UNIVERSIDADE FEDERAL DE ITAJUBÁ
Instituto de Matemática e Computação

SMAC03 – Grafos

2. Teoria dos Grafos
Busca

Rafael Frinhani
frinhani@unifei.edu.br

2º Semestre de 2025

OBJETIVOS

Apresentar as definições de busca em grafos, os algoritmos de busca em largura e busca em profundidade e suas aplicações.

AGENDA

2. Teoria dos Grafos

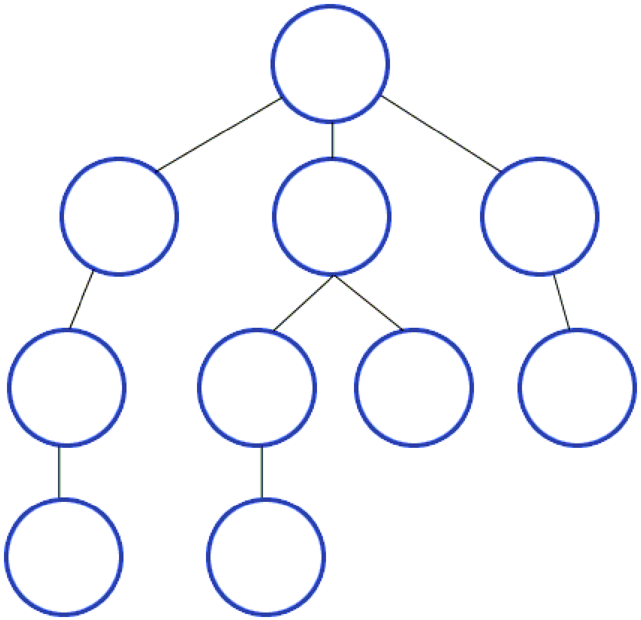
2.7. Busca em grafos

Contextualização, Aplicações, Algoritmo genérico de busca.

Busca em Largura (*Breadth First Search*), conceitos, algoritmo, níveis da busca.

Busca em Profundidade (*Deep First Search*), conceitos, algoritmo, classificação de arestas.

Análise de Complexidade algoritmos BFS e DFS.

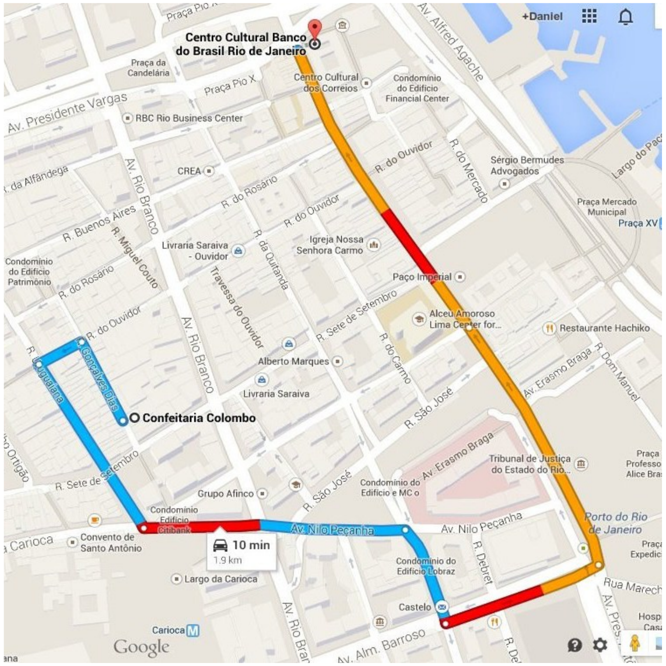


Contextualização

A busca em grafos, ou percurso em grafos, é a tarefa de explorar ou percorrer de forma sistemática seus vértices e arestas para examiná-los.

A exploração pode ser utilizada para diversos propósitos como:

- encontrar um elemento (busca)
- identificar estruturas e propriedades
- descobrir caminhos
- base para métodos mais avançados
- resolver problemas modelados em grafos etc.



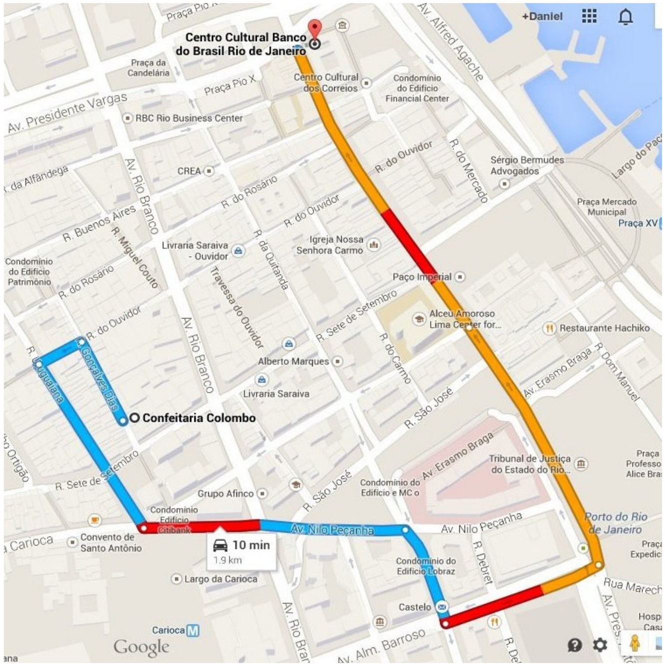
Contextualização

A busca em grafos, ou percurso em grafos, é a tarefa de explorar ou percorrer de forma sistemática seus vértices e arestas para examiná-los.

A exploração pode ser utilizada para diversos propósitos como:

- encontrar um elemento (busca)
- identificar estruturas e propriedades
- descobrir caminhos
- base para métodos mais avançados
- resolver problemas modelados em grafos etc.

A busca em grafo é análoga a tarefa de encontrar um caminho em um labirinto que consiste de passagens (arestas) e suas interseções (vértices). Pode ser mais fácil encontrar a saída via um procedimento sistemático do que aleatoriamente.



Algoritmos de Busca – Aplicações

- Determinar se um grafo é conexo
- Calcular distância entre dois vértices (comprimento do menor caminho)
- Determinar um caminho (problema do labirinto), ou caminho mínimo
- Detectar se o grafo possui ciclos
- Encontrar componentes biconexas, testar se um grafo é bipartido
- Classificar arestas
- Encontrar componentes fortemente conexas
- Obter uma árvore geradora mínima
- *Flood Fill* (inundação)
- Ordenação Topológica
- Casos que requerem uma lista com a ordem que os vértices foram visitados.

Busca Genérica em Grafos

Algoritmos de busca evitam revisitar vértices já explorados utilizando uma marcação:

Não-Visitado: desconhecido ou não-explorado (*white*) corresponde aos vértices ainda não descobertos pelo algoritmo.

Visitado: ou descoberto (*gray*) são aos vértices visitados, mas cujos adjacentes ainda não foram analisados.

Analisado: ou explorado (*black*) são os vértices cujos adjacentes foram todos visitados.

Busca Genérica em Grafos

Algoritmos de busca evitam revisitar vértices já explorados utilizando uma marcação:

Não-Visitado: desconhecido ou não-explorado (*white*) corresponde aos vértices ainda não descobertos pelo algoritmo.

Visitado: ou descoberto (*gray*) são aos vértices visitados, mas cujos adjacentes ainda não foram analisados.

Analisado: ou explorado (*black*) são os vértices cujos adjacentes foram todos visitados.

Algoritmo Genérico de Busca

Passo Inicial
marcar todos os vértices como não-visitados
selecionar vértice origem e marcá-lo como visitado

Passo geral
Enquanto existir vértice visitado
Selecionar algum vértice visitado v_i
Se v_i possuir adjacente não-visitado v_j
marcar v_j como visitado
Senão, marcar v_i como analisado.

Busca Genérica em Grafos

Algoritmos de busca evitam revisitar vértices já explorados utilizando uma marcação:

Não-Visitado: desconhecido ou não-explorado (*white*) corresponde aos vértices ainda não descobertos pelo algoritmo.

Visitado: ou descoberto (*gray*) são aos vértices visitados, mas cujos adjacentes ainda não foram analisados.

Analisado: ou explorado (*black*) são os vértices cujos adjacentes foram todos visitados.

Algoritmo Genérico de Busca

Passo Inicial
marcar todos os vértices como não-visitados
selecionar vértice origem e marcá-lo como visitado

Passo geral
Enquanto existir vértice visitado
Selecionar algum vértice visitado v_i
Se v_i possuir adjacente não-visitado v_j
marcar v_j como visitado
Senão, marcar v_i como analisado.

Problema: <u>algoritmo genérico não estabelece uma ordem de análise</u>, ou seja, não define qual vértice v_j (visitado) deve ser escolhido para ser analisado.

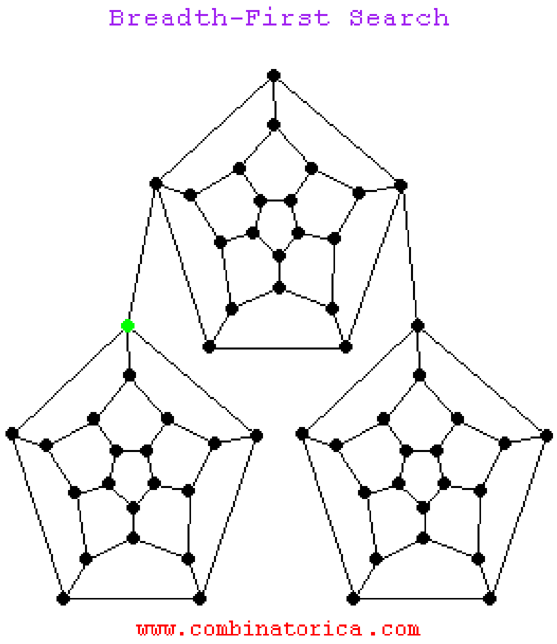
A ordem de análise define uma sistemática de exploração. Técnicas fundamentais:

- v_j é o vértice visitado mais "antigo" (BFS)
- v_j é o vértice visitado mais "recente" (DFS)

Busca em Largura

Breadth First Search (BFS) ou Busca em Nível, a exploração dos vértices progride na largura das conexões, prioriza-se a análise de todos os vizinhos de um vértice origem.

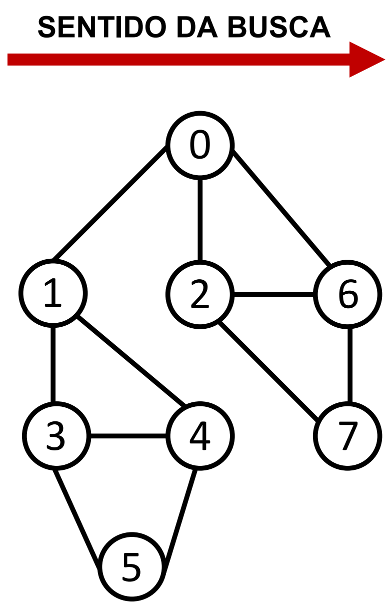
BFS expande de maneira uniforme a fronteira (nível) entre vértices descobertos e não-descobertos. Um vértice é marcado como "analisado" (*black*) quando todos os seus adjacentes forem visitados.



Busca em Largura

Breadth First Search (BFS) ou Busca em Nível, a exploração dos vértices progride na largura das conexões, prioriza-se a análise de todos os vizinhos de um vértice origem.

BFS expande de maneira uniforme a fronteira (nível) entre vértices descobertos e não-descobertos. Um vértice é marcado como "analisado" (*black*) quando todos os seus adjacentes forem visitados.



O algoritmo utiliza uma **fila** para o controle da ordem de análise dos vértices.

São analisados os vértices que foram visitados primeiro (mais antigos, adicionados na fila à mais tempo). Os vértices são geralmente visitados na ordem crescente de seus identificadores.

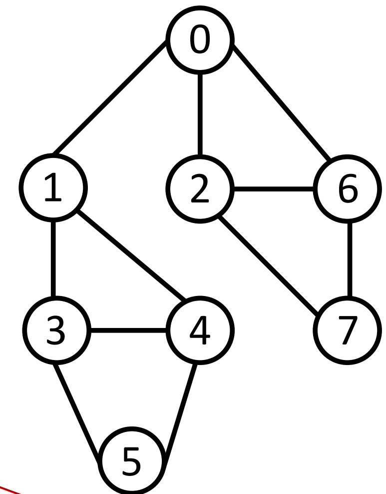
Grafos podem ser não-direcionados ou dígrafos.

Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

Inicialmente todos os vértices são considerados não-visitados (*white*)

Q corresponde a fila (*queue*) dos vértices visitados (*gray*), mas cujos vizinhos ainda não foram inspecionados.



Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]

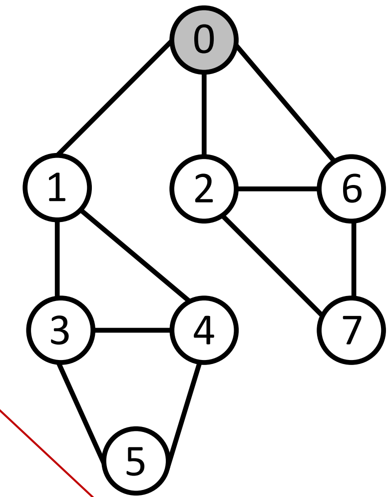
```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

Considerando o vértice de índice 0 como inicial, ele é inserido em Q.



Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]

2.7. Busca em Grafos – Busca em Largura

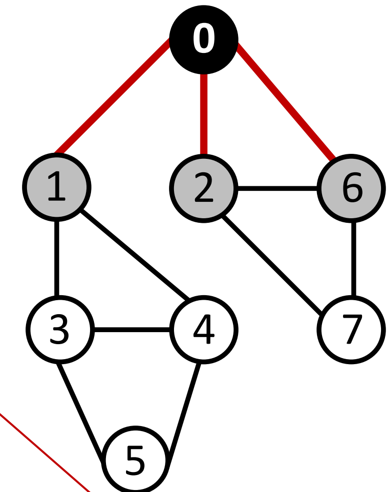
Algoritmo

BFS (G, v)

```
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

Vértice 0 é removido da fila e cada um dos seus adjacentes ainda não pertencentes a Q são inseridos a esta fila pela ordem crescente do índice.



Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[1, 2, 6]

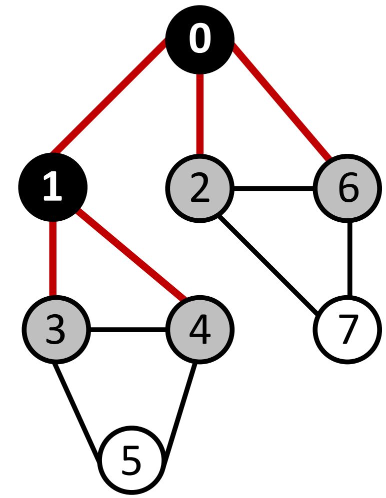
Depois que todos os adjacentes do vértice foram inseridos a Q ele é considerado analisado (preto).

Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

A sequência da busca continua com o vértice 1, que é o primeiro adjacente do vértice 0 que foi inserido à fila Q.



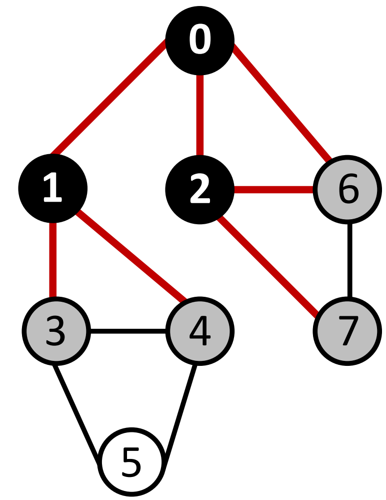
Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[<u>1</u>, 2, 6]
3	0, 1	5, **7**	[<u>2</u>, 6, 3, 4]

Os adjacentes do vértice 1 ainda não pertencentes Q a são inseridos a fila (vértices 3 e 4). Depois das inserções o vértice 1 é considerado analisado.

Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

O progresso da busca considera cada vértice no início da fila Q e a inserção dos seus adjacentes nesta fila quando necessário.




```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

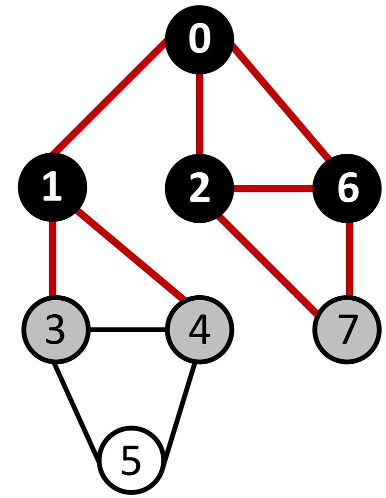
Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[1, 2, 6]
3	0, 1	5, **7**	[2, 6, 3, 4]
4	0, 1, 2	5	[6, 3, 4, 7]

Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

O progresso da busca considera cada vértice no início da fila Q e a inserção dos seus adjacentes nesta fila quando necessário.



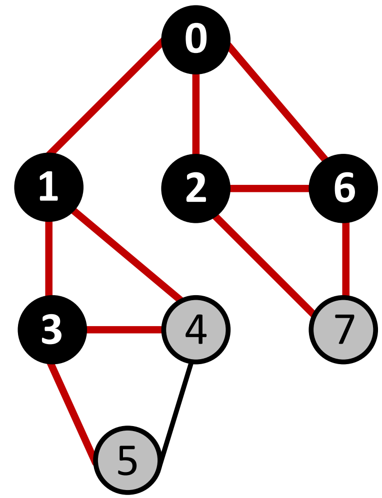
Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[1, 2, 6]
3	0, 1	5, **7**	[2, 6, 3, 4]
4	0, 1, 2	5	[6, 3, 4, 7]
5	0, 1, 2, 6	**5**	[3, 4, 7]

Algoritmo

BFS (G, v)

```
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

O progresso da busca considera cada vértice no início da fila Q e a inserção dos seus adjacentes nesta fila quando necessário.




```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

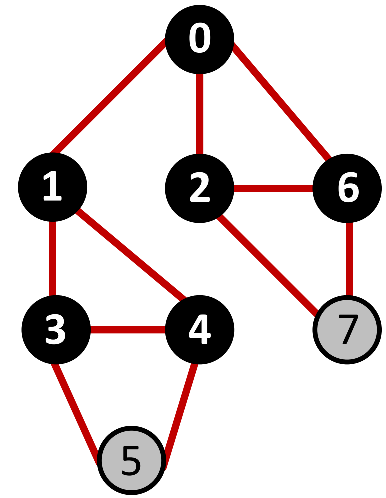
Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[1, 2, 6]
3	0, 1	5, **7**	[2, 6, 3, 4]
4	0, 1, 2	5	[6, 3, 4, 7]
5	0, 1, 2, 6	**5**	[3, 4, 7]
6	0, 1, 2, 6, 3	-	[4, 7, 5]

Algoritmo

BFS (G, v)

```
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

O progresso da busca considera cada vértice no início da fila Q e a inserção dos seus adjacentes nesta fila quando necessário.




```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

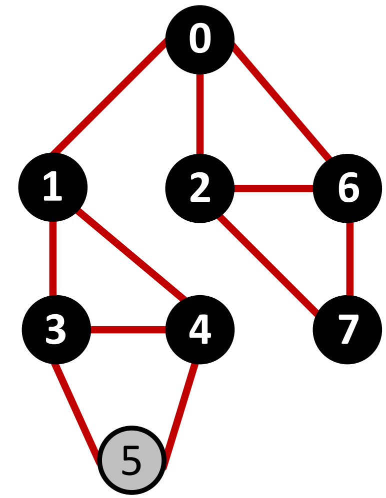
Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[1, 2, 6]
3	0, 1	5, **7**	[2, 6, 3, 4]
4	0, 1, 2	5	[6, 3, 4, 7]
5	0, 1, 2, 6	**5**	[3, 4, 7]
6	0, 1, 2, 6, 3	-	[4, 7, 5]
7	0, 1, 2, 6, 3, 4	-	[7, 5]

Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

O progresso da busca considera cada vértice no início da fila Q e a inserção dos seus adjacentes nesta fila quando necessário.



Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	**1**, **2**, 3, 4, 5, **6**, 7	[0]
2	0	**3**, **4**, 5, 7	[1, 2, 6]
3	0, 1	5, **7**	[2, 6, 3, 4]
4	0, 1, 2	5	[6, 3, 4, 7]
5	0, 1, 2, 6	**5**	[3, 4, 7]
6	0, 1, 2, 6, 3	-	[4, 7, 5]
7	0, 1, 2, 6, 3, 4	-	[7, 5]
8	0, 1, 2, 6, 3, 4, 7	-	[5]

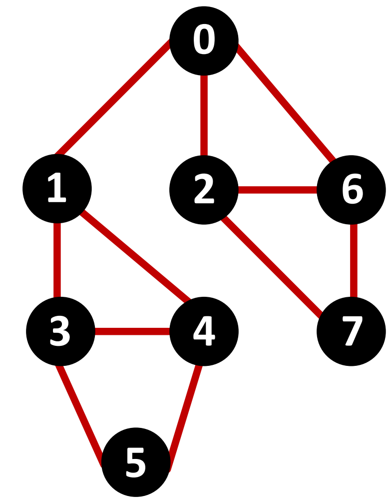
Algoritmo

```
BFS (G, v)
Q = [ ];
insere v em Q;
while Q ≠ ∅ do
    v = remove o primeiro elemento de Q;
    for cada adjacente de v do
        if adjacente ∉ Q then
            insere adjacente em Q;
        end
    end
    marca v como analisado;
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

A sequência de visitas dos vértices na Busca em Largura com o vértice 0 como origem é:

***BFS*(0) = {0, 1, 2, 6, 3, 4, 7, 5}**

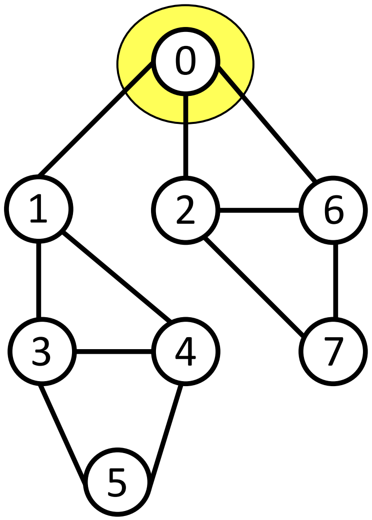


Passo	Analisado	Não-Visitado	Q (Visitado)
0	-	0, 1, 2, 3, 4, 5, 6, 7	[ ]
1	-	1, 2, 3, 4, 5, 6, 7	[0]
2	0	3, 4, 5, 7	[1, 2, 6]
3	0, 1	5, 7	[2, 6, 3, 4]
4	0, 1, 2	5	[6, 3, 4, 7]
5	0, 1, 2, 6	5	[3, 4, 7]
6	0, 1, 2, 6, 3	-	[4, 7, 5]
7	0, 1, 2, 6, 3, 4	-	[7, 5]
8	0, 1, 2, 6, 3, 4, 7	-	[5]
9	0, 1, 2, 6, 3, 4, 7, 5	-	[ ]

Níveis da Busca

A progressão da BFS é análoga a de uma onda que é propagada a partir do vértice origem.

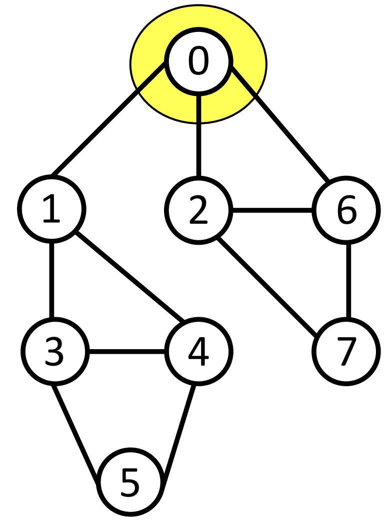
A onda expande em círculos que determinam níveis dos vizinhos a serem analisados em relação a origem.



Níveis da Busca

A progressão da BFS é análoga a de uma onda que é propagada a partir do vértice origem.

A onda expande em círculos que determinam níveis dos vizinhos a serem analisados em relação a origem.



NÍVEIS

L_0 : 0

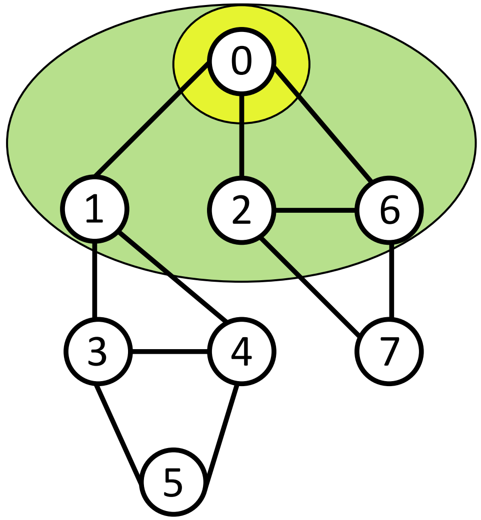
NÍVEIS (camadas)

- L_i : conjunto de vértices pertencentes ao nível i = {0, 1, 2, ... n}
- L_0 : vértice origem
- L_{i+1} : conjunto de vértices que não fazem parte de um nível anterior e que possuem uma aresta com algum vértice do nível L_i

Níveis da Busca

A progressão da BFS é análoga a de uma onda que é propagada a partir do vértice origem.

A onda expande em círculos que determinam níveis dos vizinhos a serem analisados em relação a origem.



NÍVEIS

L_0 : 0

L_1 : 1, 2, 6

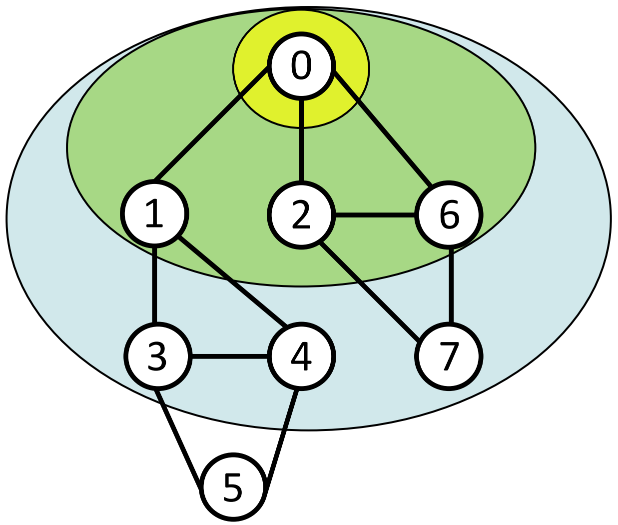
NÍVEIS (camadas)

- L_i : conjunto de vértices pertencentes ao nível i = {0, 1, 2, ... n}
- L_0 : vértice origem
- L_{i+1} : conjunto de vértices que não fazem parte de um nível anterior e que possuem uma aresta com algum vértice do nível L_i

Níveis da Busca

A progressão da BFS é análoga a de uma onda que é propagada a partir do vértice origem.

A onda expande em círculos que determinam níveis dos vizinhos a serem analisados em relação a origem.



NÍVEIS (camadas)

- L_i : conjunto de vértices pertencentes ao nível i = {0, 1, 2, ... n}
- L_0 : vértice origem
- L_{i+1} : conjunto de vértices que não fazem parte de um nível anterior e que possuem uma aresta com algum vértice do nível L_i

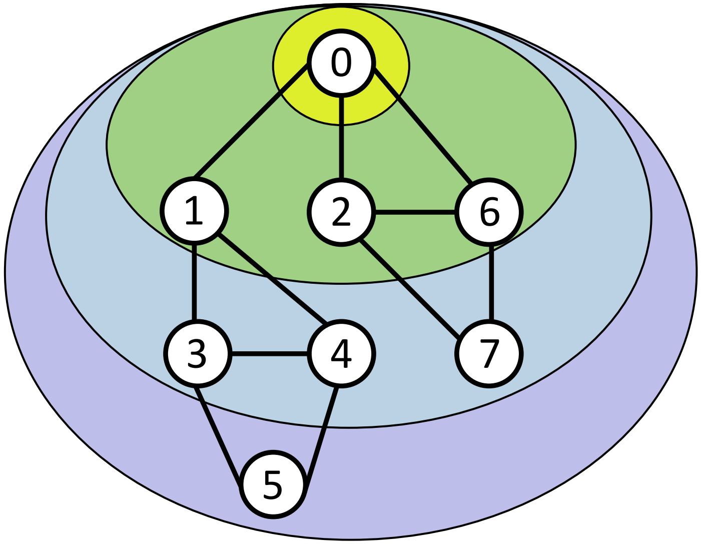
NÍVEIS

L_0 : 0
L_1 : 1, 2, 6
L_2 : 3, 4, 7

Níveis da Busca

A progressão da BFS é análoga a de uma onda que é propagada a partir do vértice origem.

A onda expande em círculos que determinam níveis dos vizinhos a serem analisados em relação a origem.



NÍVEIS (camadas)

- L_i : conjunto de vértices pertencentes ao nível $i = \{0, 1, 2, \ldots n\}$
- L_0 : vértice origem
- L_{i+1} : conjunto de vértices que não fazem parte de um nível anterior e que possuem uma aresta com algum vértice do nível L_i

Distância é o comprimento de menor caminho entre dois vértices.

NÍVEIS

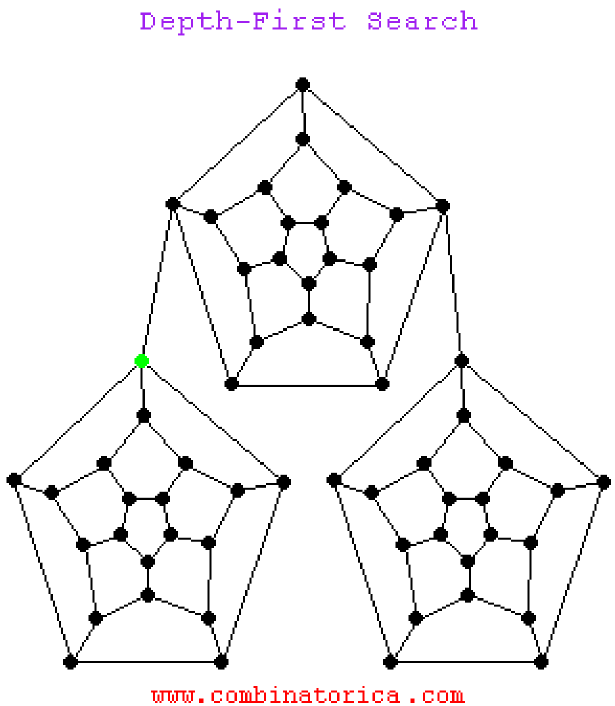
L_0 : 0
L_1 : 1, 2, 6
L_2 : 3, 4, 7
L_3 : 5

Vértices pertencentes a camada L_i têm distância i da origem.

Busca em Profundidade

Ou Deep First Search (DFS), a exploração dos vértices progride na profundidade (extensão) das conexões. A estratégia é buscar o mais profundo possível no grafo, voltando no percurso quando não existem mais vértices não-visitados pela frente.

A partir do vértice origem são explorados os adjacentes descobertos mais recentemente (mais novos). Não analisa todos os vértices de um nível antes de seguir para o próximo nível.



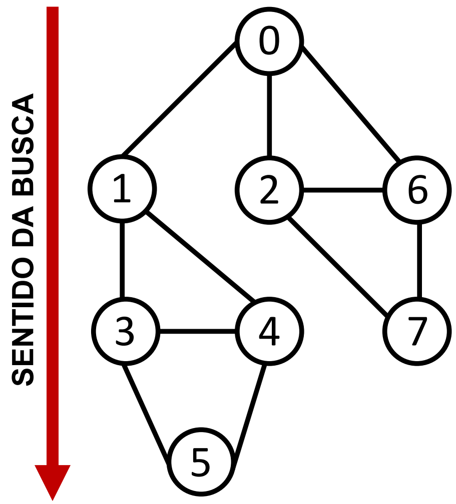
Busca em Profundidade

Ou Deep First Search (DFS), a exploração dos vértices progride na profundidade (extensão) das conexões. A estratégia é buscar o mais profundo no grafo possível, voltando no percurso quando não existem mais vértices não-visitados pela frente.

A partir do vértice origem são explorados os adjacentes descobertos mais recentemente (mais novos). Não analisa todos os vértices de um nível antes de seguir para o próximo nível.

Uma pilha (explícita ou recursiva) controla a ordem de análise dos vértices, os quais geralmente são visitados na ordem crescente de seus ids.

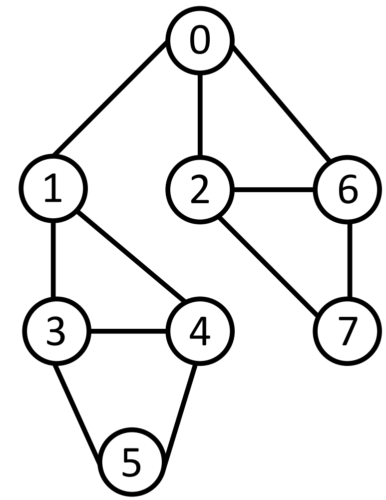
Na DFS um vértice é marcado como visitado antes que toda sua vizinhança seja visitada. Grafos podem ser não-direcionados ou dígrafos.



Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

Inicialmente todos os vértices são considerados não-visitados (*white*)




```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

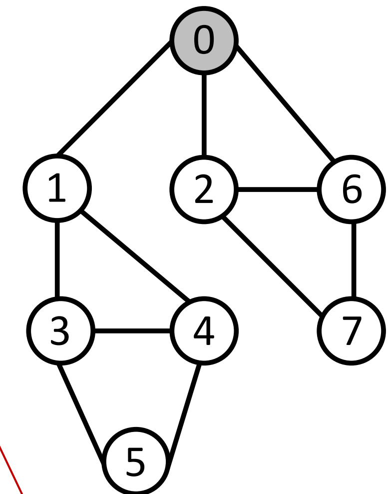
<table>
<tr><th>Passo</th><th>Visitado</th><th>Não-Visitado</th><th>Pilha</th></tr>
<tr><td>0</td><td>-</td><td>0, 1, 2, 3, 4, 5, 6, 7</td><td>| |</td></tr>
</table>

Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

Inicialmente todos os vértices são considerados não-visitados (*white*)



O algoritmo inicia empilhando o vértice dado como origem (para o caso o de índice 0).

Uma vez na pilha o vértice é considerado visitado (*gray*).

Passo	Visitado	Não-Visitado	Pilha
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |**1**, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

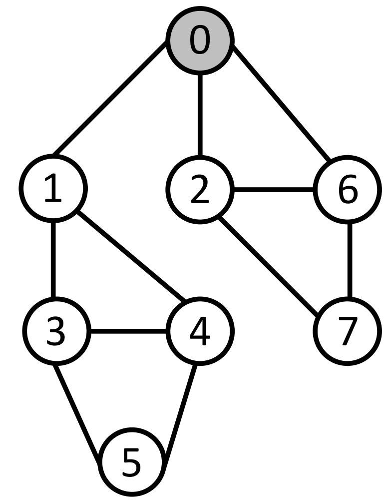
Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

Inicialmente todos os vértices são considerados não-visitados (*white*)

O algoritmo inicia empilhando o vértice dado como origem (para o caso o de índice 0).

Uma vez na pilha o vértice é considerado visitado (*gray*).




```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

Após o empilhamento do vértice 0 é selecionado um adjacente não-visitado conforme a ordem crescente do índice (vértice 1).

Passo	Visitado	Não-Visitado	Pilha
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |**1**, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

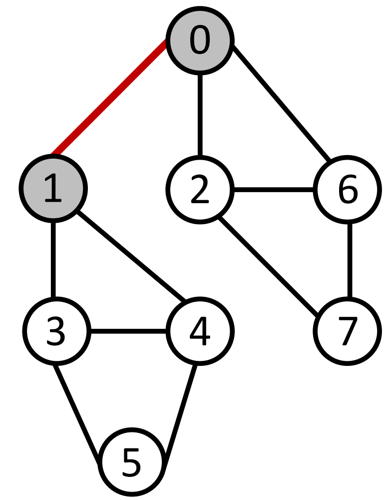
Algoritmo – Recursivo

DFS (G, v)

```
1 marca v como visitado;
2 for cada adjacente de v do
3     if adjacente não-visitado then
4         DFS(G, v);
5     end
6 end
```

O vértice 1 é dado como entrada para a função recursiva (linha 4), sendo empilhado e marcado como visitado.

A busca continua com o vértice adjacente que ainda não foi visitado (vértice 3).



0 : [1, 2, 6]
1 : [~~0~~, $\boxed{3}$, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]

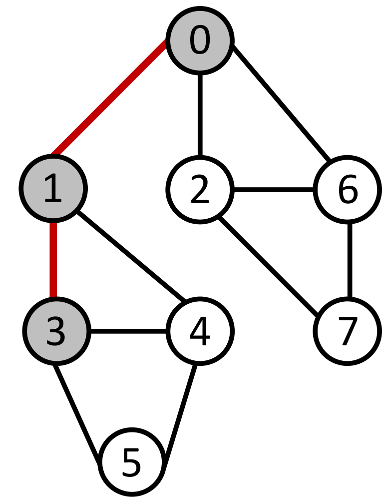
Passo	Visitado	Não-Visitado	Pilha
2	0, 1	2, 3, 4, 5, 6, 7	*DFS* (1) = [~~0~~, **3**, 4]
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |1, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

```
DFS (G, v)
1 marca v como visitado;
2 for cada adjacente de v do
3     if adjacente não-visitado then
4         DFS(G, v);
5     end
6 end
```

O vértice 1 é dado como entrada para a função recursiva (linha 4), sendo empilhado e marcado como visitado.

A busca continua com o vértice adjacente que ainda não foi visitado (vértice 3).




```
0 : [1, 2, 6]
1 : [~~0~~, 3, 4]
2 : [0, 6, 7]
3 : [~~1~~, [4], 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

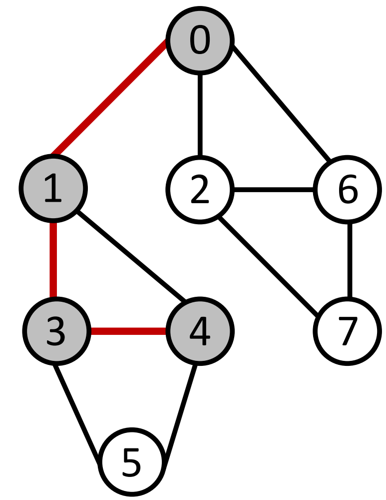
Vértice 3 é dado como entrada para a função recursiva DFS, sendo selecionado o próximo adjacente disponível (vértice 4).

Passo	Visitado	Não-Visitado	Pilha
3	0, 1, 3	2, 4, 5, 6, 7	*DFS* (3) = |~~1~~, 4, 5|
2	0, 1	2, 3, 4, 5, 6, 7	*DFS* (1) = [~~0~~, 3, 4]
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |1, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

A busca continua com o empilhamento do vértice mais recente enquanto existir adjacentes ainda não-visitados.



0 : [1, 2, 6]
1 : [~~0~~, 3, 4]
2 : [0, 6, 7]
3 : [~~1~~, 4, 5]
4 : [~~1~~, ~~3~~, $\boxed{5}$]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]

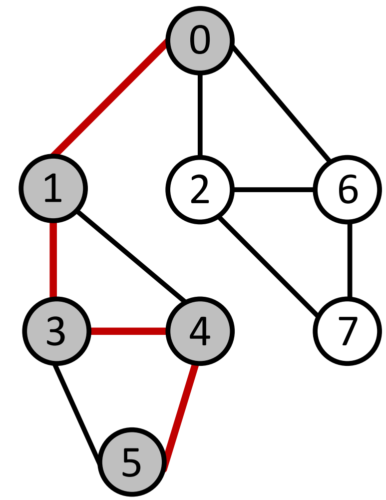
Passo	Visitado	Não-Visitado	Pilha
4	0, 1, 3, 4	2, 5, 6, 7	*DFS* (4) = |~~1~~, ~~3~~, 5|
3	0, 1, 3	2, 4, 5, 6, 7	*DFS* (3) = |~~1~~, 4, 5|
2	0, 1	2, 3, 4, 5, 6, 7	*DFS* (1) = [~~0~~, 3, 4]
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |1, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

A busca continua com o empilhamento do vértice mais recente enquanto existir adjacentes ainda não-visitados.

No vértice 5 não existem mais adjacentes não-visitados, iniciando o processo de desempilhamento (*backtracking*).




```
0 : [1, 2, 6]
1 : [~~0~~, 3, 4]
2 : [0, 6, 7]
3 : [~~1~~, 4, 5]
4 : [~~1~~, ~~3~~, 5]
5 : [~~3~~, ~~4~~]
6 : [0, 2, 7]
7 : [2, 6]
```

Passo	Visitado	Não-Visitado	Pilha
5	0, 1, 3, 4, 5	2, 6, 7	*DFS* (5) = |~~3~~, ~~4~~|
4	0, 1, 3, 4	2, 5, 6, 7	*DFS* (4) = |~~1~~, ~~3~~, 5|
3	0, 1, 3	2, 4, 5, 6, 7	*DFS* (3) = |~~1~~, 4, 5|
2	0, 1	2, 3, 4, 5, 6, 7	*DFS* (1) = [~~0~~, 3, 4]
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |1, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

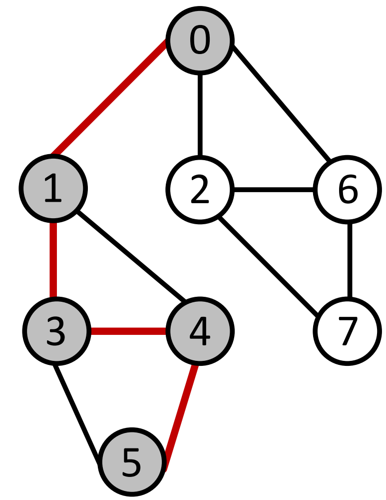
Algoritmo – Recursivo

DFS (G, v)

```
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

A busca continua com o empilhamento do vértice mais recente enquanto existir adjacentes ainda não-visitados.

No vértice 5 não existem mais adjacentes não-visitados, iniciando o processo de desempilhamento (*backtracking*).



0 : [1, 2, 6]
1 : [~~0~~, 3, 4]
2 : [0, 6, 7]
3 : [~~1~~, 4, 5]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [0, 2, 7]
7 : [2, 6]

Os vértices **4**, 3 e 1 (desempilhados nesta ordem) já tem todos seus adjacentes visitados.

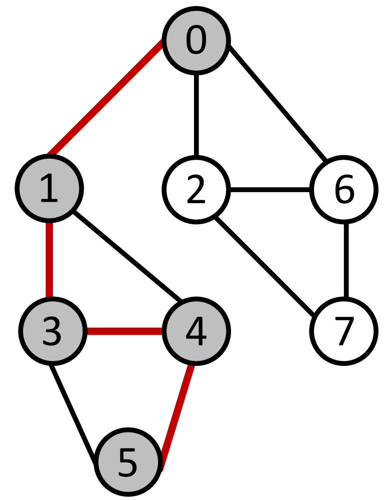
<table>
<tr><th>Passo</th><th>Visitado</th><th>Não-Visitado</th><th>Pilha</th></tr>
<tr><td>6</td><td>0, 1, 3, 4, 5</td><td>2, 6, 7</td><td>DFS (4) = |~~1~~, ~~3~~, ~~5~~|</td></tr>
<tr><td>3</td><td>0, 1, 3</td><td>2, 4, 5, 6, 7</td><td>DFS (3) = |~~1~~, 4, 5|</td></tr>
<tr><td>2</td><td>0, 1</td><td>2, 3, 4, 5, 6, 7</td><td>DFS (1) = [~~0~~, 3, 4]</td></tr>
<tr><td>1</td><td>0</td><td>1, 2, 3, 4, 5, 6, 7</td><td>DFS (0) = |1, 2, 6|</td></tr>
<tr><td>0</td><td>-</td><td>0, 1, 2, 3, 4, 5, 6, 7</td><td>| |</td></tr>
</table>

Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

A busca continua com o empilhamento do vértice mais recente enquanto existir adjacentes ainda não-visitados.

No vértice 5 não existem mais adjacentes não-visitados, iniciando o processo de desempilhamento (*backtracking*).




```
0 : [1, 2, 6]
1 : [0, 3, 4]
2 : [0, 6, 7]
3 : [1, 4, 5]
4 : [1, 3, 5]
5 : [3, 4]
6 : [0, 2, 7]
7 : [2, 6]
```

Os vértices 4, **3** e 1 (desempilhados nesta ordem) já tem todos seus adjacentes visitados.

Passo	Visitado	Não-Visitado	Pilha
7	0, 1, 3, 4, 5	2, 6, 7	*DFS* (3) = |~~1~~, ~~4~~, ~~5~~|
2	0, 1	2, 3, 4, 5, 6, 7	*DFS* (1) = [~~0~~, 3, 4]
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |1, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

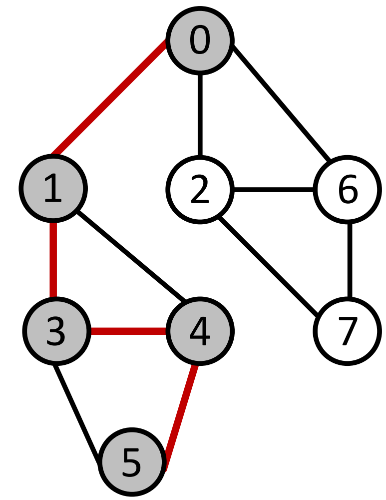
Algoritmo – Recursivo

DFS (G, v)

```
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

A busca continua com o empilhamento do vértice mais recente enquanto existir adjacentes ainda não-visitados.

No vértice 5 não existem mais adjacentes não-visitados, iniciando o processo de desempilhamento (*backtracking*).



0 : [1, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [0, 6, 7]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [0, 2, 7]
7 : [2, 6]

Os vértices 4, 3 e **1** (desempilhados nesta ordem) já tem todos seus adjacentes visitados.

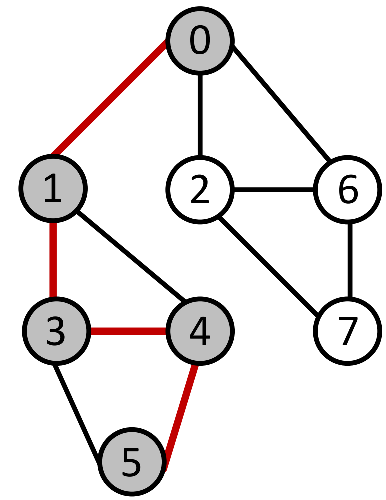
Passo	Visitado	Não-Visitado	Pilha
8	0, 1, 3, 4, 5	2, 6, 7	*DFS* (1) = [~~0~~, ~~3~~, ~~4~~]
1	0	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |1, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

DFS (G, v)

marca *v* como *visitado*;
for *cada adjacente de v* **do**
    if *adjacente não-visitado* **then**
        DFS(*G*, *v*);
    end
end

Quando a busca retorna a origem (vértice 0) é identificado um adjacente ainda não-visitado (vértice 2).



0 : [~~1~~, $\boxed{2}$, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [0, 6, 7]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [0, 2, 7]
7 : [2, 6]

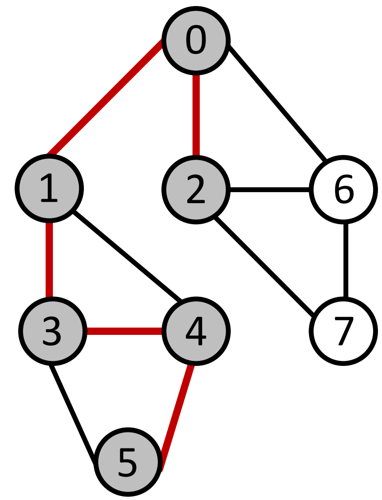
Passo	Visitado	Não-Visitado	Pilha
9	0, 1, 3, 4, 5	2, 6, 7	*DFS* (0) = |~~1~~, **2**, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

```
DFS (G, v)
marca v como visitado;
for cada adjacente de v do
    if adjacente não-visitado then
        DFS(G, v);
    end
end
```

Quando a busca retorna a origem (vértice 0) é identificado um adjacente ainda não-visitado (vértice 2).

O processo de empilhamento recomeça a partir do vértice 2.



0 : [~~1~~, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [0, 6, 7]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [0, 2, 7]
7 : [2, 6]

Passo	Visitado	Não-Visitado	Pilha
10	0, 1, 3, 4, 5, 2	6, 7	*DFS* (2) = |~~0~~, 6, 7|
9	0, 1, 3, 4, 5	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |~~1~~, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

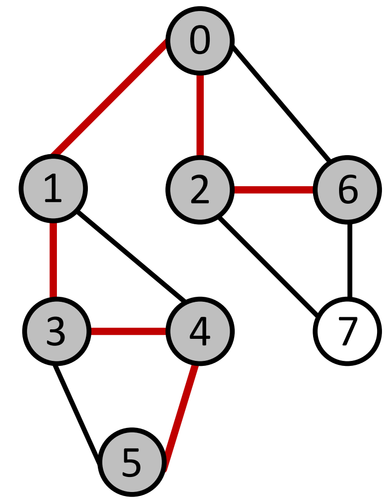
Algoritmo – Recursivo

DFS (G, v)

marca *v* como *visitado*;
for *cada adjacente de v* **do**
  if *adjacente não-visitado* **then**
    DFS(*G*, *v*);
  end
end

Quando a busca retorna a origem (vértice 0) é identificado um adjacente ainda não-visitado (vértice 2).

O processo de empilhamento recomeça a partir do vértice 2.



Continua com o vértice 6, sendo selecionado o vértice 7 que é o único adjacente ainda não visitado.

0 : [~~1~~, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [~~0~~, ~~6~~, ~~7~~]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [~~0~~, ~~2~~, 7]
7 : [2, 6]

Passo	Visitado	Não-Visitado	Pilha
11	0, 1, 3, 4, 5, 6	7	*DFS* (6) = |~~0~~, ~~2~~, 7|
10	0, 1, 3, 4, 5, 2	6, 7	*DFS* (2) = |~~0~~, 6, 7|
9	0, 1, 3, 4, 5	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |~~1~~, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

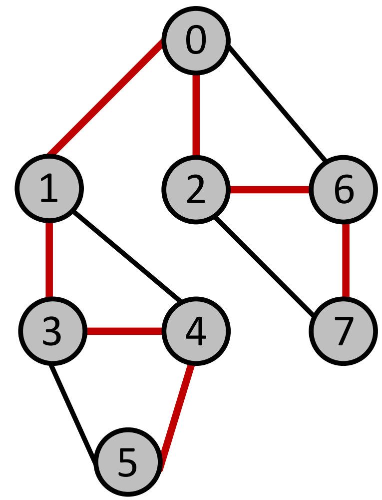
DFS (G, v)

marca *v* como *visitado*;
for *cada adjacente de v* **do**
    if *adjacente não-visitado* **then**
        DFS(*G*, *v*);
    end
end

Quando a busca retorna a origem (vértice 0) é identificado um adjacente ainda não-visitado (vértice 2).

O processo de empilhamento recomeça a partir do vértice 2.

Continua com o vértice 6, sendo selecionado o vértice 7 que é o único adjacente ainda não visitado.



0 : [~~1~~, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [~~0~~, ~~6~~, ~~7~~]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [~~0~~, ~~2~~, 7]
7 : [~~2~~, ~~6~~]

No vértice 7 constata-se que todos seus adjacentes já foram visitados, iniciando o processo de desempilhamento.

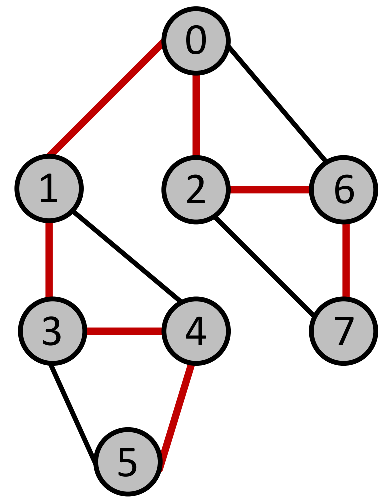
Passo	Visitado	Não-Visitado	Pilha
12	0, 1, 3, 4, 5, 2, 6, 7	-	*DFS* (7) = |~~2~~, ~~6~~|
11	0, 1, 3, 4, 5, 2, 6	7	*DFS* (6) = |~~0~~, ~~2~~, 7|
10	0, 1, 3, 4, 5, 2	6, 7	*DFS* (2) = |~~0~~, 6, 7|
9	0, 1, 3, 4, 5	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |~~1~~, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

DFS (G, v)

marca *v* como *visitado*;
for *cada adjacente de v* **do**
  if *adjacente não-visitado* **then**
    DFS(*G*, *v*);
  end
end

Vértices **6** e 2 são desempilhados, pois todos seus adjacentes já foram analisados.



0 : [~~1~~, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [~~0~~, ~~6~~, ~~7~~]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [~~0~~, ~~2~~, ~~7~~]
7 : [~~2~~, ~~6~~]

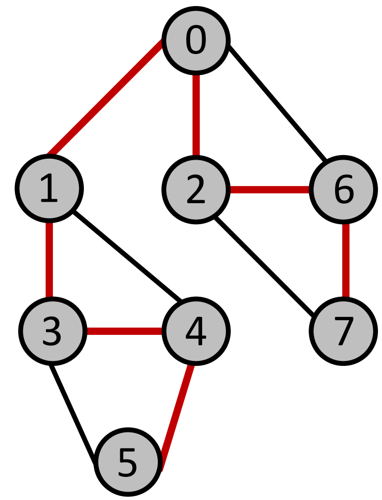
Passo	Visitado	Não-Visitado	Pilha
13	0, 1, 3, 4, 5, 2, 6, 7	-	*DFS* (6) = |~~0~~, ~~2~~, ~~7~~|
10	0, 1, 3, 4, 5, 2	6, 7	*DFS* (2) = |~~0~~, 6, 7|
9	0, 1, 3, 4, 5	1, 2, 3, 4, 5, 6, 7	*DFS* (0) = |~~1~~, 2, 6|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

Algoritmo – Recursivo

DFS (G, v)

marca *v* como *visitado*;
for *cada adjacente de v* **do**
  if *adjacente não-visitado* **then**
    DFS(*G*, *v*);
  end
end

Vértices 6 e **2** são desempilhados, pois todos seus adjacentes já foram analisados.



0 : [~~1~~, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [~~0~~, ~~6~~, ~~7~~]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [~~0~~, ~~2~~, ~~7~~]
7 : [~~2~~, ~~6~~]

<table>
<tr><th>Passo</th><th>Visitado</th><th>Não-Visitado</th><th>Pilha</th></tr>
<tr><td>14</td><td>0, 1, 3, 4, 5, 2, 6, 7</td><td>-</td><td>DFS (2) = |~~0~~, ~~6~~, ~~7~~|</td></tr>
<tr><td>9</td><td>0, 1, 3, 4, 5</td><td>1, 2, 3, 4, 5, 6, 7</td><td>DFS (0) = |~~1~~, 2, 6|</td></tr>
<tr><td>0</td><td>-</td><td>0, 1, 2, 3, 4, 5, 6, 7</td><td>| |</td></tr>
</table>

Algoritmo – Recursivo

DFS (G, v)

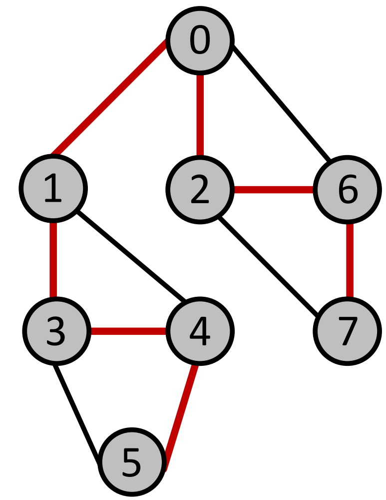
marca *v* como *visitado*;
for *cada adjacente de v* **do**
  if *adjacente não-visitado* **then**
    DFS(*G*, *v*);
  end
end

0 : [~~1~~, 2, 6]
1 : [~~0~~, ~~3~~, ~~4~~]
2 : [~~0~~, ~~6~~, ~~7~~]
3 : [~~1~~, ~~4~~, ~~5~~]
4 : [~~1~~, ~~3~~, ~~5~~]
5 : [~~3~~, ~~4~~]
6 : [~~0~~, ~~2~~, ~~7~~]
7 : [~~2~~, ~~6~~]

Vértices 6 e **2** são desempilhados, pois todos seus adjacentes já foram analisados.

A busca retorna ao vértice origem.

Como todos os adjacentes da origem foram visitados e a pilha está vazia, a busca é finalizada.



A sequência de visitas dos vértices na Busca em Profundidade considerando o vértice 0 como origem é:

***DFS*(0) = {0, 1, 3, 4, 5, 2, 6, 7}**

Passo	Visitado	Não-Visitado	Pilha
15	0, 1, 3, 4, 5, 2, 6, 7	-	*DFS* (0) = |~~1~~,~~2~~, ~~6~~|
0	-	0, 1, 2, 3, 4, 5, 6, 7	| |

2.7. Busca em Grafos – Busca em Profundidade

Algoritmo – Interativo

```
DFS(listaAdj, v)
E ← |listaAdj|;
idsVertices ← [0, 1, ..., E − 1];
sequencia ← [ ];
P ← [v];
while idsVertices ≠ ∅ do
    while P ≠ ∅ do
        t ← P[topo];
        if t ∉ sequencia then
            sequencia ← sequencia + t;
        adjViaveis ← listaAdj[t] \ sequencia;
        if adjViaveis ≠ ∅ then
            P ← adjViaveis[0]
        else
            P.remove(topo);
    idsVertices ← idsVertices \ sequencia;
    if idsVertices ≠ ∅ then
        P ← idsVertices[0];
return sequencia;
```

Cria lista com ids de todos os vértices para garantir que todos os vértices do grafo sejam sequenciados.

Armazena a sequência de vértices conforme a visita pela estratégia da DFS.

Adiciona o vértice v origem (inicial) na pilha P.

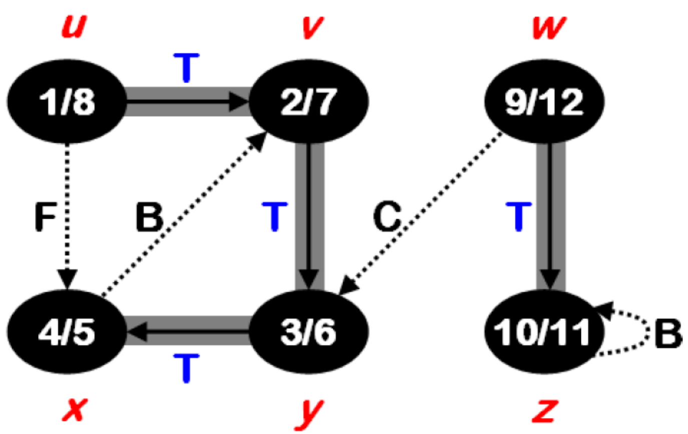
Id do vértice no topo de P.

Adjacentes de t excluídos os que já estão sequência

Adiciona apenas o primeiro adjacente viável de t na pilha P.

Classificação de Arestas

A DFS pode ser usada para classificar arestas de um grafo (ex. Um digrafo é acíclico sse não possui nenhuma aresta de retorno). **Tipos de Arestas**:

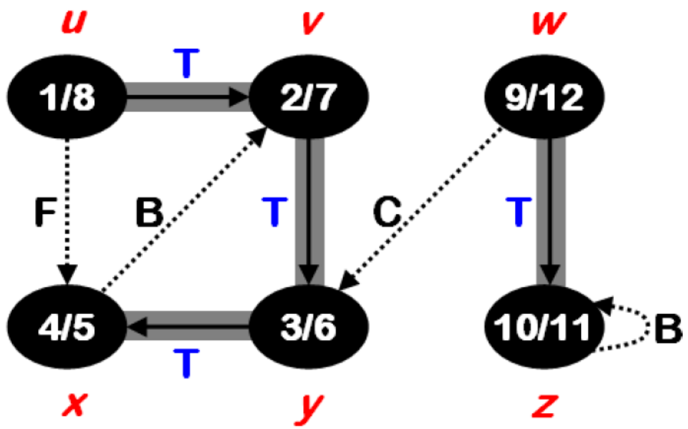


Árvore (*Tree*): A aresta (u, v) é do tipo árvore se v foi descoberta pela primeira vez ao percorrer o grafo.

Retorno (*Back*): conectam um vértice u com um antecessor v em uma árvore DFS (inclui *loops*). Liga um vértice a um antecessor na árvore.

Classificação de Arestas

A DFS pode ser usada para classificar arestas de um grafo (ex. Um digrafo é acíclico sse não possui nenhuma aresta de retorno). **Tipos de Arestas**:



Árvore (*Tree*): A aresta (u, v) é do tipo árvore se v foi descoberta pela primeira vez ao percorrer o grafo.

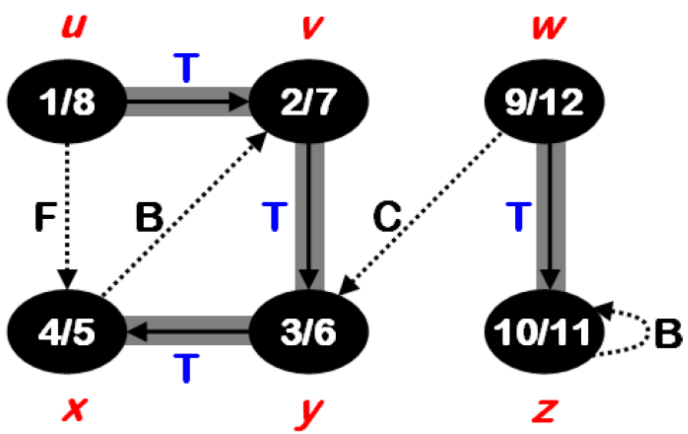
Retorno (*Back*): conectam um vértice u com um antecessor v em uma árvore DFS (inclui *loops*). Liga um vértice a um antecessor na árvore.

Avanço (*Forward*): não pertencem à árvore da DFS mas conectam um vértice a um descendente que pertence à árvore.

Cruzamento (*Cross*): podem conectar vértices na mesma árvore DFS, ou em duas árvores diferentes.

Classificação de Arestas

A DFS pode ser usada para classificar arestas de um grafo (ex. Um digrafo é acíclico sse não possui nenhuma aresta de retorno). **Tipos de Arestas**:



- Tempo de Descoberta: Dado por $td[u]$, corresponde ao momento em que o vértice u foi visitado pela primeira vez.
- Tempo de Término: Dado por $tt[u]$, é o momento em que a visita a toda lista de vértices adjacentes a u foi concluída.

Árvore (*Tree*): A aresta (u, v) é do tipo árvore se v foi descoberta pela primeira vez ao percorrer o grafo.

Retorno (*Back*): conectam um vértice u com um antecessor v em uma árvore DFS (inclui *loops*). Liga um vértice a um antecessor na árvore.

Avanço (*Forward*): não pertencem à árvore da DFS mas conectam um vértice a um descendente que pertence à árvore.

Cruzamento (*Cross*): podem conectar vértices na mesma árvore DFS, ou em duas árvores diferentes.

Classificação de Arestas

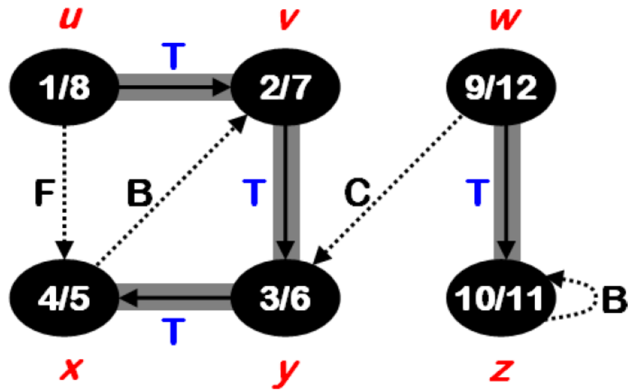
A aresta pode ser classificada conforme a cor do vértice que ela incide:

white : aresta **T (*tree*)**

- Identificação inicial de todo vértice não-descoberto.

gray : aresta **B (*back*)**

- Vértices cinza (*gray*) formam uma sequência linear de descendentes que correspondem à pilha de invocações ativas ao procedimento DFS.
- Nesse processo de exploração, se um vértice cinza encontra outro vértice cinza então foi encontrado um ancestral.



Classificação de Arestas

A aresta pode ser classificada conforme a cor do vértice que ela incide:

white : aresta **T** (***tree***)

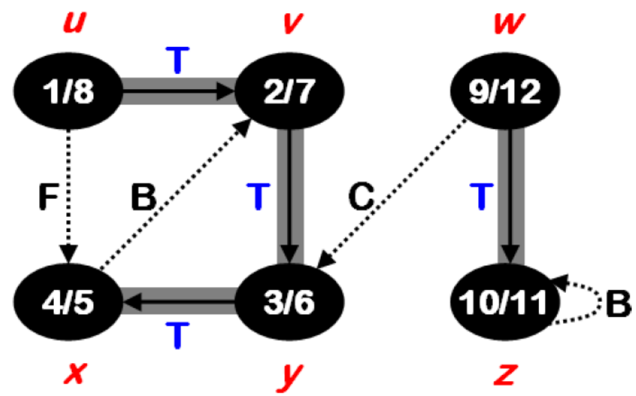
- Identificação inicial de todo vértice não-descoberto.

gray : aresta **B** (***back***)

- Vértices cinza (*gray*) formam uma sequência linear de descendentes que correspondem à pilha de invocações ativas ao procedimento DFS.
- Nesse processo de exploração, se um vértice cinza encontra outro vértice cinza então foi encontrado um ancestral.

black : aresta **F** (***forward***) ou **C** (***cross***)

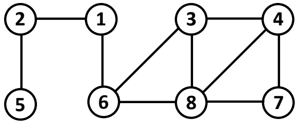
- Possibilidade restante.
- Uma aresta (u, v) é:
  - **F** se $td[u] < td[v]$
  - **C** se $td[u] > td[v]$.



Classificação de Arestas – Estratégia

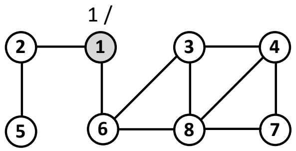
1. Todos os vértices inicialmente são brancos
2. Quando um vértice v é descoberto pela primeira vez ele torna-se cinza e recebe um marcador de tempo de descoberta $td[v]$.
3. Quando todos os vértices adjacentes a v forem completamente descobertos, v torna-se preto e recebe um marcador de tempo de término $tt[v]$.

Classificação de Arestas – Exemplo



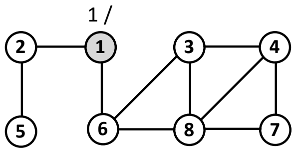
Árvore da DFS

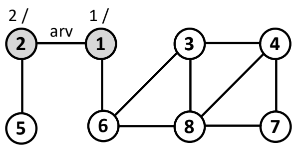
Classificação de Arestas – Exemplo



Árvore da DFS

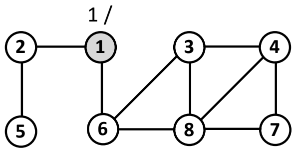
Classificação de Arestas – Exemplo

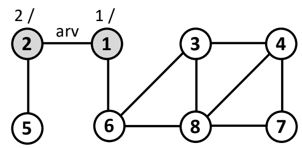


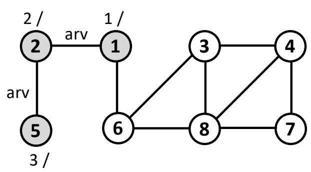


Árvore da DFS

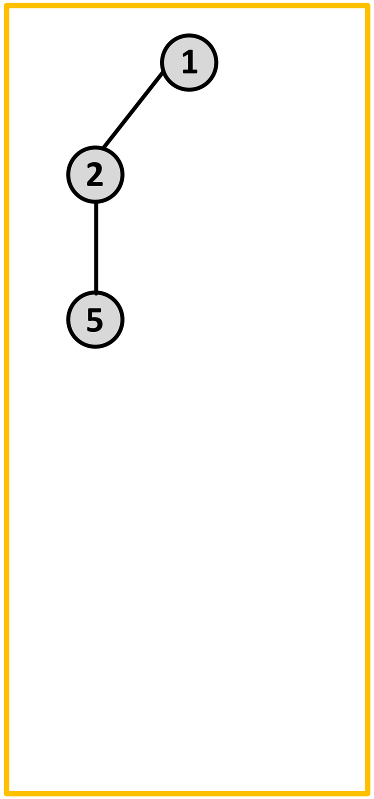
Classificação de Arestas – Exemplo



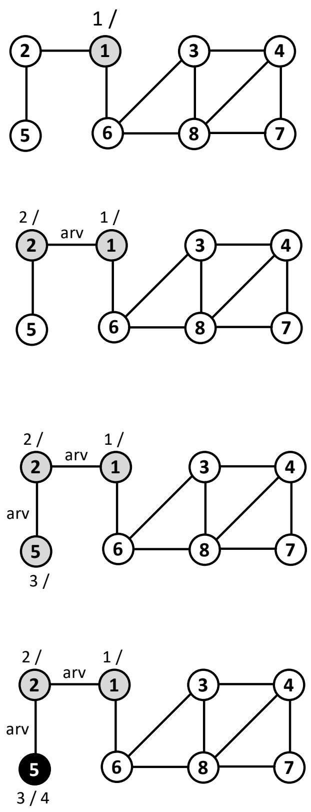




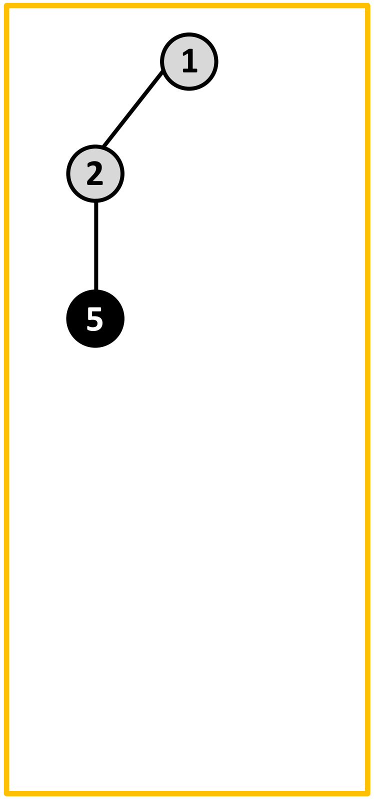
Árvore da DFS



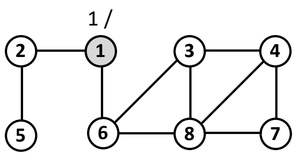
Classificação de Arestas – Exemplo

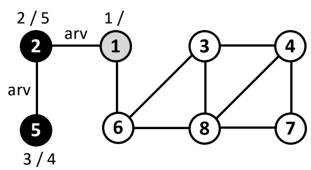


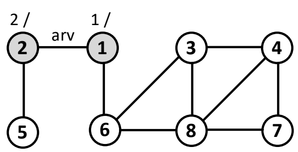
Árvore da DFS

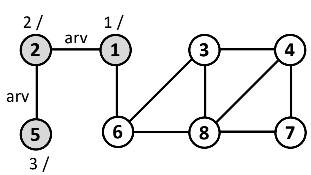


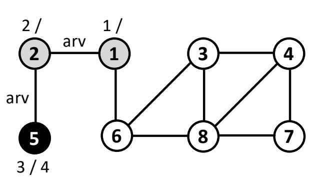
Classificação de Arestas – Exemplo



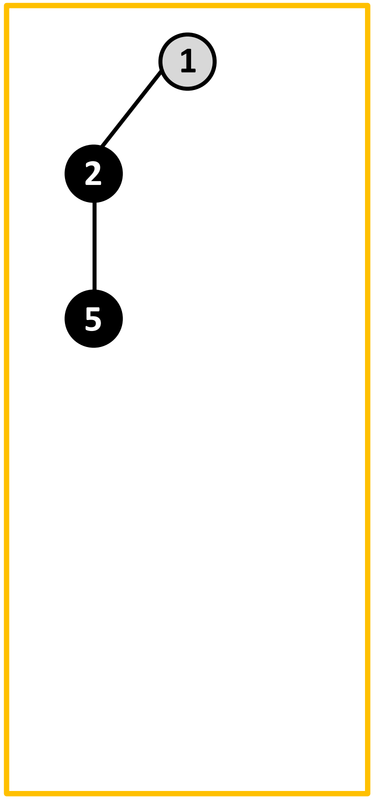




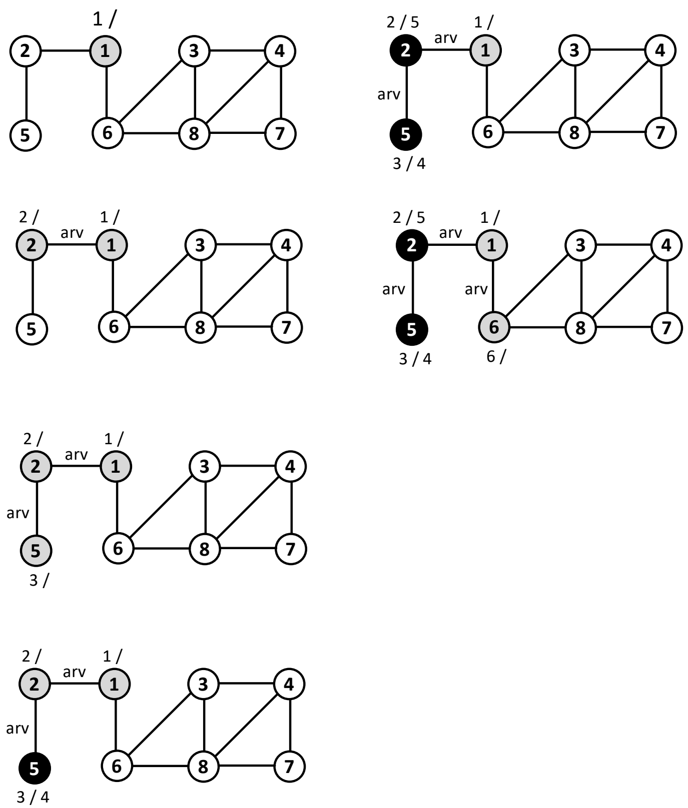




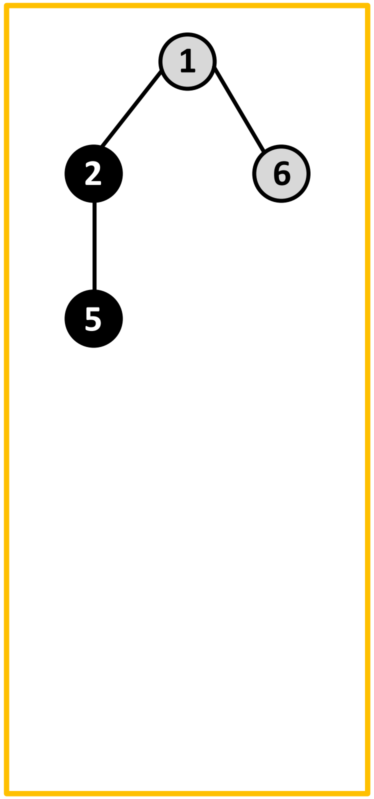
Árvore da DFS



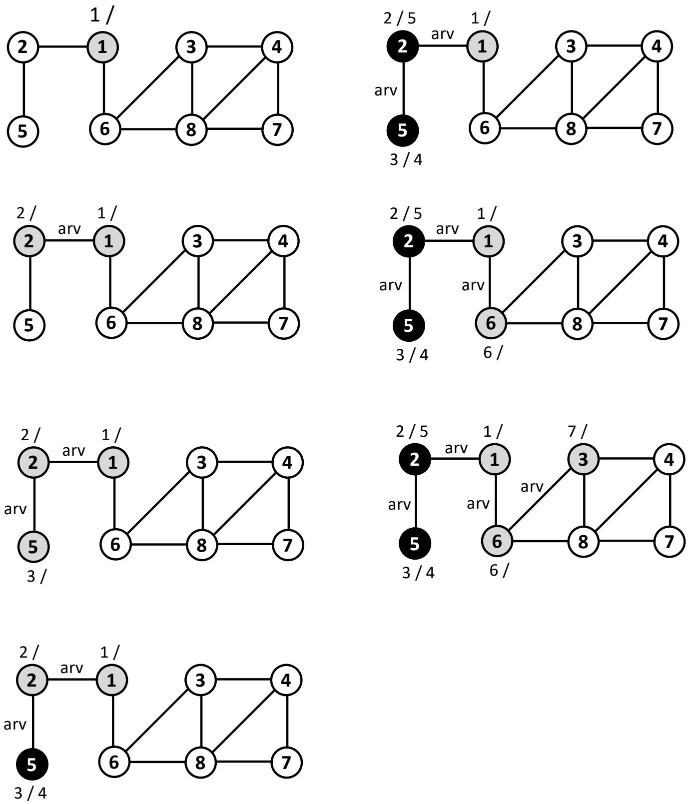
Classificação de Arestas – Exemplo



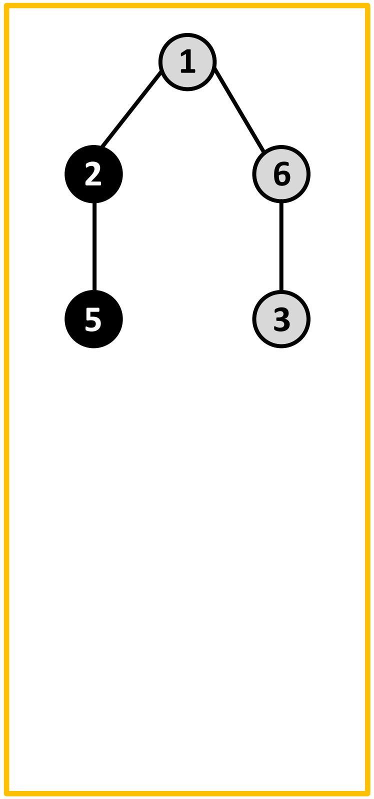
Árvore da DFS



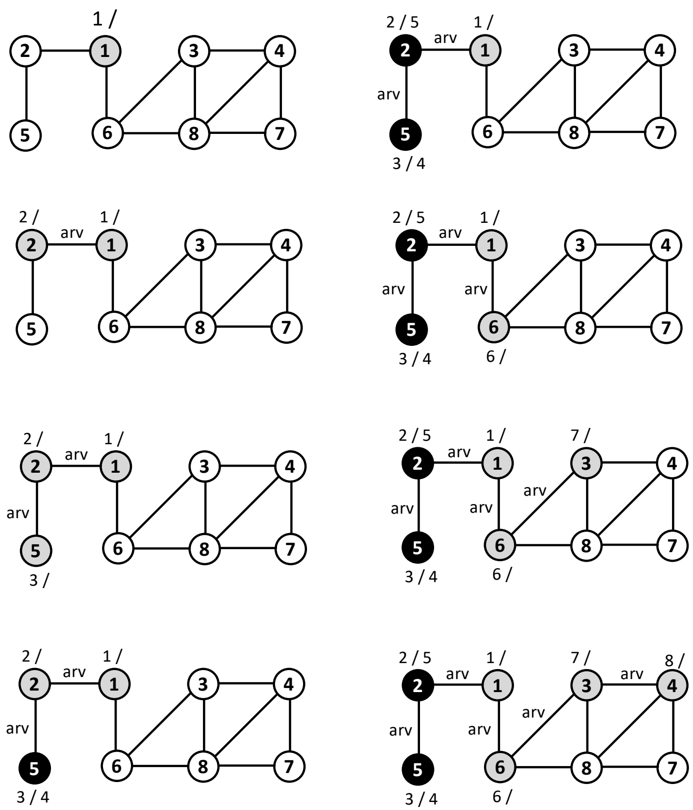
Classificação de Arestas – Exemplo



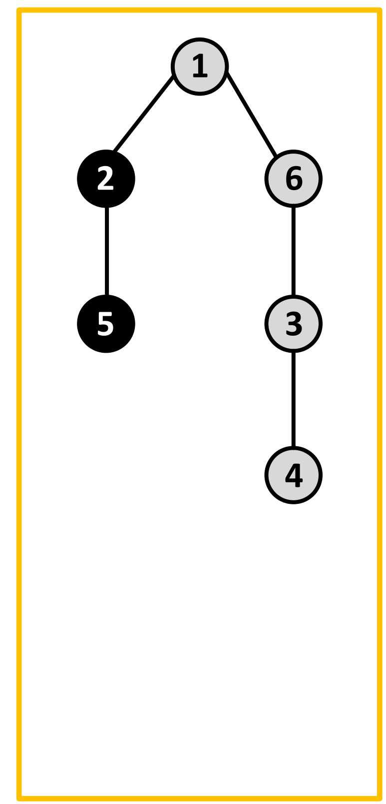
Árvore da DFS



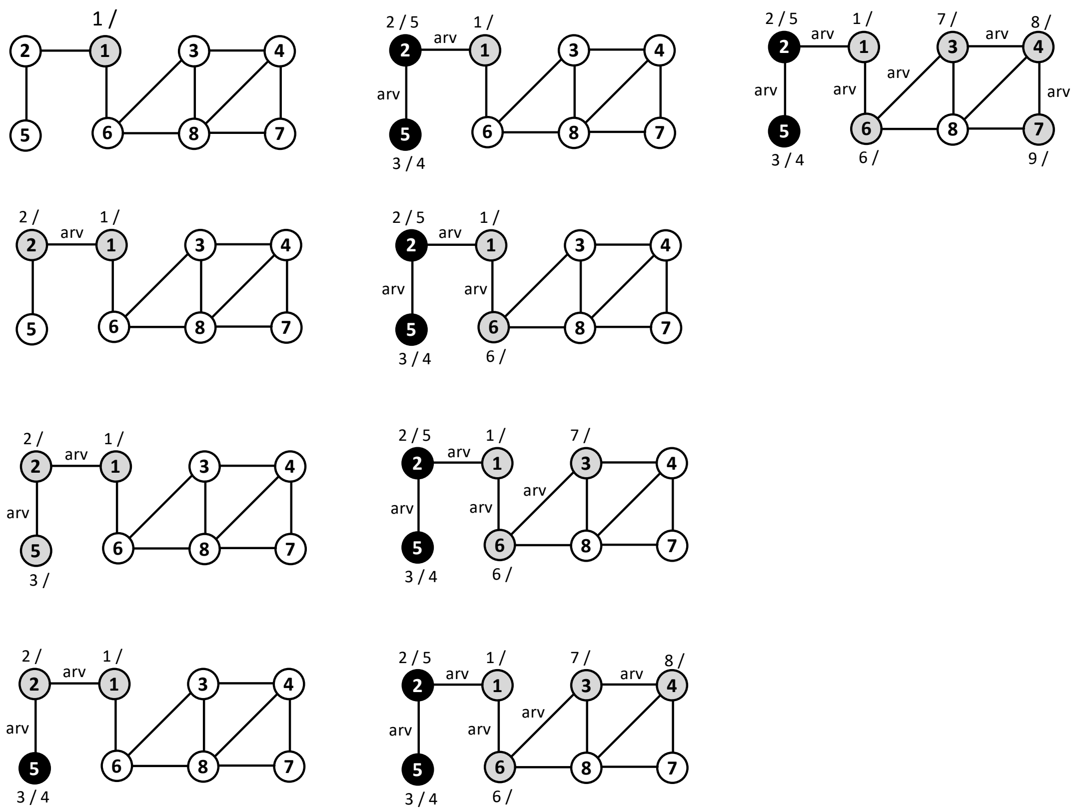
Classificação de Arestas – Exemplo



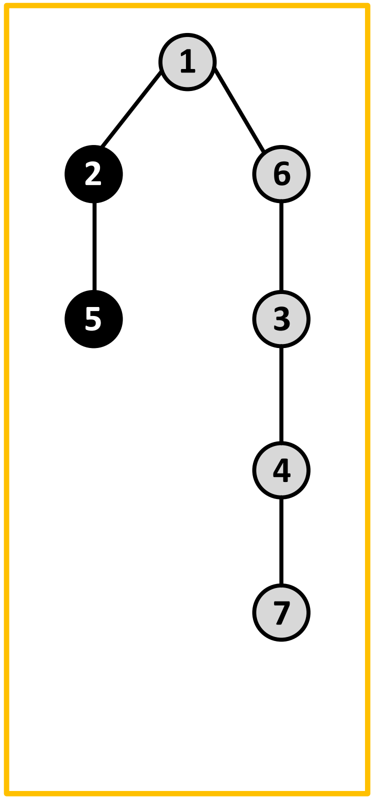
Árvore da DFS



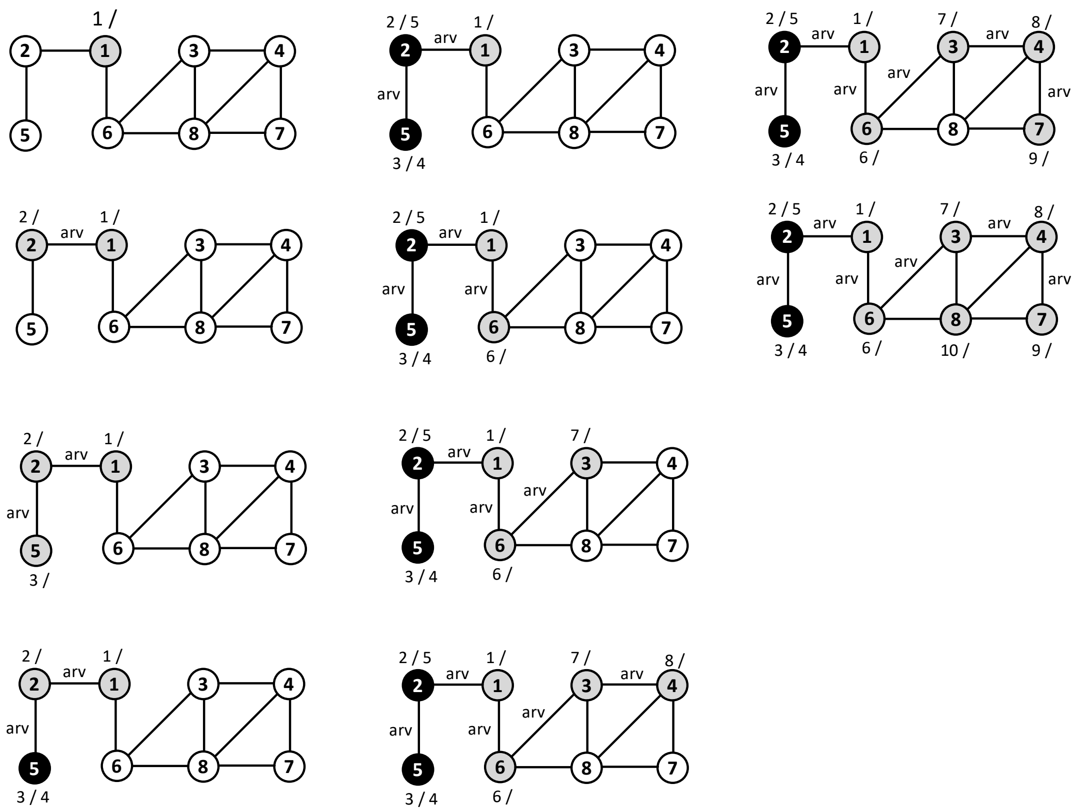
Classificação de Arestas – Exemplo



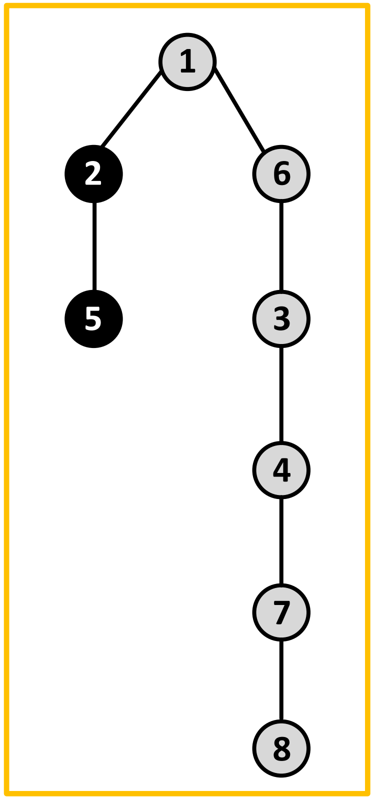
Árvore da DFS



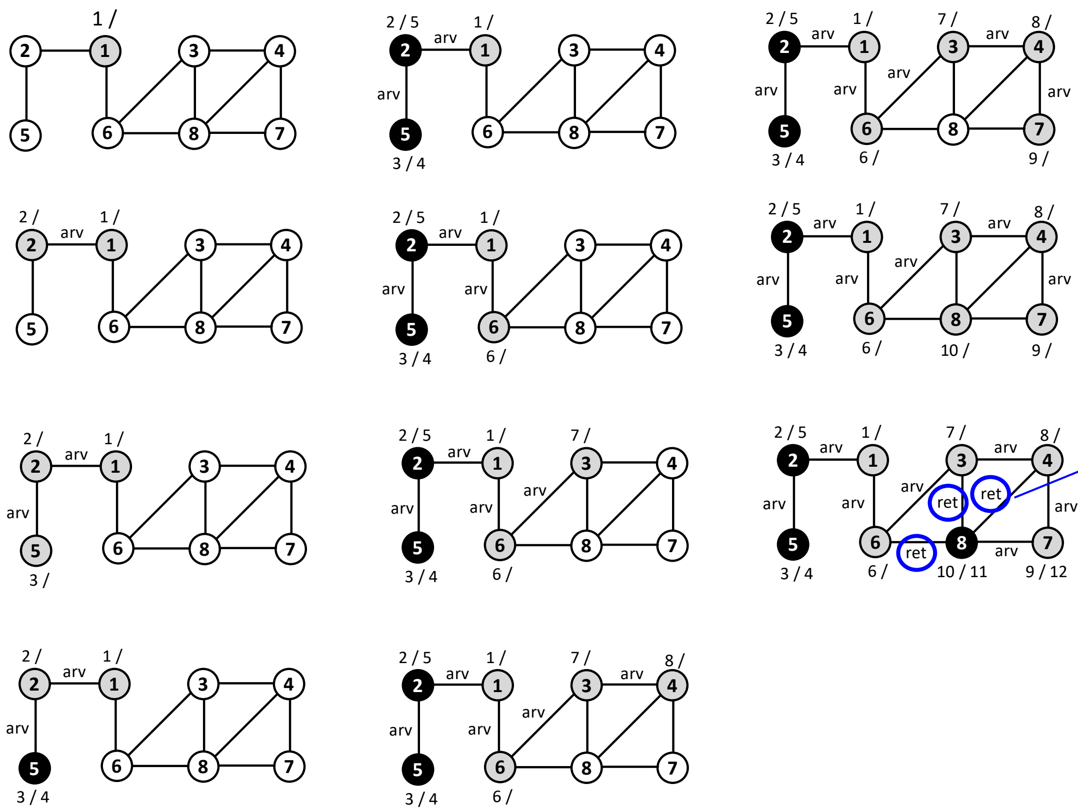
Classificação de Arestas – Exemplo



Árvore da DFS



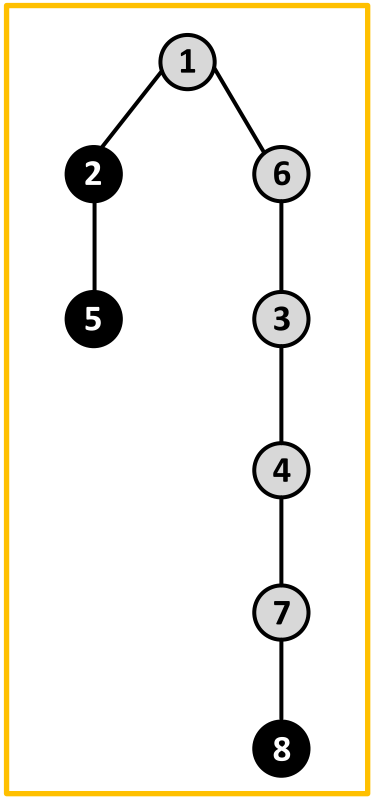
Classificação de Arestas – Exemplo



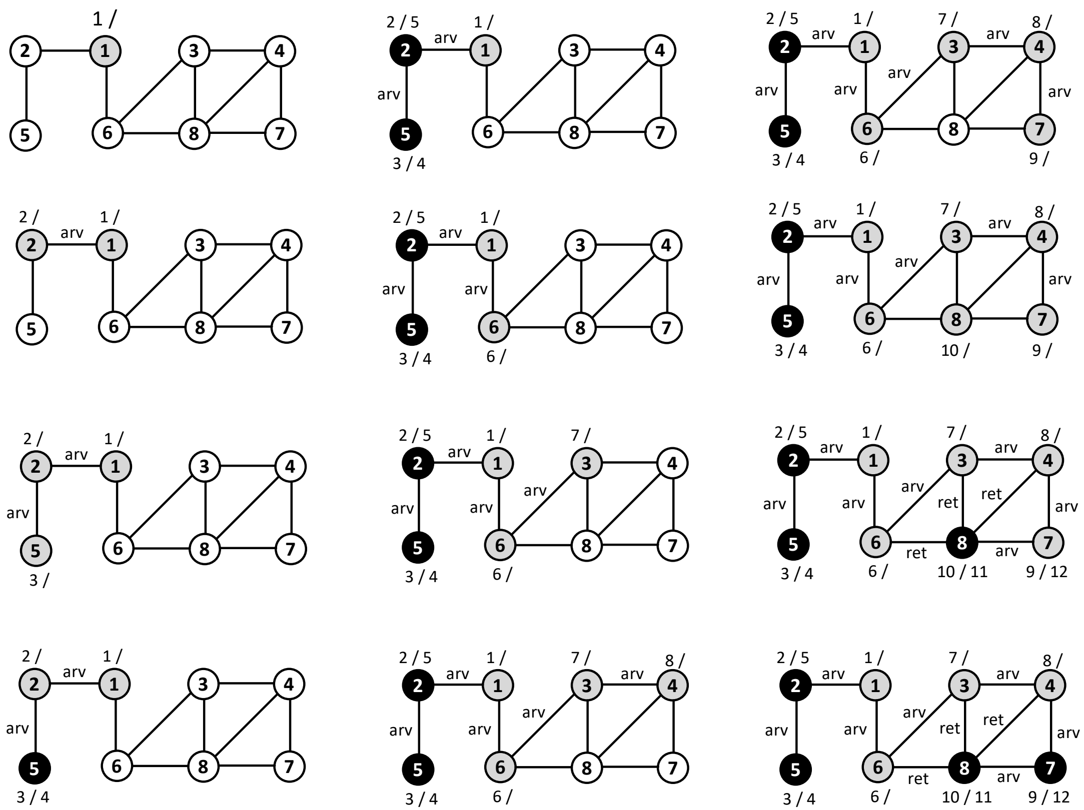
Arestas tipo Retorno (Back): um vértice cinza encontra outro vértice cinza (seu ancestral).

Vértice 8 torna-se preto pois todos seus adjacentes já foram visitados.

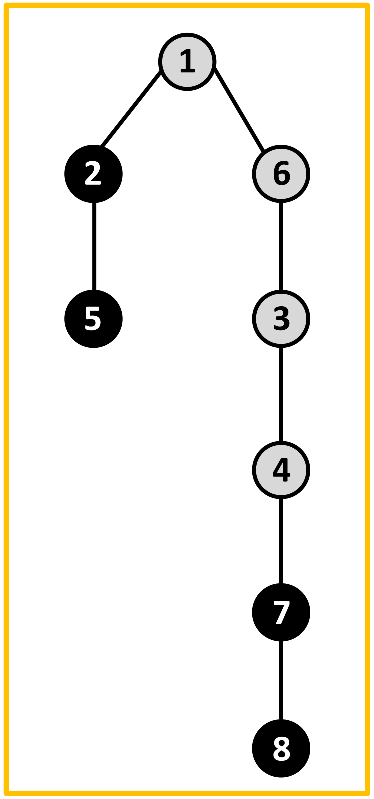
Árvore da DFS



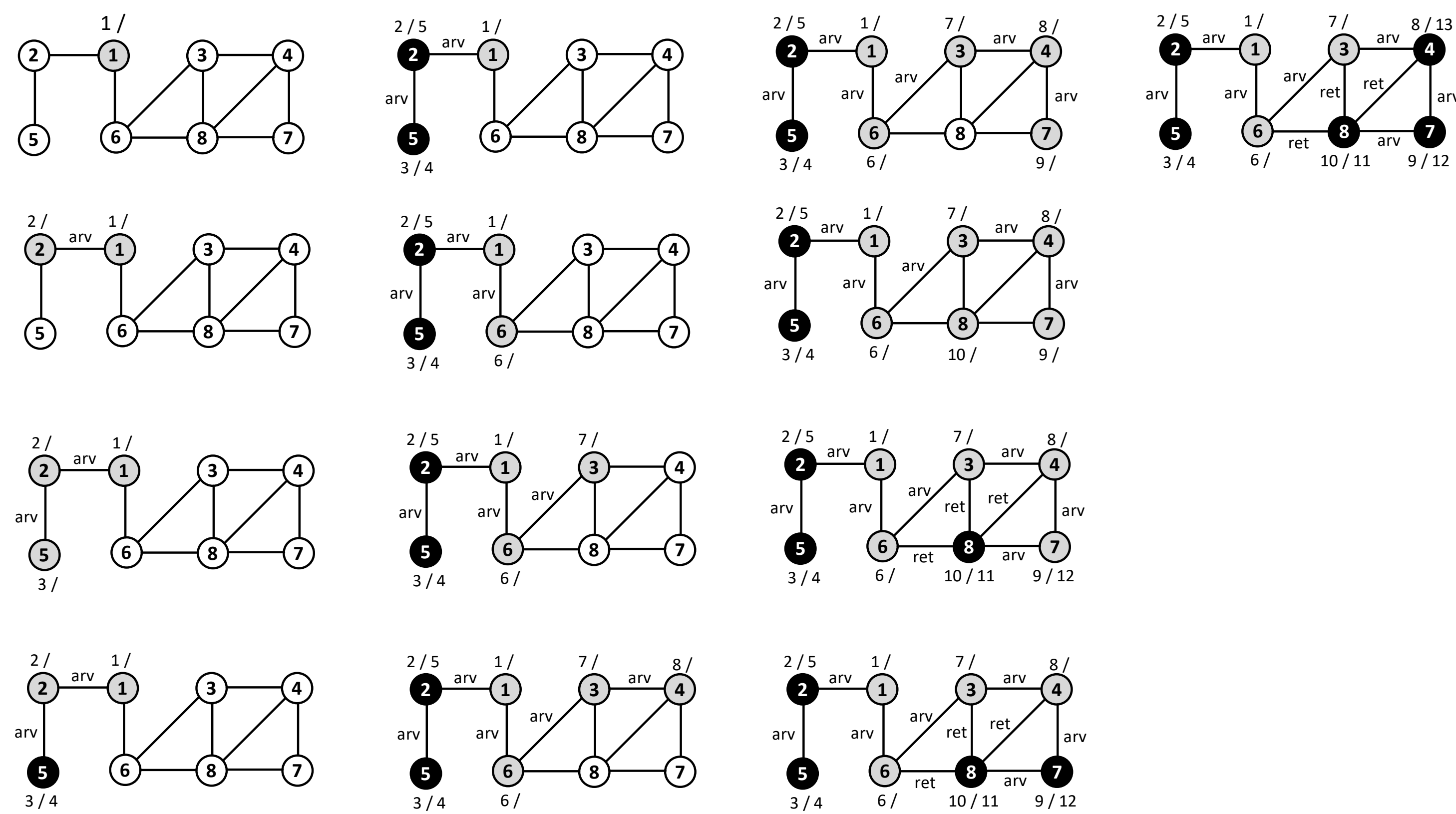
Classificação de Arestas – Exemplo



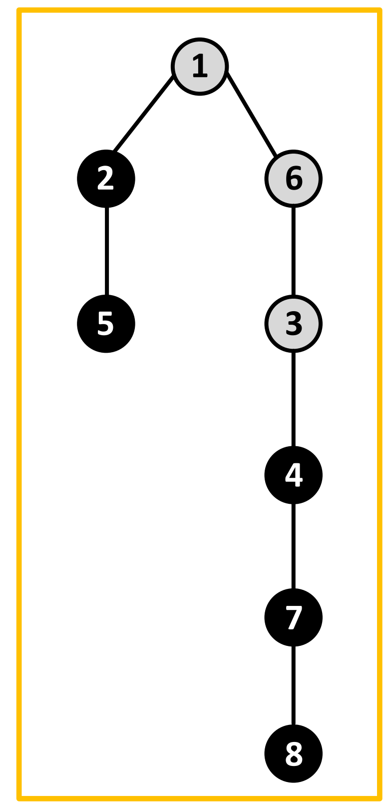
Árvore da DFS



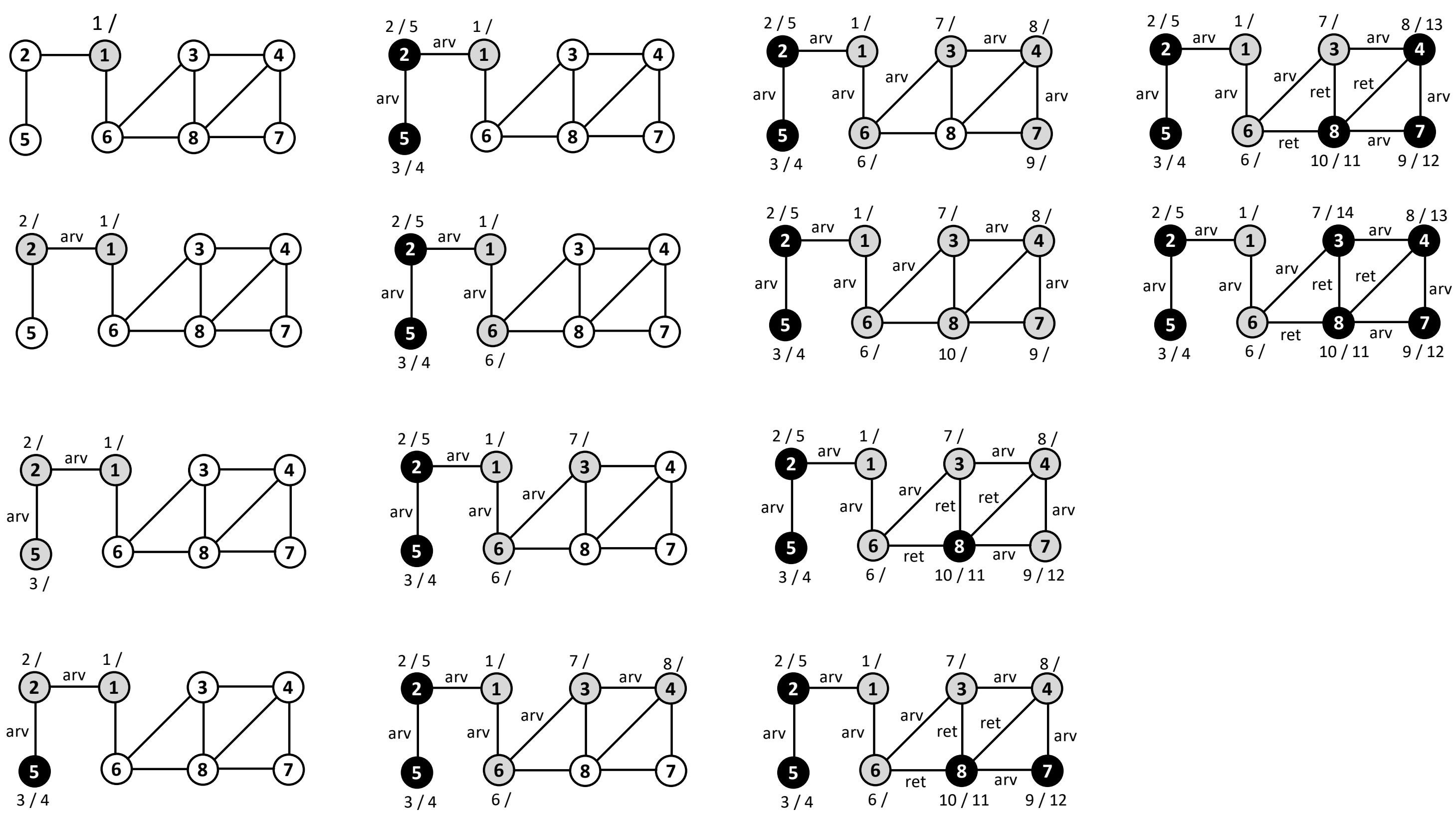
Classificação de Arestas – Exemplo



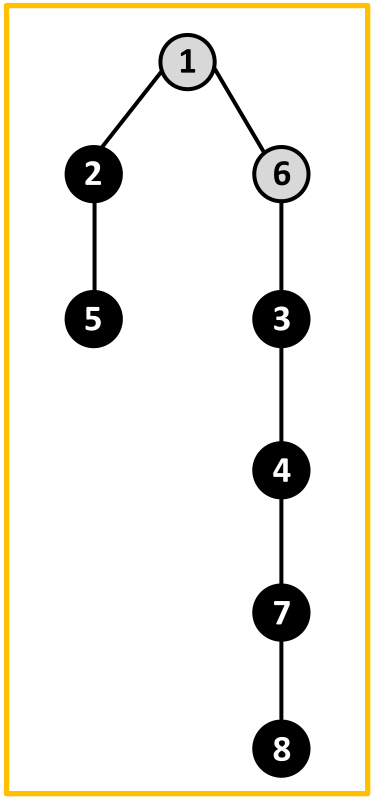
Árvore da DFS



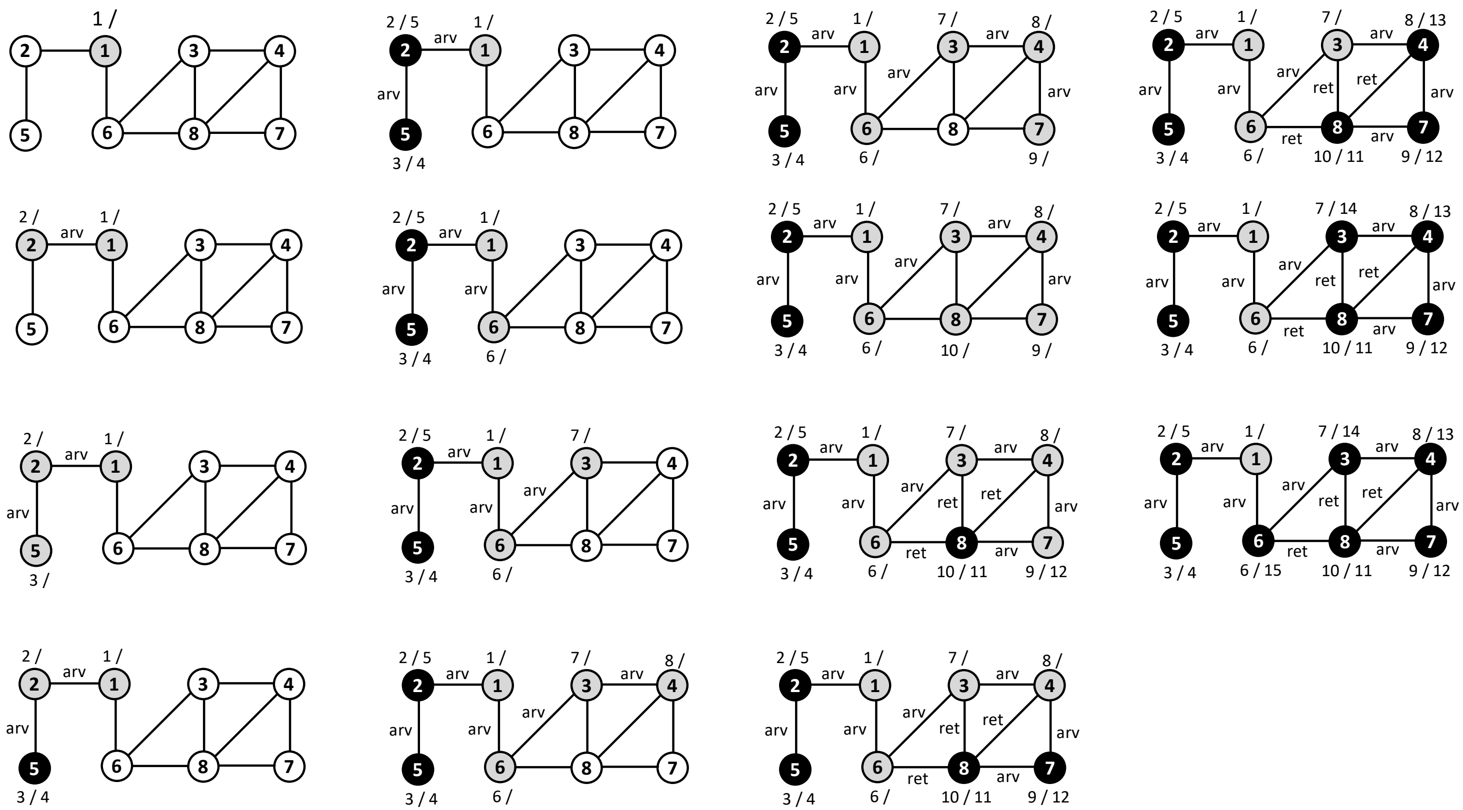
Classificação de Arestas – Exemplo



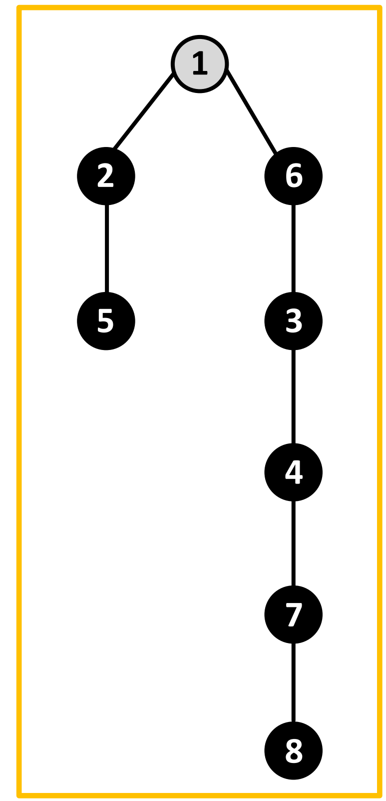
Árvore da DFS



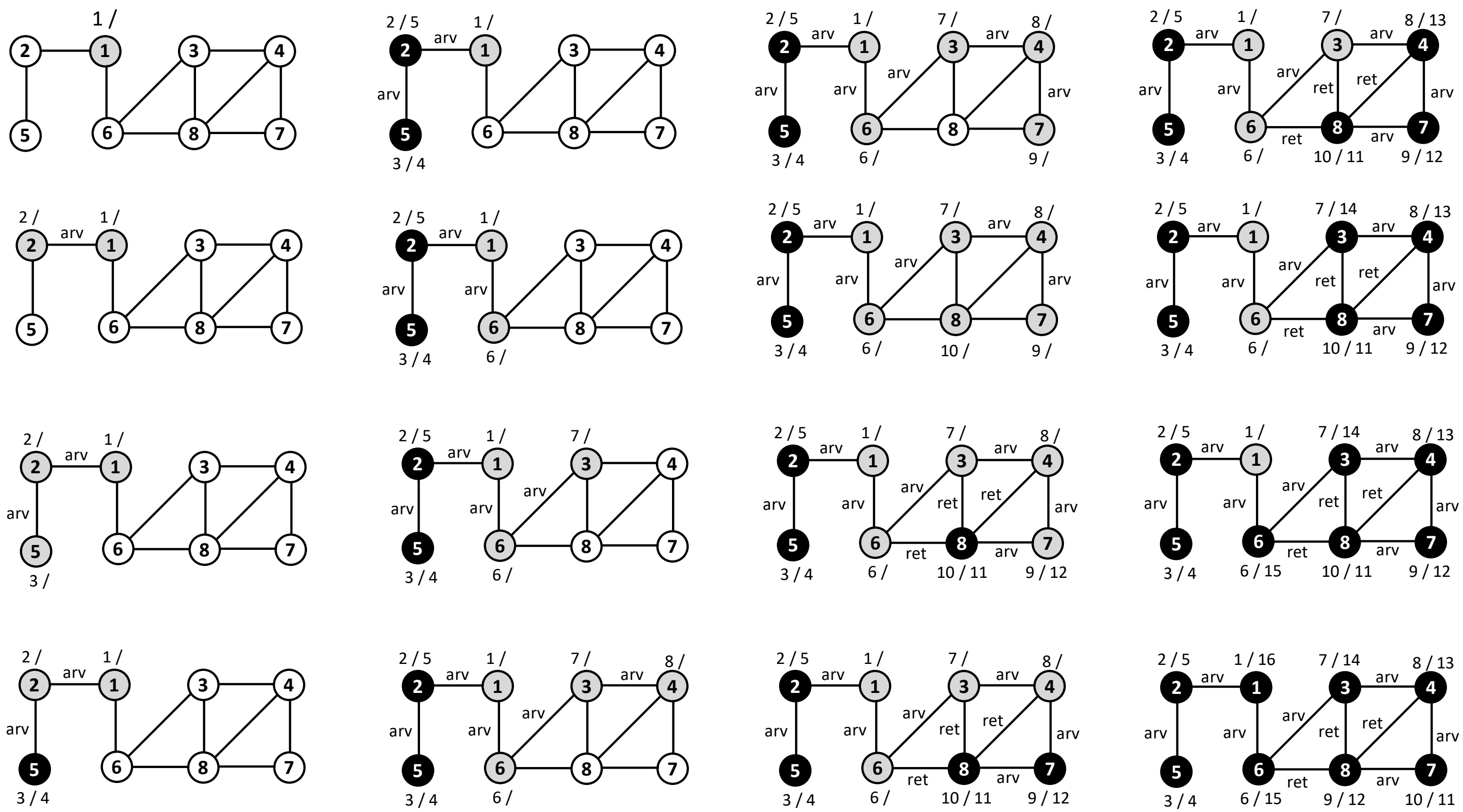
Classificação de Arestas – Exemplo



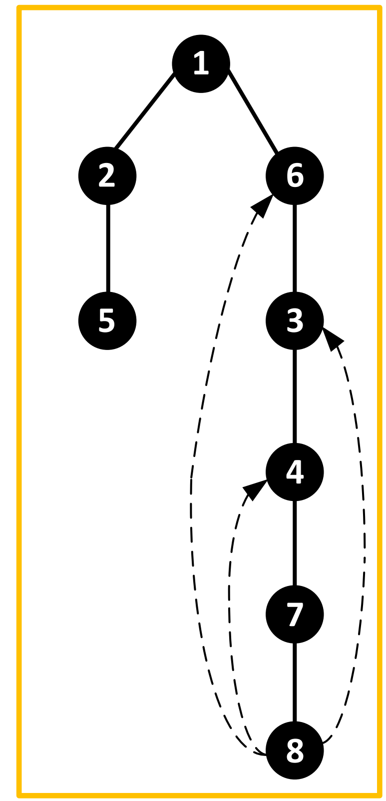
Árvore da DFS



Classificação de Arestas – Exemplo

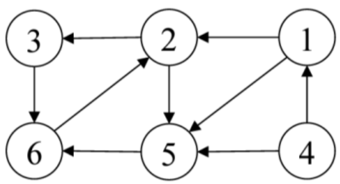


Árvore da DFS



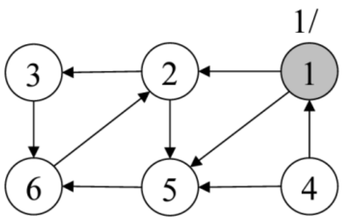
Em um grafo não-direcionado as arestas são <u>apenas</u> do tipo **árvore** ou **retorno**.

Classificação de Arestas – Exemplo (digrafo)



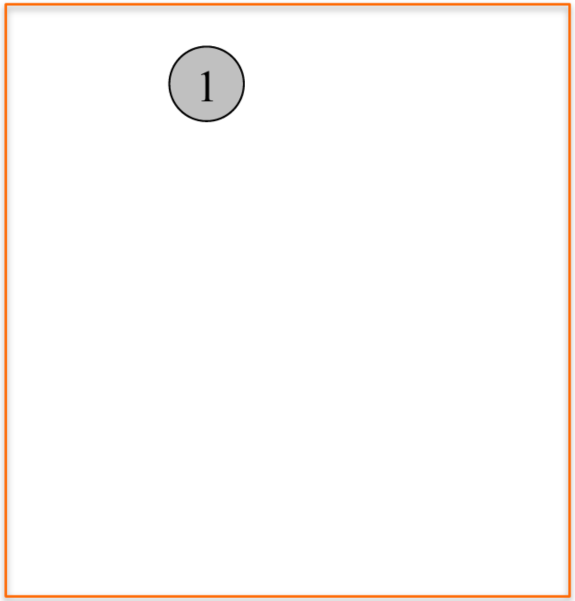
Árvore da DFS

Classificação de Arestas – Exemplo (digrafo)

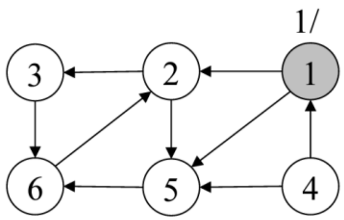


Vértice origem: 1
Tempo de descoberta: 1
Ação: vértice 1 torna-se cinza
Tempo de término: -

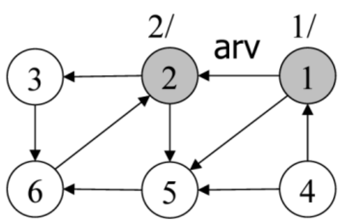
Árvore da DFS



Classificação de Arestas – Exemplo (digrafo)

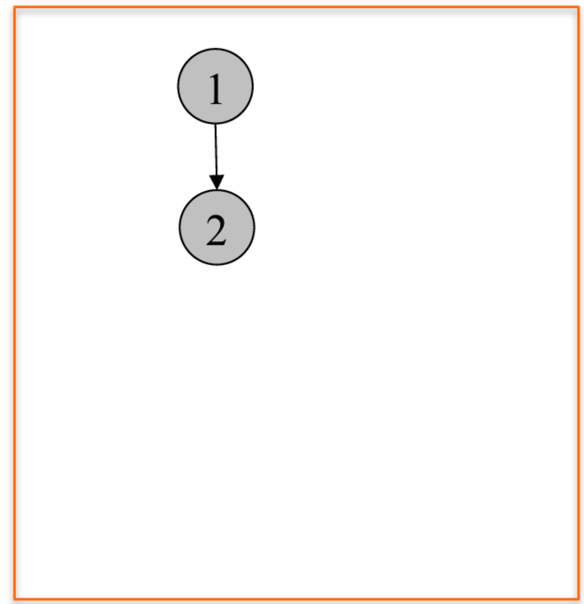


Vértice origem: 1
Tempo de descoberta: 1
Ação: vértice 1 torna-se cinza
Tempo de término: -

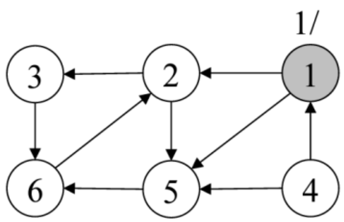


Adjacente à 1 não-descoberto: 2
Tempo de descoberta: 2
Ação: vértice 2 torna-se cinza
Tempo de término: -

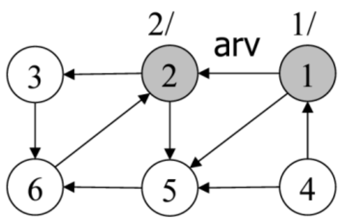
Árvore da DFS



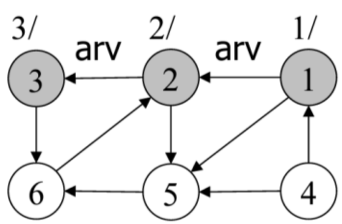
Classificação de Arestas – Exemplo (digrafo)



Vértice origem: 1
Tempo de descoberta: 1
Ação: vértice 1 torna-se cinza
Tempo de término: -

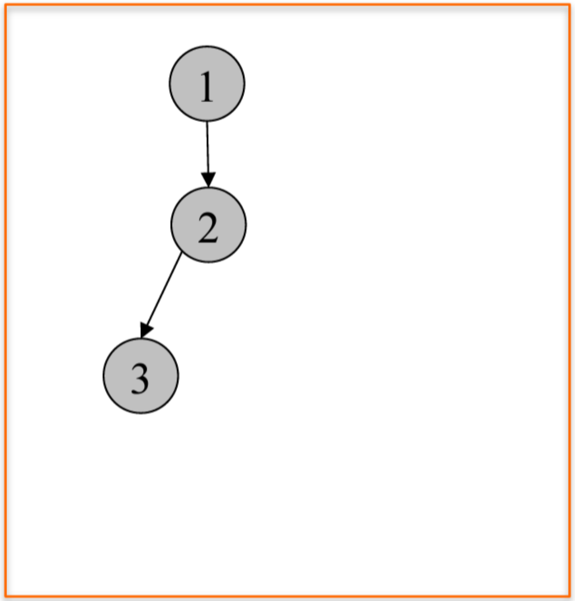


Adjacente à 1 não-descoberto: 6
Tempo de descoberta: 2
Ação: vértice 6 torna-se cinza
Tempo de término: -

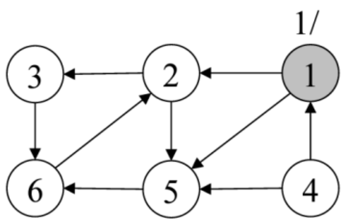


Adjacente à 2 não-descoberto: 3
Tempo de descoberta: 3
Ação: vértice 3 torna-se cinza
Tempo de término: -

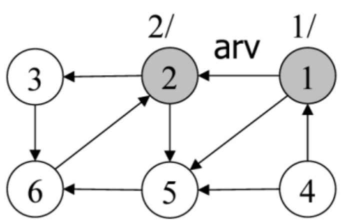
Árvore da DFS



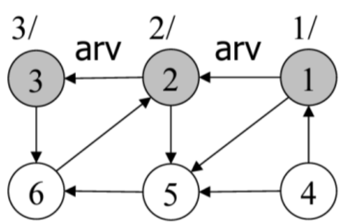
Classificação de Arestas – Exemplo (digrafo)



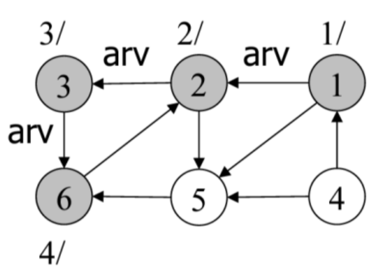
Vértice origem: 1
Tempo de descoberta: 1
Ação: vértice 1 torna-se cinza
Tempo de término: -



Adjacente à 1 não-descoberto: 6
Tempo de descoberta: 2
Ação: vértice 6 torna-se cinza
Tempo de término: -

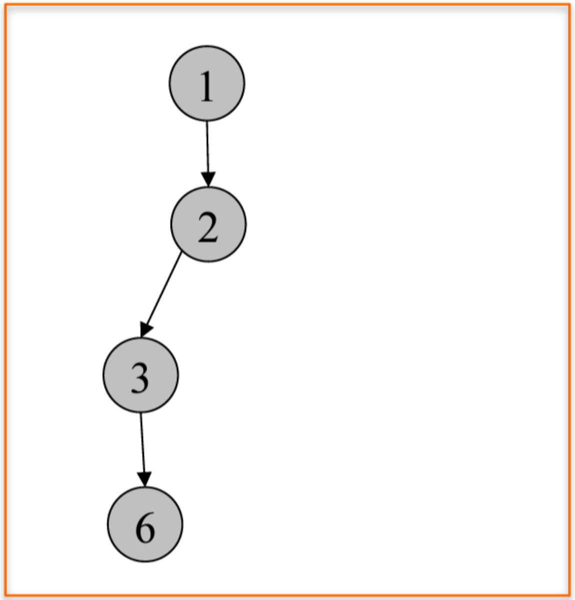


Adjacente à 2 não-descoberto: 3
Tempo de descoberta: 3
Ação: vértice 3 torna-se cinza
Tempo de término: -

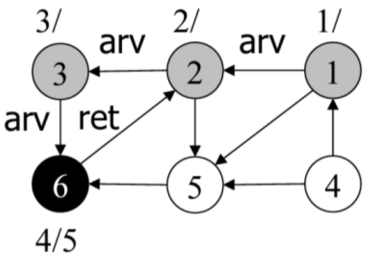


Adjacente à 3 não-descoberto: 6
Tempo de descoberta: 4
Ação: vértice 6 torna-se cinza
Tempo de término: -

Árvore da DFS

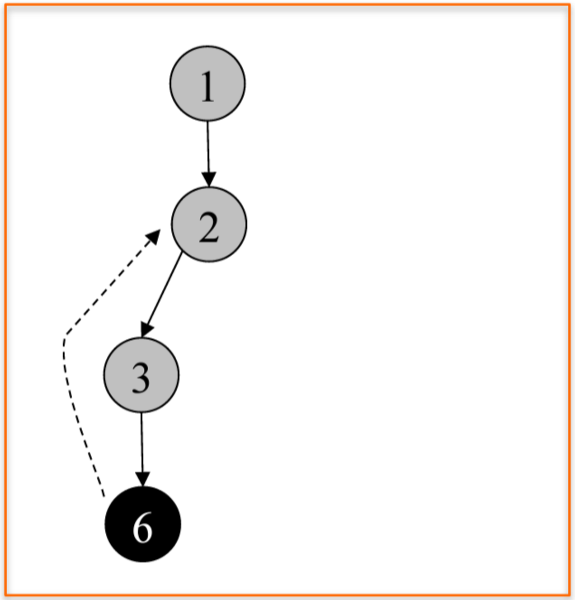


Classificação de Arestas – Exemplo (digrafo)

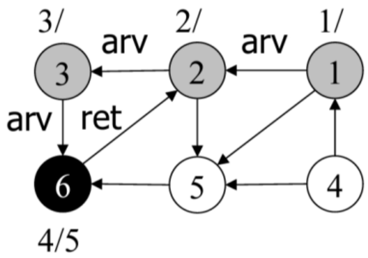


Adjacente à 6 não-descoberto: nenhum
Tempo de descoberta: 4
Ação: vértice 6 torna-se preto
Tempo de término: 5

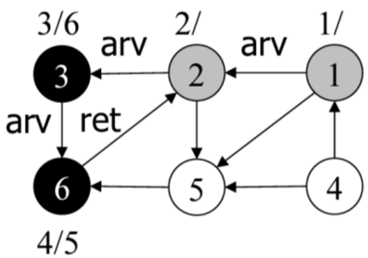
Árvore da DFS



Classificação de Arestas – Exemplo (digrafo)

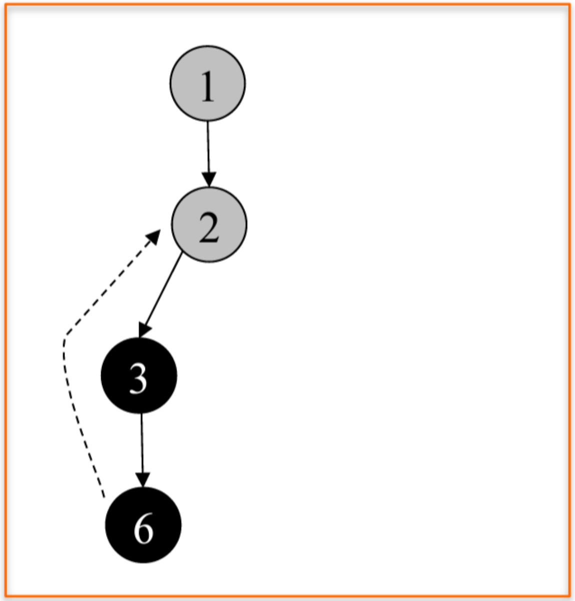


Adjacente à 6 não-descoberto: nenhum
Tempo de descoberta: 4
Ação: vértice 6 torna-se preto
Tempo de término: 5

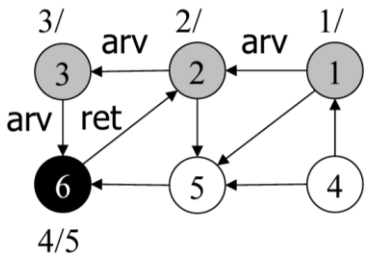


Adjacente à 3 não-descoberto: nenhum
Tempo de descoberta: 3
Ação: vértice 3 torna-se preto
Tempo de término: 6

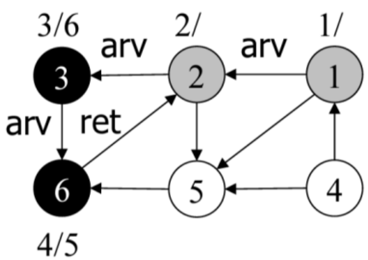
Árvore da DFS



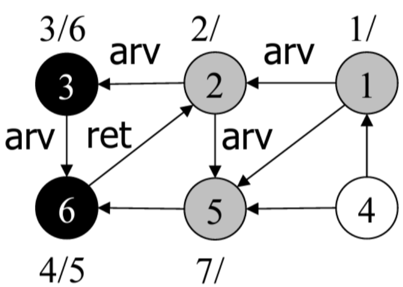
Classificação de Arestas – Exemplo (digrafo)



Adjacente à 6 não-descoberto: nenhum
Tempo de descoberta: 4
Ação: vértice 6 torna-se preto
Tempo de término: 5

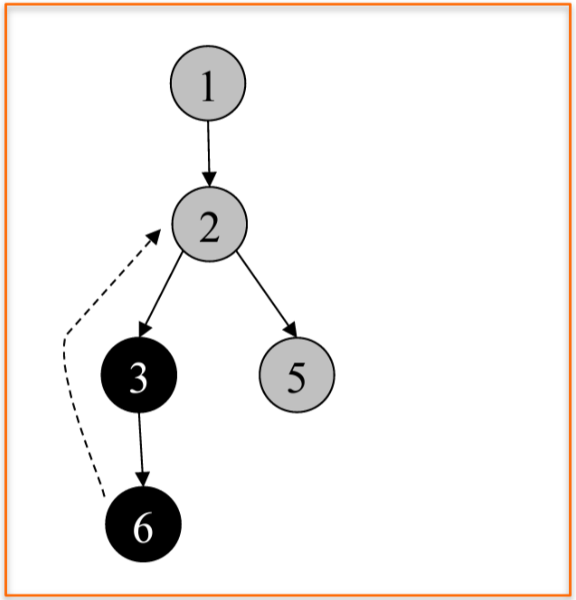


Adjacente à 3 não-descoberto: nenhum
Tempo de descoberta: 3
Ação: vértice 3 torna-se preto
Tempo de término: 6

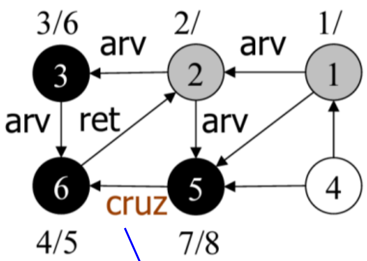


Adjacente à 2 não-descoberto: 5
Tempo de descoberta: 7
Ação: vértice 5 torna-se cinza
Tempo de término: -

Árvore da DFS



Classificação de Arestas – Exemplo (digrafo)

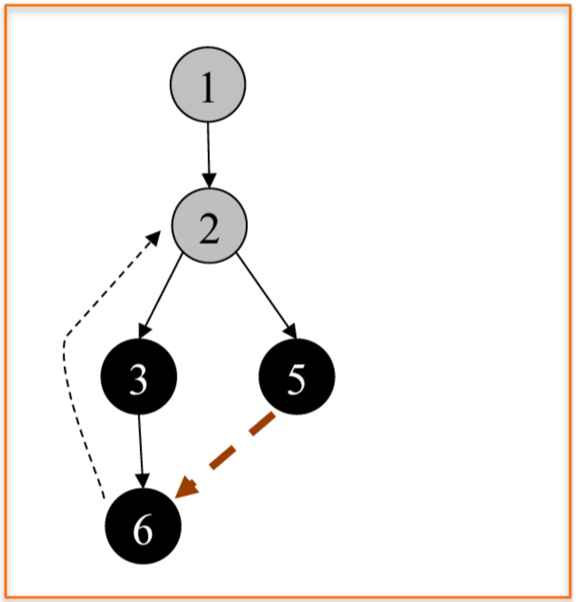


Adjacente à 5 não-descoberto: nenhum
Tempo de descoberta: 7
Ação: vértice 5 torna-se preto
Tempo de término: 8

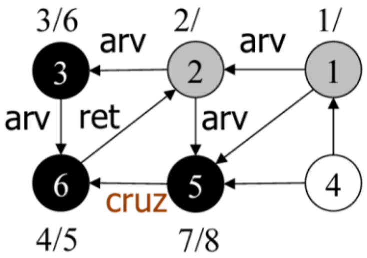
Aresta tipo Cruzamento (Cross): $td[u] > td[v]$

$td[5] = 7$ e $td[6] = 4$.

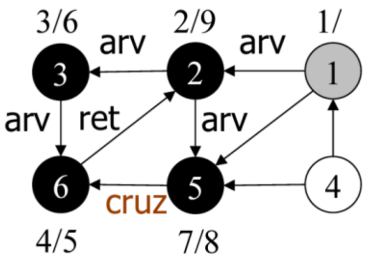
Árvore da DFS



Classificação de Arestas – Exemplo (digrafo)

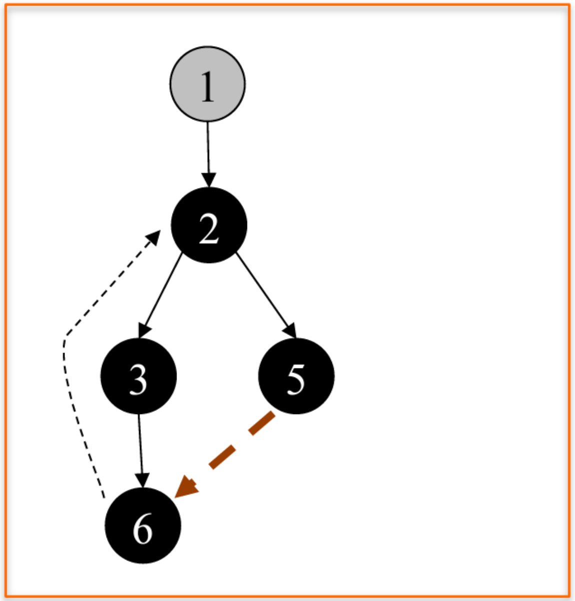


Adjacente à 5 não-descoberto: nenhum
Tempo de descoberta: 7
Ação: vértice 5 torna-se preto
Tempo de término: 8

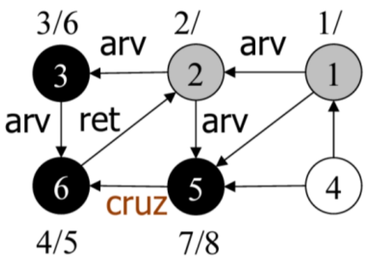


Adjacente à 2 não-descoberto: nenhum
Tempo de descoberta: 2
Ação: vértice 2 torna-se preto
Tempo de término: 9

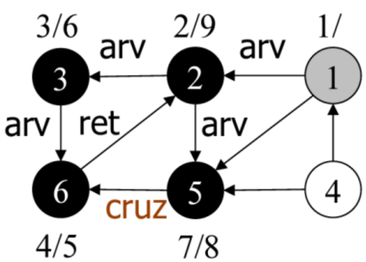
Árvore da DFS



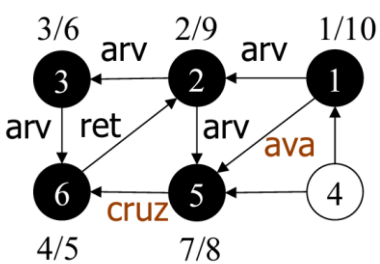
Classificação de Arestas – Exemplo (digrafo)



Adjacente à 5 não-descoberto: nenhum
Tempo de descoberta: 7
Ação: vértice 5 torna-se preto
Tempo de término: 8

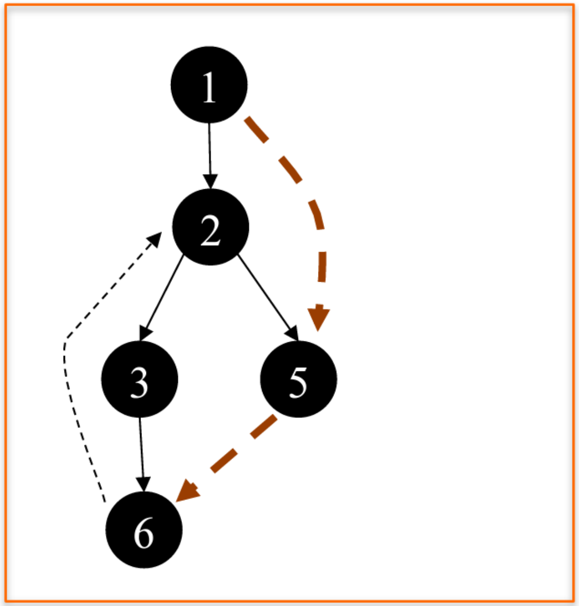


Adjacente à 2 não-descoberto: nenhum
Tempo de descoberta: 2
Ação: vértice 2 torna-se preto
Tempo de término: 9

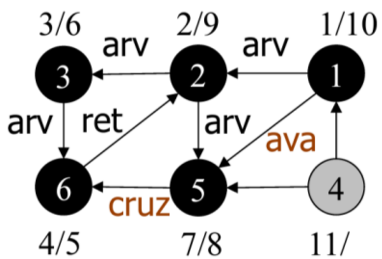


Adjacente à 1 não-descoberto: nenhum
Tempo de descoberta: 1
Ação: vértice 1 torna-se preto
Tempo de término: 10

Árvore da DFS

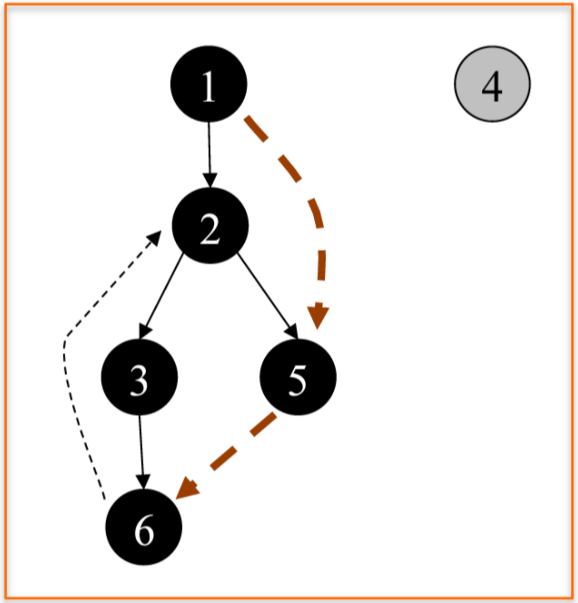


Classificação de Arestas – Exemplo (digrafo)

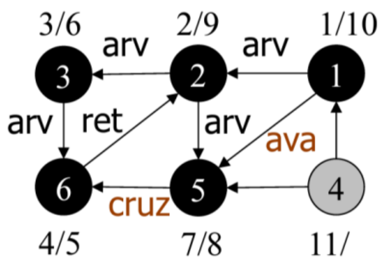


Vértice origem: 4
Tempo de descoberta: 11
Ação: vértice 4 torna-se cinza
Tempo de término: -

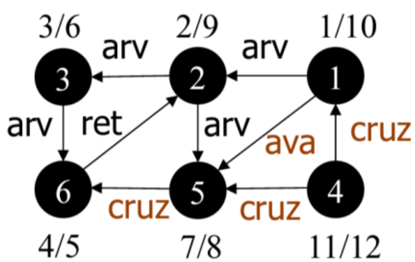
Árvore da DFS



Classificação de Arestas – Exemplo (digrafo)

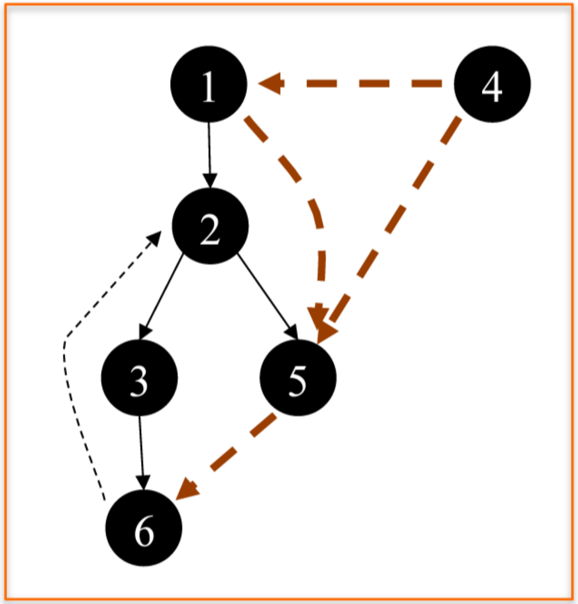


Vértice origem: 4
Tempo de descoberta: 11
Ação: vértice 4 torna-se cinza
Tempo de término: -



Adjacente à 4 não-descoberto: nenhum
Tempo de descoberta: 11
Ação: vértice 4 torna-se preto
Tempo de término: 12

Árvore da DFS



Análise de Complexidade

Busca em Largura (BFS)

Cada vértice só entra na fila uma vez. Inserir e remover na fila possuem complexidade constante, as quais são realizadas $|V|$ vezes cada.

A lista de adjacências de cada vértice é examinada apenas uma vez, e a soma dos comprimentos de todas as listas é $(|E|)$.

Logo, se o grafo for representado por uma lista de adjacências, a complexidade $O(V + E)$.

Análise de Complexidade

Busca em Largura (BFS)

Cada vértice só entra na fila uma vez. Inserir e remover na fila possuem complexidade constante, as quais são realizadas $|V|$ vezes cada.

A lista de adjacências de cada vértice é examinada apenas uma vez, e a soma dos comprimentos de todas as listas é $(|E|)$.

Logo, se o grafo for representado por uma lista de adjacências, a complexidade $O(V + E)$.

Busca em Profundidade (DFS)

Para cada vértice do grafo a busca percorre todos os seus vizinhos.

Cada aresta é visitada duas vezes (empilhamento e desempilhamento).

Se o grafo for representado por uma lista de adjacências, a complexidade é $O(V + E)$.

Possui um maior custo de espaço devido a pilha recursiva.

DÚVIDAS?

Perguntas? Sugestões?