

INTELIGÊNCIA ARTIFICIAL

PARTE 2

Grafos e Algoritmos de Busca Exaustiva

- *Breadth-First Search*
- *Depth-First Search*
- *Greedy Search*



1

Introdução aos Grafos

1 - Introdução aos Grafos

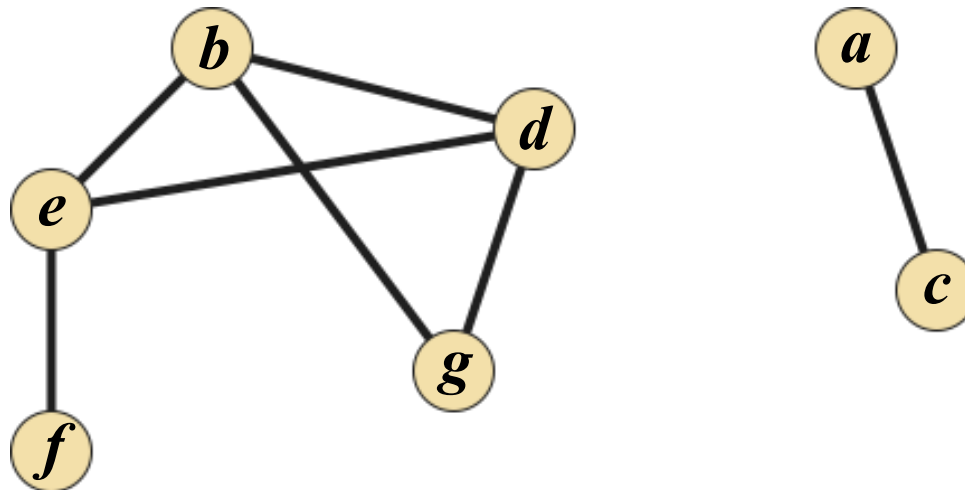
Definições:

- Grafos são estruturas representadas por $G(V, E)$, onde V é um conjunto não vazio de **vértices** e E é um conjunto de pares não orientados de V , chamado **arestas** (*edges*). Cada aresta é formada por um par de vértices.

Exemplo:

$$V = \{a, b, c, d, e, f, g\}$$

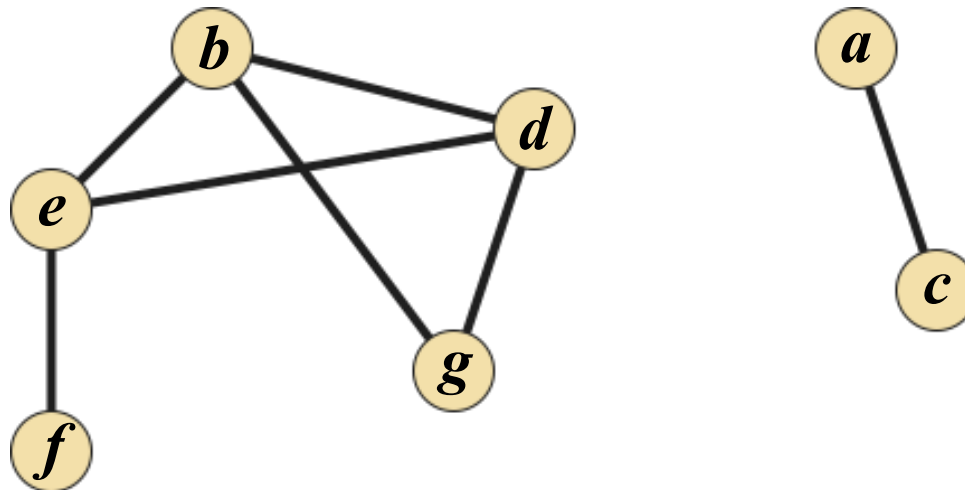
$$E = \{(a, c), (b, d), (b, e), (b, g), (d, e), (d, g), (e, f)\}$$



1 - Introdução aos Grafos

Propriedades:

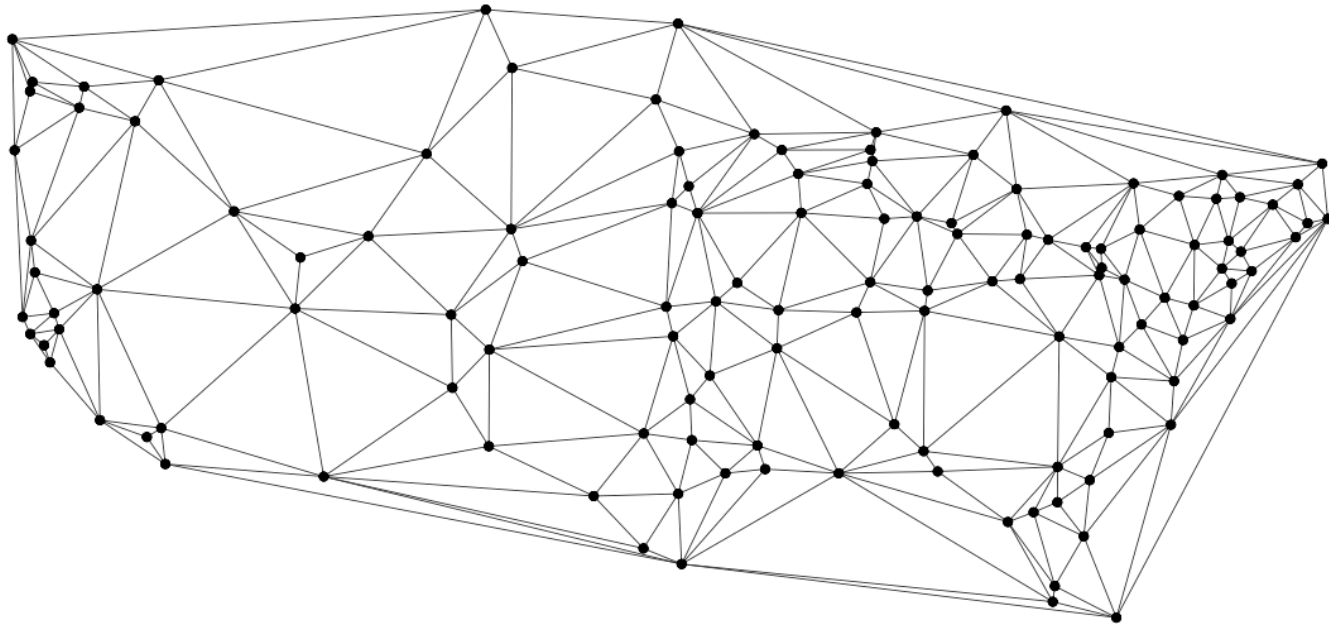
- Vértices a e c são **adjacentes**. Vértices f e g não!
- Vértice b e aresta (b,g) são **incidentes**. Vértice f e aresta (b,g) não!
- A vizinhança de b é $N(b)=\{d,e,g\}$
- O grau do vértice b é 3.



1 - Introdução aos Grafos

Grafo Não-orientado:

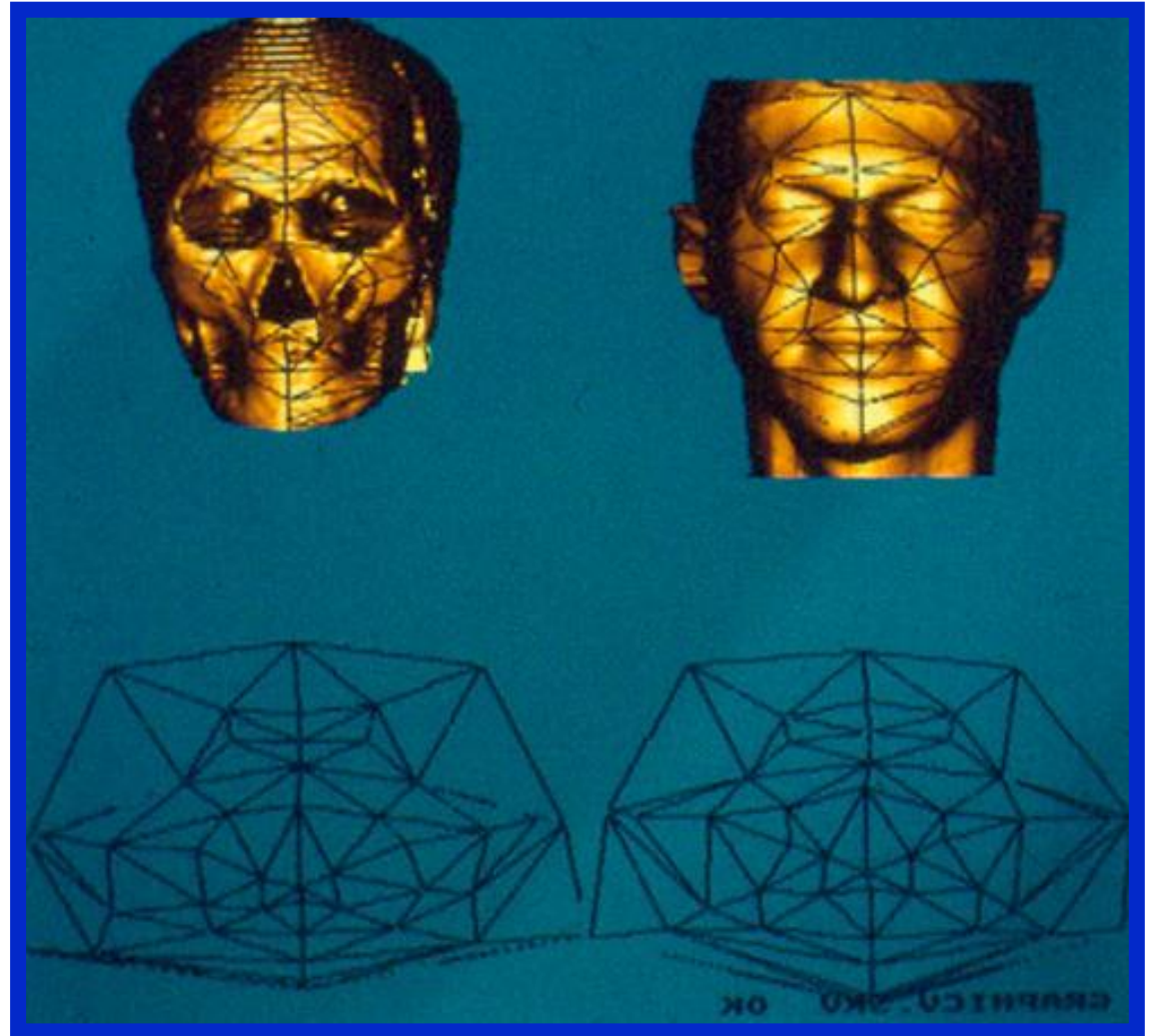
- Arestas são pares não-orientados (v,w) , ou seja, a aresta (v,w) e a aresta (w,v) são a mesma aresta.
- Não há direção!



Este grafo representa 128 cidades dos EUA
(extraído do livro *The Stanford GraphBase*.)

1 - Introdução aos Grafos

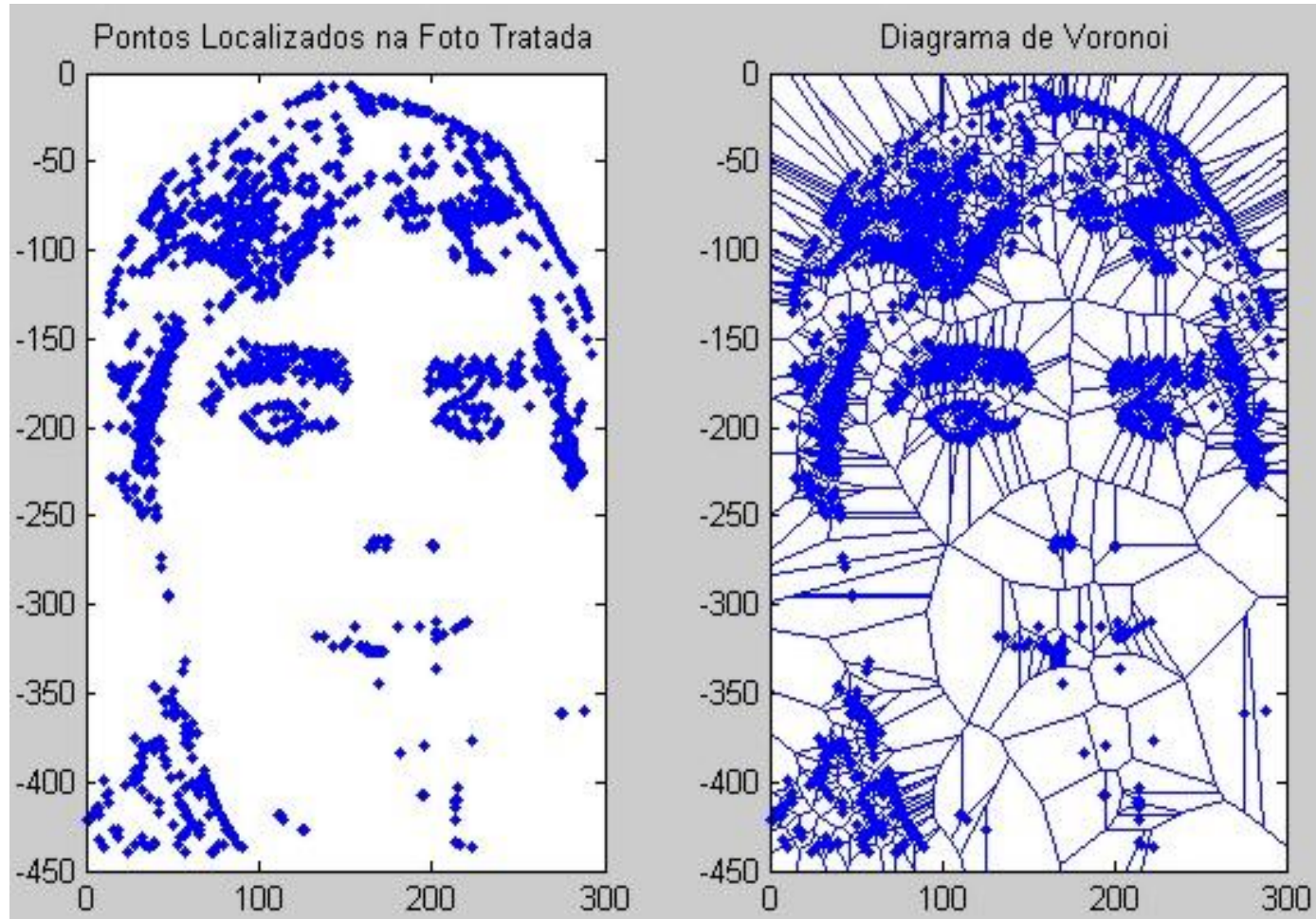
Grafo Não-orientado:



Exemplo de
Triangularização de Delaunay
extraído das aulas de
Geometria Computacional do
Prof. Dr. Paulo Sérgio Rodrigues
(FEI/2010)

1 - Introdução aos Grafos

Grafo Não-orientado:

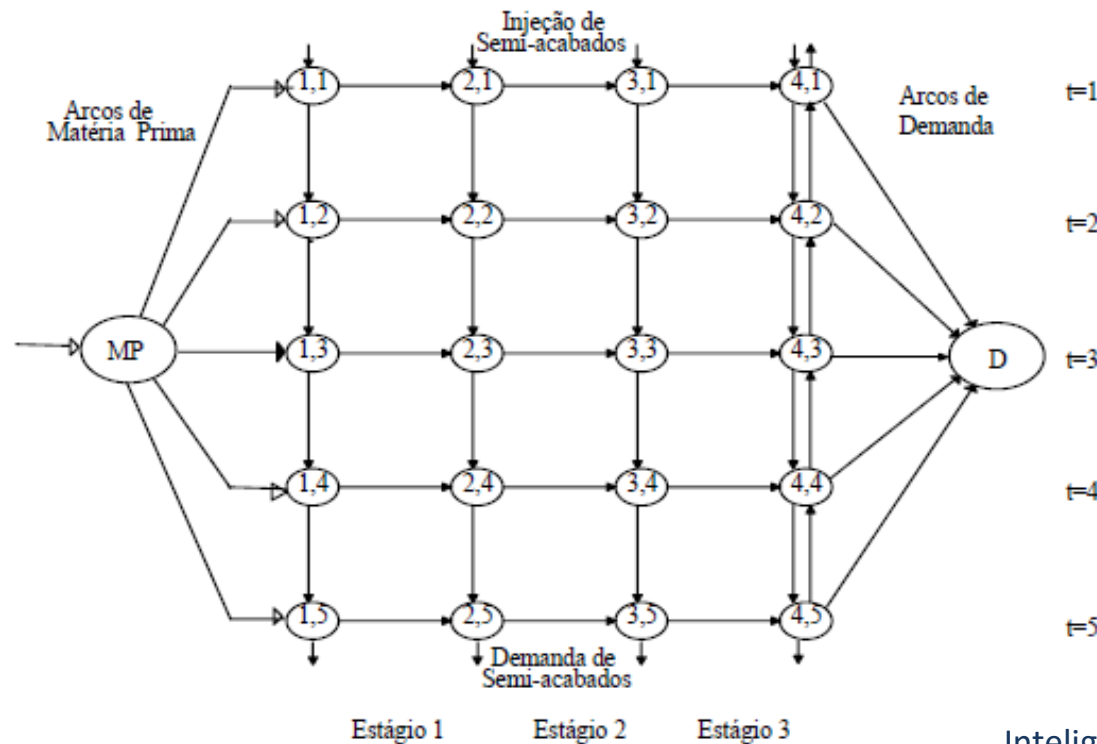


Exemplo de
Diagrama de Voronoi

1 - Introdução aos Grafos

Grafo Orientado:

- Arestas são pares orientados (v,w) , ou seja, a aresta (v,w) e a aresta (w,v) são diferentes.
- Aresta (v,w) é direcionada $v \rightarrow w$, onde v é a origem (*source*) e w é o alvo (*target*).

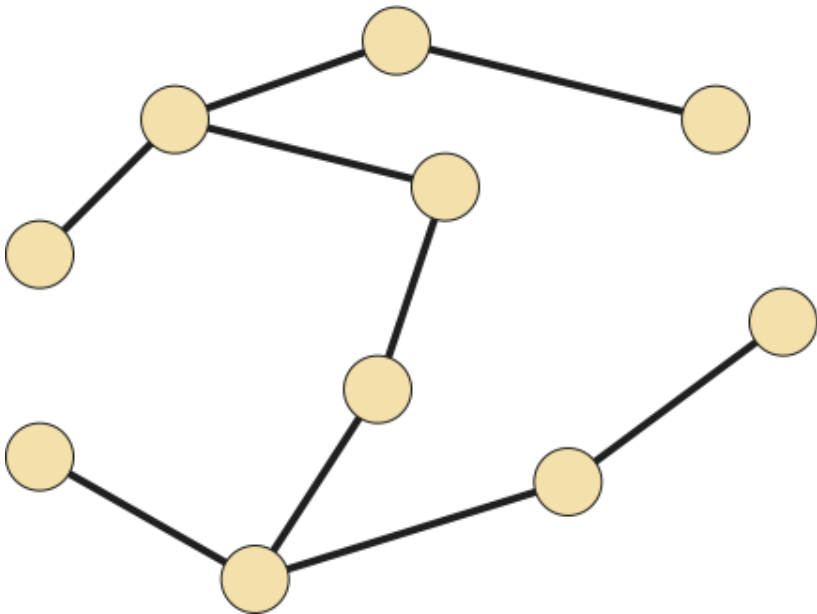


Este grafo representa um sistema produtivo através de uma linha de montagem serial, com 3 estágios e 5 períodos.

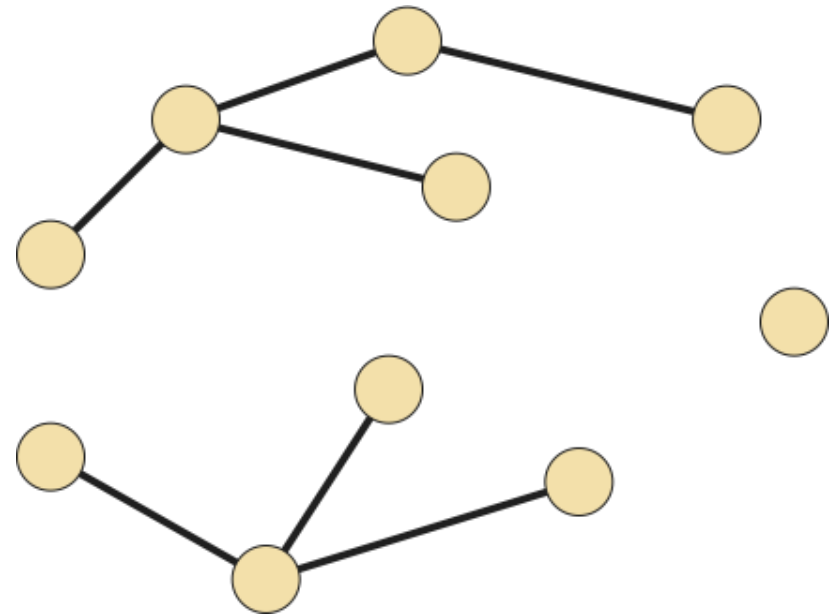
1 - Introdução aos Grafos

Árvores e Florestas:

Uma **árvore** é um grafo que é conectado e não contém ciclos.



Uma **floresta** é um grafo que não contém ciclos. Os componentes conectados de uma floresta são árvores.



1 - Introdução aos Grafos

Representação por MATRIZ de Adjacências:

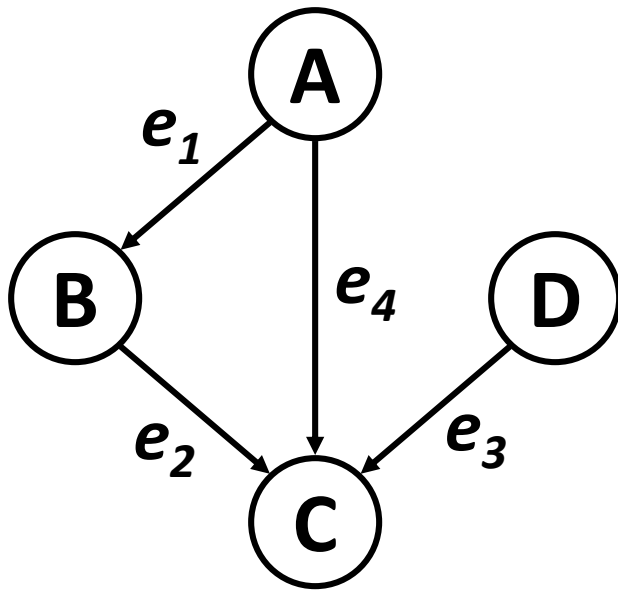
- Assuma $V = \{1, 2, \dots, n\}$
- Uma matriz de adjacências representa um grafo como uma matriz $A_{n \times n}$:

$$A[i, j] \begin{cases} = 1 & \text{se aresta } (i, j) \in E \\ = 0 & \text{se aresta } (i, j) \notin E \end{cases}$$

1 - Introdução aos Grafos

Representação por MATRIZ de Adjacências:

- Exemplo de grafo orientado:



↗	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	0
D	0	0	1	0

1 - Introdução aos Grafos

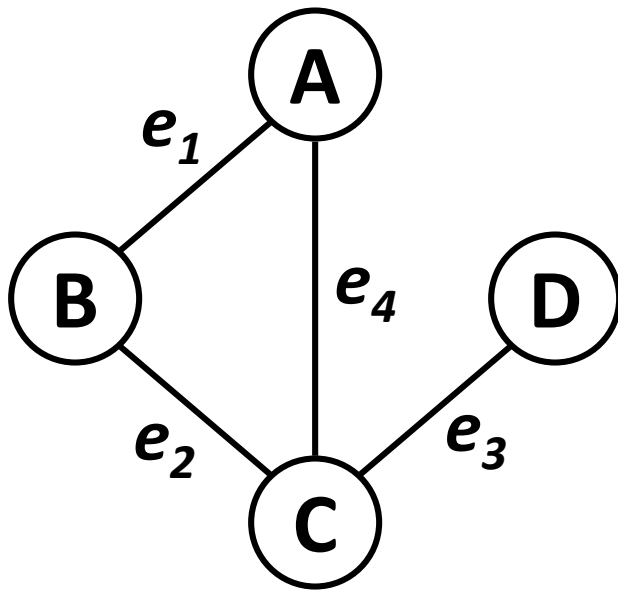
Representação por MATRIZ de Adjacências:

- **A Matriz de Adjacências requer quanto de espaço?**
 - Matriz tem tamanho $V \times V \rightarrow \Theta(V^2)$
 - A **notação assintótica** serve tanto para **tempo de execução** quanto para **espaço de armazenamento**.
 - Por exemplo, para o espaço $\Theta(g(n))$, revela o quanto a ocupação do espaço irá crescer quando ***n*** aumentar!

1 - Introdução aos Grafos

Representação por MATRIZ de Adjacências:

- Para um grafo **não-orientado**:
 - A Matriz de Adjacências é simétrica.
 - Podemos armazenar apenas metade da matriz.
- Exemplo de grafo não-orientado:



	A	B	C	D
A	0	1	1	0
B	1	0	1	0
C	1	1	0	1
D	0	0	1	0

1 - Introdução aos Grafos

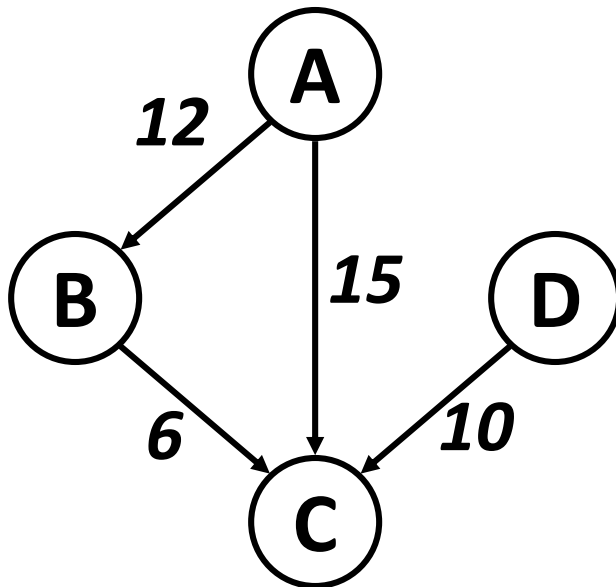
Representação por MATRIZ de Adjacências:

- **A Matriz de Adjacências:**
 - Usualmente armazena muita informação para grafos grandes.
 - Pode ser muito eficiente para grafos pequenos.
 - Geralmente usado para grafos densos.

1 - Introdução aos Grafos

Representação por MATRIZ de Adjacências:

- Para um grafo **ponderado**, ou seja com pesos em suas arestas, representa-se uma aresta por meio de seu peso:
- Exemplo de grafo orientado ponderado:



↗	A	B	C	D
A	0	12	15	0
B	0	0	6	0
C	0	0	0	0
D	0	0	10	0

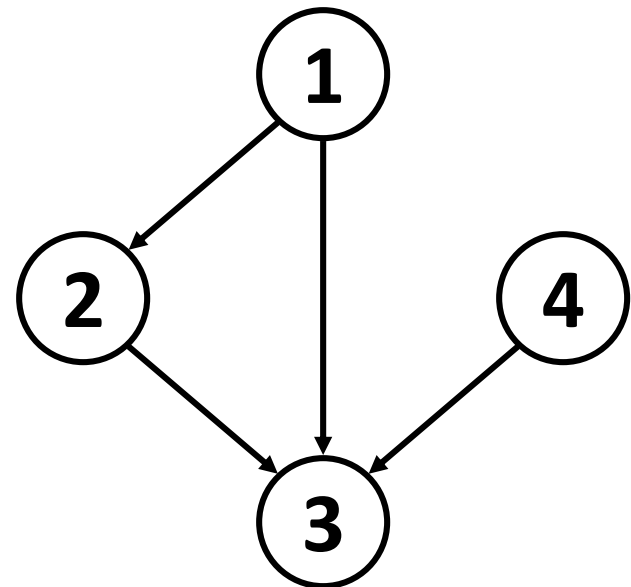
1 - Introdução aos Grafos

Representação por LISTA de Adjacências:

- Para cada vértice $v \in V$, armazena-se a lista de vértices adjacentes a v .

Exemplo:

- $Adj[1] = \{2,3\}$
- $Adj[2] = \{3\}$
- $Adj[3] = \{\}$
- $Adj[4] = \{3\}$



1 - Introdução aos Grafos

Representação por LISTA de Adjacências:

- A Lista de Adjacências requer quanto de espaço?

O grau de cada vértice v é igual a quantidade de arestas incidentes, sendo assim, é necessário $\Theta(V+E)$ de espaço para armazenamento:

- Para **grafos orientados**, o número de itens na lista de adjacências é a somatória do grau de saída $(v) = |E|$. É preciso armazenar cada vértice $v \in V$ mais a lista de arestas E , portanto, $\Theta(V + E)$.
- Para **grafos não-orientados**, o número de itens na lista de adjacências é a somatória do grau $(v) = 2|E|$, que também é $\Theta(V + E)$.

1 - Introdução aos Grafos

Características dos Algoritmos de Busca em Grafos:

- Algoritmos de Busca são técnicas de Inteligência Artificial aplicadas à problemas de **alta complexidade** teórica que não são resolvidos com técnicas de programação convencionais, principalmente as de natureza puramente numérica.
- A complexidade de um problema está diretamente relacionada ao tamanho do seu **Espaço de Busca** correspondente.

1 - Introdução aos Grafos

Objetivos dos Algoritmos de Busca em Grafos:

- Seja um grafo $G = (V, E)$, **orientado ou não**, o objetivo é explorar metodologicamente cada vértice e cada aresta do grafo, ou apenas construir uma árvore.
- Para tanto, toma-se um vértice como sendo a **raiz** e escolhe-se as arestas apropriadas para produzir a árvore. Pode-se construir uma floresta se o grafo não for conectado.

2

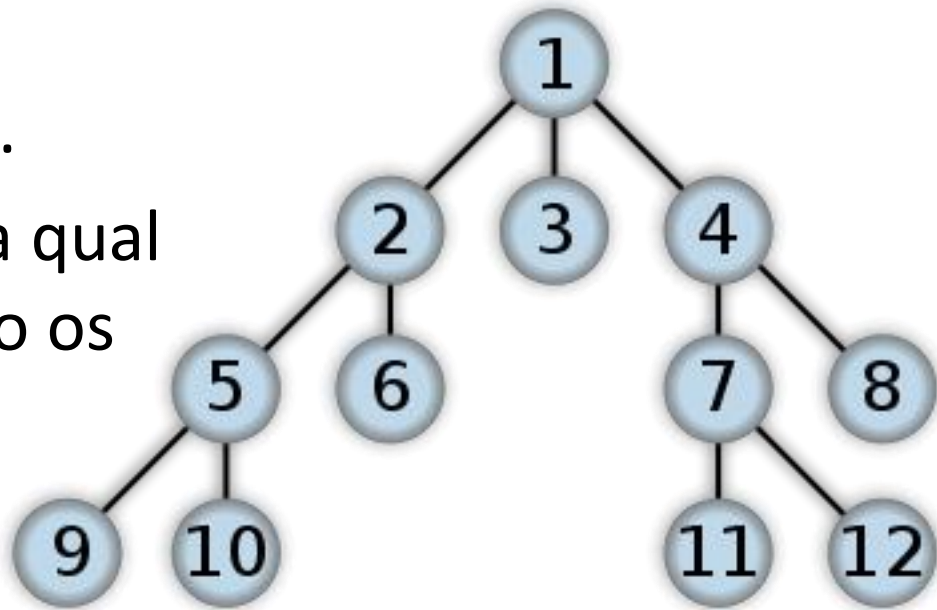
Busca em Largura ou Amplitude (Breadth-First Search)

2 - Busca em Largura (*Breadth-First Search*)

2.1 - Características:

Em uma **busca exaustiva em largura** a partir do vértice v , espera-se que todos os vizinhos de v sejam visitados antes de continuar a busca mais profundamente:

- **BFS** encontra a menor distância até um certo nó, ou seja, o número mínimo de arestas.
- **BFS** constroi uma árvore na qual os caminhos de cada nó são os menores até a raiz.



2 - Busca em Largura (*Breadth-First Search*)

2.1 - Características:

- **Explora-se** todo o grafo, tornando-o uma árvore de busca:
 - Um vértice por vez.
 - Expande em largura a fronteira de vértices explorados.
 - Busca exaustiva (todos os nós são visitados).
- **Constrói-se** a árvore de busca sobre o grafo:
 - Toma-se um vértice pai como raiz (Geração 0),
 - Descobre-se todos os filhos (Geração 1),
 - Para cada filho (Geração 1) descobre-se todos os seus filhos (Geração 2), e assim por diante.

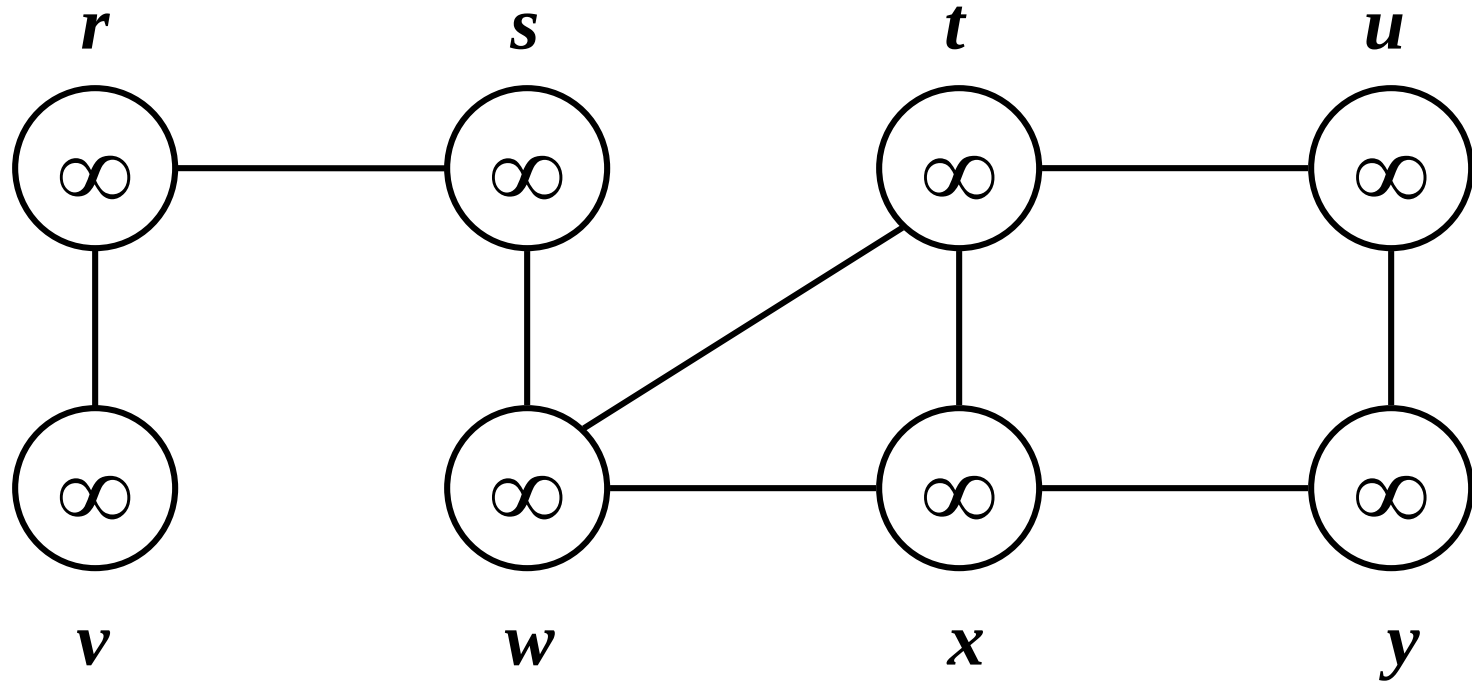
2 - Busca em Largura (Breadth-First Search)

2.2 - Metodologia BFS:

- Colorindo os vértices:
 - **Vértices brancos** ainda não foram considerados
Todos começam como branco.
 - **Vértices cinza** foram considerados mas não totalmente explorados
Eles podem ser adjacentes de vértices brancos.
 - **Vértices pretos** foram totalmente explorados
São vértices adjacentes a vértices pretos ou cinzas.
- Explorando vértices escaneando a lista de adjacências de vértices cinzas.

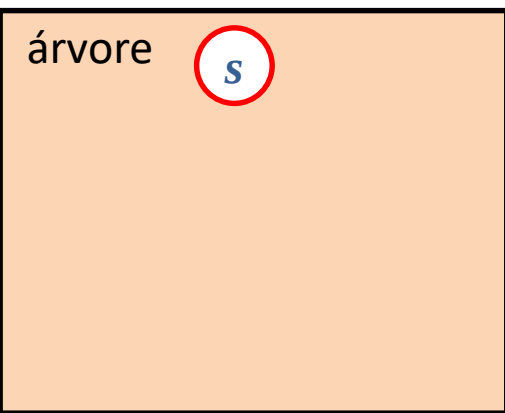
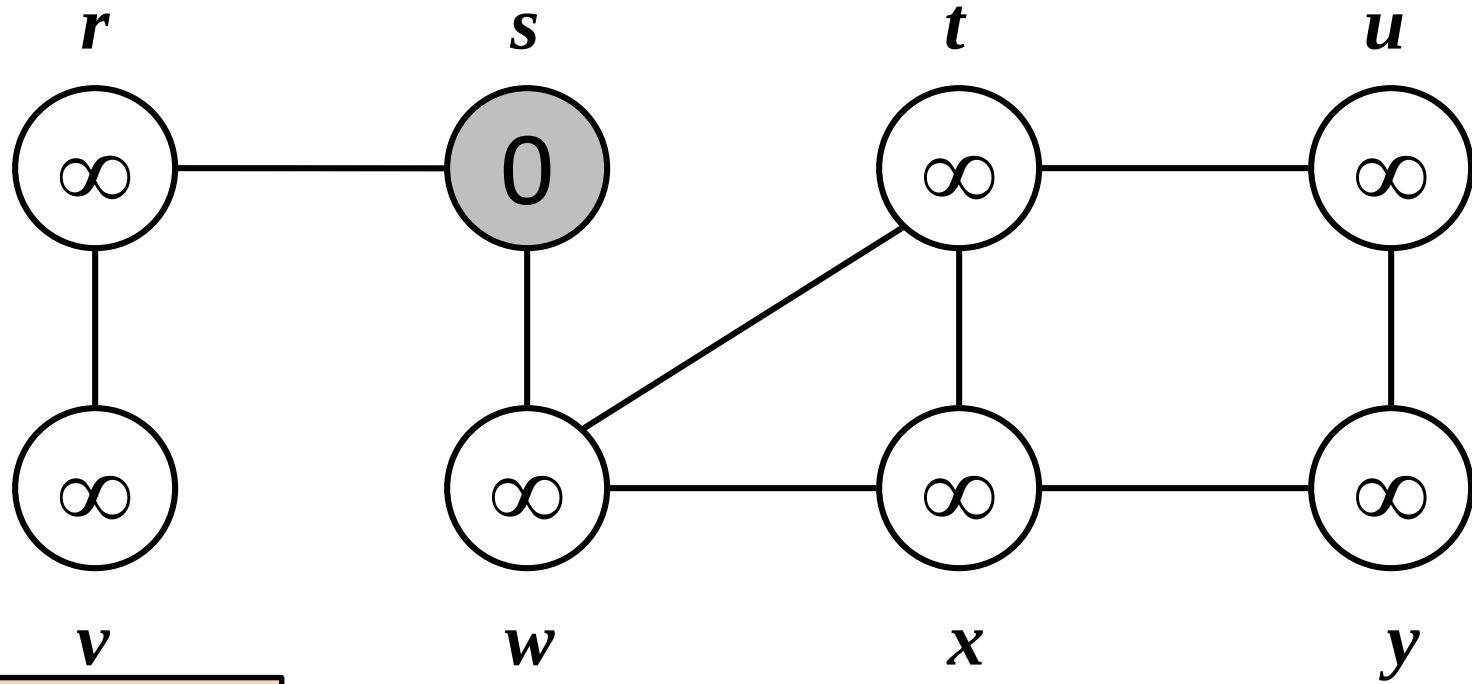
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS (iniciando no vértice s):



2 - Busca em Largura (Breadth-First Search)

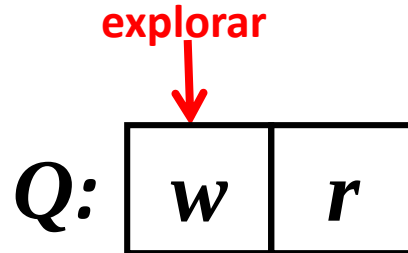
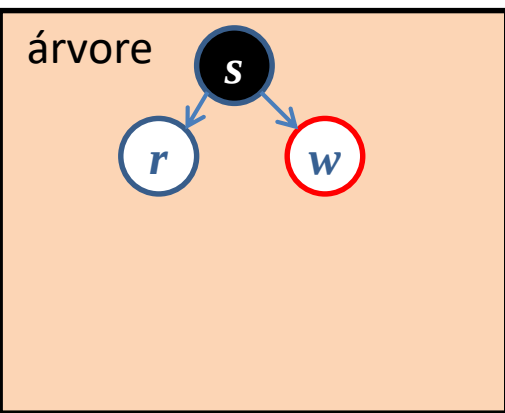
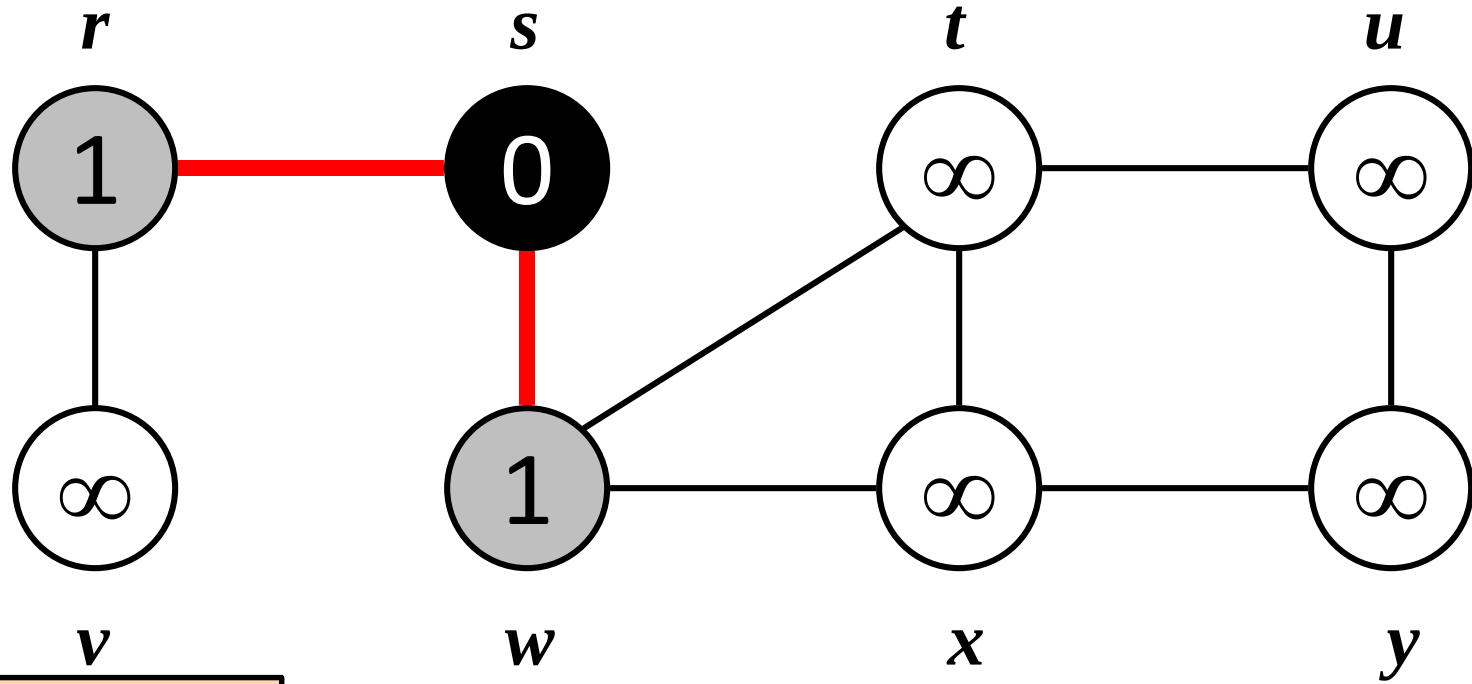
Exemplo BFS:



Q : s

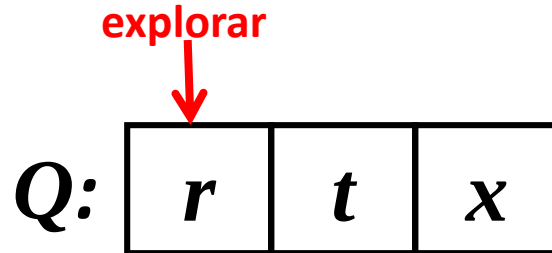
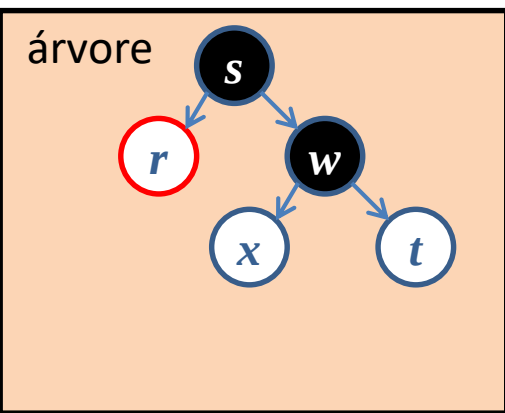
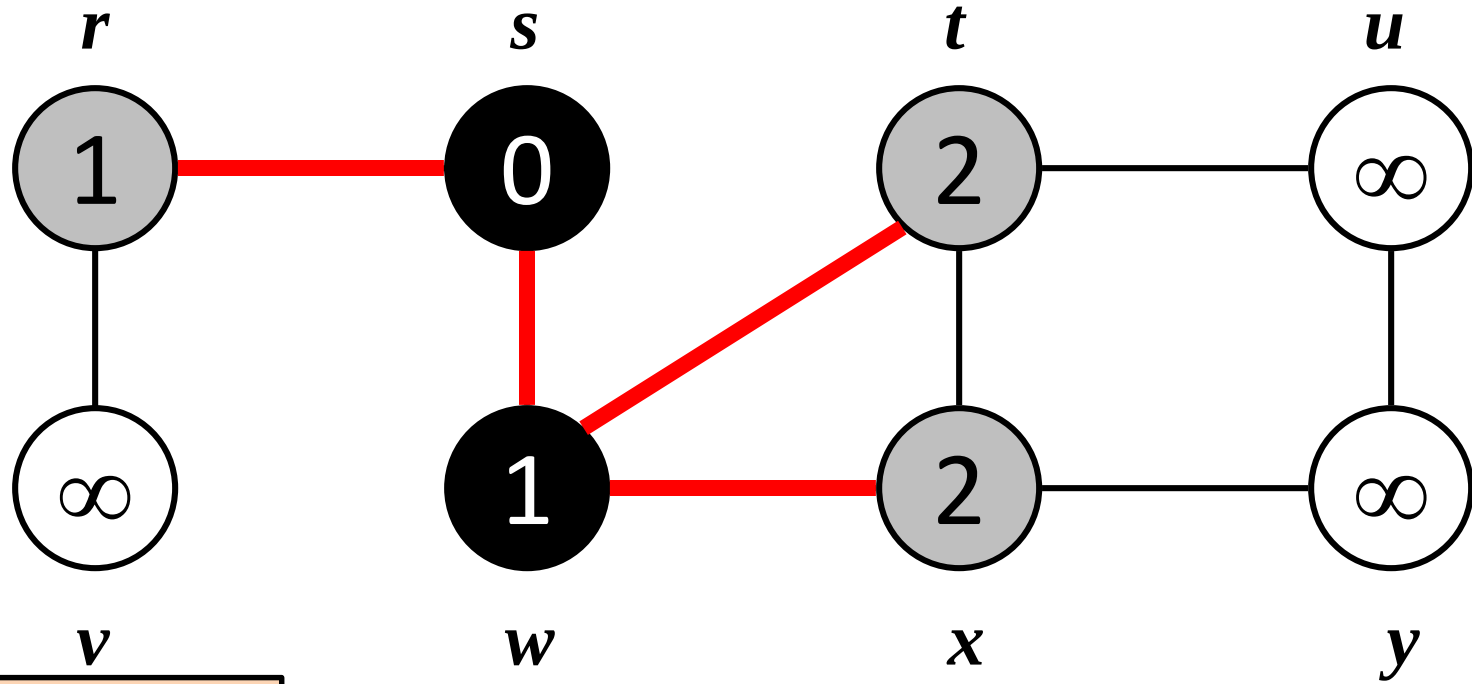
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



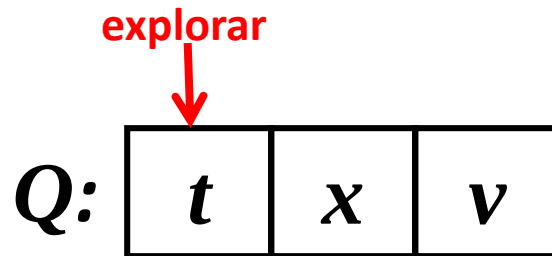
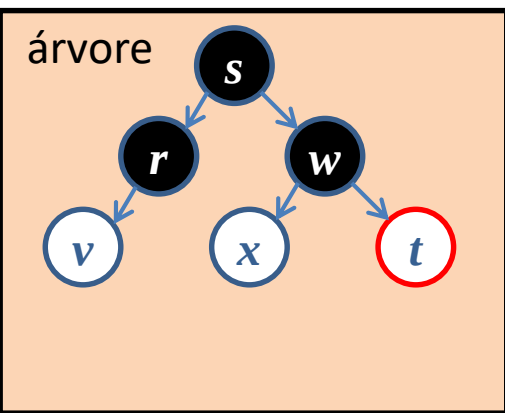
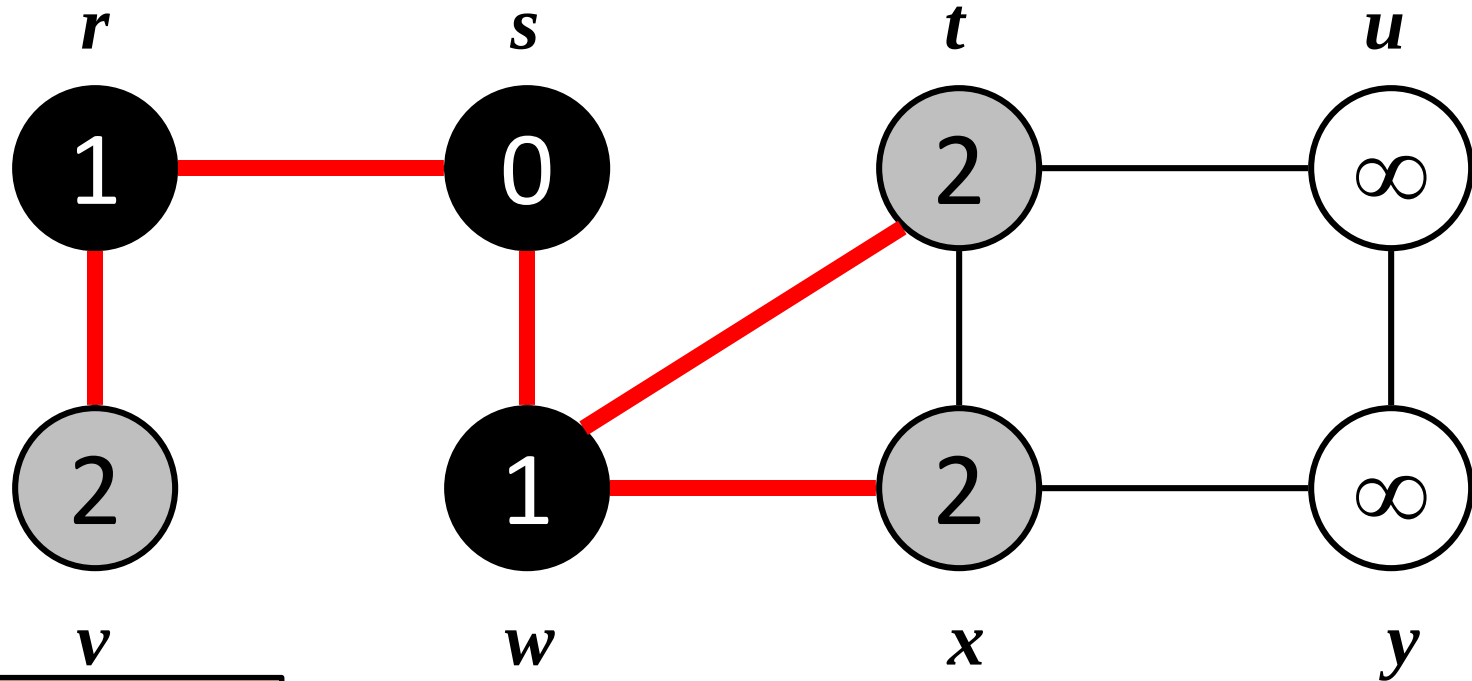
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



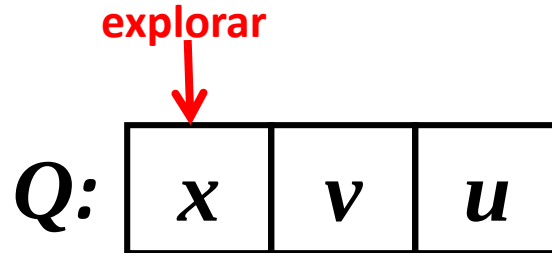
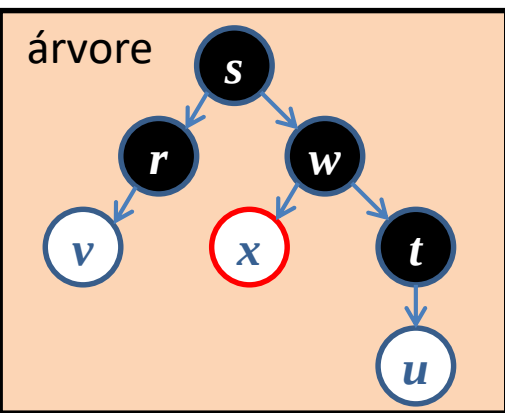
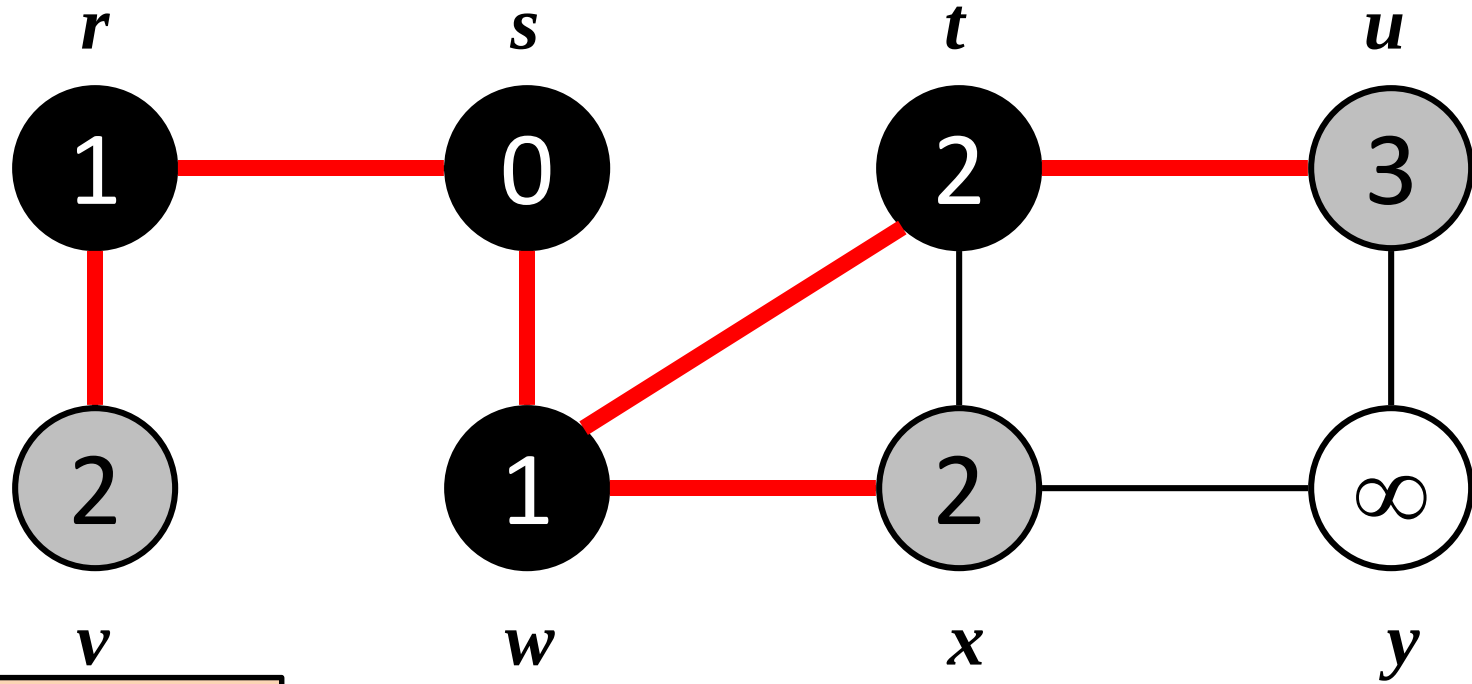
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



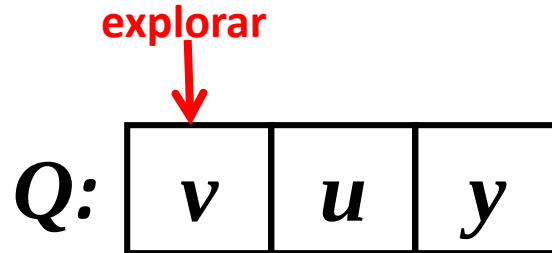
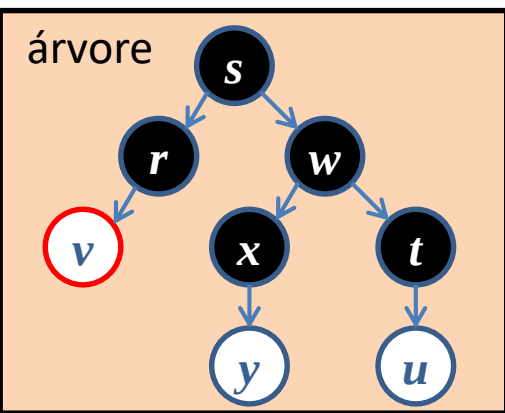
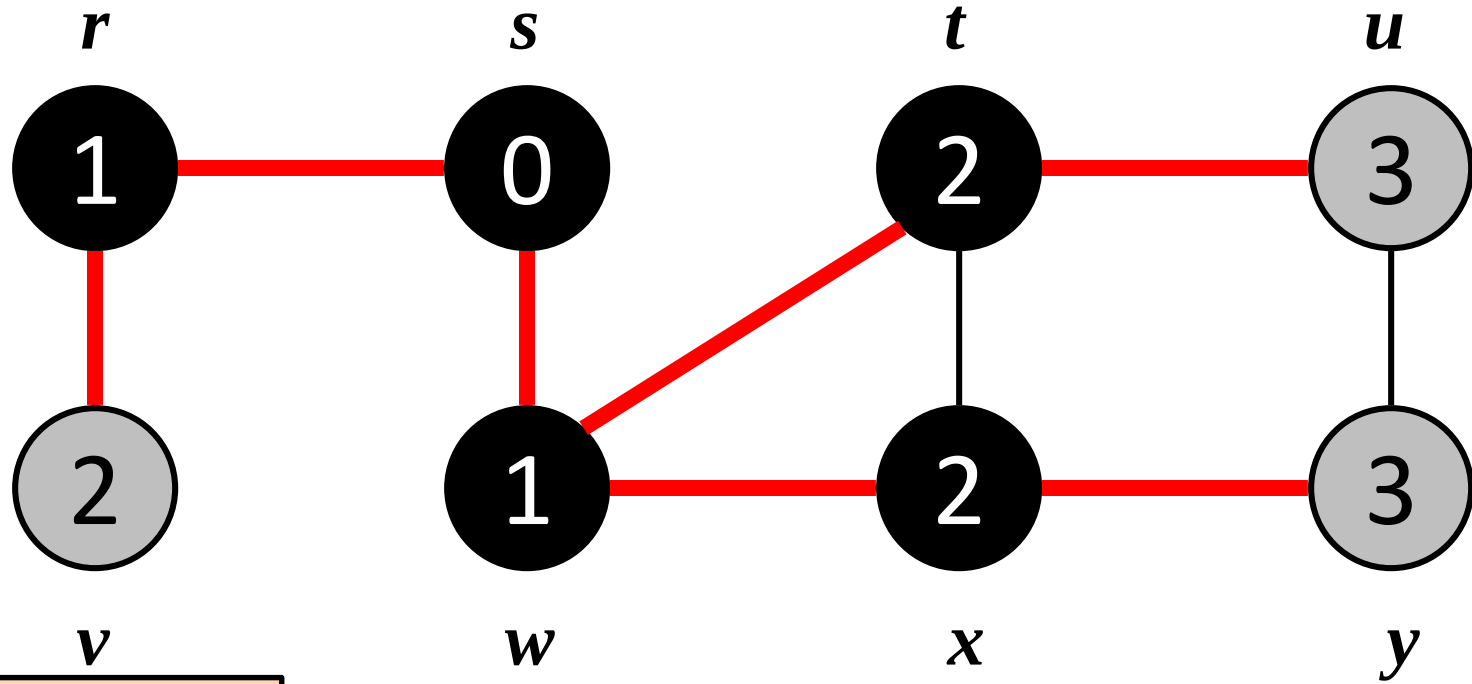
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



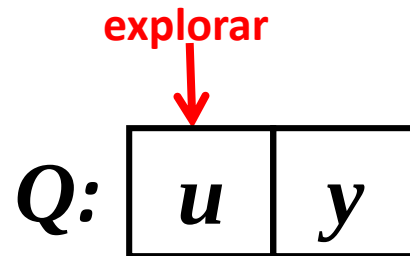
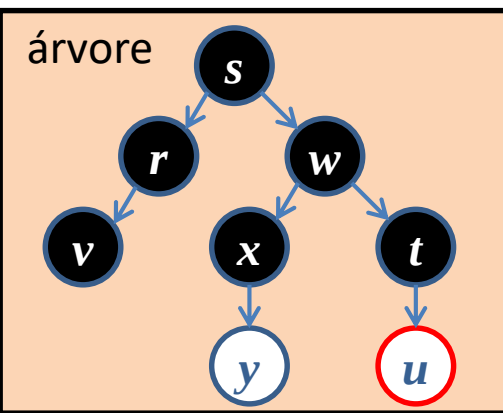
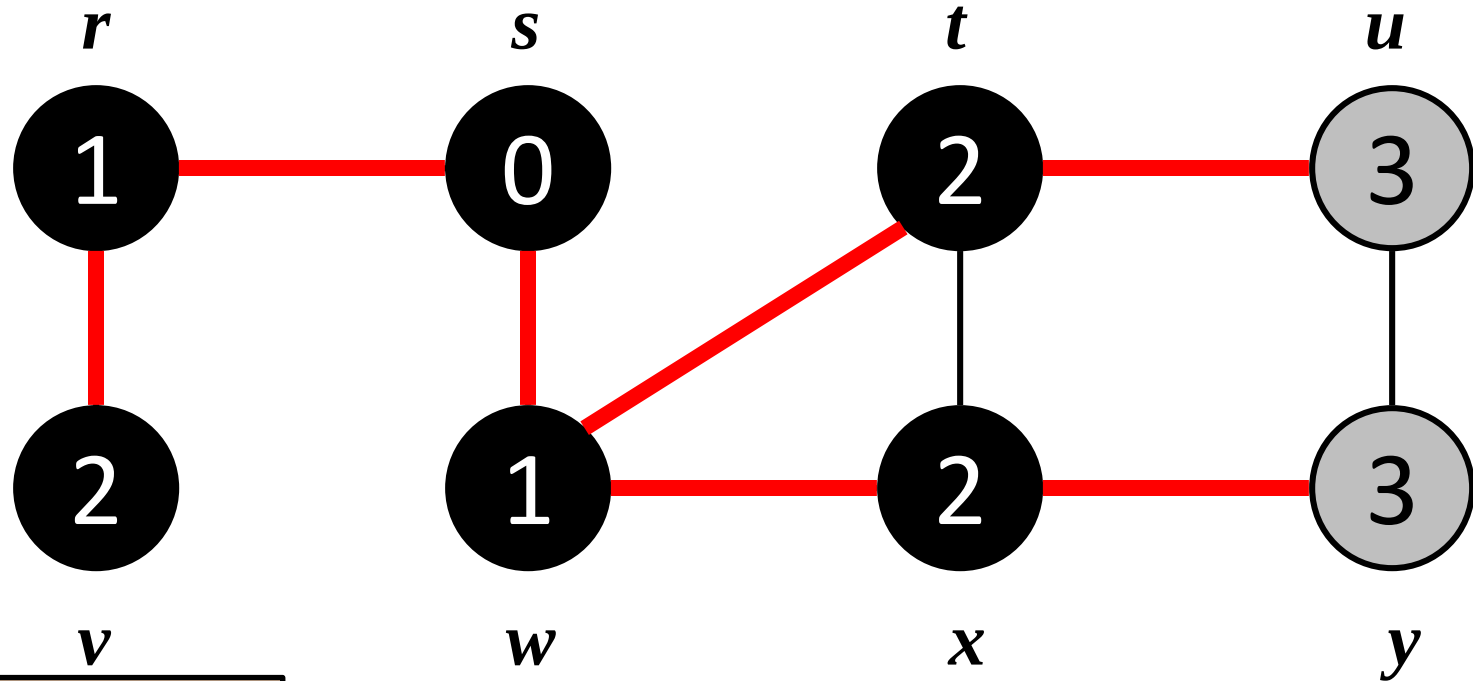
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



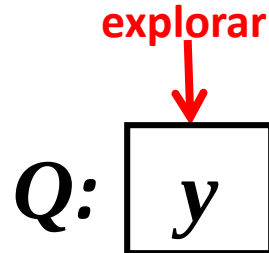
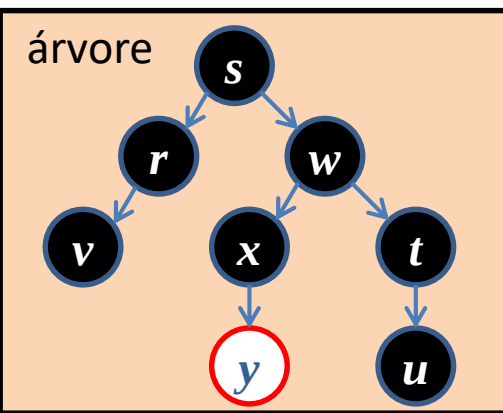
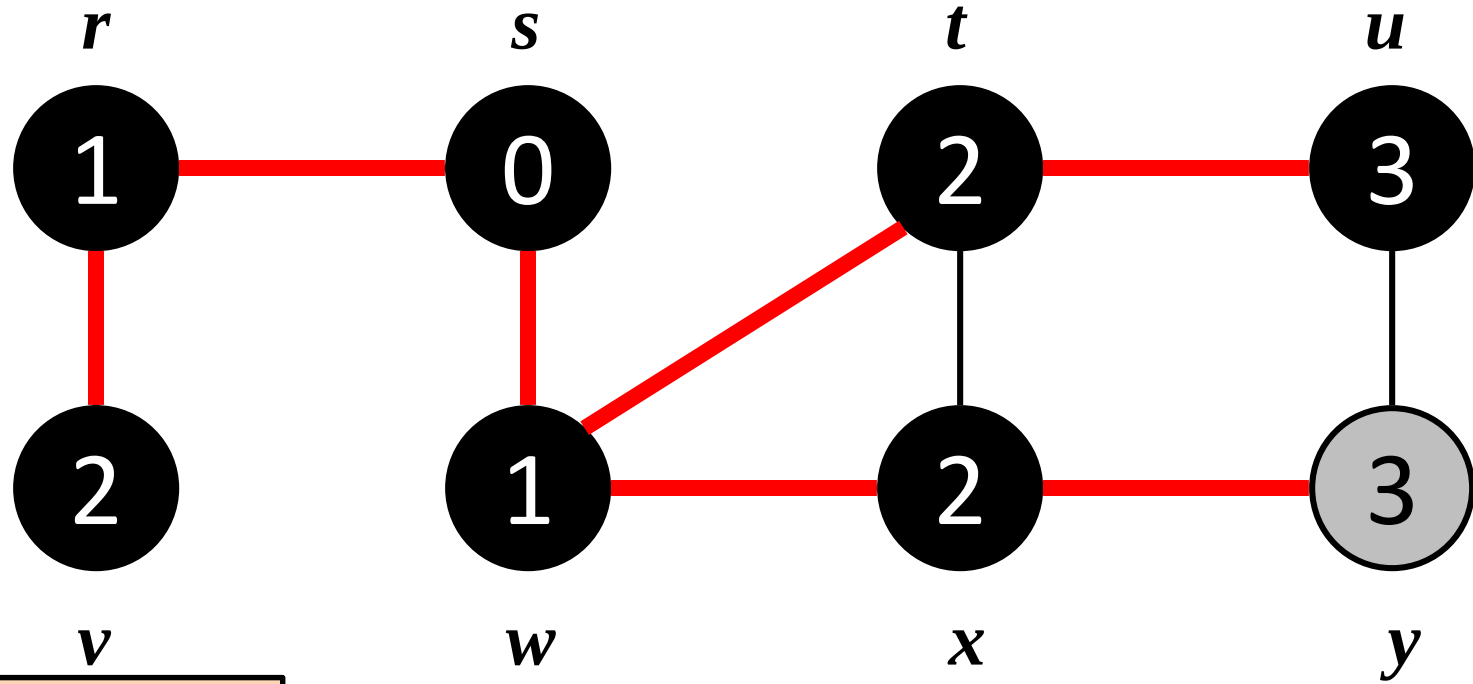
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



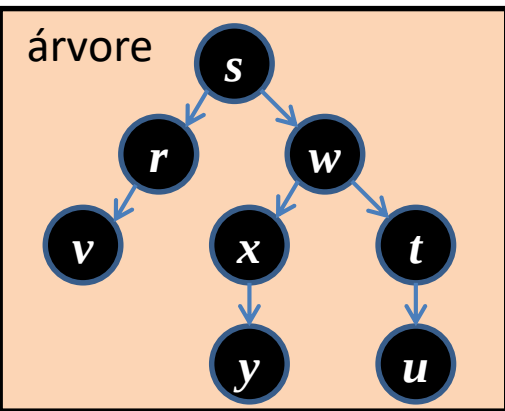
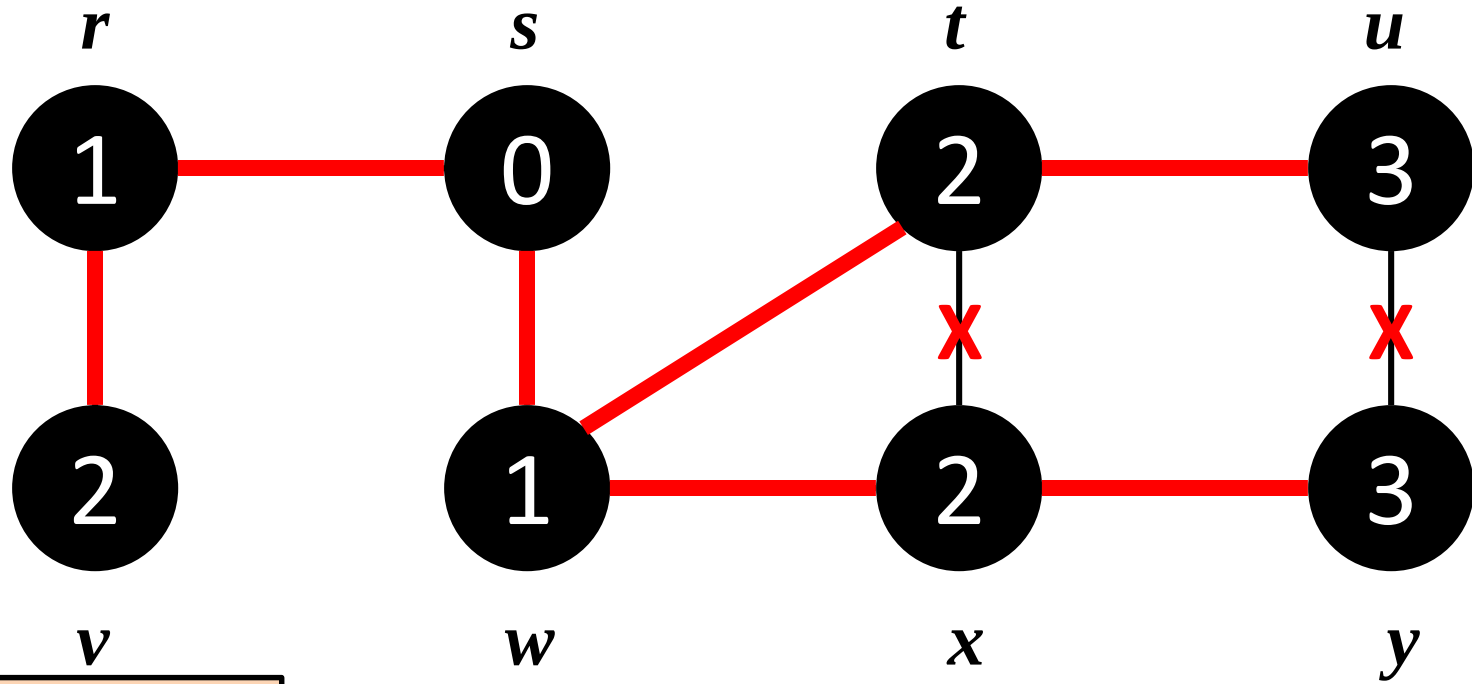
2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



2 - Busca em Largura (Breadth-First Search)

Exemplo BFS:



$Q: \boxed{\emptyset}$

2 - Busca em Largura (Breadth-First Search)

2.3 - O Método BFS:

1. Definir o vértice inicial s .
2. Explorar as arestas do grafo para descobrir todos os vértices alcançáveis a partir de s .
3. Computar a distância (menor número de arestas) de s para todos os vértices alcançáveis.
4. Produzir uma árvore de amplitude cuja raiz é s e contém todos os vértices alcançáveis.
5. Para todo vértice v alcançável a partir de s , o caminho na árvore de amplitude corresponde ao menor caminho de s para v no grafo.

2 - Busca em Largura (Breadth-First Search)

2.4 - O Algoritmo BFS:

- Assumir que o grafo $G = (V, E)$ é representado com lista de adjacências.
- Para cada vértice no grafo, o algoritmo mantém estruturas auxiliares:
 - A variável $cor[v']$ mantém a informação sobre a cor de cada vértice.
 - A variável $pai[v']$ mantém a informação do predecessor de cada vértice. Quando não existe predecessor $pai[v'] = NIL$.
 - A variável $d[v']$ mantém o valor da distância entre o vértice inicial e v' .

2 - Busca em Largura (Breadth-First Search)

2.4 - O Algoritmo BFS:

- Assumir que o grafo $G = (V, E)$ é representado com lista de adjacências.
- Para cada vértice no grafo, o algoritmo mantém estruturas auxiliares:
 - Uma fila Q com política FIFO para gerenciar a lista de vértices de cor cinza.
 - O vértice inicial é s .
 - O vértice observado é v' .

2 - Busca em Largura (*Breadth-First Search*)

2.4 - O Algoritmo BFS:

BFS(G, s)

for $\forall v' \in V[G] - \{s\}$ do

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$d[v'] \leftarrow \infty$

$\text{pai}[v'] \leftarrow \text{NIL}$

$\text{cor}[s] \leftarrow \text{CINZA}$

$d[s] \leftarrow 0$

$\text{pai}[s] \leftarrow \text{NIL}$

$Q \leftarrow \{s\}$

2 - Busca em Largura (*Breadth-First Search*)

2.4 - O Algoritmo BFS:

BFS(G, s)

for $\forall v' \in V[G] - \{s\}$ do

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$d[v'] \leftarrow \infty$

$\text{pai}[v'] \leftarrow \text{NIL}$

$\text{cor}[s] \leftarrow \text{CINZA}$

$d[s] \leftarrow 0$

$\text{pai}[s] \leftarrow \text{NIL}$

$Q \leftarrow \{s\}$

Inicia variáveis auxiliares para cada um dos vértices, com exceção da origem.

2 - Busca em Largura (Breadth-First Search)

2.4 - O Algoritmo BFS:

BFS(G, s)

for $\forall v' \in V[G] - \{s\}$ do

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$d[v'] \leftarrow \infty$

$\text{pai}[v'] \leftarrow \text{NIL}$

$\text{cor}[s] \leftarrow \text{CINZA}$

$d[s] \leftarrow 0$

$\text{pai}[s] \leftarrow \text{NIL}$

$Q \leftarrow \{s\}$

**Inicia variáveis auxiliares da origem s
e a fila Q .**

2 - Busca em Largura (*Breadth-First Search*)

2.4 - O Algoritmo BFS:

```
while  $Q \neq \emptyset$  do  
     $v' \leftarrow \text{Desenfileira}[Q]$   
    for  $\forall v \in \text{Adjacente}[v']$  do  
        if  $\text{cor}[v] = \text{BRANCO}$  then  
             $\text{cor}[v] \leftarrow \text{CINZA}$   
             $d[v] \leftarrow d[v'] + 1$   
             $\text{pai}[v] \leftarrow v'$   
             $\text{Enfileira}(Q, v)$   
  
     $\text{cor}[v'] \leftarrow \text{PRETO}$ 
```

Será executado
enquanto a fila não
estiver vazia.

2 - Busca em Largura (*Breadth-First Search*)

2.4 - O Algoritmo BFS:

while $Q \neq \emptyset$ **do**

$v' \leftarrow \text{Desenfileira}[Q]$

Explora o 1º vértice
da fila: v'

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{cor}[v] \leftarrow \text{CINZA}$

$d[v] \leftarrow d[v'] + 1$

$\text{pai}[v] \leftarrow v'$

$\text{Enfileira}(Q, v)$

$\text{cor}[v'] \leftarrow \text{PRETO}$

2 - Busca em Largura (Breadth-First Search)

2.4 - O Algoritmo BFS:

while $Q \neq \emptyset$ **do**

$v' \leftarrow \text{Desenfileira}[Q]$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{cor}[v] \leftarrow \text{CINZA}$

$d[v] \leftarrow d[v'] + 1$

$\text{pai}[v] \leftarrow v'$

$\text{Enfileira}(Q, v)$

$\text{cor}[v'] \leftarrow \text{PRETO}$

Para todos os filhos
de v' (vértices
adjacentes), faça:

2 - Busca em Largura (*Breadth-First Search*)

2.4 - O Algoritmo BFS:

while $Q \neq \emptyset$ **do**

$v' \leftarrow \text{Desenfileira}[Q]$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{cor}[v] \leftarrow \text{CINZA}$

$d[v] \leftarrow d[v'] + 1$

$\text{pai}[v] \leftarrow v'$

$\text{Enfileira}(Q, v)$

$\text{cor}[v'] \leftarrow \text{PRETO}$

Se o vértice adjacente é branco, significa que ele nunca foi visitado. Deve ser pintado de CINZA e enfileirado para posterior processamento.

2 - Busca em Largura (*Breadth-First Search*)

2.4 - O Algoritmo BFS:

while $Q \neq \emptyset$ **do**

$v' \leftarrow \text{Desenfileira}[Q]$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{cor}[v] \leftarrow \text{CINZA}$

$d[v] \leftarrow d[v'] + 1$

$\text{pai}[v] \leftarrow v'$

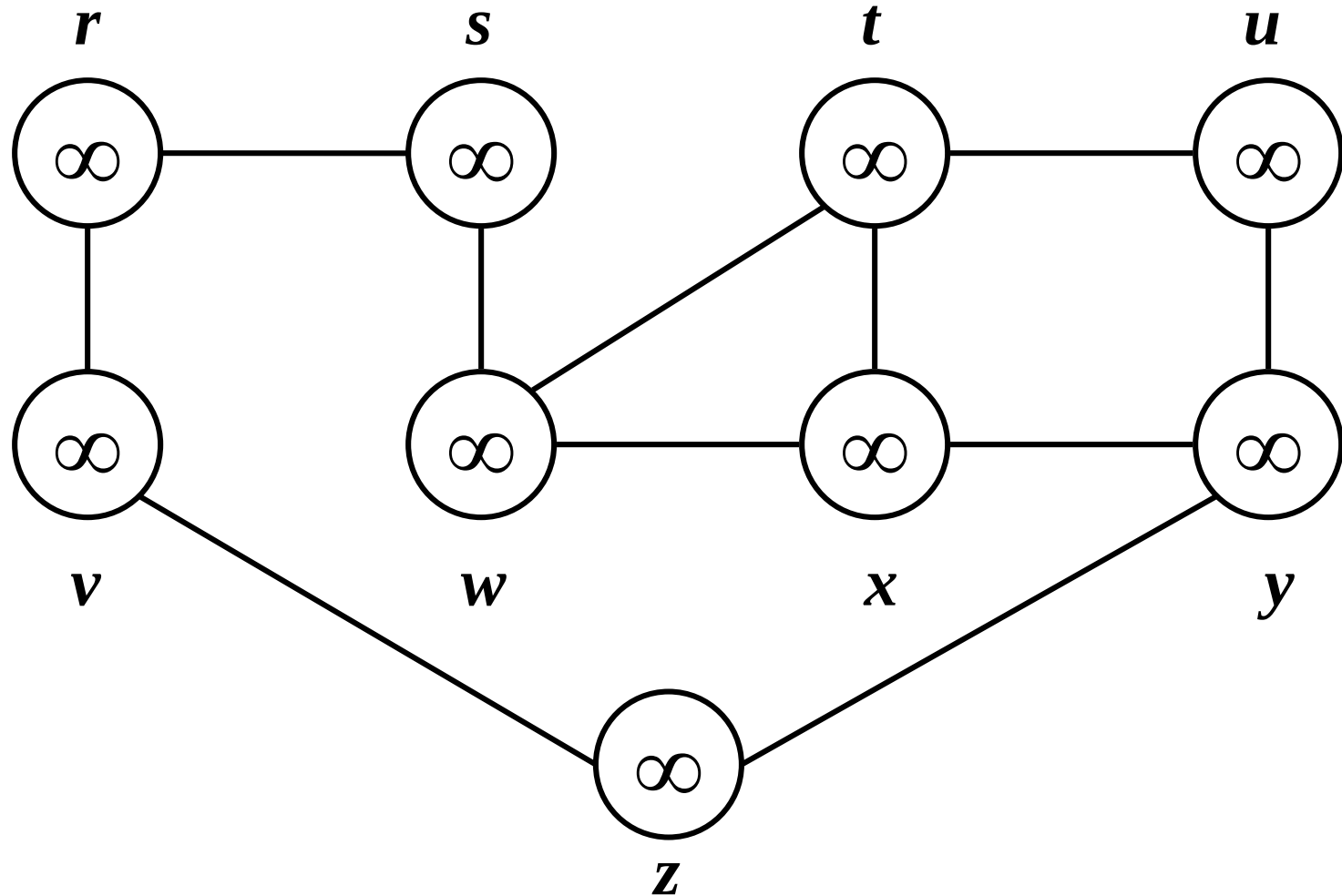
$\text{Enfileira}(Q, v)$

$\text{cor}[v'] \leftarrow \text{PRETO}$

Quando todos os adjacentes de v' forem processados, ele passa a ser PRETO.

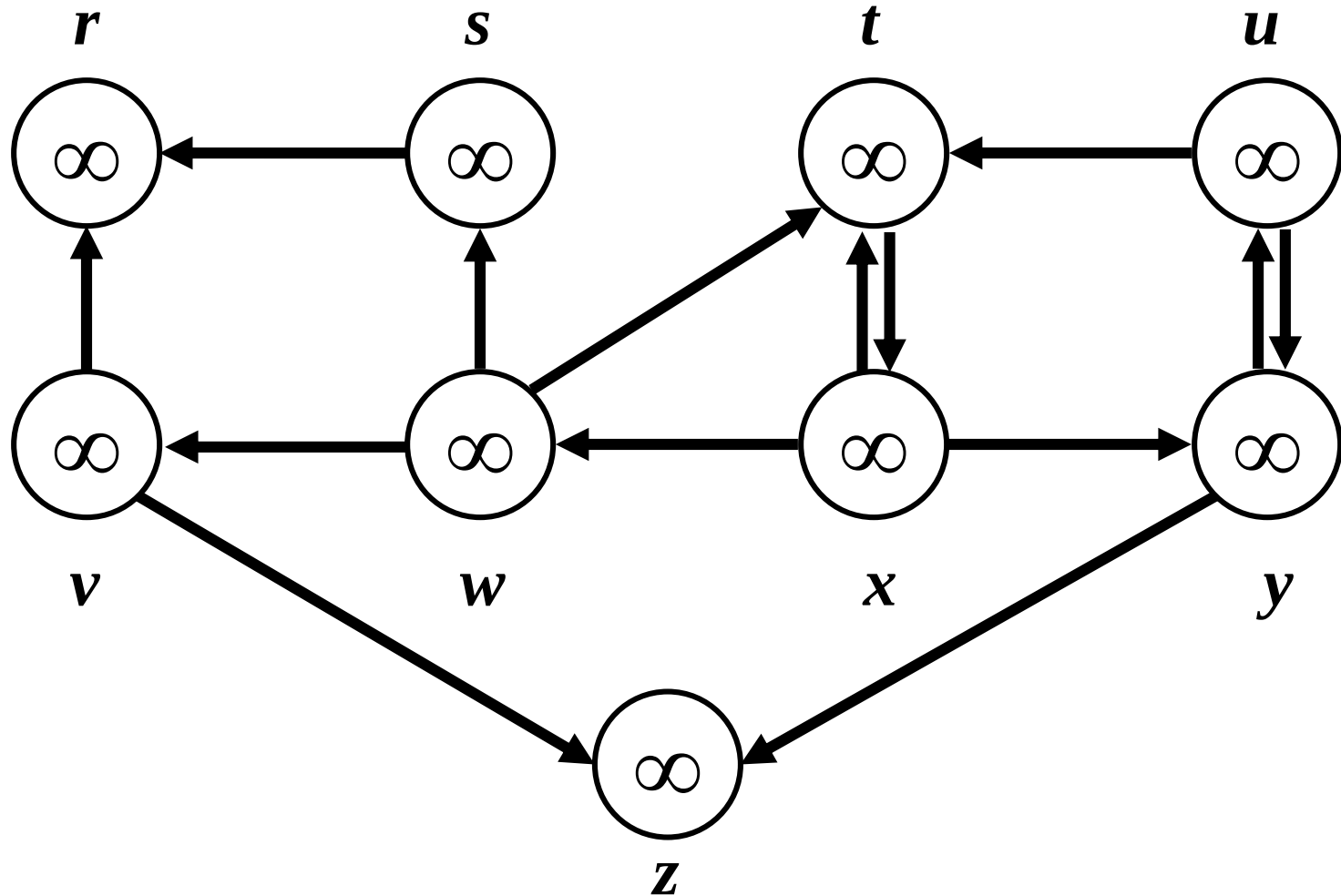
2 - Busca em Largura (Breadth-First Search)

2.5 – Exercícios de BFS: 1) iniciar em x .



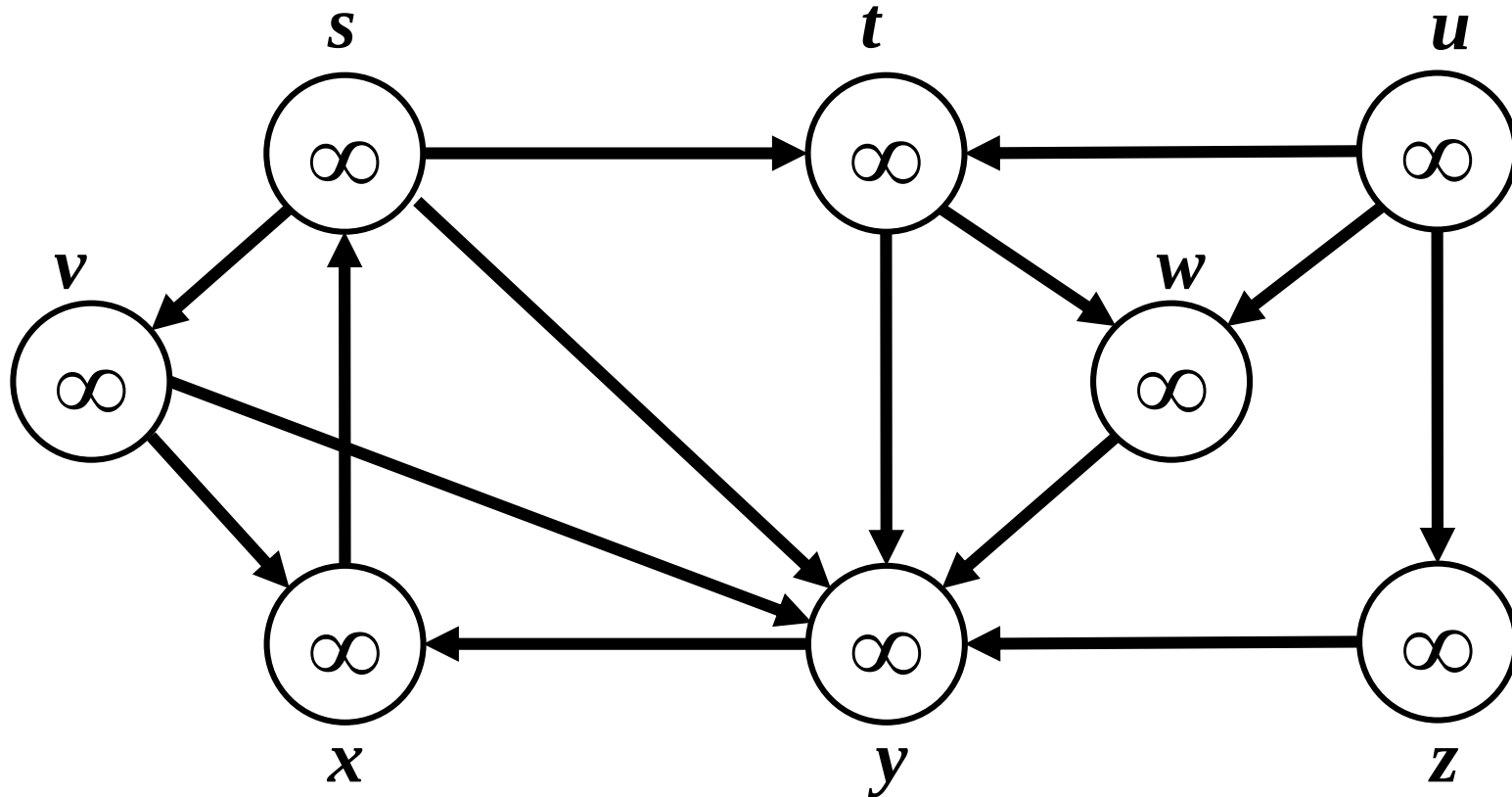
2 - Busca em Largura (Breadth-First Search)

2.5 – Exercícios de BFS: 2) iniciar em x .



2 - Busca em Largura (Breadth-First Search)

2.5 – Exercícios de BFS: 3) iniciar em x .



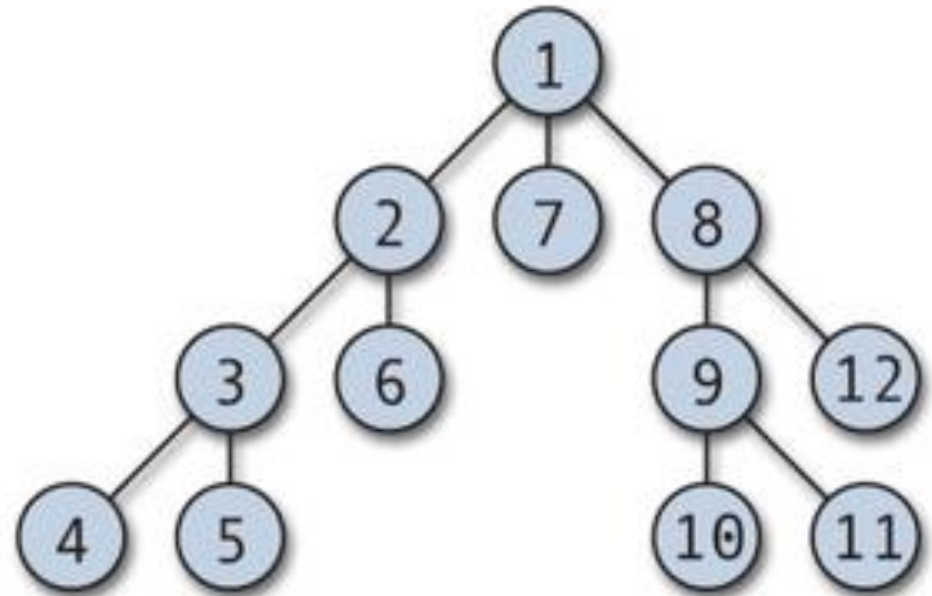
3

Busca em Profundidade (*Depth-First Search*)

3 - Busca em Profundidade (*Depth-First Search*)

3.1 - Características DFS:

Em uma **busca exaustiva em profundidade** a partir de um vértice v , espera-se que todos os filhos, e os filhos dos filhos de v sejam visitados antes de continuar a busca no vizinho:



3 - Busca em Profundidade (Depth-First Search)

3.1 - Características DFS:

- **Explora-se** todo o grafo, tornando-o árvore de busca:
 - Um vértice por vez, expandindo em profundidade a fronteira de vértices explorados.
 - Busca exaustiva (todos os nós são visitados).
- **Constrói-se** a árvore de busca sobre o grafo:
 - Toma-se um vértice pai como raiz (Geração 0),
 - Descobre-se apenas 1 filho (Geração 1),
 - Para este filho (Geração 1) descobre-se apenas 1 filho (Geração 2), e assim sucessivamente, até não haver mais Gerações.
 - Quando todas as arestas de um vértice já foram exploradas retorna (*backtracking*) à geração anterior e descobre-se outro filho, e assim por diante.

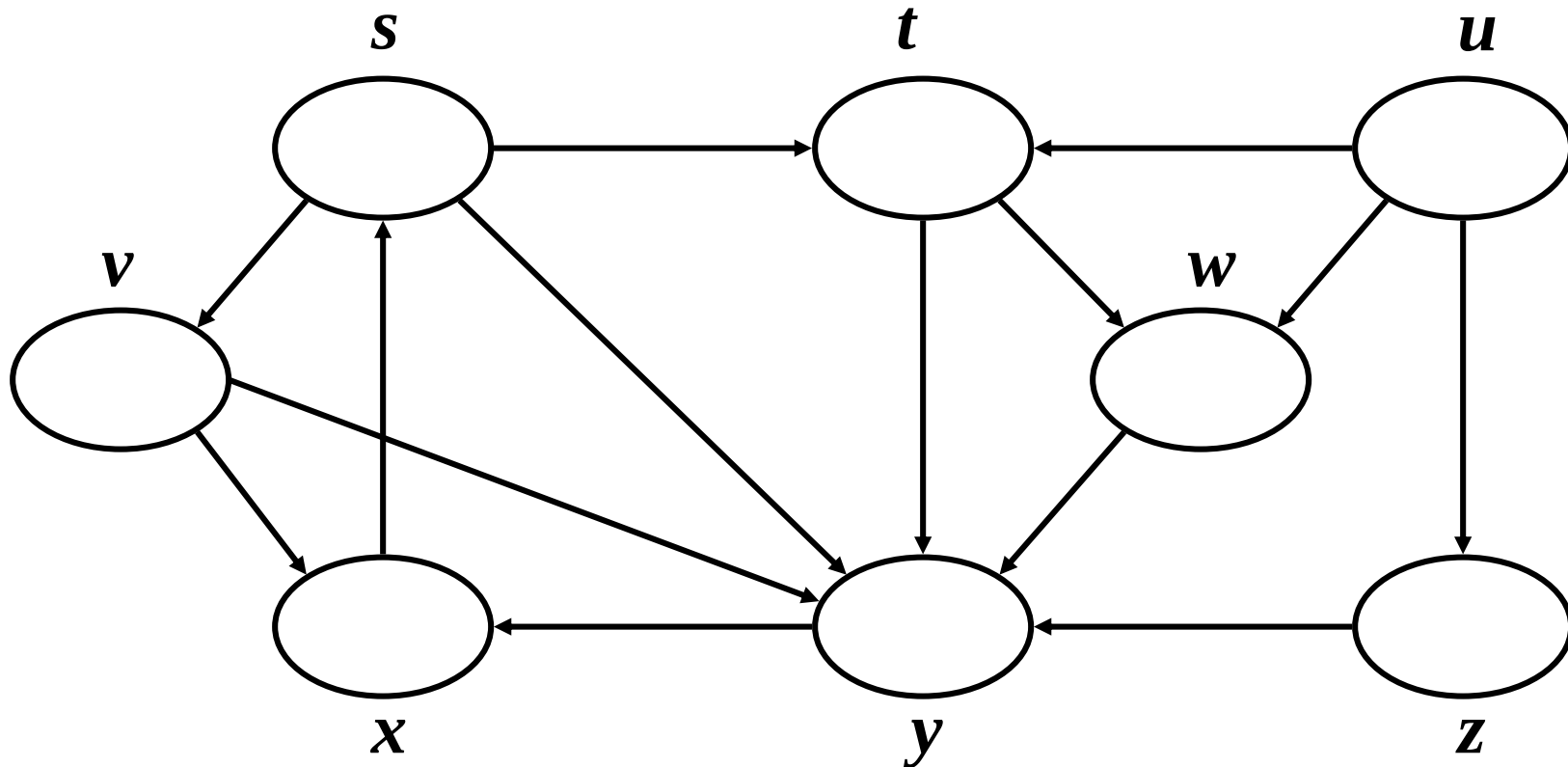
3 - Busca em Profundidade (Depth-First Search)

3.2 - Metodologia DFS:

- Colorindo os vértices:
 - **Vértices brancos** ainda não foram considerados
Todos começam como branco.
 - **Vértices cinza** foram considerados mas não totalmente explorados
Eles podem ser adjacentes de vértices brancos.
 - **Vértices pretos** foram totalmente explorados
São vértices adjacentes a vértices pretos ou cinzas.
- Explorando vértices escaneando a lista de adjacências de vértices cinzas.

3 - Busca em Profundidade (*Depth-First Search*)

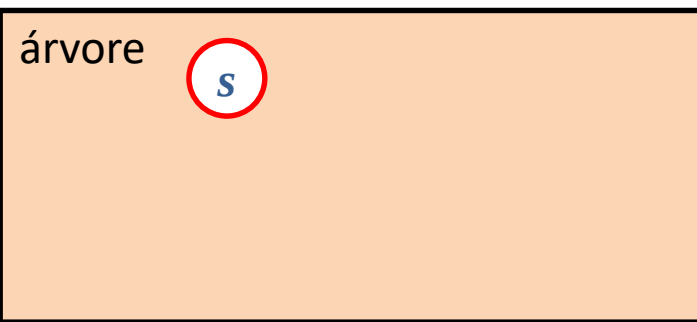
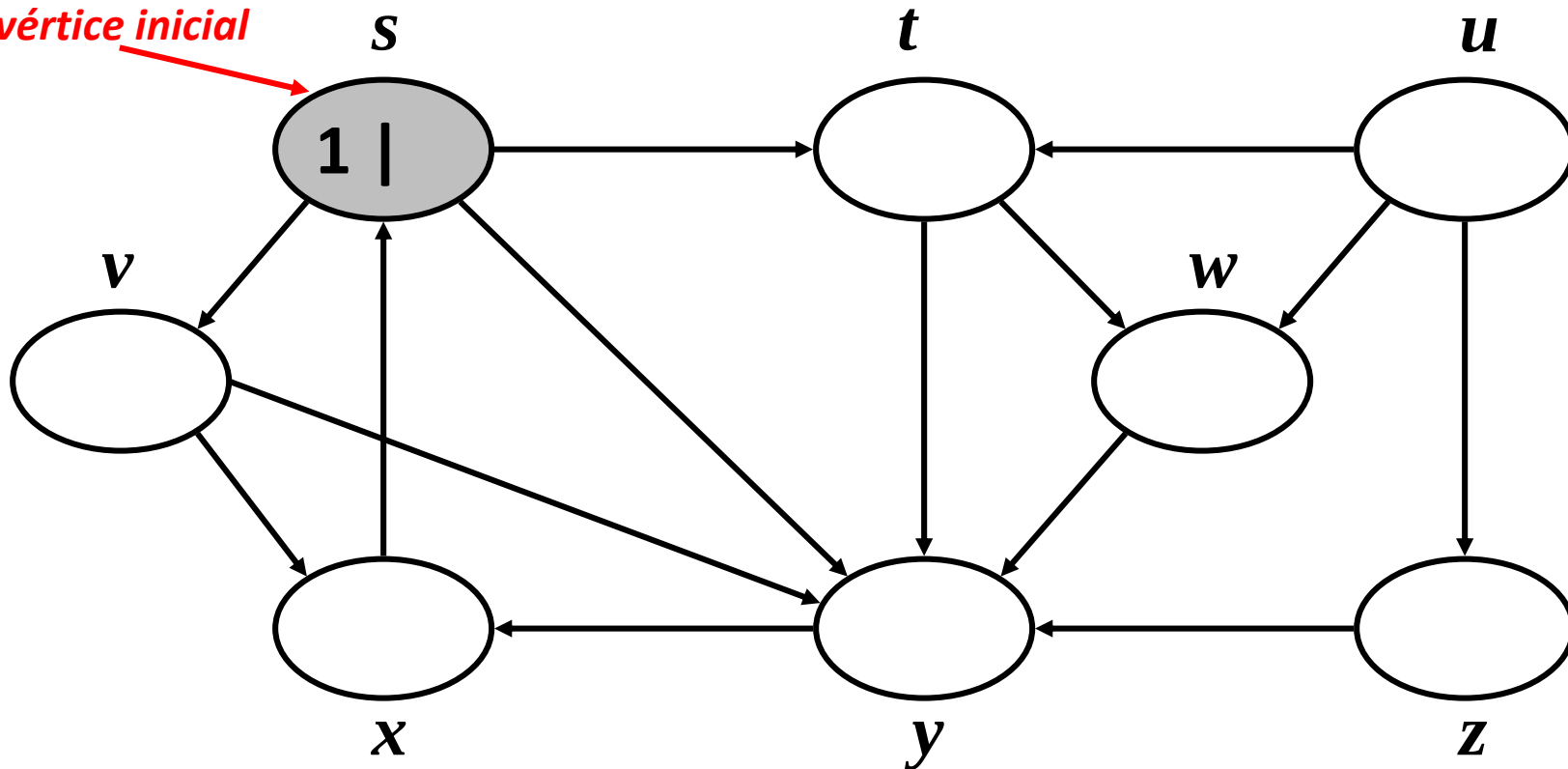
Exemplo DFS (iniciando no vértice *s*):



3 - Busca em Profundidade (*Depth-First Search*)

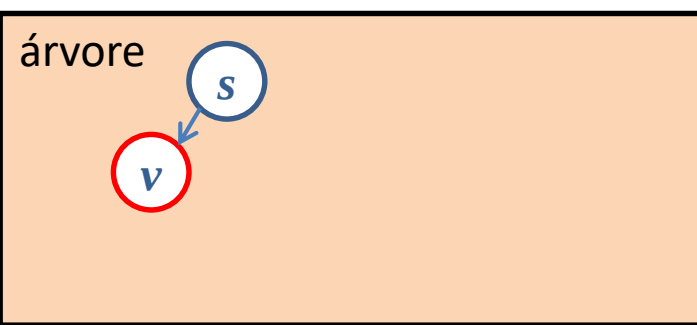
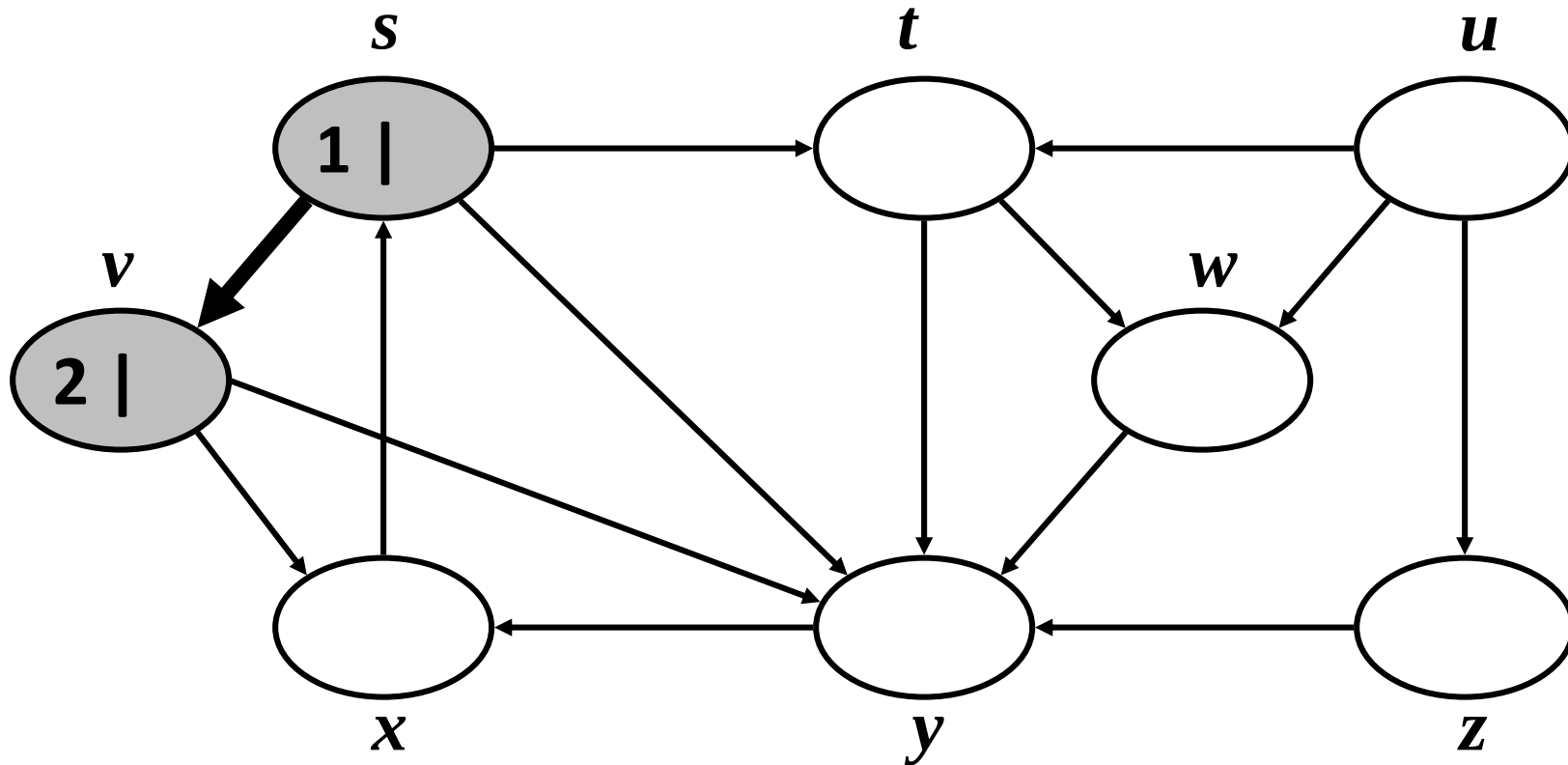
Exemplo DFS:

vértice inicial



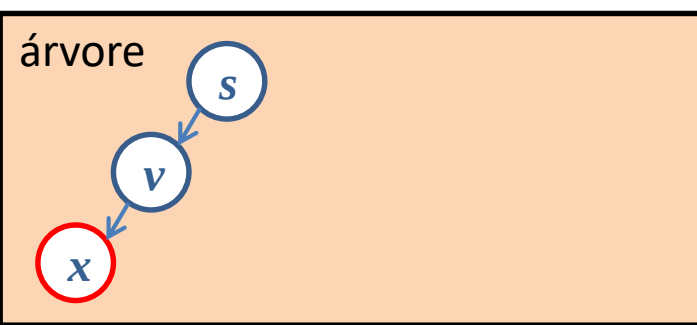
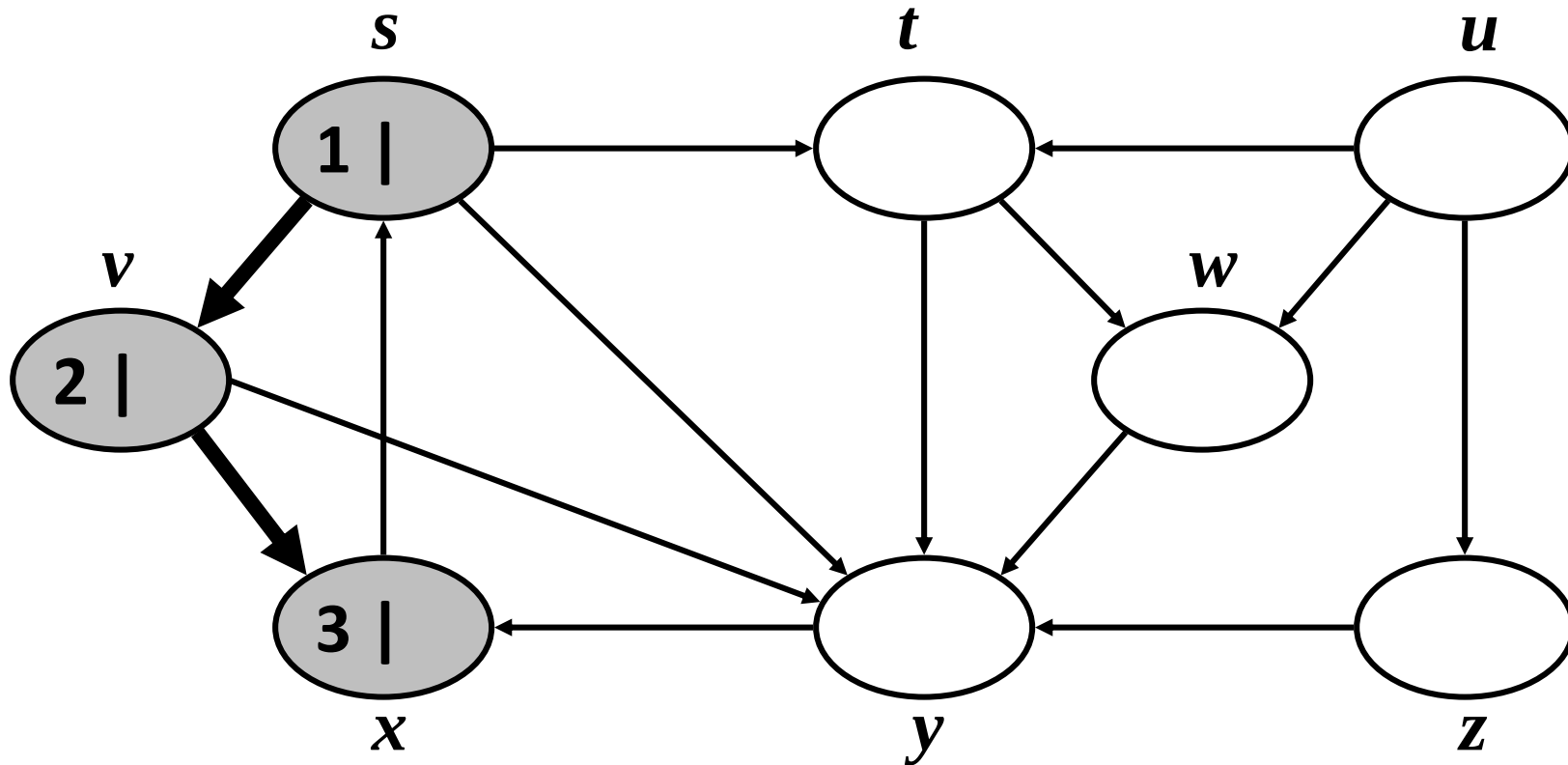
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



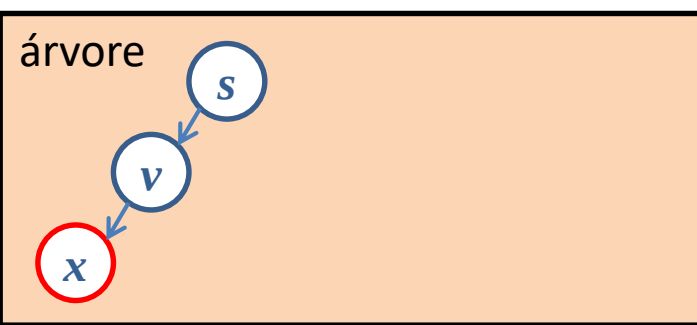
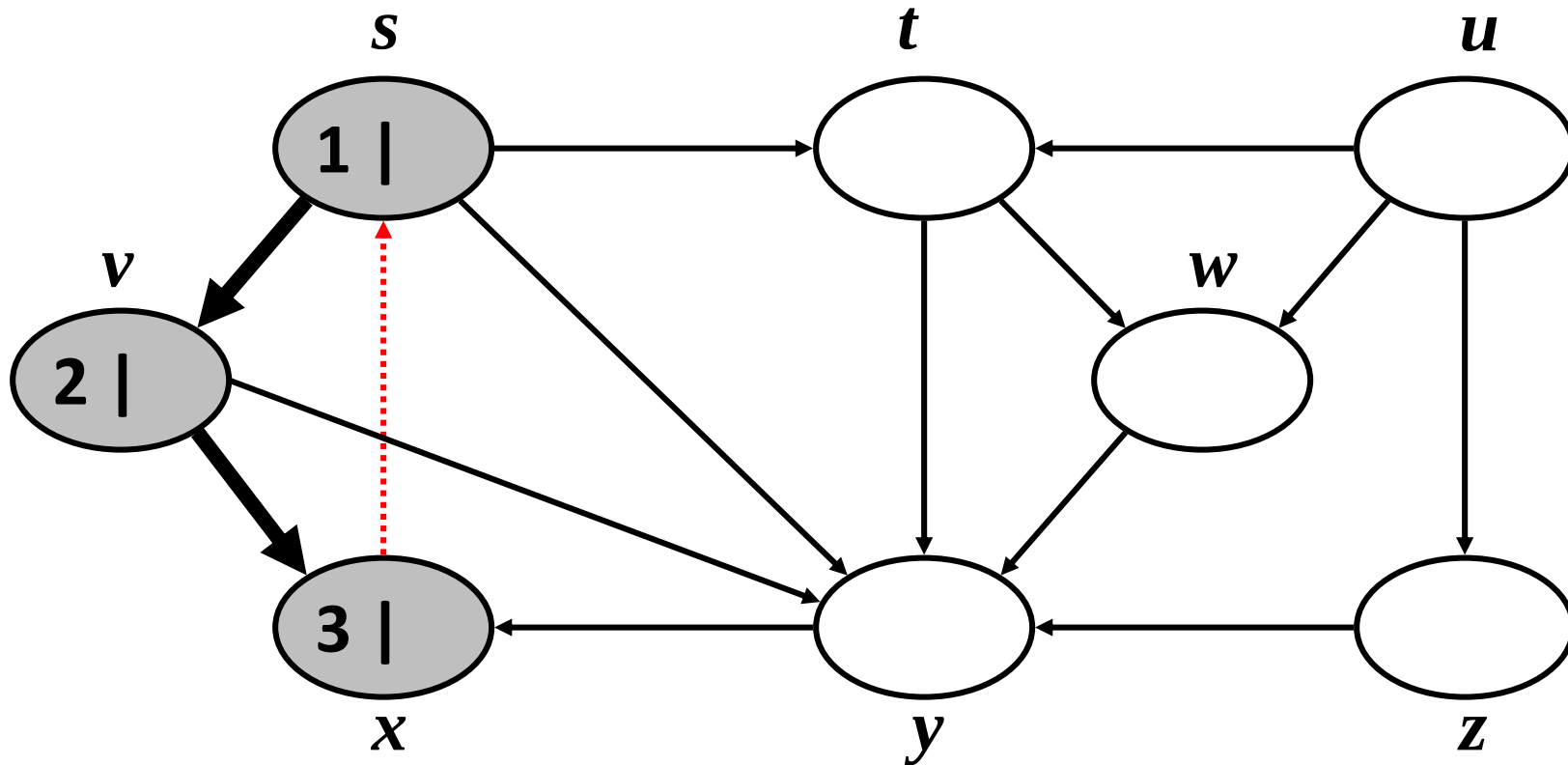
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



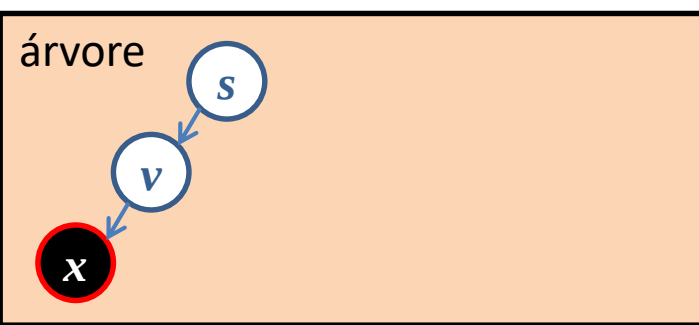
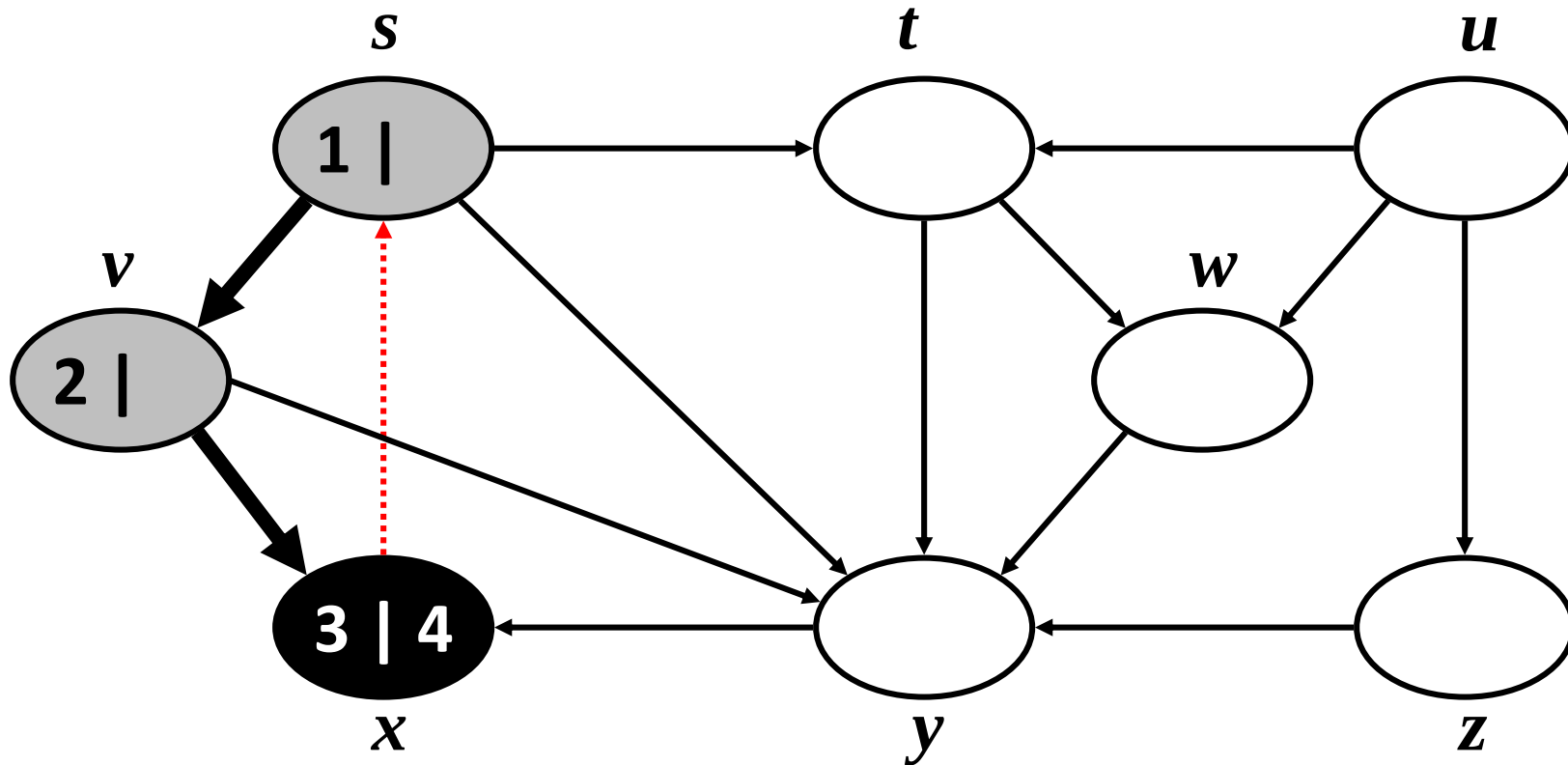
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



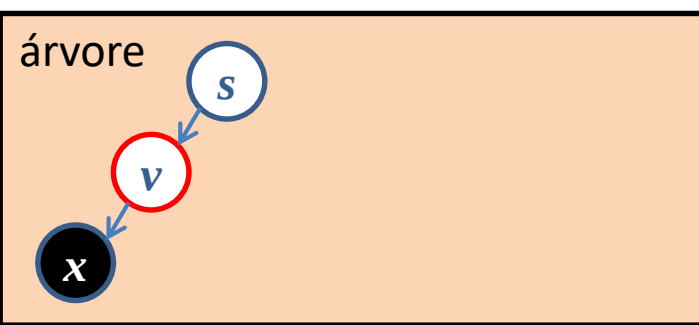
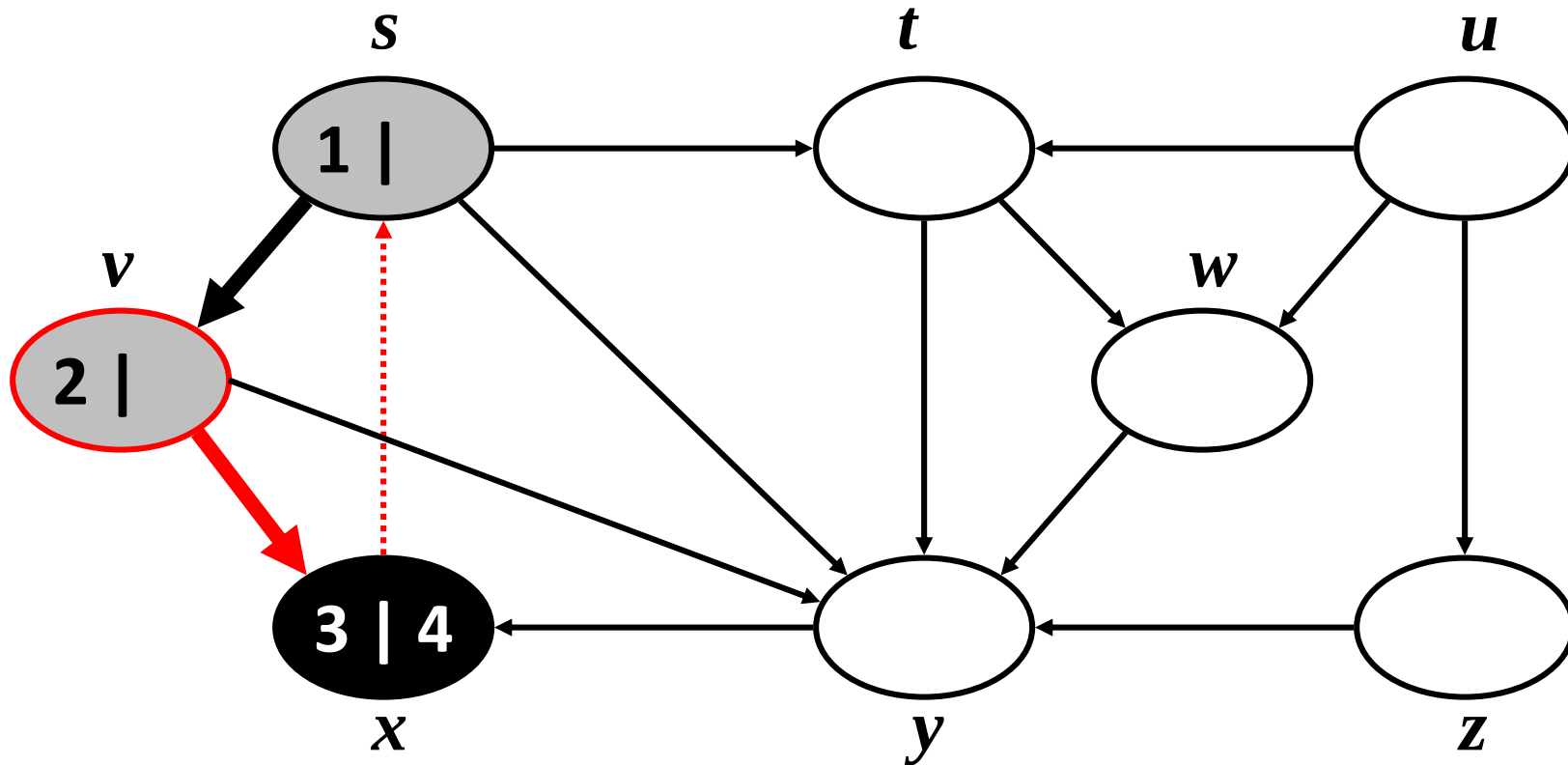
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



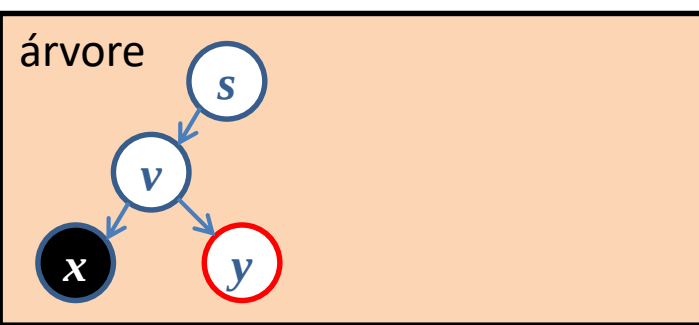
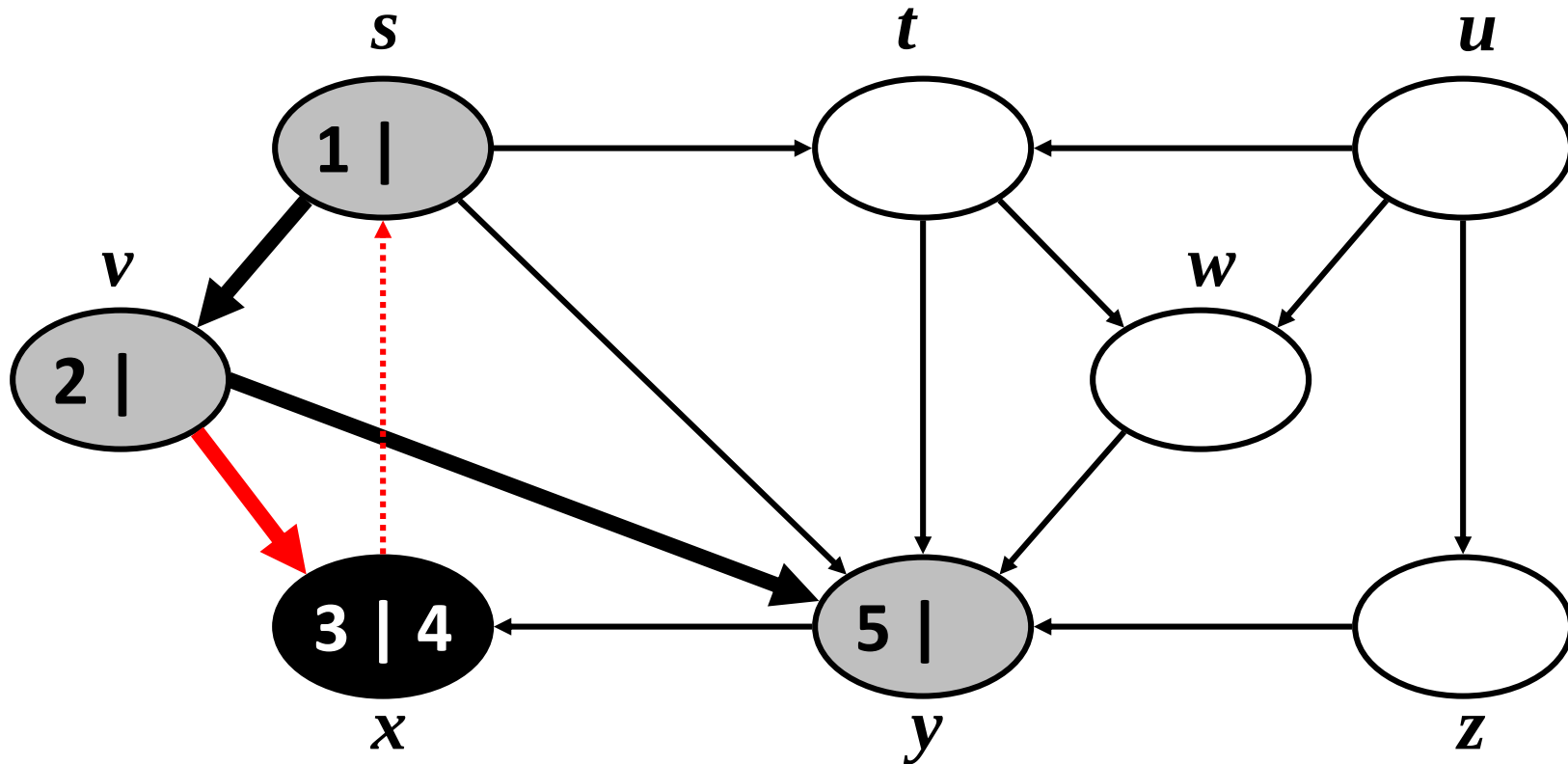
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



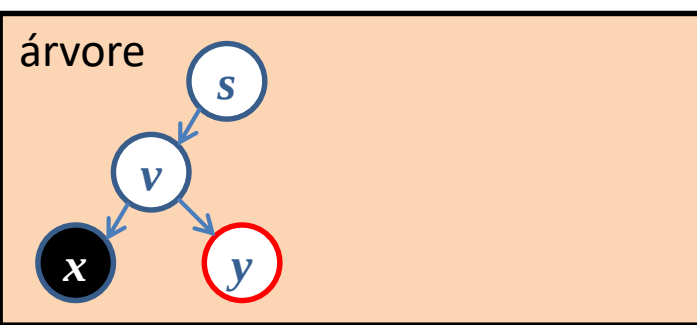
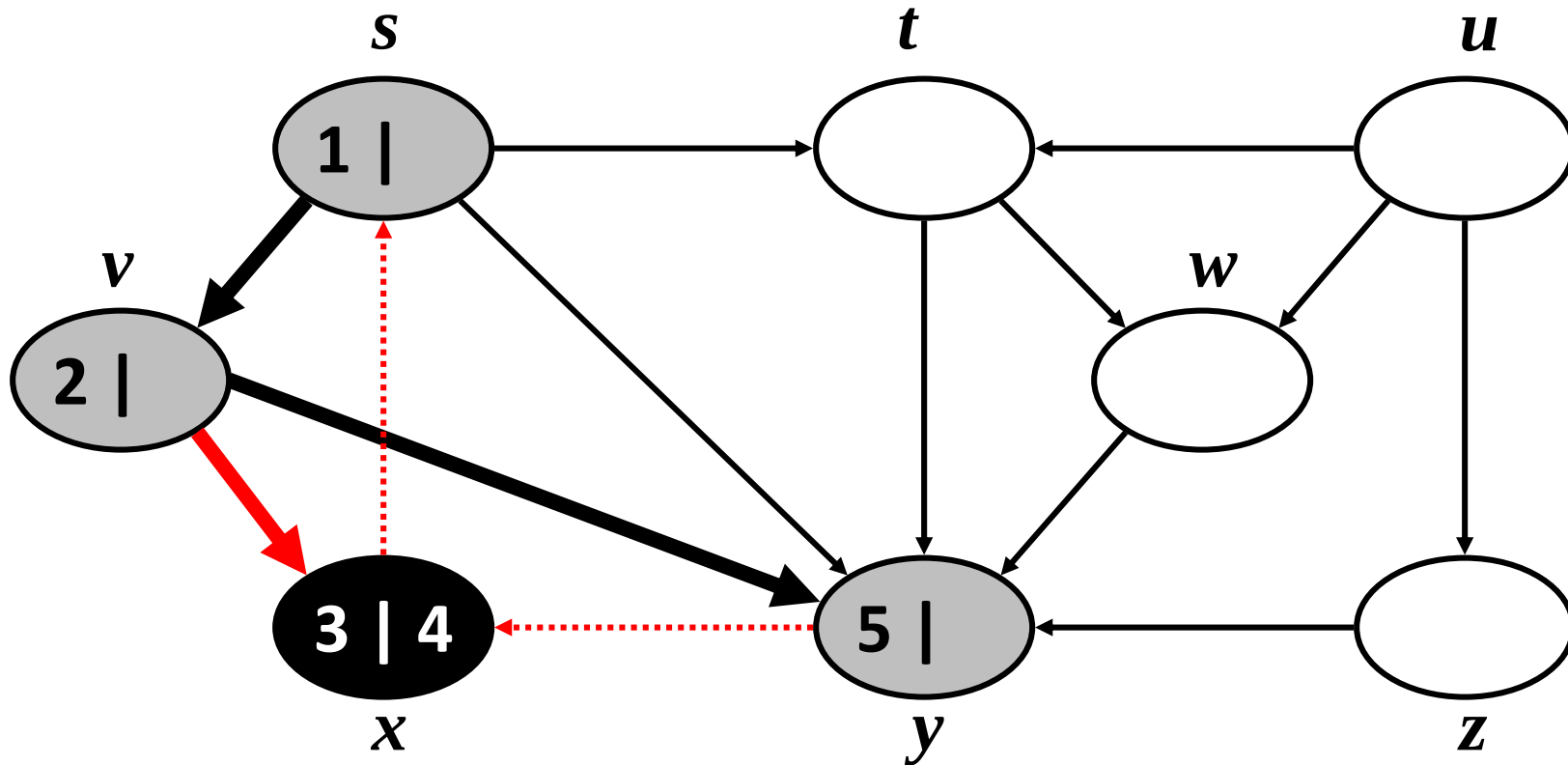
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



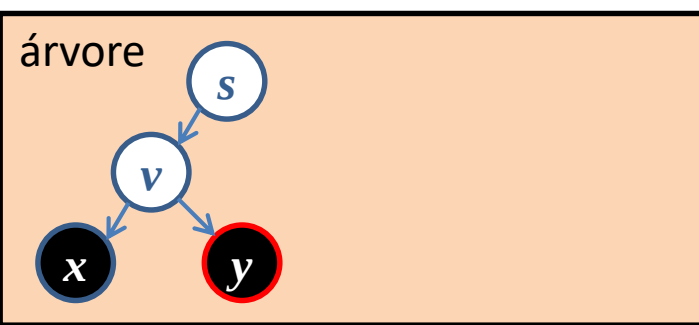
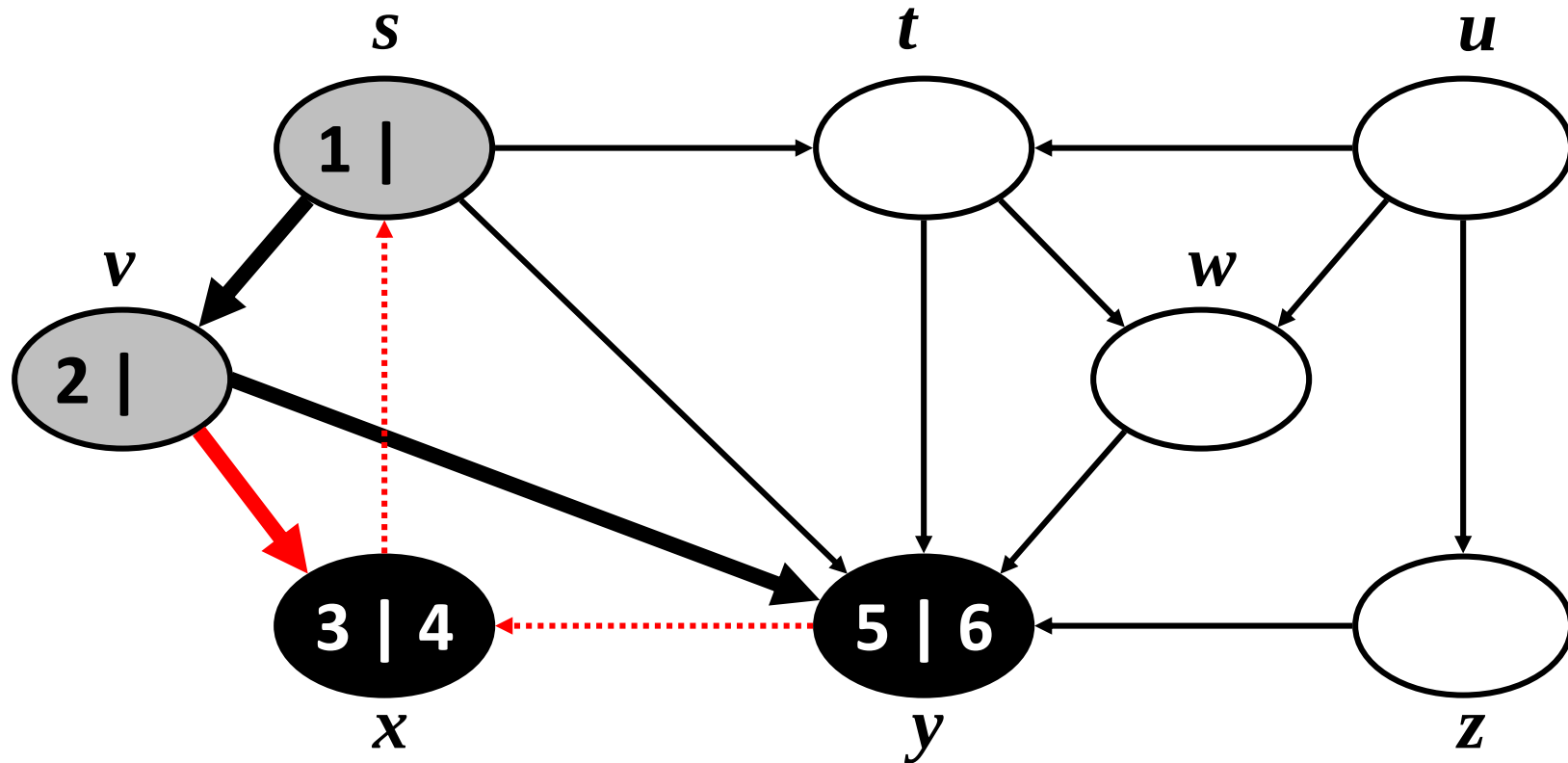
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



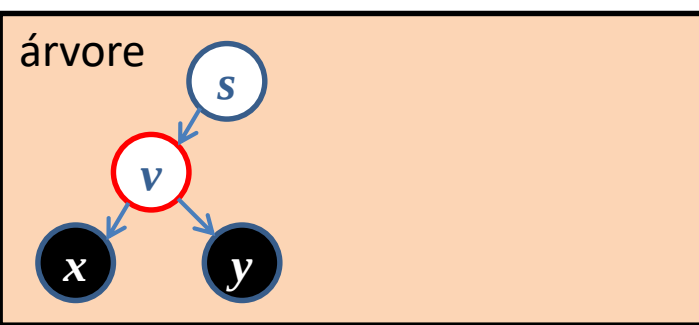
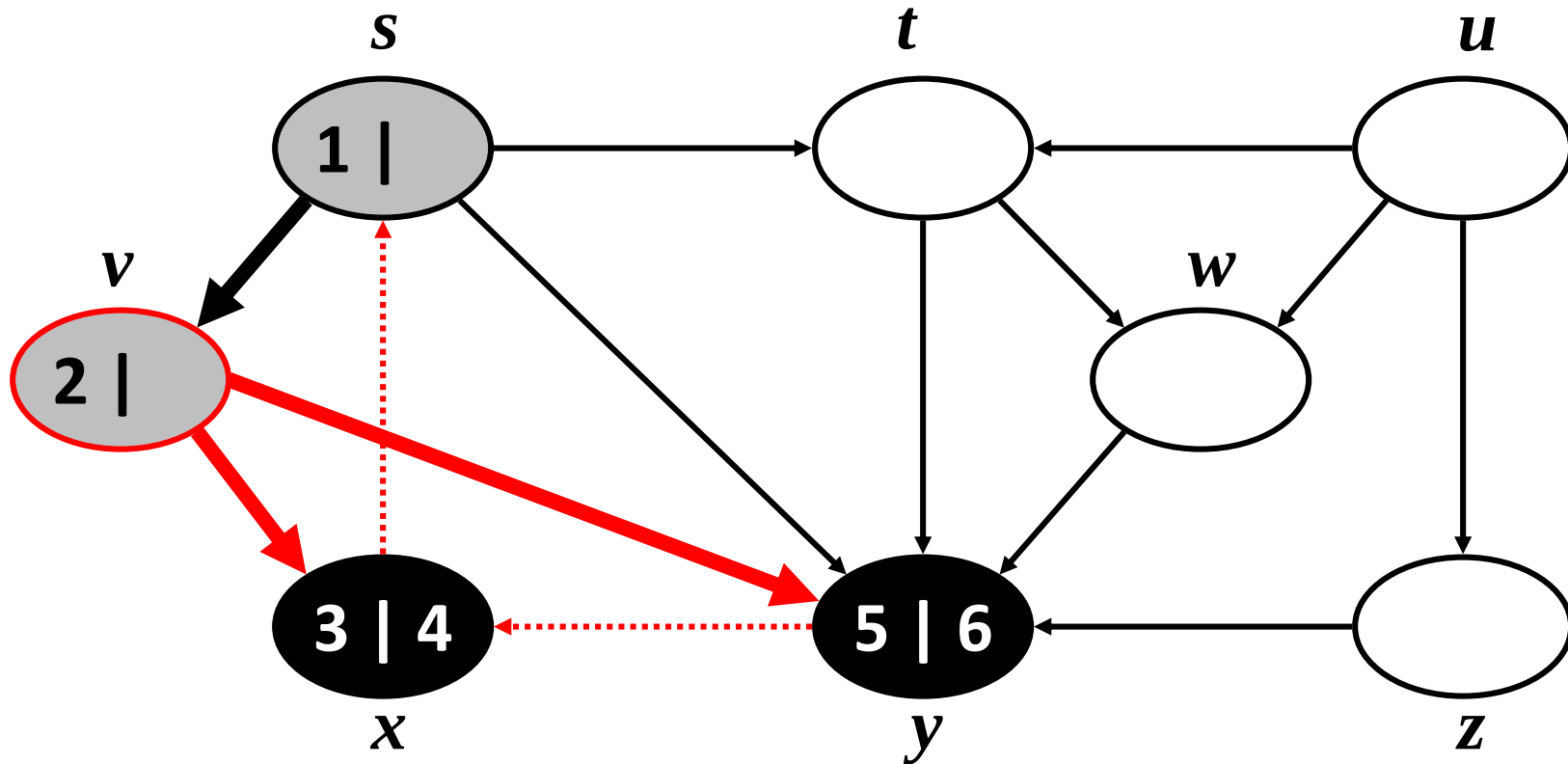
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



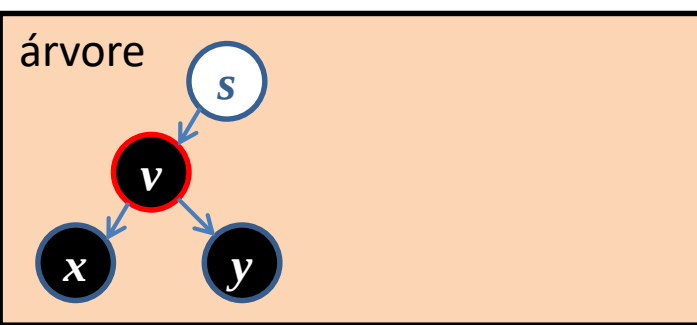
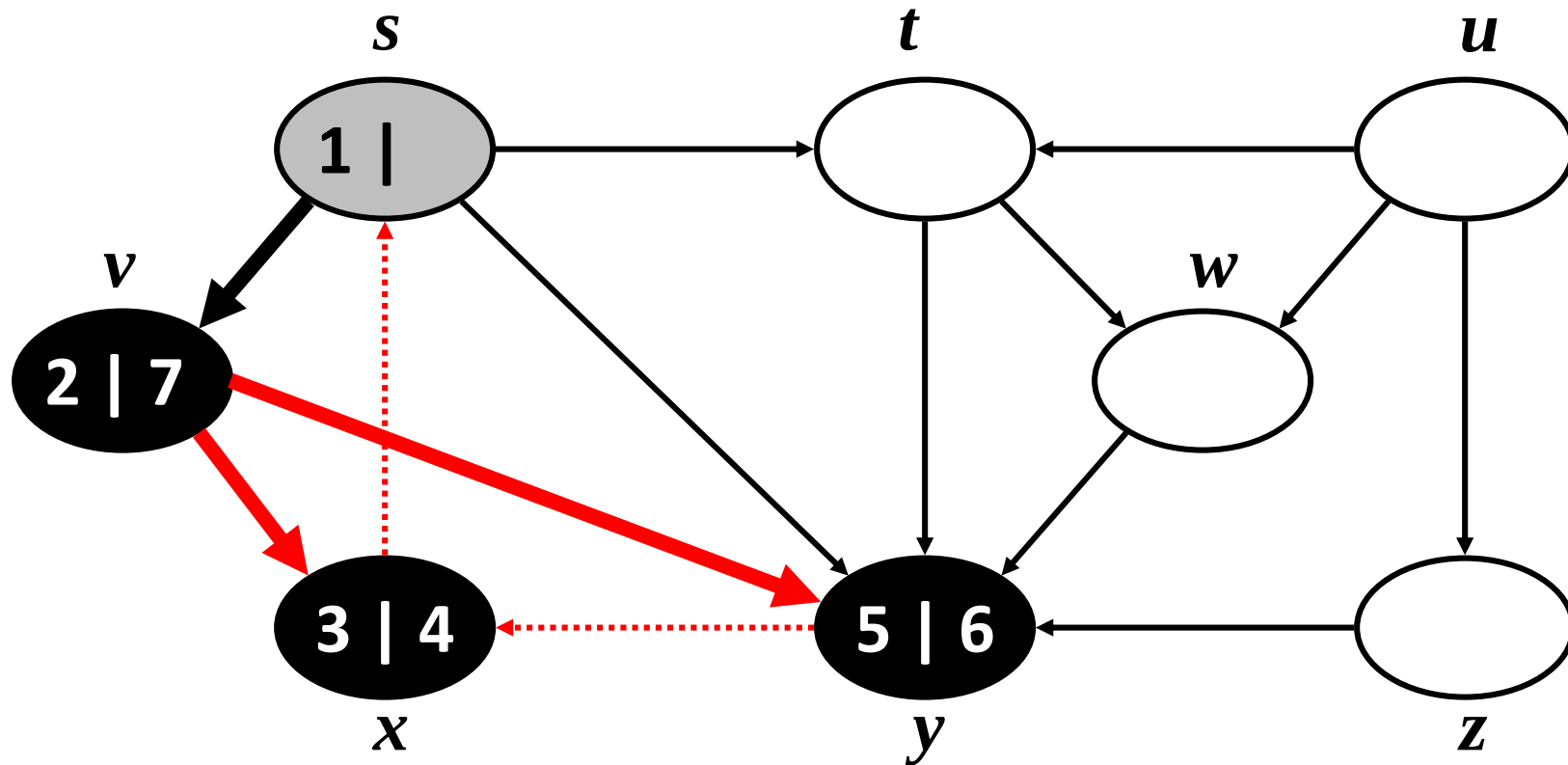
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



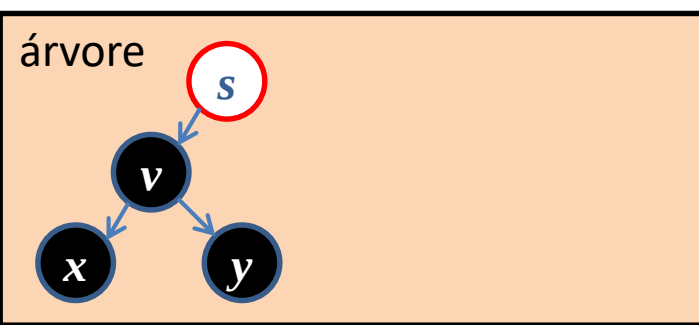
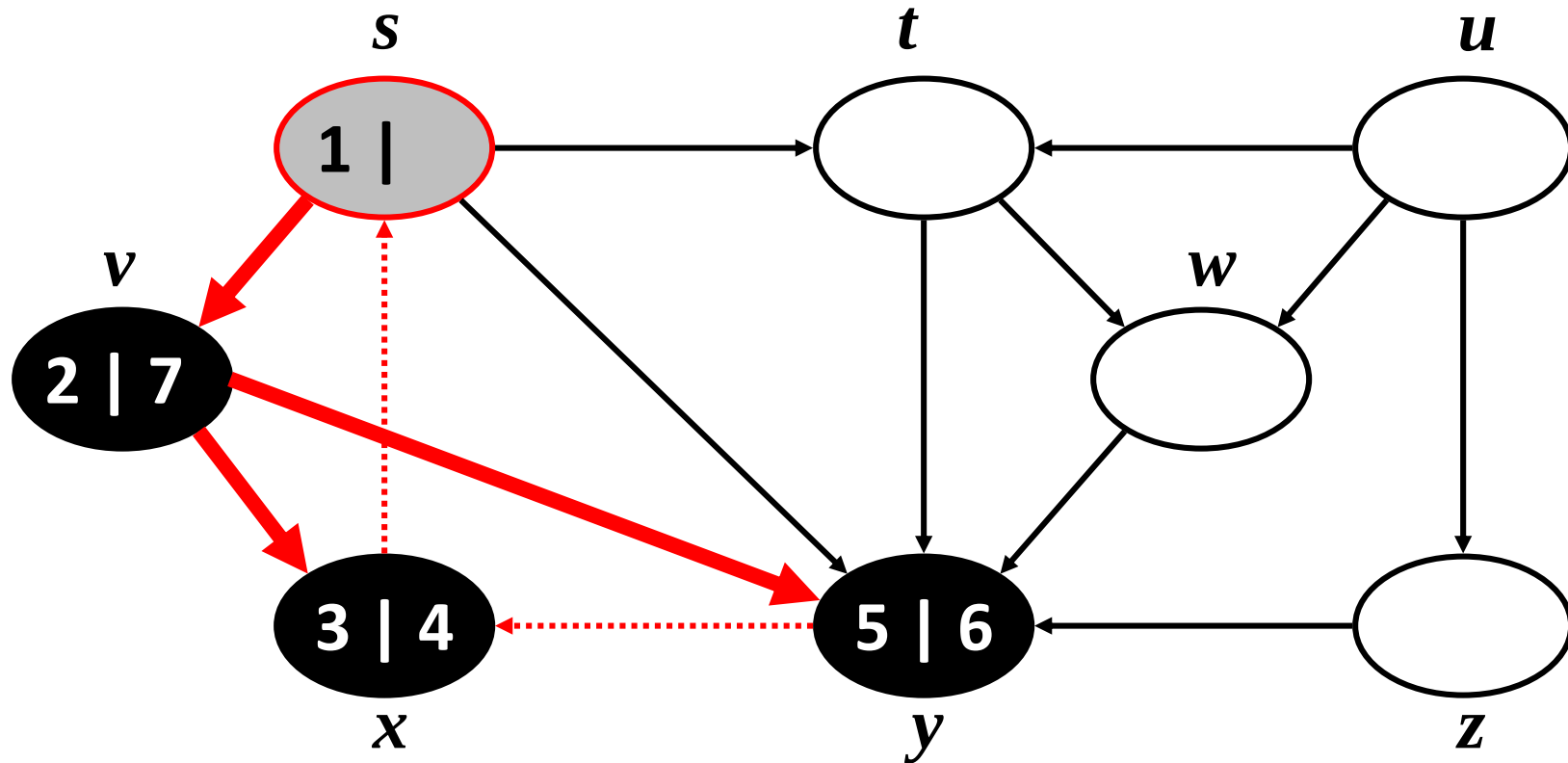
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



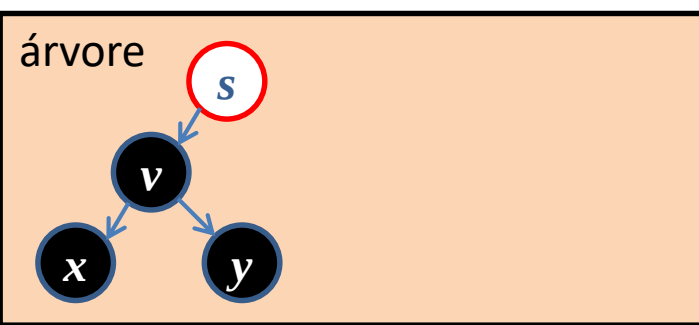
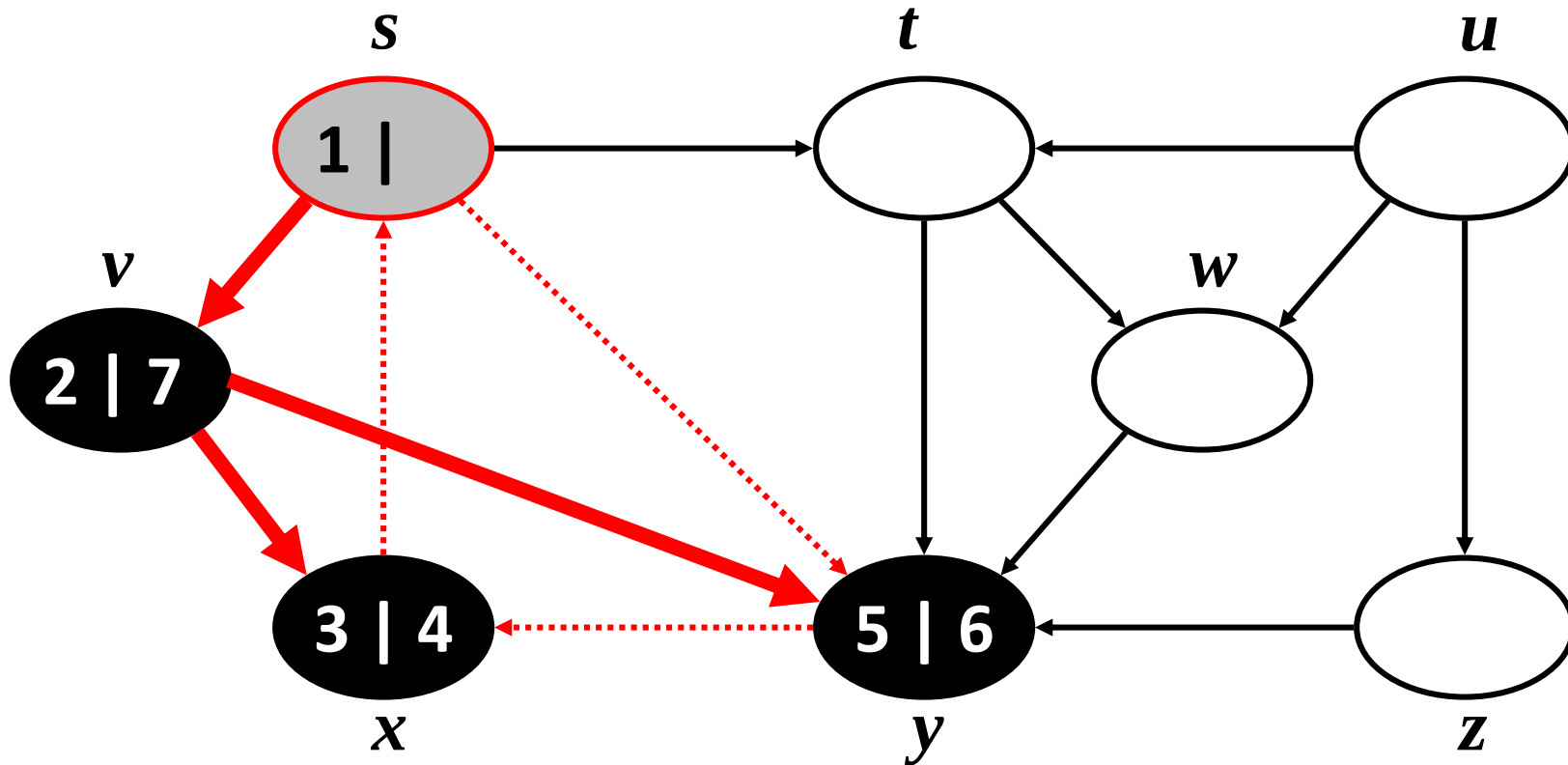
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



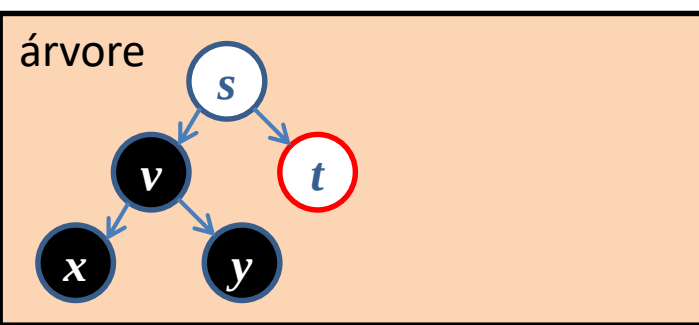
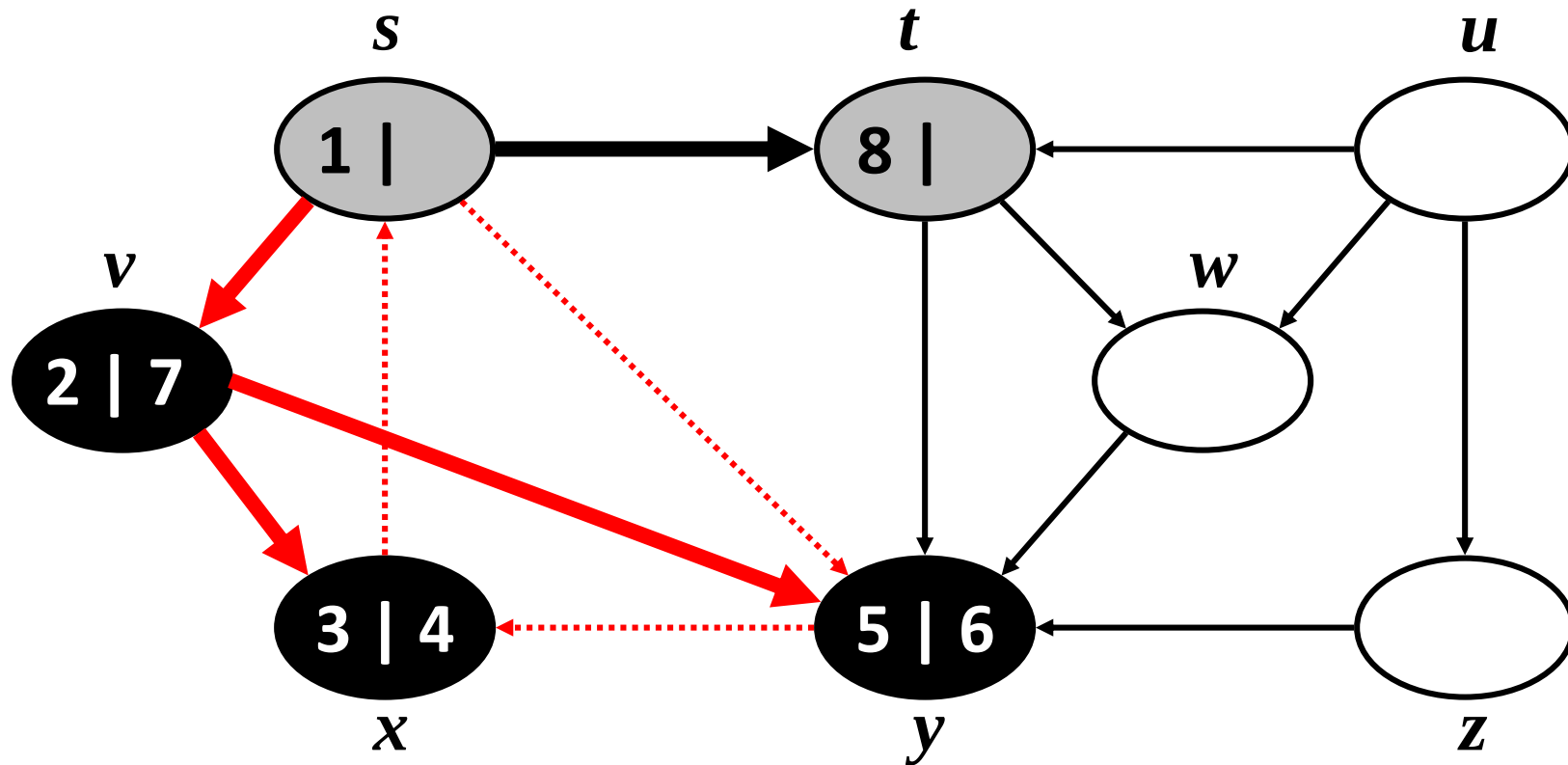
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



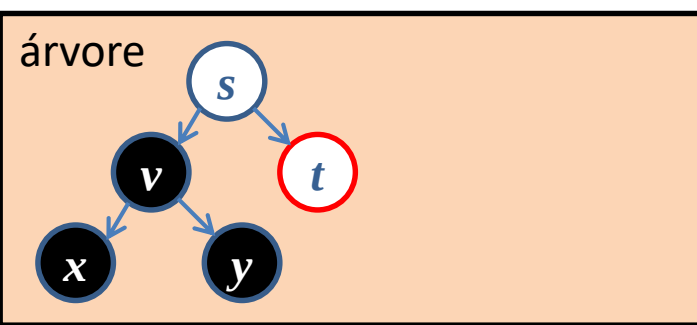
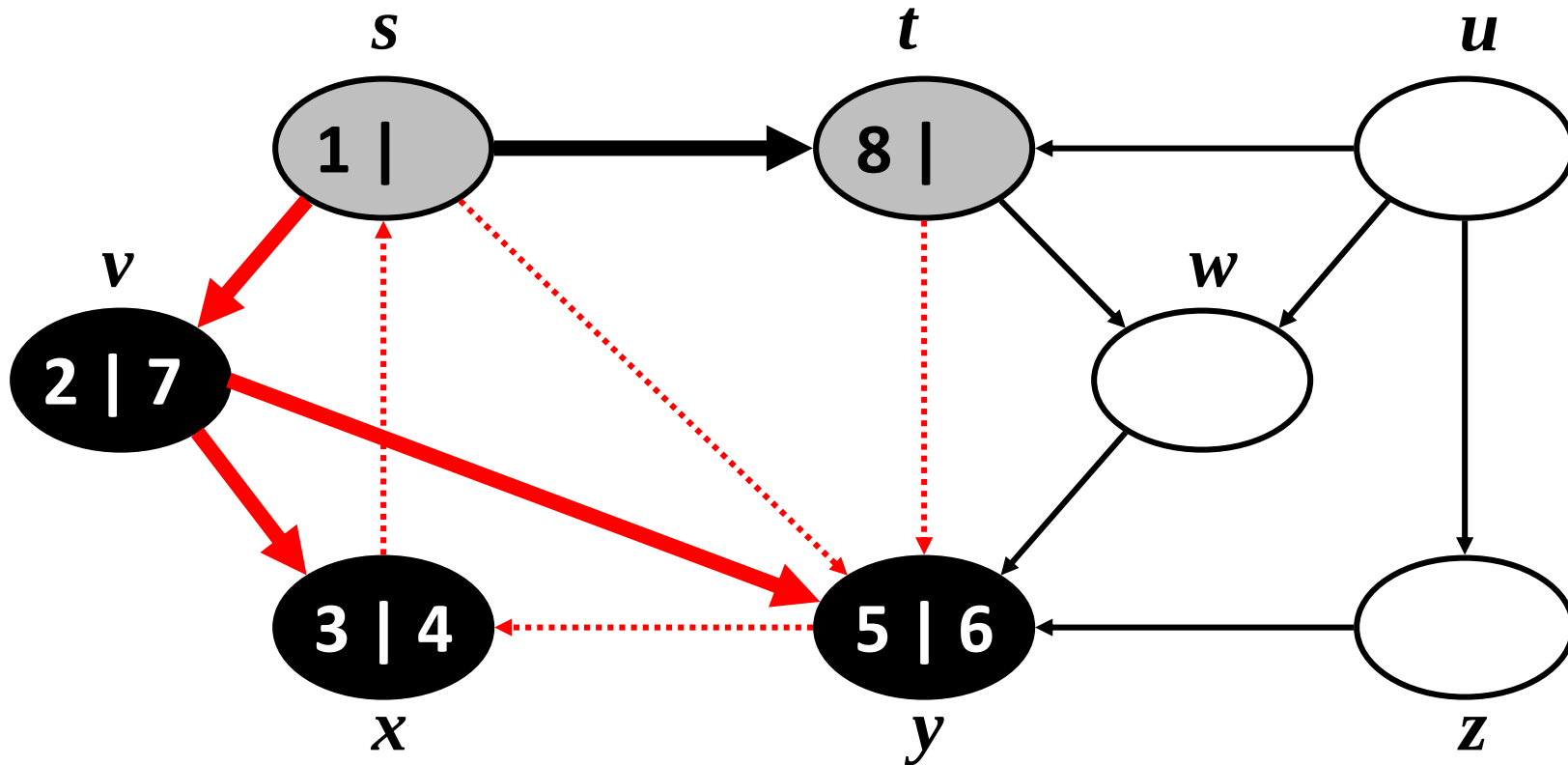
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



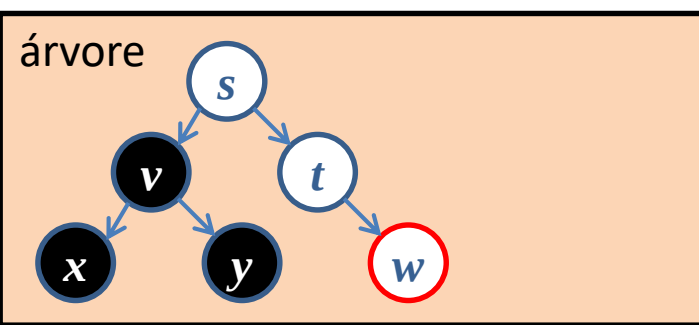
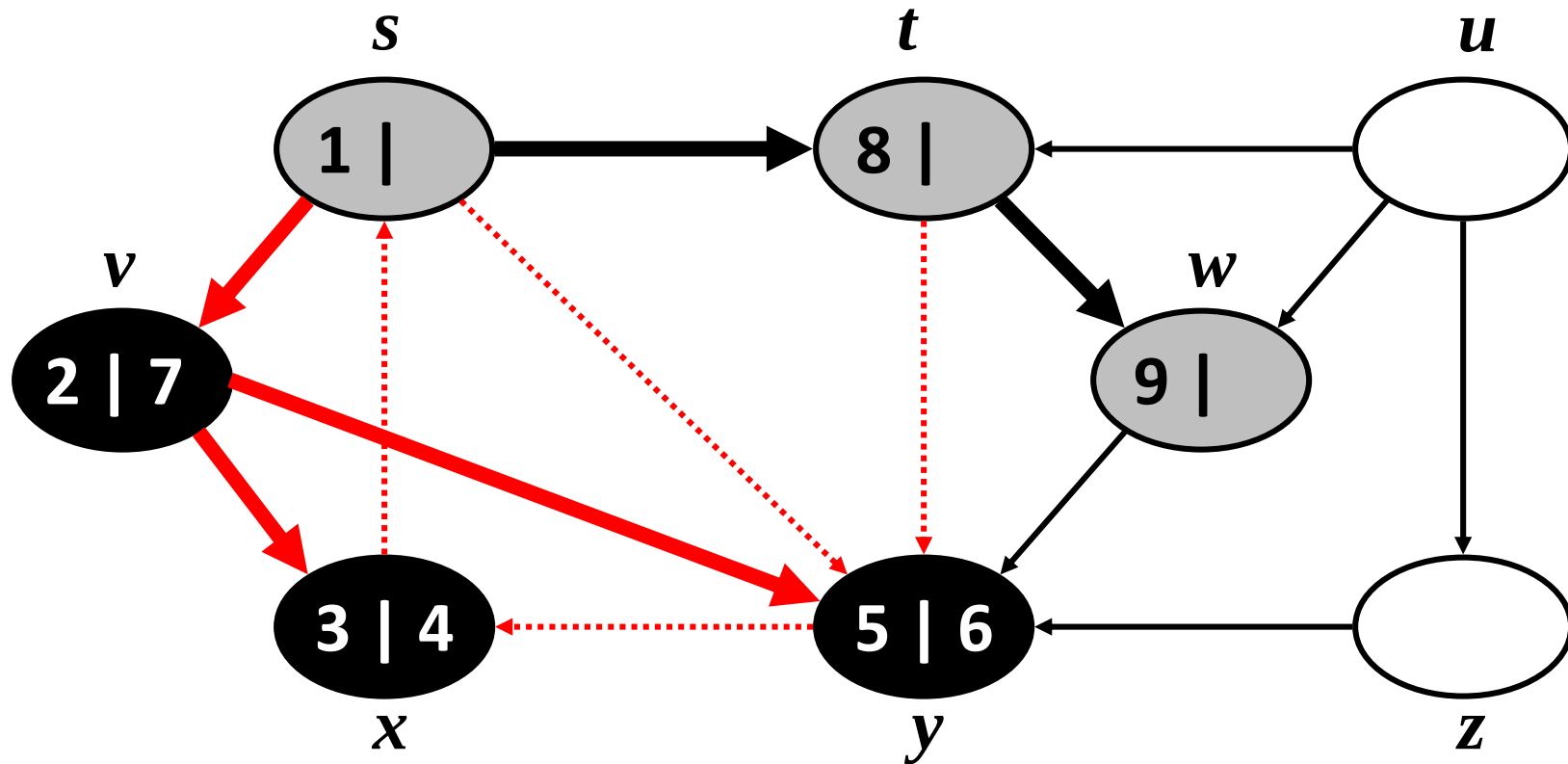
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



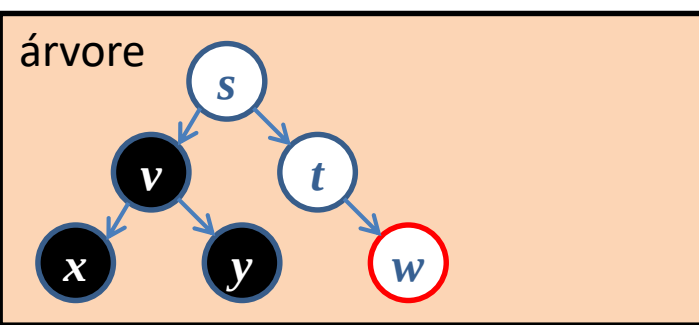
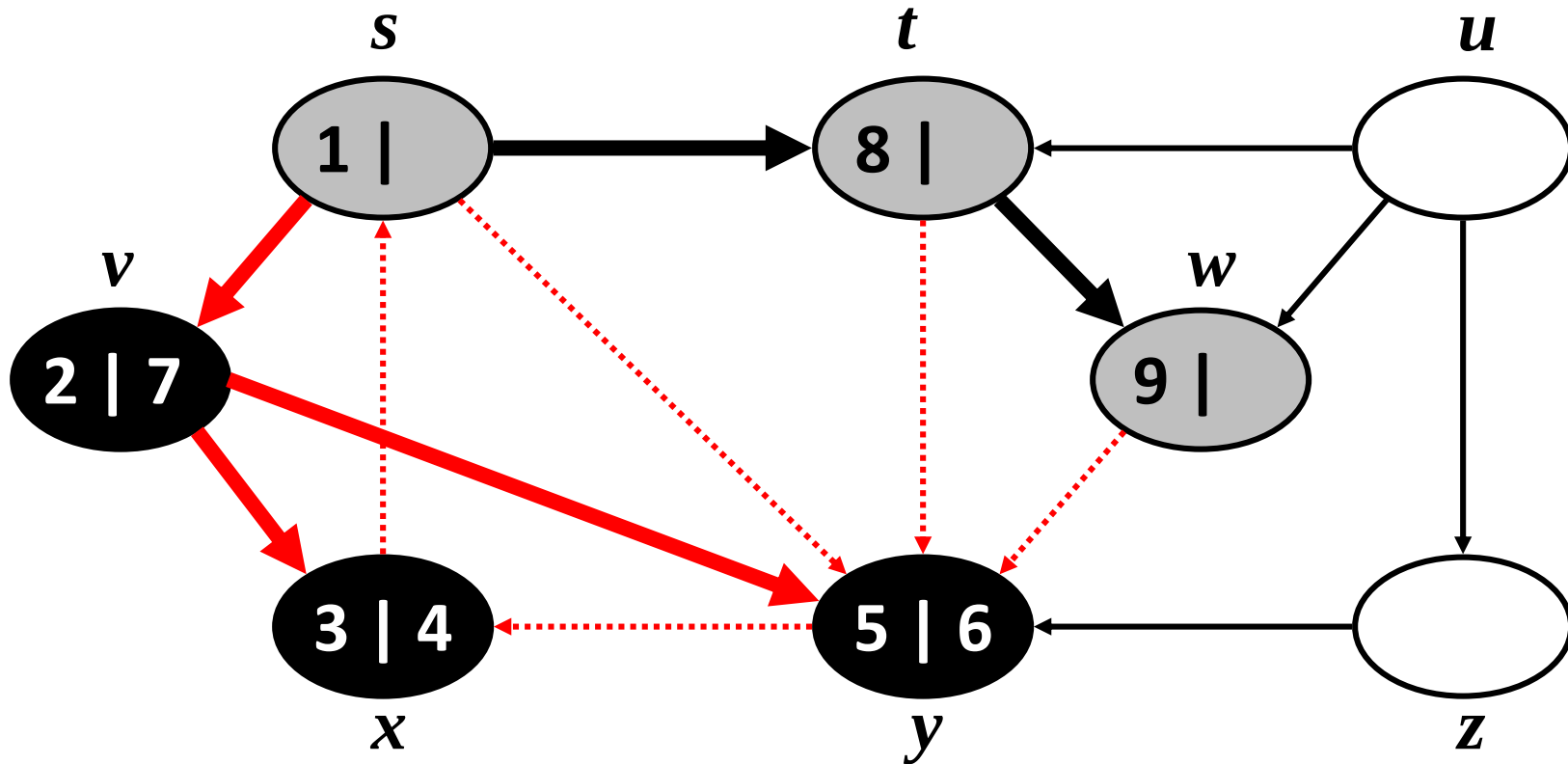
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



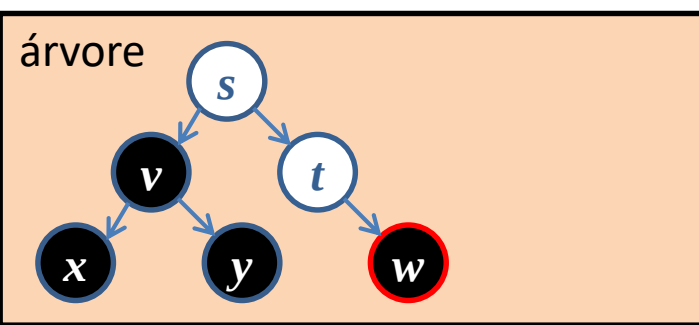
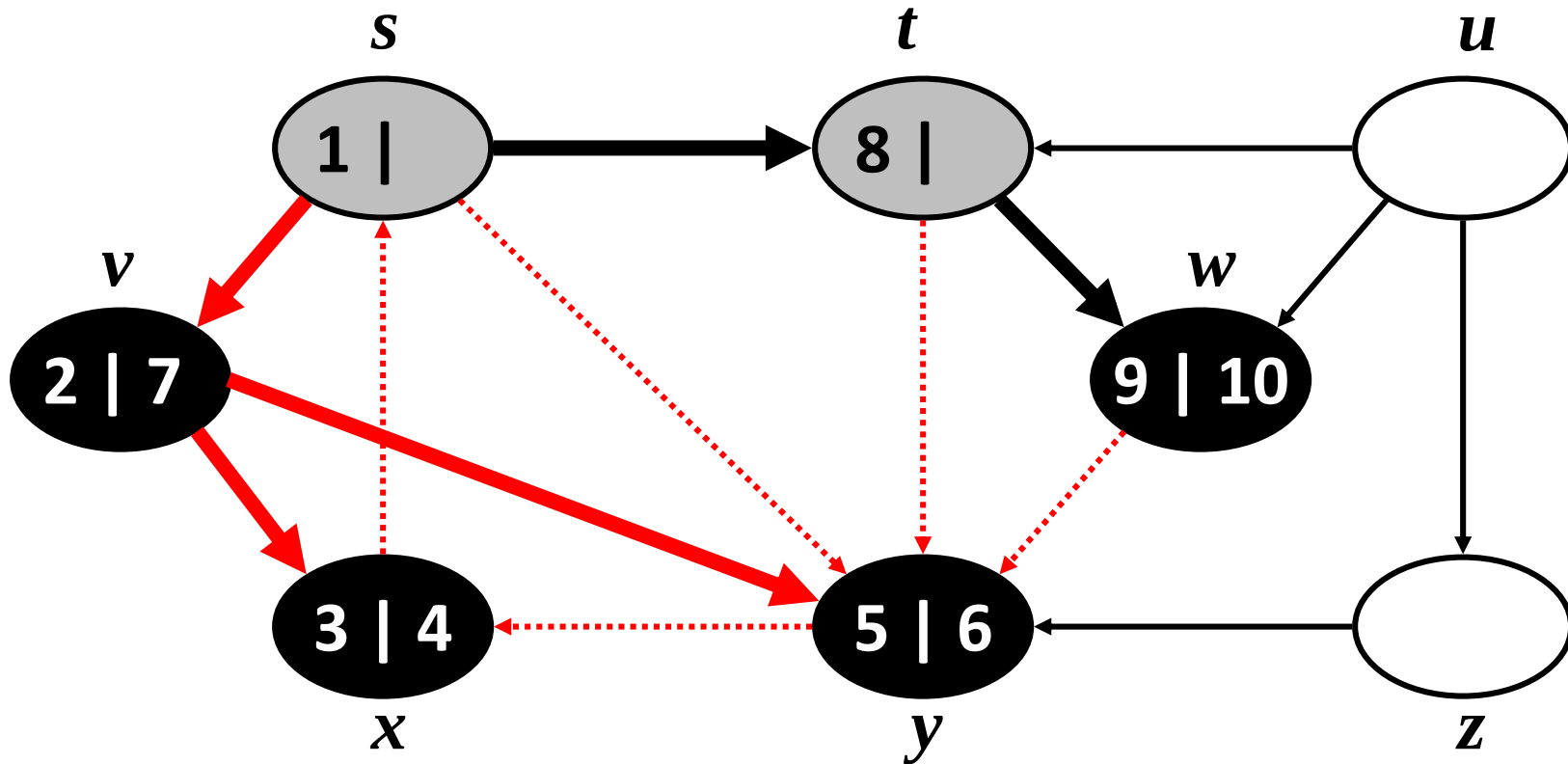
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



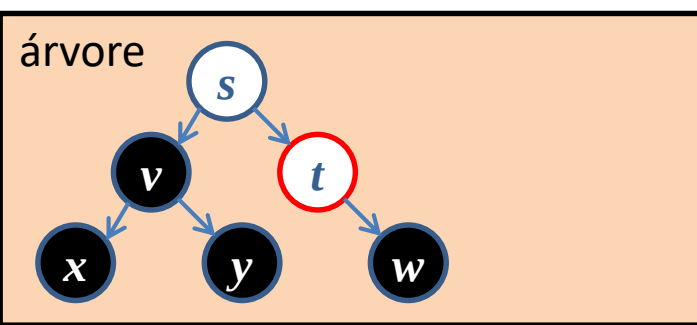
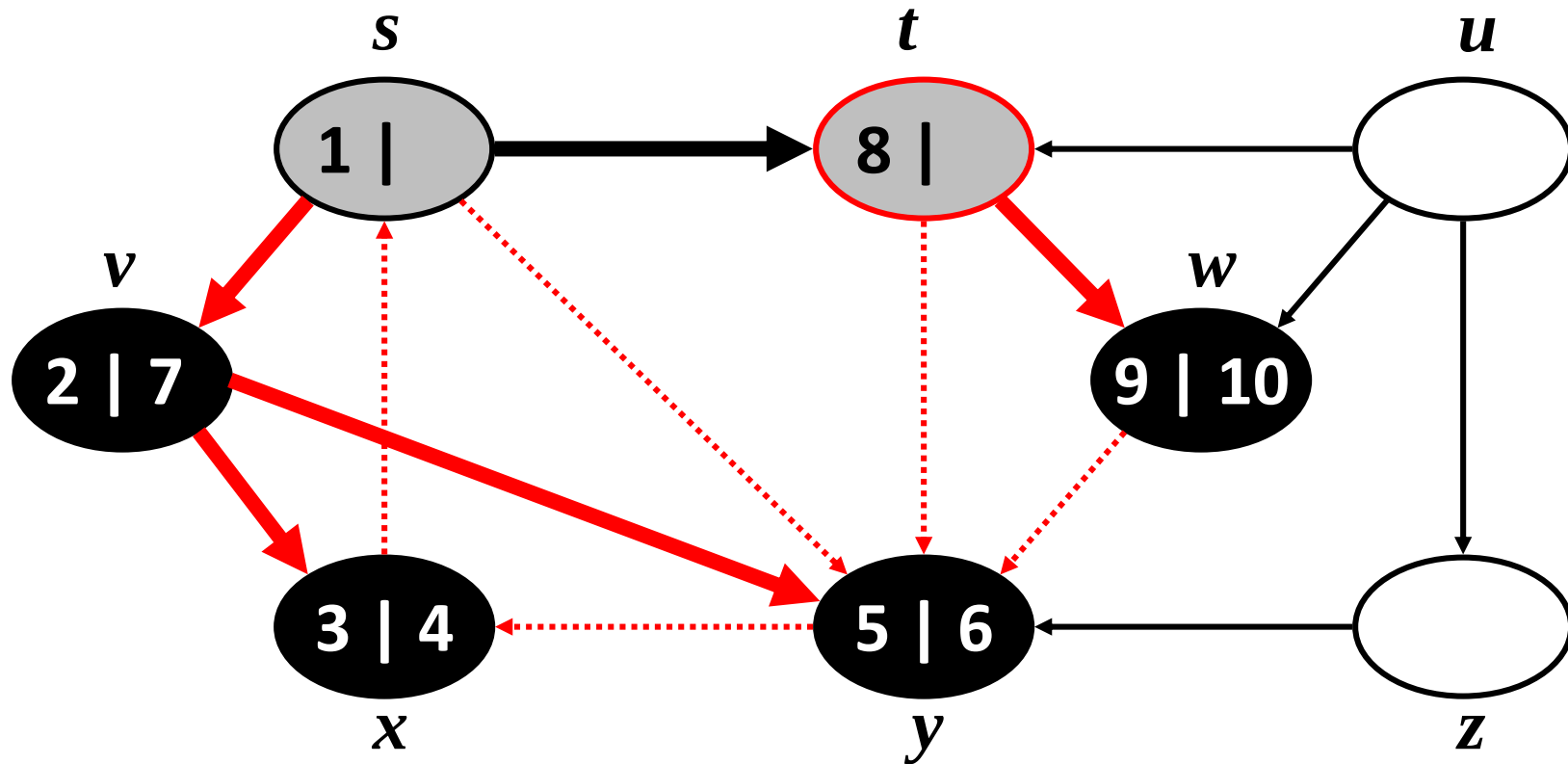
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



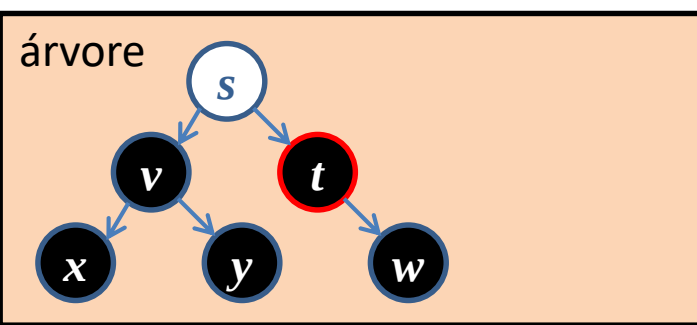
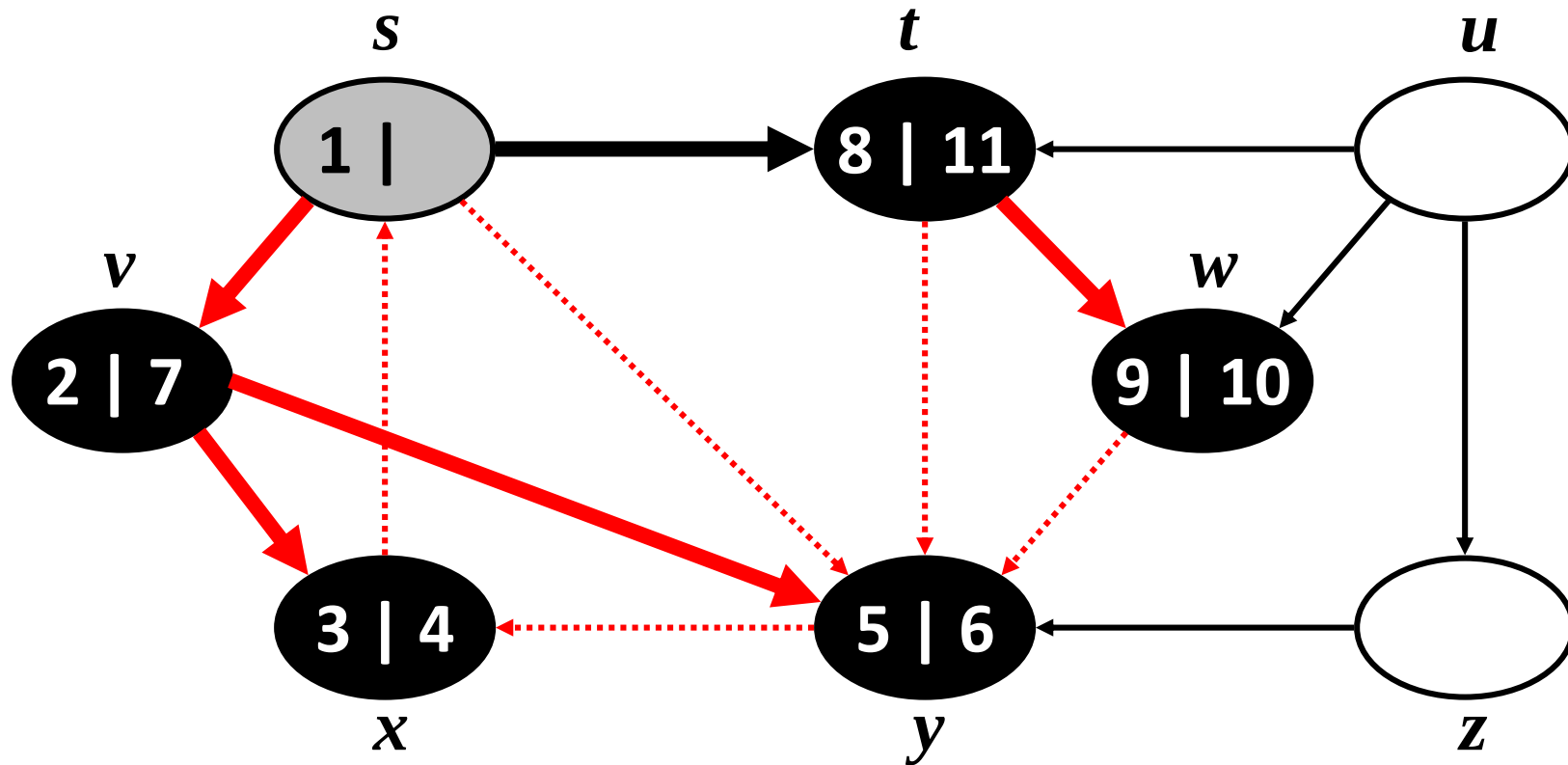
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



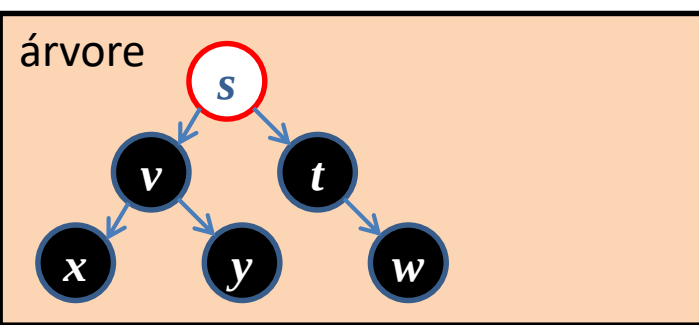
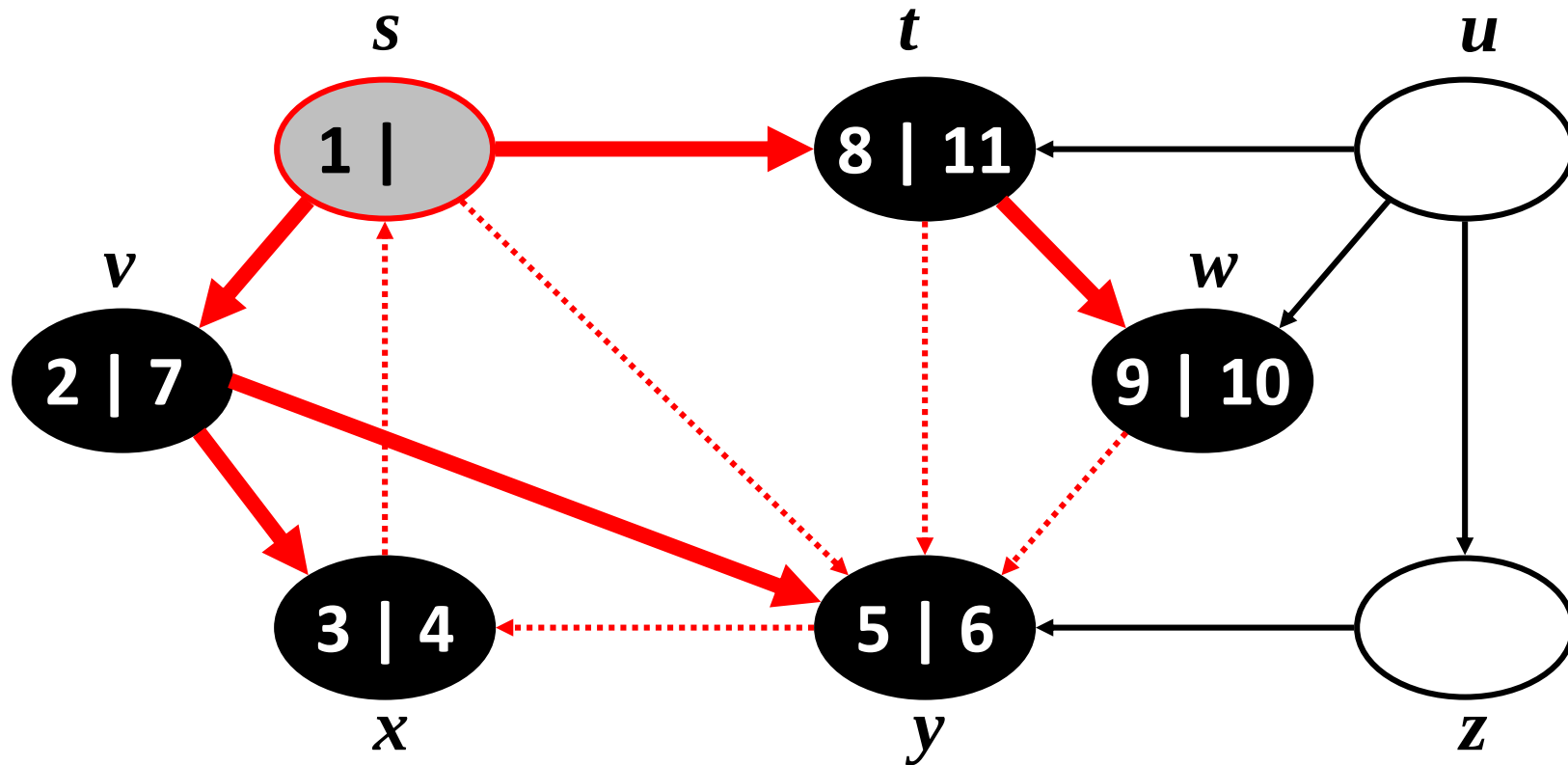
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



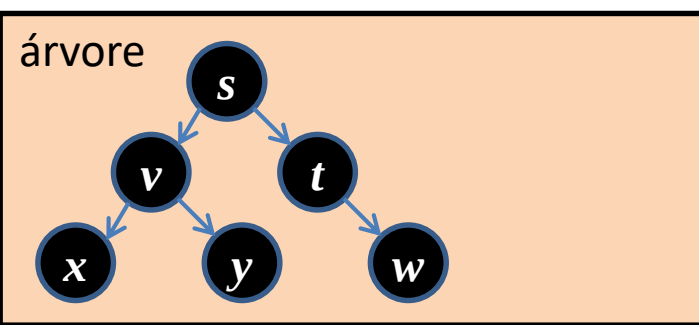
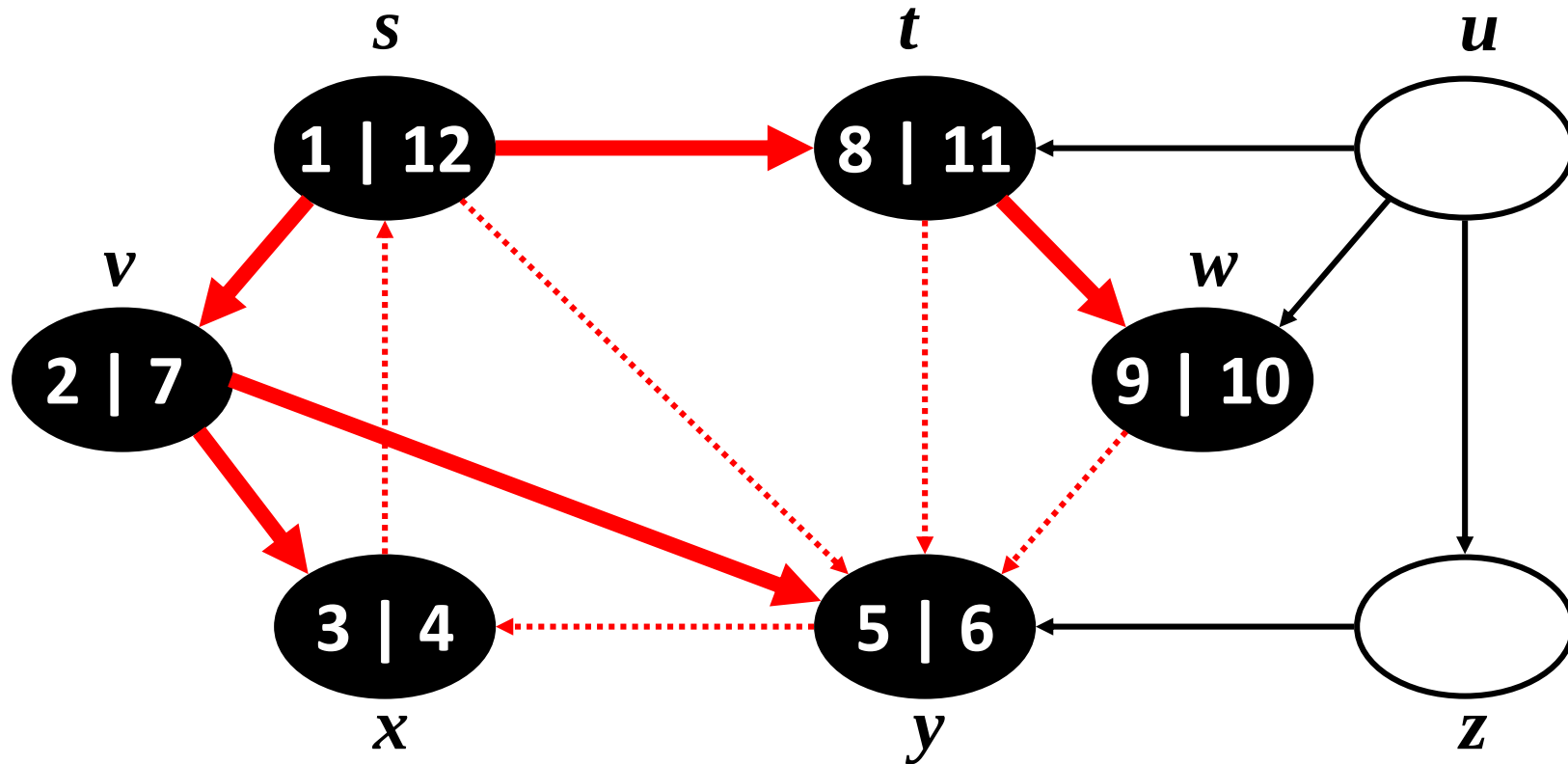
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



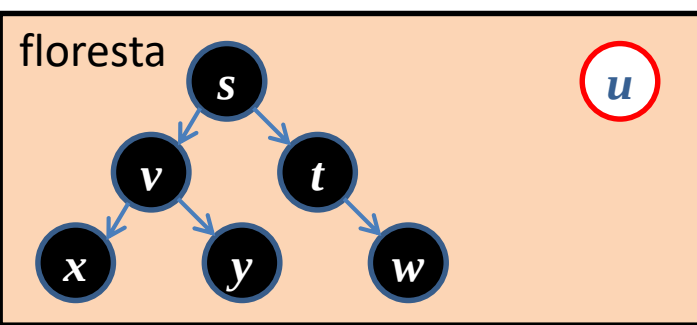
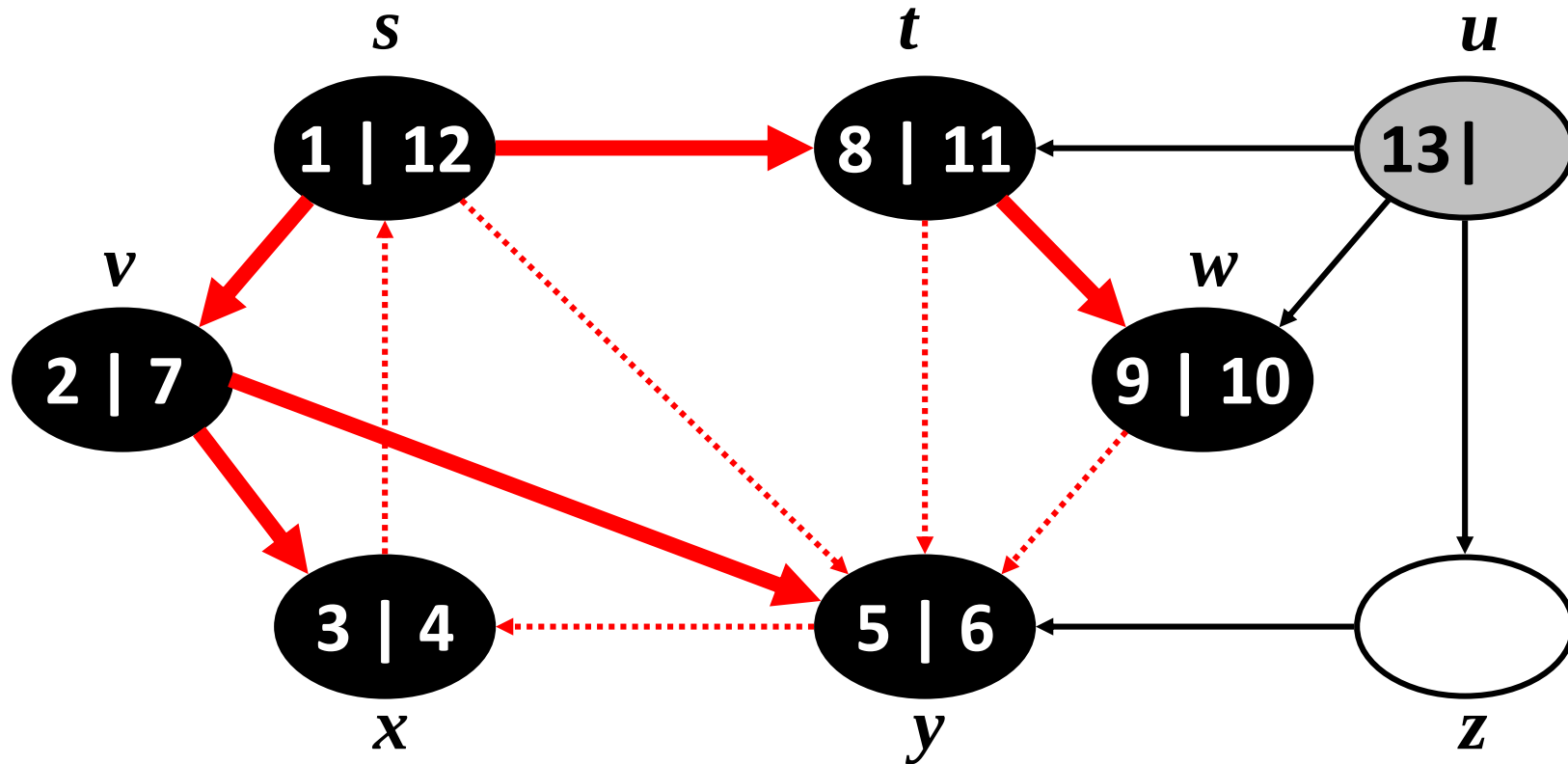
3 - Busca em Profundidade (Depth-First Search)

Exemplo DFS:



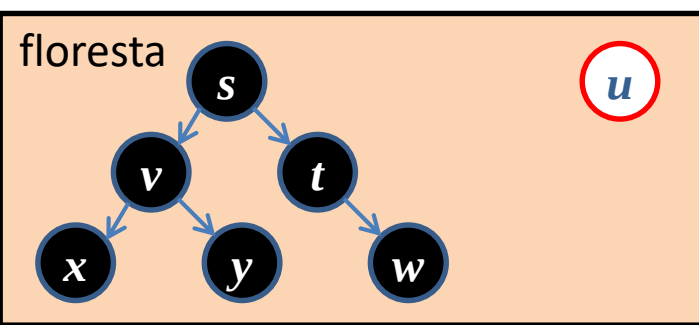
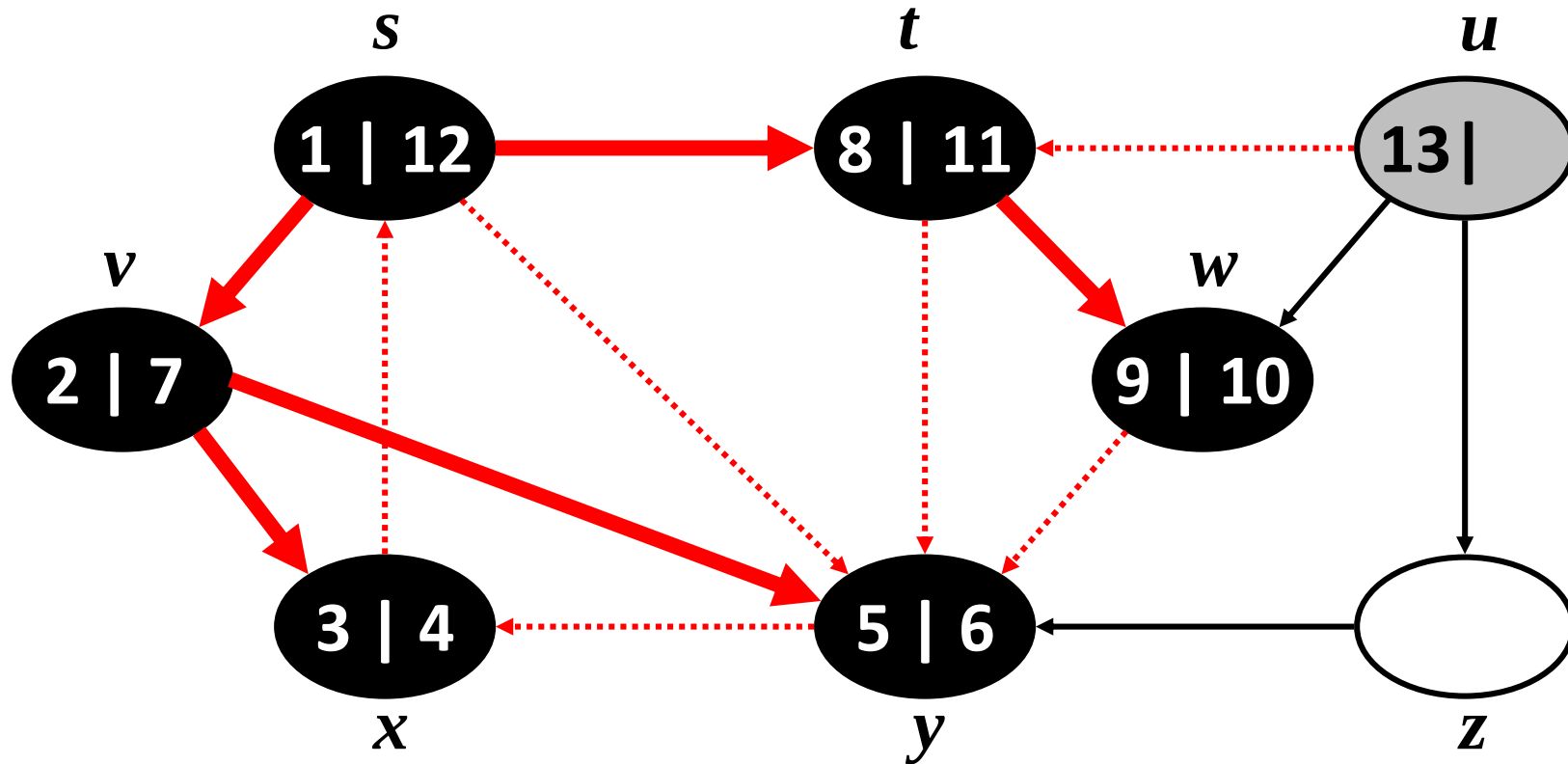
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



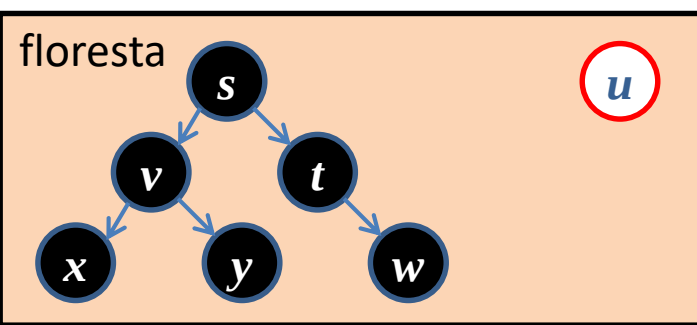
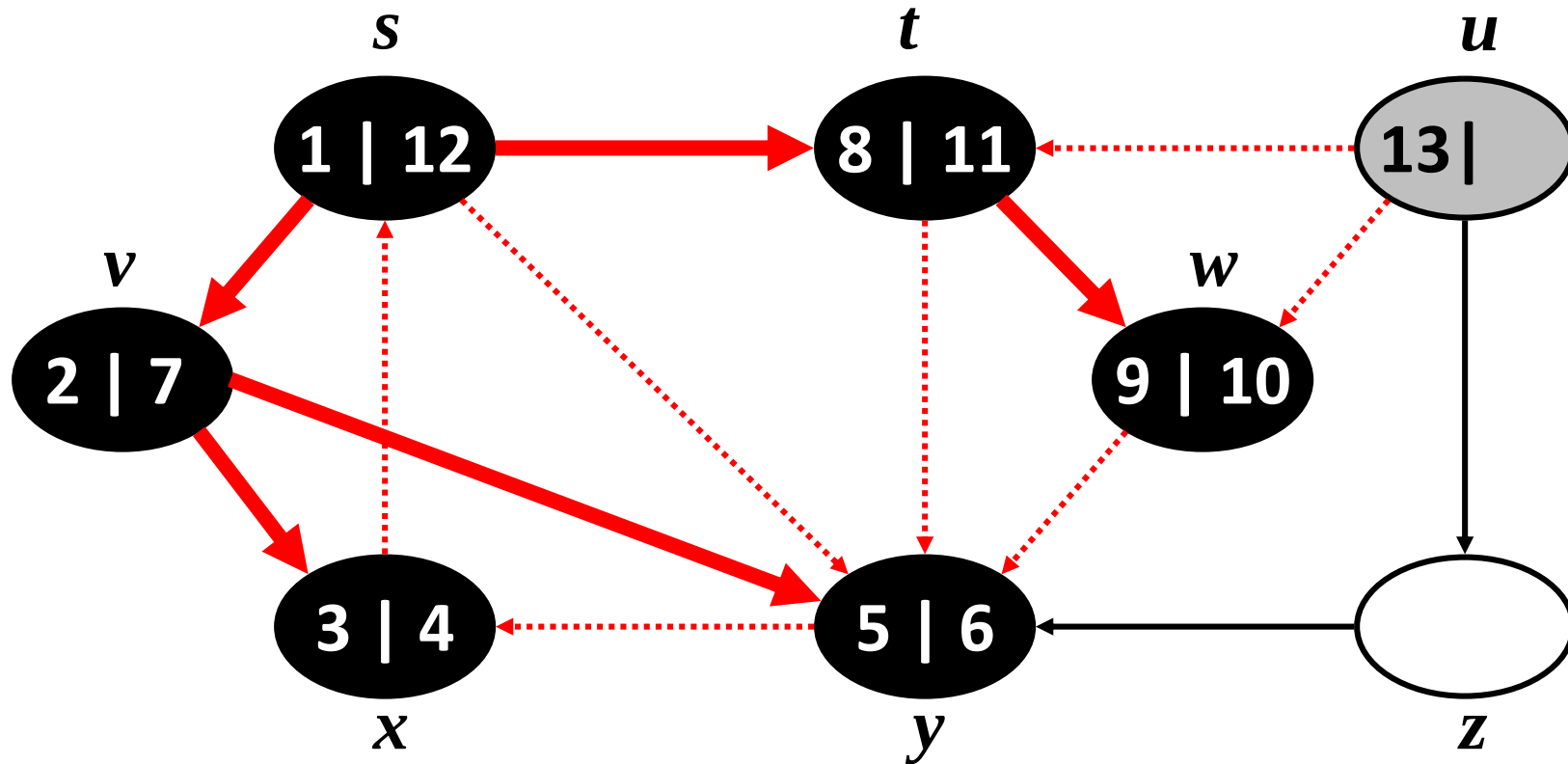
3 - Busca em Profundidade (Depth-First Search)

Exemplo DFS:



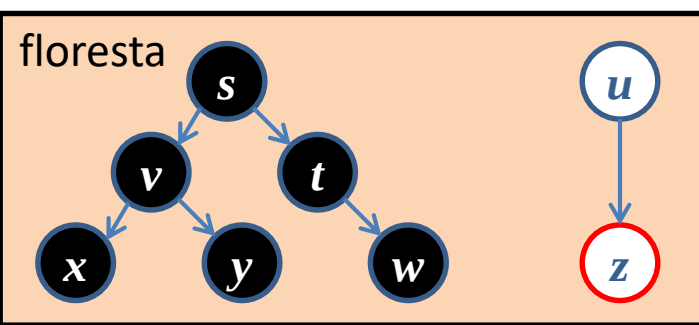
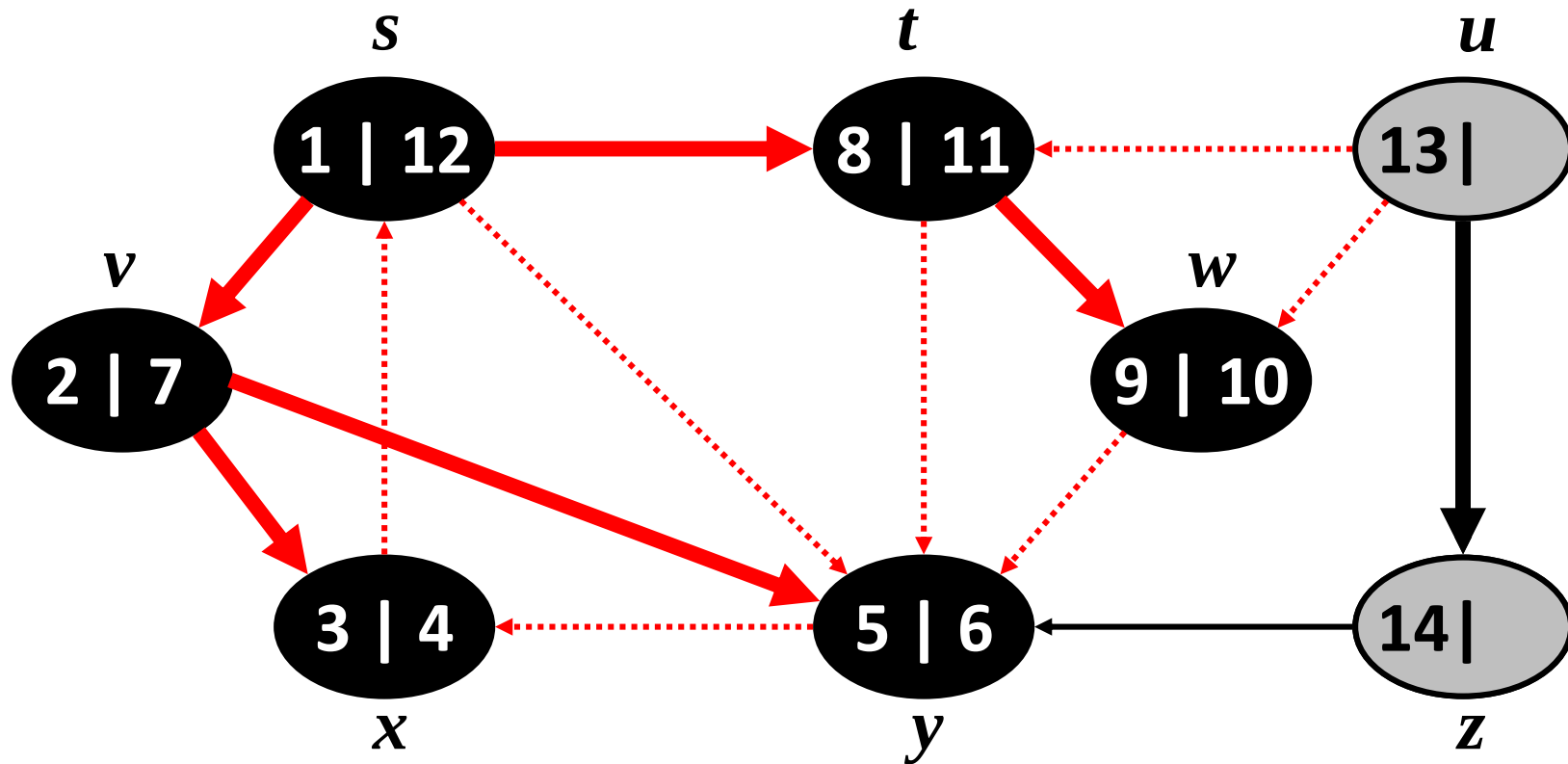
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



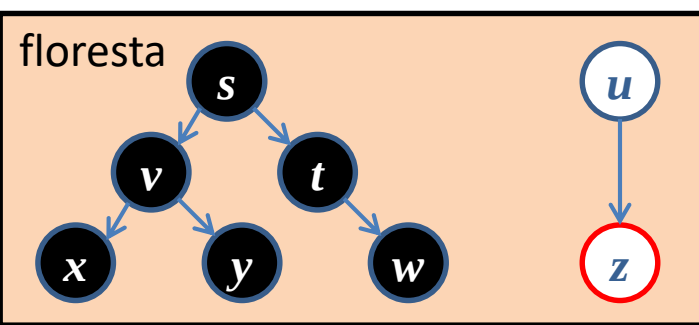
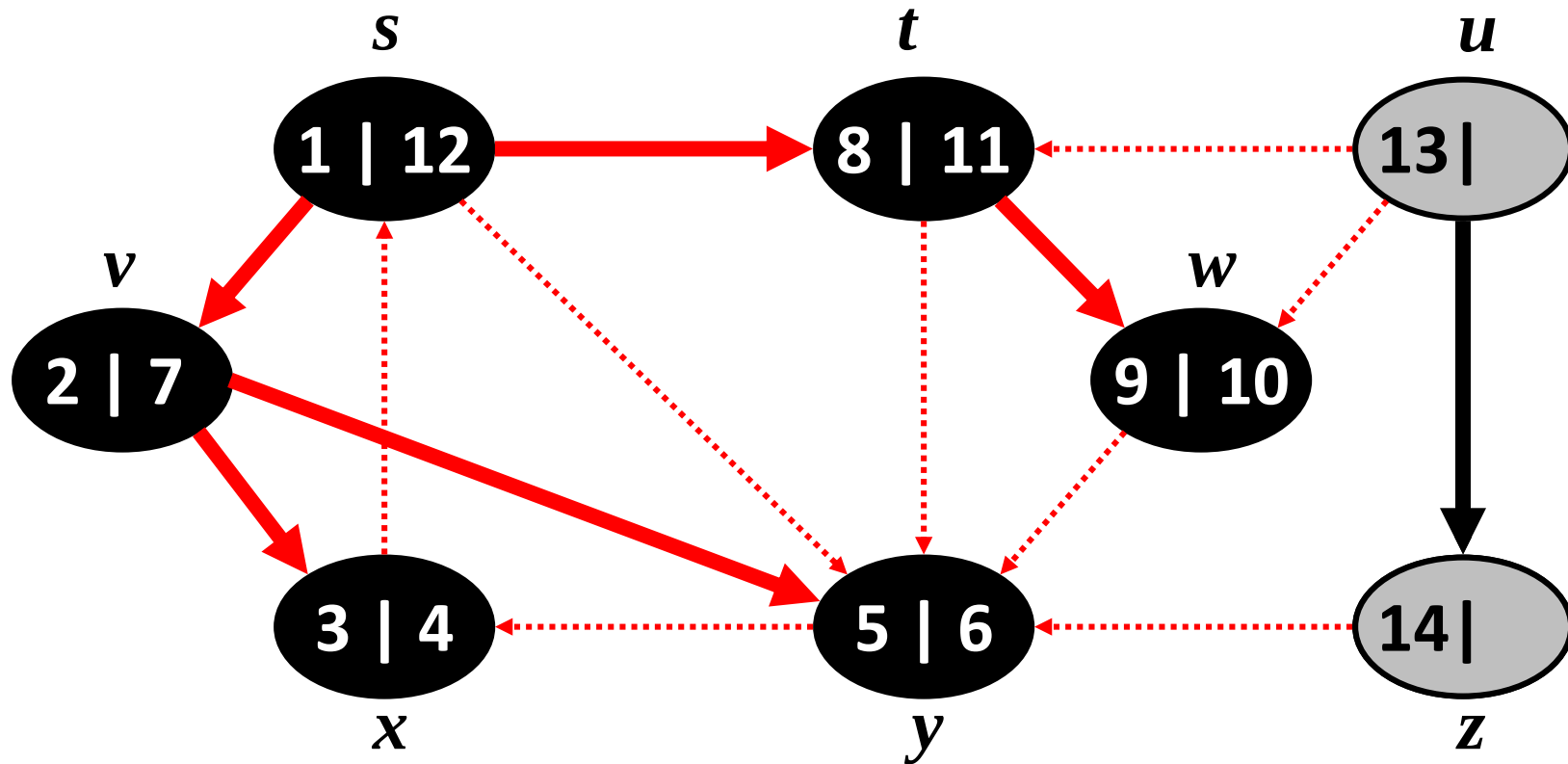
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



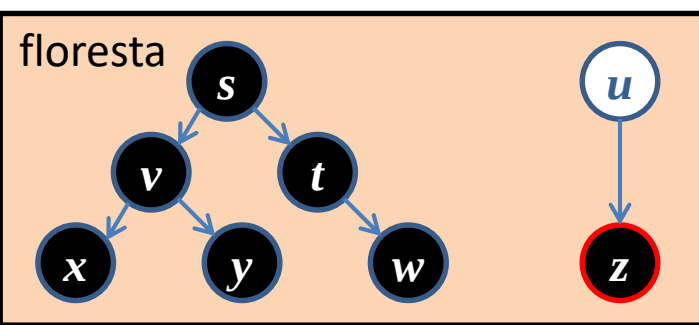
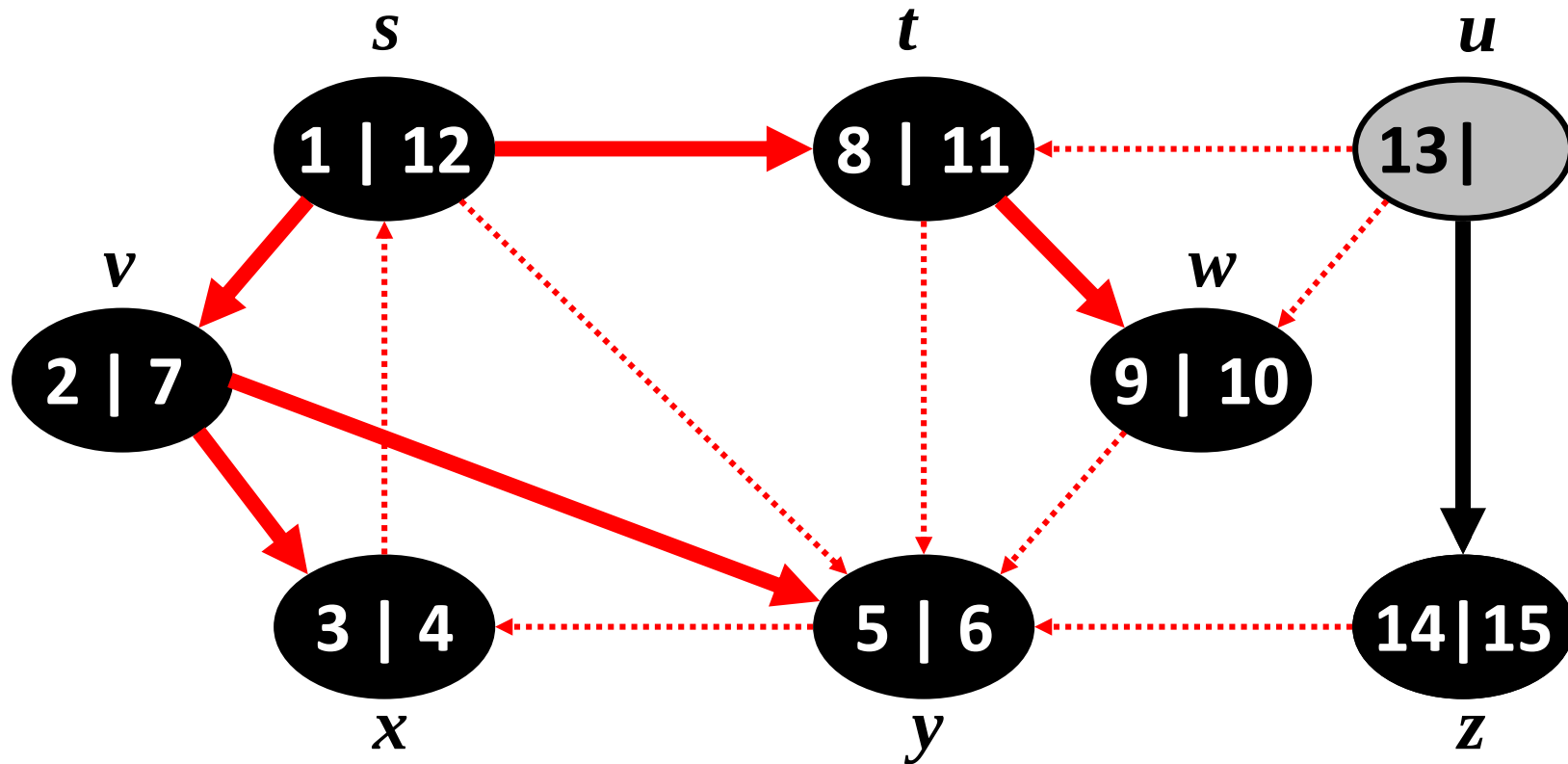
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



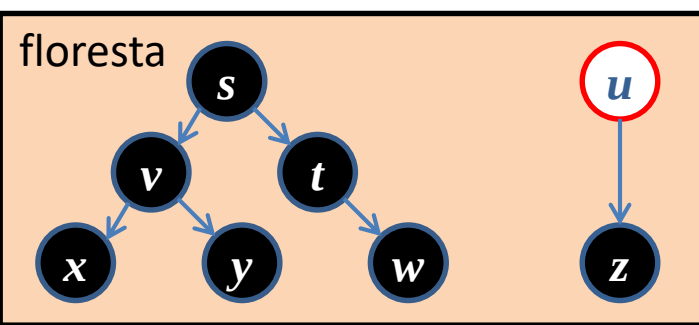
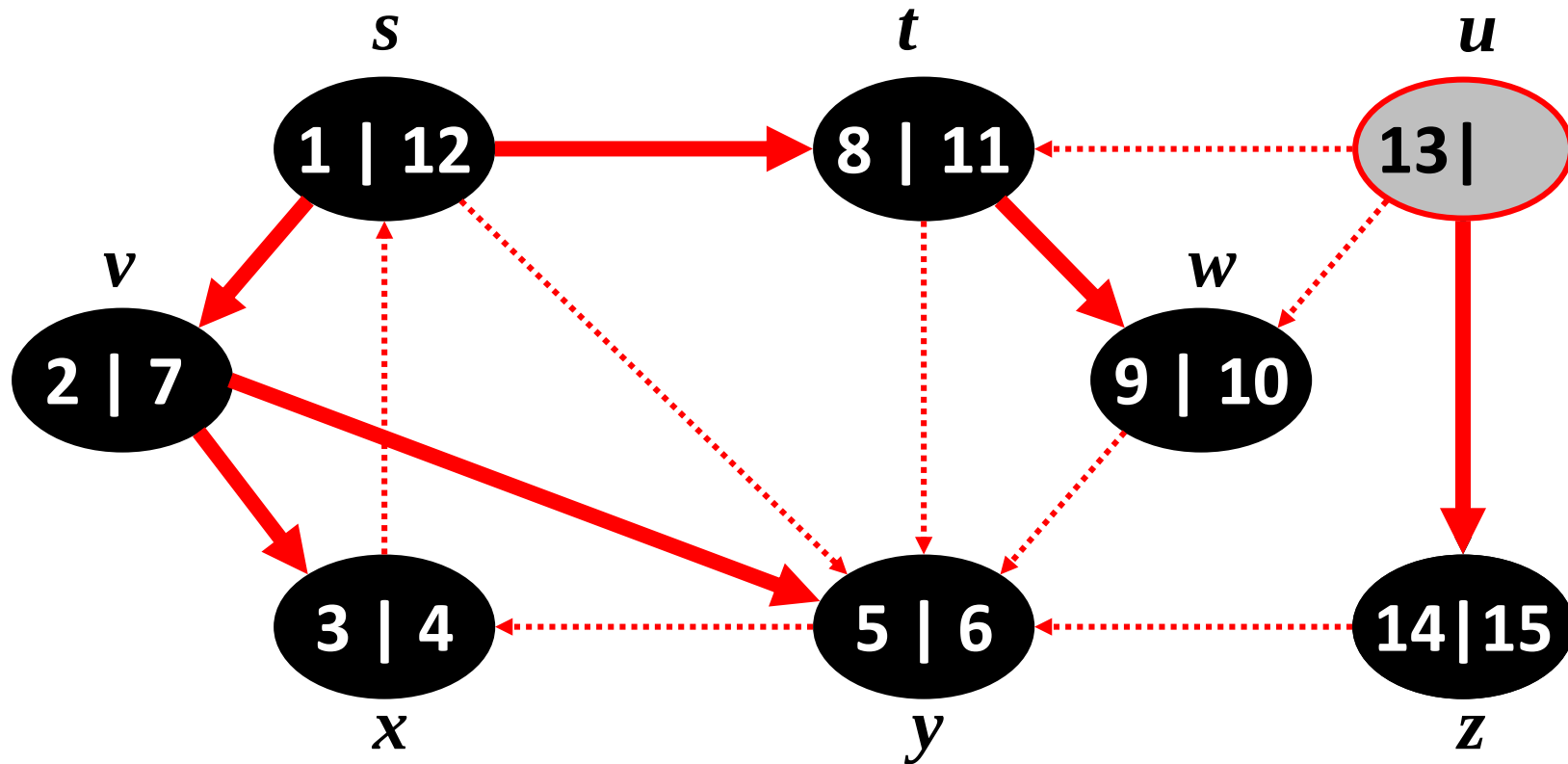
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



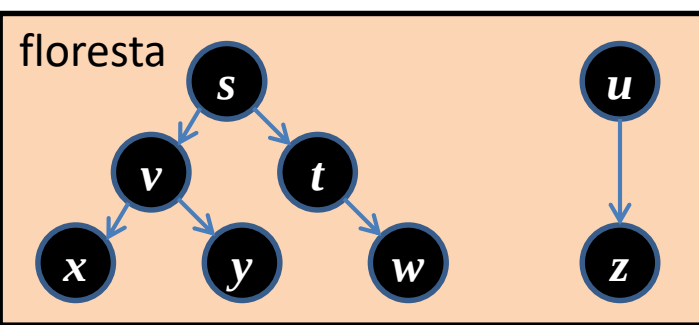
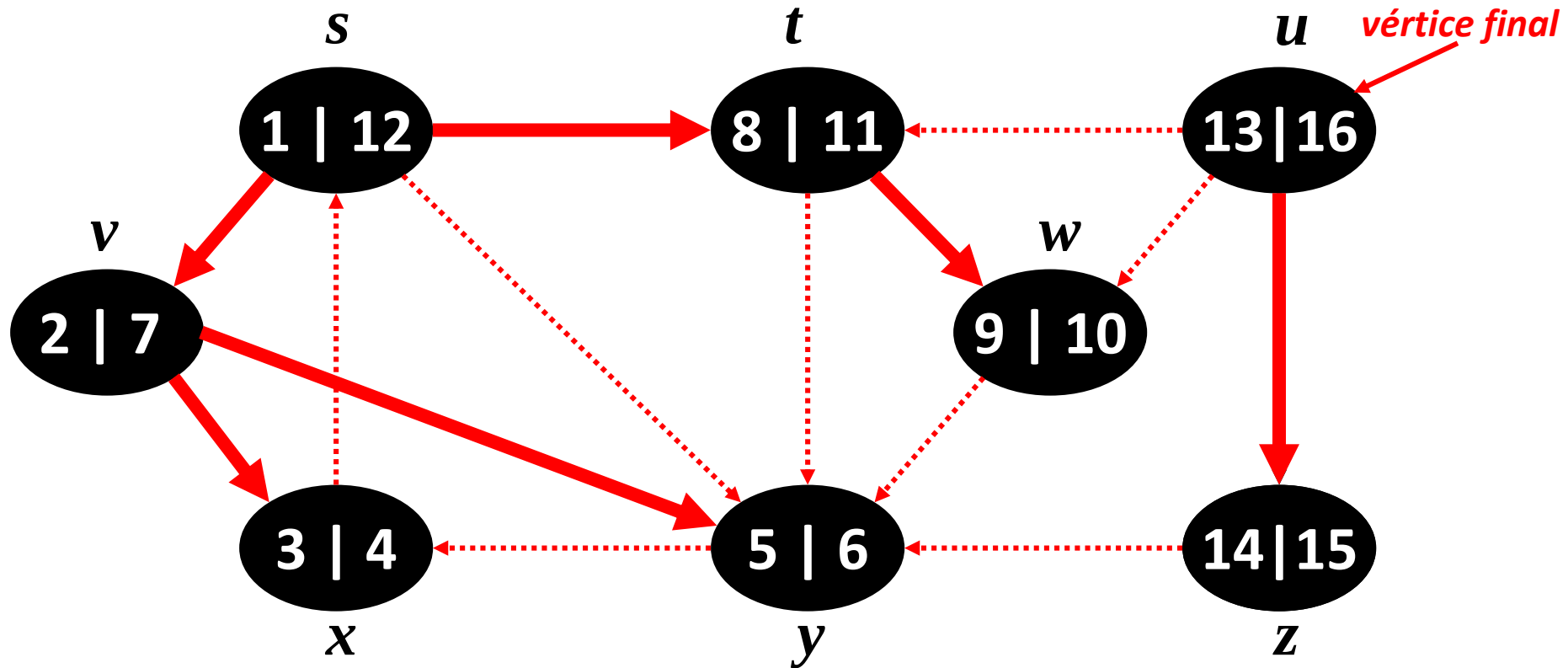
3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



3 - Busca em Profundidade (*Depth-First Search*)

Exemplo DFS:



3 - Busca em Profundidade (Depth-First Search)

3.3 – O Algoritmo DFS:

- Assumir que o grafo $G = (V, E)$ é representado com lista de adjacências.
- Para cada vértice no grafo, o algoritmo mantém estruturas auxiliares:
 - A variável ***cor[v']*** mantém a informação sobre a cor de cada vértice.
 - A variável ***pai[v']*** mantém a informação do predecessor de cada vértice. Quando não existe predecessor ***pai[u] = NIL***.

3 - Busca em Profundidade (Depth-First Search)

3.3 – O Algoritmo DFS:

- Assumir que o grafo $G = (V, E)$ é representado com lista de adjacências.
- Para cada vértice no grafo, o algoritmo mantém estruturas auxiliares:
 - A variável $d[v']$ mantém o valor do tempo quando v' foi visitado pela primeira vez.
 - A variável $F[v']$ mantém o valor do tempo quando v' foi totalmente explorado.
 - O vértice inicial é s .
 - O vértice observado é v' .

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

$\text{tempo} \leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

VisitaDFS(u)

$\text{cor}[v'] \leftarrow \text{CINZA}$

$d[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{pai}[v] \leftarrow v'$

VisitaDFS(v)

$\text{cor}[v'] \leftarrow \text{PRETO}$

$F[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

$\text{tempo} \leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**
 VisitaDFS(u)

VisitaDFS(u)

$\text{cor}[v'] \leftarrow \text{CINZA}$

$d[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{pai}[v] \leftarrow v'$

VisitaDFS(v)

$\text{cor}[v'] \leftarrow \text{PRETO}$

$F[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

VisitaDFS(u)

$\text{cor}[v'] \leftarrow \text{CINZA}$

$d[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{pai}[v] \leftarrow v'$

VisitaDFS(v)

$\text{cor}[v'] \leftarrow \text{PRETO}$

$F[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

VisitaDFS(u)

$\text{cor}[v'] \leftarrow \text{CINZA}$

$d[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{pai}[v] \leftarrow v'$

VisitaDFS(v)

$\text{cor}[v'] \leftarrow \text{PRETO}$

$F[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

3 - Busca em Profundidade (*Depth-First Search*)

3.3 – O Algoritmo DFS:

DFS(G)

for $\forall u \in V[G]$ **do**

$\text{cor}[v'] \leftarrow \text{BRANCO}$

$\text{pai}[v'] \leftarrow \text{NIL}$

tempo $\leftarrow 0$

for $\forall v' \in V[G]$ **do**

if $\text{cor}[v'] = \text{BRANCO}$ **then**

VisitaDFS(u)

VisitaDFS(u)

$\text{cor}[v'] \leftarrow \text{CINZA}$

$d[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

for $\forall v \in \text{Adjacente}[v']$ **do**

if $\text{cor}[v] = \text{BRANCO}$ **then**

$\text{pai}[v] \leftarrow v'$

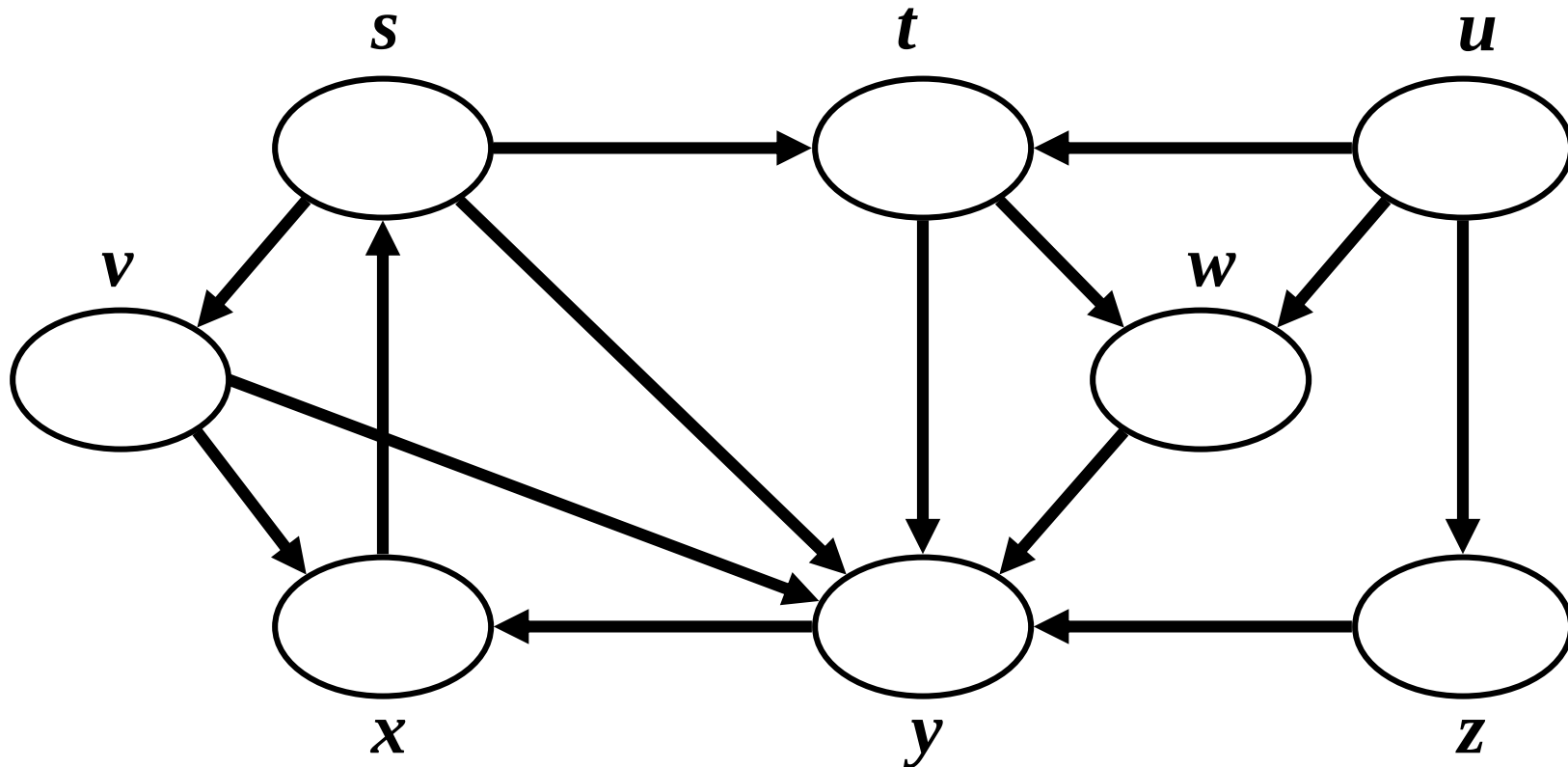
VisitaDFS(v)

$\text{cor}[v'] \leftarrow \text{PRETO}$

$F[v'] \leftarrow \text{tempo} \leftarrow \text{tempo} + 1$

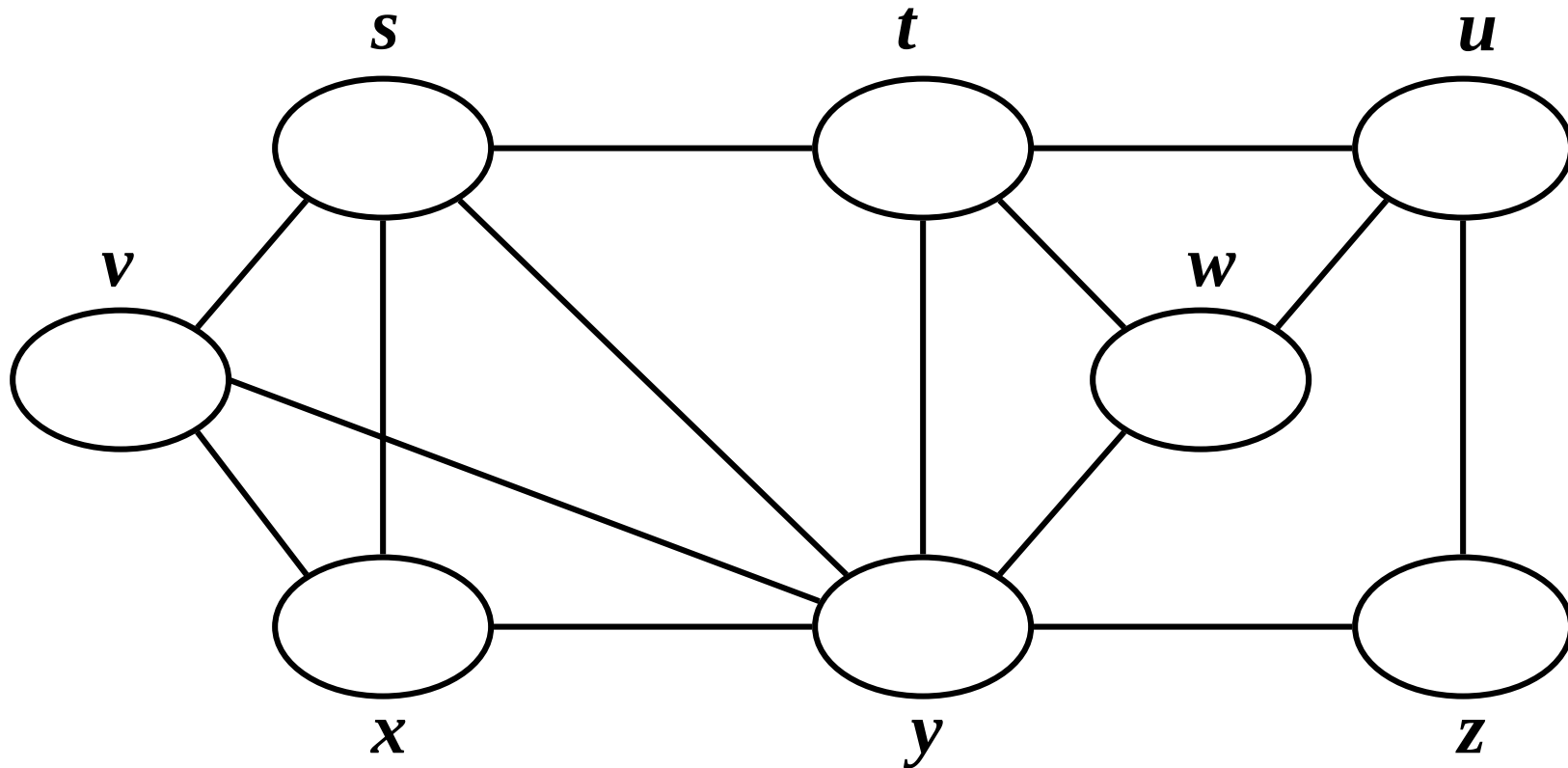
3 - Busca em Profundidade (*Depth-First Search*)

3.4 – Exercícios de DFS: 1) início em *u*:



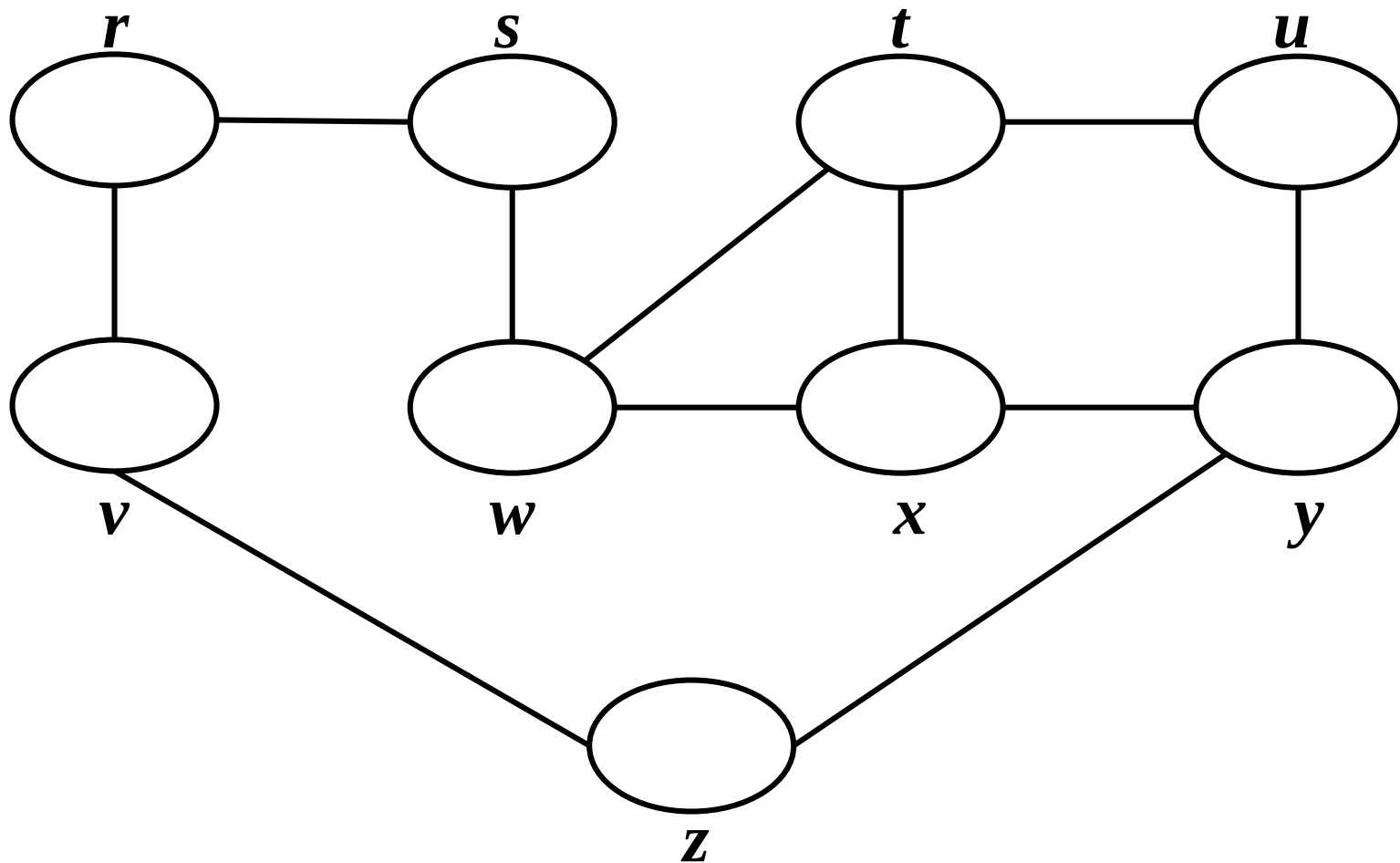
3 - Busca em Profundidade (*Depth-First Search*)

3.4 – Exercícios de DFS: 2) início em x :



3 - Busca em Profundidade (*Depth-First Search*)

3.4 – Exercícios de DFS: 3) início em *x*:



4

Busca em Grafos Ponderados

4 – Busca em Grafos Ponderados

Características da *Greedy Search*:

Um grafo é classificado como sendo ponderado se suas arestas possuem pesos diferentes:

- Uma função de avaliação define o peso das arestas:

$$f(n) = g(n)$$

onde $g(n)$ é o peso da aresta que conecta o nó atual ao nó observado.

- A árvore de busca é criada a partir da escolha da melhor aresta, dado o seu peso calculado pela função de avaliação.
- A qualidade da resolução depende da eficiência da função de avaliação na geração da árvore de busca.

4 – Busca em Grafos Ponderados

Características da *Greedy Search*:

Um algoritmo guloso escolhe o caminho mais "apetitoso" observável localmente, em cada iteração. Este apetite é estabelecido *a priori*.

- Toma decisões com base nas informações disponíveis na iteração corrente, sem prever as consequências e sem explorar outras.
- Faz a escolha ótima para as condições locais, na esperança que tal escolha resulte em um resultado ótimo global.

2 - Algoritmos *Best-First* (*Busca Gulosa*)

Características da *Greedy Search*:

Espera-se, com isso, que uma **escolha ótima local** leve até a **escolha ótima global**. Para alguns problemas, isso funciona bem.

- Jamais se arrepende ou volta atrás: as escolhas que faz em cada iteração são definitivas (sem *backtracking*).
- Algoritmos gulosos são muito rápidos e geralmente eficientes, mas nem sempre apresentam a solução ótima global.

5

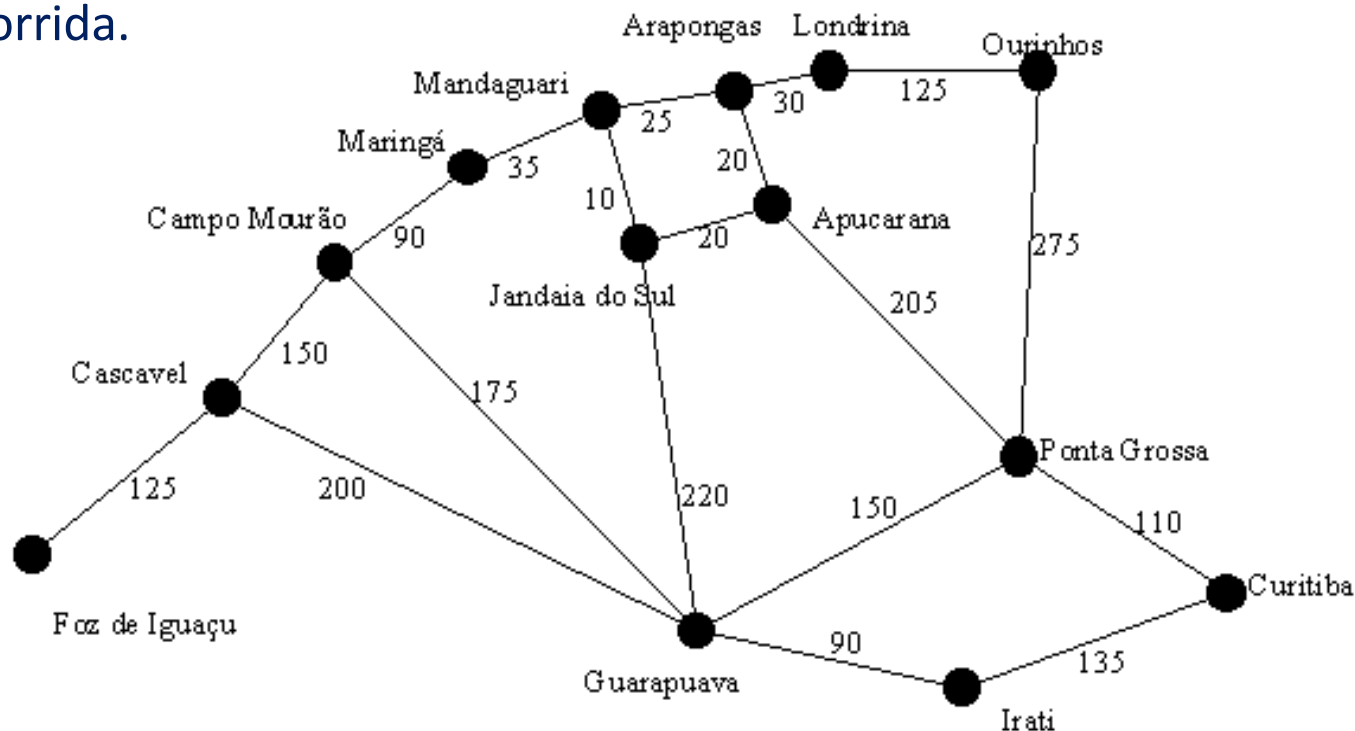
Exercícios

5 - Exercícios

Exercício 5.1:

Dado o **mapa** e a **distância em Km** entre as cidades do Paraná, tendo como objetivo encontrar o caminho de **Curitiba** até **Londrina**, faça:

A árvore de busca completa, baseada em **amplitude**, utilizando um algoritmo guloso com $f(n) = g(n)$ para priorizar os nós de uma mesma geração. Mostre a distância a ser percorrida.

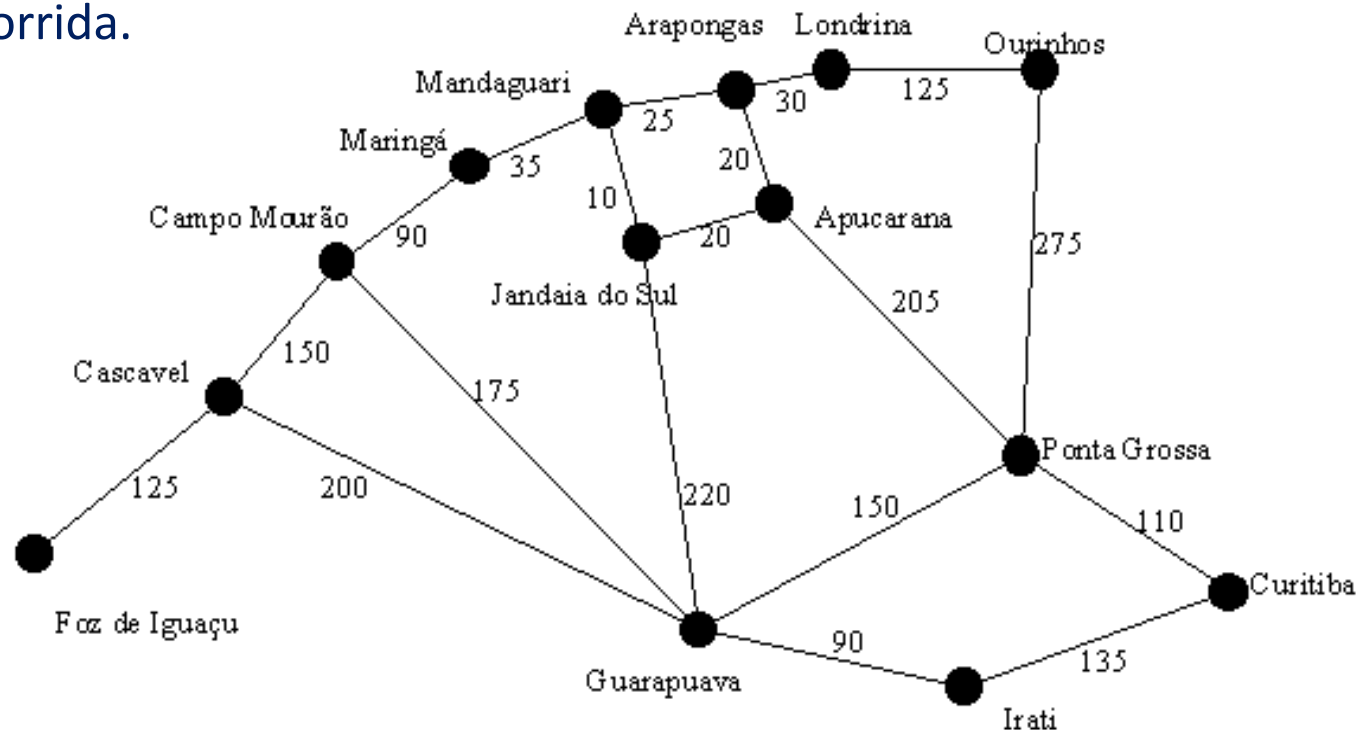


5 - Exercícios

Exercício 5.2:

Dado o **mapa** e a **distância em Km** entre as cidades do Paraná, tendo como objetivo encontrar o caminho de **Curitiba** até **Londrina**, faça:

A árvore de busca completa, baseada em **profundidade**, utilizando um algoritmo guloso com $f(n) = g(n)$ para escolher o próximo nó a ser explorado. Mostre a distância a ser percorrida.

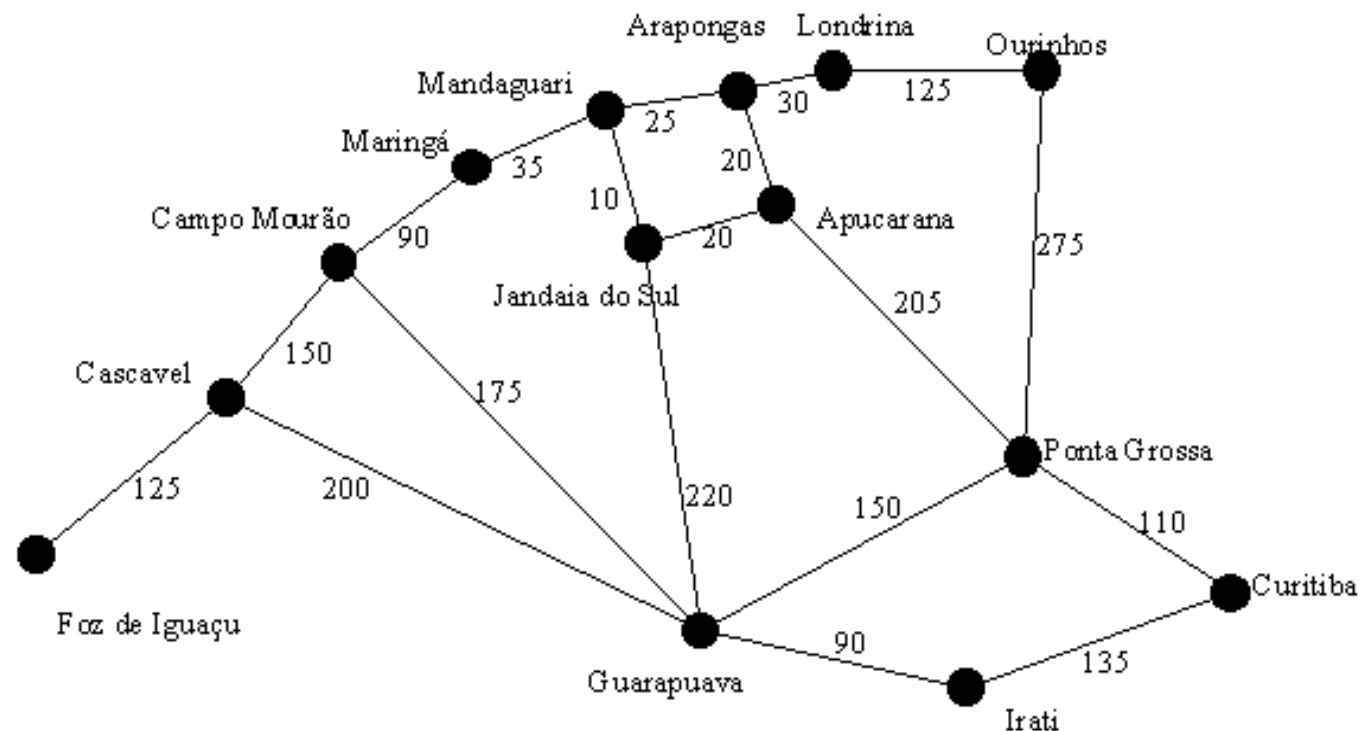


5 - Exercícios

Exercício 5.3:

Dado o **mapa** e a **distância em Km** entre as cidades do Paraná, tendo como objetivo encontrar o caminho de **Curitiba** até **Londrina**, faça:

Qual dos 2 algoritmos anteriores foi o mais eficiente e porquê?



Bibliografias

Obrigatórias:

1. CORMEN, LEISERSON, RIVEST, STEIN, **Algoritmos: Teoria e Prática**. 2ª ed. Rio de Janeiro: Campus, 2002: Capítulo 22 e Parte VIII Apêndice B.4.
2. RUSSELL, Stuart J; NORVIG, Peter. **Inteligência Artificial**. 2ª ed. Rio de Janeiro: Campus, 2004, Capítulos 1 e 2.
3. LUGER, George. **Inteligência Artificial: Estruturas e Estratégias para a Resolução de Problemas Complexos**. 4ª ed. Porto Alegre: Bookman, 2004, Capítulo 1.

Bibliografias

Recomendadas:

1. CORMEN, LEISERSON, RIVEST, STEIN, **Algoritmos: Teoria e Prática**. 2ª ed. Rio de Janeiro: Campus, 2002: Capítulo 3.
2. ARTERO, Almir Olivette, **Inteligência Artificial: Teórica e Prática**. 1ª ed. São Paulo: Livraria da Física, 2009: Capítulo 4.
3. ROSA, João Luis Garcia, **Fundamentos da Inteligência Artificial**. 1ª ed. Rio de Janeiro: LTC, 2011: Capítulo 2.