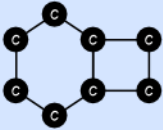
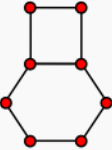
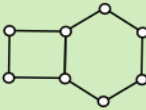



MAC328 Algoritmos em Grafos

Arnaldo Mandel

1º Semestre 2012

MAC0328 Algoritmos em Grafos

CHEMISTRY	SOCIAL NETWORKS	BIOLOGY	MATH
 <p>BENZOCYCLOBUTADIENE</p> <p>● CARBON ATOMS — σ-ELECTRON BONDS</p>	<p>spikedmath.com © 2011</p>  <p>● INDIVIDUALS — FRIENDSHIPS</p>	 <p>PPI (SUB)NETWORK OF A SIMPLE ORGANISM</p> <p>○ PROTEINS — INTERACTIONS</p>	<p>THEY LOOK THE SAME TO ME.</p> <p>LET'S CALL IT A GRAPH.</p> 

"MATHEMATICS IS THE ART OF GIVING THE SAME NAME TO DIFFERENT THINGS."
JULES HENRI POINCARÉ (1854-1912)

<http://spikedmath.com/250.html>

Administração

Página da disciplina:

www.ime.usp.br/~am/328

Livro:

- **PF** = Paulo Feofiloff,
Algoritmos para Grafos em C via Sedgwick
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

Os slides usados neste curso são derivados do material produzido pelo Prof. José Coelho de Pina Jr.

Professor e alunos agradecem desde já!

MAC0328

MAC0328 Algoritmos em grafos é:

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

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

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

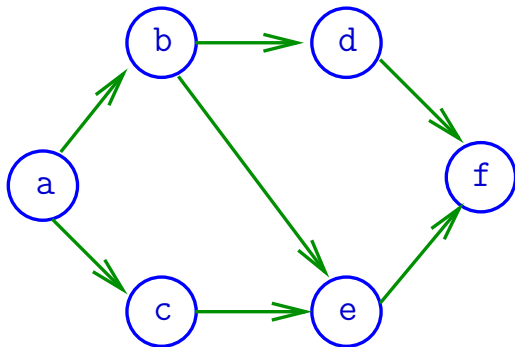
Digrafos

S 17.0, 17.1

Digrafos

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

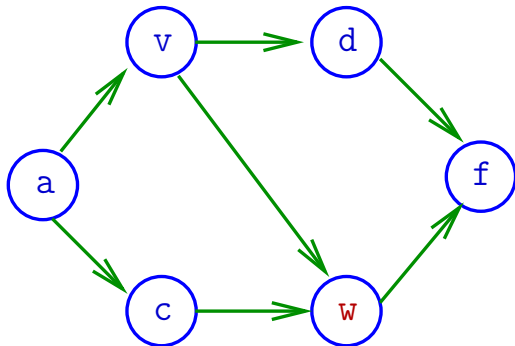
Exemplo: representação de um grafo



Arcos

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

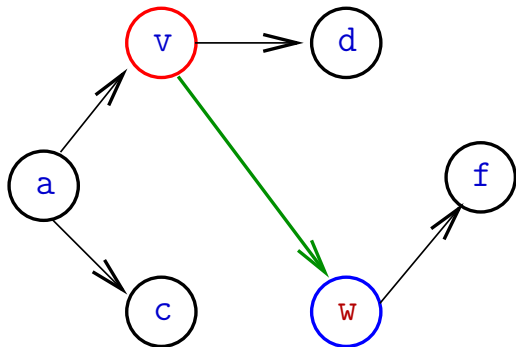
Exemplo: v e w são vértices e $v-w$ é um arco



Ponta inicial e final

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

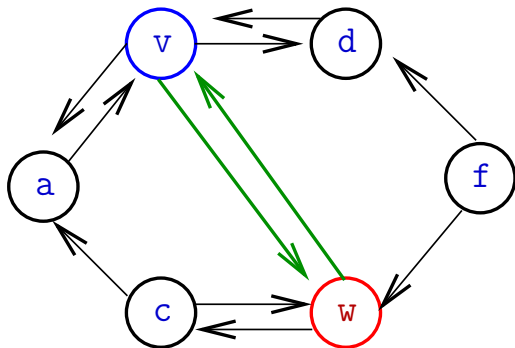
Exemplo: v é ponta inicial e w é ponta final de $v-w$



Arcos anti-paralelos

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

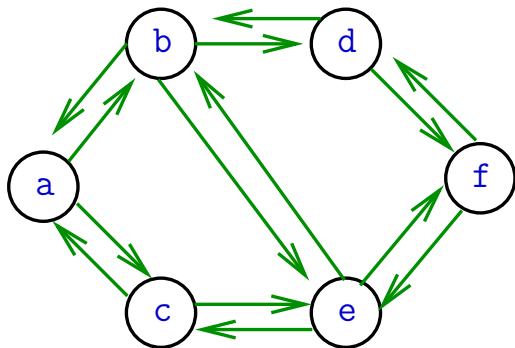
Exemplo: $v-w$ e $w-v$ são anti-paralelos



Digrafos simétricos

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

Exemplo: digrafo simétrico

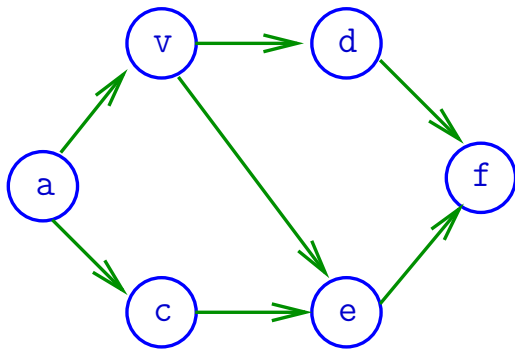


Graus de entrada e saída

grau de entrada de $v = n^{\circ}$ arcos com final v

grau de saída de $v = n^{\circ}$ arcos com início 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?

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

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

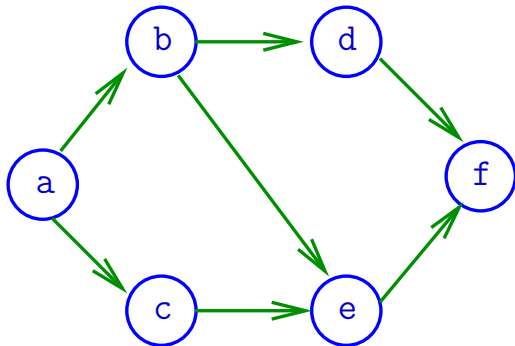
digrafo **denso** = tem “muitos” muitos arcos

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

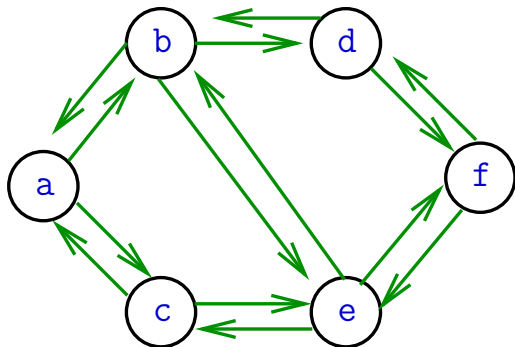
Grafos

S 17.0, 17.1

Grafos

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

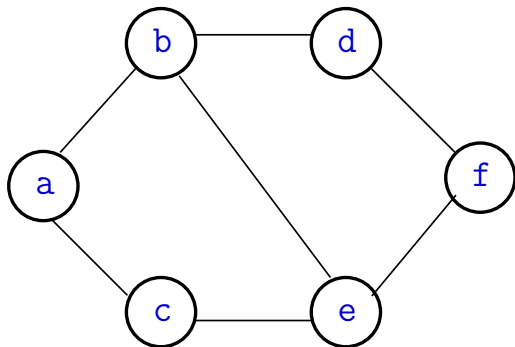
Exemplo: um grafo



Grafos

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

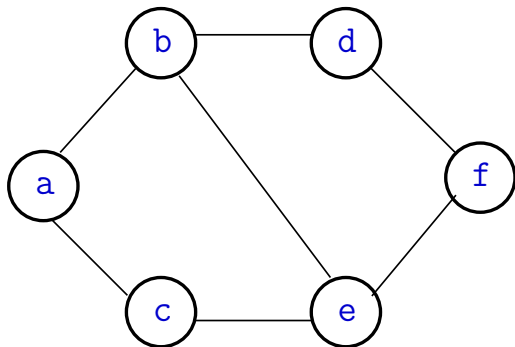
Exemplo: representação usual



Arestas

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

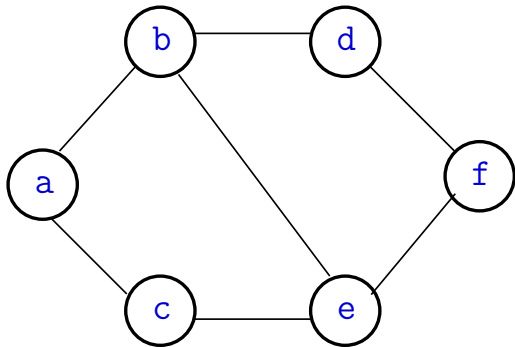
Exemplo: $b-a$ e $a-b$ são a **mesma** aresta



Especificação

Grafos podem ser especificados através de sua lista de arestas

Exemplo:



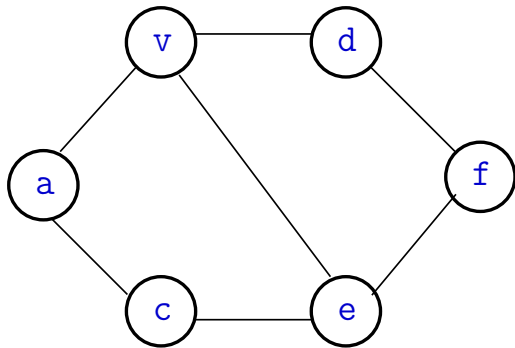
f-d
b-d
c-a
e-b
e-f
a-b

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?

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

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

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

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`.

```
#define Edge Arc
```

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**.

```
#define EDGE ARC
```

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 (**agora**)
- vetor de listas de adjacência (**próximas aulas**)

Há várias outras maneiras, como, por exemplo

- matriz de incidência

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

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

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 i, j;  
1     int **m = malloc(r * sizeof(int *));  
2     for (i = 0; i < r; i++)  
3         m[i] = malloc(c * sizeof(int));  
4     for (i = 0; i < r; i++)  
5         for (j = 0; j < c; j++)  
6             m[i][j] = 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 $\Theta(r \cdot c)$.

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;  
}
```

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;  
}
```