

# 02 – Grafos: estruturas de dados

## SCC0503 – Algoritmos e Estruturas de Dados II

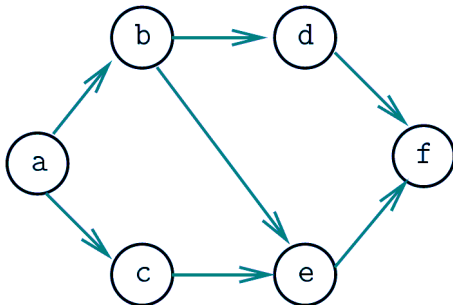
Prof. Moacir Ponti Jr.  
[www.icmc.usp.br/~moacir](http://www.icmc.usp.br/~moacir)

Instituto de Ciências Matemáticas e de Computação – USP

2011/1

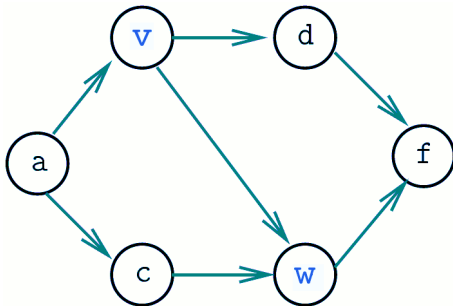
- 1 Recordando
- 2 Estruturas de Dados
  - Lista de Arcos
  - Lista de Adjacência
  - Matriz de Adjacência
  - Matriz de Incidência
- 3 Desempenho assintótico
- 4 Estruturas alternativas

- *Directed graph*, ou **digrafo** é um conjunto de **vértices** e um conjunto de **arcos**



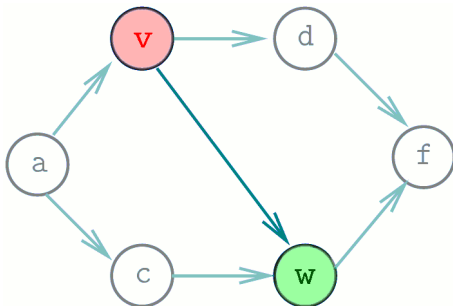
# Arcos e Vértices

- Um **arco**, é um par ordenado de vértices
- **Exemplo:**  $v$  e  $w$  são vértices e  $v-w$  é um arco.



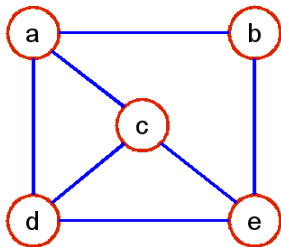
# Examinando um arco

- O primeiro vértice do par ordenado é a **ponta inicial** do arco, e o segundo a **ponta final**.



# Conceitos Básicos

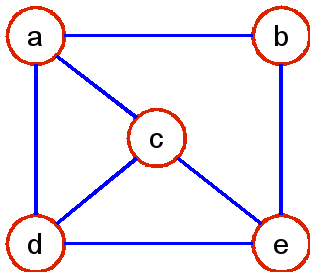
- Um grafo  $G = (V, E)$  é composto de:
  - $V$ : conjunto de **vértices**
  - $E$ : conjunto de **arestas**
- Se  $\alpha = \{v, w\}$  é uma aresta de um grafo, dizemos que  $\alpha$  **liga** os vértices  $v$  e  $w$ , ou que **incide** em  $v$  (e em  $w$ ).



$$V = \{a, b, c, d, e\}$$
$$E = \{(a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e)\}$$

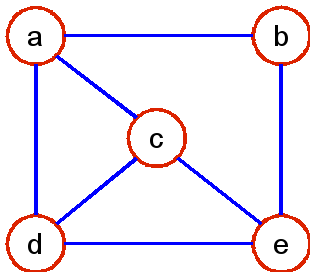
# Adjacência e Grau

- **Vértices adjacentes:** vértices conectados por uma aresta.



# Adjacência e Grau

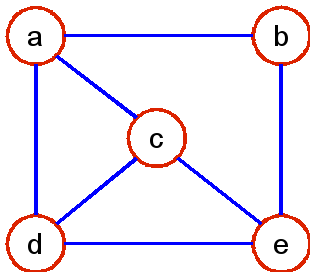
- **Vértices adjacentes:** vértices conectados por uma aresta.
- **Grau** de um vértice: número de vértices adjacentes





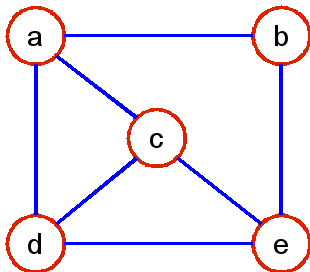
# Adjacência e Grau

- **Vértices adjacentes:** vértices conectados por uma aresta.
  - **Grau** de um vértice: número de vértices adjacentes
- 
- Qual a soma dos graus de todos os vértices?



# Adjacência e Grau

- **Vértices adjacentes:** vértices conectados por uma aresta.
  - **Grau** de um vértice: número de vértices adjacentes
- 
- Qual a soma dos graus de todos os vértices?  
O dobro de números de arestas



Existem vários meios para se representar um grafo. Três estruturas são mais comuns:

- Lista de Arcos
- Lista de Adjacência
- Matriz de Adjacência
- Matriz de Incidência

# Estruturas de Dados: uma proposta em C

```
// Os vértices (0,1,...,V-1) são representados por um tipo Vertex.
#define Vertex int

// Uma estrutura do tipo ARC representa um arco
// com ponta inicial v e ponta final w.
typedef struct {
    Vertex v;
    Vertex w;
} ARC;

// A função Arc recebe dois vértices v, w e devolve
// um arco com ponta inicial v e ponta final w.
ARC Arc (Vertex v, Vertex w) {
    ARC e;
    e.v = v;
    e.w = w;
    return e;
}
```

# Estruturas de Dados: uma proposta em C

```
// Os vértices (0,1,...,V-1) são representados por um tipo Vertex.
#define Vertex int

// Uma estrutura do tipo ARC representa um arco
// com ponta inicial v e ponta final w.
typedef struct {
    Vertex v;
    Vertex w;
} ARC;

// A função Arc recebe dois vértices v, w e devolve
// um arco com ponta inicial v e ponta final w.
ARC Arc (Vertex v, Vertex w) {
    ARC e;
    e.v = v;
    e.w = w;
    return e;
}
```

# Estruturas de Dados: uma proposta em C

```
// Os vértices (0,1,...,V-1) são representados por um tipo Vertex.
#define Vertex int

// Uma estrutura do tipo ARC representa um arco
// com ponta inicial v e ponta final w.
typedef struct {
    Vertex v;
    Vertex w;
} ARC;

// A função Arc recebe dois vértices v, w e devolve
// um arco com ponta inicial v e ponta final w.
ARC Arc (Vertex v, Vertex w) {
    ARC e;
    e.v = v;
    e.w = w;
    return e;
}
```

# Estruturas de Dados: uma proposta em C

```
// Os vértices (0,1,...,V-1) são representados por um tipo Vertex.
#define Vertex int

// Uma estrutura do tipo ARC representa um arco
// com ponta inicial v e ponta final w.
typedef struct {
    Vertex v;
    Vertex w;
} ARC;

// A função Arc recebe dois vértices v, w e devolve
// um arco com ponta inicial v e ponta final w.
ARC Arc (Vertex v, Vertex w) {
    ARC e;
    e.v = v;
    e.w = w;
    return e;
}
```

Um TAD digrafo deve fornecer algumas funções, por exemplo:

- `GRAPH initGraph(int)`
- `destroyGraph(GRAPH)`
- `insertArc(ARC)`
- `searchArc(ARC)`
- `removeArc(ARC)`
- `GRAPH copyGraph(GRAPH)`



Um TAD digrafo deve fornecer algumas funções, por exemplo:

- `GRAPH initGraph(int)`
- `destroyGraph(GRAPH)`
- `insertArc(ARC)`
- `searchArc(ARC)`
- `removeArc(ARC)`
- `GRAPH copyGraph(GRAPH)`

As funções podem mudar a depender da estrutura utilizada, e GRAPH também pode ser definido de várias formas

- 1 Recordando
- 2 Estruturas de Dados
  - Lista de Arcos
  - Lista de Adjacência
  - Matriz de Adjacência
  - Matriz de Incidência
- 3 Desempenho assintótico
- 4 Estruturas alternativas

- armazena cada arco presente no digrafo
- considera que todo vértice possui ao menos uma conexão
- implementação em **lista ligada** ou **arranjo**, cada elemento representa um arco

- 1 Recordando
- 2 Estruturas de Dados
  - Lista de Arcos
  - Lista de Adjacência
  - Matriz de Adjacência
  - Matriz de Incidência
- 3 Desempenho assintótico
- 4 Estruturas alternativas

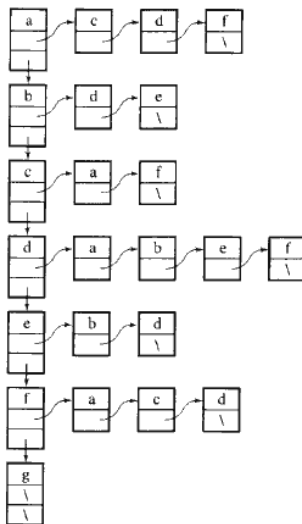
Especifica todos os vértices adjacentes a cada vértice do grafo.

Implementações possíveis.

- representação estrela: implementação em **tabela** (arranjo)
- **lista ligada** de vértices com ponteiro para o início de uma lista de adjacência (também uma lista ligada).
- **arranjo** de vértices com ponteiro para o início de uma lista de adjacência (lista ligada).

# Lista de Adjacência: estrela e lista ligada

a	c	d	f	
b	d	e		
c	a	f		
d	a	b	e	f
e	b	d		
f	a	c	d	
g				



Questões importantes:

- a implementação para um **digrafo** é um pouco diferente daquela para um **grafo**.
- se desejarmos que os vértices e arcos tenha um “nome” ou valor temos que pensar numa estrutura para armazenar esses valores
- a lista de adjacência é geralmente utilizada para a implementação de grafos esparsos.

## Lista de Adjacência: uma proposta em C

```
// Um 'node' representa um elemento na lista de adjacencia
//     e o tipo 'Link' e um ponteiro para a estrutura 'node'
typedef struct node *Link;
struct node {
    Vertex w;
    Link next;
};

// O digrafo armazena: o numero de vertices V, o numero de arcos A
//     e um ponteiro para a lista de adjacencia
struct digraph {
    int V;
    int A;
    Link *adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```



# Lista de Adjacência: uma proposta em C

```
// Um 'node' representa um elemento na lista de adjacencia
//     e o tipo 'Link' e um ponteiro para a estrutura 'node'
typedef struct node *Link;
struct node {
    Vertex w;
    Link next;
};

// O digrafo armazena: o numero de vertices V, o numero de arcos A
//     e um ponteiro para a lista de adjacencia
struct digraph {
    int V;
    int A;
    Link *adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

# Lista de Adjacência: uma proposta em C

```
// Um 'node' representa um elemento na lista de adjacencia
//     e o tipo 'Link' e um ponteiro para a estrutura 'node'
typedef struct node *Link;
struct node {
    Vertex w;
    Link next;
};

// O digrafo armazena: o numero de vertices V, o numero de arcos A
//     e um ponteiro para a lista de adjacencia
struct digraph {
    int V;
    int A;
    Link *adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

# Lista de Adjacência: uma proposta em C

```
// Um 'node' representa um elemento na lista de adjacencia
//     e o tipo 'Link' e um ponteiro para a estrutura 'node'
typedef struct node *Link;
struct node {
    Vertex w;
    Link next;
};

// 0 digrafo armazena: o numero de vertices V, o numero de arcos A
//     e um ponteiro para a lista de adjacencia
struct digraph {
    int V;
    int A;
    Link *adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

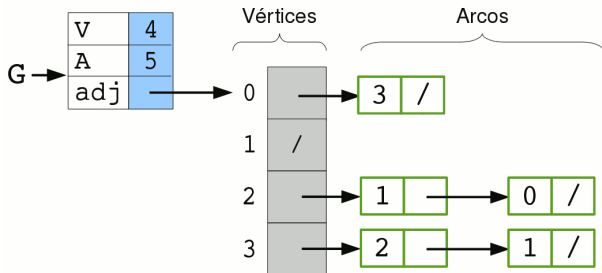
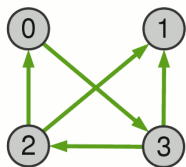
# Lista de Adjacência: uma proposta em C

```
// Um 'node' representa um elemento na lista de adjacencia
//     e o tipo 'Link' e um ponteiro para a estrutura 'node'
typedef struct node *Link;
struct node {
    Vertex w;
    Link next;
};

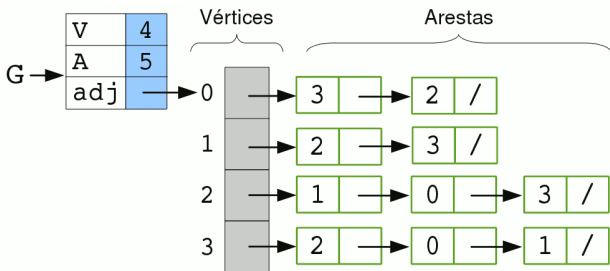
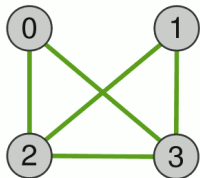
// O digrafo armazena: o numero de vertices V, o numero de arcos A
//     e um ponteiro para a lista de adjacencia
struct digraph {
    int V;
    int A;
    Link *adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

# Lista de Adjacência: arranjo de vértices e lista ligada de adjacência — representação de digrafo



# Lista de Adjacência: arranjo de vértices e lista ligada de adjacência — representação de grafo



- 1 Recordando
- 2 Estruturas de Dados
  - Lista de Arcos
  - Lista de Adjacência
  - **Matriz de Adjacência**
  - Matriz de Incidência
- 3 Desempenho assintótico
- 4 Estruturas alternativas

# Matriz de Adjacência

- matriz binária de tamanho  $V(G) \times V(G)$ , tal que cada entrada

$$d_{ij} = \begin{cases} 1, & \text{se existe } A(v_i, v_j) \\ 0, & \text{caso contrário.} \end{cases}$$



- matriz binária de tamanho  $V(G) \times V(G)$ , tal que cada entrada

$$d_{ij} = \begin{cases} 1, & \text{se existe } A(v_i, v_j) \\ 0, & \text{caso contrário.} \end{cases}$$

- pode ser generalizada para multigrafos modificando a definição:  
 $d_{ij}$  = número de arestas entre  $v_i$  e  $v_j$

- matriz binária de tamanho  $V(G) \times V(G)$ , tal que cada entrada

$$d_{ij} = \begin{cases} 1, & \text{se existe } A(v_i, v_j) \\ 0, & \text{caso contrário.} \end{cases}$$

- pode ser generalizada para multigrafos modificando a definição:  
 $d_{ij}$  = número de arestas entre  $v_i$  e  $v_j$
- pode ser generalizada para incluir laços permitindo a inclusão de valores na diagonal principal.

# Lista de Adjacência: uma proposta em C

```
// 0 digrafo armazena: o numero de vertices V, o numero de arcos A
//          e um ponteiro para a matriz de adjacencia
//          a matriz devera ser alocada dinamicamente
struct digraph {
    int V;
    int A;
    int **adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

# Lista de Adjacência: uma proposta em C

```
// 0 digrafo armazena: o numero de vertices V, o numero de arcos A
//          e um ponteiro para a matriz de adjacencia
//          a matriz devera ser alocada dinamicamente
struct digraph {
    int V;
    int A;
    int **adj;
};

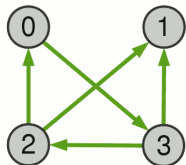
// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

# Lista de Adjacência: uma proposta em C

```
// O digrafo armazena: o numero de vertices V, o numero de arcos A
//          e um ponteiro para a matriz de adjacencia
//          a matriz devera ser alocada dinamicamente
struct digraph {
    int V;
    int A;
    int **adj;
};

// Digraph sera um ponteiro para a criacao de um digrafo
typedef struct digraph *Digraph;
```

# Matriz de Adjacência: representação de digrafo

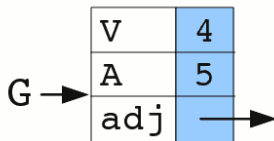
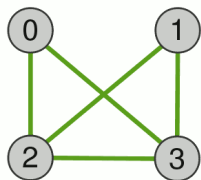


$G \rightarrow$

V	4
A	5
adj	$\rightarrow$

	0	1	2	3
0	0	0	0	<b>1</b>
1	0	0	0	0
2	<b>1</b>	<b>1</b>	0	0
3	0	<b>1</b>	<b>1</b>	0

# Matriz de Adjacência: representação de grafo



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

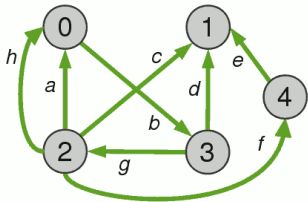
- 1 Recordando
- 2 Estruturas de Dados
  - Lista de Arcos
  - Lista de Adjacência
  - Matriz de Adjacência
  - Matriz de Incidência
- 3 Desempenho assintótico
- 4 Estruturas alternativas



- baseada na incidência de vértices e arestas, é uma matriz de tamanho  $V(G) \times A(G)$ , tal que cada entrada

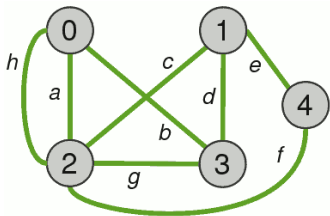
$$c_{ij} = \begin{cases} 1, & \text{se a aresta } a_j \text{ é incidente com o vértice } v_i \\ 0, & \text{caso contrário.} \end{cases}$$

# Matriz de Indicência: representação de digrafo



	a	b	c	d	e	f	g	h
0	<b>1</b>	<b>-1</b>	0	0	0	0	0	<b>1</b>
1	0	0	<b>1</b>	<b>1</b>	<b>1</b>	0	0	0
2	<b>-1</b>	0	<b>-1</b>	0	0	<b>-1</b>	<b>1</b>	<b>-1</b>
3	0	<b>1</b>	0	<b>-1</b>	0	0	<b>-1</b>	0
4	0	0	0	0	<b>-1</b>	<b>1</b>	0	0

# Matriz de Indicência: representação de grafo



	a	b	c	d	e	f	g	h
0	1	1	0	0	0	0	0	1
1	0	0	1	1	1	0	0	0
2	1	0	1	0	0	1	1	1
3	0	1	0	1	0	0	1	0
4	0	0	0	0	1	1	0	0





# Desempenho assintótico

Considerando um digrafo simples de  $n$  vértices e  $m$  arestas.  
Pela notação  $O()$ :

	Lista de Arcos	Lista de Adjacência	Matriz de Adjacência
Espaço	$n + m$	$n + m$	$n^2 + m$
Verificar arcos incidentes	$m$	$\text{grau}(v)$	$n$
Verificar adjacência	$m$	$\text{grau}(v)$	1
Inserir vértice	1	1	$n^2$
Inserir arco	1	1	1
Remover vértice	$m$	$\text{grau}(v)$	$n^2$
Remover arco	1	1	1

# Estruturas alternativas às sugeridas

- Montar uma *lista de arcos* e uma *lista de vértices*
- Cada elemento **arco** aponta para um par de elementos **vértice**
- Separadamente, gerar e manipular uma lista ou matriz de adjacência
- Cada elemento da lista ou matriz aponta para o arco correspondente àquela informação de adjacência, ou contém nulo se não houver conexão.

-  SEDGEWICK, R.  
**Algorithms in C: part 5**, 3.ed., Addison-Wesley, 2002.  
Graph ADT—Adjacency-Lists Representation (Seções 17.2, 17.3 e 17.4)
-  ZIVIANI, N.  
**Projeto de Algoritmos**, 3.ed. Cengage, 2004.  
(Capítulo 7)
-  TENEMBAUM, A.M. et al.  
**Estruturas de Dados Usando C**. Pearson Makron, 1995.  
Grafos e suas aplicações (Capítulo 8).
-  CORMEN, T. H. et al.  
**Algoritmos: teoria e prática**. Campus-Elsevier, 2002.  
Algoritmos de Grafos (parte IV).



KNUTH, D.

**The Art of Computer Programming:** fundamental algorithms, v.1.  
Addison-Wesley, 1969.

Basic Mathematical Properties of Trees (Seção 2.3.4)



FEOFILOFF, P.

Estruturas de Dados.

[http://www.ime.usp.br/~pf/algoritmos\\_para\\_grafos/aulas/  
digraphdatastructs.html](http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/digraphdatastructs.html)



FEOFILOFF, P.

Listas de adjacência.

[http://www.ime.usp.br/~pf/algoritmos\\_para\\_grafos/aulas/adjlists.html](http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/adjlists.html)



FEOFILOFF, P.

Matrizes de adjacência.

http:

[//www.ime.usp.br/~pf/algoritmos\\_para\\_grafos/aulas/adjmatrix.html](http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/adjmatrix.html)