
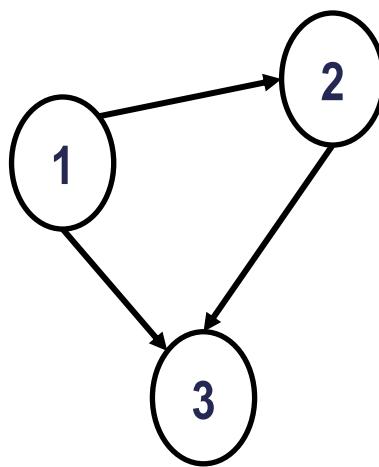
# BUSCA POR ARESTA ( $v_1, v_2$ ) OU $\{v_1, v_2\}$

```
TVizinho* busca_aresta(TGrafo *g, int v1, int v2) {
    TGrafo *pv1 = busca_vertice(g, v1);
    TGrafo *pv2 = busca_vertice(g, v2);
    TVizinho *resp = NULL;
    //checa se ambos os vértices existem
    if((pv1 != NULL) && (pv2 != NULL)) {
        //percorre a lista de vizinhos de v1 procurando por v2
        resp = pv1->prim_vizinho;
        while ((resp != NULL) && (resp->id_vizinho != v2)) {
            resp = resp->prox;
        }
    }
    return resp;
}
```

# INSERÇÃO DE VÉRTICE

Insere o vértice na lista encadeada de vértices, como **primeiro vértice da lista**

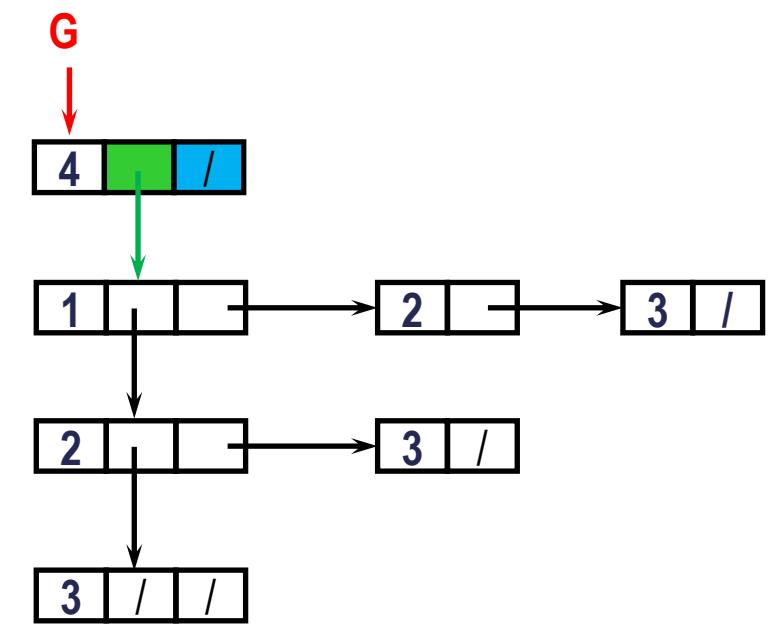
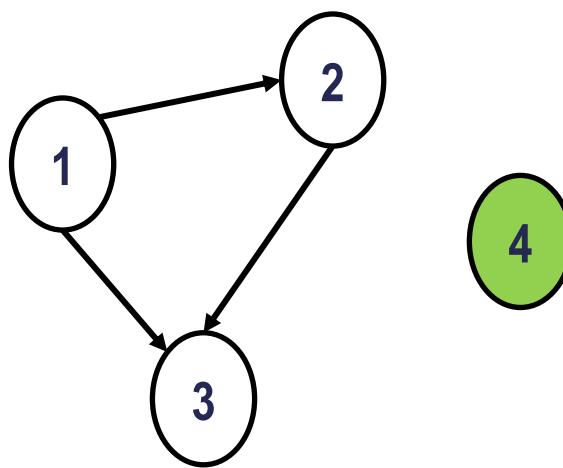
Exemplo: inserir vértice 4



# INSERÇÃO DE VÉRTICE

Insere o vértice na lista encadeada de vértices, como primeiro vértice da lista

Exemplo: inserir vértice 4



# INSERÇÃO DE VÉRTICE

```
TGrafo *insere_vertice(TGrafo *g, int x) {
    TGrafo *p = busca_vertice(g, x);
    if(p == NULL) {
        p = (TGrafo*) malloc(sizeof(TGrafo));
        p->id_vertice = x;
        p->prox = g;
        p->prim_vizinho = NULL;
        g = p;
    }
    return g;
}
```

# INSERÇÃO DE ARESTA

## Grafo não orientado (digrafo)

- **Inserção de aresta  $\{v_1, v_2\}$ :** inserir  $v_2$  na lista de vizinhos de  $v_1$ , e  $v_1$  na lista de vizinhos de  $v_2$  (ou seja, inserir as arestas  $(v_1, v_2)$  e  $(v_2, v_1)$ )

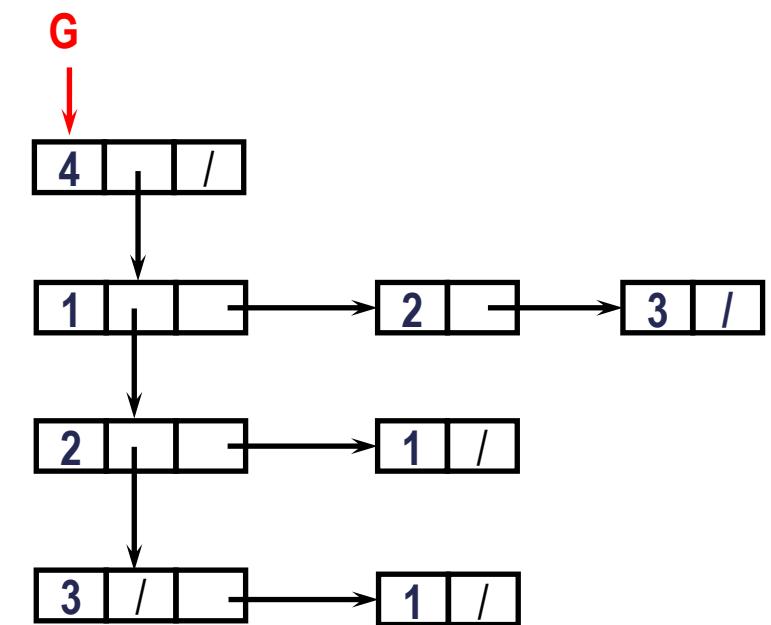
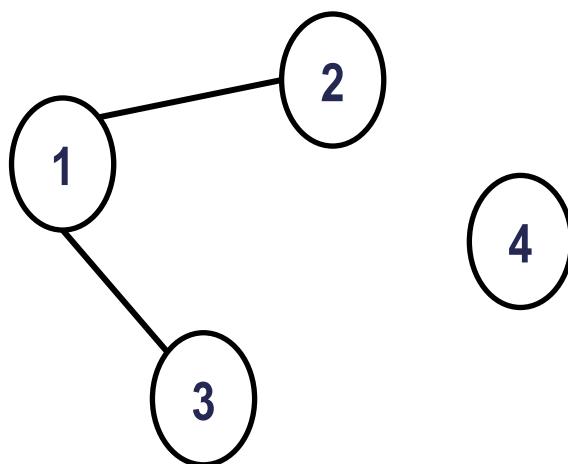
## Grafo orientado

- **Inserção de aresta  $(v_1, v_2)$ :** inserir  $v_2$  na lista de vizinhos de  $v_1$

Em ambos os casos, verificar se a aresta já existe antes de realizar a inserção

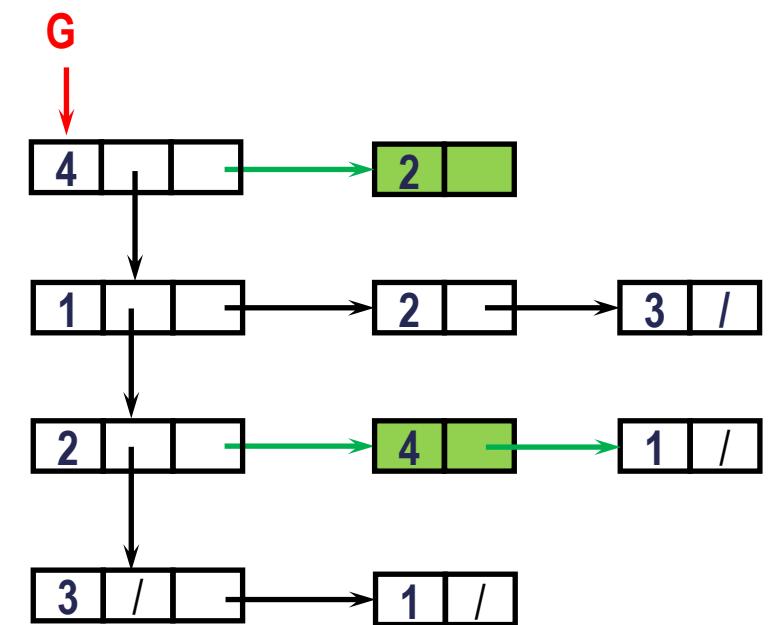
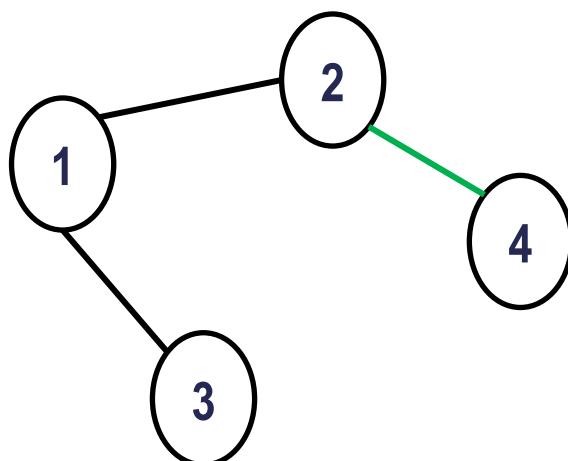
# INSERÇÃO DE ARESTA EM GRAFO NÃO ORIENTADO

Exemplo: Inserir aresta {2, 4}



# INSCRIÇÃO DE ARESTA EM GRAFO NÃO ORIENTADO

Exemplo: Inserir aresta {2, 4}



# INSERÇÃO DE ARESTA EM GRAFO NÃO ORIENTADO

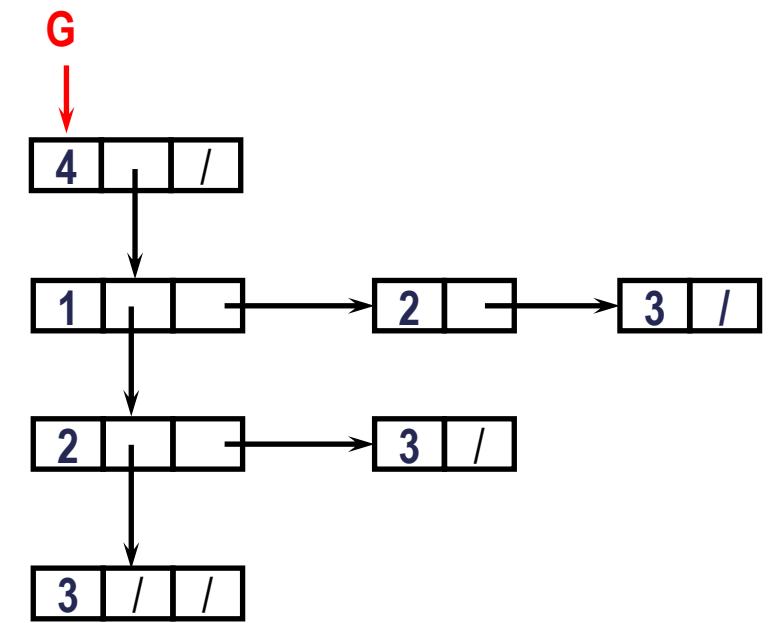
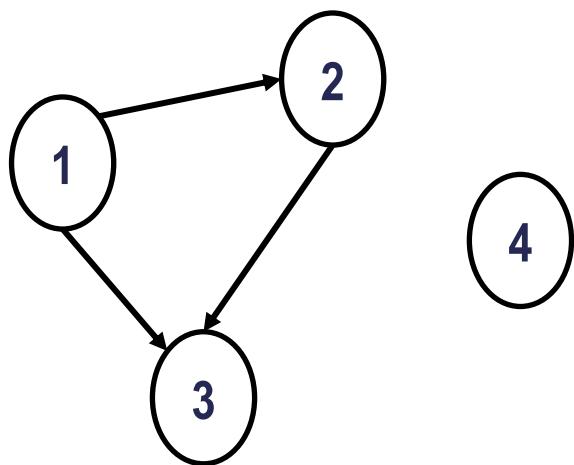
```
void insere_um_sentido(TGrafo *g, int v1, int v2) {
    TGrafo *p = busca_vertice(g, v1);
    TVizinho *nova = (TVizinho *) malloc(sizeof(TVizinho));
    nova->id_vizinho = v2;
    nova->prox = p->prim_vizinho;
    p->prim_vizinho = nova;
}

void insere_aresta(TGrafo *g, int v1, int v2) {
    TVizinho *v = busca_aresta(g, v1, v2);
    if(v == NULL)
        insere_um_sentido(g, v1, v2);
        insere_um_sentido(g, v2, v1);
}
```

Se grafo é não orientado,  
usar essa função

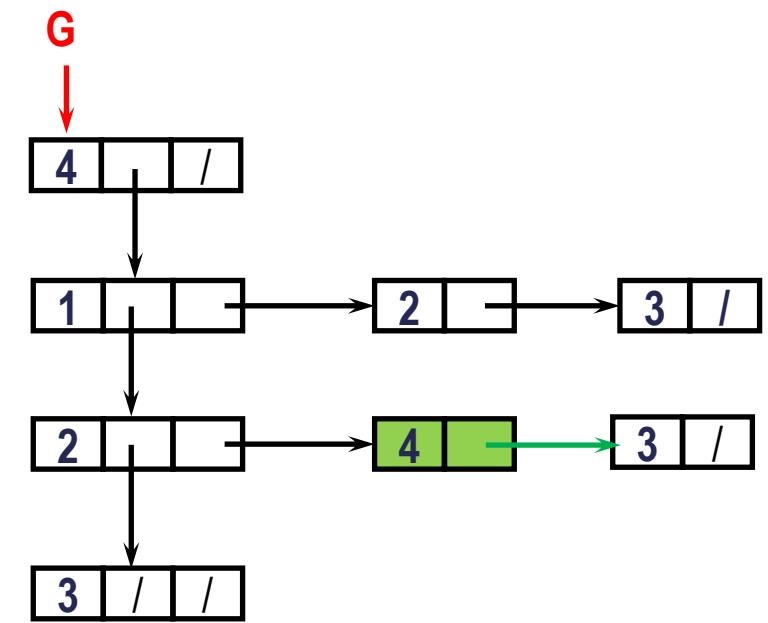
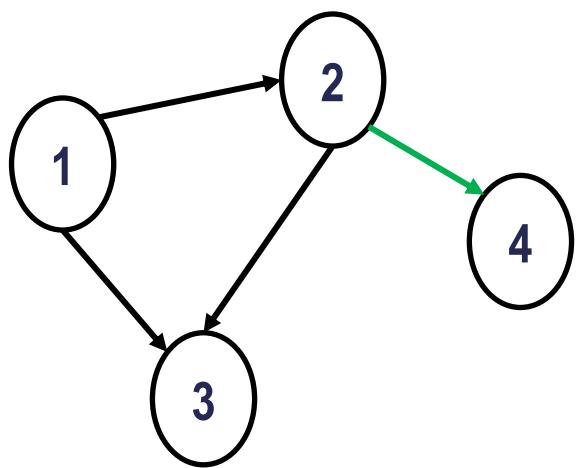
# INSERÇÃO DE ARESTA EM DIGRAFO

Exemplo: Inserir aresta (2, 4)



# INSERÇÃO DE ARESTA EM DIGRAFO

Exemplo: Inserir aresta  $(2, 4)$



# INSERÇÃO DE ARESTA EM DIGRAFO

```
void insere_aresta_digrafo(TGrafo *g, int v1, int v2) {
    TVizinho *v = busca_aresta(g, v1, v2);
    if(v == NULL) {
        insere_um_sentido(g, v1, v2);
    }
}
```

# EXCLUSÃO DE ARESTA

## Grafo não orientado (digrafo)

- **Exclusão de aresta  $\{v_1, v_2\}$ :** excluir  $v_2$  da lista de vizinhos de  $v_1$ , e  $v_1$  da lista de vizinhos de  $v_2$  (ou seja, excluir as arestas  $(v_1, v_2)$  e  $(v_2, v_1)$ )

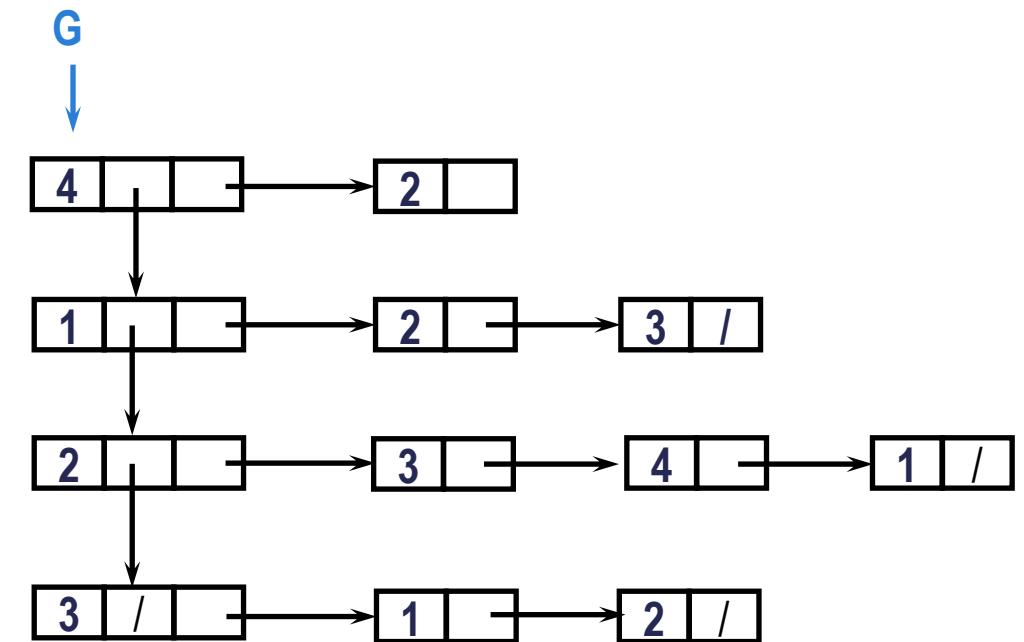
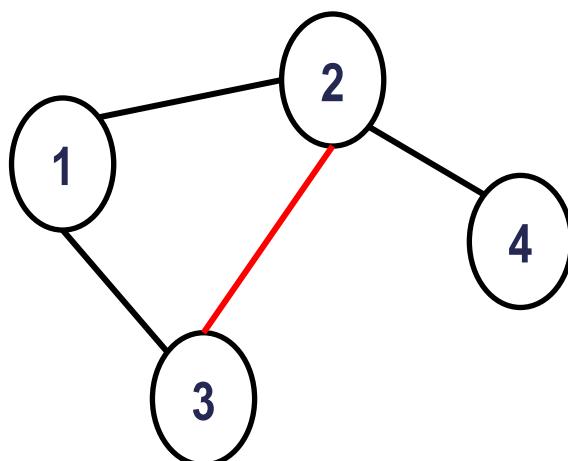
## Grafo orientado

- **Exclusão de aresta  $(v_1, v_2)$ :** excluir  $v_2$  da lista de vizinhos de  $v_1$

Em ambos os casos, liberar a memória

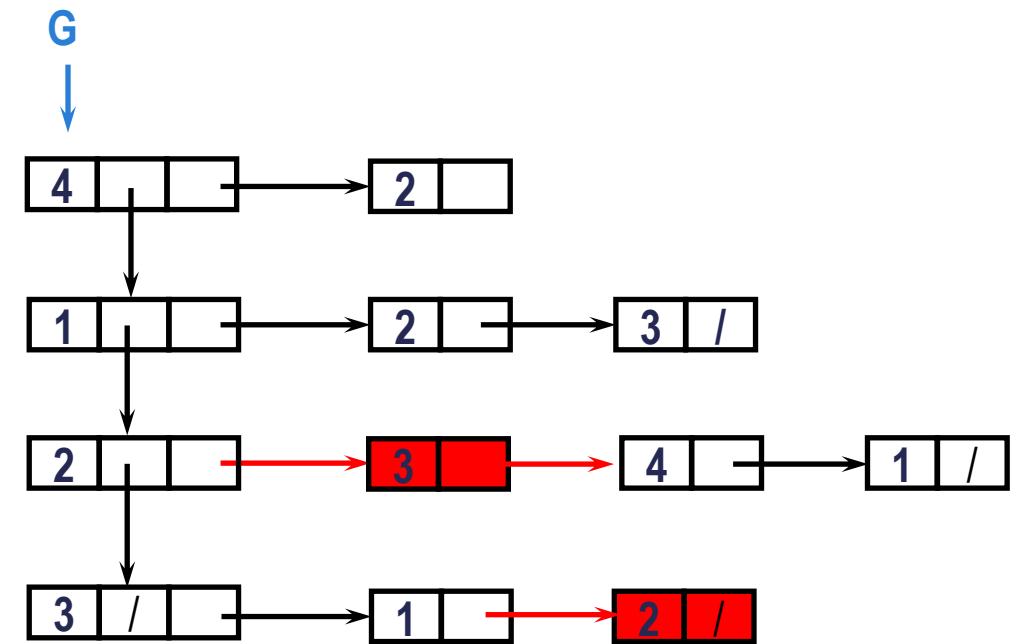
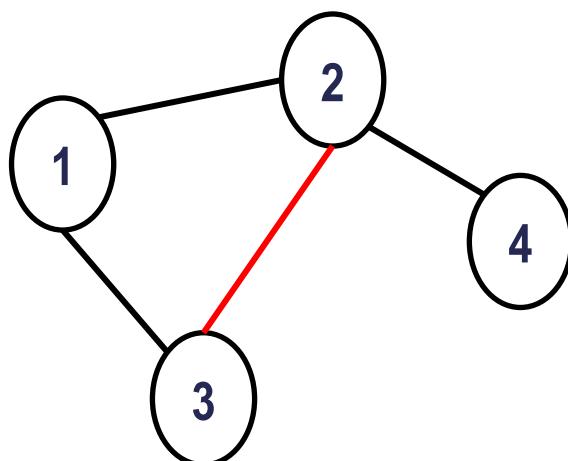
# EXCLUSÃO DE ARESTA EM GRAFO NÃO ORIENTADO

Exemplo: exclusão da aresta {2, 3}



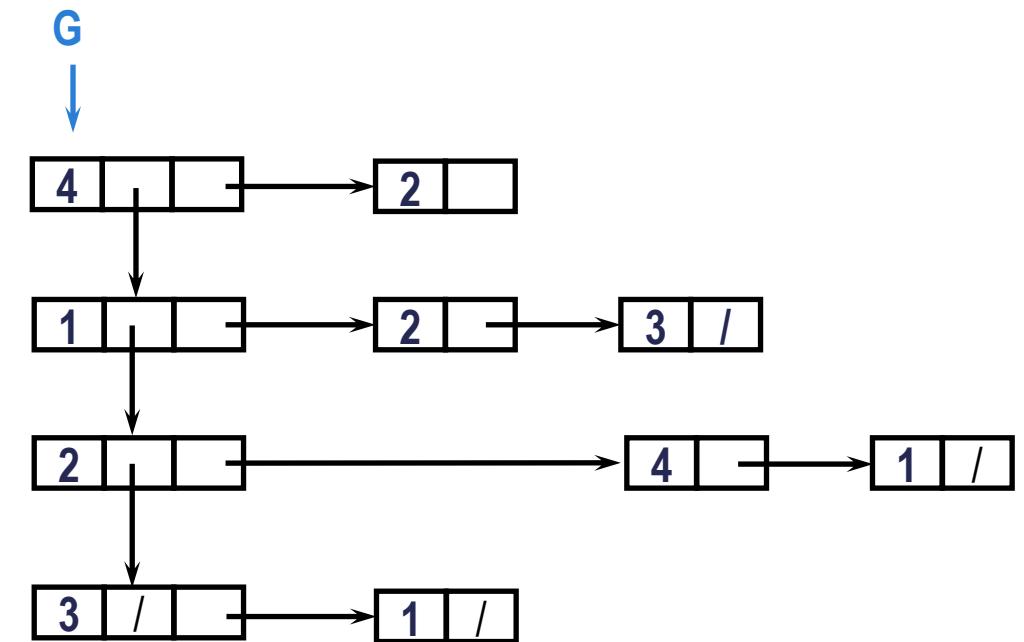
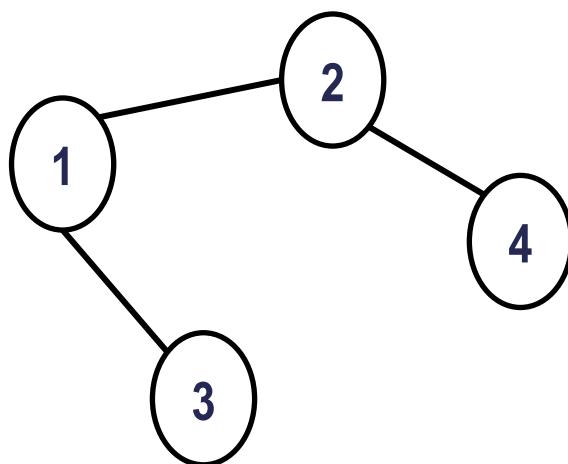
# EXCLUSÃO DE ARESTA EM GRAFO NÃO ORIENTADO

Exemplo: exclusão da aresta {2, 3}



# EXCLUSÃO DE ARESTA EM GRAFO NÃO ORIENTADO

Exemplo: exclusão da aresta {2, 3}



# EXCLUSÃO DE ARESTA EM GRAFO NÃO ORIENTADO

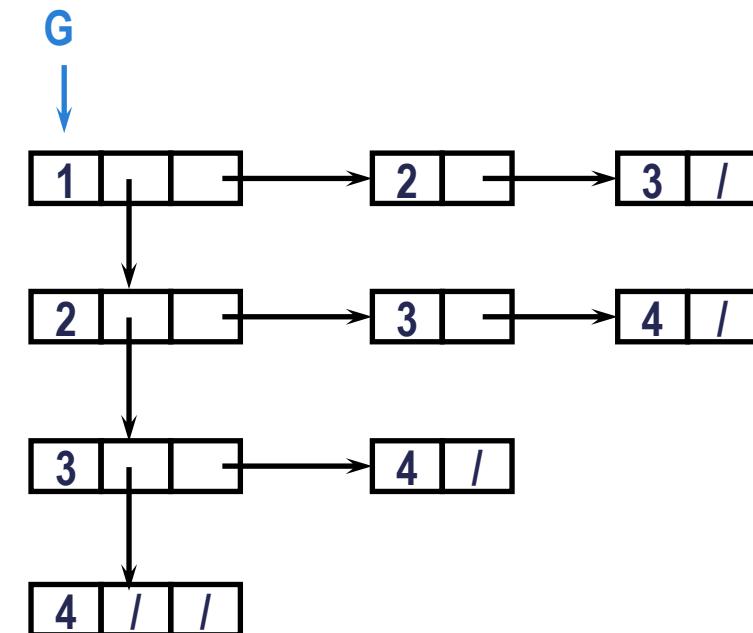
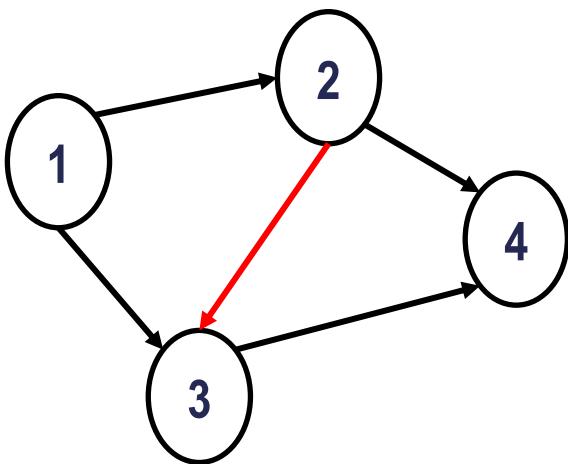
```
void retira_um_sentido(TGrafo *g, int v1, int v2) {
    TGrafo *p = busca_vertice(g, v1);
    if(p != NULL) {
        TVizinho *ant = NULL;
        TVizinho *atual = p->prim_vizinho;
        while ((atual) && (atual->id_vizinho != v2)) {
            ant = atual;
            atual = atual->prox;
        }
        if (ant == NULL) //v2 era o primeiro nó da lista
            p->prim_vizinho = atual->prox;
        else
            ant->prox = atual->prox;
        free(atual);
    }
}
```

# EXCLUSÃO DE ARESTA EM GRAFO NÃO ORIENTADO (CONT.)

```
void retira_aresta(TGrafo *g ,int v1, int v2) {
    TVizinho* v = busca_aresta(g,v1,v2);
    if(v != NULL) {
        retira_um_sentido(g, v1, v2);
        retira_um_sentido(g, v2, v1);
    }
}
```

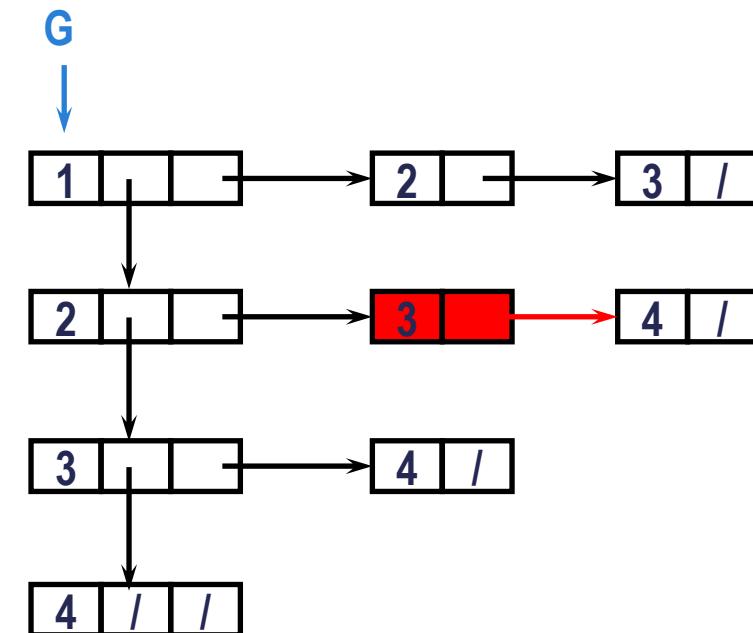
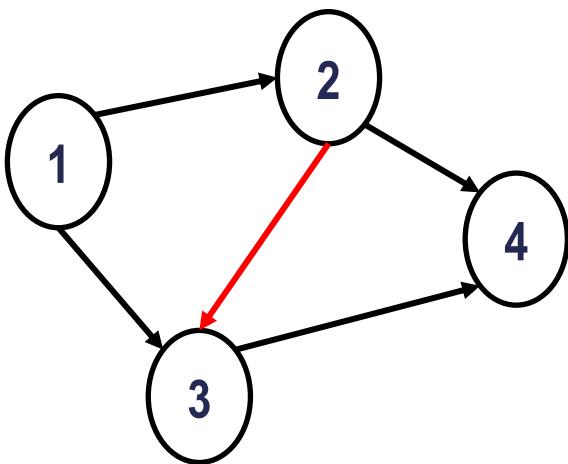
# EXCLUSÃO DE ARESTA EM DIGRAFO

Exemplo: Exclusão de aresta (2, 3)



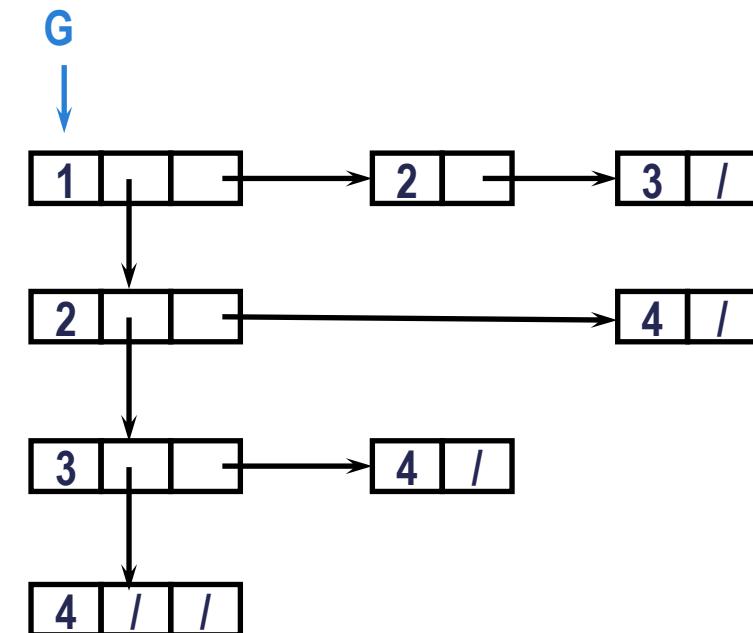
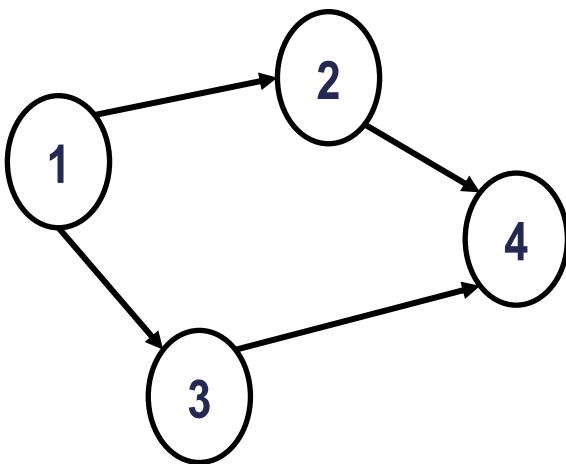
# EXCLUSÃO DE ARESTA EM DIGRAFO

Exemplo: Exclusão de aresta (2, 3)



# EXCLUSÃO DE ARESTA EM DIGRAFO

Exemplo: Exclusão de aresta (2, 3)



# EXCLUSÃO DE ARESTA EM DIGRAFO

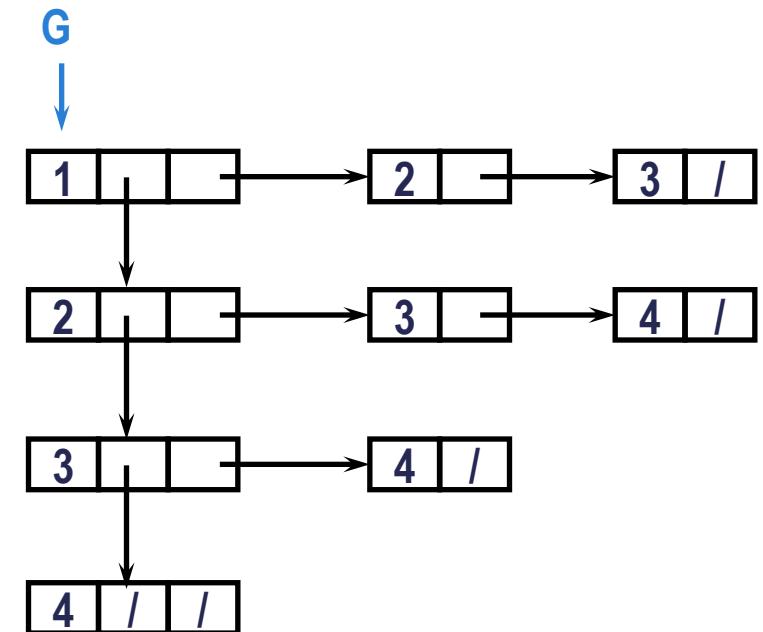
```
void retira_aresta_digrafo(TGrafo *g ,int v1, int v2) {  
    TVizinho* v = busca_aresta(g,v1,v2);  
    if(v != NULL) {  
        retira_um_sentido(g, v1, v2);  
    }  
}
```

# EXCLUSÃO DE VÉRTICE

Exclui

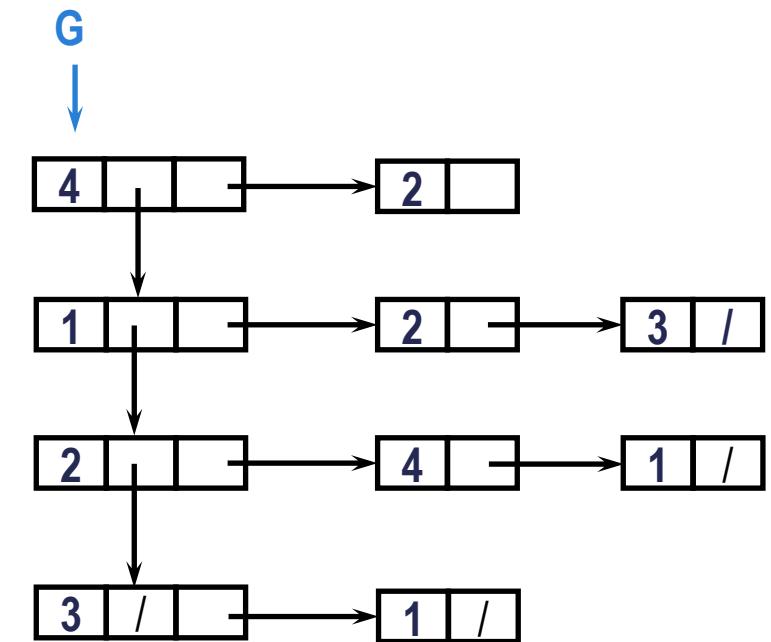
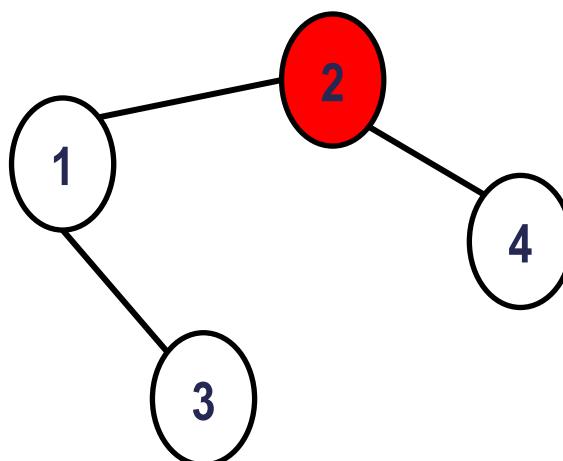
- vértice
- sua lista de vizinhos
- todos os vizinhos que tinham esse vértice como extremidade

Libera memória



# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

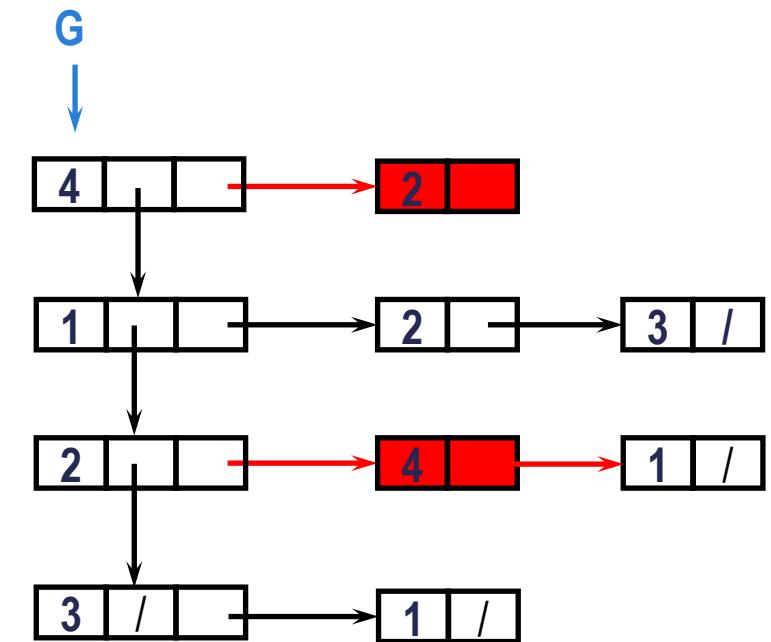
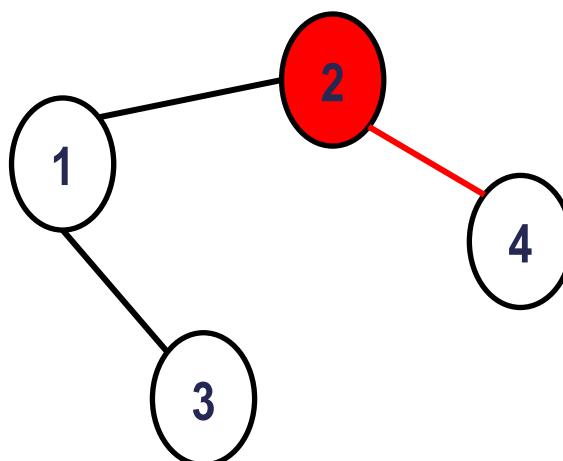
Exemplo: exclusão do vértice 2



# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

Retira todos os vizinhos de 2

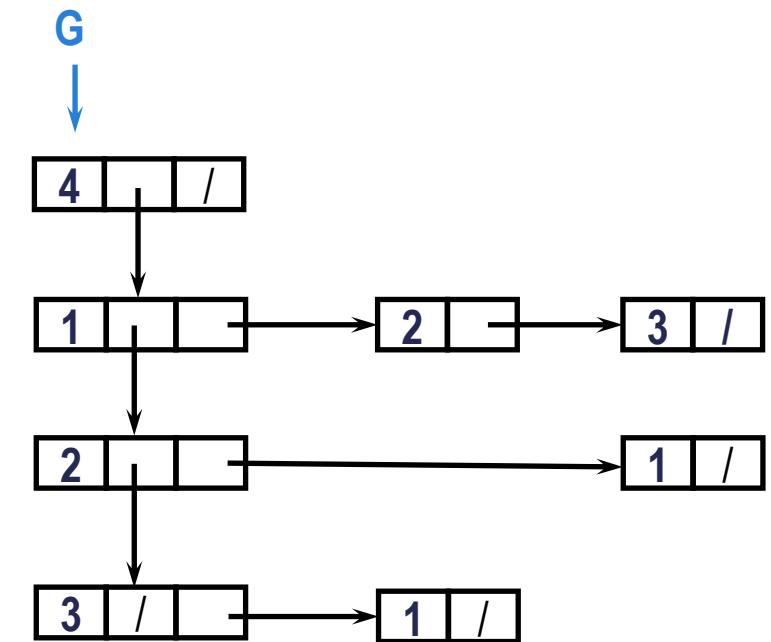
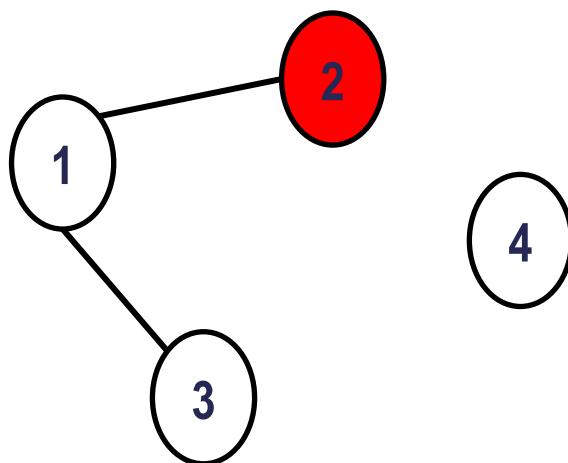
- Retira vizinho 4



# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

Retira todos os vizinhos de 2

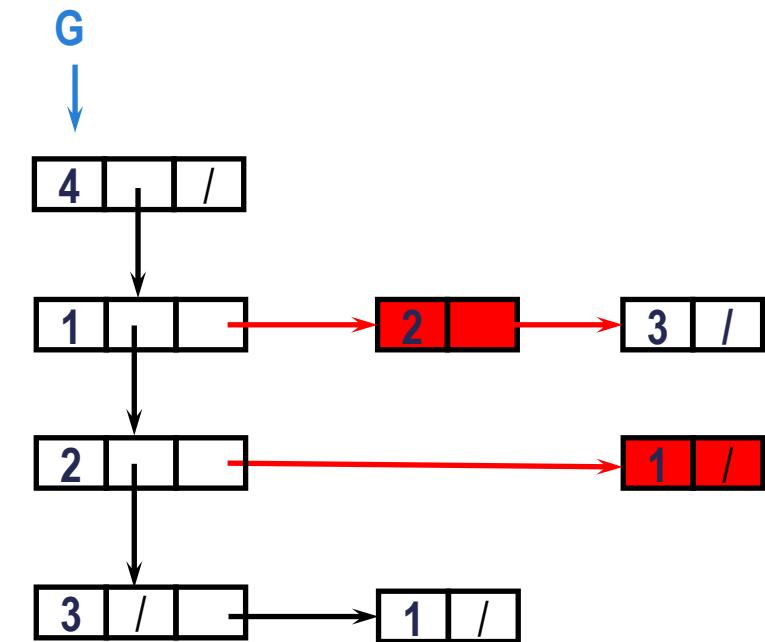
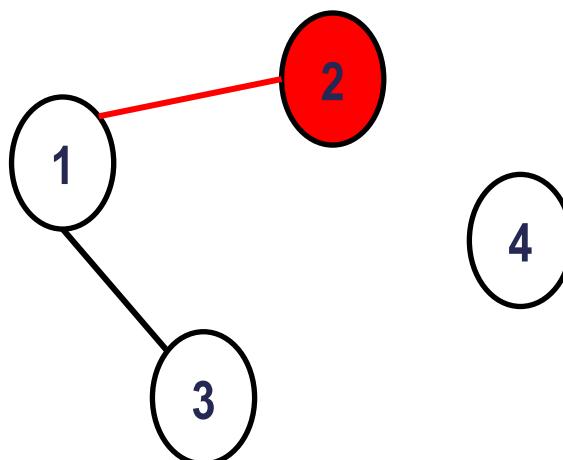
- Retira vizinho 4



# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

Retira todos os vizinhos de 2

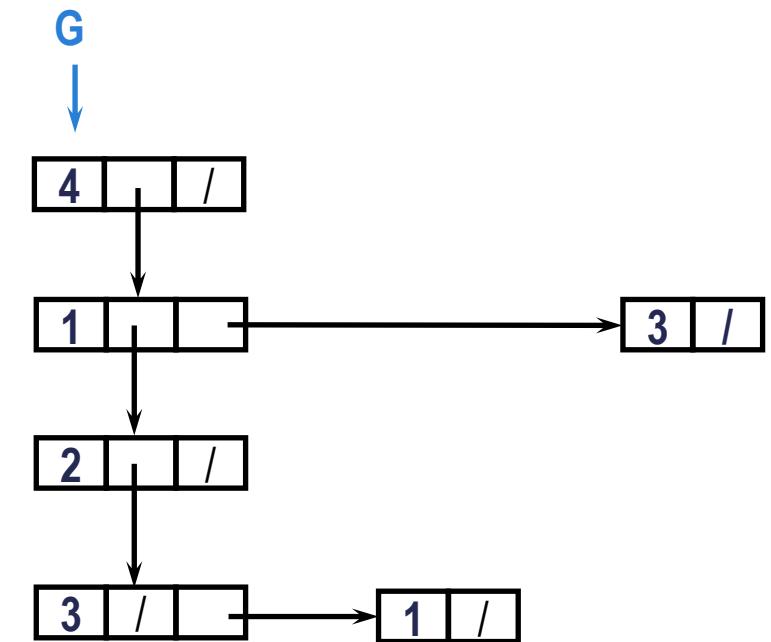
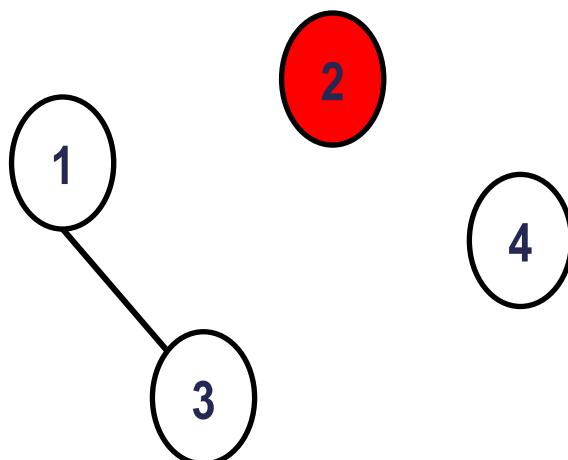
- Retira vizinho 1



# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

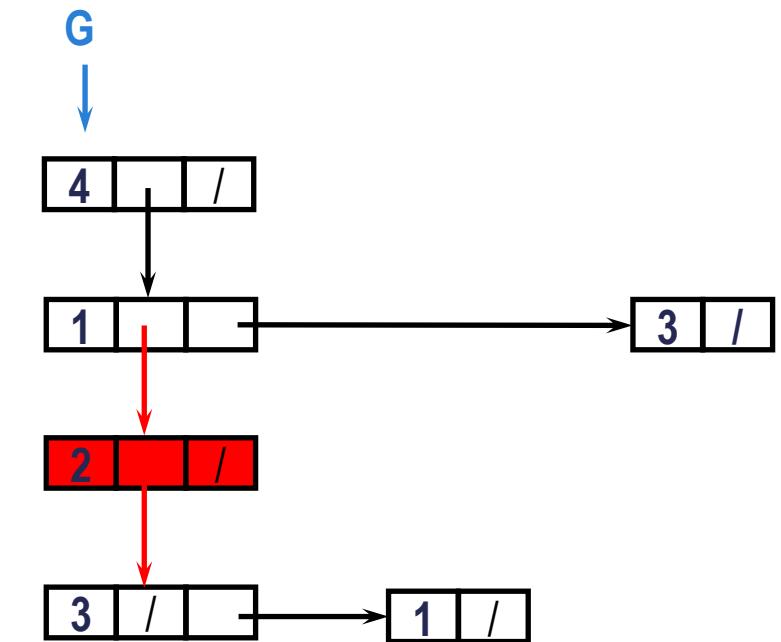
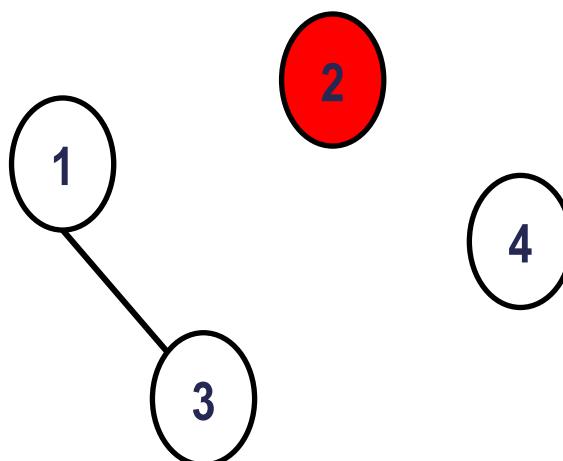
Retira todos os vizinhos de 2

- Retira vizinho 1



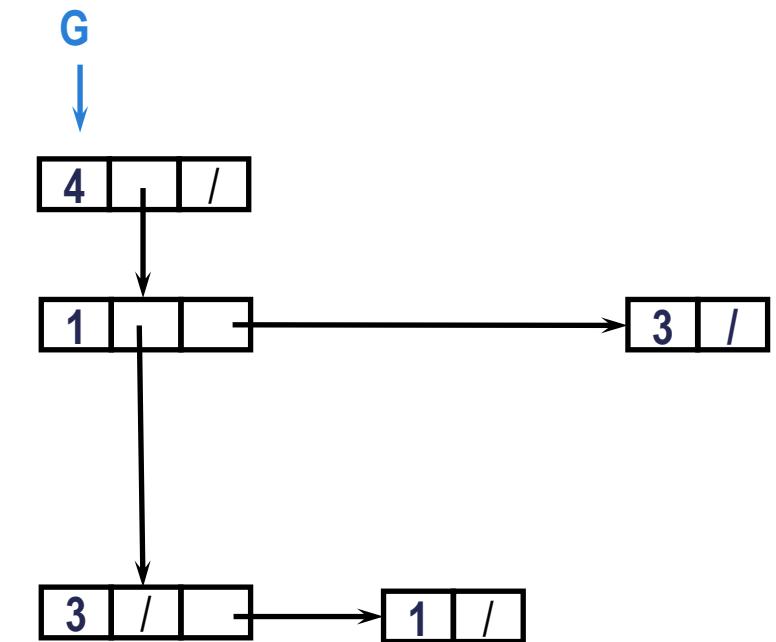
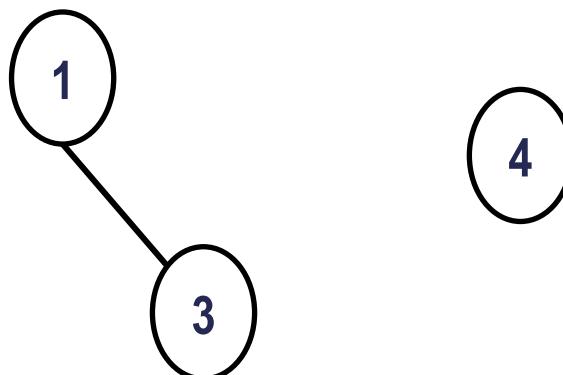
# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

Retira vértice 2



# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

Retira vértice 2

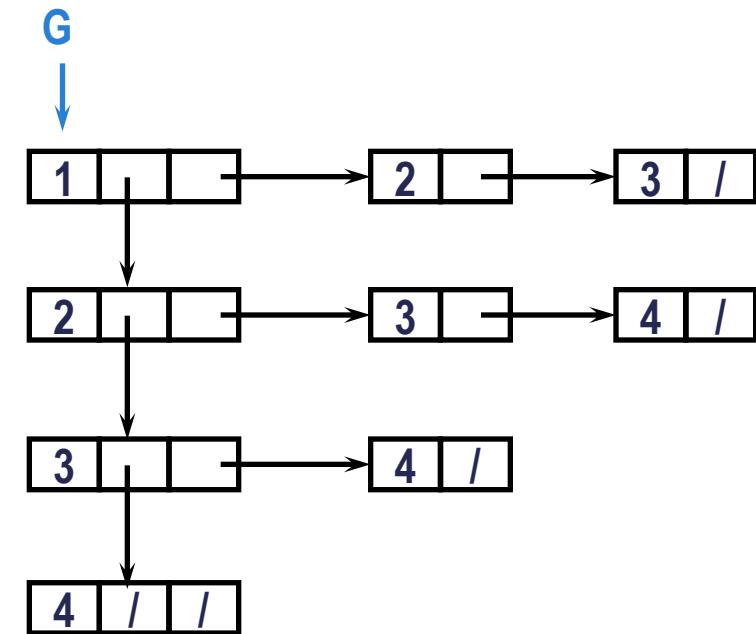
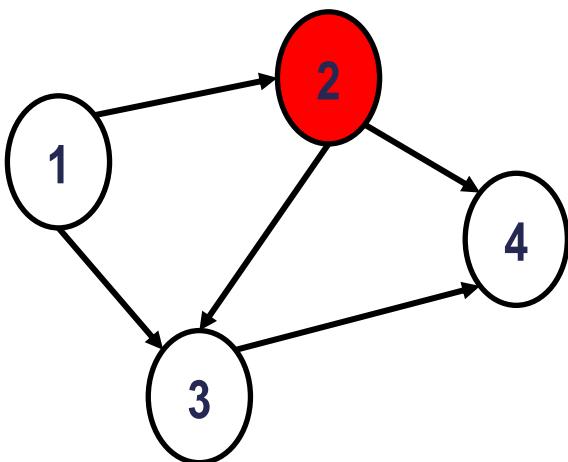


# EXCLUSÃO DE VÉRTICE EM GRAFO NÃO ORIENTADO

```
TGrafo *retira_vertice(TGrafo *g, int v) {
    TGrafo *p = g;
    TGrafo *ant = NULL;
    while( (p != NULL) && (p->id_vertice != v) ) {
        ant = p;
        p = p->prox;
    }
    if(p != NULL) {
        while (p->prim_vizinho != NULL)
            retira_aresta(g, v, p->prim_vizinho->id_vizinho);
        if (ant == NULL)
            g = g->prox;
        else
            ant->prox = p->prox;
        free(p);
    }
    return g;
}
```

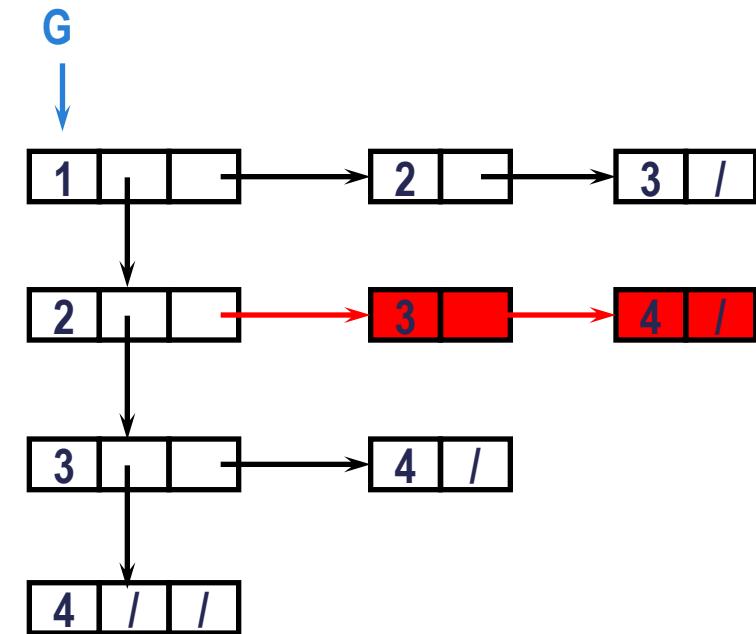
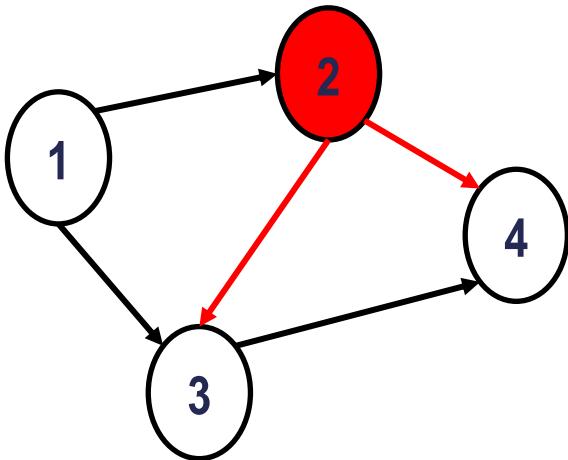
# EXCLUSÃO DE VÉRTICE EM DIGRAFO

Exemplo: exclusão do vértice 2



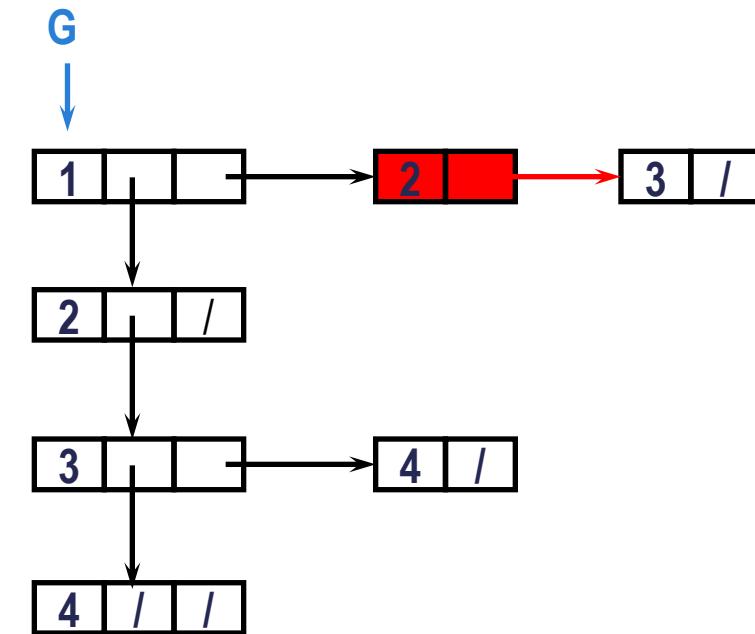
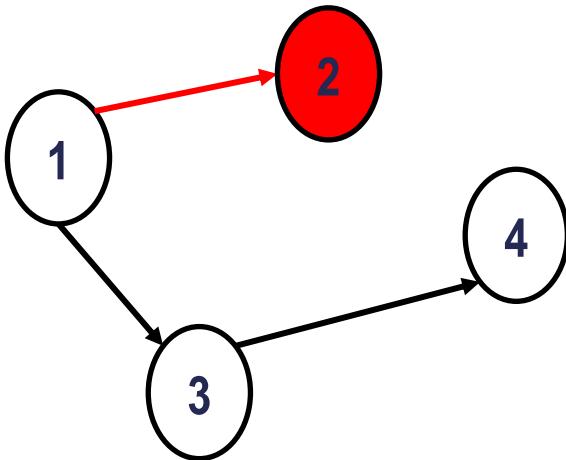
# EXCLUSÃO DE VÉRTICE EM DIGRAFO

Retira todos os vizinhos de 2



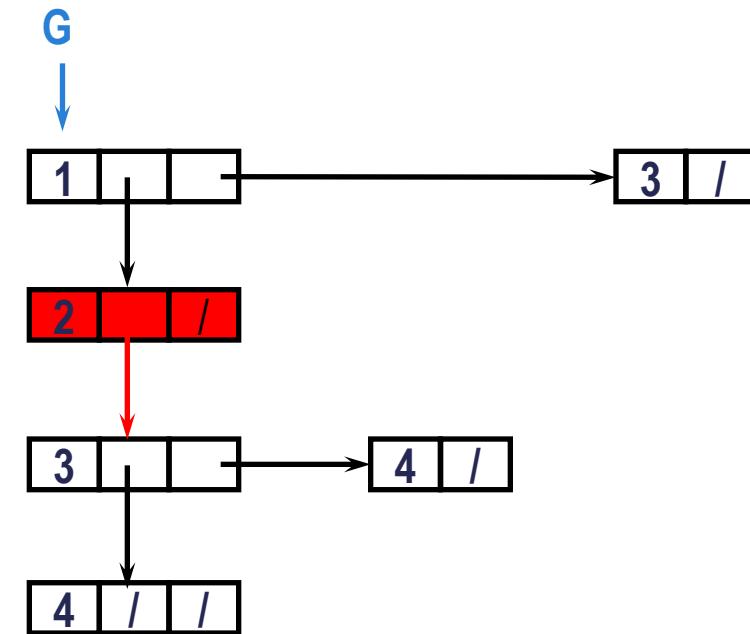
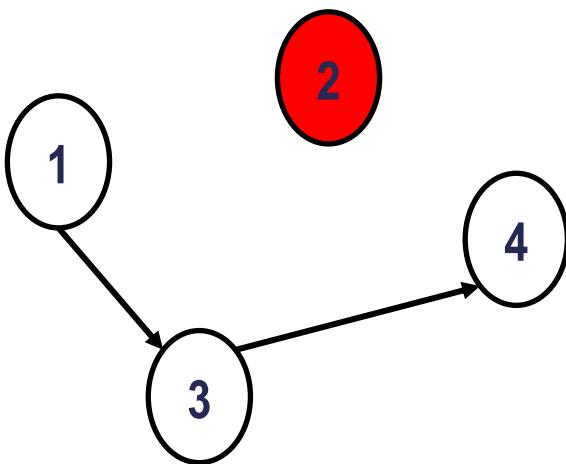
# EXCLUSÃO DE VÉRTICE EM DIGRAFO

Retira 2 da lista de vizinhos dos outros nós



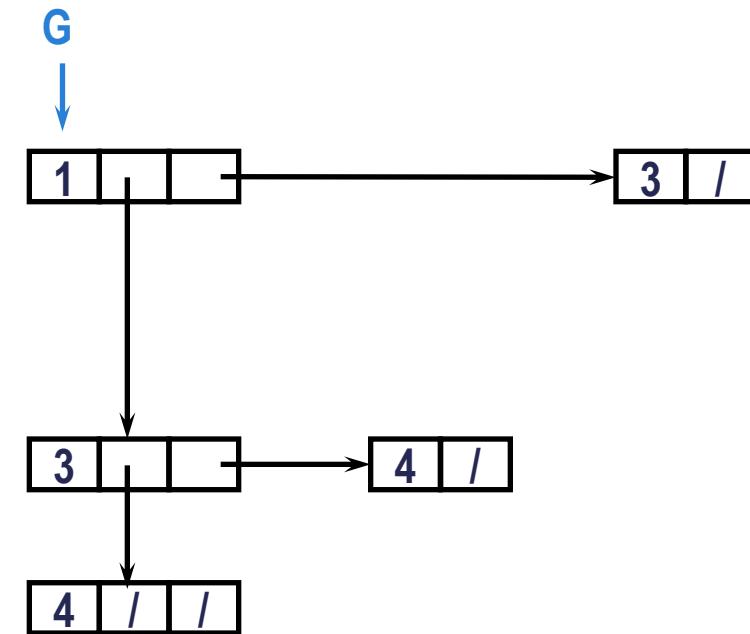
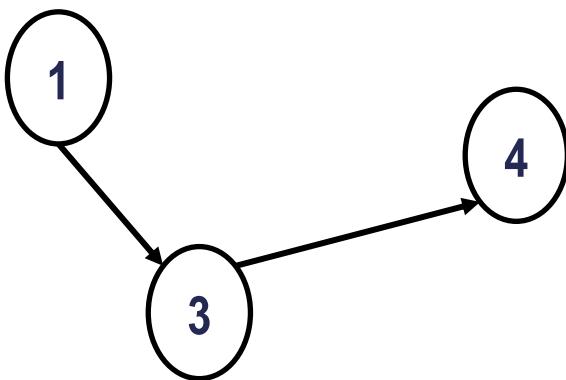
# EXCLUSÃO DE VÉRTICE EM DIGRAFO

Retira vértice 2



# EXCLUSÃO DE VÉRTICE EM DIGRAFO

Retira vértice 2



# EXCLUSÃO DE VÉRTICE EM DIGRAFO

**Exercício:** escreva uma função em C para exclusão de vértice em grafo orientado

- `TGrafo *retira_vértice_digrafo(TGrafo *g, int v);`

# ALGORITMOS EM GRAFOS

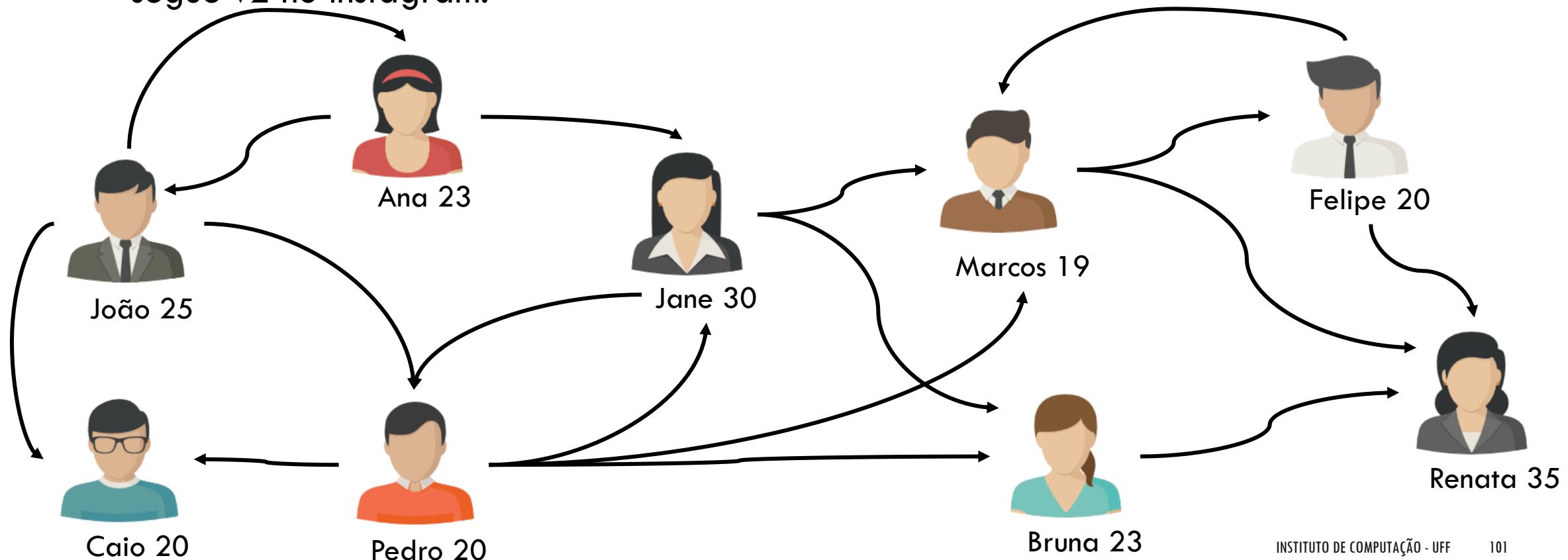
Existem diversos algoritmos eficientes para encontrar **caminhos** em grafos

- Eles serão aprendidos na disciplina de Algoritmos em Grafos (6º. Semestre)

Aqui na disciplina faremos exercícios procurando por caminhos e resolvendo pequenos problemas em estruturas de grafos, para motivar o uso de grafos e o aprendizado dos algoritmos na disciplina de Algoritmos em Grafos

# EXERCÍCIOS

Considere o grafo a seguir, que representa seguidores no Instagram. Cada pessoa tem nome e idade (**nome** é o **id do vértice**). Uma aresta  $(v_1, v_2)$  significa que  $v_1$  segue  $v_2$  no Instagram.



# EXERCÍCIOS

Implementar funções em C para responder às seguintes questões:

1. Quantas pessoas uma determinada pessoa segue?

```
int numero_seguidos (TGrafo *g, char *nome);
```

2. Quem são os seguidores de uma determinada pessoa? (função imprime os nomes dos seguidores, caso a flag **imprime** seja True, e retorna quantidade de seguidores)

```
int seguidores (TGrafo *vertice, char *nome, int imprime);
```

3. Quem é a pessoa mais popular? (tem mais seguidores)

```
TGrafo *mais_popular (TGrafo *g);
```

# EXERCÍCIOS

4. Quais são as pessoas que só seguem pessoas mais velhas do que ela própria?  
(função imprime os nomes das pessoas, caso a flag **imprime** seja True, e retorna quantidade de pessoas)

```
int segue_mais_velho(TGrafo *g, int imprime);
```

# REFERÊNCIA

Celes, W.; Cerqueira, R.; Rangel, J.L. Introdução a Estruturas de Dados com Técnicas de Programação em C, 2a. ed. Elsevier. Cap. 22

# AGRADECIMENTOS

Material baseado nos slides de Renata Galante, UFRGS

Implementação em C baseada no material de Isabel Rosseti, UFF