

# MAC0328 Algoritmos em Grafos

Edição 2009

# AULA 1

# Administração

Página da disciplina: aulas, cadastro, fórum, ...

<http://paca.ime.usp.br/>

Livro:

- ▶ **PF** = Paulo Feofiloff,  
*Algoritmos para Grafos em C via Sedgwick*  
[www.ime.usp.br/~pf/algoritmos\\_para\\_grafos/](http://www.ime.usp.br/~pf/algoritmos_para_grafos/)
- ▶ **S** = Robert Sedgwick,  
*Algorithms in C (part 5: Graph Algorithms)*
- ▶ **CLRS** = Cormen-Leiserson-Rivest-Stein,  
*Introductions to Algorithms*

# MAC0328

MAC0328 Algoritmos em grafos é:

- ▶ uma disciplina introdutória em projeto e análise de algoritmos sobre grafos
- ▶ um **laboratório de algoritmos** sobre grafos

# MAC0328

MAC0328 combina técnicas de

- ▶ programação
- ▶ estruturas de dados
- ▶ análise de algoritmos
- ▶ teoria dos grafos

para resolver problemas sobre grafos.

# Pré-requisitos

O pré-requisito oficial de [MAC0328](#) é

- ▶ [MAC0122](#) Princípios de Desenvolvimento de Algoritmos.

No entanto, é recomendável que já tenham cursado

- ▶ [MAC0211](#) Laboratório de programação; e
- ▶ [MAC0323](#) Estruturas de dados

Costuma ser conveniente cursar [MAC0328](#) simultaneamente com

- ▶ [MAC0338](#) Análise de algoritmos.

# Principais tópicos

- ▶ grafos dirigidos
- ▶ estruturas de dados para grafos
- ▶ construção de grafos aleatórios
- ▶ florestas e árvores
- ▶ caminhos e ciclos
- ▶ **busca em largura**
- ▶ caminhos mínimos
- ▶ grafos bipartidos
- ▶ **busca em profundidade**
- ▶ grafos dirigidos acíclicos
- ▶ ordenação topológica
- ▶ pontes e ciclos
- ▶ grafos conexos e componentes
- ▶ grafos biconexos
- ▶ árvores geradoras mínimas
- ▶ fluxo em redes

# Digrafos

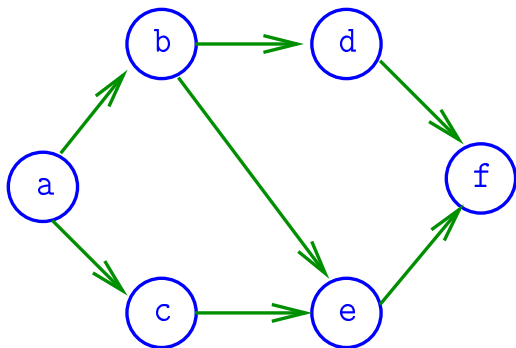
S 17.0, 17.1



# Digrafos

Um **digrafo** (*directed graph*) consiste de um conjunto de **vértices** (bolas) e um conjunto de **arcos** (flechas)

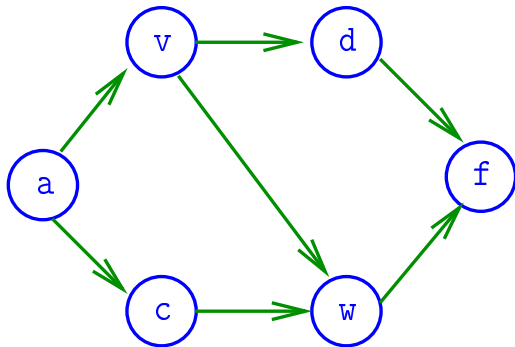
**Exemplo:** representação de um grafo



# Arcos

Um **arco** é um par ordenado de vértices

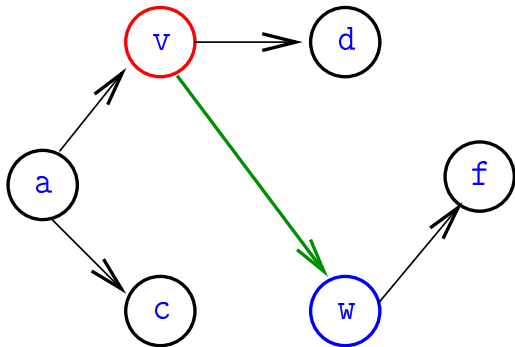
Exemplo:  $v$  e  $w$  são vértices e  $v-w$  é um arco



## Ponta inicial e final

Para cada arco  $v-w$ , o vértice  $v$  é a **ponta inicial** e  $w$  é a **ponta final**

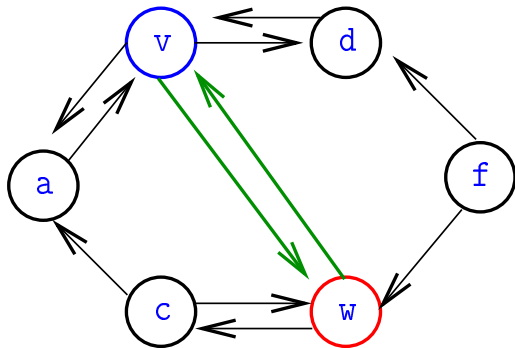
Exemplo:  $v$  é ponta inicial e  $w$  é ponta final de  $v-w$



## Arcos anti-paralelos

Dois arcos são **anti-paralelos** se a ponta inicial de um é ponta final do outro

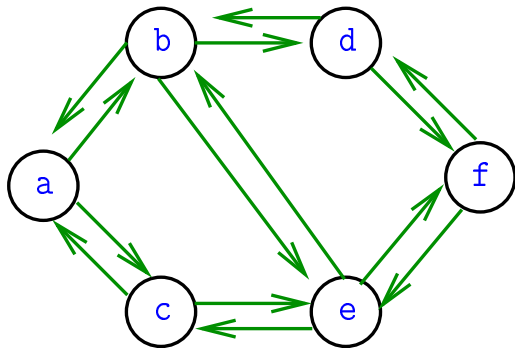
Exemplo:  $v-w$  e  $w-v$  são anti-paralelos



# Digrafos simétricos

Um digrafo é **simétrico** se cada um de seus arcos é anti-paralelo a outro

Exemplo: digrafo simétrico

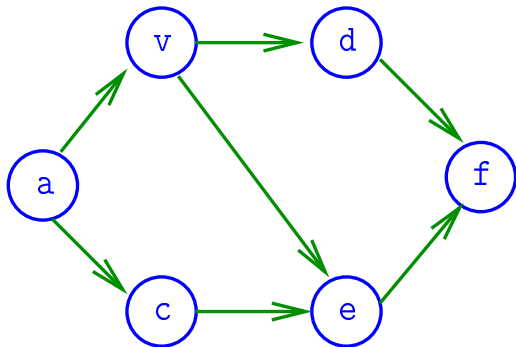


## Graus de entrada e saída

**grau de entrada** de  $v$  = no. arcos com ponta final  $v$

**grau de saída** de  $v$  = no. arcos com ponta inicial  $v$

Exemplo:  $v$  tem grau de entrada 1 e de saída 2



# Número de arcos

Quantos arcos, no máximo, tem um digrafo com  $V$  vértices?

## Número de arcos

Quantos arcos, no máximo, tem um digrafo com  $V$  vértices?

A resposta é  $V \times (V - 1) = \Theta(V^2)$

digrafo **completo** = todo par ordenado de vértices distintos é arco

digrafo **denso** = tem “muitos” muitos arcos

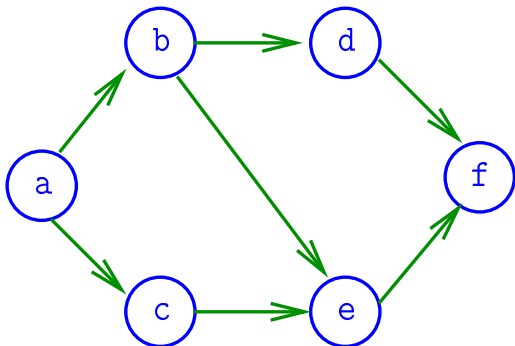
digrafo **esparso** = tem “poucos” arcos



# Especificação

Digrafos podem ser especificados através de sua lista de arcos

Exemplo:



d-f

b-d

a-c

b-e

e-f

a-b

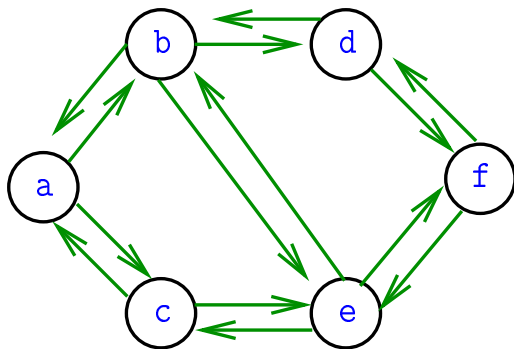
# Grafos

S 17.0, 17.1

# Grafos

Um **grafo** é um digrafo **simétrico**

Exemplo: um grafo

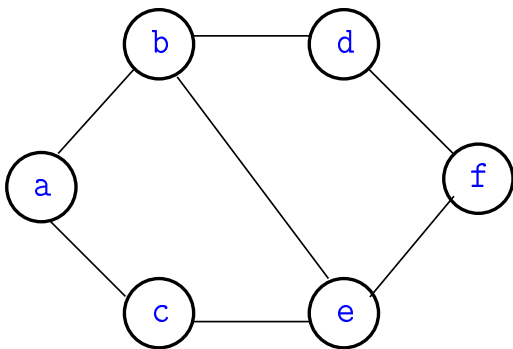




# Arestas

Uma **aresta** é um par de arcos anti-paralelos.

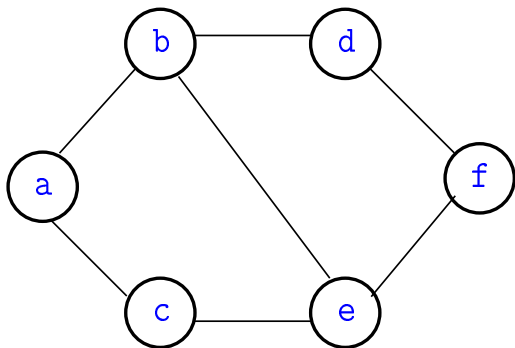
Exemplo:  $b-a$  e  $a-b$  são a **mesma** aresta



# Especificação

Grafos podem ser especificados através de sua lista de arestas

Exemplo:



f-d

b-d

c-a

e-b

e-f

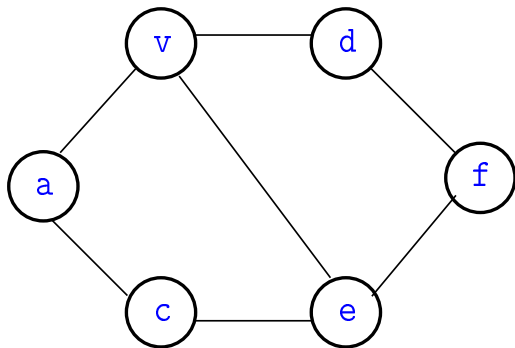
a-b

# Graus de vértices

Em um grafo

**grau** de  $v$  = número de arestas com ponta em  $v$

Exemplo:  $v$  tem grau 3



# Número de arestas

Quantas arestas, no máximo, tem um grafo com  $V$  vértices?



# Número de arestas

Quantas arestas, no máximo, tem um grafo com  $V$  vértices?

A resposta é  $V \times (V - 1)/2 = \Theta(V^2)$

grafo **completo** = todo par **não**-ordenado de vértices distintos é aresta

# Estruturas de dados

S 17.2

# Vértices

Vértices são representados por objetos do tipo **Vertex**.

Os vértices de um digrafo são  $0, 1, \dots, V-1$ .

```
#define Vertex int
```

# Arcos

Um objeto do tipo **Arc** representa um arco com ponta inicial **v** e ponta final **w**.

```
typedef struct {  
    Vertex v;  
    Vertex w;  
} Arc;
```

# ARC

A função **ARC** recebe dois vértices **v** e **w** e devolve um arco com ponta inicial **v** e ponta final **w**.

# ARC

A função **ARC** recebe dois vértices **v** e **w** e devolve um arco com ponta inicial **v** e ponta final **w**.

```
Arc ARC (Vertex v, Vertex w) {  
1     Arc e;  
2     e.v = v;  
3     e.w = w;  
4     return e;  
}
```

# Arestas

Um objeto do tipo **Edge** representa uma aresta com pontas **v** e **w**.

# Arestas

Um objeto do tipo `Edge` representa uma aresta com pontas `v` e `w`.

```
#define Edge Arc
```



# Arestas

A função **EDGE** recebe dois vértices **v** e **w** e devolve uma aresta com pontas **v** e **w**.

# Arestas

A função **EDGE** recebe dois vértices **v** e **w** e devolve uma aresta com pontas **v** e **w**.

```
#define EDGE ARC
```

# Grafos no computador

Usaremos duas representações clássicas:

- ▶ matriz de adjacência
- ▶ vetor de listas de adjacência

Há várias outras maneiras, como, por exemplo

- ▶ matriz de incidência

que é apropriada para [MAC0315 Prog. Linear](#).

# Matrizes de adjacência

S 17.3

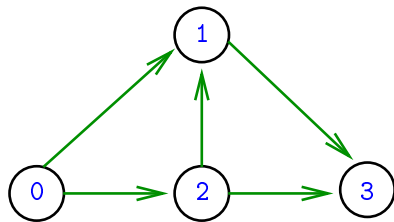
# Matriz de adjacência de digrafo

**Matriz de adjacência** de um digrafo tem linhas e colunas indexadas por vértices:

$\text{adj}[v][w] = 1$  se  $v-w$  é um arco

$\text{adj}[v][w] = 0$  em caso contrário

Exemplo:



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

Consumo de espaço:  $\Theta(V^2)$

fácil de implementar

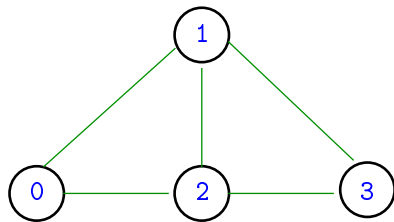
# Matriz de adjacência de grafo

**Matriz de adjacência** de um grafo tem linhas e colunas indexadas por vértices:

$\text{adj}[v][w] = 1$  se  $v-w$  é um aresta

$\text{adj}[v][w] = 0$  em caso contrário

Exemplo:



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

Consumo de espaço:  $\Theta(V^2)$

fácil de implementar

# Estrutura **digraph**

A estrutura **digraph** representa um digrafo

**V** contém o número de vértices

**A** contém o número de arcos do digrafo

**adj** é um ponteiro para a matriz de adjacência

## Estrutura digraph

A estrutura **digraph** representa um digrafo

**V** contém o número de vértices

**A** contém o número de arcos do digrafo

**adj** é um ponteiro para a matriz de adjacência

```
struct digraph {  
    int V;  
    int A;  
    int **adj;  
};
```



# Estrutura Digraph

Um objeto do tipo `Digraph` contém o endereço de um `digraph`

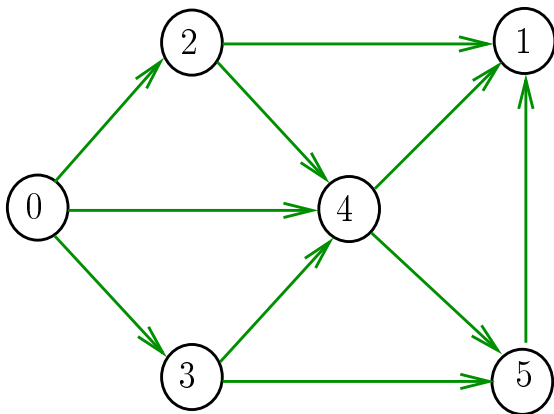
# Estrutura Digraph

Um objeto do tipo `Digraph` contém o endereço de um `digraph`

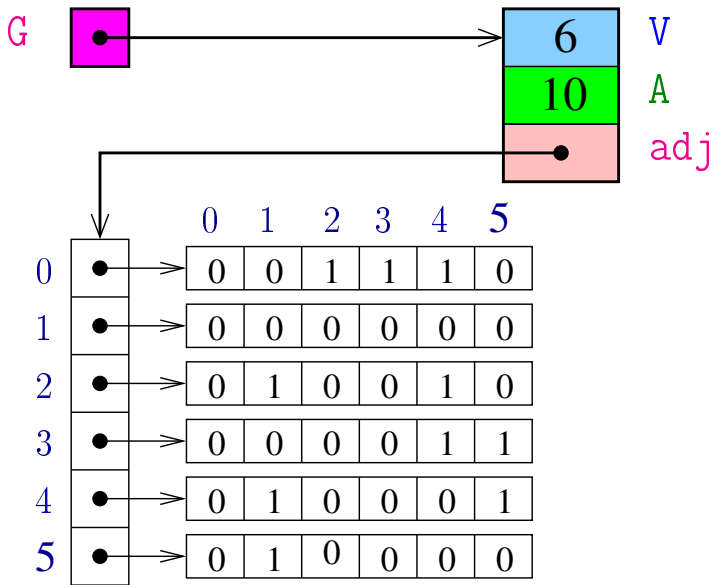
```
typedef struct digraph *Digraph;
```

# Digrafo

Digraph  $G$



# Estruturas de dados



# Estruturas `graph` e `Graph`

Essa mesma estrutura será usada para representar grafos

# Estruturas `graph` e `Graph`

Essa mesma estrutura será usada para representar grafos

```
#define graph digraph
```

```
#define Graph Digraph
```

O número de arestas de um grafo  $G$  é

# Estruturas `graph` e `Graph`

Essa mesma estrutura será usada para representar grafos

```
#define graph digraph
```

```
#define Graph Digraph
```

O número de arestas de um grafo  $G$  é

$$(G \rightarrow A) / 2$$

# Funções básicas

S 17.3



## MATRIXint

Aloca uma matriz com linhas  $0 \dots r-1$  e colunas  $0 \dots c-1$ , cada elemento da matriz recebe valor **val**

```
int **MATRIXint (int r, int c, int val) {
```

## MATRIXint

Aloca uma matriz com linhas  $0 \dots r-1$  e colunas  $0 \dots c-1$ , cada elemento da matriz recebe valor **val**

```
int **MATRIXint (int r, int c, int val) {  
0     Vertex v, w;  
1     int **m = malloc(r * sizeof(int *));  
2     for (v = 0; v < r; v++)  
3         m[v] = malloc(c * sizeof(int));  
4     for (v = 0; v < r; v++)  
5         for (w = 0; w < c; w++)  
6             m[v][w] = val;  
7     return m;  
}
```

## Consumo de tempo

linha	número de execuções da linha	
1	$= 1$	$= \Theta(1)$
2	$= r + 1$	$= \Theta(r)$
3	$= r$	$= \Theta(r)$
4	$= r + 1$	$= \Theta(r)$
5	$= r \times (c + 1)$	$= \Theta(r c)$
6	$= r \times c$	$= \Theta(r c)$
total	$\Theta(1) + 3 \Theta(r) + 2 \Theta(r c)$ $= \Theta(r c)$	

# Conclusão

Supondo que o consumo de tempo da função `malloc` é constante

O consumo de tempo da função `MATRIXint` é  $\Theta(r c)$ .

# DIGRAPHinit

Devolve (o endereço de) um novo digrafo com vértices  $0, \dots, V-1$  e nenhum arco.

```
Digraph DIGRAPHinit (int V) {
```

# DIGRAPHinit

Devolve (o endereço de) um novo digrafo com vértices  $0, \dots, V-1$  e nenhum arco.

```
Digraph DIGRAPHinit (int V) {  
0     Digraph G = malloc(sizeof *G);  
1     G->V = V;  
2     G->A = 0;  
3     G->adj = MATRIXint(V, V, 0);  
4     return G;  
}
```