

# Grafos – Aula 3

## **Roteiro**

- Representando grafos
- Matriz e lista
- Operações básicas
- Tempos de acesso

# Grafo

## ■ Grafo $G=(V, E)$

- $V$  = conjunto de vértices, cada vértice representado por um número natural

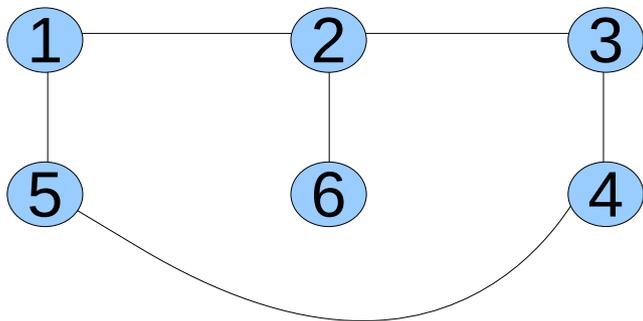
- $E$  = conjunto de arestas (pares não-ordenados, ou pares ordenados)

representação  
matemática  
de grafos

## ■ Exemplo

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

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



Representação  
visual de grafos

# Representando Grafos

- Como representar grafos no computador?

## Estrutura de dados

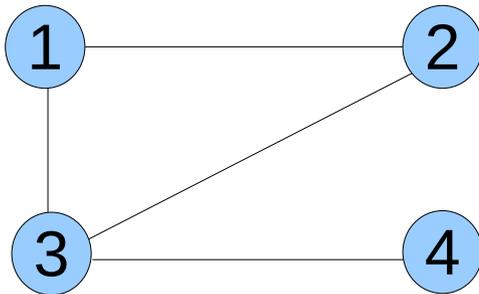
- Estruturas de dados fundamentais
  - vetor
  - matriz
  - lista
  - tabela hash (dicionário)

# Representação via Matriz

- Como representar utilizando matrizes?
- **Idéia:** associar vértices às linhas e colunas da matriz
  - elemento da matriz indica se há aresta
- **Matriz de adjacência**
- Matriz  $n \times n$  ( $n$  é número de vértices)
  - $a_{ij} = 1$  , se existe aresta entre vértices  $i$  e  $j$
  - $a_{ij} = 0$  , caso contrário.

# Matriz de Adjacência

## Exemplo



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

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	1
3	1	1	0	1	0
4	0	0	1	0	1
5	0	1	0	1	0

?

**Algumas  
propriedades?**

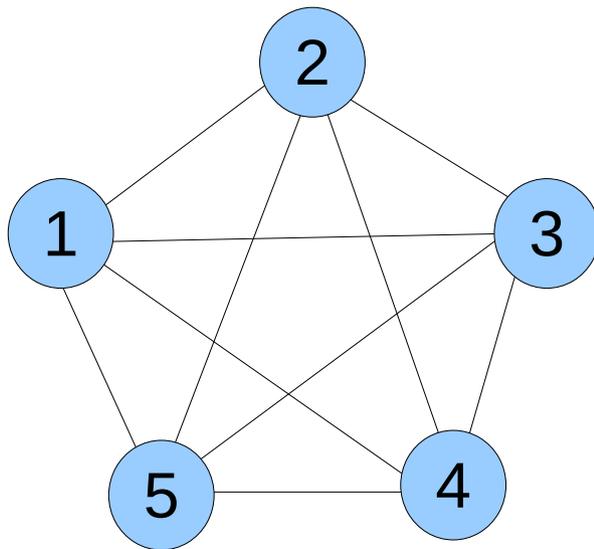
# Representação via Matriz

- **Matriz de adjacência**

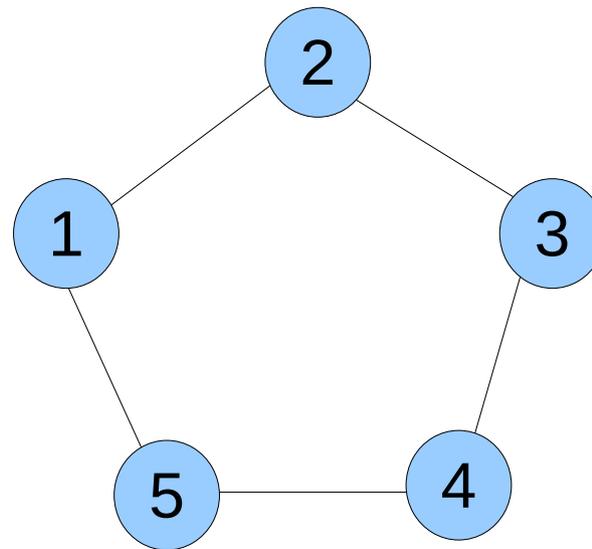
- Grafo completo?

- Grafo em ciclo?

$K_5$



$C_5$

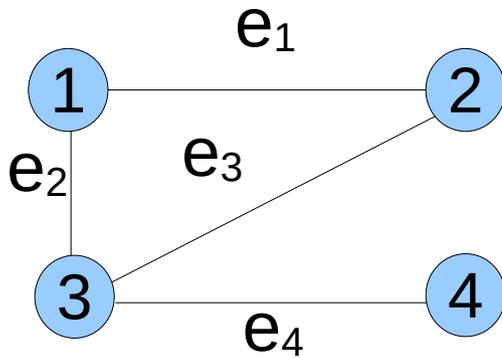


# Matriz de Incidência

- **Idéia:** associar vértices às linhas e arestas às colunas
  - elemento da matriz indica se aresta incide sobre o vértice
  - precisa enumerar as arestas do grafo
- **Matriz de incidência**
- Matriz  $n \times m$  ( $n$  vértices,  $m$  arestas)
  - $a_{ij} = 1$  , se vértice  $i$  incide sobre aresta  $j$
  - $a_{ij} = 0$  , caso contrário.

# Matriz de Incidência

## Exemplo



	$e_1$	$e_2$	$e_3$	$e_4$
1	1	1	0	0
2	1	0	1	0
3	0	1	1	1
4	0	0	0	1

	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$
1	1	1	0	0	0	1
2	1	0	1	1	1	0
3	0	1	0	1	0	0
4	0	0	0	0	1	0
5	0	0	1	0	0	1

→ ?

**Algumas  
propriedades?**

# Custo de Memória

- Quanto espaço (memória) precisamos, em função de  $n$  (vértices) e  $m$  (arestas)?
- Matriz de adjacência ←  $n^2$  posições
- Matriz de incidência ←  $nm$  posições
- Quantos bytes (ou bits) por posição?
  - 1 bit (no mínimo), 1 byte em geral
- Exemplos
  - $n=10^4 \rightarrow 10^8$  posições  $\rightarrow 10^8$  bytes = 100MB
  - $n=10^5 \rightarrow 10^{10}$  posições  $\rightarrow 10^{10}$  bytes = 10GB
  - $n=10^6 \rightarrow 10^{12}$  posições  $\rightarrow 10^{12}$  bytes = 1TB

**Complexidade quadrática, não escala!**

# Grafos Esparsos

- Considere grafos grandes e esparsos
  - *grande*: muitos vértices
  - *esparso*: relativamente poucas arestas
- Exemplo no Facebook:  $n=10^9$ ,  $m=10^{11}$ 
  - grau médio =  $2m/n = 200$
- Matriz formada principalmente de zeros!

**Grande consumo de memória (desnecessário)!**

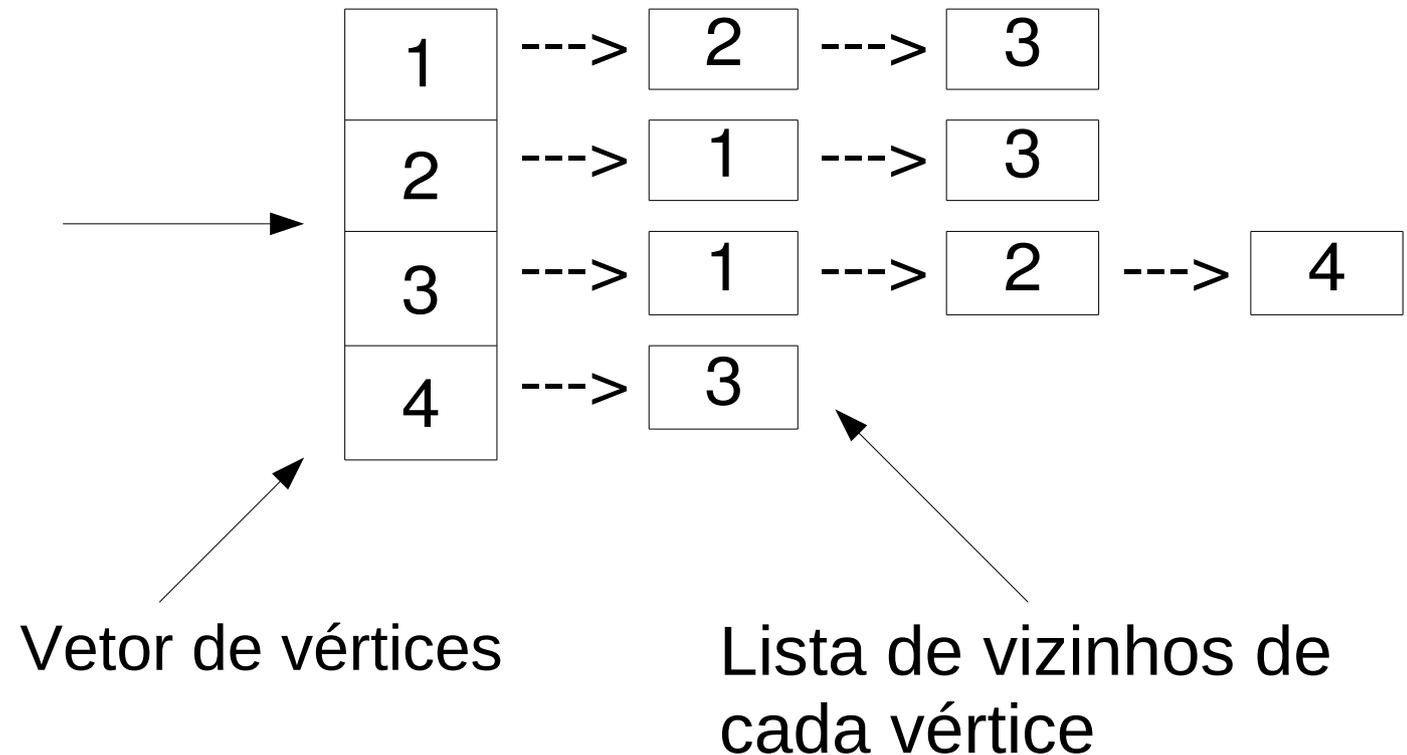
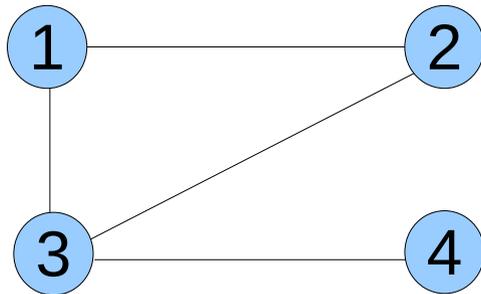
- Como resolver este problema?

# Representação via Listas

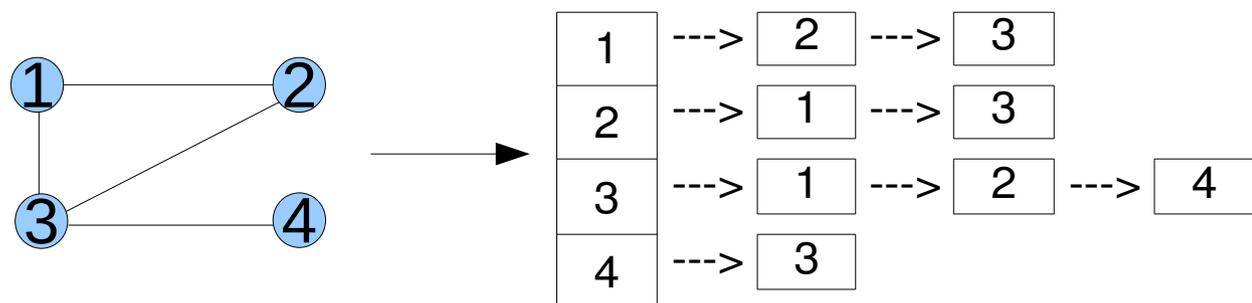
- **Idéia:** associar a cada vértice uma lista de vértices adjacentes (lista de vizinhos)
- **Lista de adjacência**
- Vértices associados a um vetor de dimensão  $n$  (número de vértices no grafo)
- Cada vértice possui uma lista de vértices adjacentes

# Lista de Adjacência

## ■ Exemplo



# Custo de Memória



- Cada aresta ocorre duas vezes na estrutura de dados
  - ex. (1,2) está na lista de 1 e na lista de 2
- Memória necessária será proporcional a 2 vezes o número de arestas
  - manter índice do vértice, mais ponteiro da lista
- Muito eficiente
  - grafo com 2 milhões de vértices e 100 milhões de arestas  $\rightarrow 2 * 10^7 * 12 \text{ bytes} = 240\text{M bytes}$

# Vantagens e Desvantagens

- Vantagem: listar todos os vizinhos de um vértice
  - basta percorrer a lista de vizinhos
- Desvantagem: decidir se um vértice é vizinho de outro
  - precisa percorrer toda a lista no pior caso

**Tempo de acesso depende da operação**

# Tempo de Acesso

## ■ Vetor x Lista (código em Java)

```
int n = 1000000;  
Vector vec = new Vector(n);
```

```
for(int i = 1; i <= n; i=i+1)  
    vec.add((String) i);
```

```
String p = vec[n-1];
```

“constante”: não depende de n

```
int n=1000000;  
LinkedList ll = new LinkedList();
```

```
for(int i = 1; i <= n; i=i+1)  
    ll.add((String) i);
```

```
String p = ll.get(n-1);
```

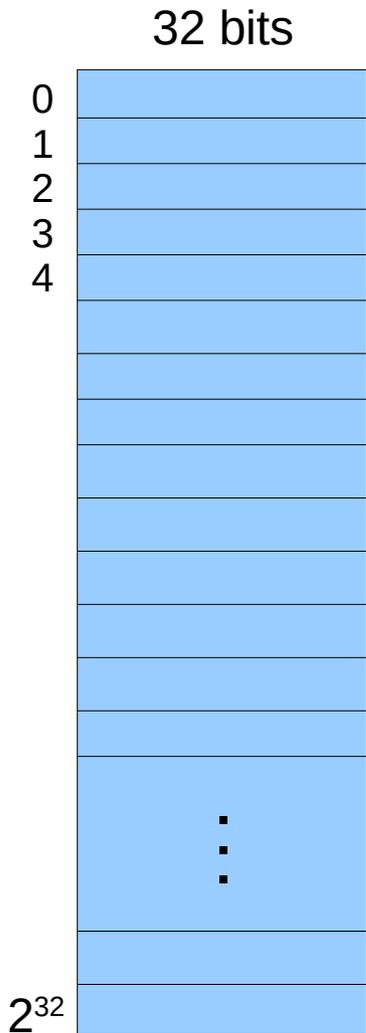
“linear”: depende linearmente de n, 1000000 vezes maior!

■ Quanto tempo para executar última linha?

**Por que?**

# Vetor x Lista

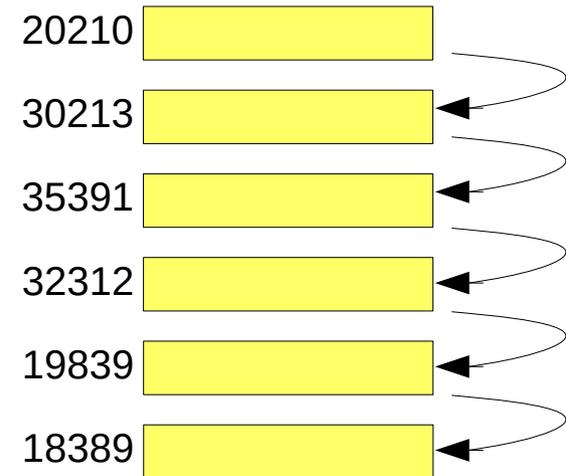
- Memória do computador organizada como uma matriz



**Vetor:** alocação contígua em memória



**Lista:** alocação elemento a elemento, ponteiro para próximo



- Como “chegar” no k-ésimo elemento?
  - qual endereço do k-ésimo elemento?

Capacidade: 4GB

# Vetor x Lista

■ Como chegar no k-ésimo elemento?

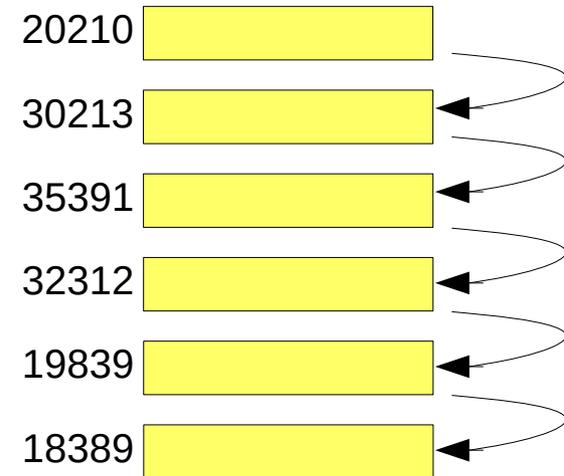
## Vetor



- Variável *vec* tem o endereço base, onde começa o vetor na memória
- Cada elemento tem tamanho (*L*)
- Endereço do elemento *k*:  
 $base + (k-1)*L$

**Constante!**

## Lista



- Variável *ll* tem o endereço do primeiro elemento
- Cada elemento tem endereço do próximo elemento
- Endereço do elemento *k*:  
tem que percorrer a lista!

**Proporcional a *k*!**

# Vantagens/Desvantagens

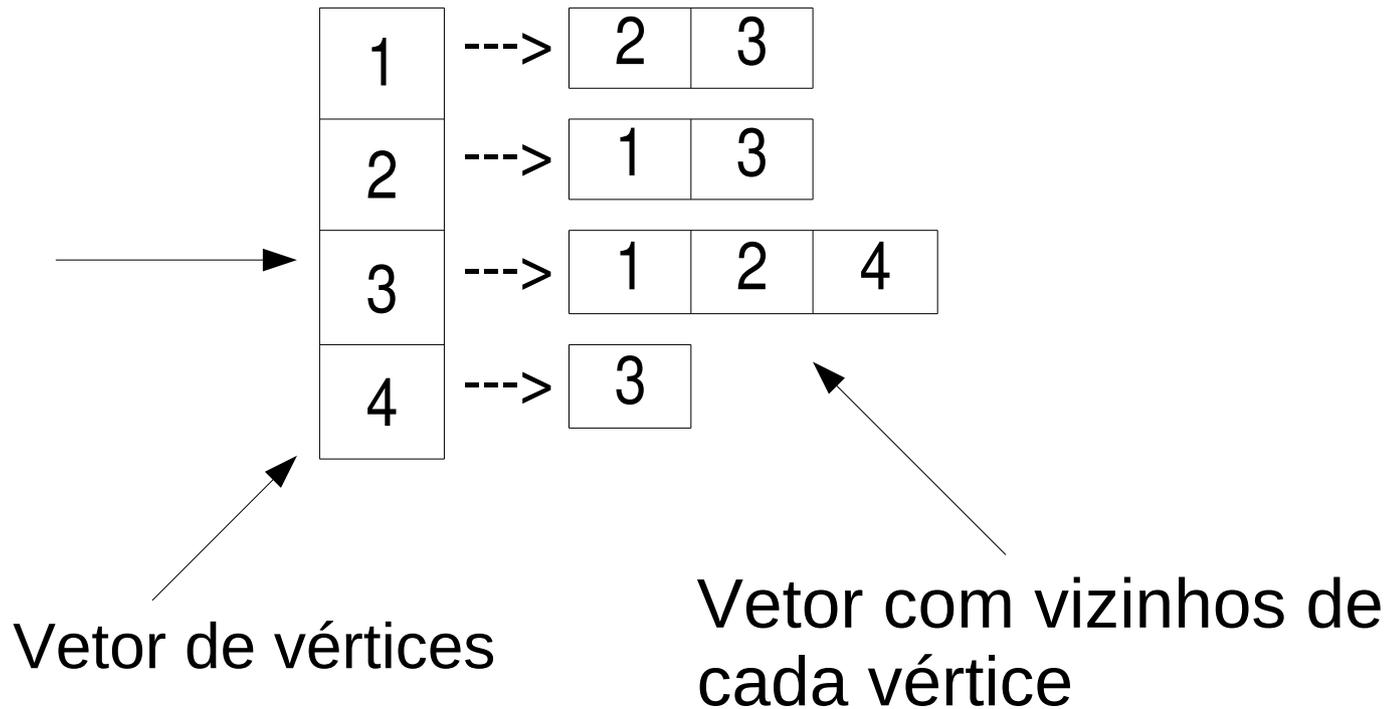
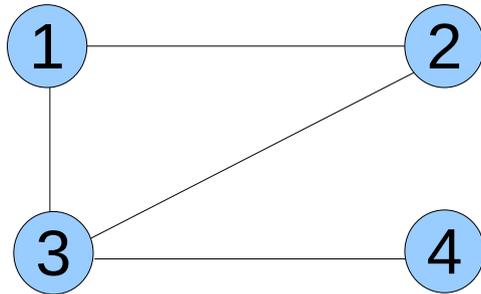
	Matriz	Lista
■ Tempo de execução		
■ Inserir aresta?	$O(1)$	$O(1)$
■ Remover aresta?	$O(1)$	$O(g_{\max})$
■ Testar adjacência ( $v_1$ e $v_2$ são vizinhos)?	$O(1)$	$O(g_{\max})$
■ Listar vizinhos de $v$ ?	$O(n)$	$O(g_{\max})$
■ Descobrir vértice de maior grau?	$O(n^2)$	$O(m)$

$O(1)$  = tempo constante

$g_{\max}$  = maior grau do grafo

# Vetor de Adjacência

## Exemplo



- Parecida com lista de adjacência
- Mesma complexidade (memória e tempo)
  - vantagens na prática

**Por que?**

# Tabela Hash

- Tabela hash (dicionário) com as aretas do grafo
  - aretas são a chave, valor é uma constante
- Aresta é um par não ordenado
  - ordem importa para a chave
  - convenção na chave: “menor maior”
- Exemplo:
  - $(1,2) \rightarrow$  “1 2” chave,  $(4,2) \rightarrow$  “2 4” chave
  - `arestas.add(“1 2”) = 1`
  - `If (arestas.present(“2 4”)) ...`
- Acesso a tabela hash em tempo constante
- Memória cresce com aresta

# Vantagens/Desvantagens

	Dicion.	Lista
■ Tempo de execução		
■ Inserir aresta?	$O(1)$	$O(1)$
■ Remover aresta?	$O(1)$	$O(g_{\max})$
■ Testar adjacência ( $v_1$ e $v_2$ são vizinhos)?	$O(1)$	$O(g_{\max})$
■ Listar vizinhos de $v$ ?	$O(n)$	$O(g_{\max})$
■ Descobrir vértice de maior grau?	$O(n^2)$	$O(m)$

$O(1)$  = tempo constante

$g_{\max}$  = maior grau do grafo

# Matriz x Lista x Hash

- Qual é a estrutura mais adequada (ou mais eficiente)?

**Depende do algoritmo!**

- Memória x Tempo: eficiente em que aspecto?
  - matriz utiliza mais memória que tabela hash que utiliza mais memória que lista
  - matriz ou lista ou tabela hash podem levar menos tempo

**Implementarão matriz e lista!**