

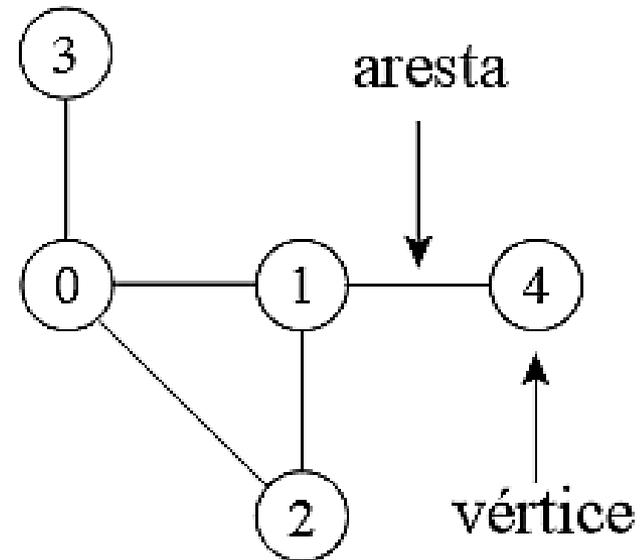
# Projeto e Análise de Algoritmos

## Aula 06 – Busca em Profundidade e Busca em Largura

Edirlei Soares de Lima  
<edirlei@iprj.uerj.br>

# Grafos (Revisão)

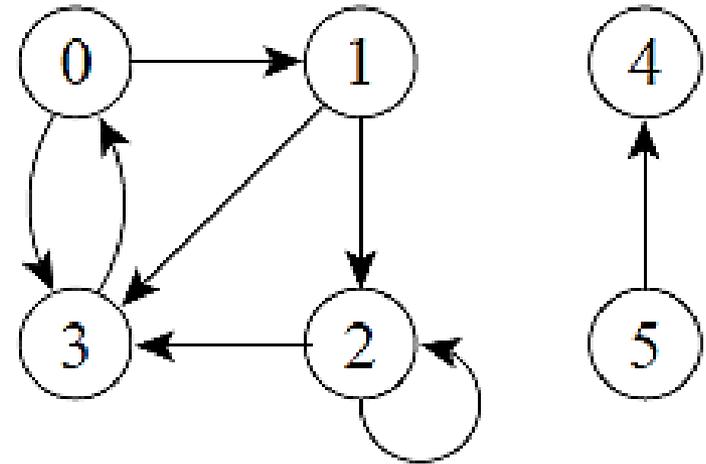
- **$G = (V, A)$** 
  - G: grafo;
  - V: conjunto de vértices;
  - A: conjunto de arestas;



# Grafos (Revisão)

- **Grafos Direcionados**

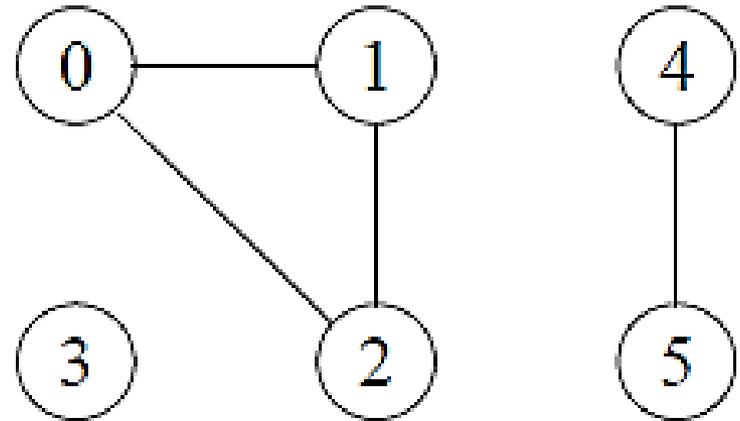
- Uma aresta  $(u, v)$  sai do vértice  $u$  e entra no vértice  $v$ .
  - O vértice  $v$  é adjacente ao vértice  $u$ .
- Podem existir arestas de um vértice para ele mesmo, chamadas de self-loops.



# Grafos (Revisão)

- **Grafos Não Direcionados**

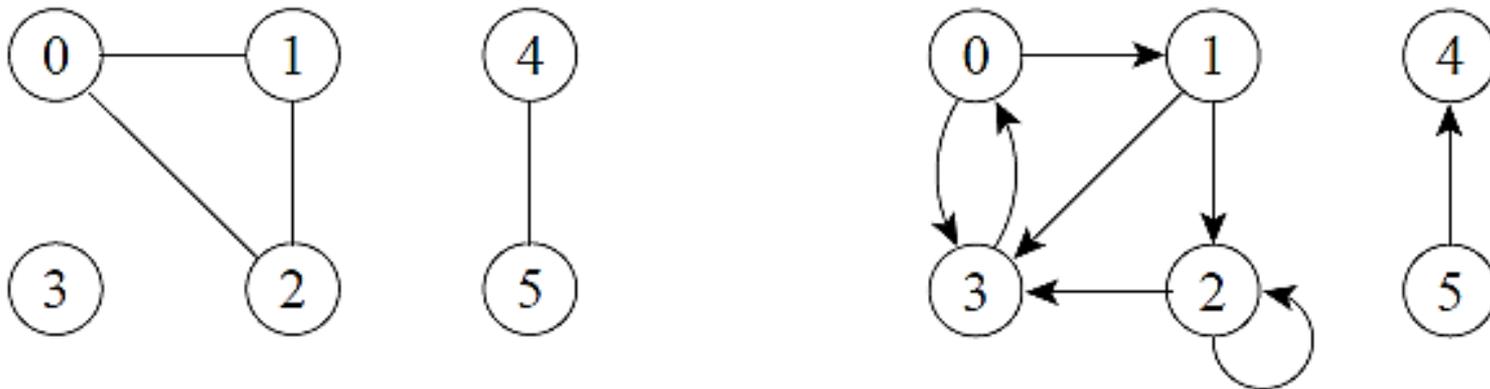
- As arestas  $(u, v)$  e  $(v, u)$  são consideradas como uma única aresta. A relação de adjacência é simétrica.
- Self-loops não são permitidos.



# Grafos (Revisão)

- **Grau de um Vértice**

- **Em grafos não direcionados:** é o número de arestas que incidem nele.
- **Em grafos direcionados:** é o número de arestas que saem dele (out-degree) mais o número de arestas que chegam nele (in-degree).
- Um vértice de grau zero é dito **isolado** ou **não conectado**.



# Grafos (Revisão)

- **Caminho entre Vértices**

- Um caminho de comprimento  $k$  de um vértice  $x$  a um vértice  $y$  em um grafo  $G = (V, A)$  é uma sequência de vértices  $(v_0, v_1, v_2, \dots, v_k)$  tal que  $x = v_0$  e  $y = v_k$ , e  $v_i \in V$  para  $i = 1, 2, \dots, k$ .
- O comprimento de um caminho é o número de arestas nele, isto é, o caminho contém os vértices  $v_0, v_1, v_2, \dots, v_k$  e as arestas  $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ .
- Se existir um caminho  $c$  de  $x$  a  $y$  então  $y$  é **alcançável** a partir de  $x$  via  $c$ .

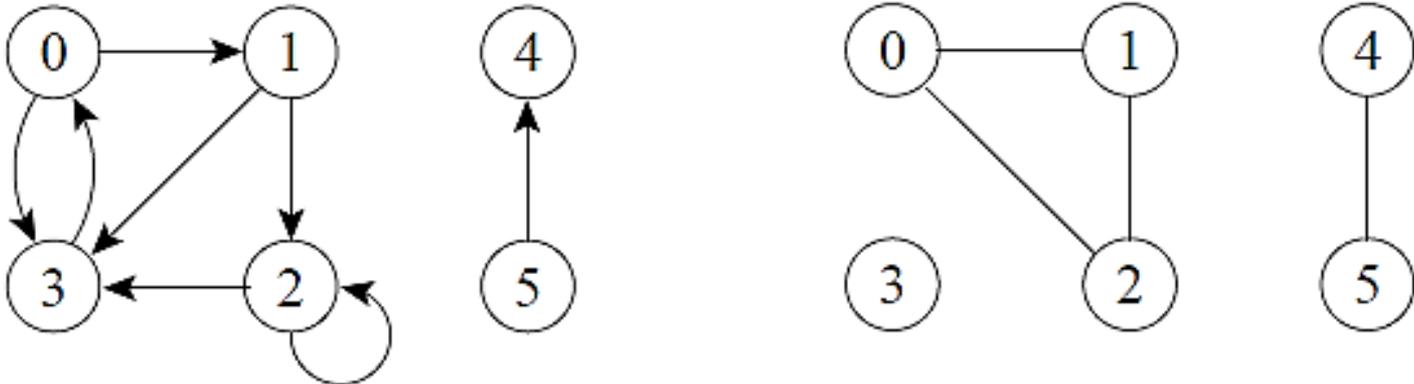
# Grafos (Revisão)

- **Ciclos**

- **Em um grafo direcionado:** um caminho  $(v_0, v_1, \dots, v_k)$  forma um ciclo se  $v_0 = v_k$  e o caminho contém pelo menos uma aresta.

- O *self-loop* é um ciclo de tamanho 1.

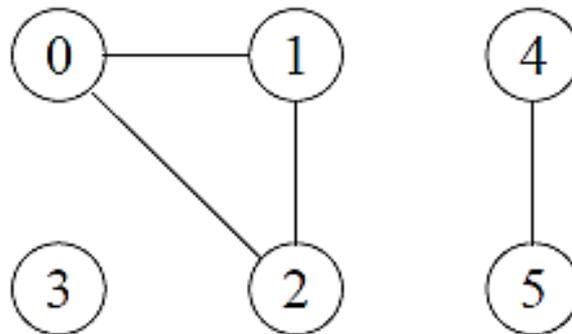
- **Em um grafo não direcionado:** um caminho  $(v_0, v_1, \dots, v_k)$  forma um ciclo se  $v_0 = v_k$  e o caminho contém pelo menos três arestas.



# Grafos (Revisão)

- **Componentes Conectados**

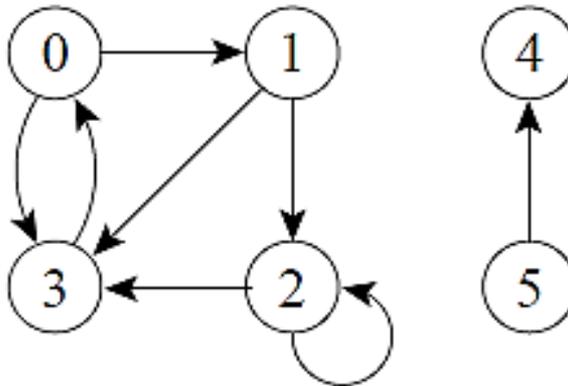
- Um grafo não direcionado é **conectado** se cada par de vértices está conectado por um caminho.
- Os **componentes conectados** são as porções conectadas de um grafo.
- Um grafo não direcionado é conectado se ele tem exatamente **um componente conectado**.



# Grafos (Revisão)

- **Componentes Fortemente Conectados**

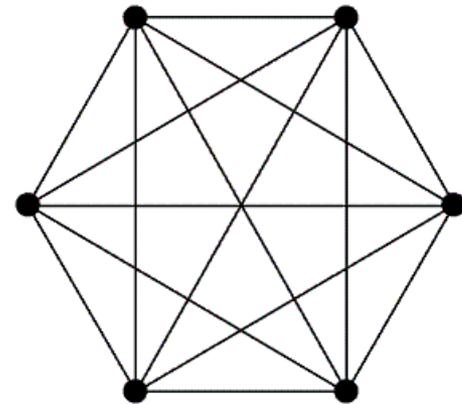
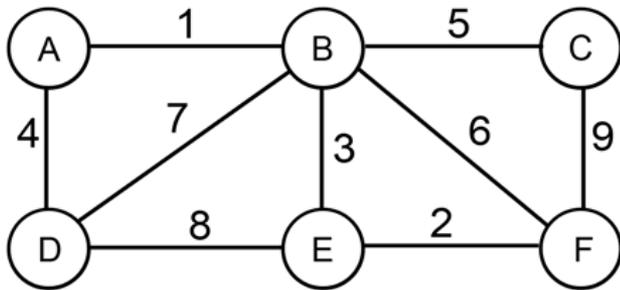
- Um grafo direcionado  $G = (V, A)$  é **fortemente conectado** se cada dois vértices quaisquer são alcançáveis a partir um do outro.
- Os **componentes fortemente conectados** de um grafo direcionado são conjuntos de vértices sob a relação “são mutuamente alcançáveis”.
- Um **grafo direcionado fortemente conectado** tem apenas um componente fortemente conectado.



# Grafos (Revisão)

- **Outras Classificações de Grafos**

- Um **grafo ponderado** possui pesos associados às arestas.
- Um **grafo completo** é um grafo não direcionado no qual todos os pares de vértices são adjacentes.

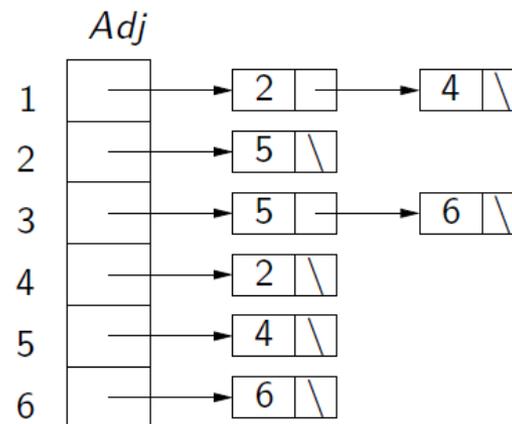
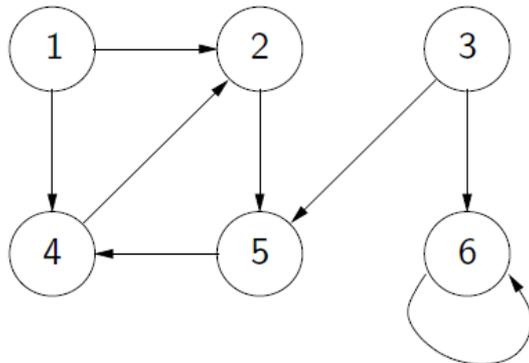


# Grafos (Revisão)

- **Representação de Grafos**

- **Lista de Adjacências:**

- Consiste em um vetor  $Adj$  com  $|V|$  listas de adjacências, uma para cada vértice  $v \in V$ .
- Para cada  $u \in V$ ,  $Adj[u]$  contém ponteiros para todos os vértices  $v$  tal que  $(u, v) \in A$ . Ou seja,  $Adj[u]$  consiste de todos os vértices que são adjacentes a  $u$ .



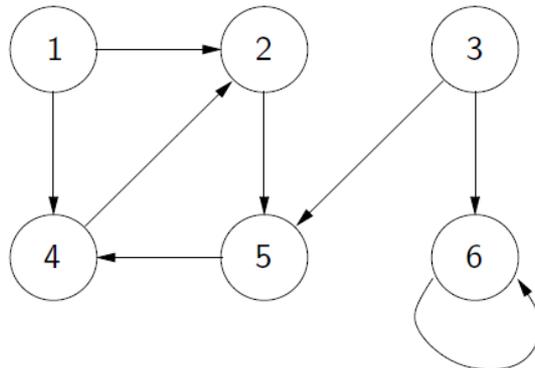
# Grafos (Revisão)

- **Representação de Grafos**

- **Matriz de Adjacências:**

- Para um grafo  $G = (V, E)$ , assumimos que os vértices são rotulados com números  $1, 2, \dots, |V|$ .
- A representação consiste de uma matriz  $A_{ij}$  de dimensões  $|V| \times |V|$ , onde:

$$a_{ij} = \begin{cases} 1 & \text{se } (i, j) \in A \\ 0 & \text{caso contrario} \end{cases}$$



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

# Busca em Largura

- A busca em largura é um dos algoritmos mais simples para exploração de um grafo.
  - Dados um grafo  $G = (V, E)$  e um vértice  $s$ , chamado de **fonte**, a busca em largura sistematicamente explora as arestas de  $G$  de maneira a visitar todos os vértices alcançáveis a partir de  $s$ .
- Expande a fronteira entre vértices descobertos e não descobertos uniformemente através da largura da **fronteira**.
  - O algoritmo descobre todos os vértices a uma distância  $k$  do vértice origem antes de descobrir qualquer vértice a uma distância  $k + 1$ .
- O grafo pode ser direcionado ou não direcionado.

# Busca em Largura

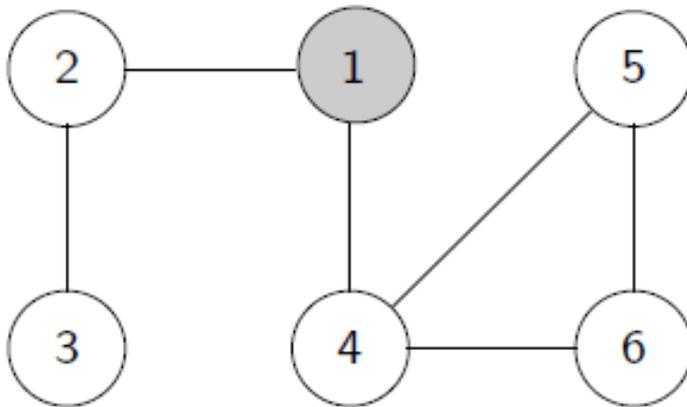
- **Algoritmo:**

- Para controlar a busca, o algoritmo da Busca em Largura pinta cada vértice na cor branca, cinza ou preto.
- Todos os vértices iniciam com a cor branca e podem, mais tarde, se tornar cinza e depois preto.
  - **Branca:** não visitado;
  - **Cinza:** visitado;
  - **Preto:** visitado e seus nós adjacentes visitados.

# Busca em Largura

```
BuscaEmLargura(G, s)
  for each  $u \in V[G]$ 
     $c[u] \leftarrow \text{white}$ 
     $d[u] \leftarrow \infty$ 
     $\pi[u] \leftarrow \text{NULL}$ 
   $c[s] \leftarrow \text{gray}$ 
   $d[s] \leftarrow 0$ 
   $Q \leftarrow \{s\}$  //Queue
  while  $Q \neq \emptyset$ 
     $u \leftarrow \text{head}[Q]$ 
    for each  $v \in \text{Adj}[u]$ 
      if  $c[v] = \text{white}$ 
         $c[v] \leftarrow \text{gray}$ 
         $d[v] \leftarrow d[u] + 1$ 
         $\pi[v] \leftarrow u$ 
        enqueue( $Q, v$ )
    dequeue( $Q$ )
   $c[u] \leftarrow \text{black}$ 
```

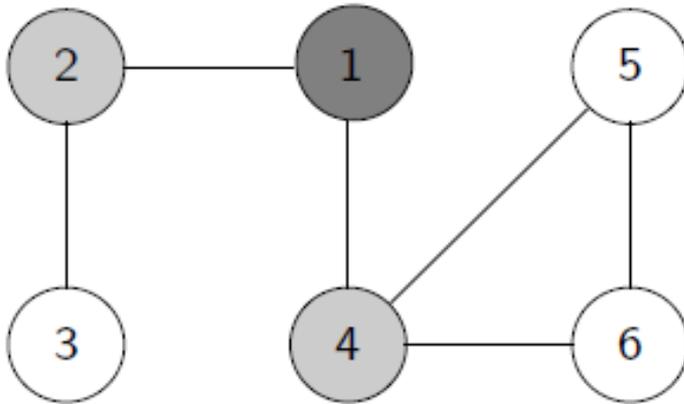
# Busca em Largura



```
for each  $u \in V[G]$   
   $c[u] \leftarrow \text{white}$   
   $d[u] \leftarrow \infty$   
   $\pi[u] \leftarrow \text{NULL}$   
 $c[s] \leftarrow \text{gray}$   
 $d[s] \leftarrow 0$   
 $Q \leftarrow \{s\}$  //Queue
```

	1	2	3	4	5	6
$d$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$\pi$	\	\	\	\	\	\
$c$	<i>g</i>	<i>w</i>	<i>w</i>	<i>w</i>	<i>w</i>	<i>w</i>
$Q$	1					

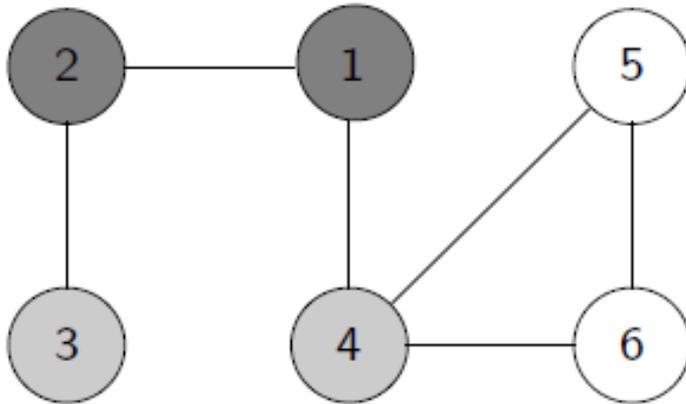
# Busca em Largura



```
u ← head[Q]
for each v ∈ Adj[u]
  if c[v] = white
    c[v] ← gray
    d[v] ← d[u] + 1
    π[v] ← u
    enqueue(Q, v)
dequeue(Q)
c[u] ← black
```

	1	2	3	4	5	6
$d$	0	1	$\infty$	1	$\infty$	$\infty$
$\pi$	\	1	\	1	\	\
$c$	<i>b</i>	<i>g</i>	<i>w</i>	<i>g</i>	<i>w</i>	<i>w</i>
$Q$	2	4				

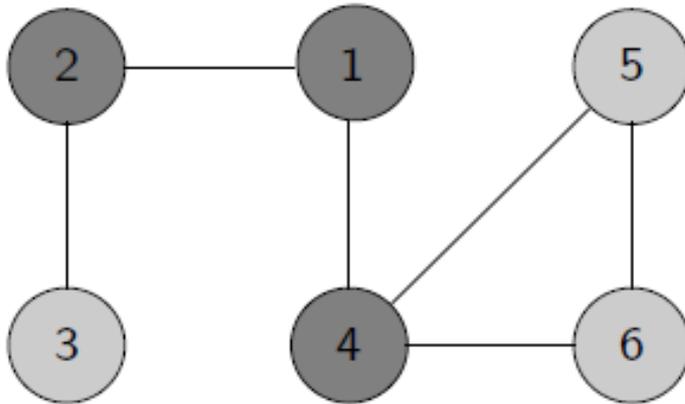
# Busca em Largura



```
u ← head[Q]
for each v ∈ Adj[u]
  if c[v] = white
    c[v] ← gray
    d[v] ← d[u] + 1
    π[v] ← u
    enqueue(Q, v)
dequeue(Q)
c[u] ← black
```

	1	2	3	4	5	6
$d$	0	1	2	1	$\infty$	$\infty$
$\pi$	\	1	2	1	\	\
$c$	<i>b</i>	<i>b</i>	<i>g</i>	<i>g</i>	<i>w</i>	<i>w</i>
$Q$	4	3				

# Busca em Largura



	1	2	3	4	5	6
$d$	0	1	2	1	2	2
$\pi$	\	1	2	1	4	4
$c$	<i>b</i>	<i>b</i>	<i>g</i>	<i>b</i>	<i>g</i>	<i>g</i>
$Q$	3	5	6			

```
u ← head[Q]
for each v ∈ Adj[u]
  if c[v] = white
    c[v] ← gray
    d[v] ← d[u] + 1
    π[v] ← u
    enqueue(Q, v)
dequeue(Q)
c[u] ← black
```

# Busca em Largura – Análise

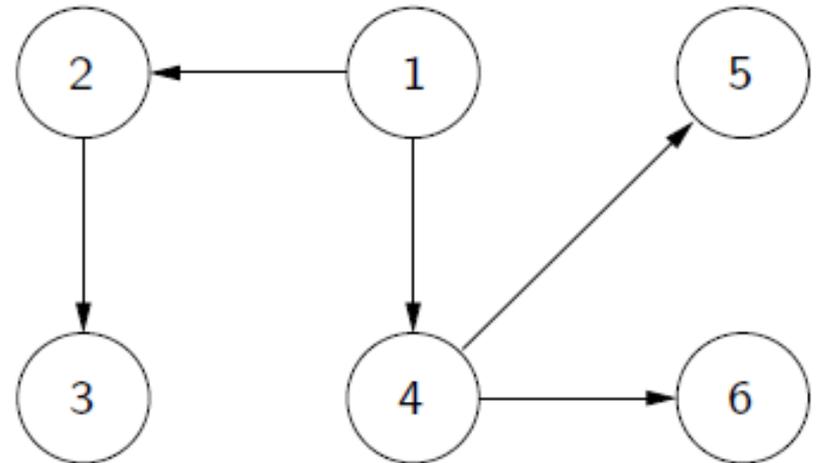
```
BuscaEmLargura(G, s)
  for each  $u \in V[G]$ 
     $c[u] \leftarrow \text{white}$ 
     $d[u] \leftarrow \infty$ 
     $\pi[u] \leftarrow \text{NULL}$ 
   $c[s] \leftarrow \text{gray}$ 
   $d[s] \leftarrow 0$ 
   $Q \leftarrow \{s\}$  //Queue
  while  $Q \neq \emptyset$ 
     $u \leftarrow \text{head}[Q]$ 
    for each  $v \in \text{Adj}[u]$ 
      if  $c[v] = \text{white}$ 
         $c[v] \leftarrow \text{gray}$ 
         $d[v] \leftarrow d[u] + 1$ 
         $\pi[v] \leftarrow u$ 
        enqueue( $Q, v$ )
    dequeue( $Q$ )
   $c[u] \leftarrow \text{black}$ 
```

- Cada vértice de  $V$  é colocado na fila  $Q$  no máximo uma vez:  $O(V)$ ;
- A lista de adjacência de um vértice qualquer de  $u$  é percorrida somente quando o vértice é removido da fila;
- A soma de todas as listas de adjacentes é  $O(A)$ , então o tempo total gasto com as listas de adjacentes é  $O(A)$ ;
- Enfileirar e desenfileirar tem custo  $O(1)$ ;
- Complexidade:  **$O(V + A)$**

# Busca em Largura

- A partir de  $\pi$  é possível reconstruir a **árvore da busca em largura**:

	1	2	3	4	5	6
$d$	0	1	2	1	2	2
$\pi$	\	1	2	1	4	4
$c$	<i>b</i>	<i>b</i>	<i>g</i>	<i>b</i>	<i>g</i>	<i>g</i>
$Q$	3	5	6			



# Busca em Profundidade

- A estratégia é buscar o vértice mais profundo no grafo sempre que possível:
  - As arestas são exploradas a partir do vértice  $v$  mais recentemente descoberto que ainda possui arestas não exploradas saindo dele.
- Quando todas as arestas adjacentes a  $v$  tiverem sido exploradas a busca anda para trás para explorar vértices que saem do vértice do qual  $v$  foi descoberto (**backtraking**).
- O algoritmo é a base para muitos outros algoritmos importantes, tais como verificação de grafos acíclicos, ordenação topológica e componentes fortemente conectados.

# Busca em Profundidade

- **Algoritmo:**

- Para controlar a busca, o algoritmo da Busca em Largura pinta cada vértice na cor branca, cinza ou preto.
- Todos os vértices iniciam com a cor branca e podem, mais tarde, se tornar cinza e depois preto.
  - **Branca:** não visitado;
  - **Cinza:** visitado;
  - **Preto:** visitado e seus nós adjacentes visitados.

# Busca em Profundidade

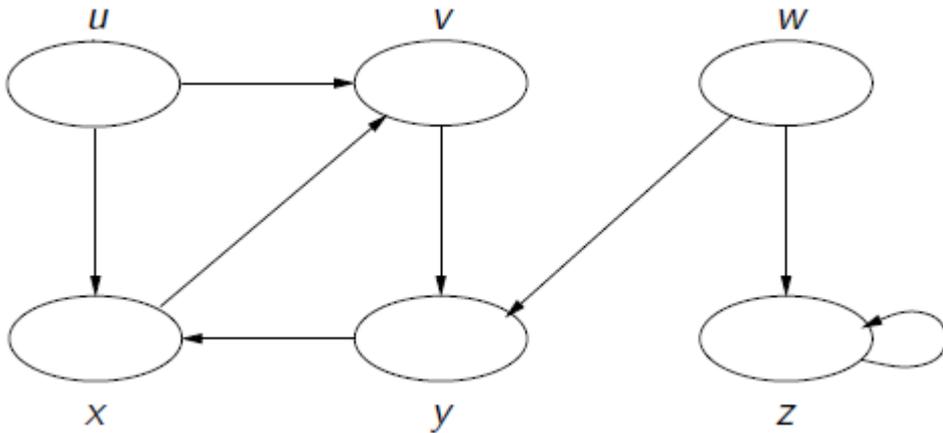
- **Algoritmo:**
  - A busca em profundidade também marca cada vértice com um *timestamp*.
  - Cada vértice tem dois *timestamps*:
    - **$d[v]$** : indica o instante em que  $v$  foi visitado (pintado com cinza);
    - **$f[v]$** : indica o instante em que a busca pelos vértices na lista de adjacência de  $v$  foi completada (pintado de preto).

# Busca em Profundidade

```
BuscaEmProfundidade(G)
  for each  $u \in V[G]$ 
     $c[u] \leftarrow \text{white}$ 
     $\pi[u] \leftarrow \text{NULL}$ 
  time  $\leftarrow 0$ 
  for each  $u \in V[G]$ 
    if  $c[u] = \text{white}$ 
      visita(u)
```

```
visita(u)
   $c[u] \leftarrow \text{gray}$ 
   $d[u] \leftarrow \text{time} \leftarrow \text{time} + 1$ 
  for each  $v \in \text{Adj}[u]$ 
    if  $c[v] = \text{white}$ 
       $\pi[v] \leftarrow u$ 
      visita(v)
   $c[u] \leftarrow \text{black}$ 
   $f[u] \leftarrow \text{time} \leftarrow \text{time} + 1$ 
```

# Busca em Profundidade

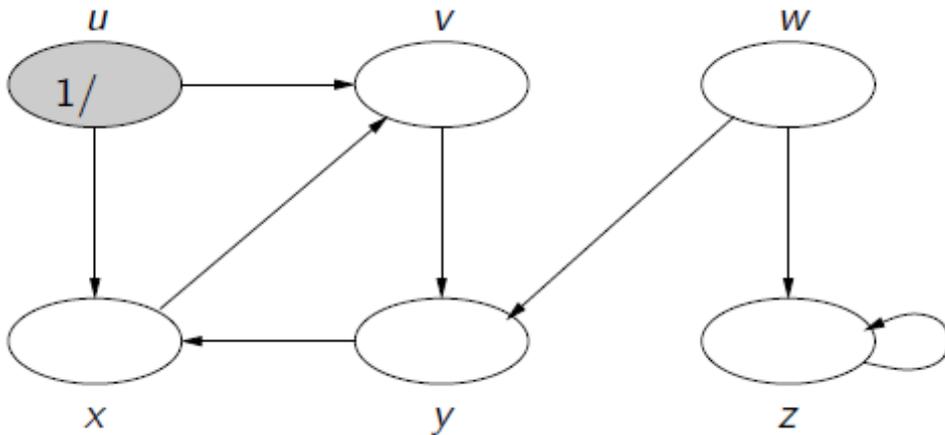


```

for each  $u \in V[G]$ 
   $c[u] \leftarrow \text{white}$ 
   $\pi[u] \leftarrow \text{NULL}$ 
time  $\leftarrow 0$ 
for each  $u \in V[G]$ 
  if  $c[u] = \text{white}$ 
    visita( $u$ )
  
```

	u	v	y	x	w	z
c	w	w	w	w	w	w
$\pi$	/	/	/	/	/	/
d						
f						

# Busca em Profundidade

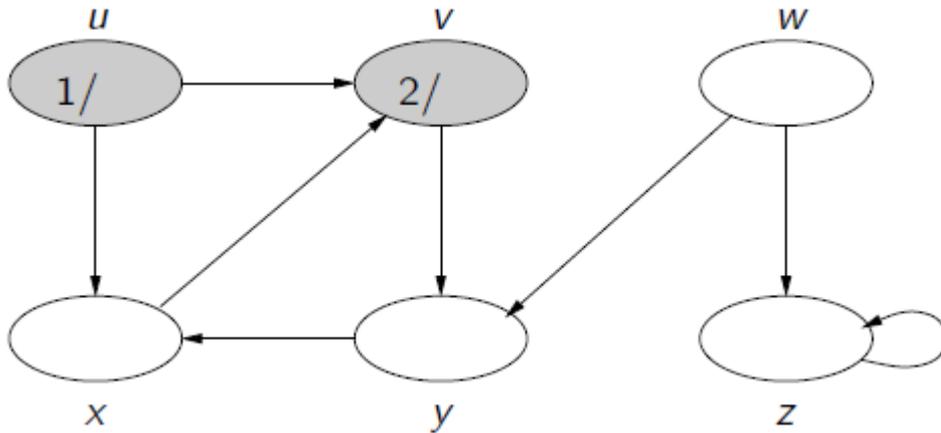


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	g	w	w	w	w	w
π	/	/	/	/	/	/
d	1					
f						

# Busca em Profundidade

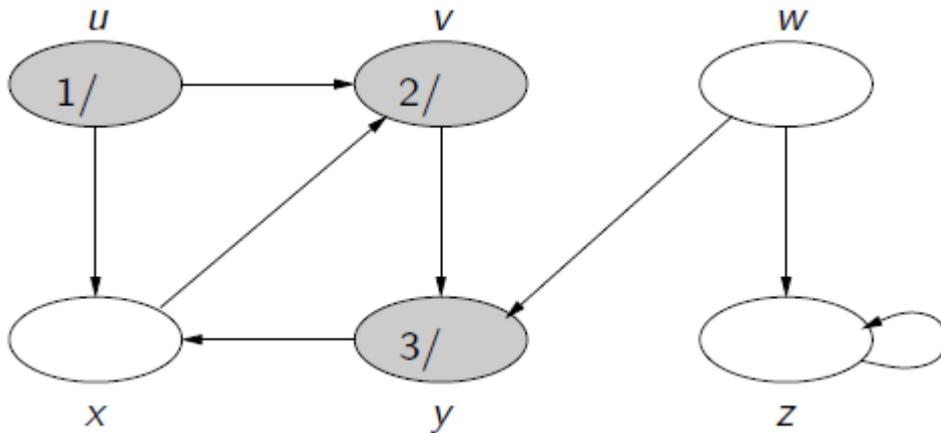


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	g	g	w	w	w	w
π	/	u	/	/	/	/
d	1	2				
f						

# Busca em Profundidade

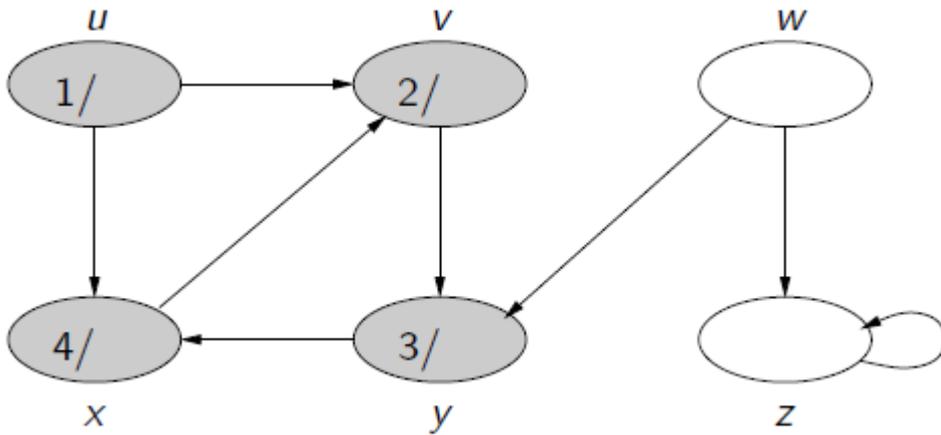


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	g	g	g	w	w	w
$\pi$	/	u	v	/	/	/
d	1	2	3			
f						

# Busca em Profundidade

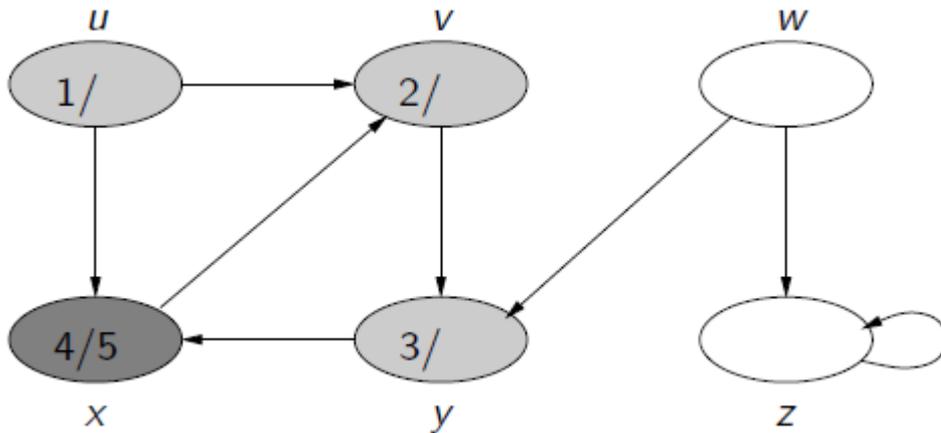


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	g	g	g	g	w	w
$\pi$	/	u	v	y	/	/
d	1	2	3	4		
f						

# Busca em Profundidade

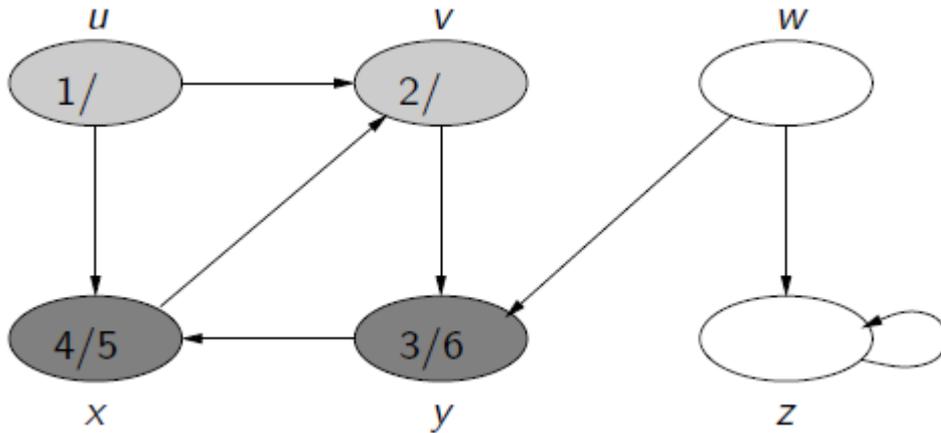


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
    if c[v] = white
        π[v] ← u
        visita(v)
c[u] ← black
f[u] ← time ← time + 1
    
```

	u	v	y	x	w	z
c	g	g	g	b	w	w
π	/	u	v	y	/	/
d	1	2	3	4		
f				5		

# Busca em Profundidade

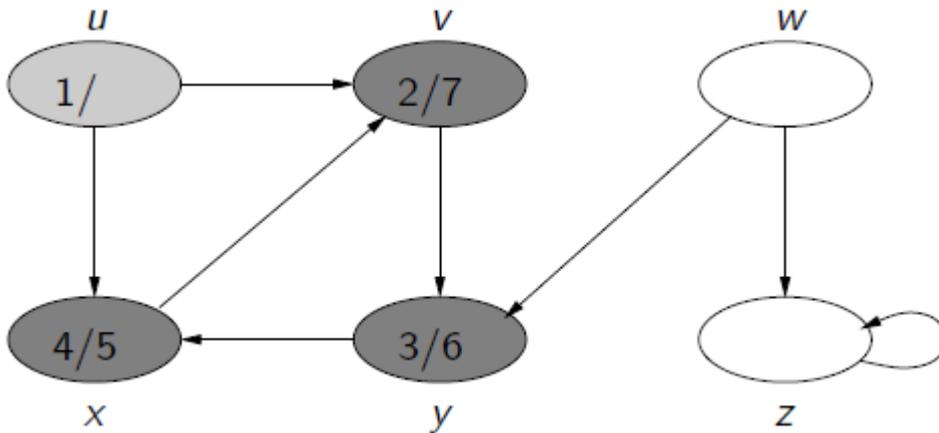


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	g	g	b	b	w	w
$\pi$	/	u	v	y	/	/
d	1	2	3	4		
f			6	5		

# Busca em Profundidade

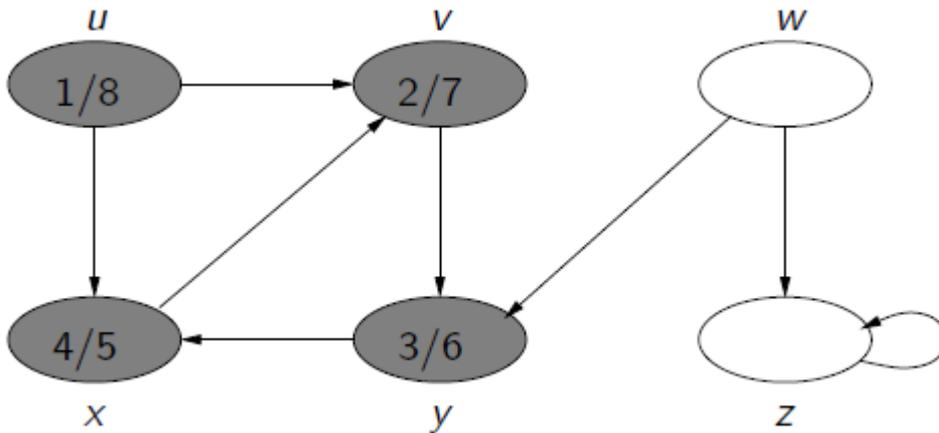


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
    if c[v] = white
        π[v] ← u
        visita(v)
c[u] ← black
f[u] ← time ← time + 1
    
```

	u	v	y	x	w	z
c	g	b	b	b	w	w
$\pi$	/	u	v	y	/	/
d	1	2	3	4		
f		7	6	5		

# Busca em Profundidade

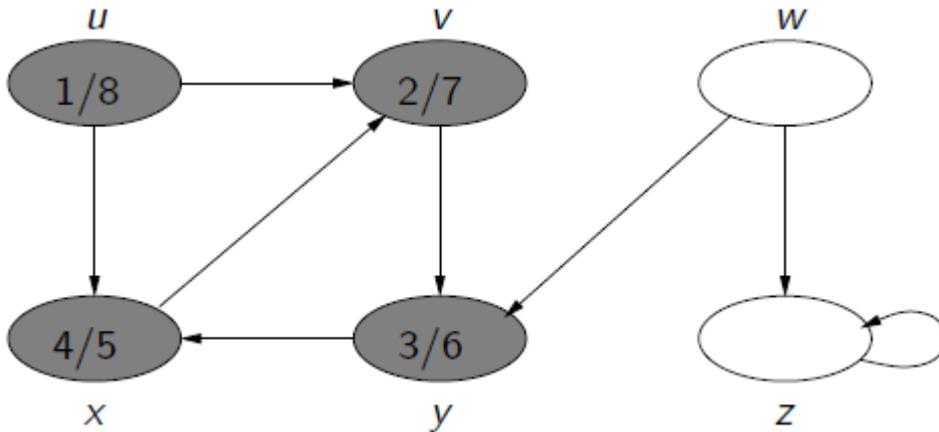


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	b	b	b	b	w	w
$\pi$	/	u	v	y	/	/
d	1	2	3	4		
f	8	7	6	5		

# Busca em Profundidade

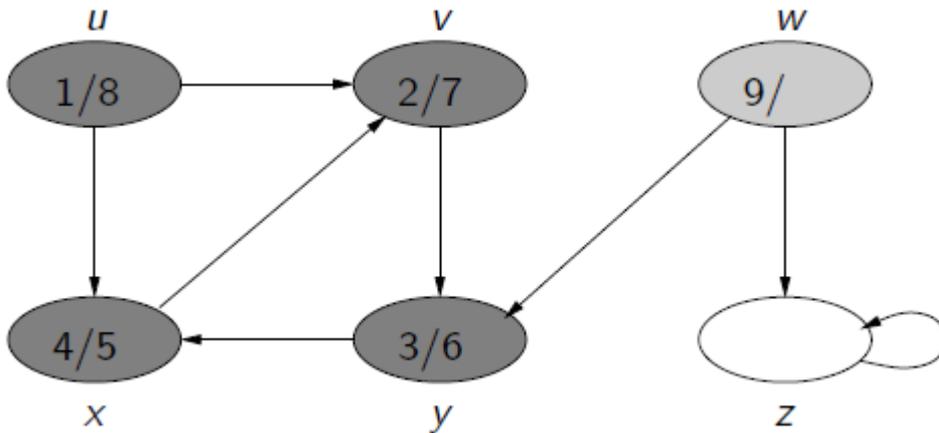


```

for each  $u \in V[G]$ 
   $c[u] \leftarrow \text{white}$ 
   $\pi[u] \leftarrow \text{NULL}$ 
time  $\leftarrow 0$ 
for each  $u \in V[G]$ 
  if  $c[u] = \text{white}$ 
    visita( $u$ )
  
```

	u	v	y	x	w	z
c	b	b	b	b	w	w
$\pi$	/	u	v	y	/	/
d	1	2	3	4		
f	8	7	6	5		

# Busca em Profundidade

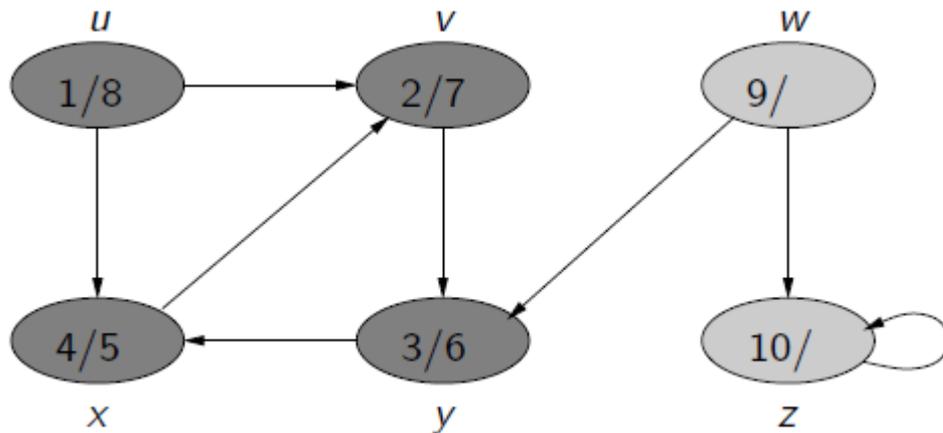


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	b	b	b	b	g	w
$\pi$	/	u	v	y	/	/
d	1	2	3	4	9	
f	8	7	6	5		

# Busca em Profundidade

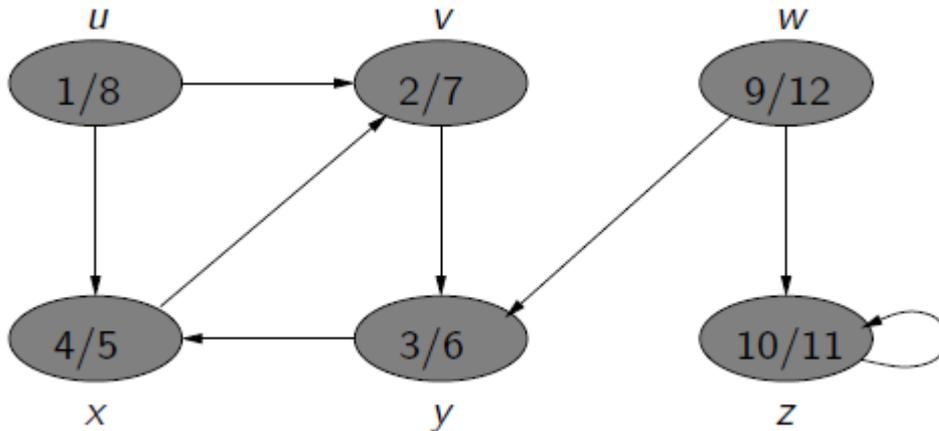


```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	b	b	b	b	g	g
$\pi$	/	u	v	y	/	w
d	1	2	3	4	9	10
f	8	7	6	5		

# Busca em Profundidade



```

c[u] ← gray
d[u] ← time ← time + 1
for each v ∈ Adj[u]
  if c[v] = white
    π[v] ← u
    visita(v)
c[u] ← black
f[u] ← time ← time + 1
  
```

	u	v	y	x	w	z
c	b	b	b	b	b	b
$\pi$	/	u	v	y	/	w
d	1	2	3	4	9	10
f	8	7	6	5	11	12

# Busca em Profundidade – Análise

```
BuscaEmProfundidade(G)
  for each  $u \in V[G]$ 
     $c[u] \leftarrow \text{white}$ 
     $\pi[u] \leftarrow \text{NULL}$ 
  time  $\leftarrow 0$ 
  for each  $u \in V[G]$ 
    if  $c[u] = \text{white}$ 
      visita(u)
```

```
visita(u)
   $c[u] \leftarrow \text{gray}$ 
   $d[u] \leftarrow \text{time} \leftarrow \text{time} + 1$ 
  for each  $v \in \text{Adj}[u]$ 
    if  $c[v] = \text{white}$ 
       $\pi[v] \leftarrow u$ 
      visita(v)
   $c[u] \leftarrow \text{black}$ 
   $f[u] \leftarrow \text{time} \leftarrow \text{time} + 1$ 
```

- O procedimento `visita` é chamado exatamente uma vez para cada vértice  $u \in V$ , isso porque `visita` é chamado apenas para vértices brancos e a primeira ação é pintar o vértice de cinza:  $O(V)$ ;
- O loop principal de `visita(u)` tem complexidade  $O(A)$ ;
- Complexidade:  **$O(V + A)$**

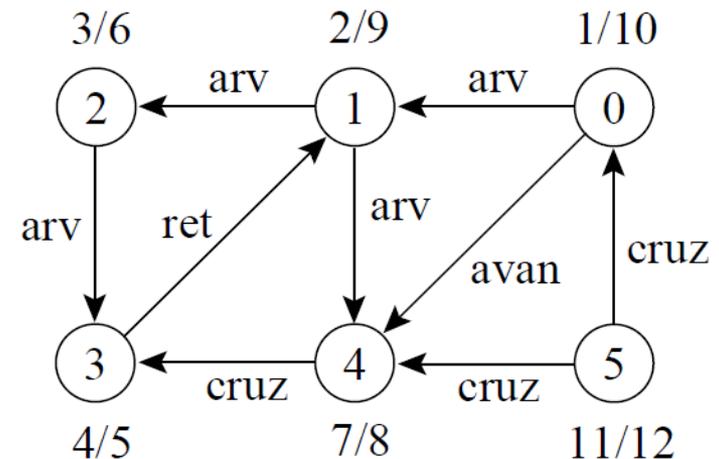
# Busca em Profundidade

- **Classificação de Arestas:**

- **Arestas de árvore:** são arestas de uma árvore de busca em profundidade. A aresta  $(u, v)$  é uma aresta de árvore se  $v$  foi descoberto pela primeira vez ao percorrer a aresta  $(u, v)$ ;
- **Arestas de retorno:** conectam um vértice  $u$  com um antecessor  $v$  em uma árvore de busca em profundidade (inclui self-loops);
- **Arestas de avanço:** não pertencem à árvore de busca em profundidade mas conectam um vértice a um descendente que pertence à árvore de busca em profundidade;
- **Arestas de cruzamento:** podem conectar vértices na mesma árvore de busca em profundidade, ou em duas árvores diferentes.

# Busca em Profundidade

- Classificação de arestas pode ser útil para derivar outros algoritmos.
- Na busca em profundidade cada aresta pode ser classificada pela cor do vértice que é alcançado pela primeira vez:
  - Branco indica uma **aresta de árvore**.
  - Cinza indica uma **aresta de retorno**.
  - Preto indica uma **aresta de avanço** quando  $u$  é descoberto antes de  $v$  ou uma **aresta de cruzamento** caso contrário.



# Exercícios

## **Lista de Exercícios 06 – Busca em Profundidade e Busca em Largura**

<http://www.inf.puc-rio.br/~elima/paa/>

