BUSCA EM PROFUNDIDADE E BUSCA EM LARGURA

DCE529 - Algoritmos e Estruturas de Dados III

Atualizado em: 26 de março de 2024

Iago Carvalho

Departamento de Ciência da Computação

BUSCA EM GRAFOS

Como vimos na última aula, grafos são estruturas muito úteis em computação

- Representar um conjunto de elementos
- Representar as conexões ou relacionamentos entre estes elementos

Por diversas vezes, estamos interessados em inferir algo sobre os dados armazenados em um grafo

A coisa mais simples que podemos fazer é uma busca!

- Saber se existe um caminho entre dois vértices quaisquer
- Descobrir se o grafo é conexo ou não

BUSCA EM PROFUNDIDADE

BUSCA EM PROFUNDIDADE

É um algoritmo para se caminhar em grafos

A estratégia é buscar o mais *profundo* no grafo sempre que possível

As arestas são exploradas a partir do vértice mais recentemente descoberto

BUSCA EM PROFUNDIDADE

Quando todas as arestas adjacentes a um vértice v já tiverem sido exploradas, o algoritmo *anda para trás*

- Tenta explorar outros vértices adjacentes ao vértice pai de v

Algoritmo com diversas aplicações

- Ordenação topológica
- Componentes conectados
- Verificação de ciclos

BUSCA EM PROFUNDIDADE

Para acompanhar o algoritmo, utilizam-se vértices de diferentes cores

- Branco, cinza e preto

Inicialmente, todos os vértices são brancos

- Quando eles são descobertos, são pintados de cinza
- QUando eles são fechados, são pintados de preto

Também utilizamos marcadores de tempo para denotar o tempo de abertura e fechamento dos vértices

PSEUDOCÓDIGO

$DFS(G)$

1 *para cada vértice* $u \leftarrow V[G]$

2 $\quad cor[u] \leftarrow BRANCO$

3 $tempo \leftarrow 0$

4 *para cada vértice* $u \in V[G]$

5 $\quad$ *se* $cor[u] = BRANCO$

6 $\quad\quad DFS - VISIT(u)$

$DFS - VISIT(u)$

1 $cor[u] \leftarrow CINZA$

2 $tempo \leftarrow tempo + 1$

3 $d[u] \leftarrow tempo$

4 *para cada vértice* $v \in Adj(u)$

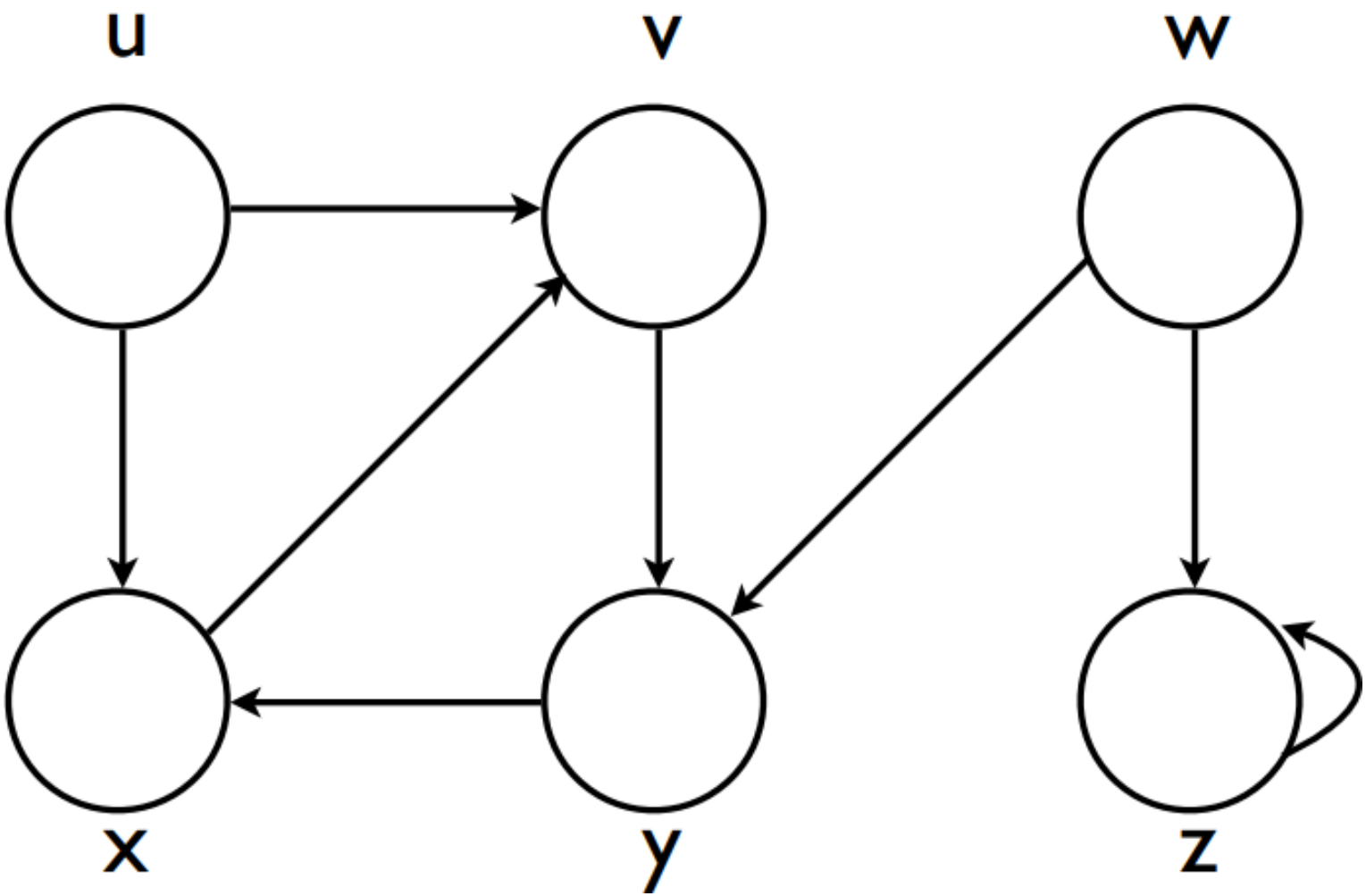
5 $\quad$ *se* $cor[v] = BRANCO$

6 $\quad\quad DFS - VISIT(v)$

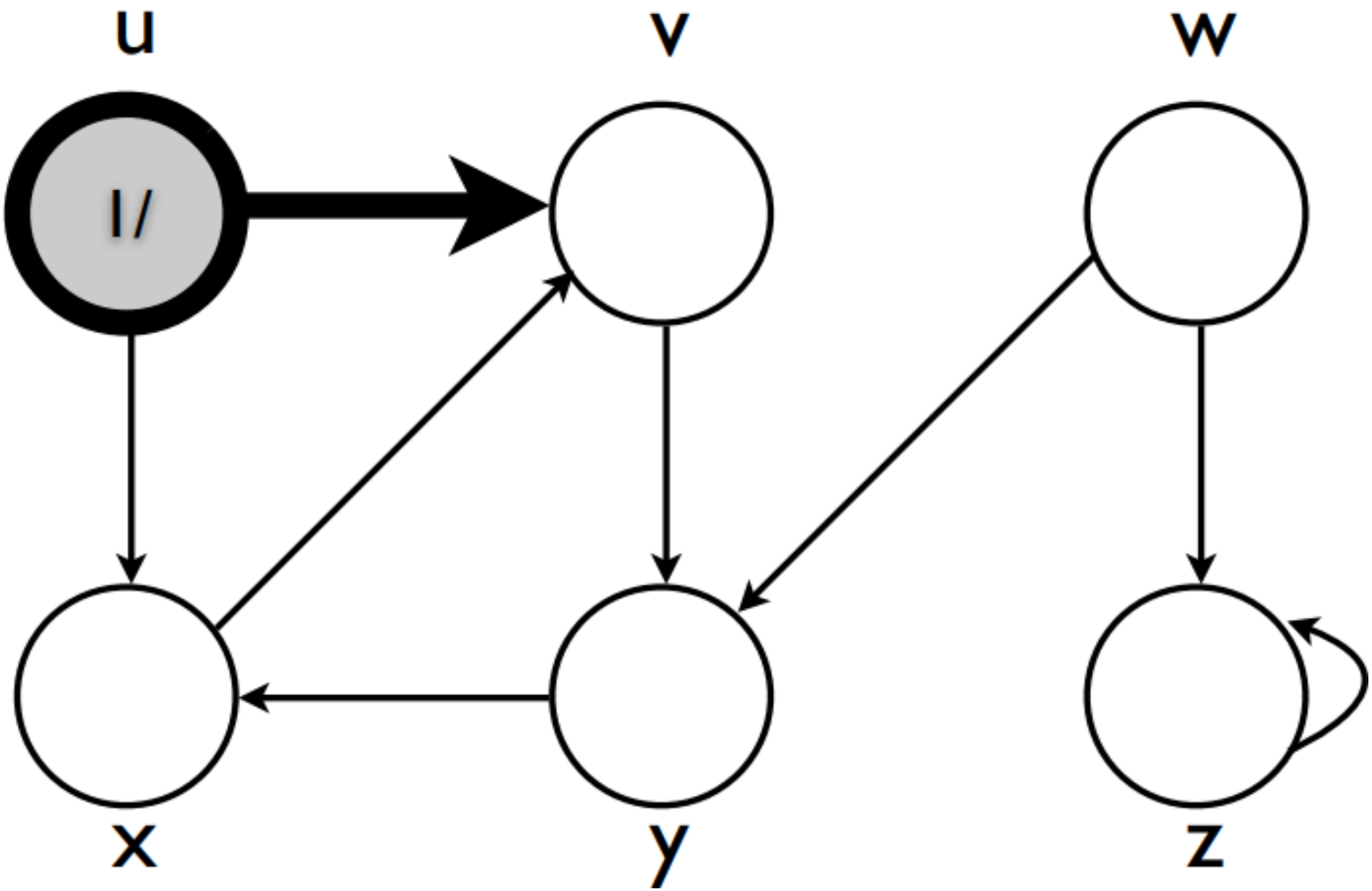
7 $cor[u] \leftarrow PRETO$

8 $f[u] \leftarrow tempo \leftarrow (tempo + 1)$

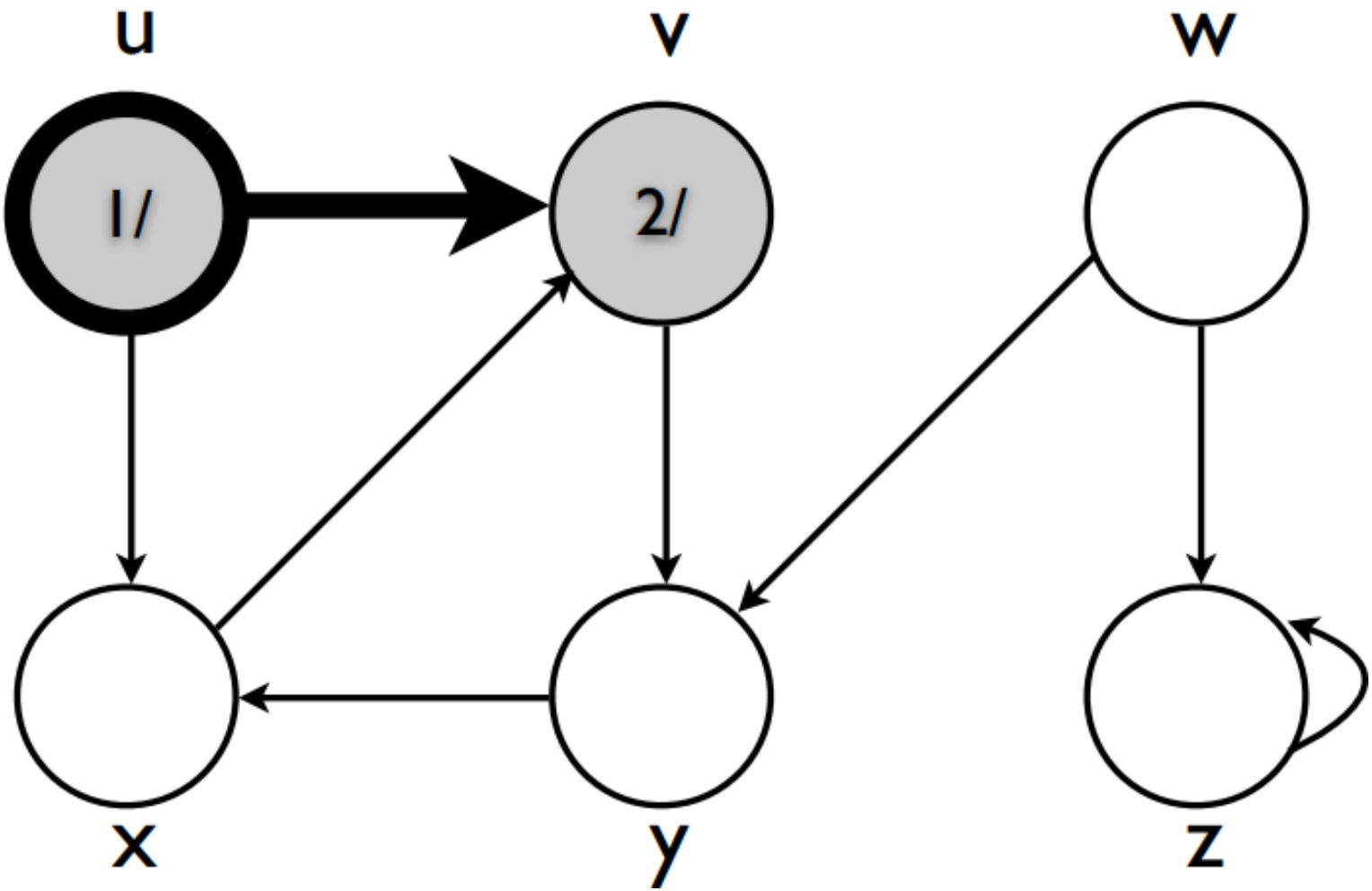
EXEMPLO



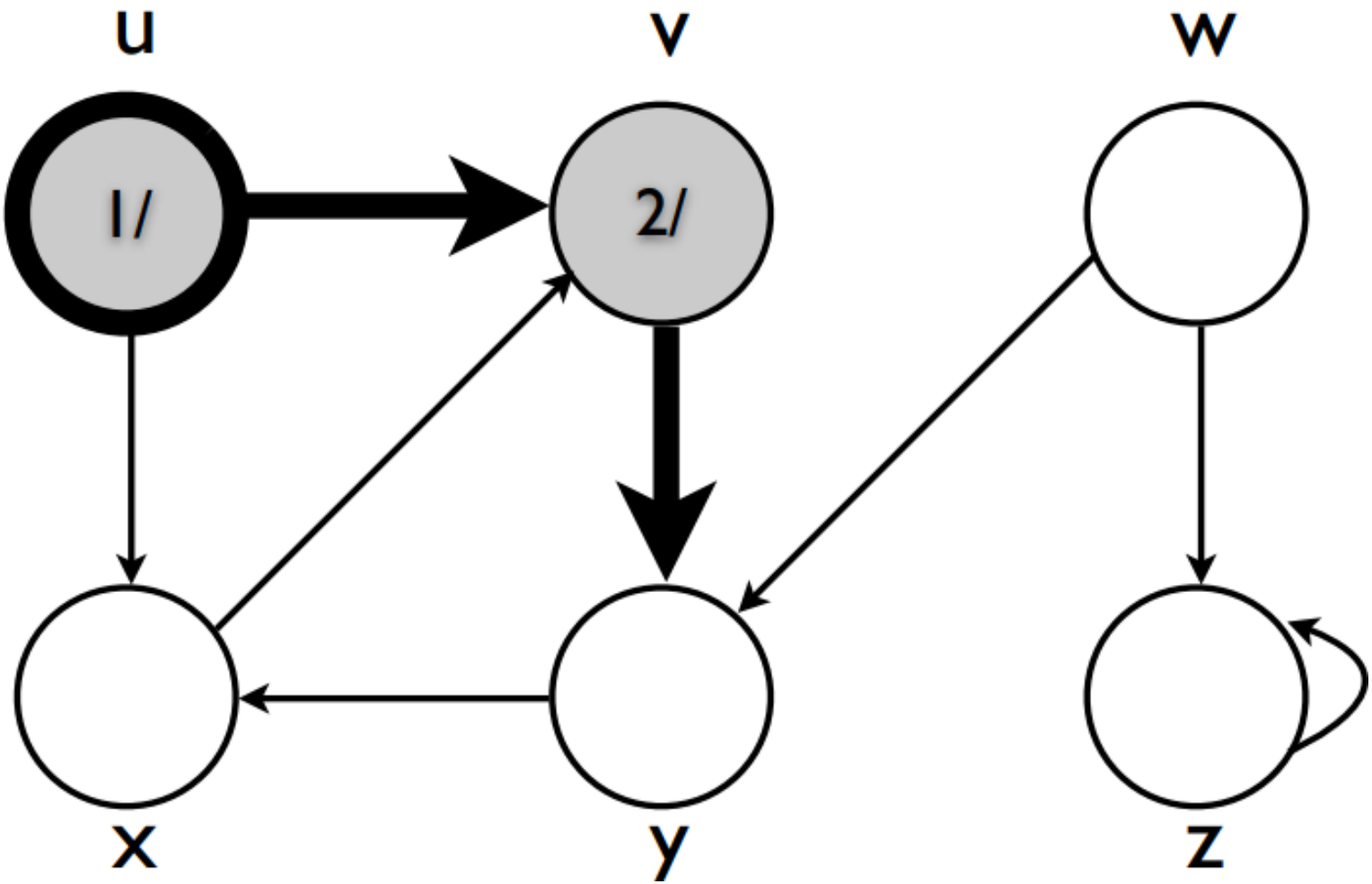
EXEMPLO



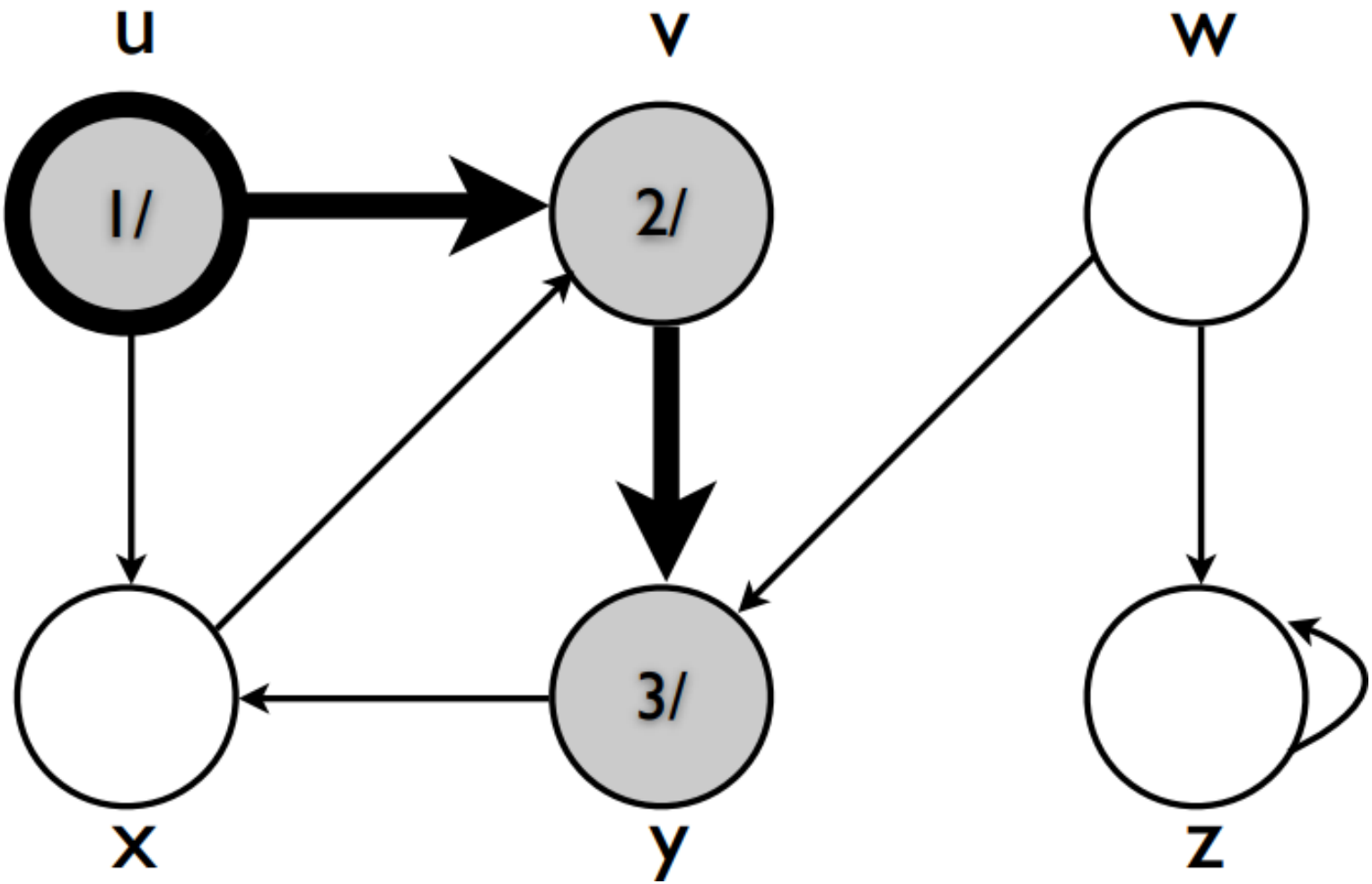
EXEMPLO



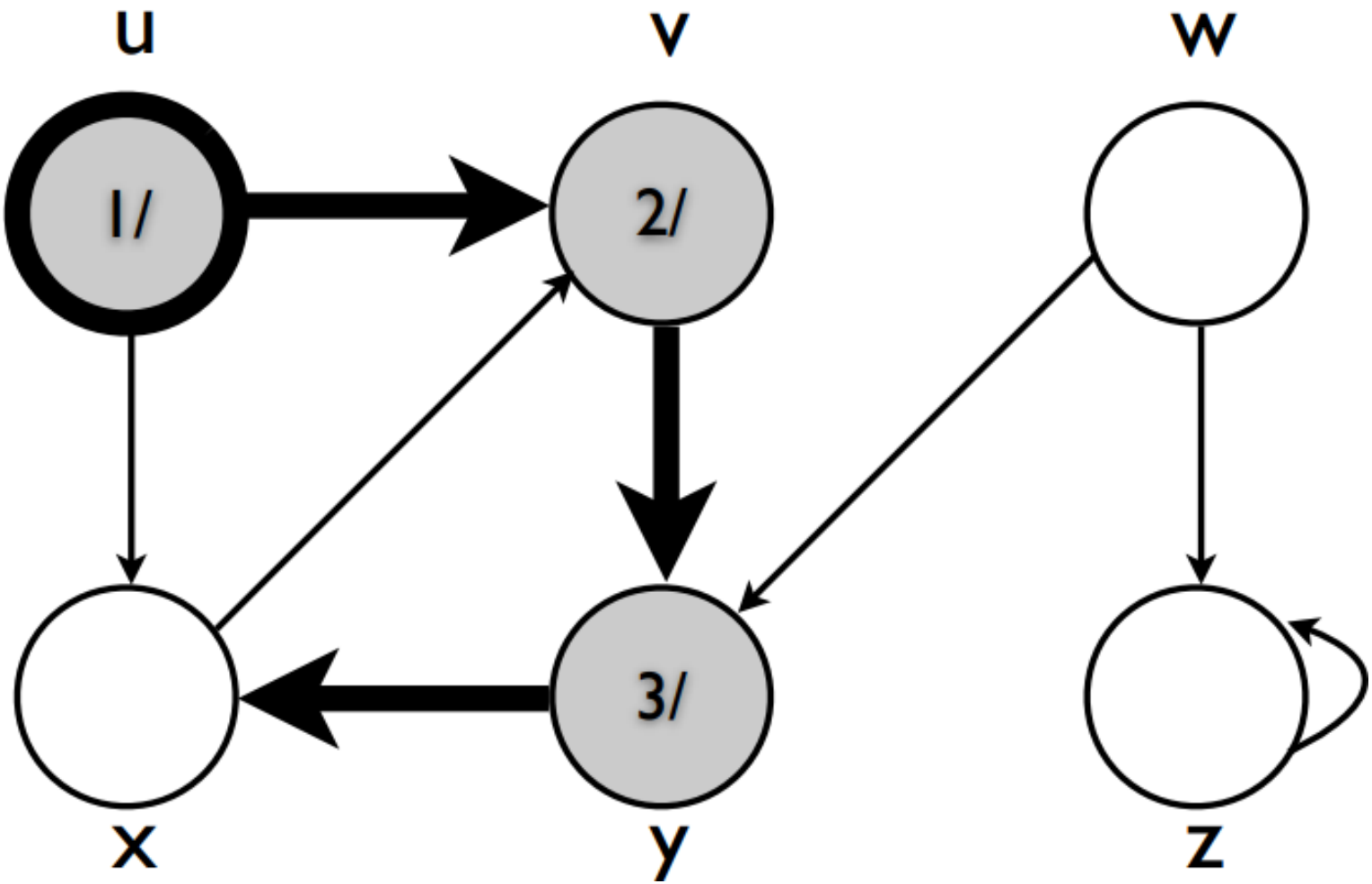
EXEMPLO



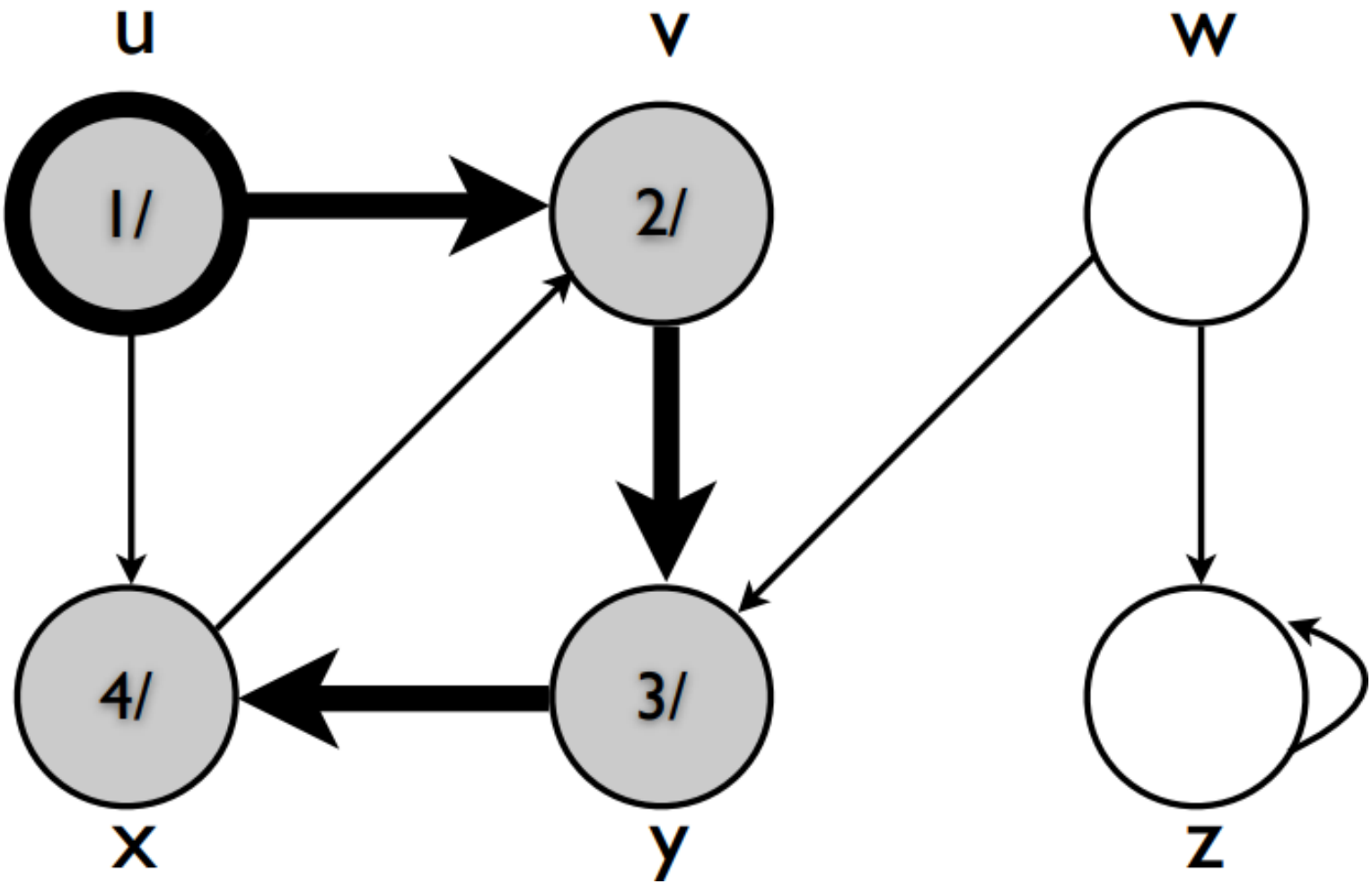
EXEMPLO



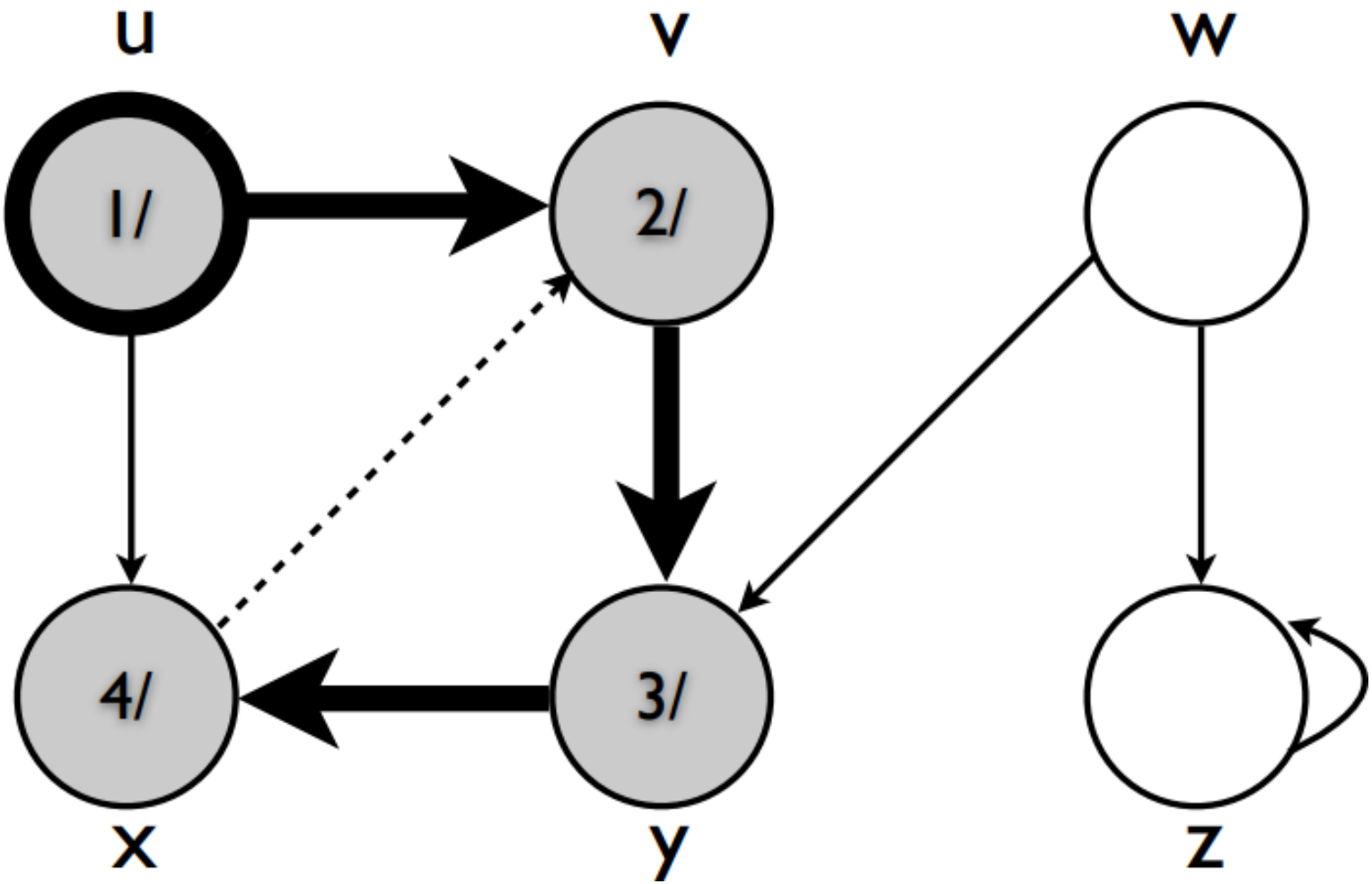
EXEMPLO



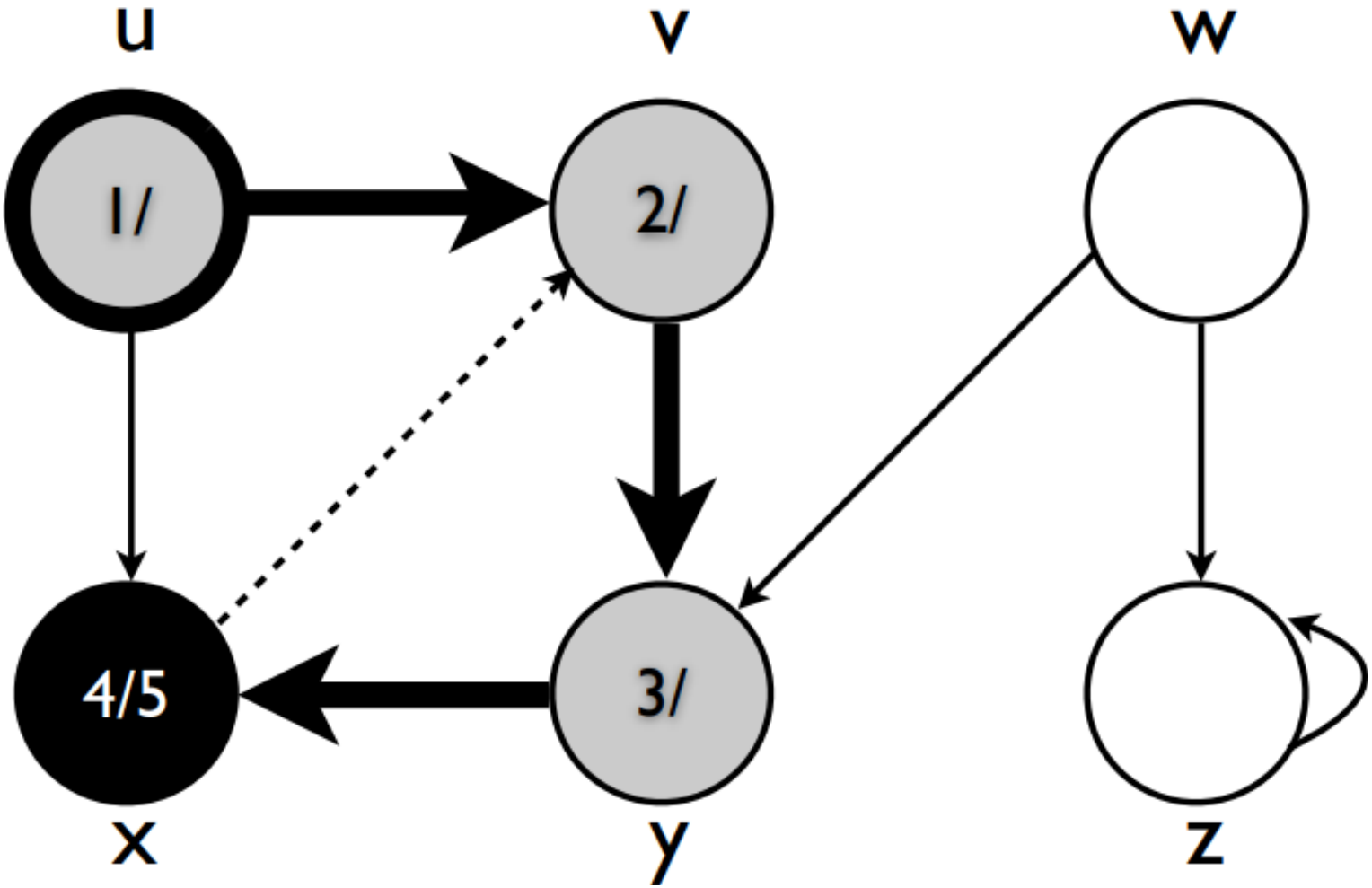
EXEMPLO



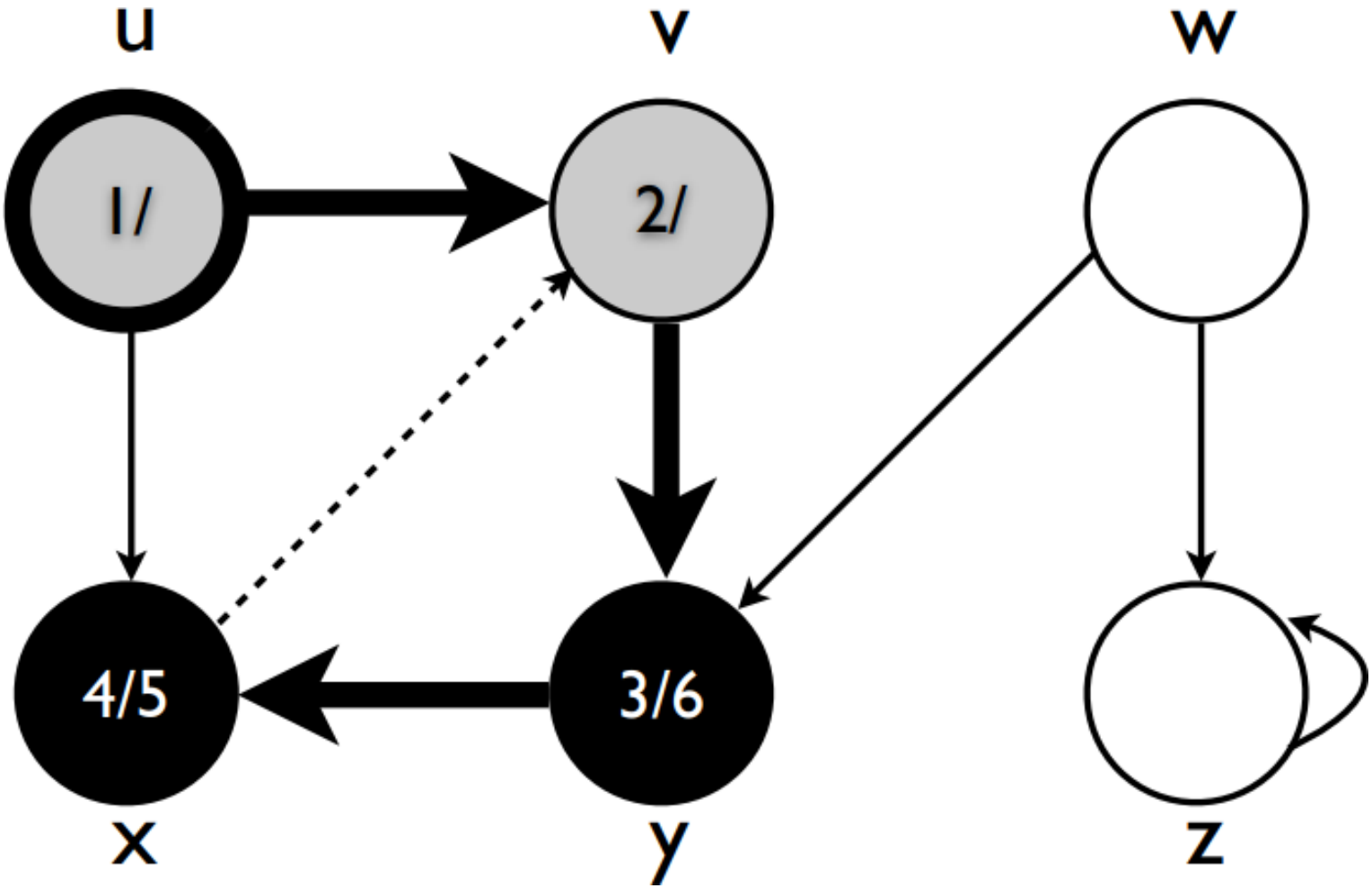
EXEMPLO



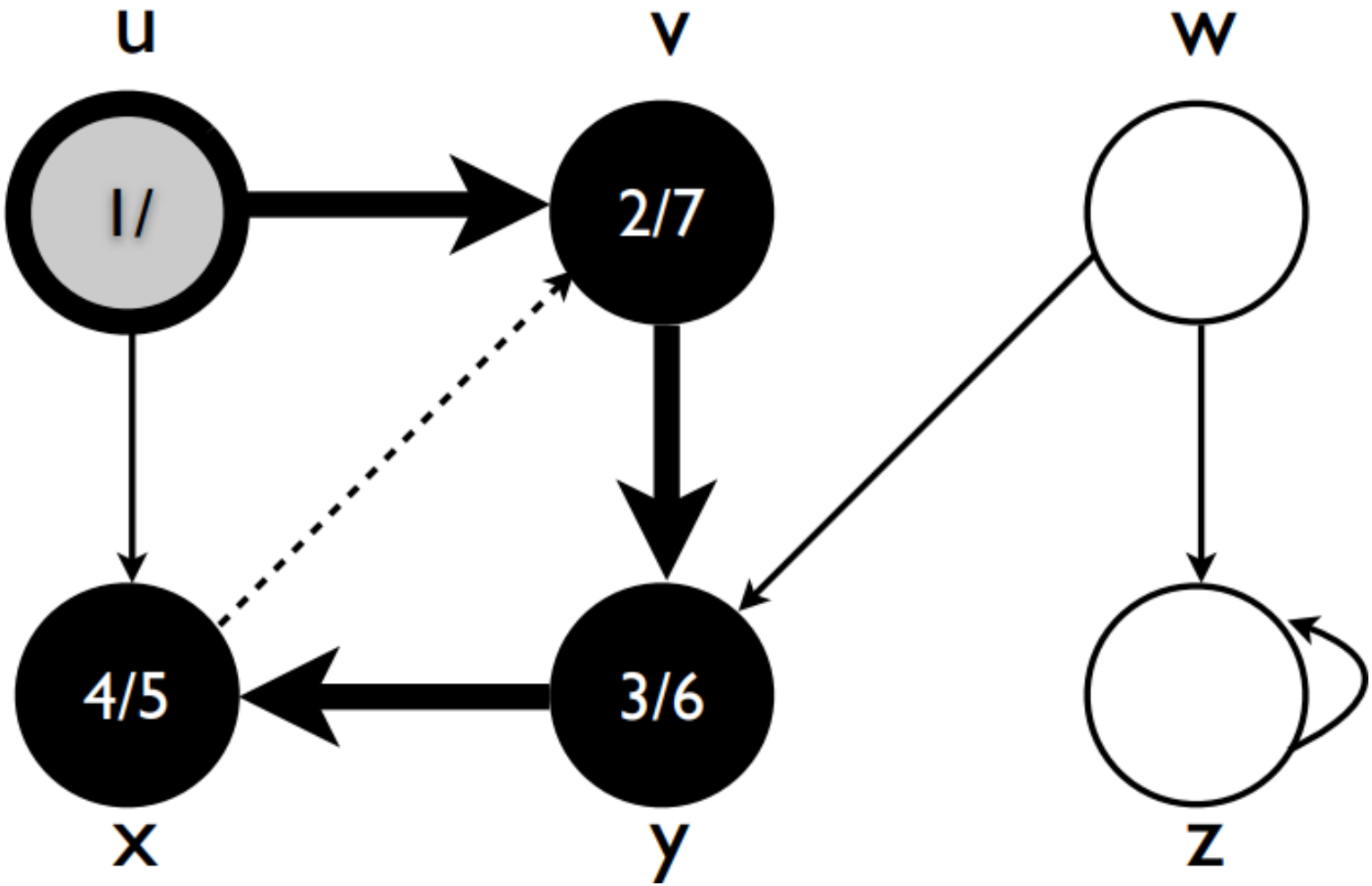
EXEMPLO



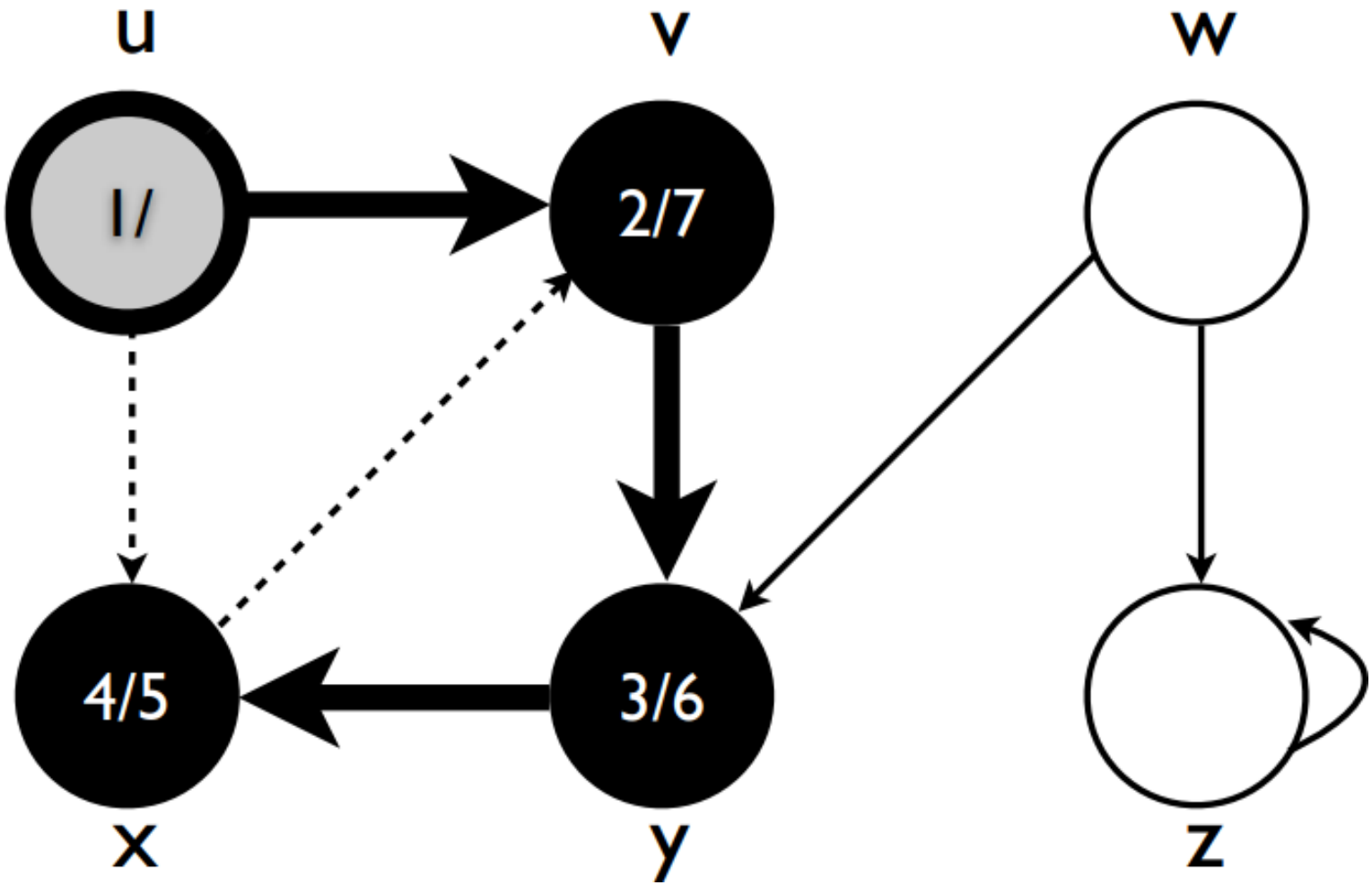
EXEMPLO



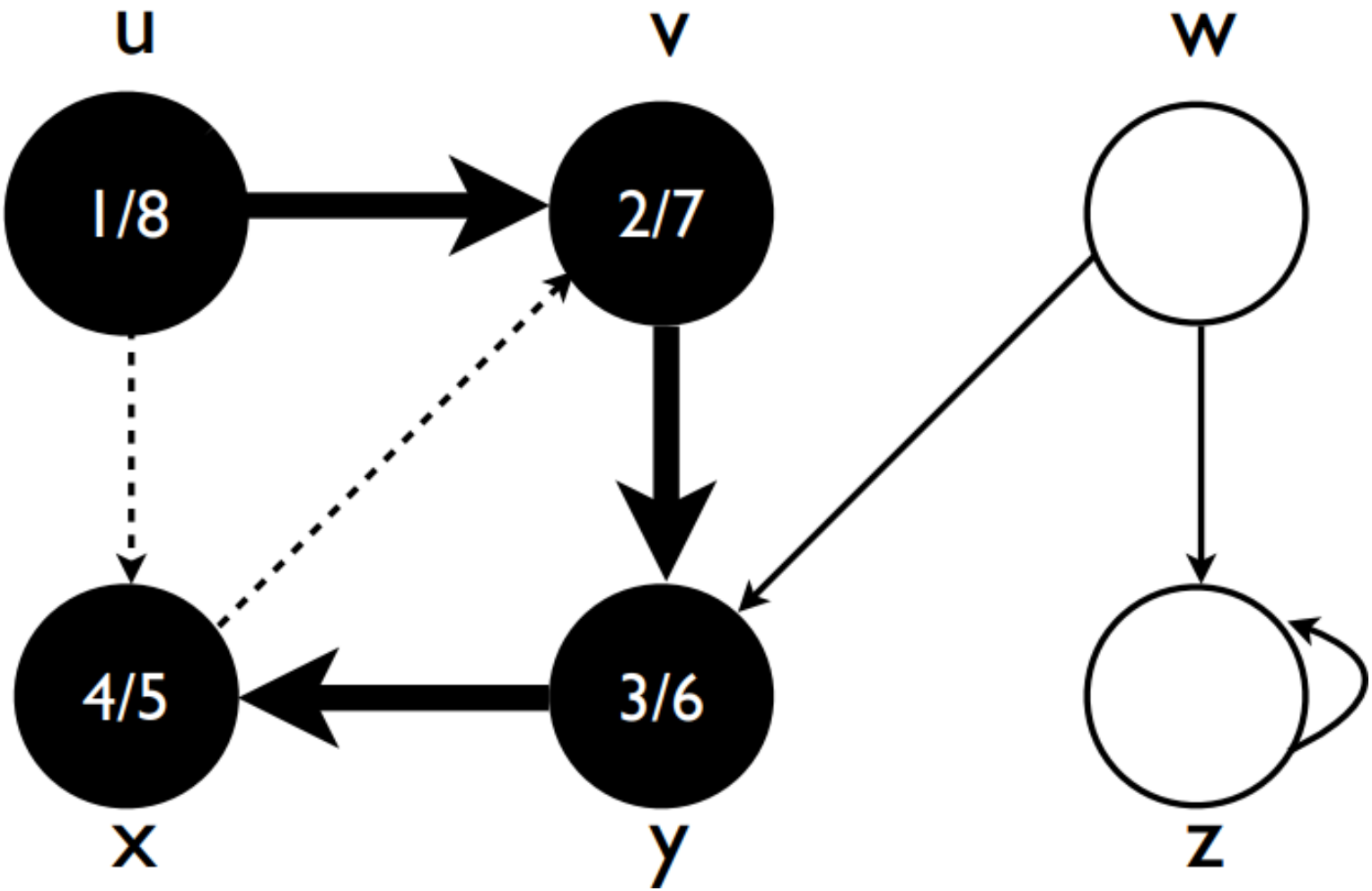
EXEMPLO



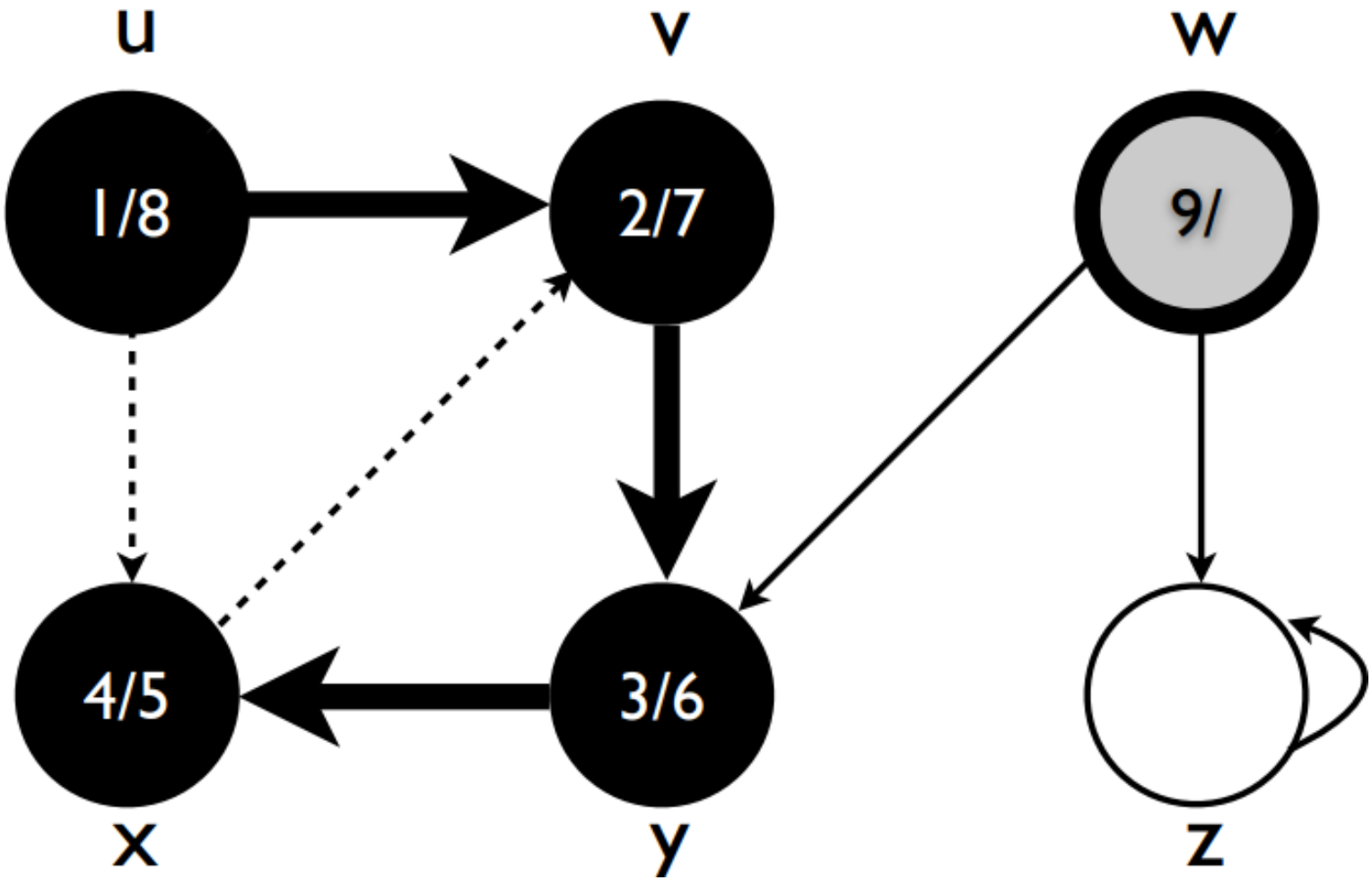
EXEMPLO



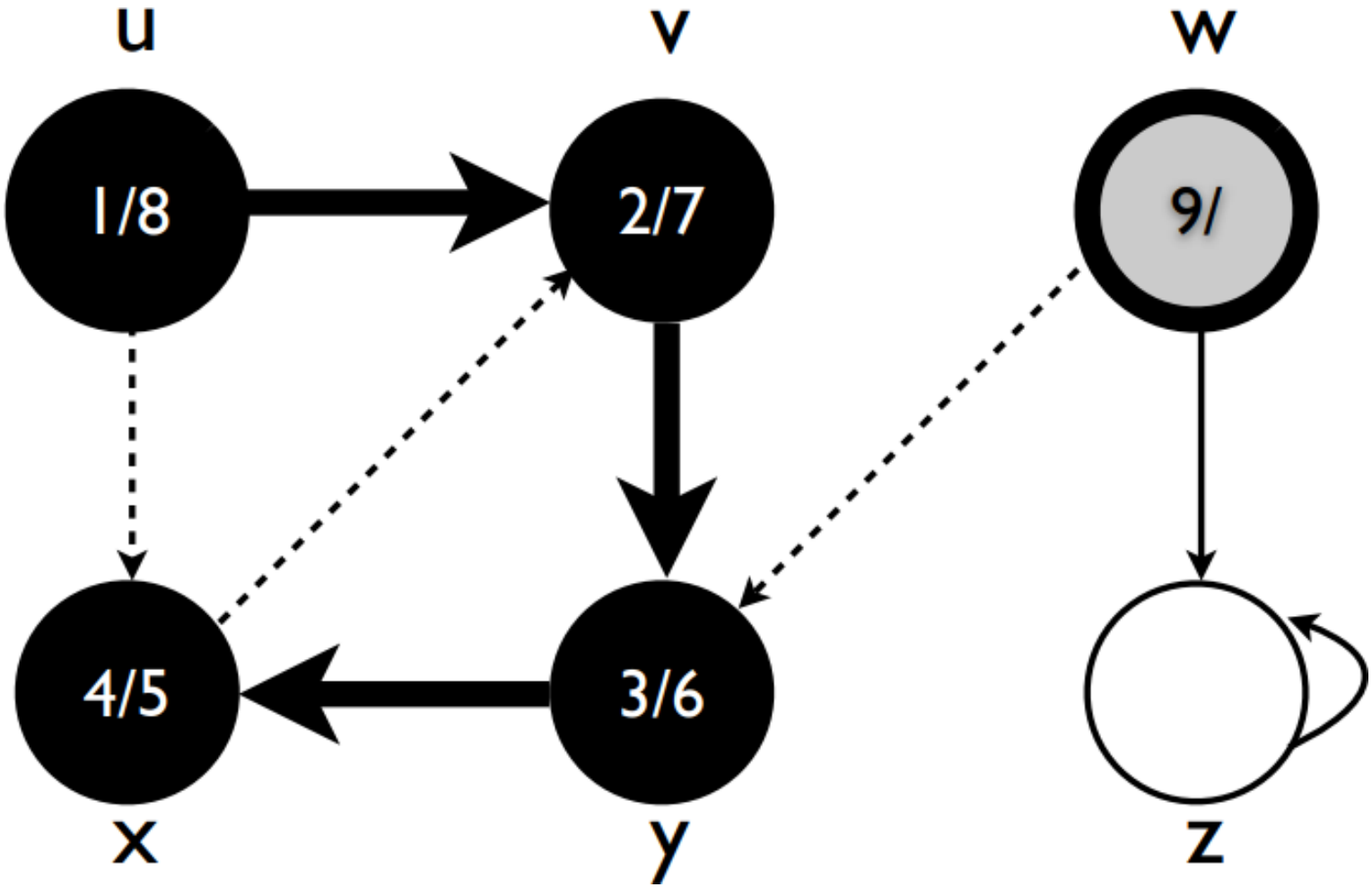
EXEMPLO



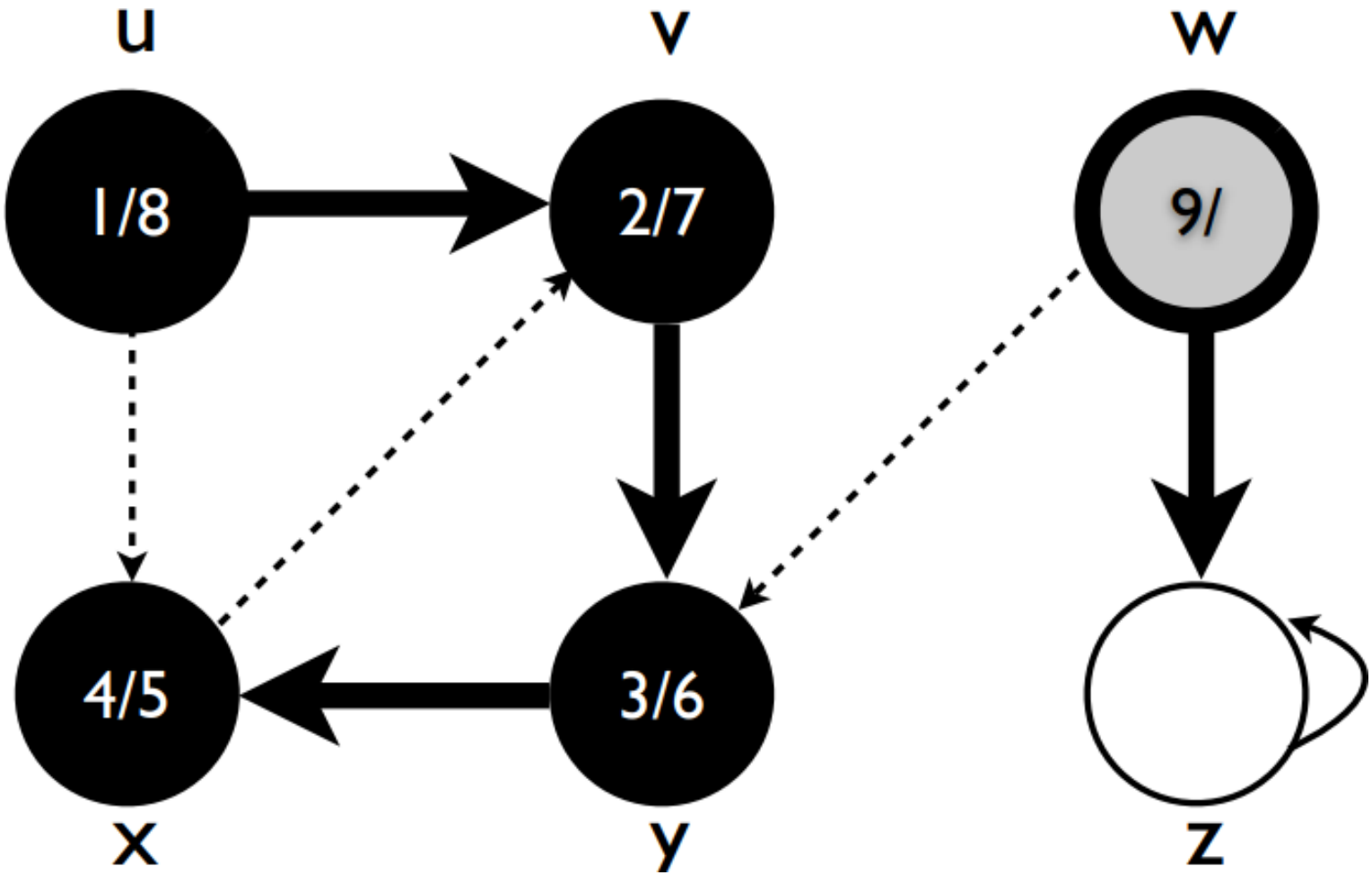
EXEMPLO



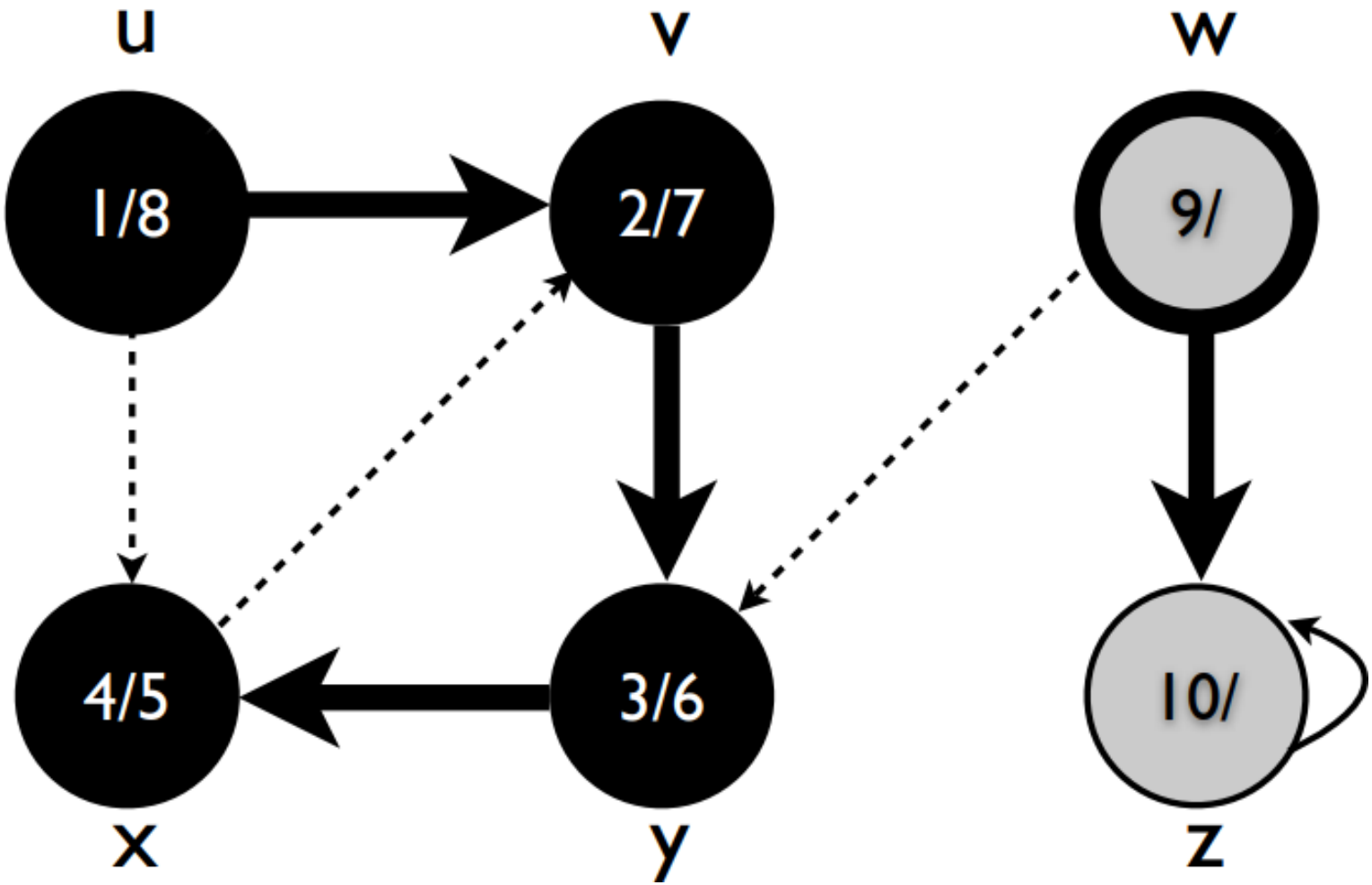
EXEMPLO



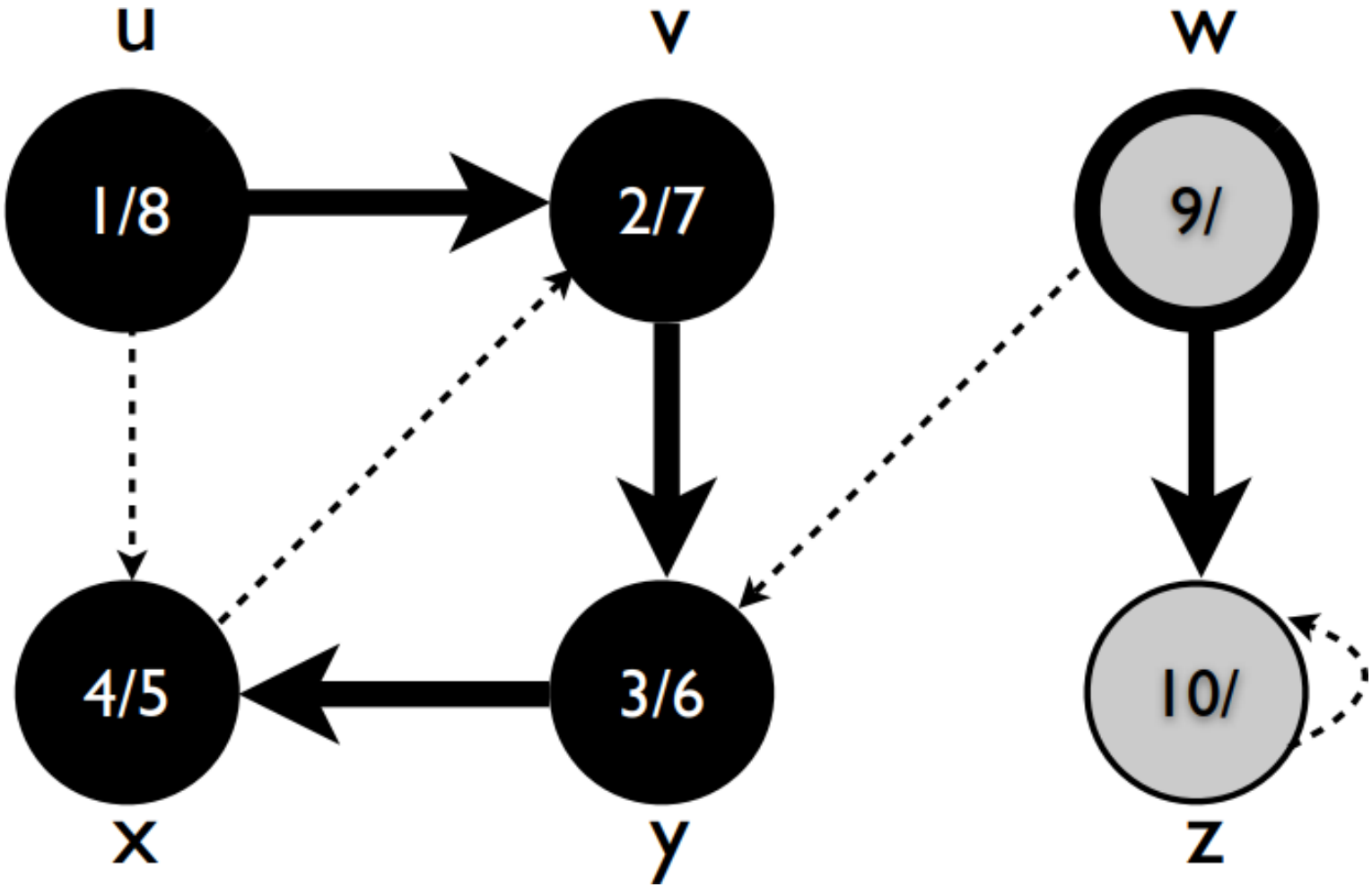
EXEMPLO



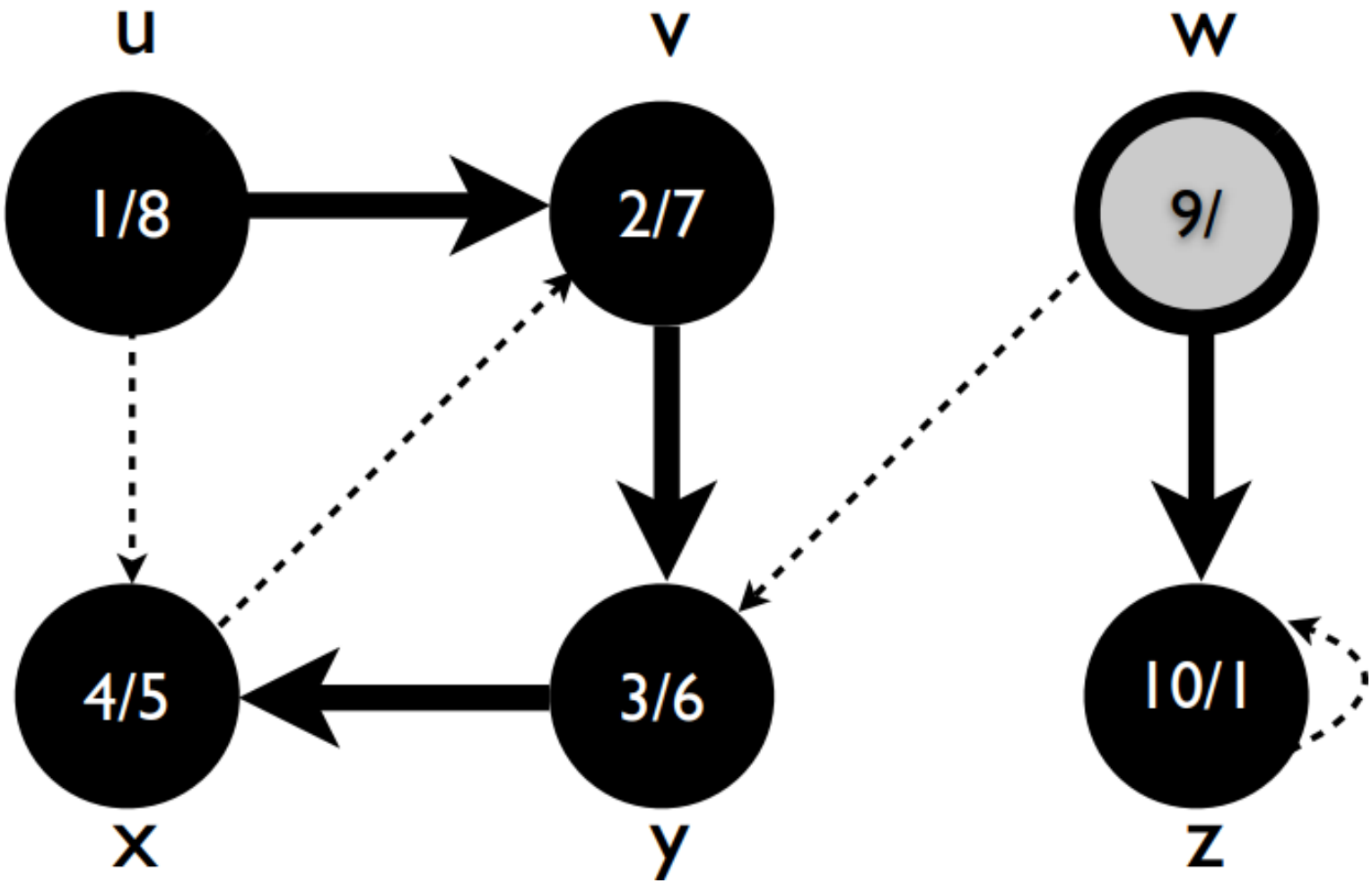
EXEMPLO



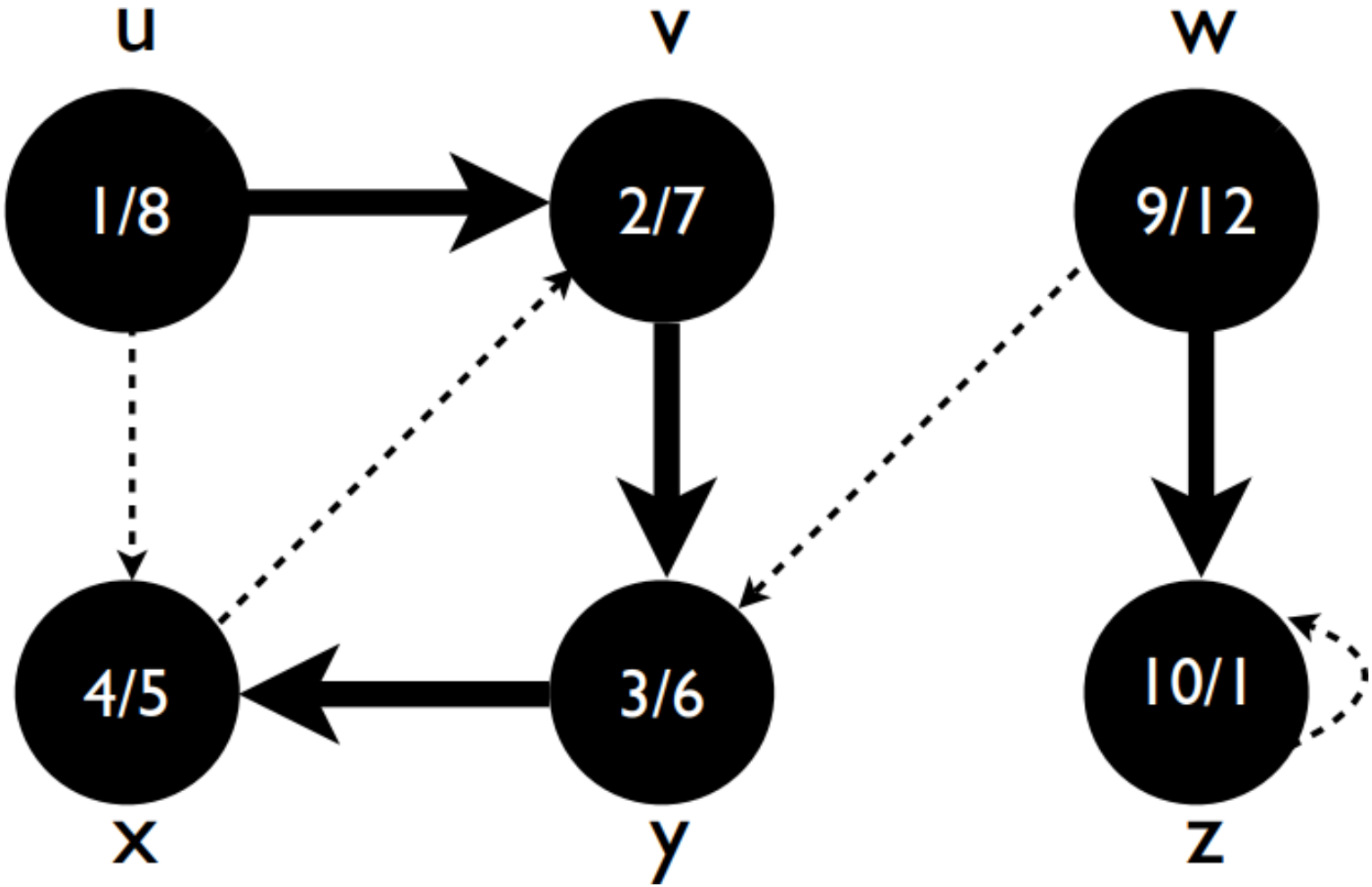
EXEMPLO



EXEMPLO



EXEMPLO



COMPLEXIDADE

A função *DFS* tem complexidade $\mathcal{O}(|V|)$

Toda chamada de *DFS-VISIT* realiza um máximo de $adj[v]$ operações

- $\sum_{v \in V} adj[v] = \Theta(|E|)$

A complexidade final do algoritmo é de $\mathcal{O}(|V| + |E|)$

BUSCA EM LARGURA

BUSCA EM LARGURA

Ideia contrária a busca em profundidade

- Aqui tentamos explorar todos os vértices que estão a uma mesma profundidade de uma só vez

Não aprofunda em um único caminho

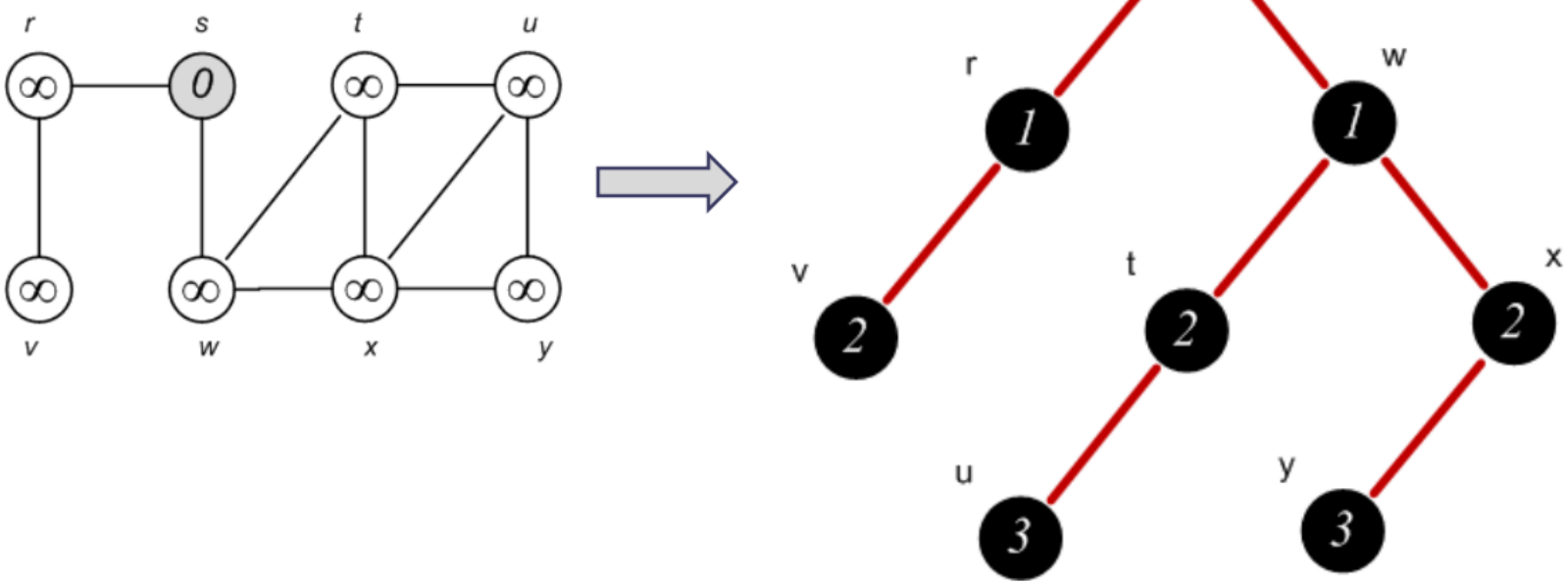
- Ao invés disso, faz uma busca *ampla* ou *larga*
- Expande a fronteira de busca de maneira uniforme a partir de um vértice raiz

BUSCA EM LARGURA

Algoritmo base para outros

- Algoritmo de Prim para Árvore Geradora Mínima
- Algoritmo de Dijkstra para Caminho Mínimo

No fim, ele produz uma árvore de níveis



BUSCA EM LARGURA

Utiliza os mesmos conceitos de cores que a busca em profundidade

- Vértices brancos, cinzas e pretos

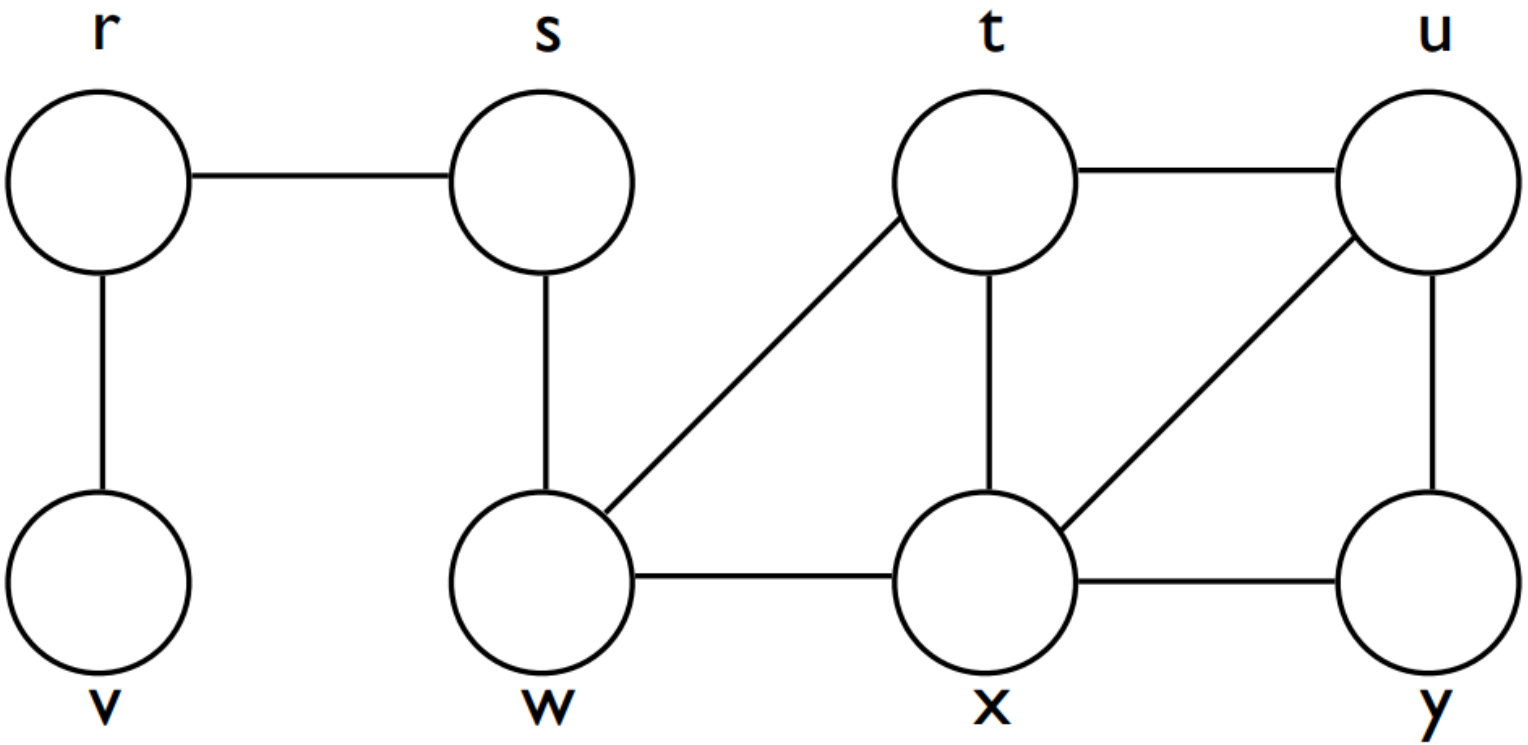
Também utiliza uma fila Q, uma medida de nível d e um vetor de antecessores π

- Fila Q armazena os vértices a serem visitados
- Medida de nível d guarda a distância do vértice até a raiz
- Vetor de antecessores π guarda o vértice pai de outro vértice

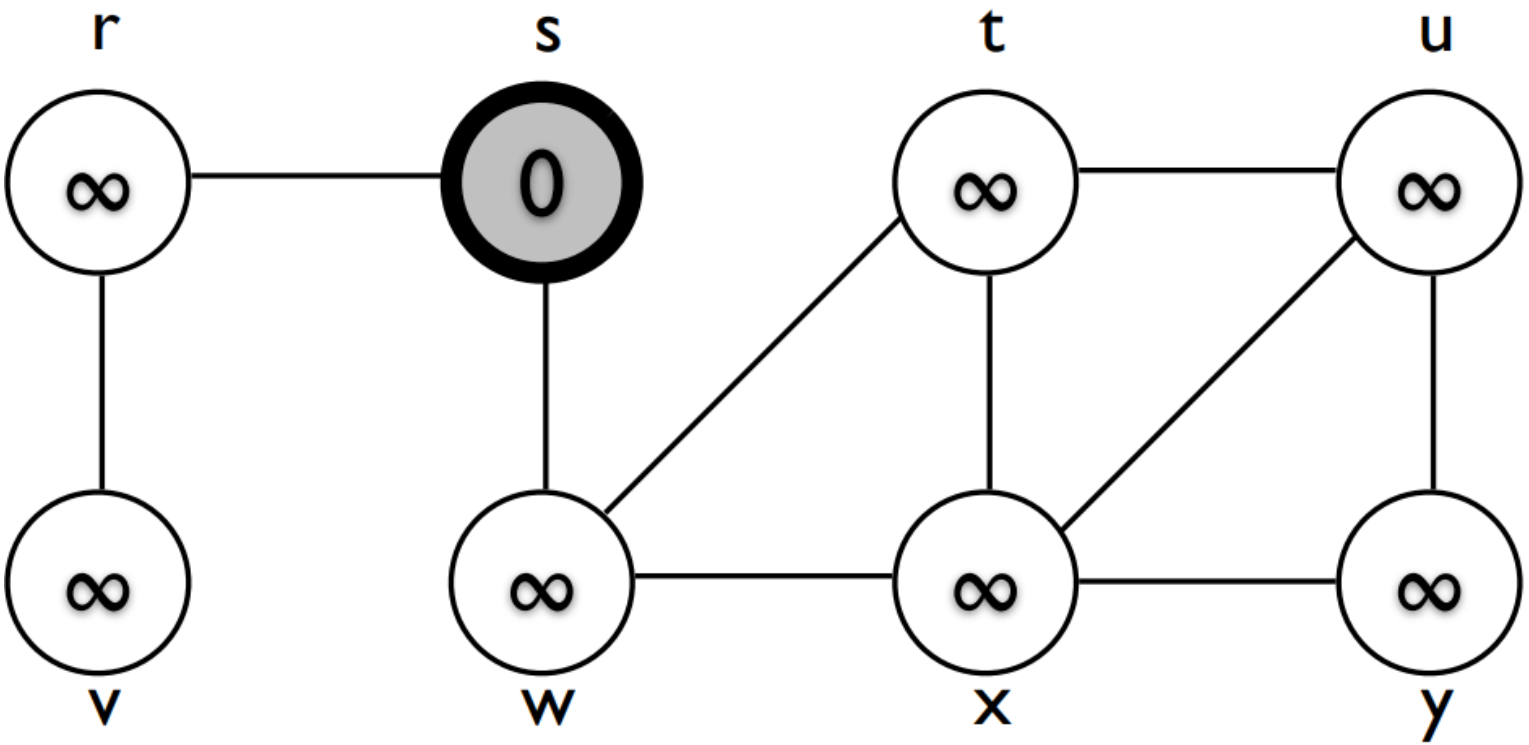
PSEUDOCÓDIGO

```
BFS(G, s)
1  para cada vértice u ← V[G] − {s}
2      cor[u] ← BRANCO
3      d[u] ← ∞
4      π[u] ← NULL
5  cor[s] ← CINZA
6  d[s] ← 0
7  π[s] ← NULL
8  Q ← novaFila()
9  ENFILEIRA(Q, s)
10 enquanto !vazia(Q)
11     u ← DESENFILEIRA(Q)
12     para cada v ← Adj[u]
13         se cor[v] = BRANCO
14             cor[v] ← CINZA
15             d[v] = d[u] + 1
16             π[v] ← u
17             ENFILEIRA(Q, v)
18     cor[u] ← PRETO
```

EXEMPLO



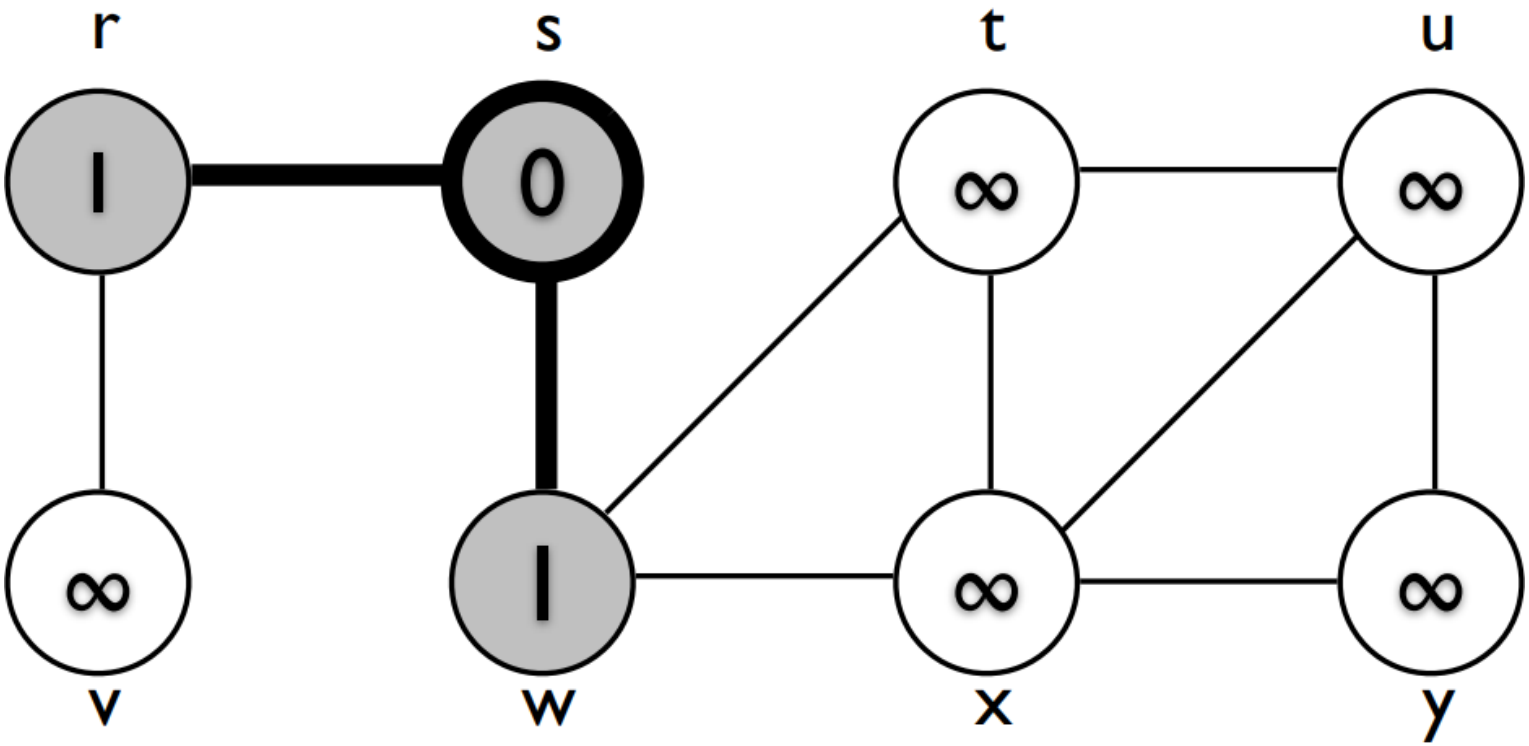
EXEMPLO



Fila Q | s

Nível | 0

EXEMPLO



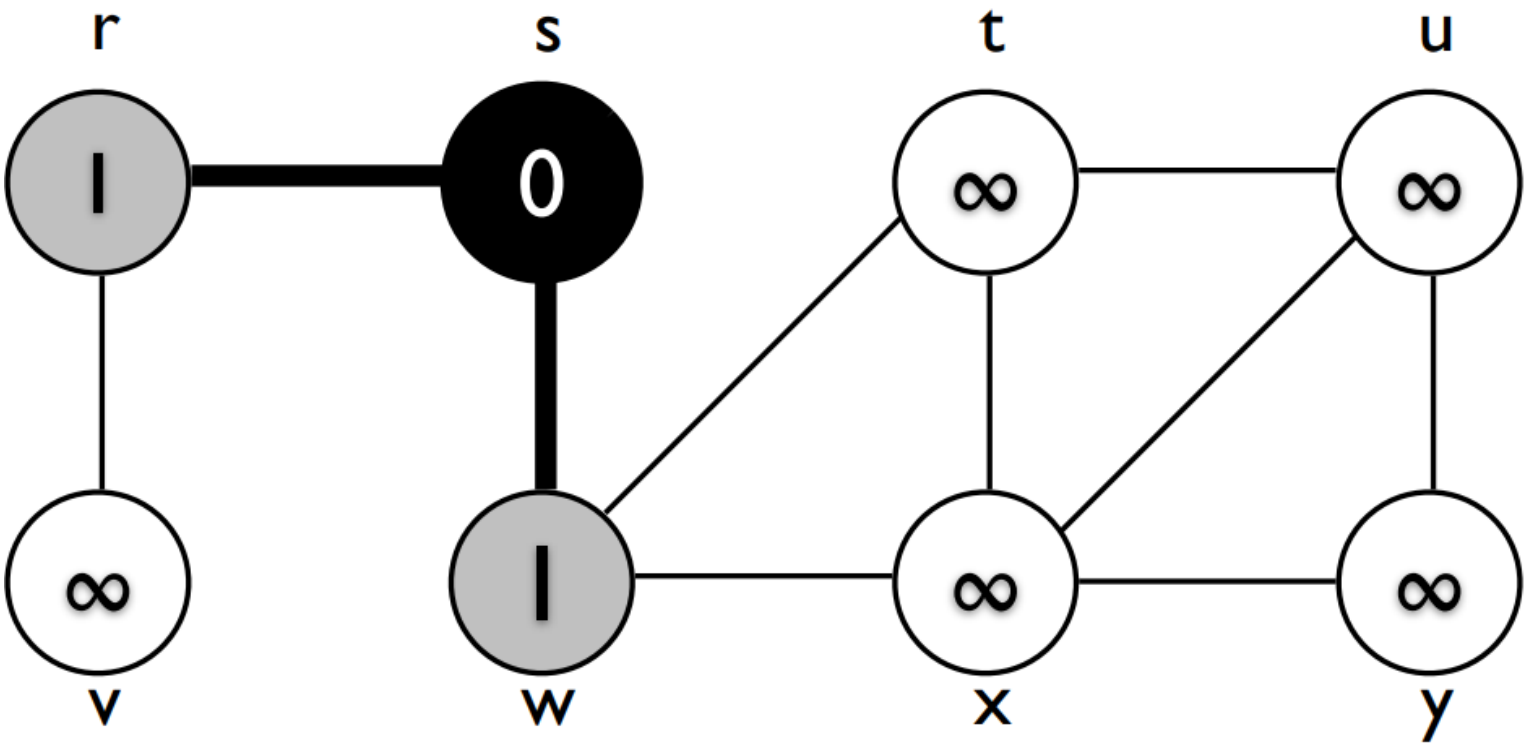
Fila Q

w	r

Nível

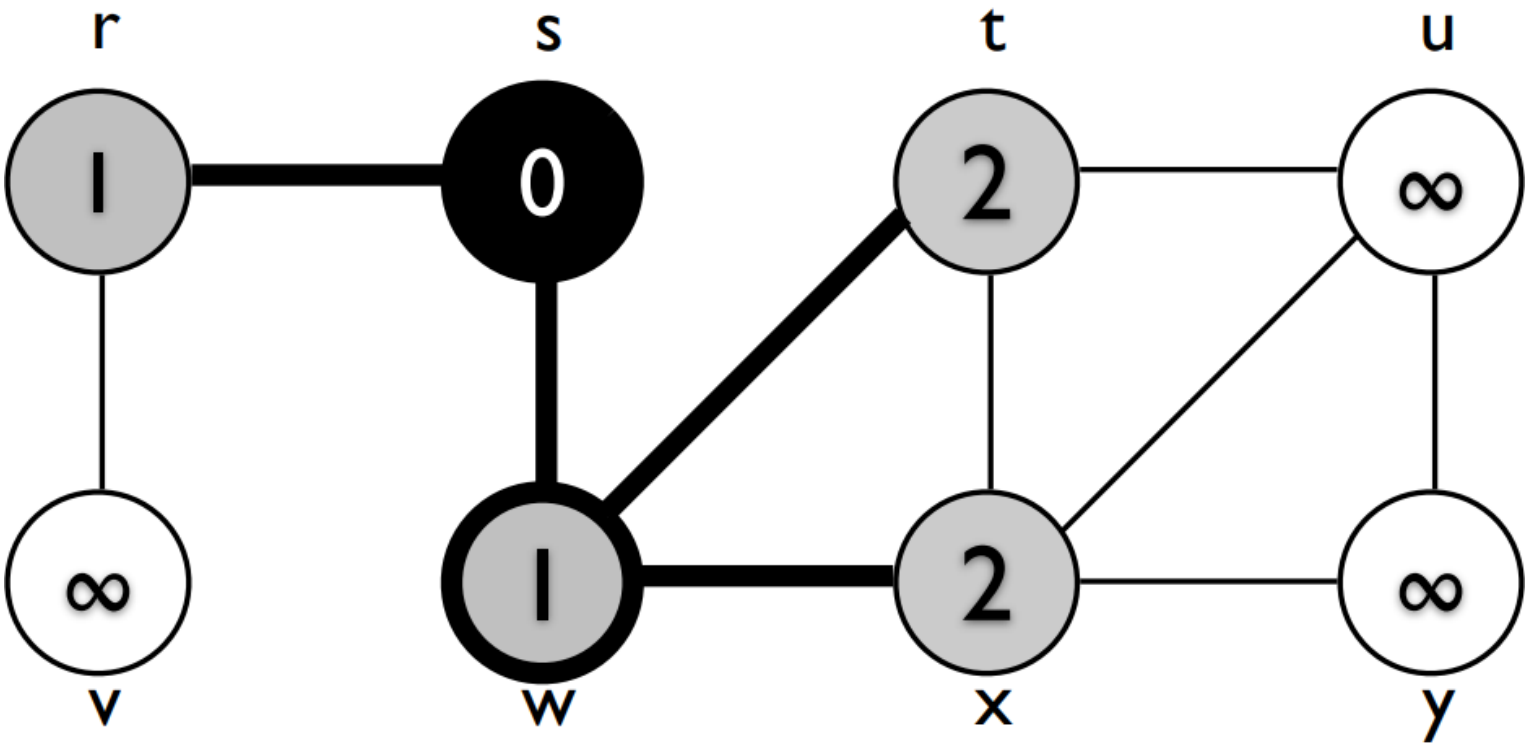
1	1

EXEMPLO



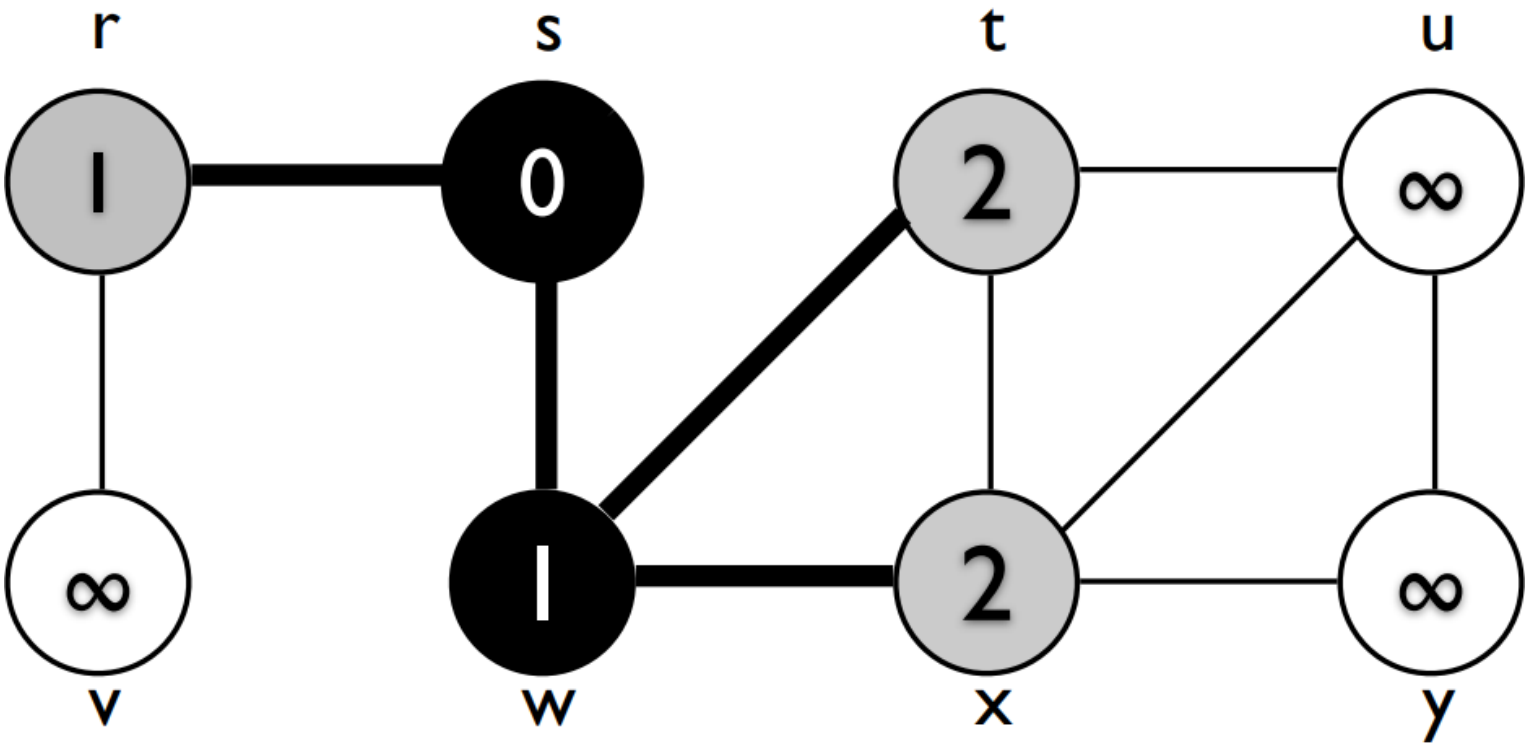
Fila Q	w	r
Nível	1	1

EXEMPLO



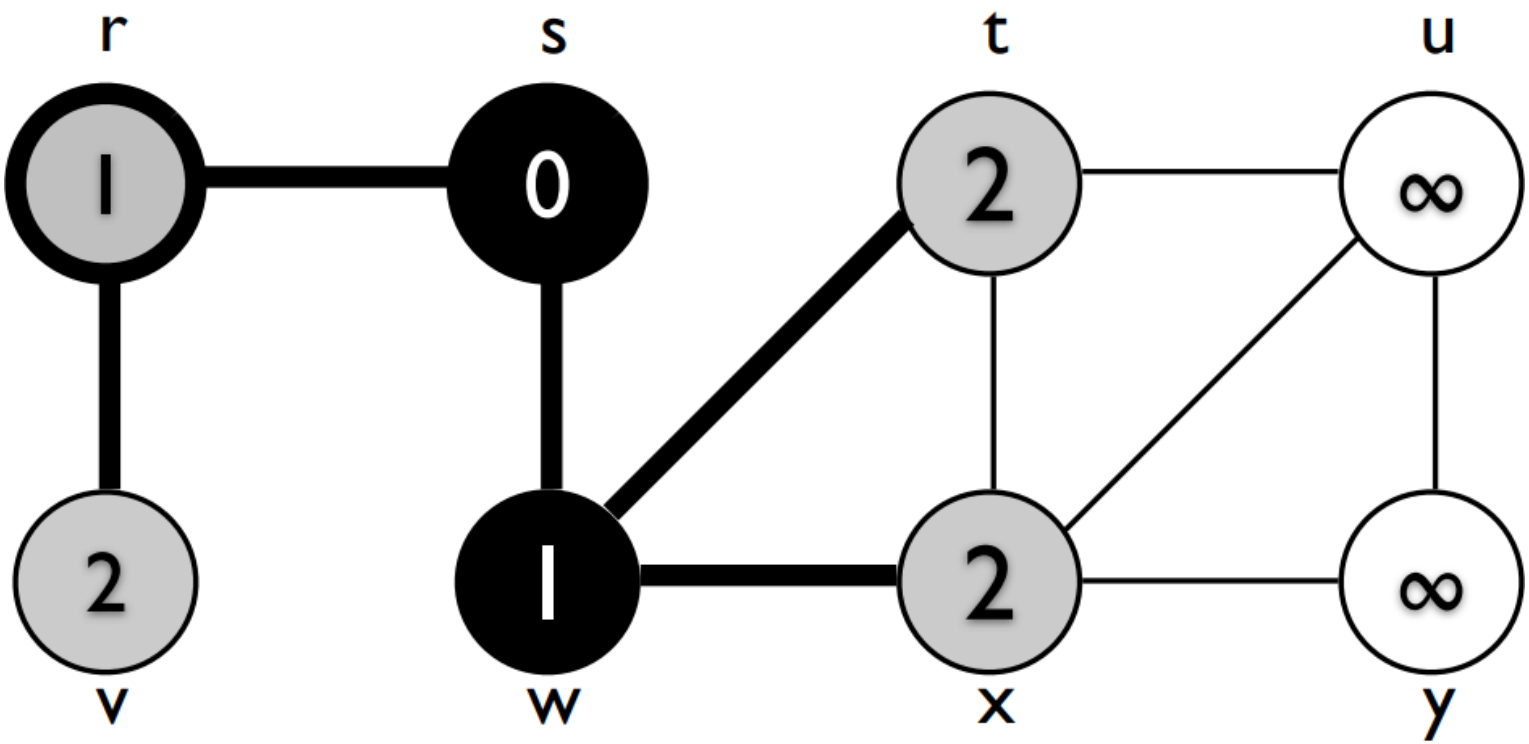
Fila Q	r	t	x
Nível	1	2	2

EXEMPLO



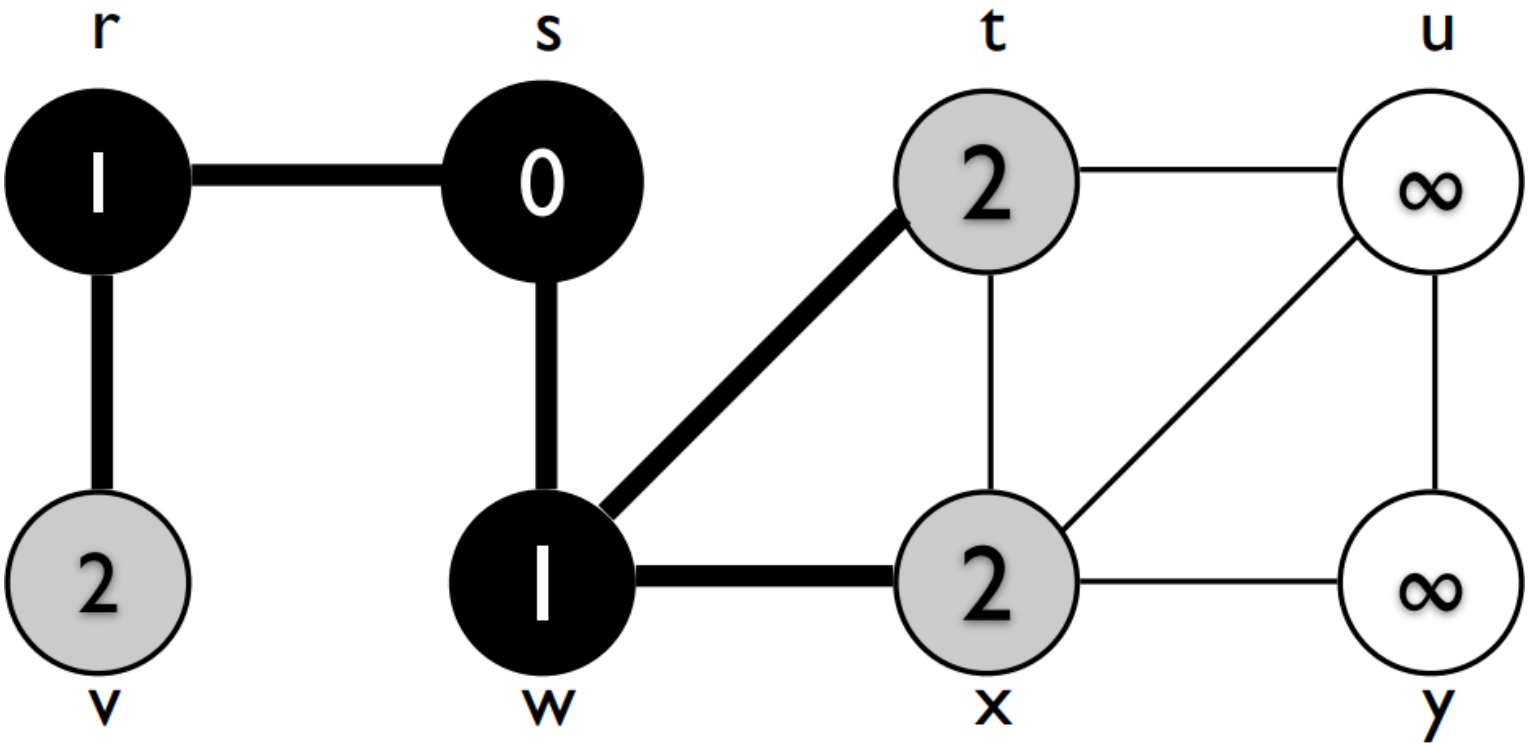
Fila Q	r	t	x
Nível	1	2	2

EXEMPLO



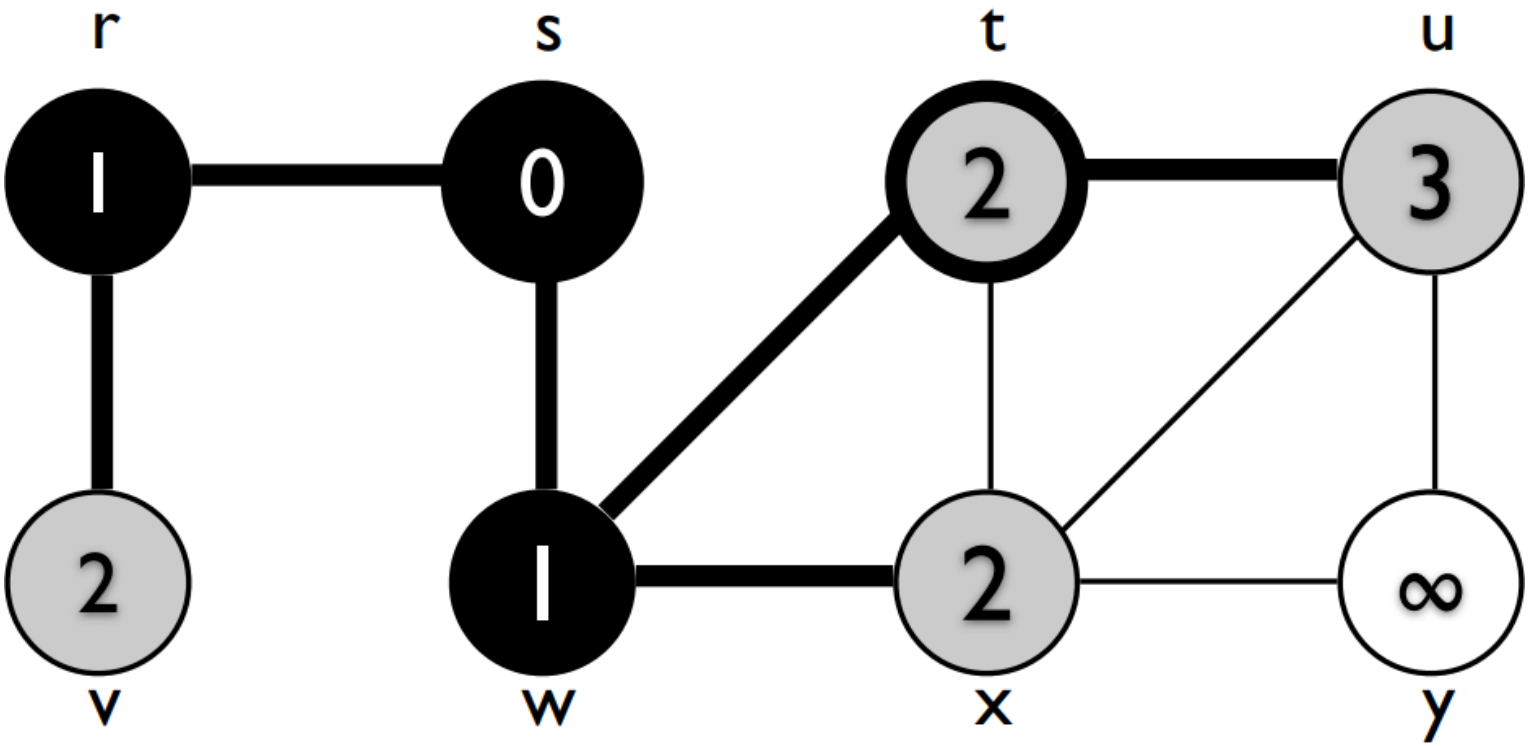
Fila Q	t	x	v
Nível	2	2	2

EXEMPLO



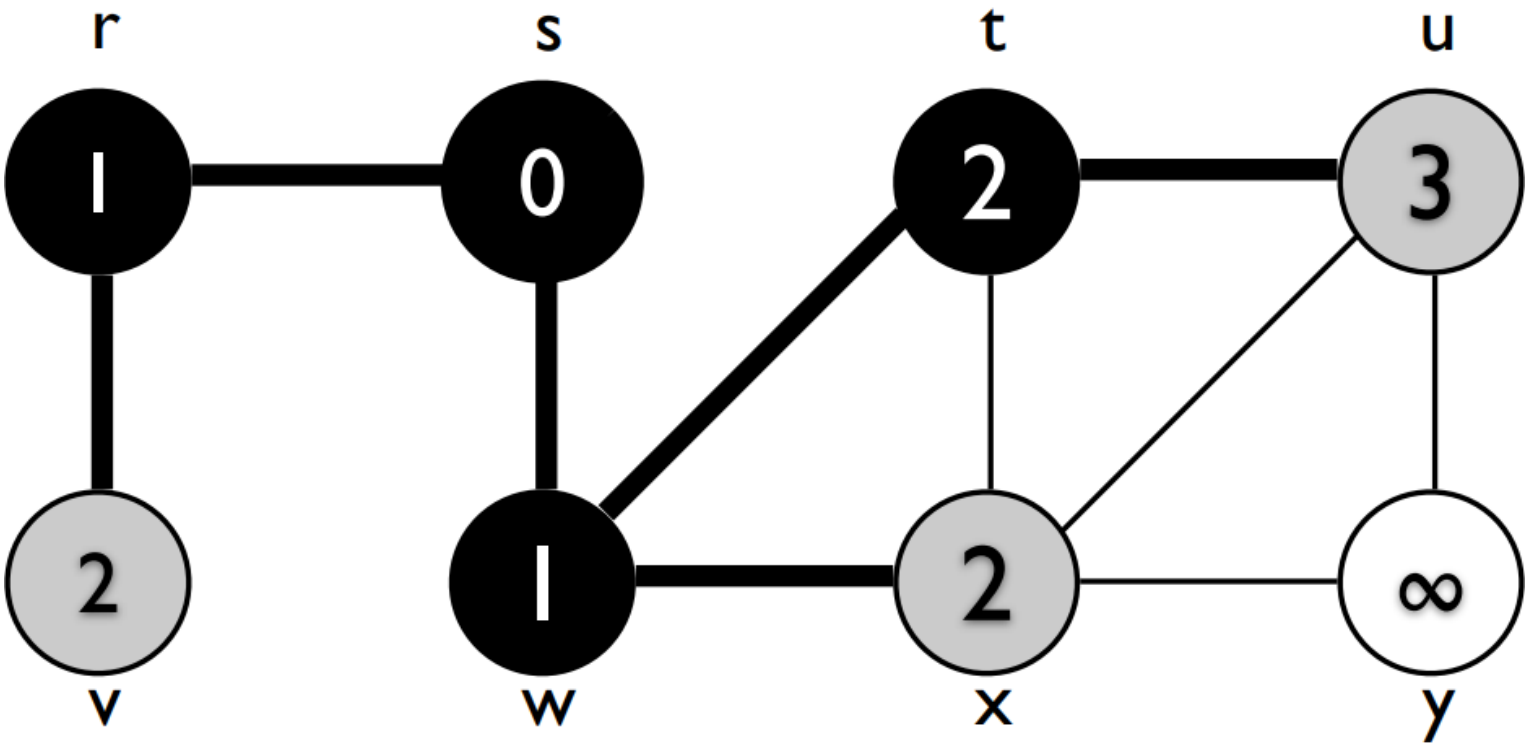
Fila Q	t	x	v
Nível	2	2	2

EXEMPLO



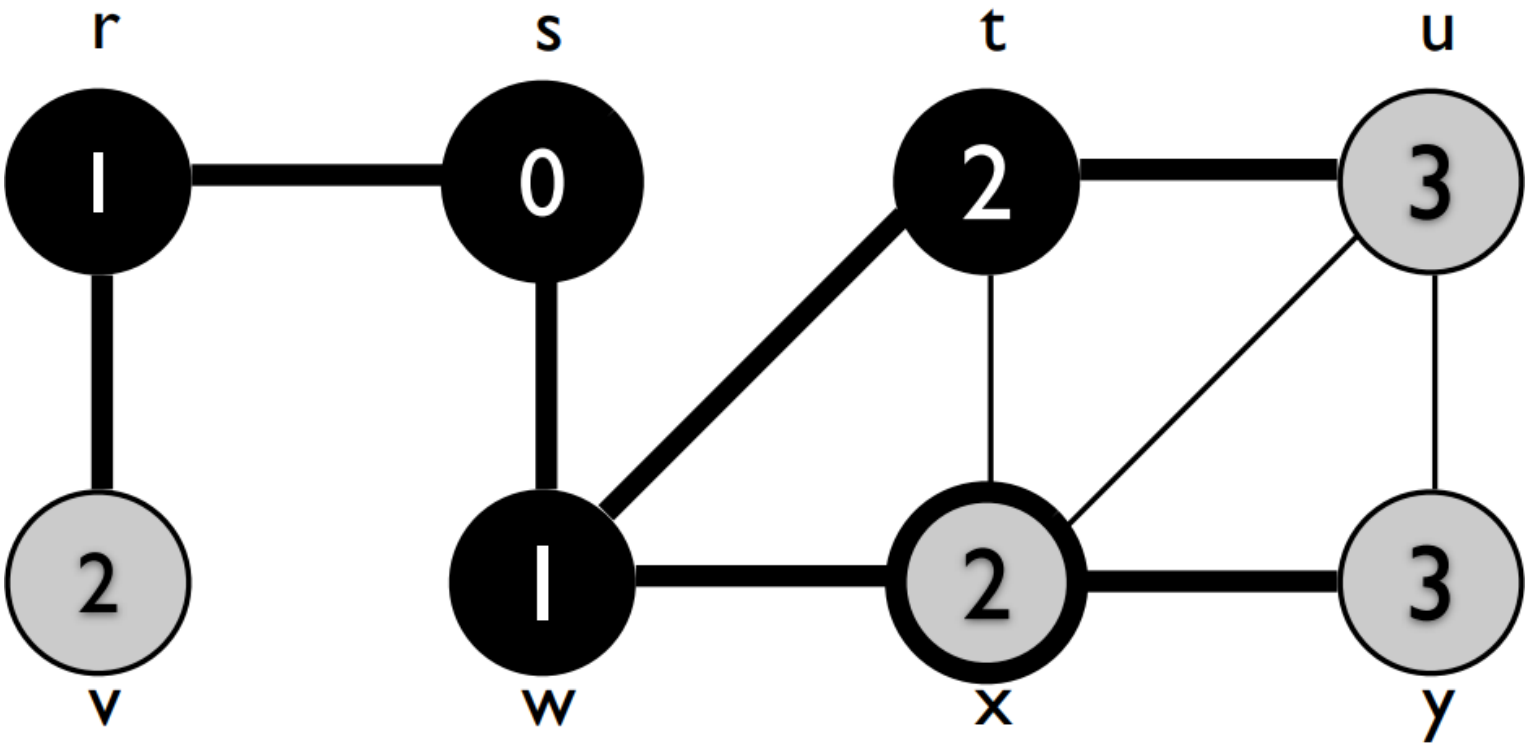
Fila Q	x	v	u
Nível	2	2	3

EXEMPLO



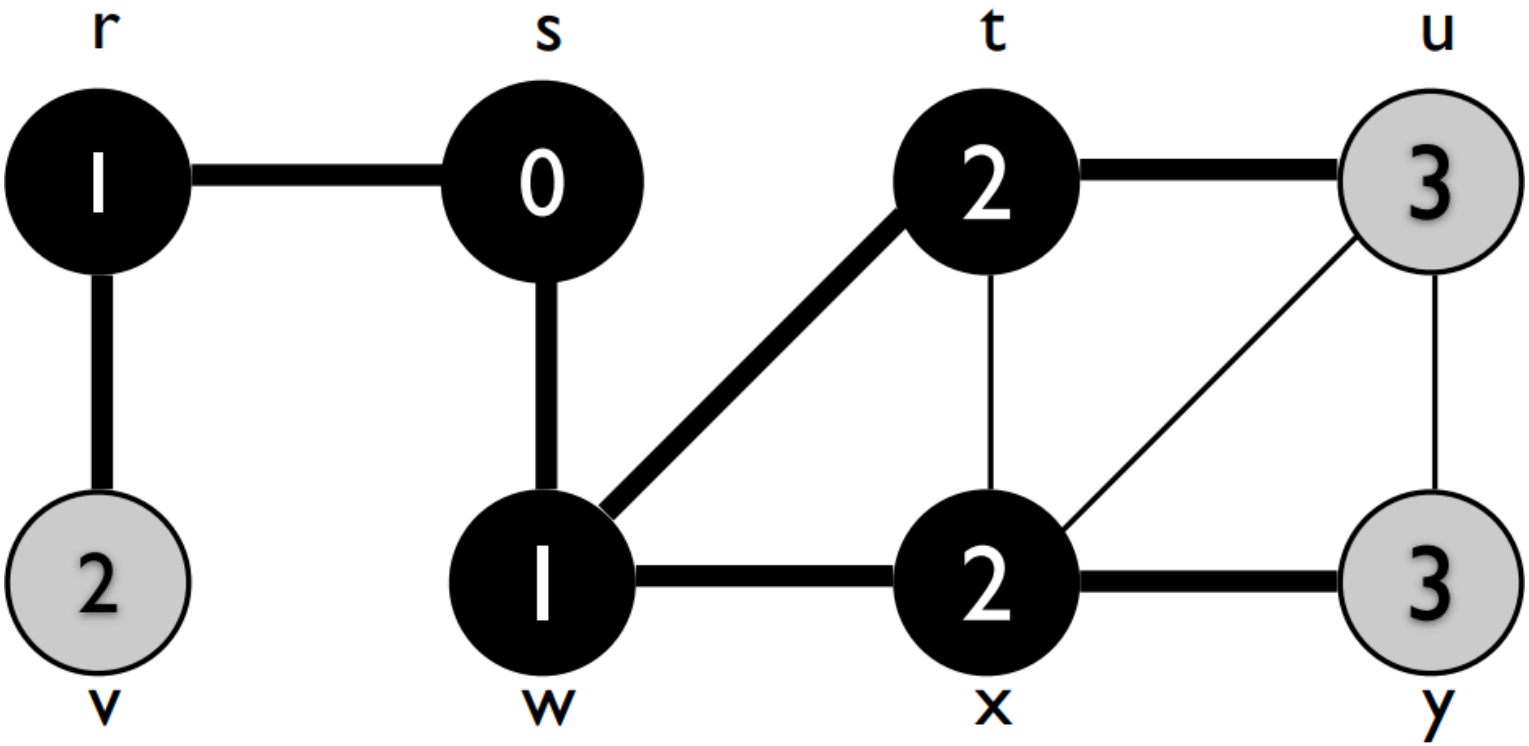
Fila Q	x	v	u
Nível	2	2	3

EXEMPLO



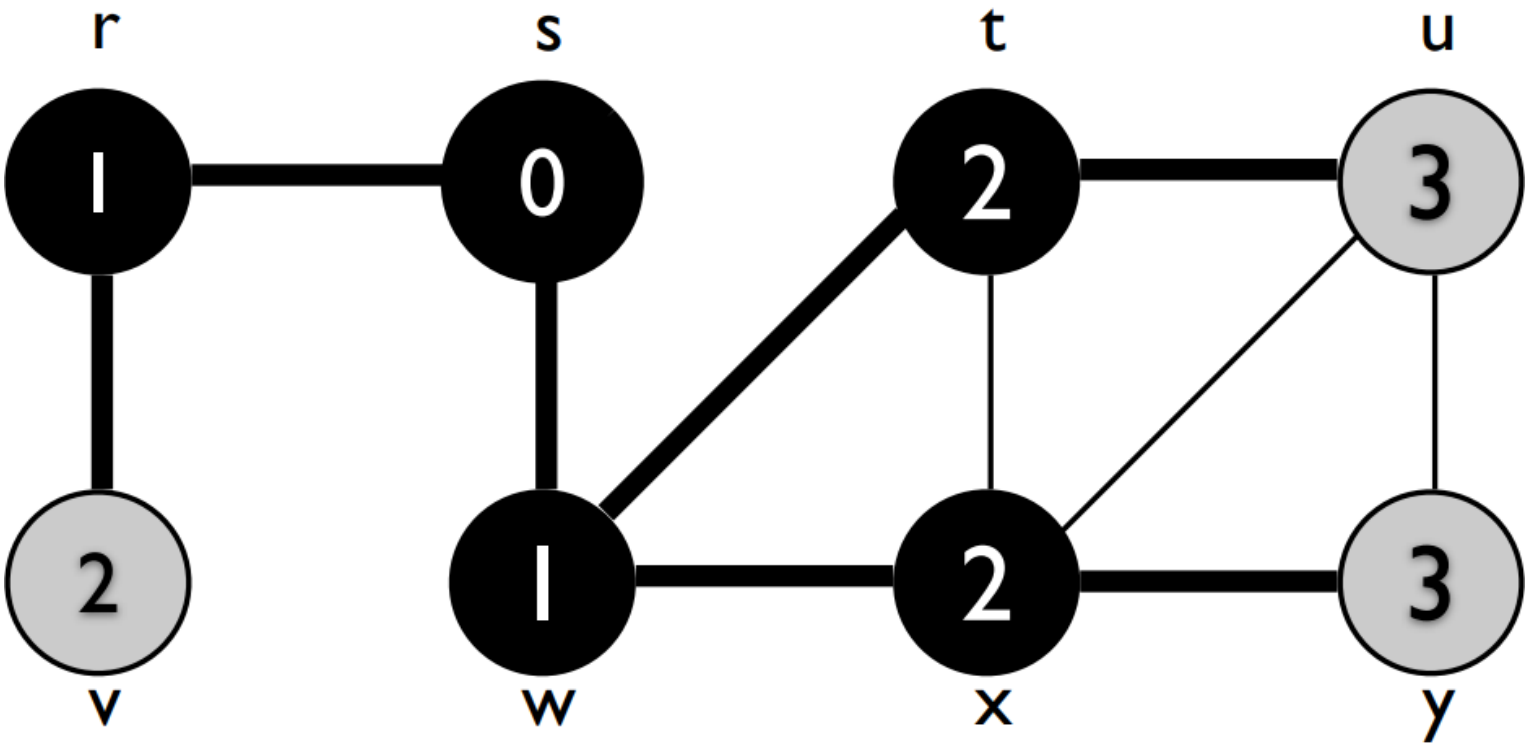
Fila Q	v	u	y
Nível	2	3	3

EXEMPLO



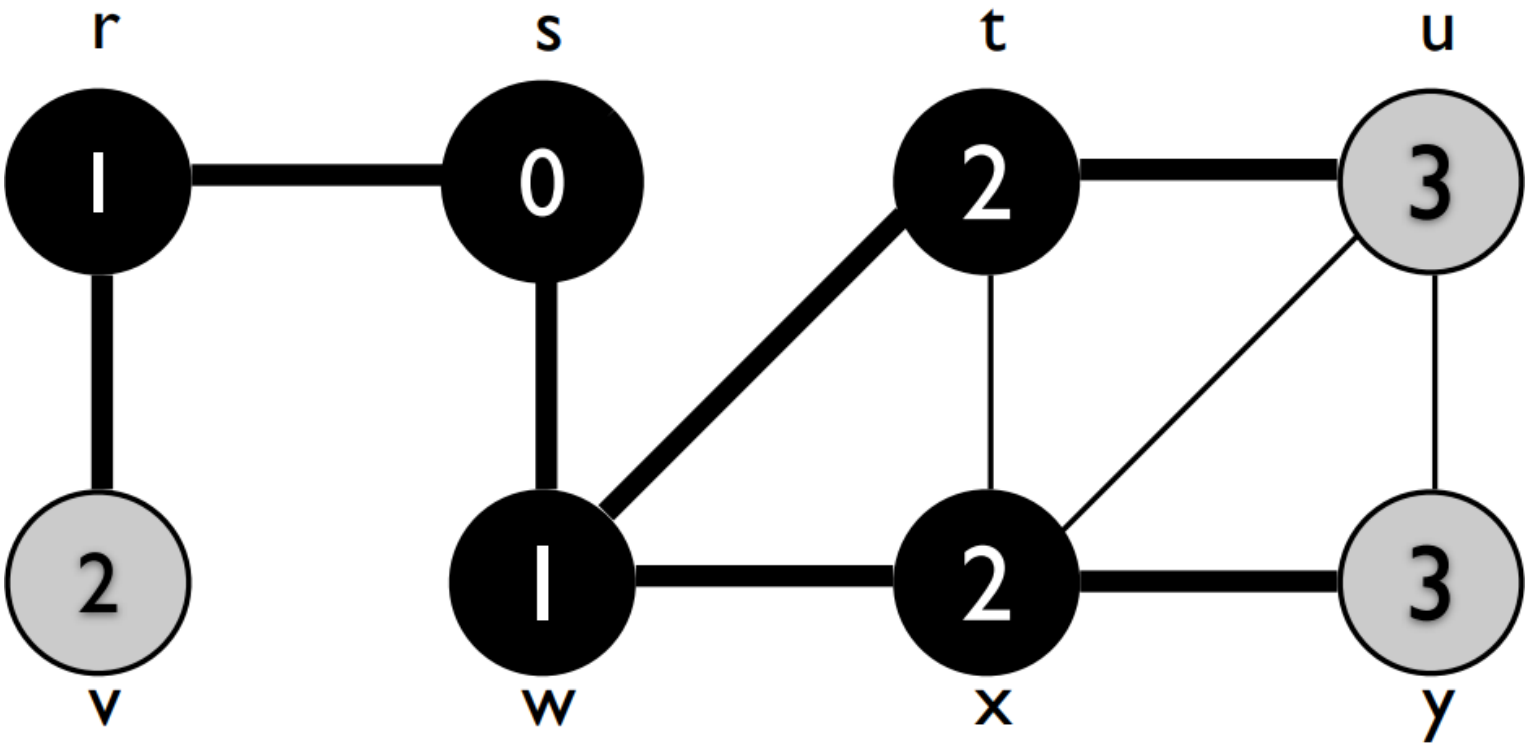
Fila Q	v	u	y
Nível	2	3	3

EXEMPLO



Fila Q	u	y
Nível	3	3

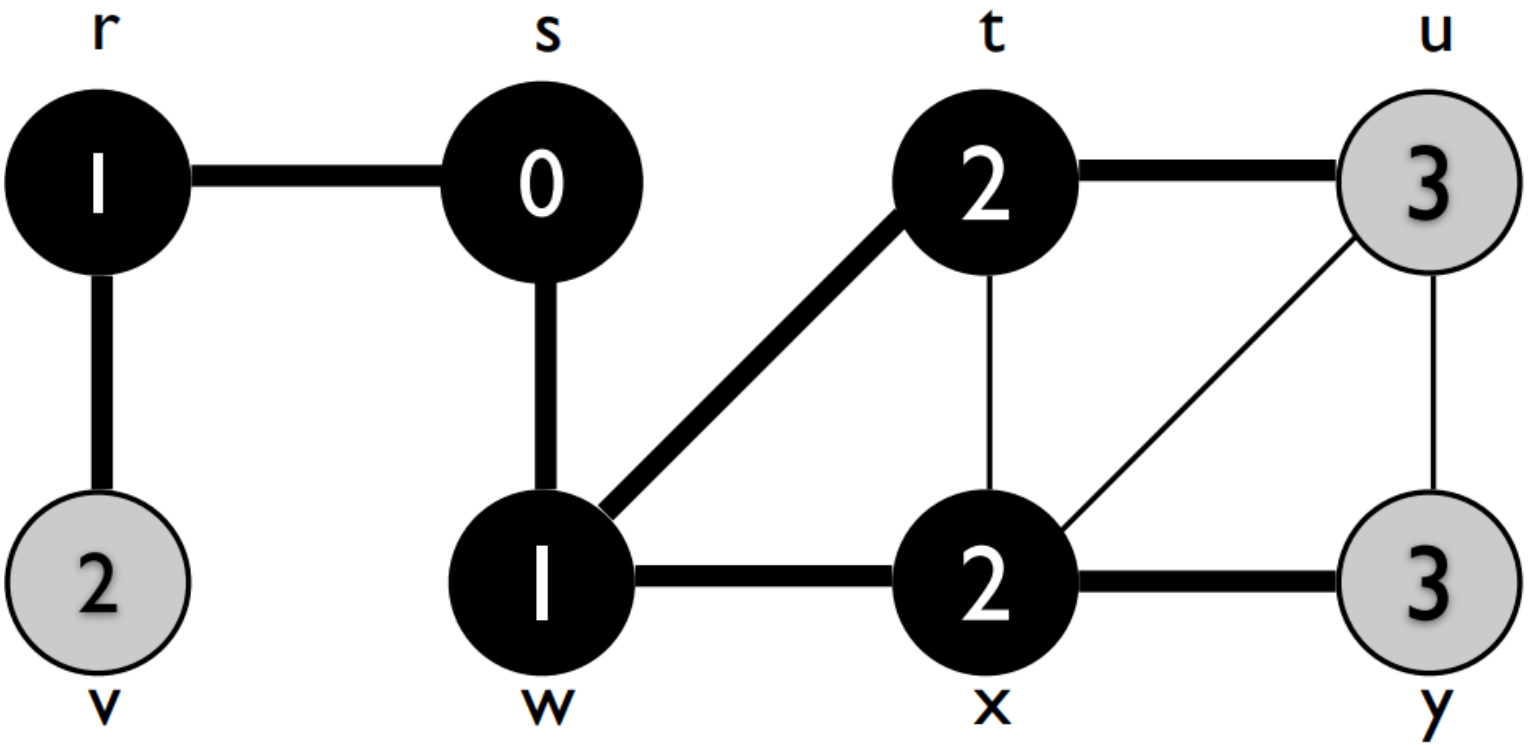
EXEMPLO



Fila Q | y

Nível | 3

EXEMPLO



Fila Q | ø

Nível | –

COMPLEXIDADE

A complexidade da busca em largura é a mesma do algoritmo de busca em profundidade

- $\mathcal{O}(|V| + |E|)$

FINALIZANDO...

PROPRIEDADES

Ambos os algoritmos funcionam bem em grafos direcionados e não direcionados

Entretanto, não consideram o peso das arestas (ou dos arcos)

A complexidade dos algoritmos depende da estrutura de dados utilizada para representar o grafo

- A complexidade apresentada é obtida utilizando uma lista de adjacência

Existe uma versão iterativa e uma recursiva para o algoritmo de busca em profundidade