Algoritmos e Estrutura de Dados II (AE23CP-3CP)

Aula #07 - Grafos - Algoritmos de Busca:
-Busca em Largura

Profª Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Algoritmos de busca em grafos

Busca em grafos - motivação

O que é busca em grafos?

Processo de seguir sistematicamente pelas arestas a fim de visitar os vértices do grafo.

Para que serve?

Pode-se obter várias informações sobre a estrutura do grafo, que podem ser úteis para projetar algoritmos eficientes para determinados problemas.

Algoritmos

Grafos são estruturas mais complicadas do que vetores, listas e árvores (binárias). Temos dois métodos para explorar/percorrer um grafo (orientado ou não-orientado):

- Busca em largura (*breadth-first search*)
- Busca em profundidade (*depth-first search*)

Busca em grafos - motivação

O que é busca em grafos?

Processo de seguir sistematicamente pelas arestas a fim de visitar os vértices do grafo.

Para que serve?

Pode-se obter várias informações sobre a estrutura do grafo, que podem ser úteis para projetar algoritmos eficientes para determinados problemas.

Algoritmos

Grafos são estruturas mais complicadas do que vetores, listas e árvores (binárias). Temos dois métodos para explorar/percorrer um grafo (orientado ou não-orientado):

- Busca em largura (*breadth-first search*)
- Busca em profundidade (*depth-first search*)

Busca em grafos - motivação

O que é busca em grafos?

Processo de seguir sistematicamente pelas arestas a fim de visitar os vértices do grafo.

Para que serve?

Pode-se obter várias informações sobre a estrutura do grafo, que podem ser úteis para projetar algoritmos eficientes para determinados problemas.

Algoritmos

Grafos são estruturas mais complicadas do que vetores, listas e árvores (binárias). Temos dois métodos para explorar/percorrer um grafo (orientado ou não-orientado):

- Busca em largura (*breadth-first search*)
- Busca em profundidade (*depth-first search*)

Busca em grafos - motivação

O que é busca em grafos?

Processo de seguir sistematicamente pelas arestas a fim de visitar os vértices do grafo.

Para que serve?

Pode-se obter várias informações sobre a estrutura do grafo, que podem ser úteis para projetar algoritmos eficientes para determinados problemas.

Algoritmos

Grafos são estruturas mais complicadas do que vetores, listas e árvores (binárias). Temos dois métodos para explorar/percorrer um grafo (orientado ou não-orientado):

- Busca em largura (*breadth-first search*)
- Busca em profundidade (*depth-first search*)

Notação

- Para um grafo G (orientado ou não) denotamos por $V[G]$ seu conjunto de vértices e por $E[G]$ seu conjunto de arestas.
- Para denotar complexidades nas expressões com O ou Θ usaremos V ou E em vez de $|V[G]|$ ou $|E[G]|$.

  Por exemplo, $\Theta(V + E)$ ou $O(V^2)$.

Busca em largura

Noções básicas

- Dizemos que um vértice v é **alcançável** a partir de um vértice s em um grafo G se existe um caminho de s a v em G.
- **Definição:** a distância de s a v é o comprimento de um caminho mais curto de s a v.
- Se v **não é alcançável** a partir de s, então dizemos que a distância de s a v é ∞ (infinita).

Algoritmo de busca em largura

Entrada

Recebe um grafo $G = (V, E)$ e um vértice especificado s chamado fonte (*source*).

Processamento

Percorre todos os vértices alcançáveis a partir de s em ordem de distância deste. Vértices a mesma distância podem ser percorridos em qualquer ordem.

Saída

Constrói uma **Árvore de Busca em Largura** com raiz s. Cada caminho de s a um vértice v nesta árvore corresponde a um caminho mais curto de s a v.

Algoritmo de busca em largura

Processamento

- Inicialmente a Árvore de Busca em Largura contém apenas o vértice fonte s.
- Para cada vizinho v de s, o vértice v e a aresta (s, v) são acrescentadas à árvore.
- O processo é repetido para os vizinhos dos vizinhos de s e assim por diante, até que todos os vértices atingíveis por s sejam inseridos na árvore.
- Este processo é implementado através de uma **fila** Q**.**

Algoritmo de busca em largura

Processamento

- Busca em largura atribui cores a cada vértice: branco, cinza e preto.
- Cor branca = "não visitado". Inicialmente todos os vértices são brancos.
- Cor cinza = "visitado pela primeira vez".
- Cor preta = "teve seus vizinhos visitados".

Algoritmo de busca em largura

Recebe um grafo G (na forma de listas de adjacências) e um vértice $s \in V[G]$ e devolve

(i) para cada vértice v, a distância de s a v em G e

(ii) uma Árvore de Busca em Largura.

```
BUSCA-EM-LARGURA(G, s)
 0  ▷ Inicialização
 1  para cada u ∈ V[G] − {s} faça
 2       cor[u] ← branco
 3       d[u] ← ∞
 4       π[u] ← NIL
 5  cor[s] ← cinza
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  enquanto Q ≠ ∅ faça
11       u ← DEQUEUE(Q)
12       para cada v ∈ Adj[u] faça
13            se cor[v] = branco então
14                 cor[v] ← cinza
15                 d[v] ← d[u] + 1
16                 π[v] ← u
17                 ENQUEUE(Q, v)
18       cor[u] ← preto
```

Algoritmo de busca em largura

Recebe um grafo G (na forma de listas de adjacências) e um vértice $s \in V[G]$ e devolve

(i) para cada vértice v, a distância de s a v em G e

(ii) uma Árvore de Busca em Largura.

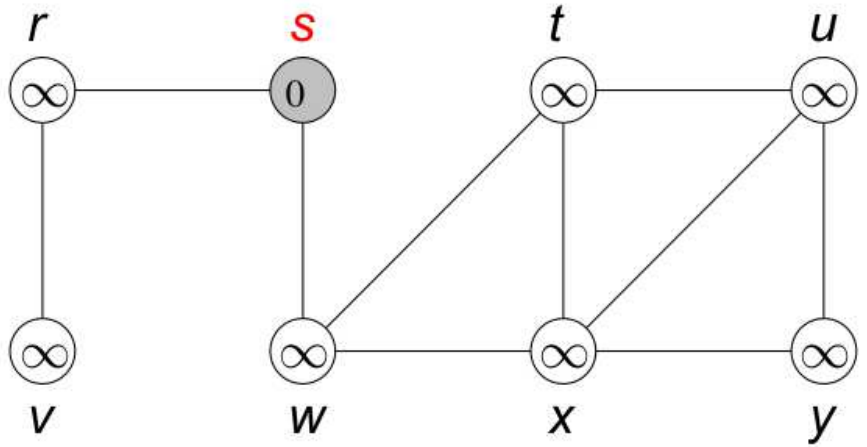
```
BUSCA-EM-LARGURA(G, s)
 0  ▷ Inicialização
 1  para cada u ∈ V[G] − {s} faça
 2       cor[u] ← branco
 3       d[u] ← ∞
 4       π[u] ← NIL
 5  cor[s] ← cinza
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  enquanto Q ≠ ∅ faça
11       u ← DEQUEUE(Q)
12       para cada v ∈ Adj[u] faça
13            se cor[v] = branco então
14                 cor[v] ← cinza
15                 d[v] ← d[u] + 1
16                 π[v] ← u
17                 ENQUEUE(Q, v)
18       cor[u] ← preto
```

Árvore de busca em largura

Algoritmo busca em largura - exemplo

```
BUSCA-EM-LARGURA(G, s)
 0  ▷ Inicialização
 1  para cada u ∈ V[G] − {s} faça
 2       cor[u] ← branco
 3       d[u] ← ∞
 4       π[u] ← NIL
 5  cor[s] ← cinza
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
```

	r	s	t	u	v	x	y	w
d	∞	0	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL



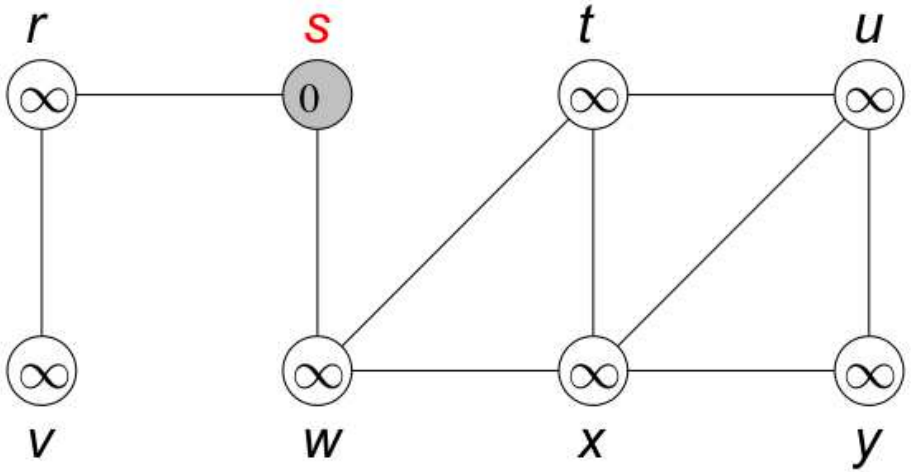
Q | s

0

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

	r	s	t	u	v	x	y	w
d	∞	0	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL



Q | s

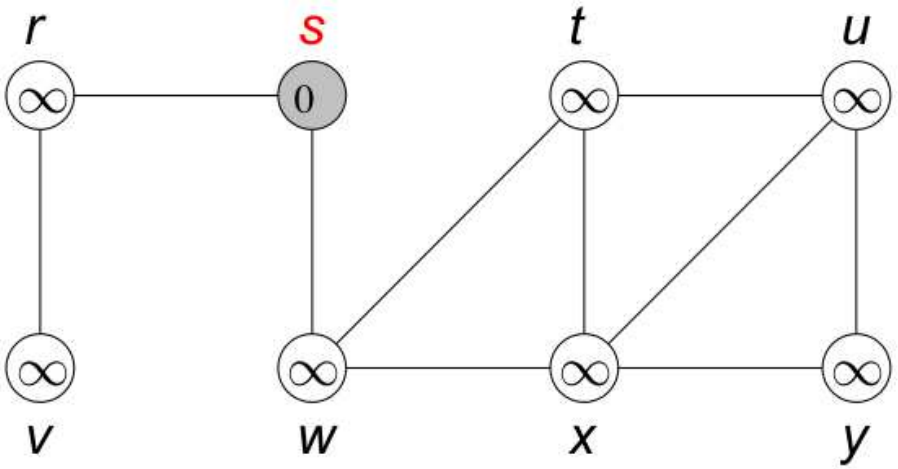
0

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = s

	r	s	t	u	v	x	y	w
d	∞	0	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL



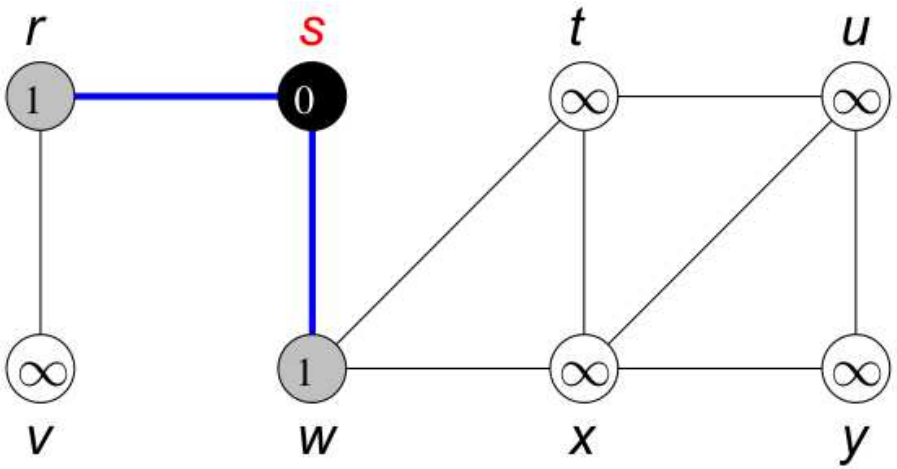
Q | s | 0

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = s

	r	s	t	u	v	x	y	w
d	1	0	∞	∞	∞	∞	∞	1
π	s	NIL	NIL	NIL	NIL	NIL	NIL	s



Q | w | r |

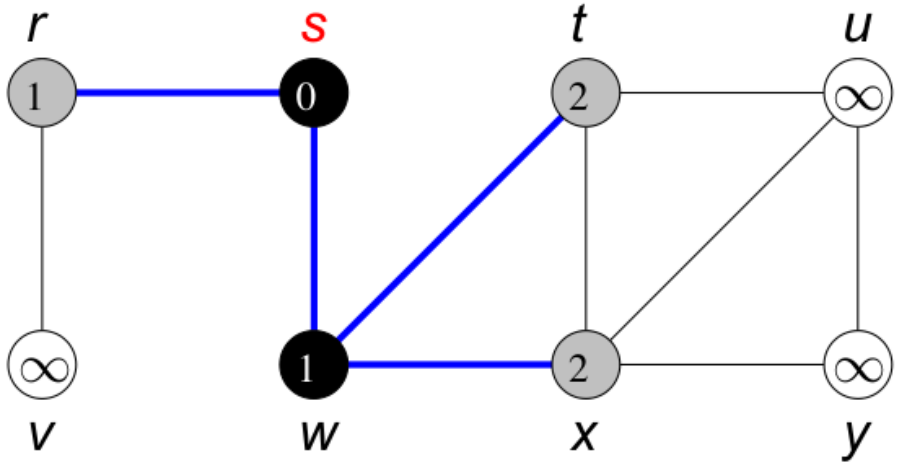
1 1

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = w

	r	s	t	u	v	x	y	w
d	1	0	2	∞	∞	2	∞	1
π	s	NIL	w	NIL	NIL	w	NIL	s



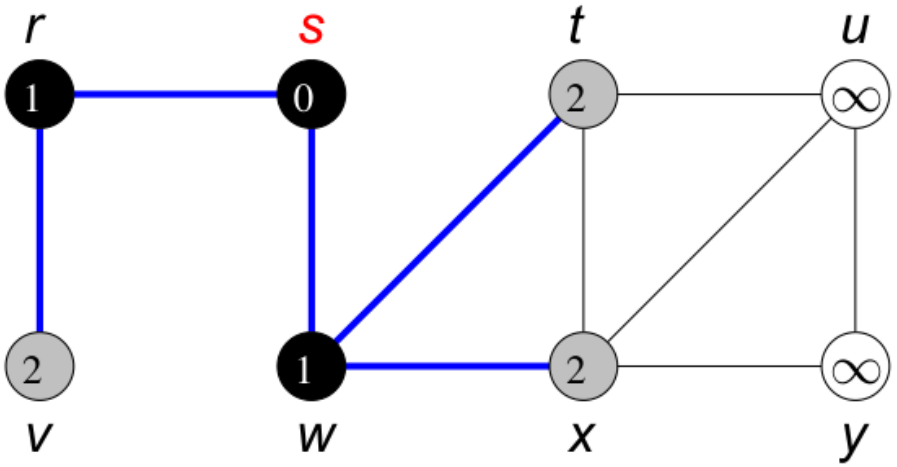
Q	r	t	x
	1	2	2

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = r

	r	s	t	u	v	x	y	w
d	1	0	2	∞	2	2	∞	1
π	s	NIL	w	NIL	r	w	NIL	s



Q | t | x | v

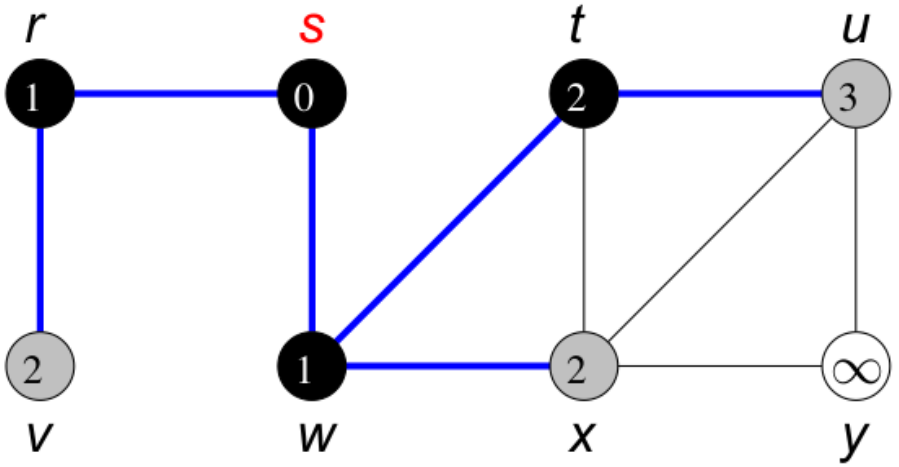
2 2 2

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = t

	r	s	t	u	v	x	y	w
d	1	0	2	3	2	2	∞	1
π	s	NIL	w	t	r	w	NIL	s



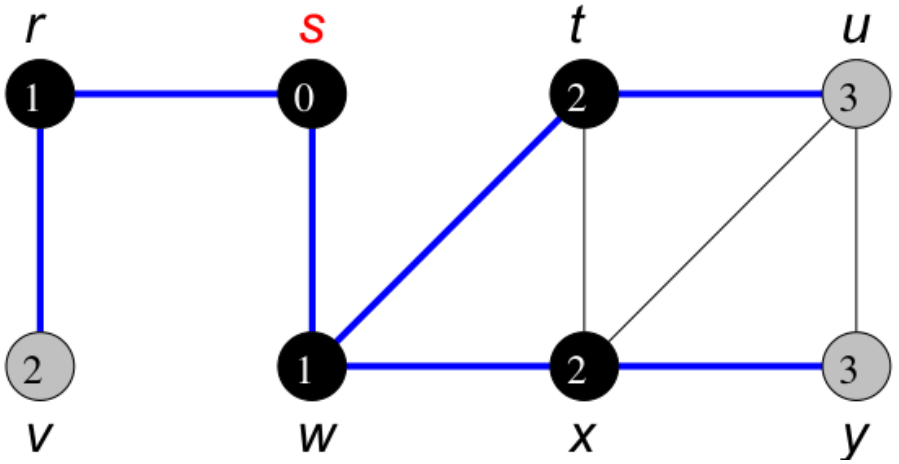
Q	x	v	u
	2	2	3

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = x

	r	s	t	u	v	x	y	w
d	1	0	2	3	2	2	3	1
π	s	NIL	w	t	r	w	x	s



Q | v | u | y |

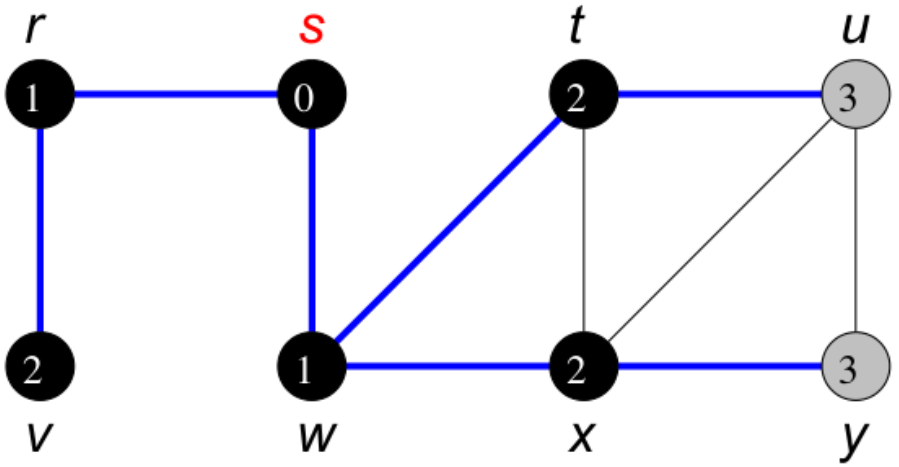
2 3 3

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = v

	r	s	t	u	v	x	y	w
d	1	0	2	3	2	2	3	1
π	s	NIL	w	t	r	w	x	s



Q | u | y |

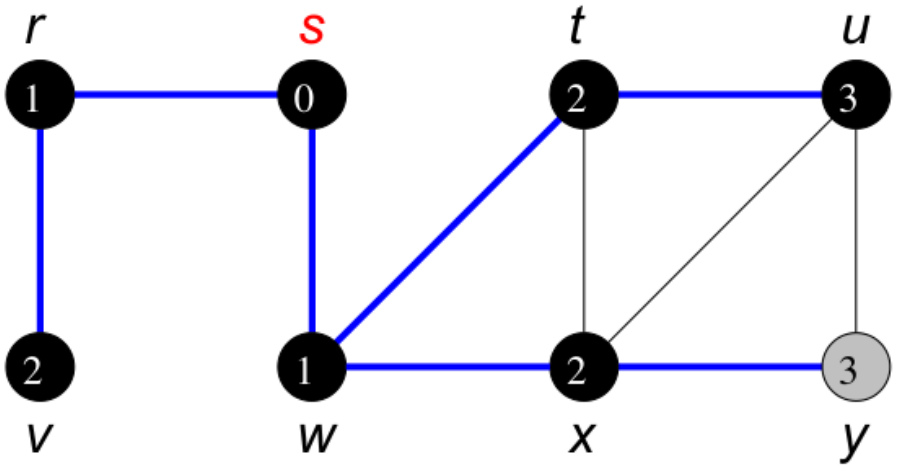
3 3

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = u

	r	s	t	u	v	x	y	w
d	1	0	2	3	2	2	3	1
π	s	NIL	w	t	r	w	x	s



Q | y |

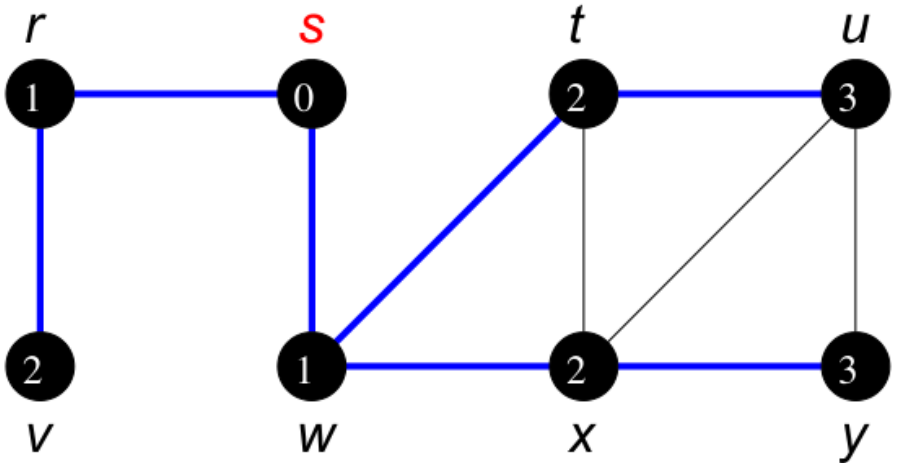
3

Algoritmo busca em largura - exemplo

```
10  enquanto Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13          se cor[v] = branco então
14              cor[v] ← cinza
15              d[v] ← d[u] + 1
16              π[v] ← u
17              ENQUEUE(Q, v)
18      cor[u] ← preto
```

u = y

	r	s	t	u	v	x	y	w
d	1	0	2	3	2	2	3	1
π	s	NIL	w	t	r	w	x	s



Q ∅

Algoritmo busca em largura

Cores

- Para cada vértice v guarda-se sua cor atual cor[v] que pode ser branco, cinza ou preto.
- Para efeito de implementação, isto não é realmente necessário, mas facilita o entendimento do algoritmo.

Representação da árvore e das distâncias

- A raiz da Árvore de Busca em Largura é s.
- Cada vértice v (diferente de s) possui um pai $\pi[v]$.
- O caminho de s a v na Árvore é dado por: $v, \pi[v], \pi[\pi[v]], \pi[\pi[\pi[v]]], \ldots, s$.
- Uma variável $d[v]$ é usada para armazenar a distância de s a v (que será determinada durante a busca).

Algoritmo busca em largura - complexidade

Consumo de tempo

- A inicialização consome tempo $\Theta(V)$.
- Depois que um vértice deixa de ser branco, ele não volta a ser branco novamente. Assim, cada vértice é inserido na fila Q no máximo uma vez. Cada operação sobre a fila consome tempo $\Theta(1)$ resultando em um total de $O(V)$.
- Em uma lista de adjacência, cada vértice é percorrido apenas uma vez. A soma dos comprimentos das listas é $\Theta(E)$. Assim, o tempo gasto para percorrer as listas é $O(E)$.

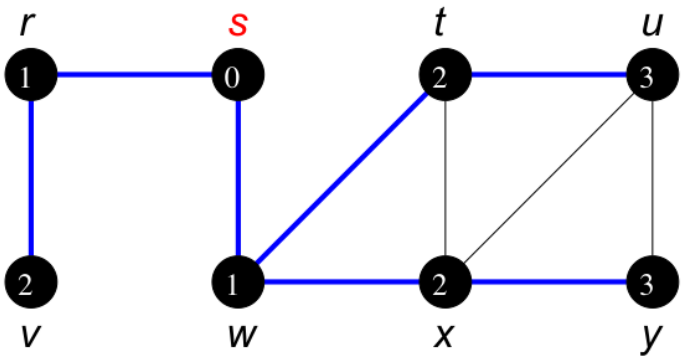
Conclusão

A complexidade de tempo de BUSCA-EM-LARGURA é $O(V + E)$.

Caminho mais curto

Imprime um caminho mais curto de s a v.

```
Print-Path(G, s, v)
1   se v = s então
2       imprime s
3   senão
4       se π[v] = NIL então
4           imprime não existe caminho de s a v.
5       senão
6           Print-Path(G, s, π[v])
7           imprime v.
```



executanto...

Print-Path(G, s, u)

Print-Path(G, s, t)

Print-Path(G, s, w)

Print-Path(G, s, s)

imprime s

imprime w

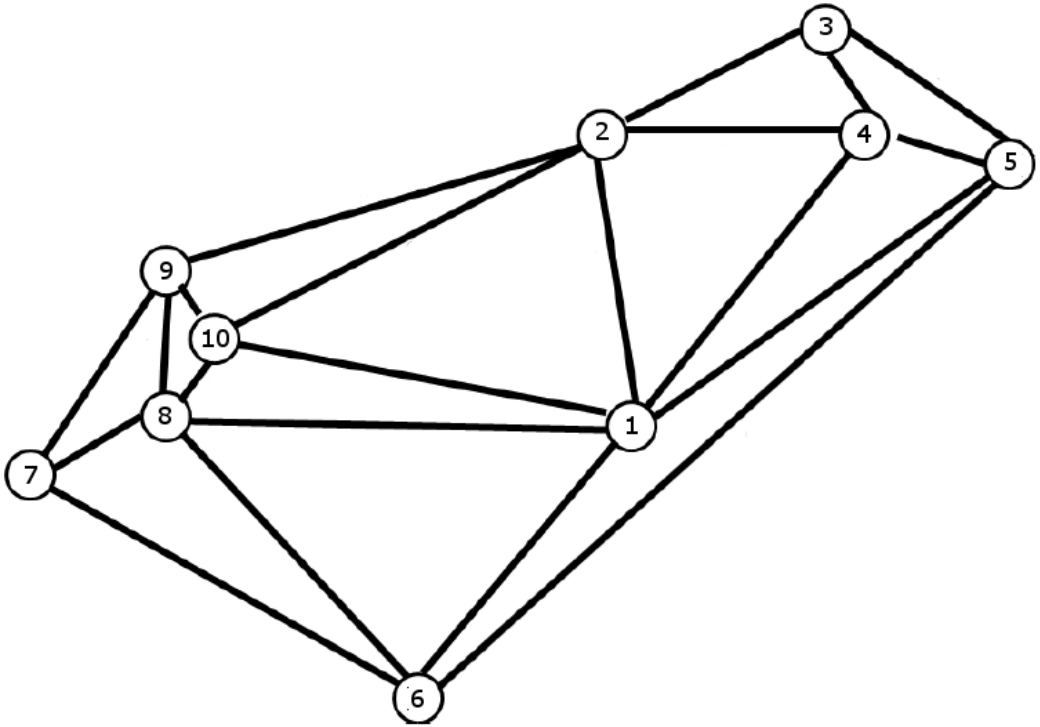
imprime t

imprime u

	r	s	t	u	v	x	y	w
π	s	NIL	w	t	r	w	x	s

Exercício

Considere o grafo abaixo, em seguida faça:



- Execute a busca em largura sobre o grafo acima, considerando o nó 1 como origem.
- Implemente o algoritmo de **busca em largura** em linguagem C.