Capítulo II

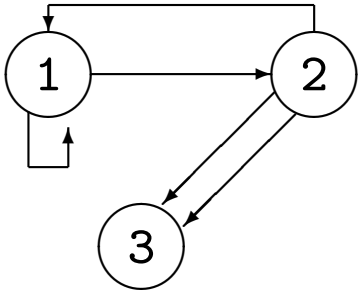
Noções Básicas de Grafos

Grafo $G = (V, A)$

V conjunto de **vértices** ou **nós**

A coleção de **arcos** ou **arestas**

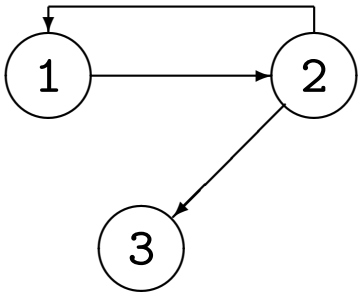
Grafo Genérico — Com arcos paralelos ou com lacetes.



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

$A = <(1,1),\ (1,2),\ (2,1),\ (2,3),\ (2,3)>$

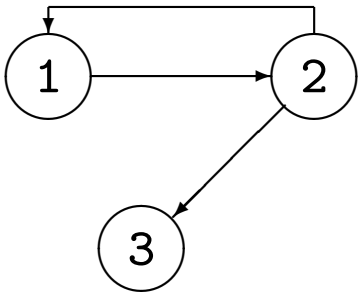
Grafo Simples — $A \subseteq V \times V$ e sem lacetes.



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

$A = \{(1,2),\ (2,1),\ (2,3)\}$

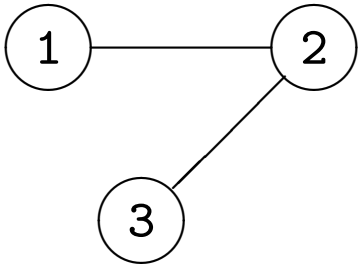
Grafo Orientado — Os arcos têm sentido.



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

$A = \{(1,2),\ (2,1),\ (2,3)\}$

Grafo Não Orientado — Os arcos não têm sentido único.



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

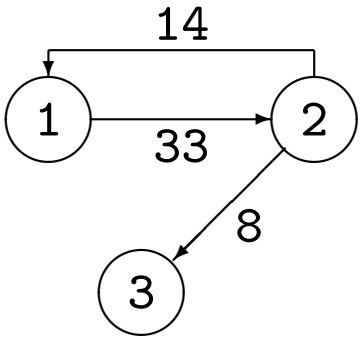
$A = \{(1,2),\ (2,3)\}$

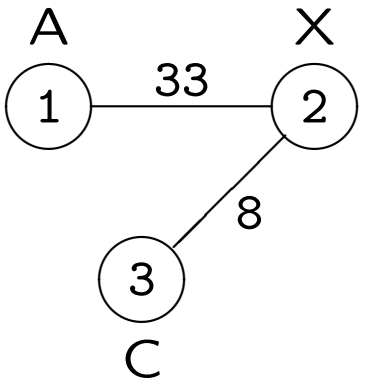
Condidera-se que $(\forall v, w \in V)\ \ (v,w) = (w,v)$.

Grafo Etiquetado (ou Pesado)

Os vértices, os arcos ou ambos têm

uma **etiqueta**, um **peso** ou um **custo**.

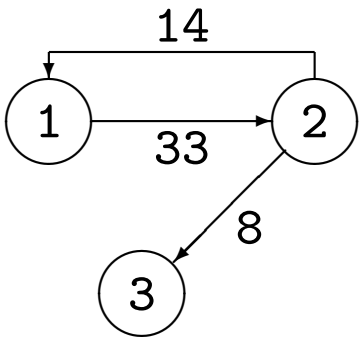




Caminho

É uma sequência não vazia de vértices $v_1, v_2, \ldots, v_n$ (com $n \geq 1$),

tal que, para qualquer $i = 1, 2, \ldots, n-1$: $(v_i, v_{i+1}) \in A$.



Caminho: 2, 1, 2, 3

Comprimento: 3

Comprimento Pesado: 55

Comprimento do Caminho: o número de arcos do caminho $(n-1)$.

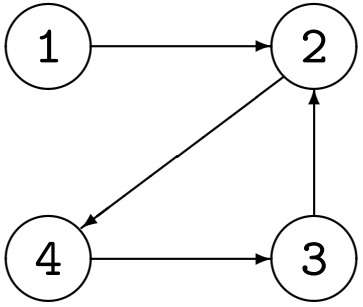
Comprimento Pesado ou Custo do Caminho: a soma dos pesos (numéricos) dos arcos do caminho (num grafo pesado).

Caminho Simples: um caminho cujos vértices são todos diferentes, exceto, possivelmente, o primeiro e o último.

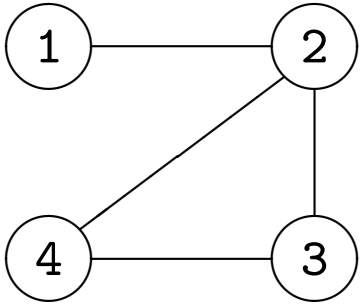
Ciclo ou Circuito

Num Grafo Orientado: um caminho de comprimento positivo onde o primeiro e o último vértices são iguais.

Num Grafo Não Orientado: um caminho de comprimento positivo, onde o primeiro e o último vértices são iguais, que não passa 2 vezes pelo mesmo arco.



Ciclo:
2, 4, 3, 2
Não é ciclo:
1



Ciclo:
2, 3, 4, 2
Não é ciclo:
1, 2, 1

Grafo Cíclico / Acíclico: um grafo com / sem ciclos.

Conectividade

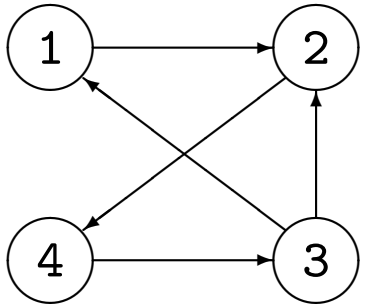
Grafo Fortemente Conexo: um **grafo orientado** tal que:

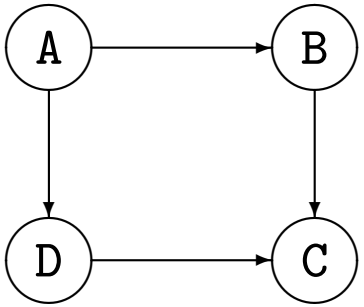
$(\forall v, w \in V)$ existe um caminho de v para w.

Grafo Fracamente Conexo: um **grafo orientado** tal que, ignorando o sentido dos arcos:

$(\forall v, w \in V)$ existe um caminho de v para w.

Grafo fortemente conexo





Grafo fracamente conexo

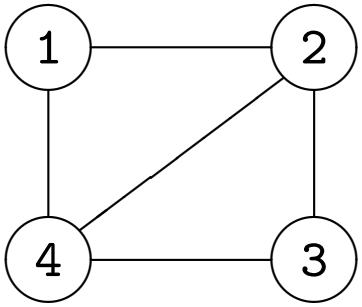
Grafo Conexo: um **grafo não orientado** tal que:

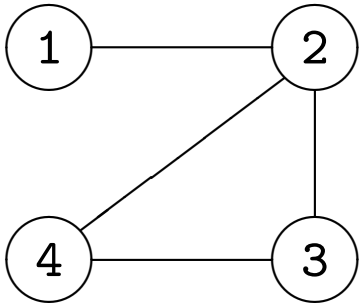
$(\forall v, w \in V)$ existe um caminho de v para w.

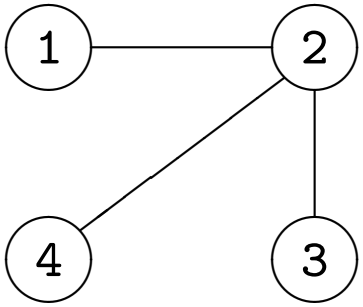
Sub-grafos e Árvores

Sub-grafo de (V, A): um grafo (V', A') tal que $V' \subseteq V$ e $A' \subseteq A$.

Sub-grafo de Cobertura de (V, A): um sub-grafo (V', A') de (V, A), com $V' = V$.







Árvore (livre): um **grafo não orientado**, conexo e acíclico.

Árvore de Cobertura de (V, A): Um sub-grafo de cobertura de (V, A) que é árvore.

Tipos Abstratos de Dados

Vértice, Arco, Grafo Não Orientado e Grafo Orientado

- As interfaces que se seguem introduzem os métodos usados na implementação (em pseudo-código) dos algoritmos de grafos estudados.

- Por questões de eficiência, não implementem estas interfaces: implementem o grafo da forma mais adequada, considerando os métodos que cada algoritmo requer, e implementem apenas esses métodos.

- Por exemplo, em geral, um vértice é um número inteiro (entre zero e número-total-de-vértices $-$ 1).

TAD Vértice

```
public interface Vertex
{
    // In practice, a vertex is an integer.
}
```

TAD Arco (com etiqueta do tipo L)

```
public interface Edge<L>
{
    // Returns the edge label.
    L label( );

    // Returns an array of length 2 with the edge end-vertices.
    Vertex[] endVertices( );

    // Returns the edge end-vertex that is distinct from the
    // specified vertex.
    Vertex oppositeVertex( Vertex endVertex );
}
```

TAD Qualquer Grafo (L) (1)

```
public interface AnyGraph<L>
{
    // Returns the number of vertices.
    int numVertices( );

    // Returns the number of edges.
    int numEdges( );

    // Returns an iterator of the vertices.
    Iterator<Vertex> vertices( );

    // Returns an iterator of the edges.
    Iterator<Edge<L>> edges( );
```

TAD Qualquer Grafo (L) (2)

```
    // Returns an arbitrary vertex.
    Vertex aVertex( );

    // Inserts and returns the edge (vertex1, vertex2) and
    // associates it with the specified label.
    Edge<L> insertEdge( Vertex vertex1, Vertex vertex2, L label );

    // Returns true iff there is an edge of the form (vertex1, vertex2).
    boolean edgeExists( Vertex vertex1, Vertex vertex2 );
}
```

TAD Grafo Não Orientado (L)

```
public interface UndiGraph<L> extends AnyGraph<L>
{
    // Returns the degree of the specified vertex.
    int degree( Vertex vertex );

    // Returns an iterator of the vertices adjacent to the specified
    // vertex.
    Iterator<Vertex> adjacentVertices( Vertex vertex );

    // Returns an iterator of the edges incident upon the specified
    // vertex.
    Iterator<Edge<L>> incidentEdges( Vertex vertex );
}
```

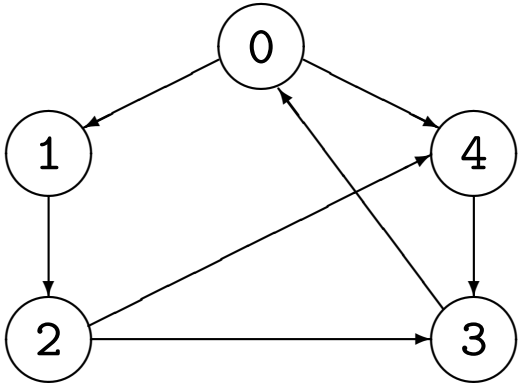
TAD Grafo Orientado (L) (1)

```
public interface Digraph<L> extends AnyGraph<L>
{
    // Returns the in-degree of the specified vertex.
    int inDegree( Vertex vertex );

    // Returns the out-degree of the specified vertex.
    int outDegree( Vertex vertex );

    // Returns an iterator of the vertices adjacent to the specified
    // vertex along incoming edges to it.
    Iterator<Vertex> inAdjacentVertices( Vertex vertex );

    // Returns an iterator of the vertices adjacent to the specified
    // vertex along outgoing edges from it.
    Iterator<Vertex> outAdjacentVertices( Vertex vertex );
```

TAD Grafo Orientado (L) (2)

```
    // Returns an iterator of the incoming edges to the specified
    // vertex.
    Iterator<Edge<L>> inIncidentEdges( Vertex vertex );

    // Returns an iterator of the outgoing edges from the specified
    // vertex.
    Iterator<Edge<L>> outIncidentEdges( Vertex vertex );
}
```

Matriz de Adjacências

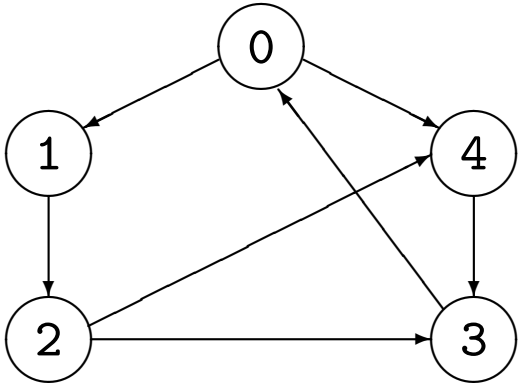


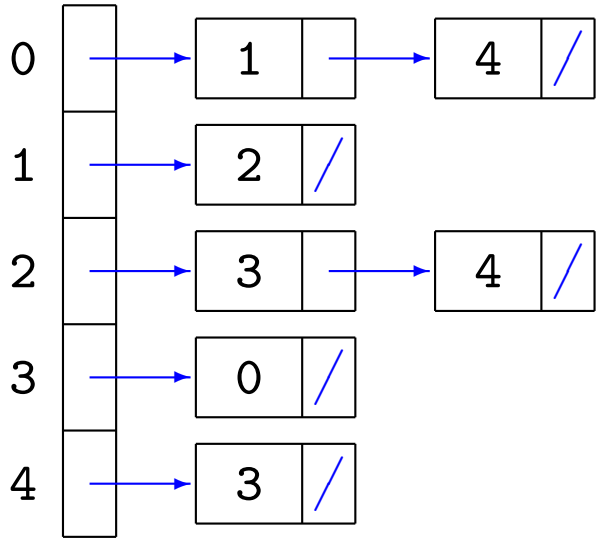
	0	1	2	3	4
0	0	1	0	0	1
1	0	0	1	0	0
2	0	0	0	1	1
3	1	0	0	0	0
4	0	0	0	1	0

Pesquisar	Arco (v_1, v_2)	$\Theta(1)$
Obter	Sucessores v	$\Theta(\|V\|)$
	Antecessores v	$\Theta(\|V\|)$
Memória Requerida		$\Theta(\|V\|^2)$

Listas Ligadas de Adjacências

de Sucessores

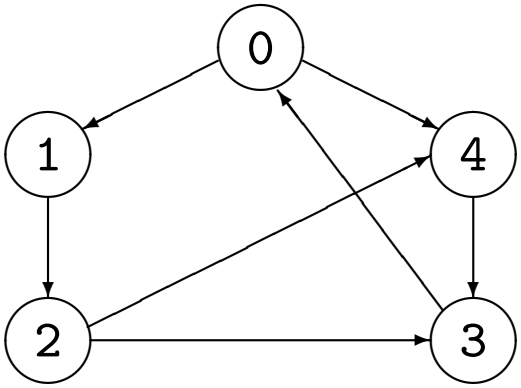


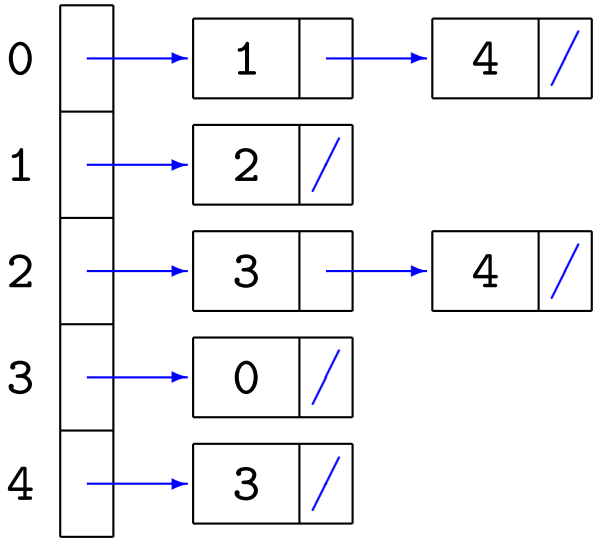


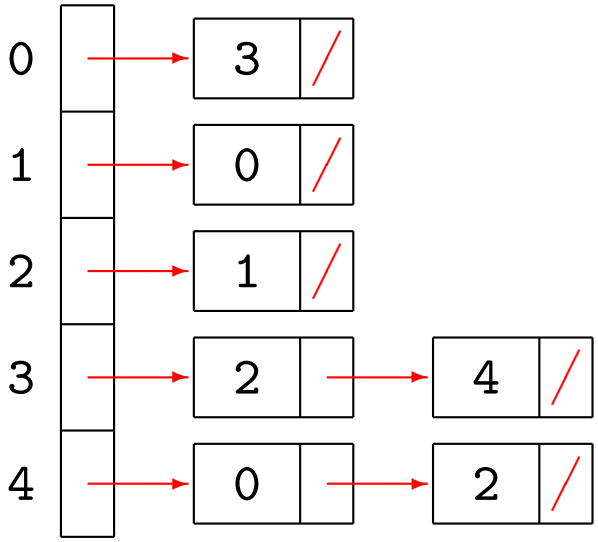
Pesquisar	Arco (v_1, v_2)	$O(\|\text{Suc}(v_1)\|)$
Obter	Sucessores v	$\Theta(\|\text{Suc}(v)\|)$
	Antecessores v	$O(\|V\| + \|A\|)$
Memória Requerida		$\Theta(\|V\| + \|A\|)$

Listas Ligadas de Adjacências

de Sucessores e de Antecessores







Pesquisar	Arco (v_1, v_2)	$O(\ \min(\lvert \text{Suc}(v_1) \rvert,\ \lvert \text{Ant}(v_2) \rvert)\ )$
Obter	Sucessores v	$\Theta(\lvert \text{Suc}(v) \rvert)$
	Antecessores v	$\Theta(\lvert \text{Ant}(v) \rvert)$
Memória Requerida		$\Theta(\lvert V \rvert + \lvert A \rvert)$