Algoritmos em Grafos

Caminhamentos

Busca em Largura

Busca em profundidade

Prof. Michelle Nery Nascimento

Propriedades de grafos

- Algumas propriedades de grafos são de simples verificação:
  - Verificação de graus dos vértices
  - Determinação se o grafo é euleriano
  - Determinação se o grafo é completo
- Outras propriedades são relacionadas às arestas e aos caminhos existentes

Caminhamentos

- *Caminhar* em um grafo é mover-se entre seus vértices, verificando propriedades enquanto se caminha
- Algoritmos de *busca em grafos* procuram caminhos com objetivos específicos:

Conectividade	*Busca de um vértice específico (estado)*
Caminho mínimo	*Existência de um caminho*

Busca em grafos

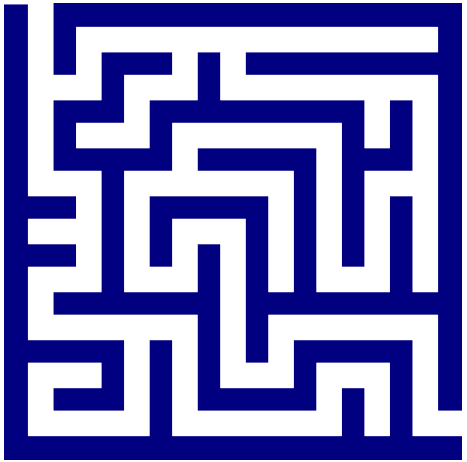
- A busca em grafos tenta encontrar uma sequência de caminhos/ações que leve até a um objetivo
- Uma vez encontrado este objetivo, um programa pode executar tal sequência de ações para atingi-lo

Busca em grafos

- Aplicações
  - Rotas em redes de computadores
  - Caixeiro viajante e variações
  - Jogos digitais
  - Navegação de robôs
  - ...

Labirinto

- Considere o problema de encontrar uma saída em um labirinto
- Modelando cada “sala” como um vértice e suas conexões como arestas, temos um problema de busca em grafos



Busca em largura

- Em inglês, *Breadth First Search* (BFS)
- Consiste em, a partir de um vértice de origem, explorar primeiramente todos os seus vizinhos e, em seguida repetir o procedimento para cada vizinho
- Base para diversos algoritmos importantes que iremos estudar

Busca em largura

- Calcula a distância (caminho mínimo) do vértice de origem até qualquer vértice que possa ser alcançado
- Produz uma árvore que indica todos os vértices que podem ser alcançados
- Funciona em grafos e digrafos

Busca em largura

- Propriedades de um vértice
  - Antecessor ou pai
  - Estado: branco, cinza, preto
  - Distância até o vértice de origem

Busca em largura

- Estados dos vértices
  - Branco: ainda não explorado
  - Cinza: explorado, mas com vizinhos não-explorados
  - Preto: explorado e sem vizinhos não explorados

Busca em largura

- Utiliza uma lista para definir as próximas visitas
- Pode armazenar a árvore de busca e/ou a sequência percorrida até um objetivo

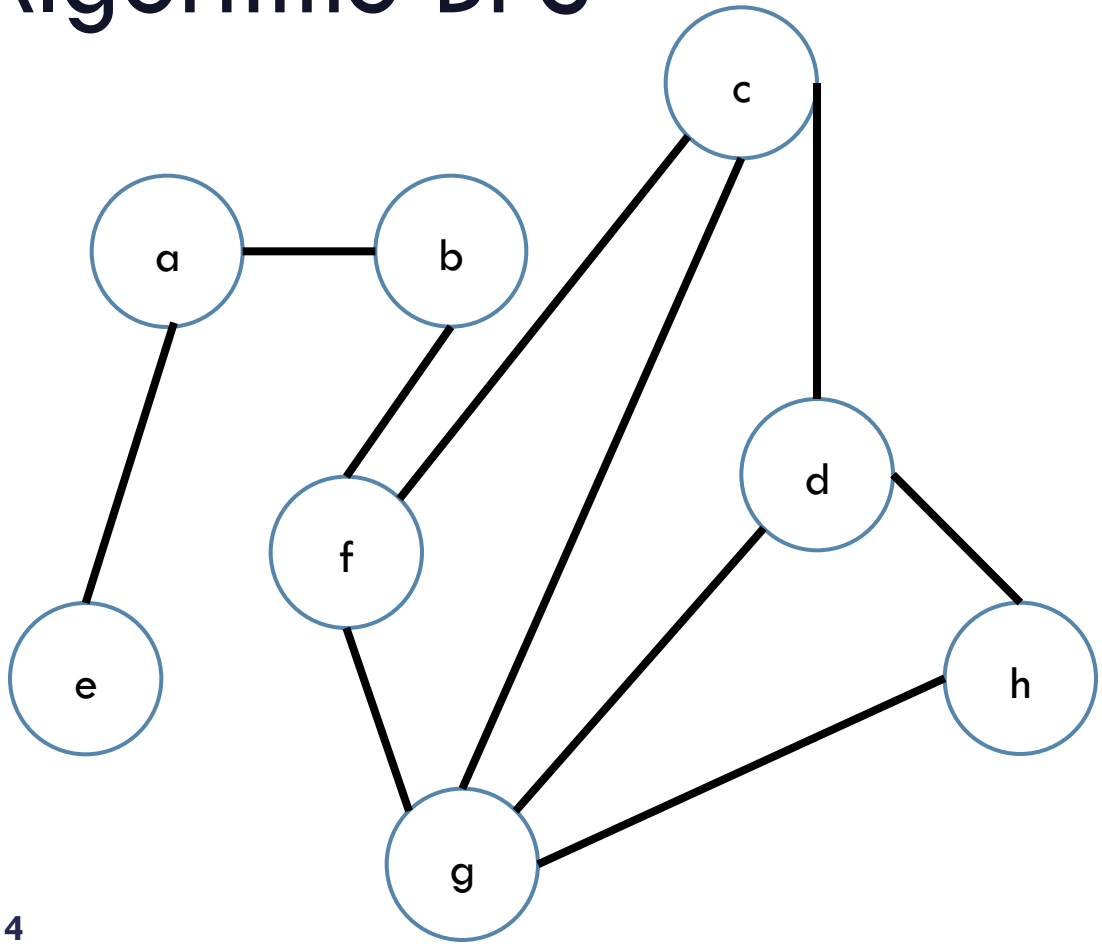
Algoritmo BFS - inicialização

```
Para cada vértice u diferente da origem s faça
        u.cor = branco;
        u.distância = max_value;
        u.pai = null;
Fim para
s.cor = cinza;
s.distância = 0;
u.pai = null;
Q = nova Fila vazia;
```

Algoritmo BFS – busca principal

```
Q.enfileirar(s);
Enquanto (!Q.vazia)
        u = Q.desenfileirar();
        Para cada vértice v adjacente a u
                se v.cor == branco
                        v.cor == cinza;
                        v.distância = u.distância+1;
                        v.pai = u
                        Q.enfileirar(v)
        Fim para
        u.cor = preto;
Fim enquanto
```

Algoritmo BFS



Distâncias

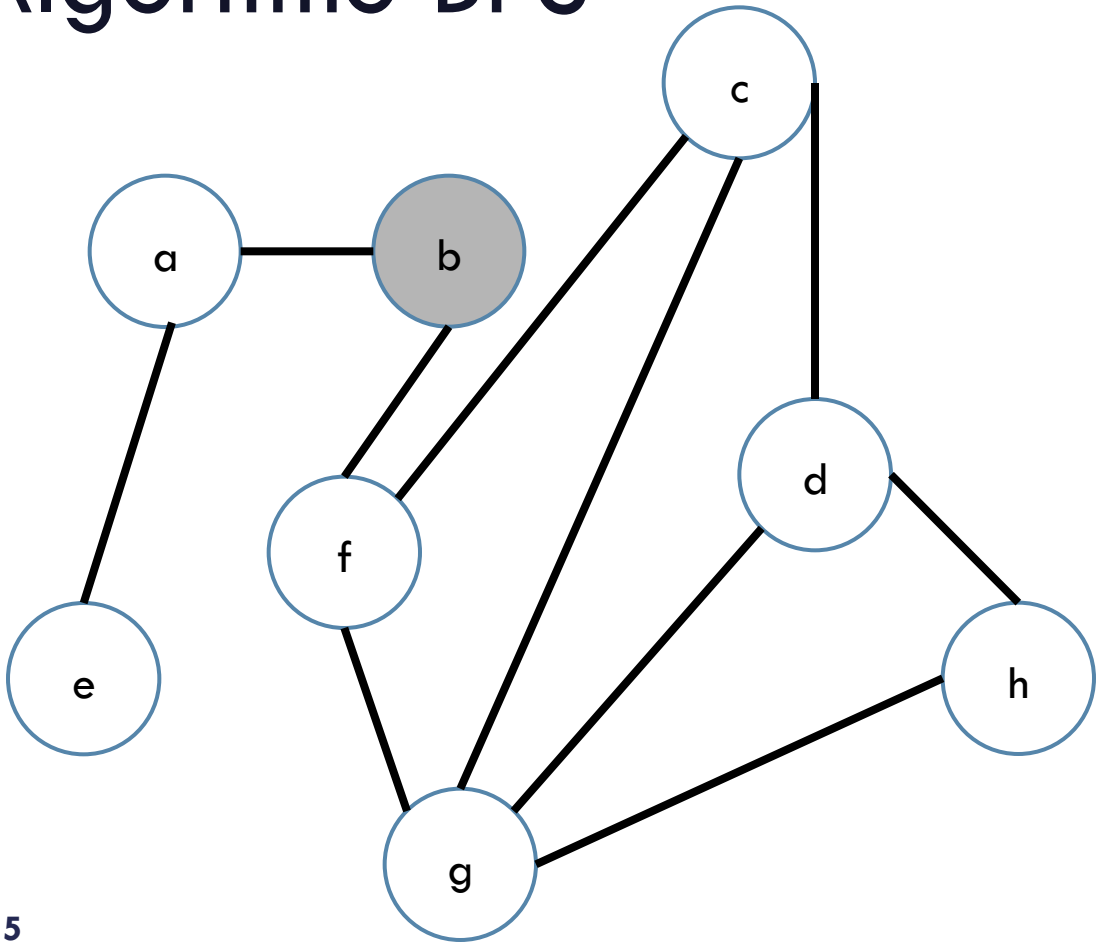
a	b	c	d	e	f	g	h
-	-	-	-	-	-	-	-

Pais

a	b	c	d	e	f	g	h
-	-	-	-	-	-	-	-

Fila Q

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
-	0	-	-	-	-	-	-

Pais

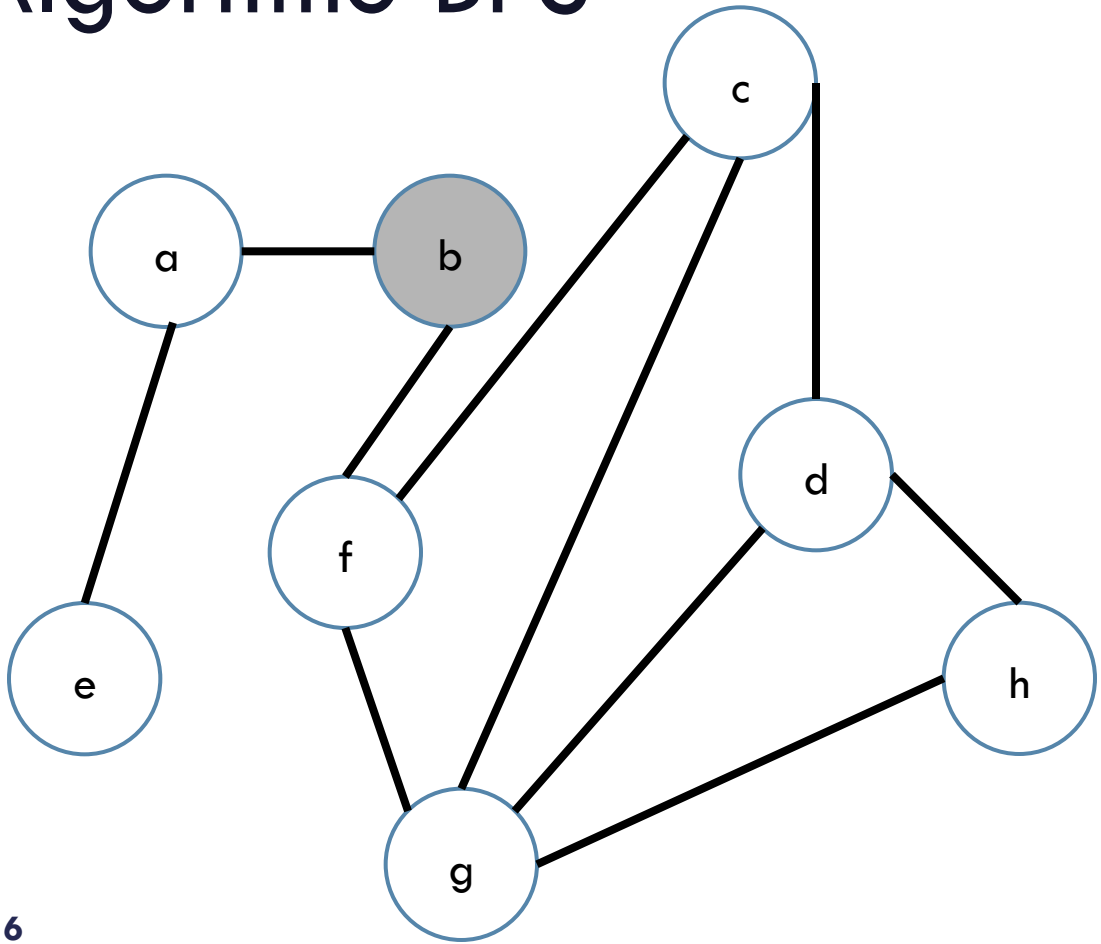
a	b	c	d	e	f	g	h
-	-	-	-	-	-	-	-

Fila Q

b							

Vértice:

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
-	0	-	-	-	-	-	-

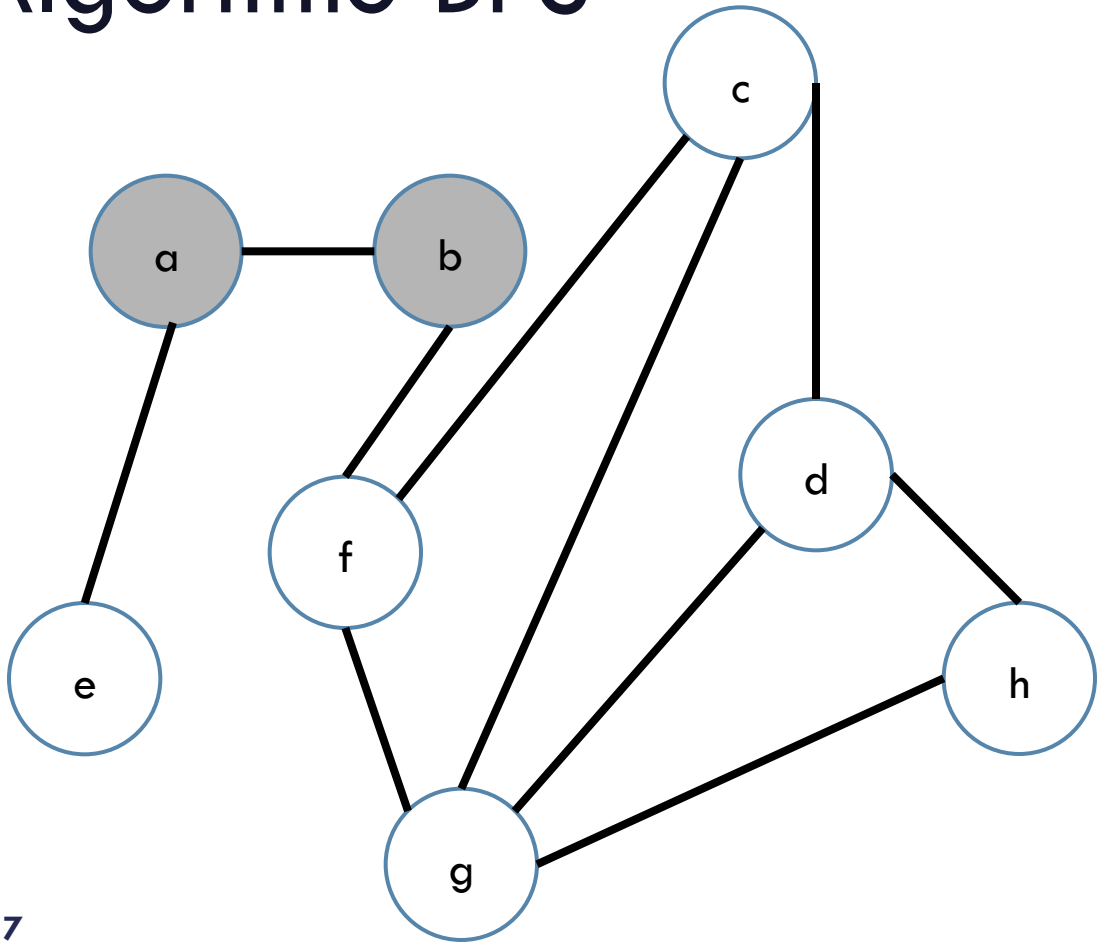
Pais

a	b	c	d	e	f	g	h
-	-	-	-	-	-	-	-

Fila Q

Vértice: b

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	-	-	-	-

Pais

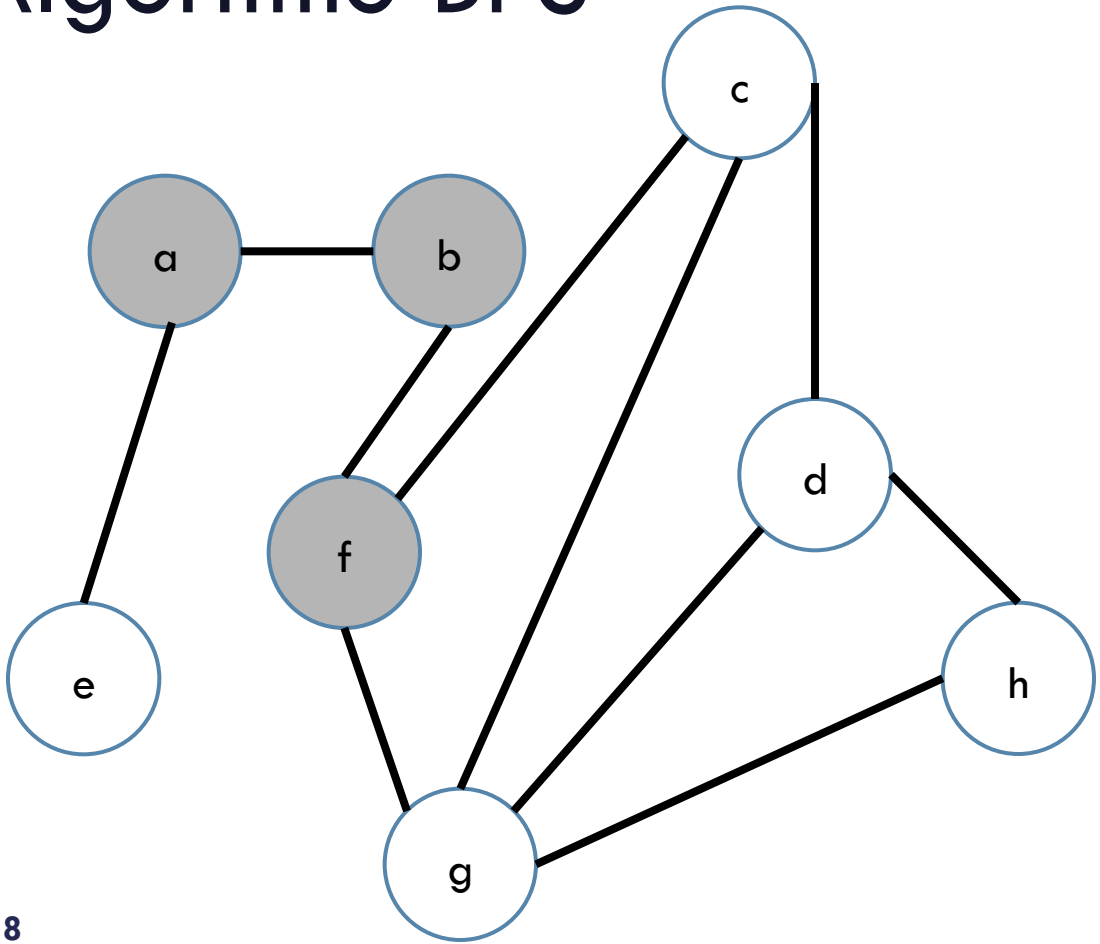
a	b	c	d	e	f	g	h
b	-	-	-	-	-	-	-

Fila Q

a							

Vértice: b

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	-	1	-	-

Pais

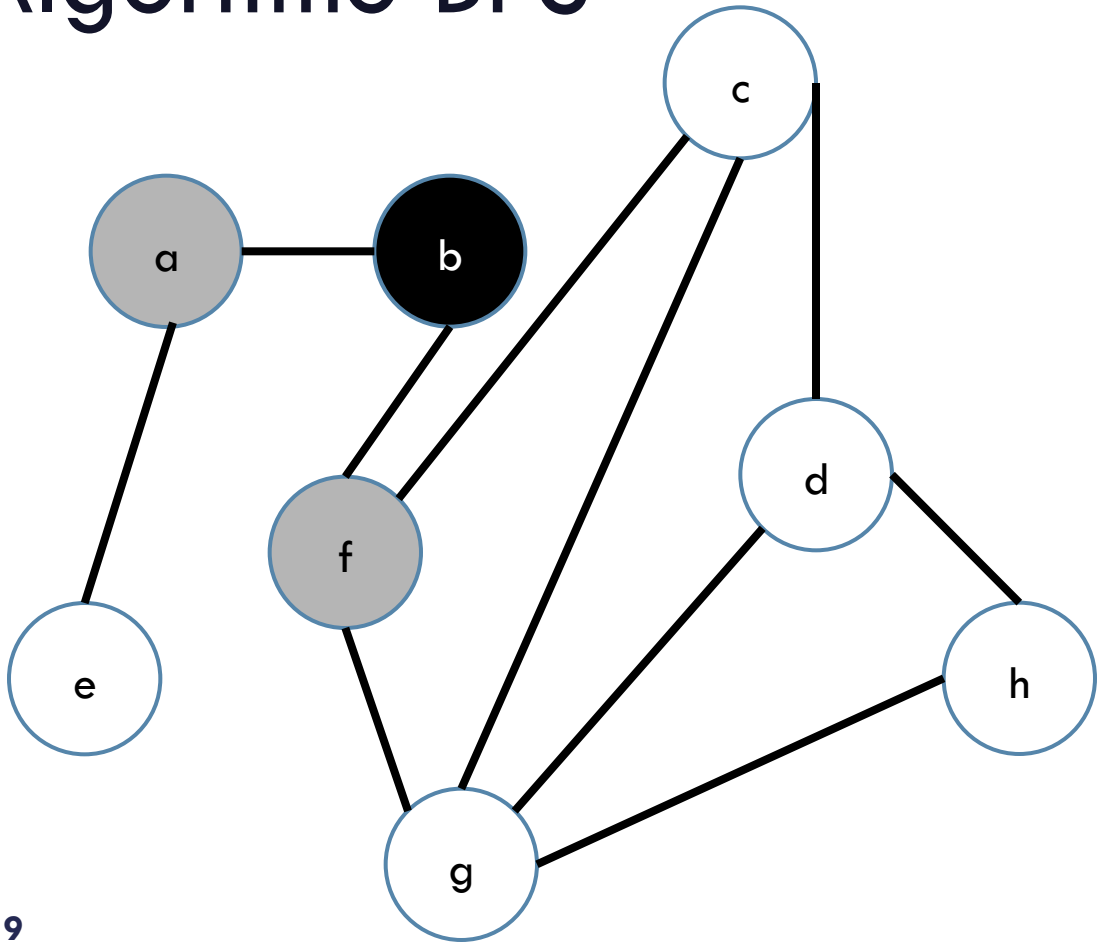
a	b	c	d	e	f	g	h
b	-	-	-	-	b	-	-

Fila Q

a	f						

Vértice: b

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	-	1	-	-

Pais

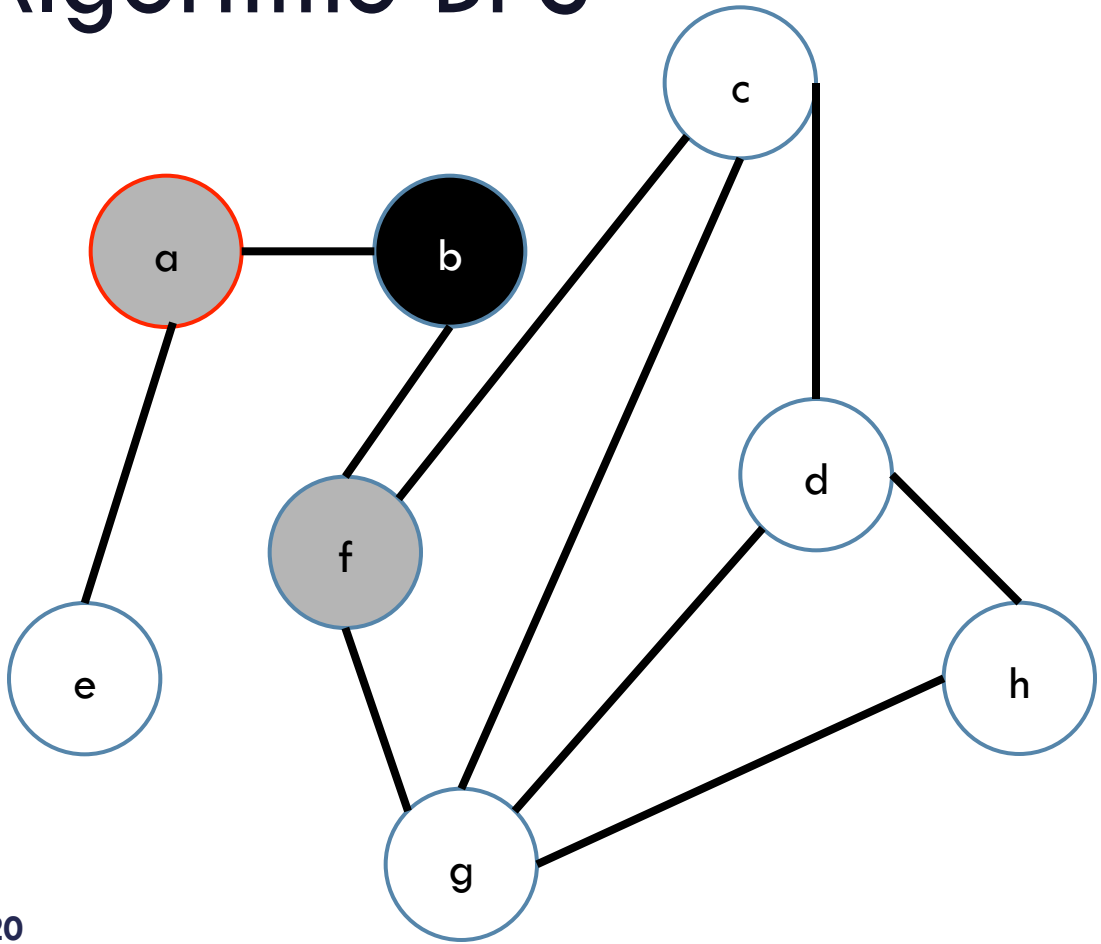
a	b	c	d	e	f	g	h
b	-	-	-	-	b	-	-

Fila Q

a	f						

Vértice: b

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	-	1	-	-

Pais

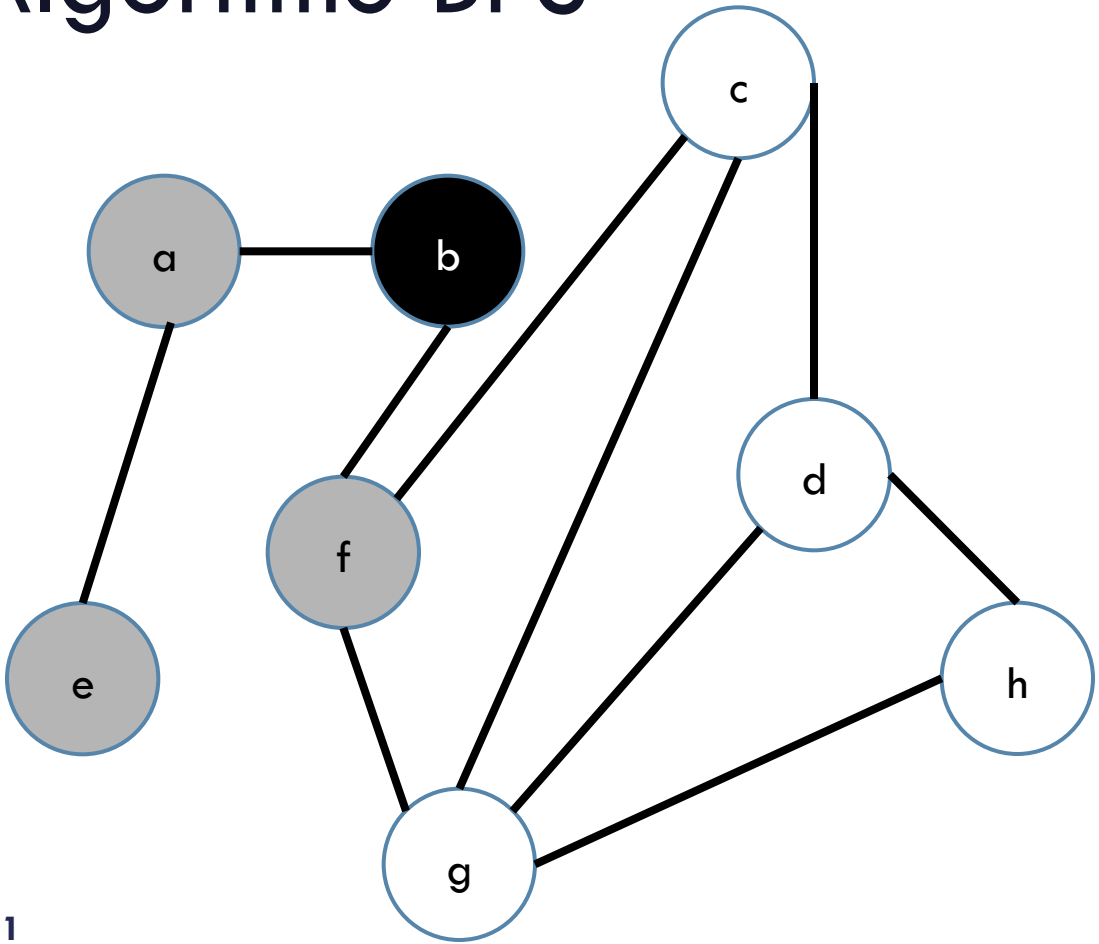
a	b	c	d	e	f	g	h
b	-	-	-	-	b	-	-

Fila Q

f							

Vértice: a

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	2	1	-	-

Pais

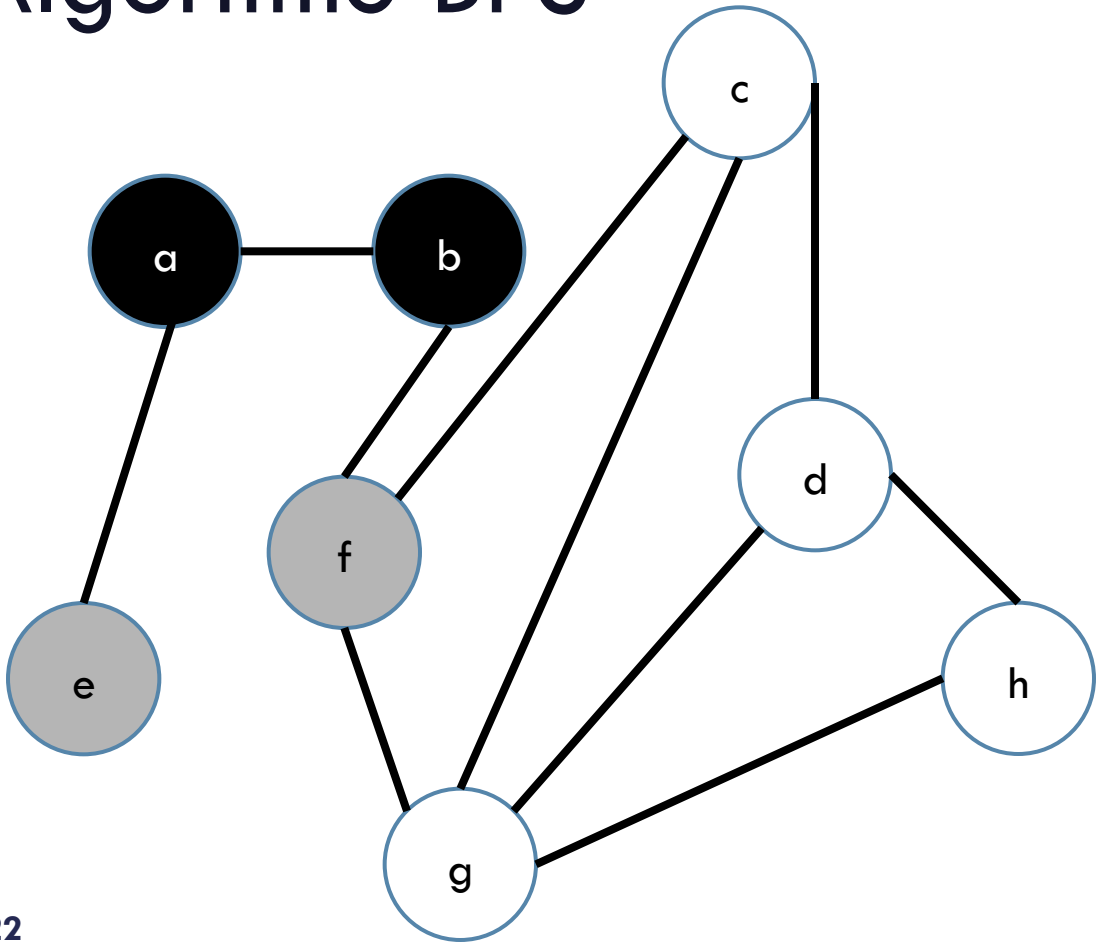
a	b	c	d	e	f	g	h
b	-	-	-	a	b	-	-

Fila Q

f	e						

Vértice: a

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	2	1	-	-

Pais

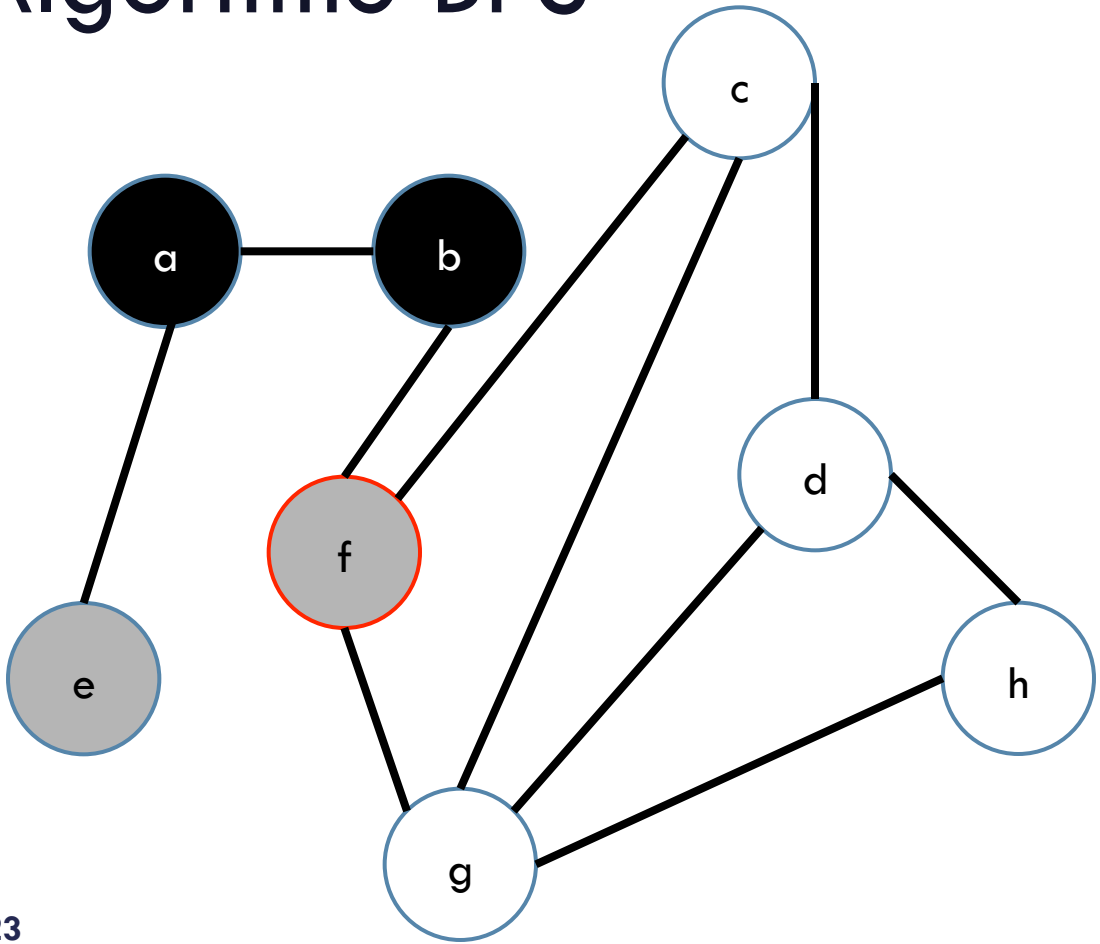
a	b	c	d	e	f	g	h
b	-	-	-	a	b	-	-

Fila Q

f	e						

Vértice: a

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	2	1	-	-

Pais

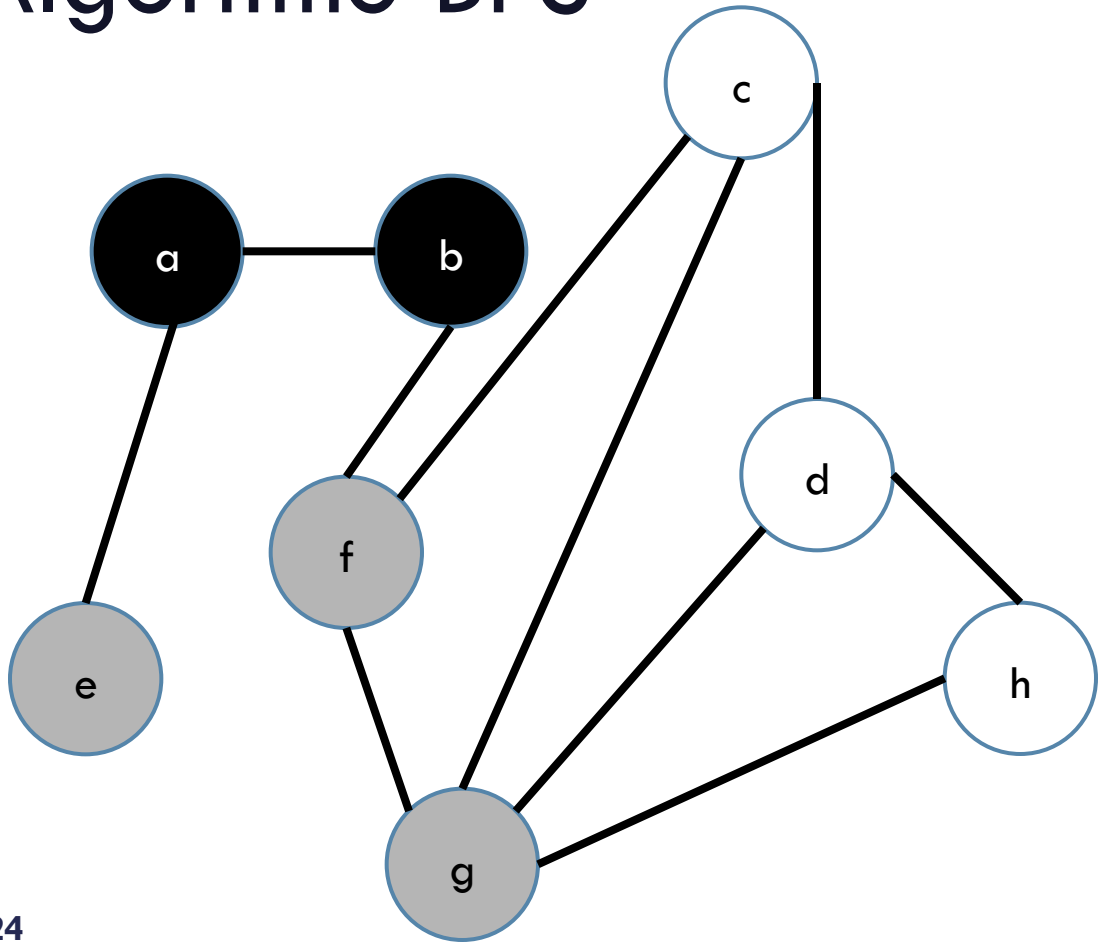
a	b	c	d	e	f	g	h
b	-	-	-	a	b	-	-

Fila Q

e							

Vértice: f

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	-	-	2	1	2	-

Pais

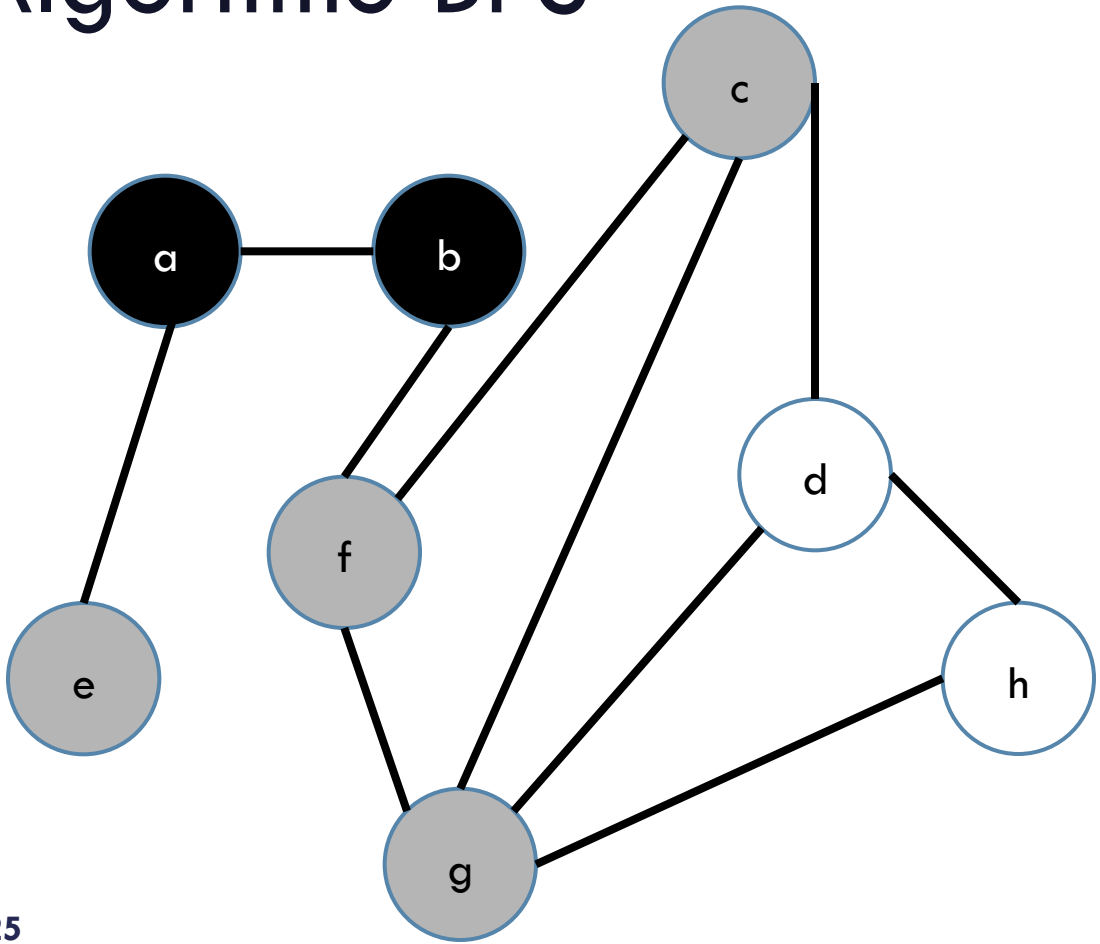
a	b	c	d	e	f	g	h
b	-	-	-	a	b	f	-

Fila Q

e	g						

Vértice: f

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	-	2	1	2	-

Pais

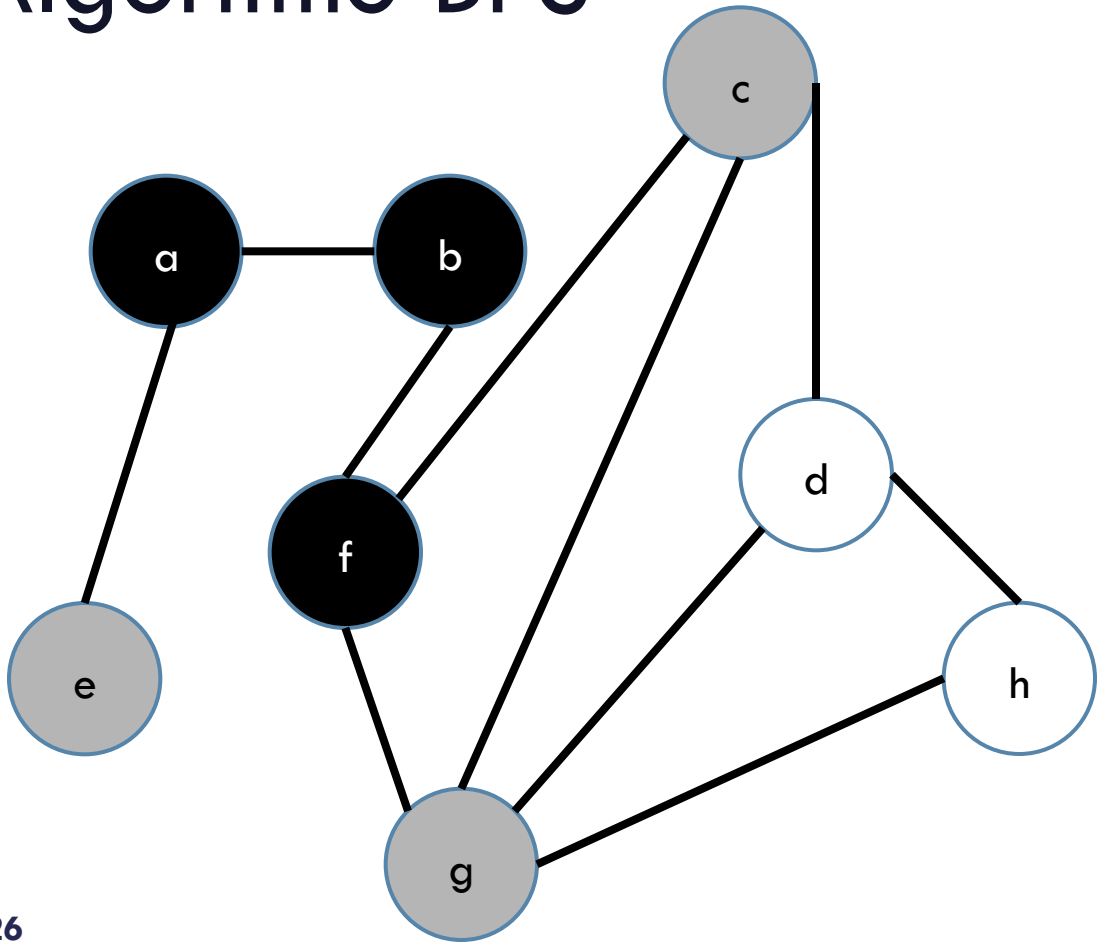
a	b	c	d	e	f	g	h
b	-	f	-	a	b	f	-

Fila Q

e	g	c					

Vértice: f

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	-	2	1	2	-

Pais

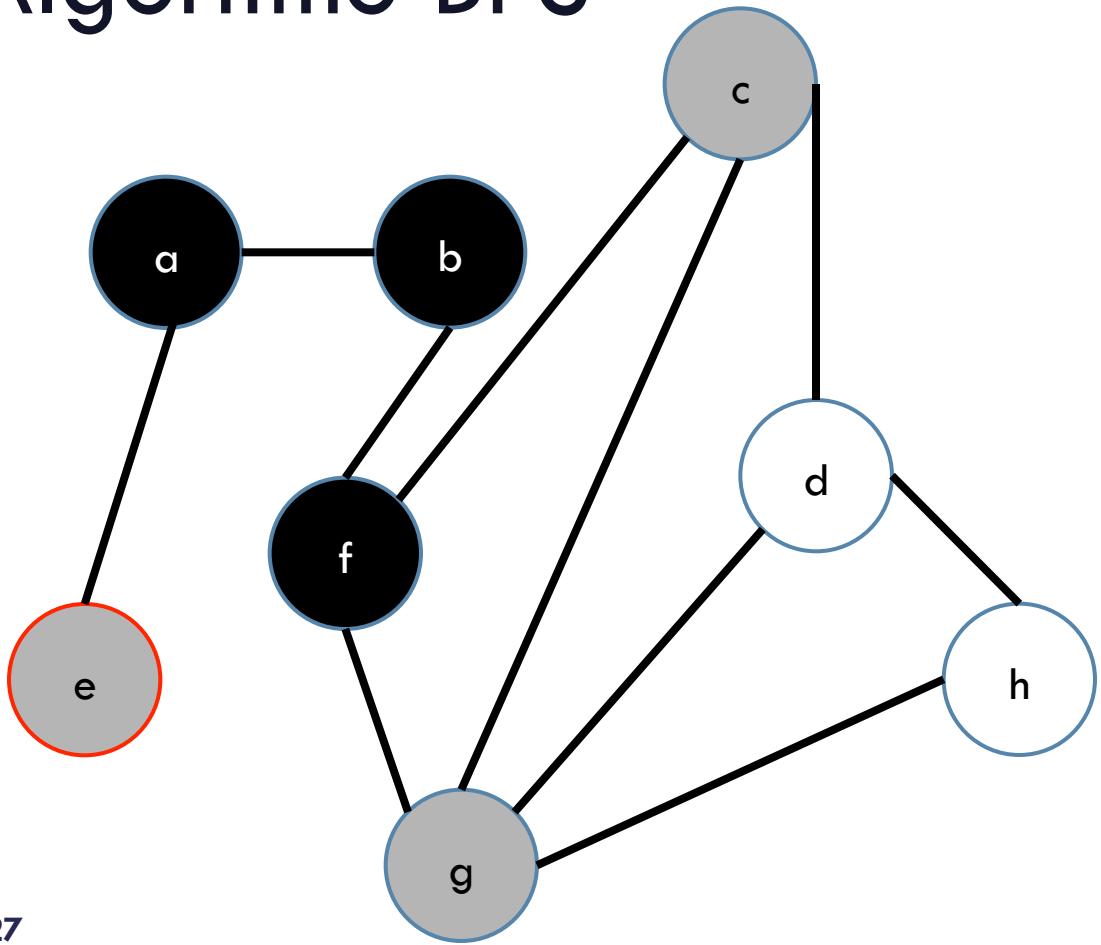
a	b	c	d	e	f	g	h
b	-	f	-	a	b	f	-

Fila Q

e	g	c					

Vértice: f

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	-	2	1	2	-

Pais

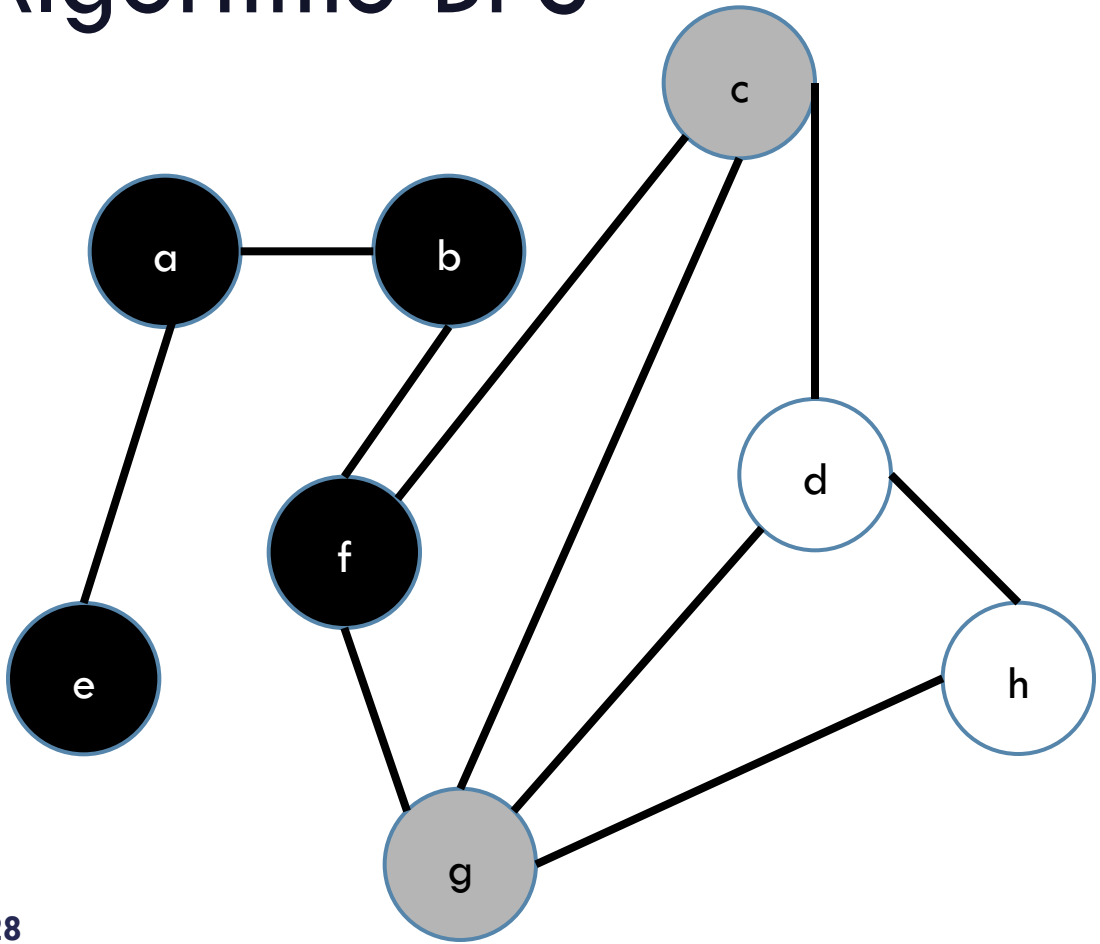
a	b	c	d	e	f	g	h
b	-	f	-	a	b	f	-

Fila Q

g	c						

Vértice: e

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	-	2	1	2	-

Pais

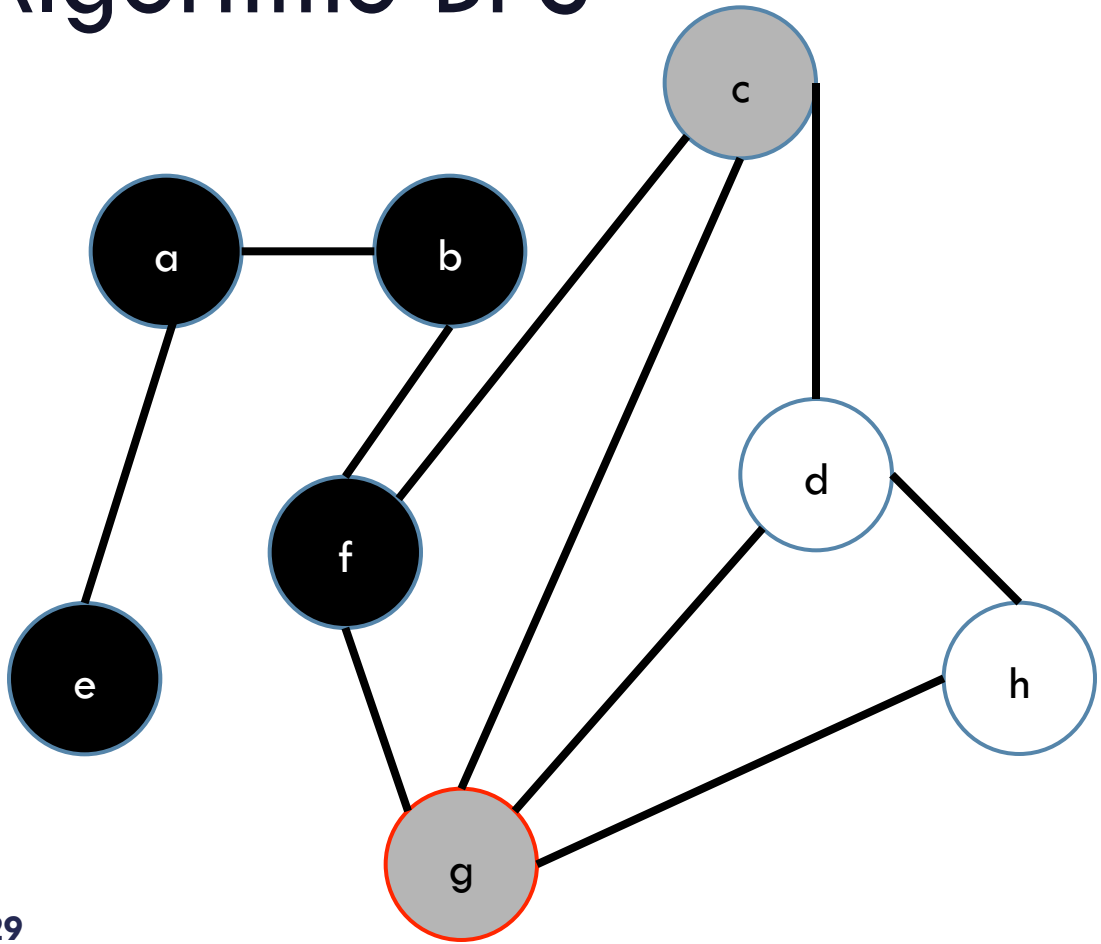
a	b	c	d	e	f	g	h
b	-	f	-	a	b	f	-

Fila Q

g	c						

Vértice: e

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	-	2	1	2	-

Pais

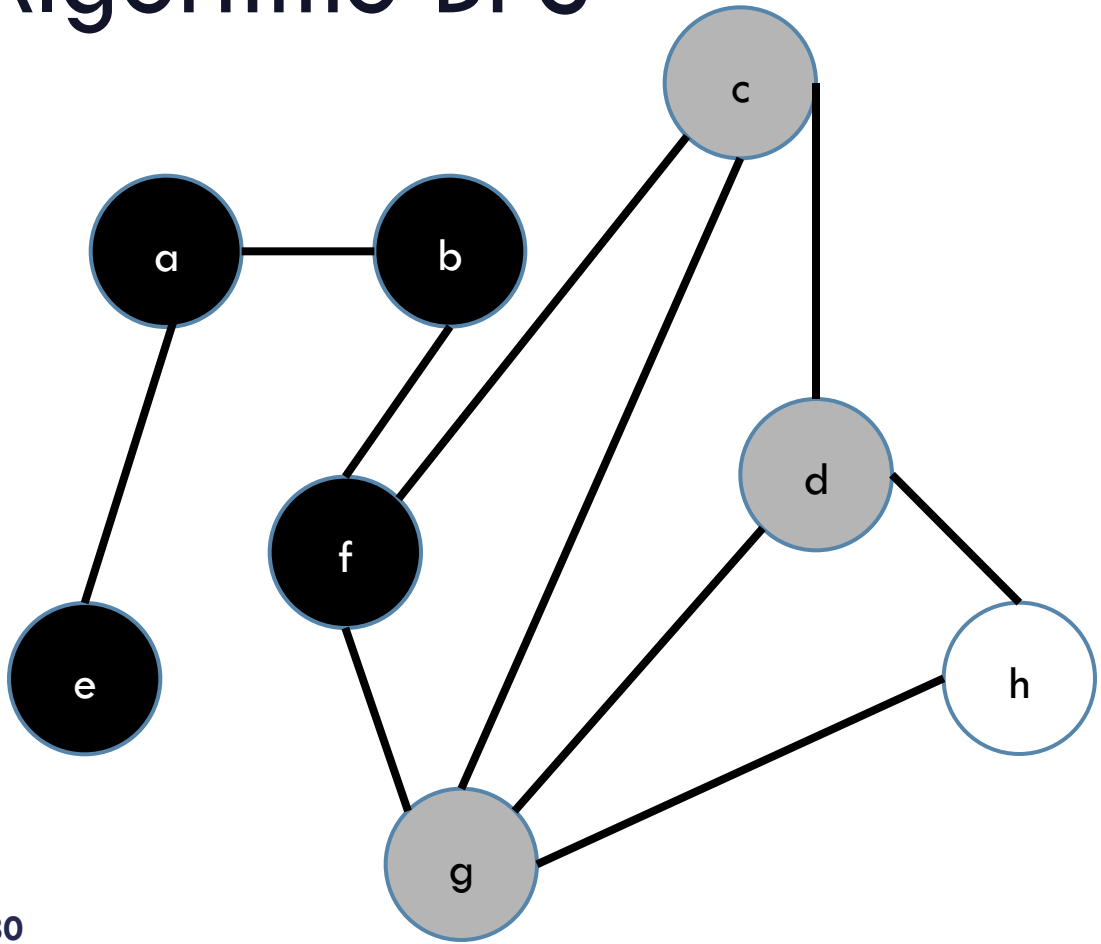
a	b	c	d	e	f	g	h
b	-	f	-	a	b	f	-

Fila Q

c							

Vértice: g

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	-

Pais

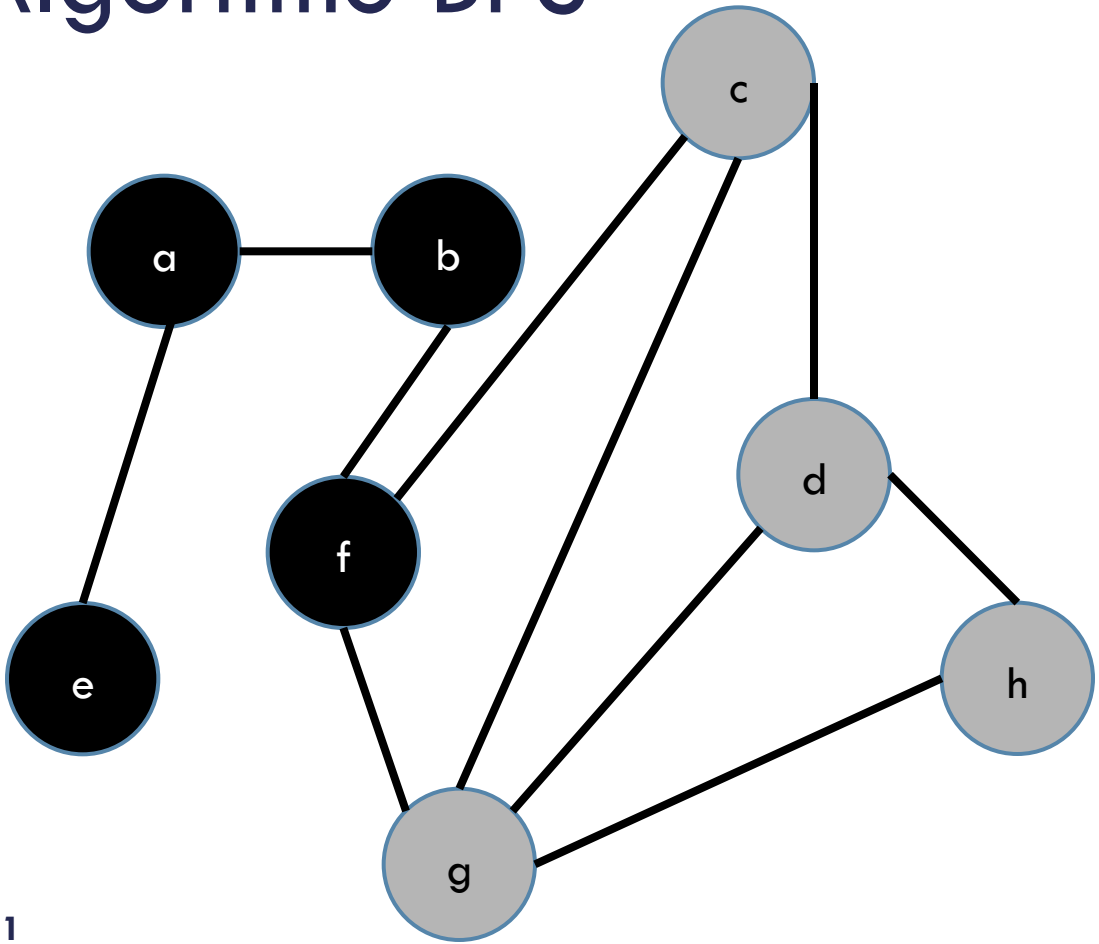
a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	-

Fila Q

c	d						

Vértice: g

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

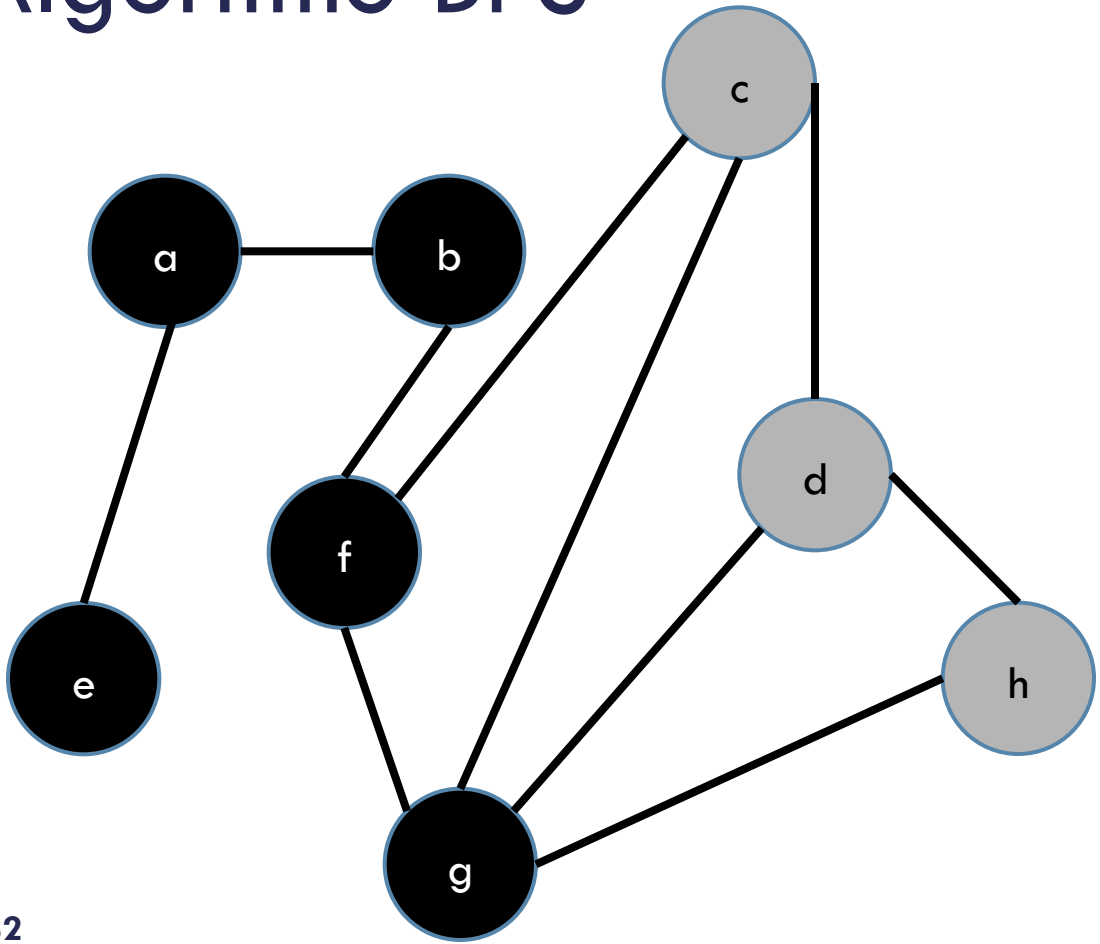
a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Fila Q

c	d	h					

Vértice: g

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

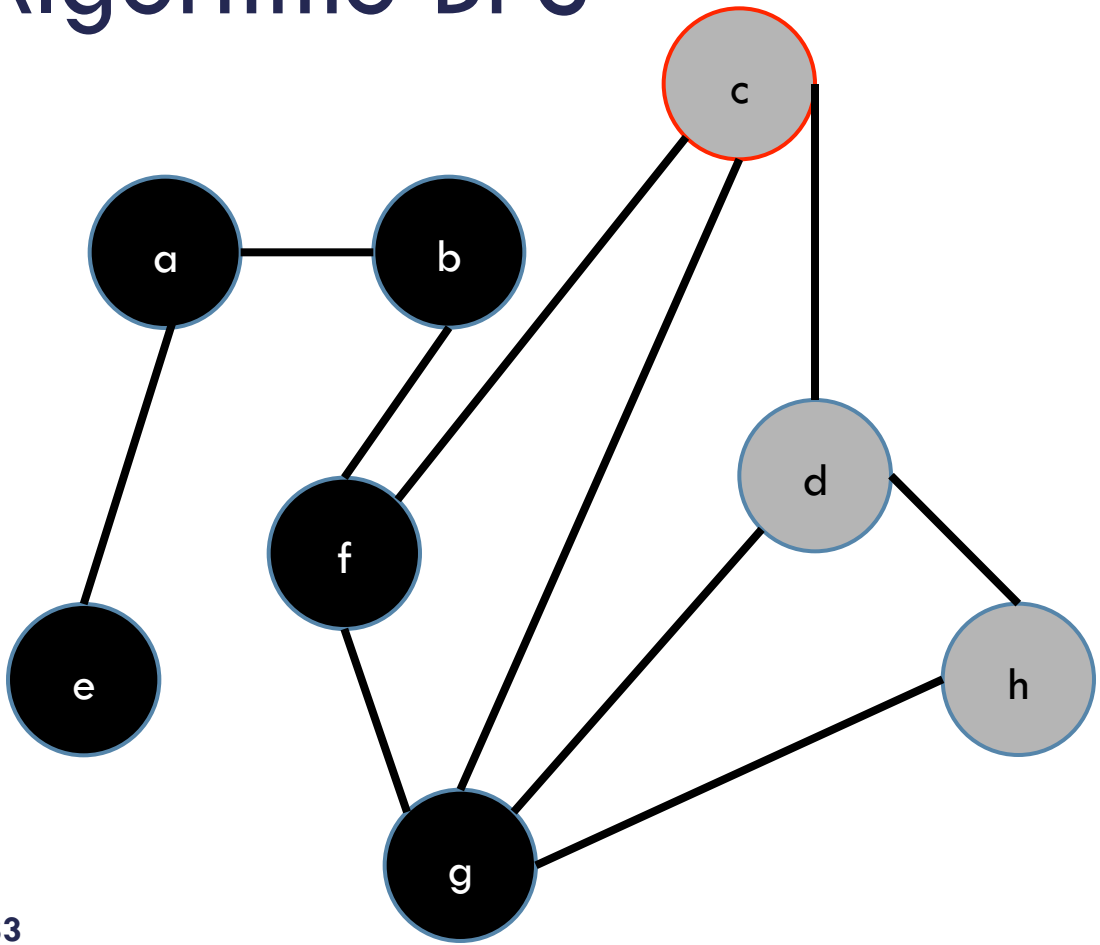
a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Fila Q

c	d	h					

Vértice:

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

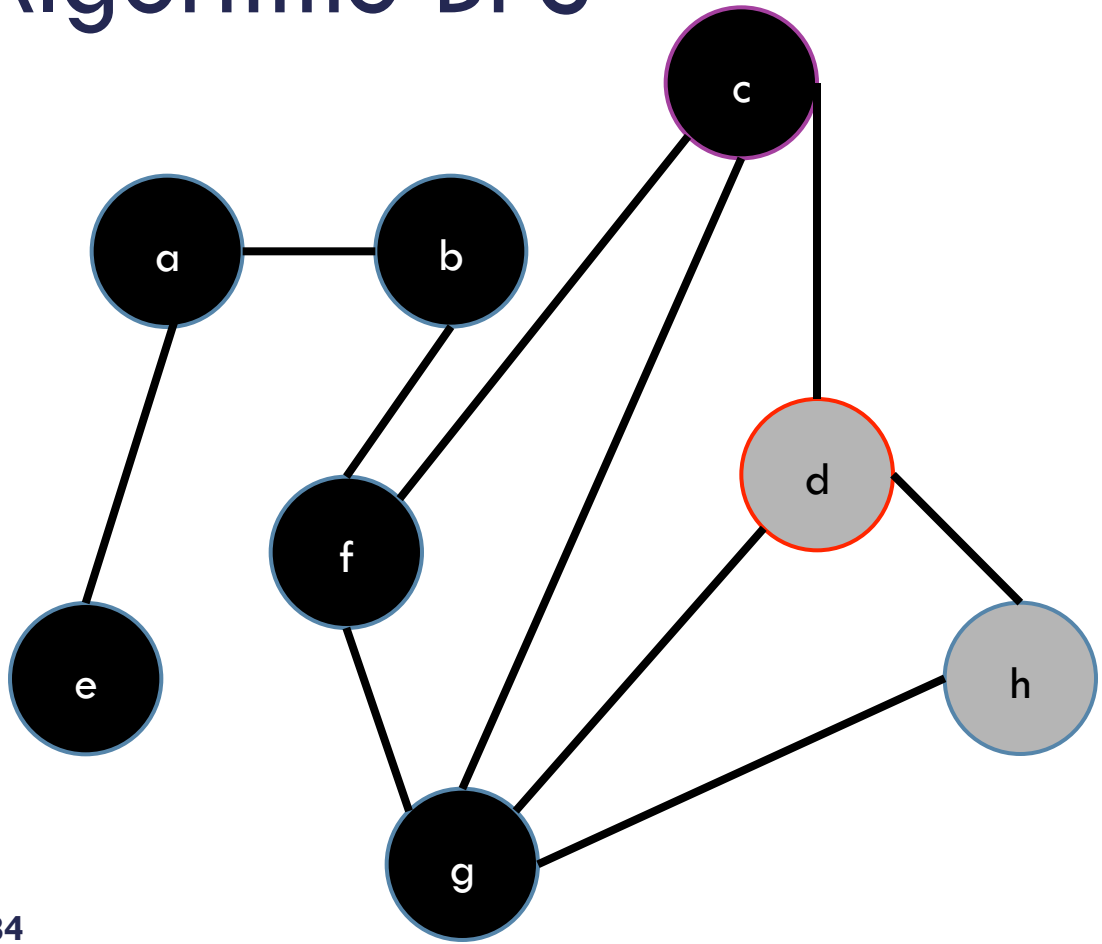
a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Fila Q

d	h						

Vértice: c

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

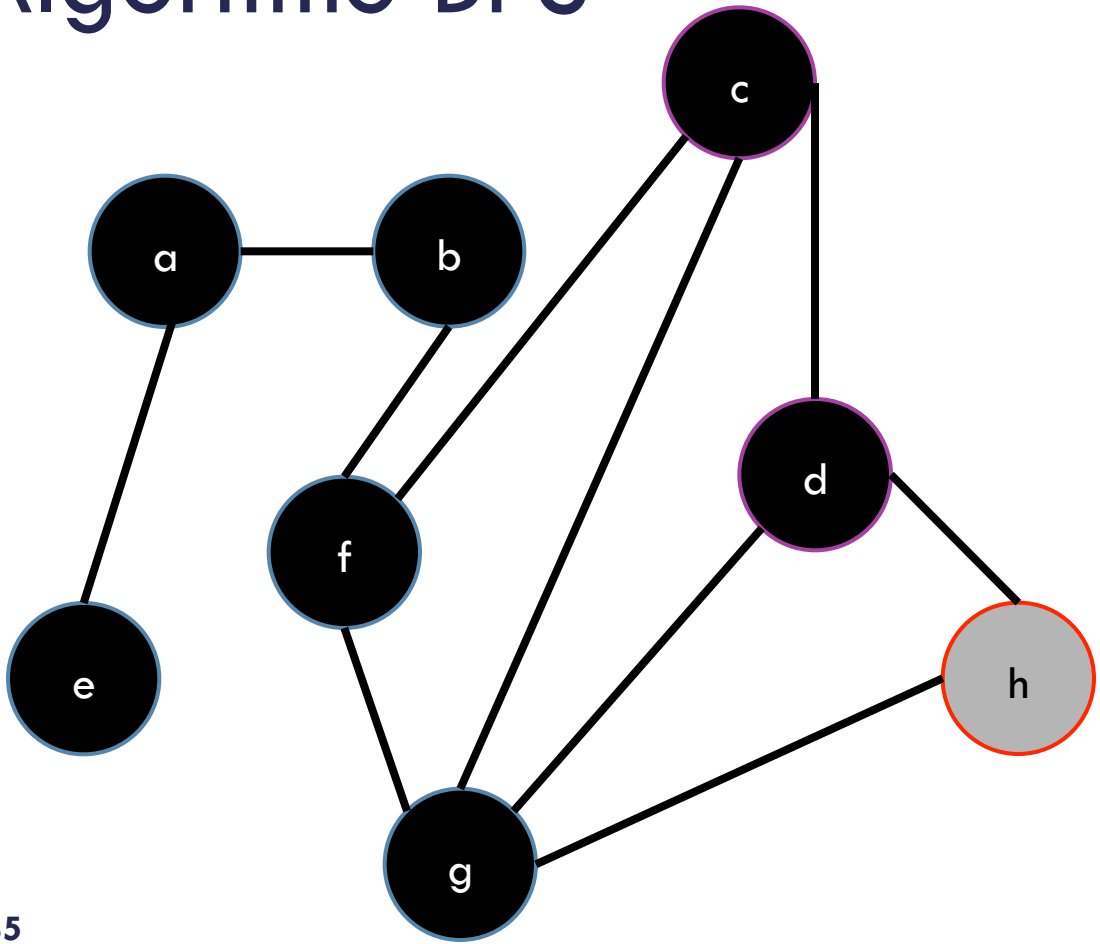
a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Fila Q

h							

Vértice: d

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

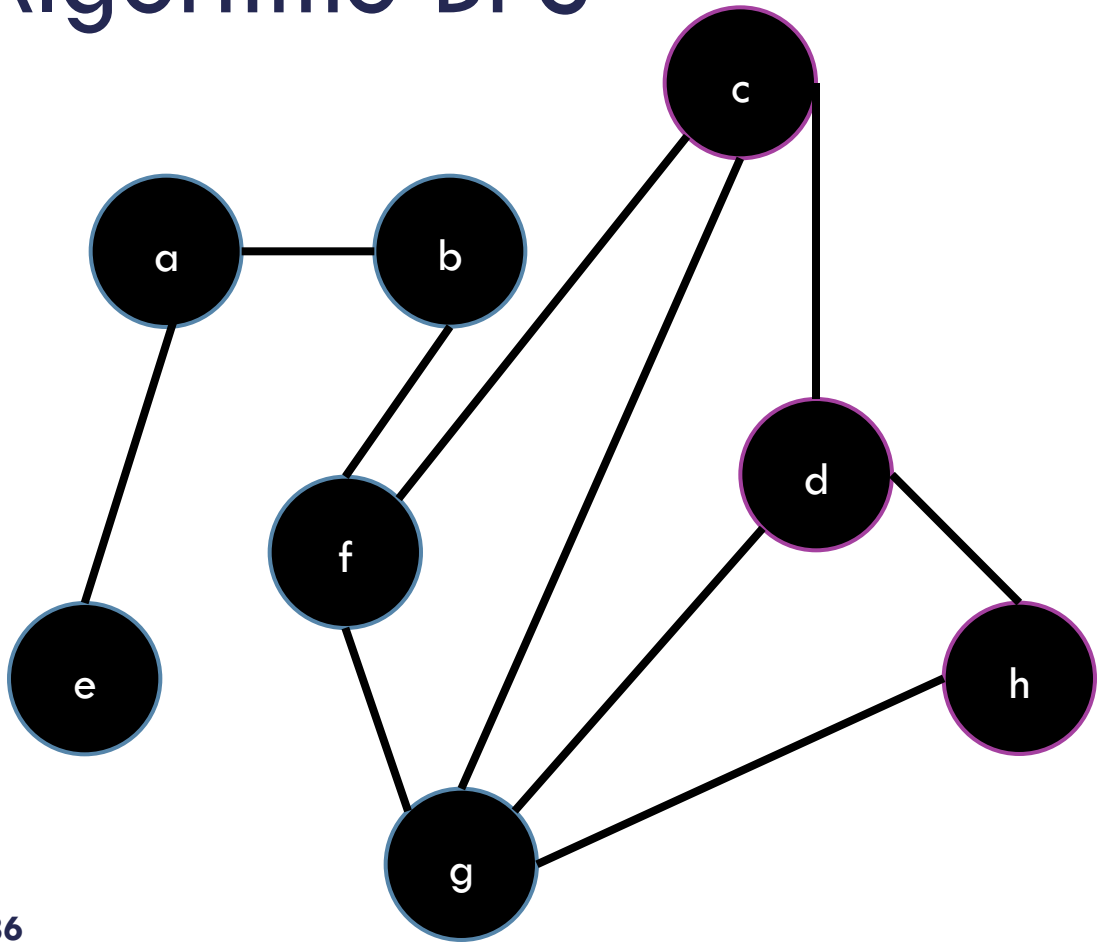
Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Fila Q

Vértice: h

Algoritmo BFS



Distâncias

a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

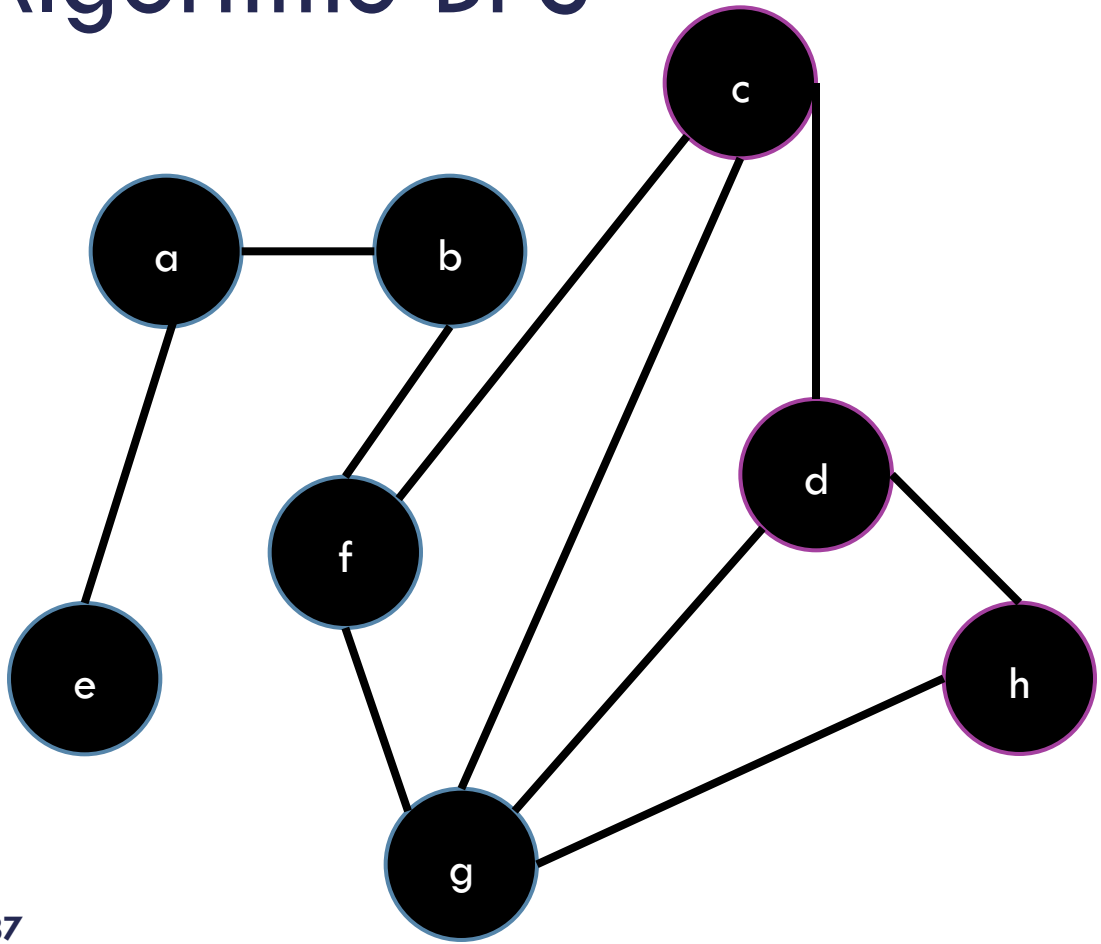
Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Fila Q

Fila vazia: fim

Algoritmo BFS



Distâncias

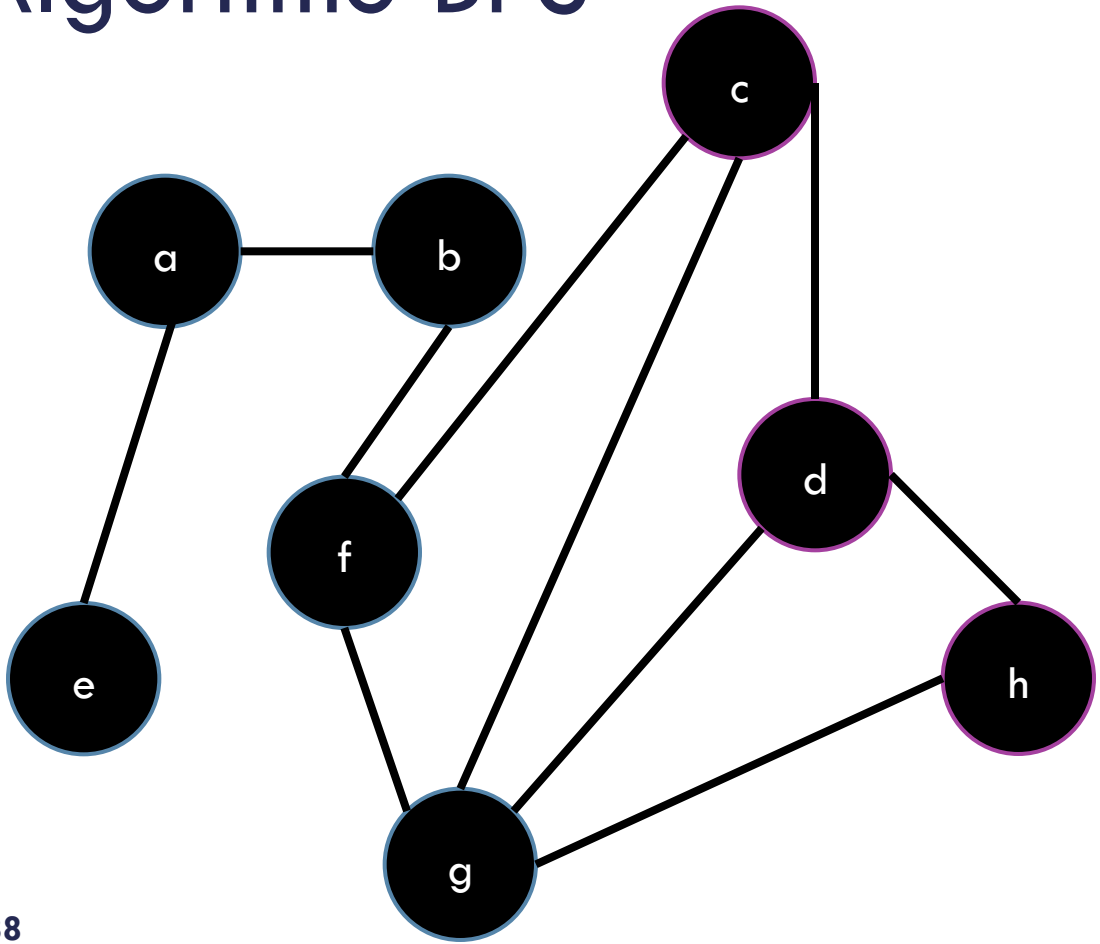
a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Caminho até o vértice h:
h

Algoritmo BFS



Distâncias

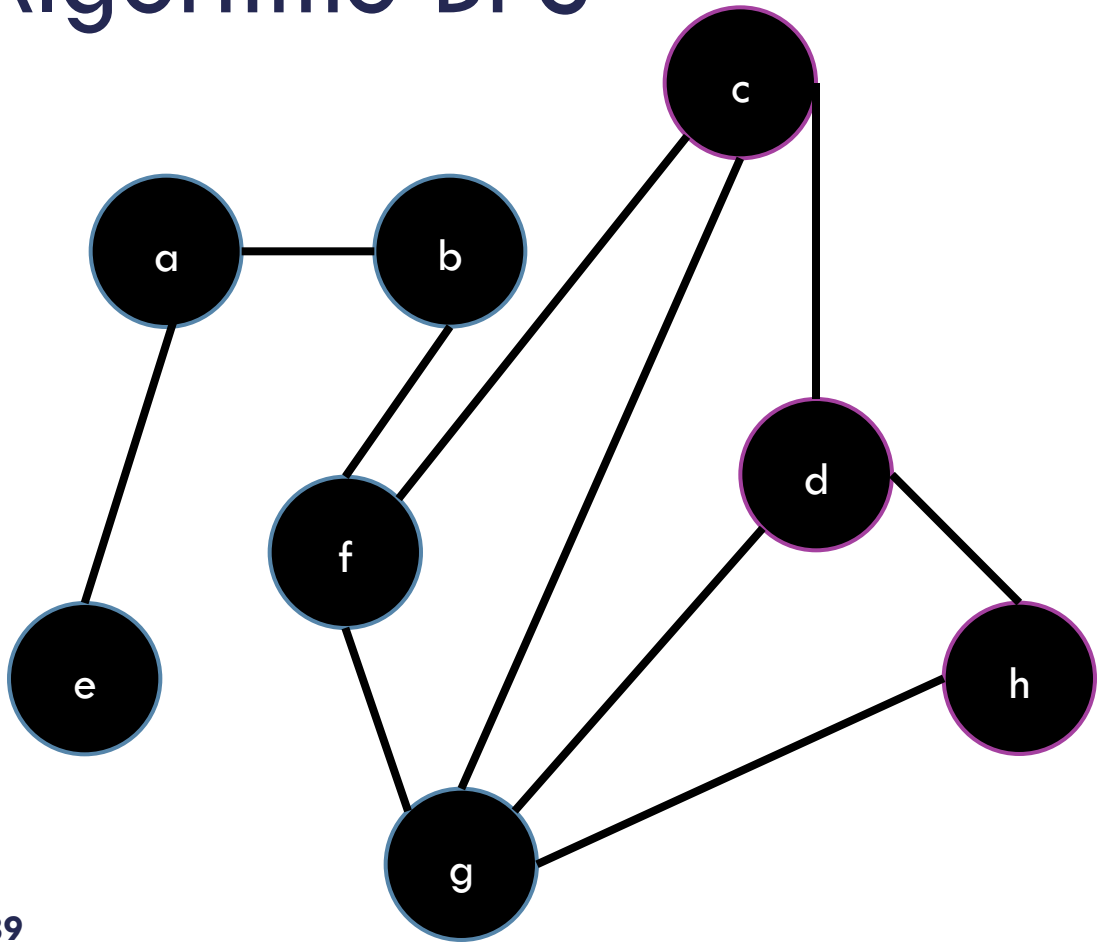
a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Caminho até o vértice h:
g-h

Algoritmo BFS



Distâncias

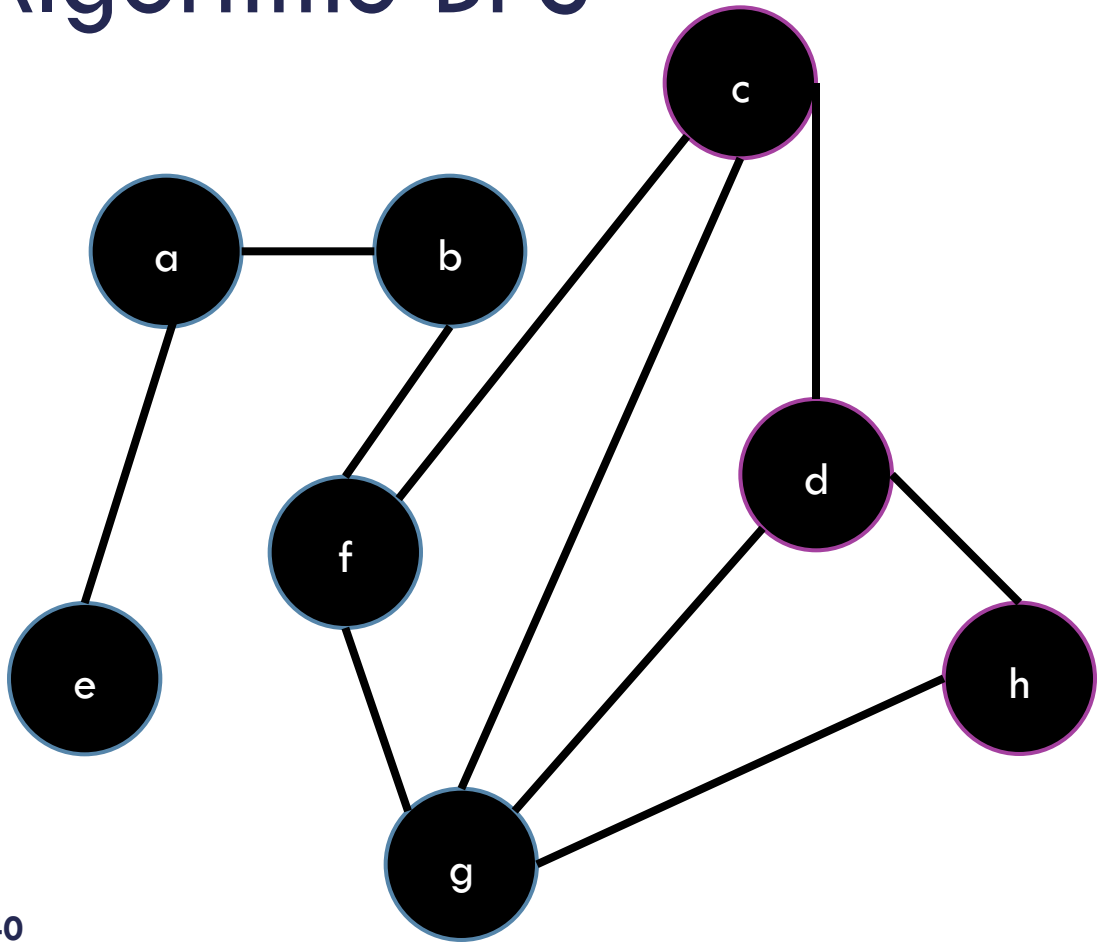
a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Caminho até o vértice h:
f-g-h

Algoritmo BFS



Distâncias

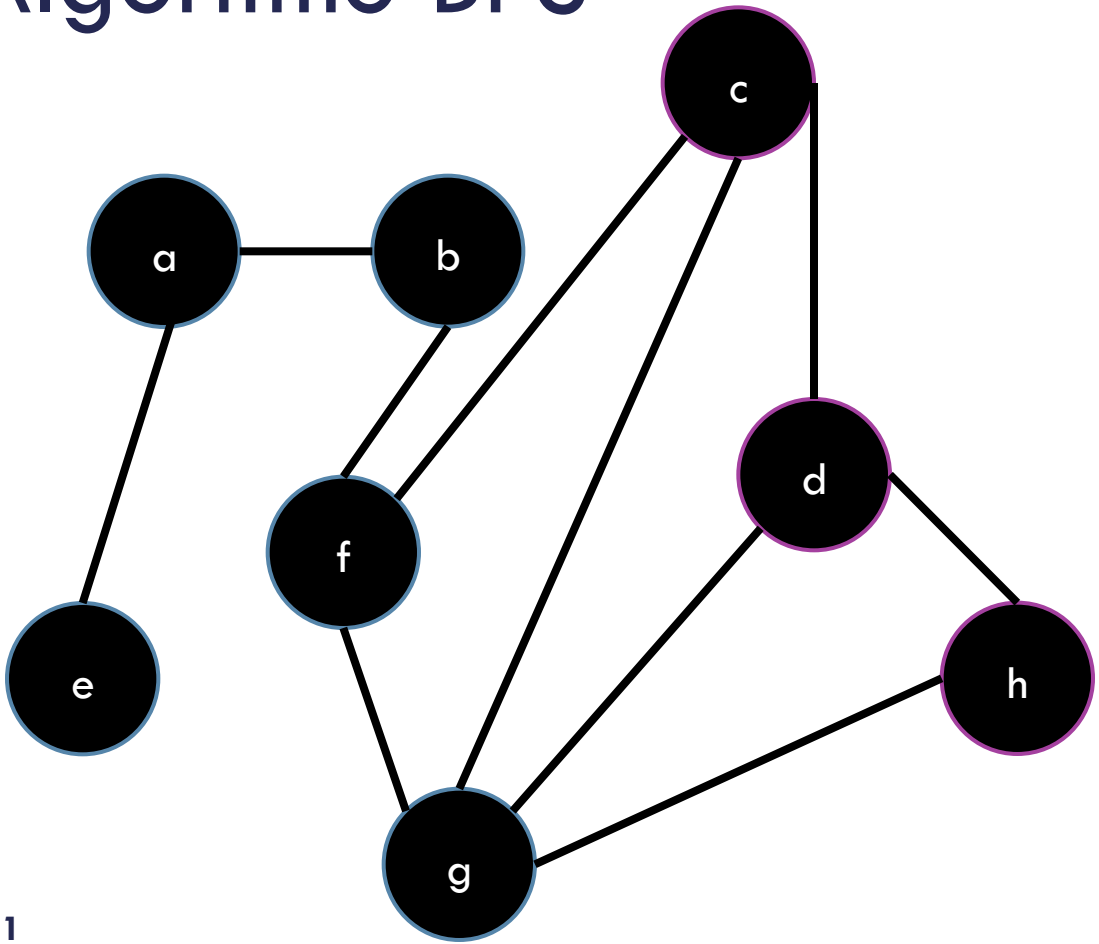
a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Caminho até o vértice h:
b-f-g-h

Algoritmo BFS



Distâncias

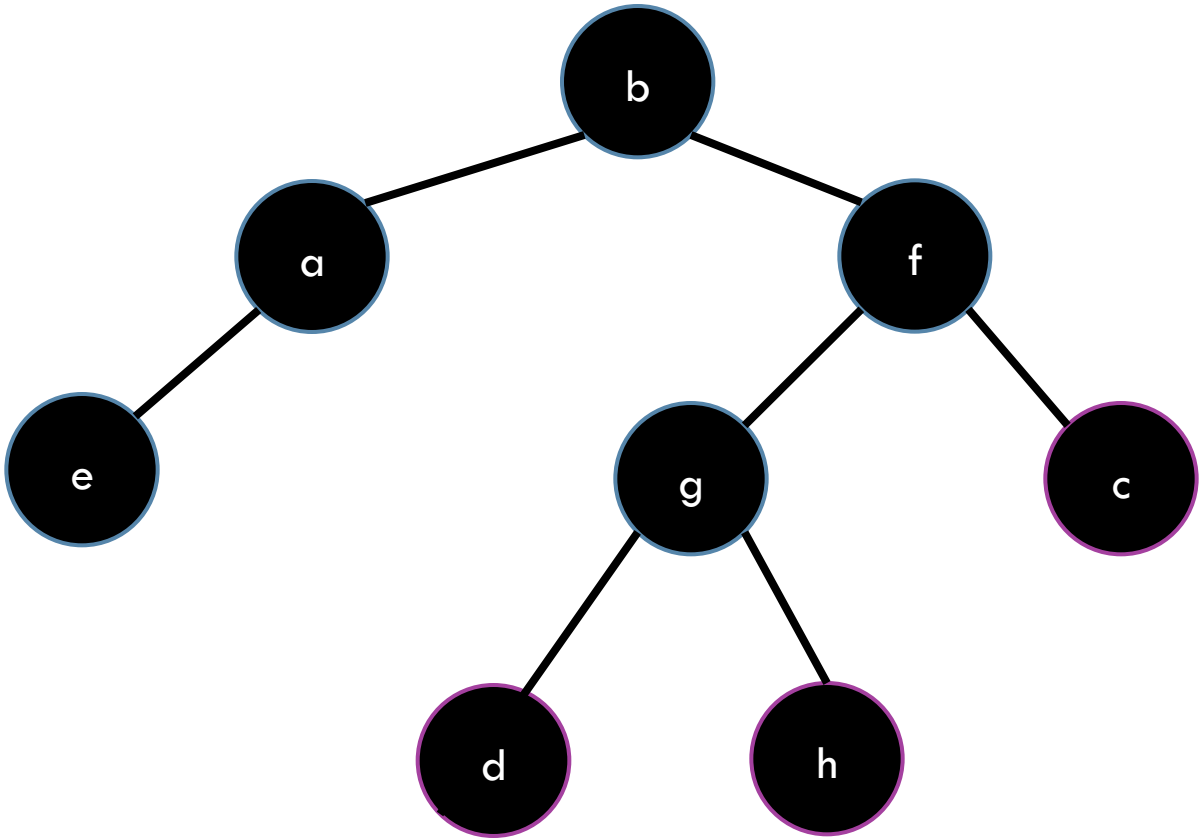
a	b	c	d	e	f	g	h
1	0	2	3	2	1	2	3

Pais

a	b	c	d	e	f	g	h
b	-	f	g	a	b	f	g

Caminho até o vértice h:
b-f-g-h

Árvore BFS



```
public class BuscaEmLargura    {
        public const byte branco = 0;
        public const byte cinza  = 1;
        public const byte preto  = 2;
        private int []d;
        private int [] antecessor;
        private Grafo grafo;
        public BuscaEmLargura (Grafo grafo) {
            this.grafo = grafo; int n = this.grafo.get_numVertices();
            this.d = new int[n]; this.antecessor = new int[n];
        }
        public int get_d (int v) { return this.d[v]; }
        public int get_antecessor(int v) { return this.antecessor[v]; }

        public void imprimeCaminho (int origem, int v) {
            if (origem == v)
                Console.WriteLine (origem);
            else if (this.antecessor[v] == -1)
                Console.WriteLine ("Nao existe caminho de " + origem + " ate " + v);
            else {
                imprimeCaminho (origem, this.antecessor[v]);
                Console.WriteLine (v);
            }
        }
```

Busca em Largura

```
public void buscaEmLargura () {
        int []cor = new int[this.grafo.get_numVertices ()];

        for (int u = 0; u < grafo.get_numVertices(); u++)
        {
            cor[u] = branco; this.d[u] = Int32.MaxValue;
            this.antecessor[u] = -1;
        }
        for (int u = 0; u < grafo.get_numVertices(); u++)
        if (cor[u] == branco)
                this.visitaBfs (u, cor);
    }
```

Busca em Largura

```
public void buscaEmLargura () {
        int []cor = new int[this.grafo.get_numVertices ()];
          //inicialização
        for (int u = 0; u < grafo.get_numVertices(); u++)
        {
            cor[u] = branco; this.d[u] = Int32.MaxValue;
            this.antecessor[u] = -1;
        }
        for (int u = 0; u < grafo.get_numVertices(); u++)
        if (cor[u] == branco)
                this.visitaBfs (u, cor);
    }
```

Início da chamada da função visitaBfs

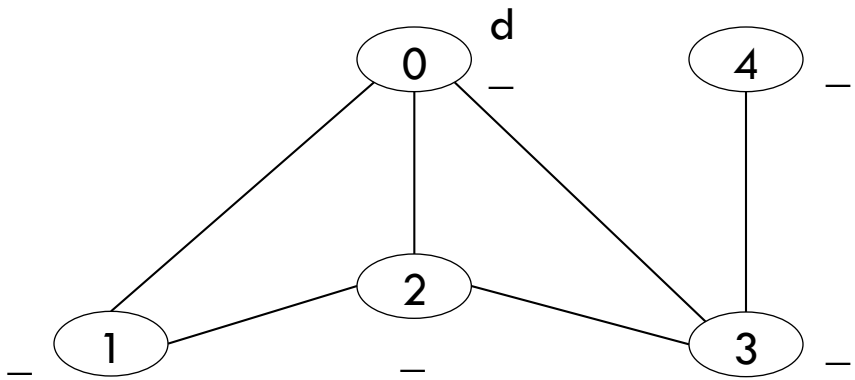
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila:

a =

v =



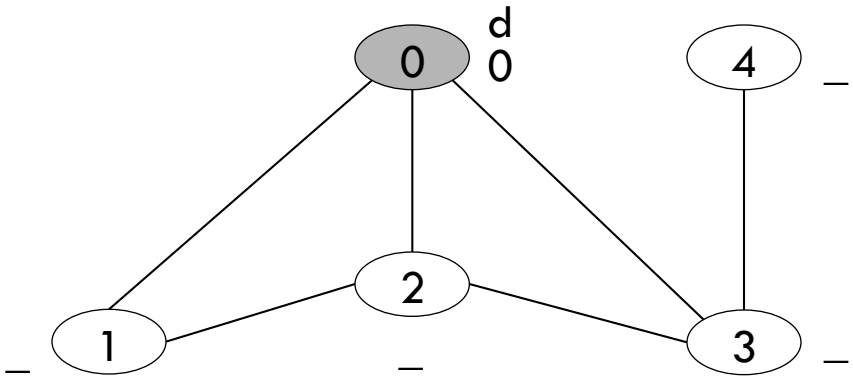
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
           int aux = (int)fila.desenfileira (); u = aux;
           if (!this.grafo.listaAdjVazia (u))
           {
              Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
              while (a != null)
              {
                 int v = a.get_v2();
                 if (cor[v] == branco) {
                    cor[v] = cinza; this.d[v] = this.d[u] + 1;
                    this.antecessor[v] = u; fila.enfileira (v);
                 }
                 a = this.grafo.proxAdj (u);
              }
           }
           cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila:

a =

v =



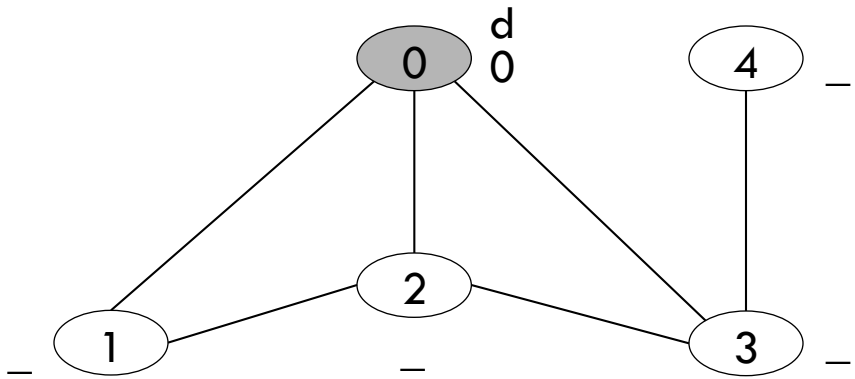
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 0 -

a =

v =



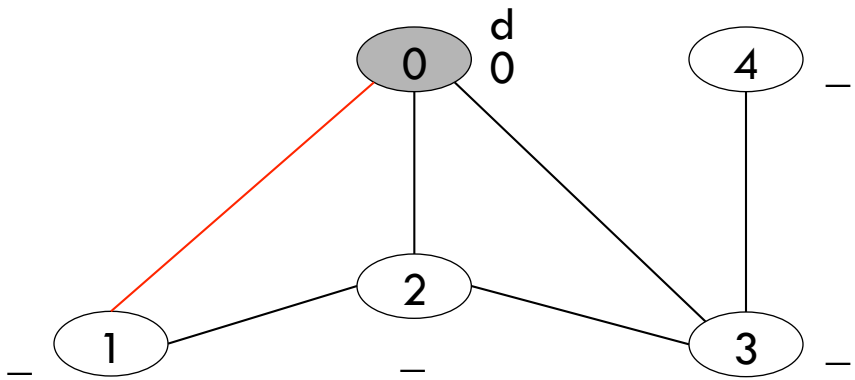
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila:

a = (0 - 1)

v =



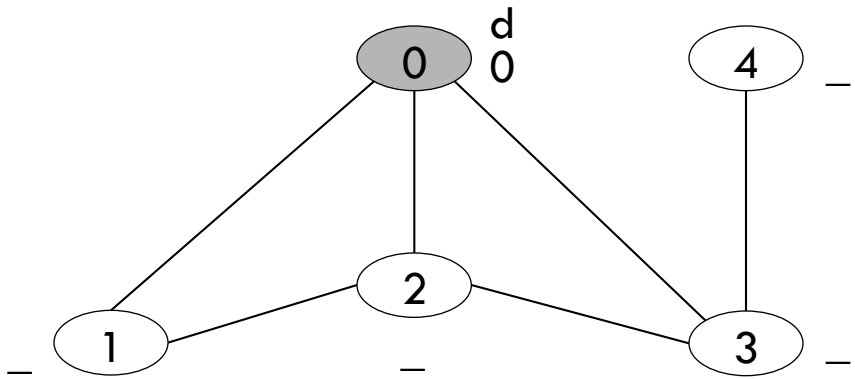
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila:

a = (0 - 1)

v = 1



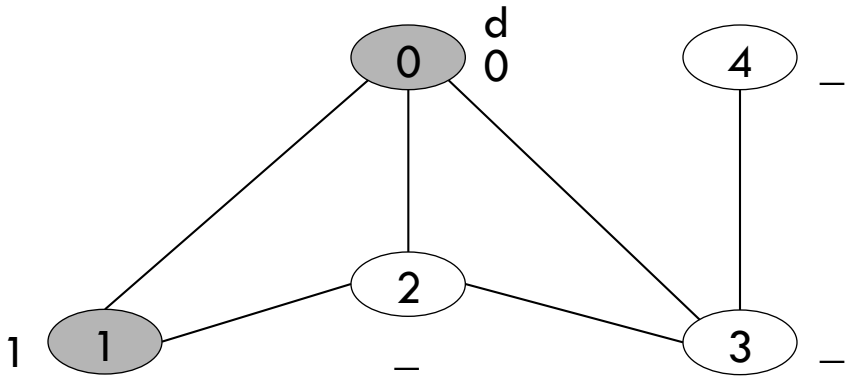
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 -

a = (0 - 1)

v = 1



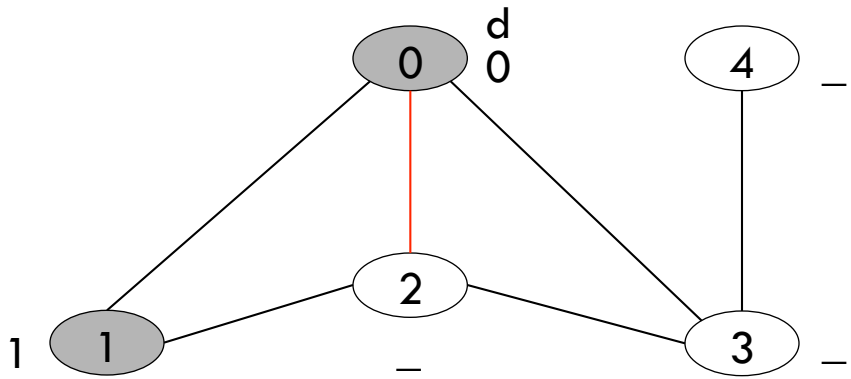
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 -

a = (0 - 2)

v = 1



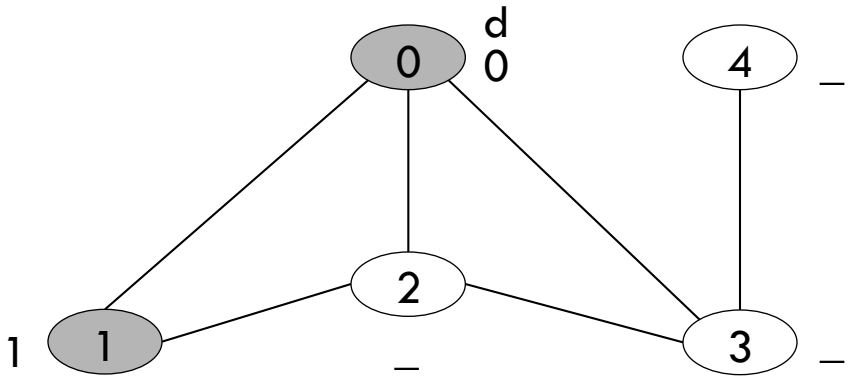
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 -

a = (0 - 2)

v = 2



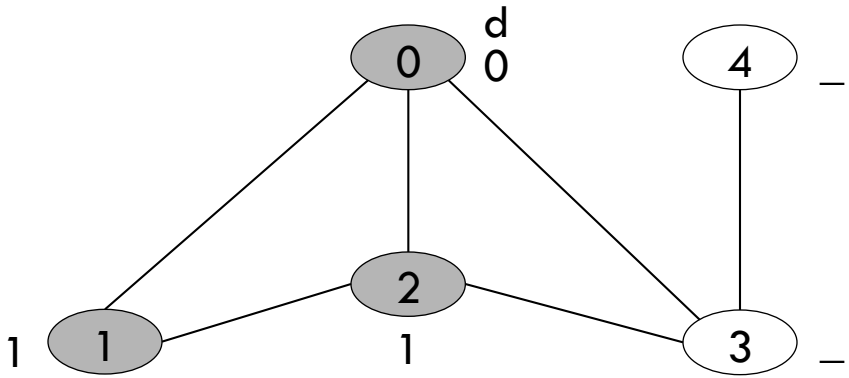
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 -

a = (0 - 2)

v = 2



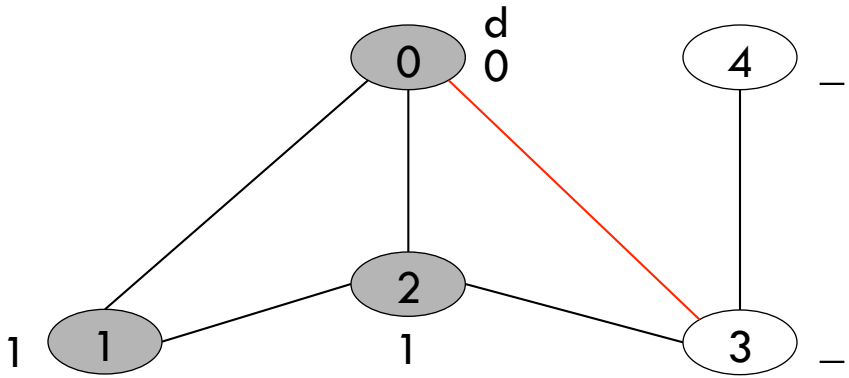
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 -

a = (0 - 3)

v = 2



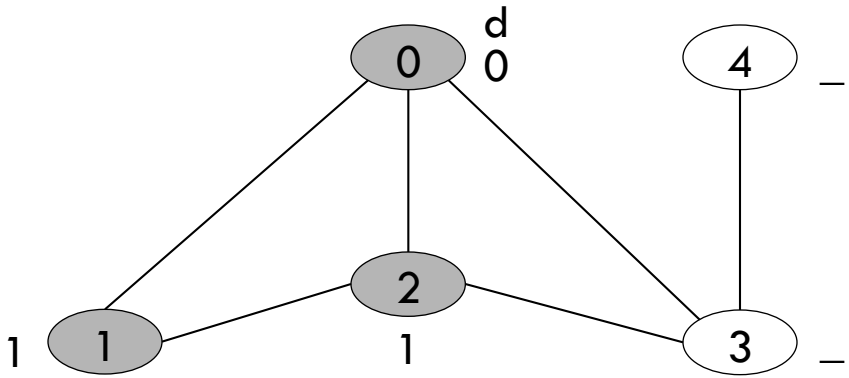
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 -

a = (0 - 3)

v = 3



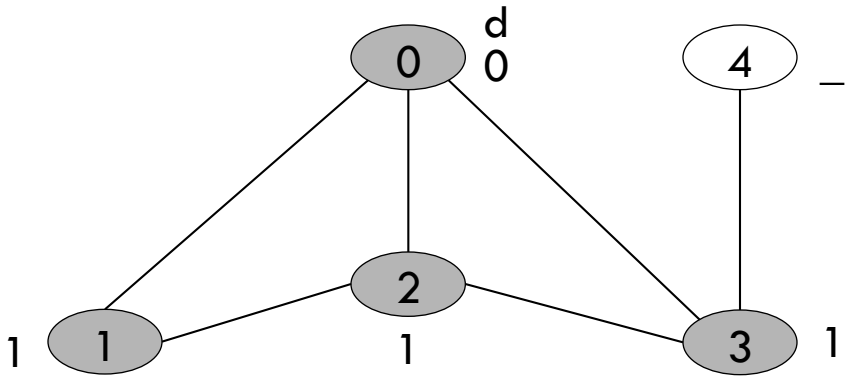
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 - 3

a = (0 - 3)

v = 3



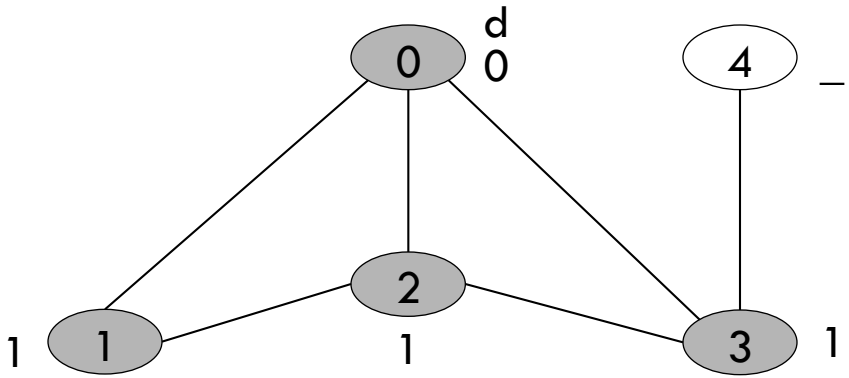
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 - 3

a = NULL

v = 3



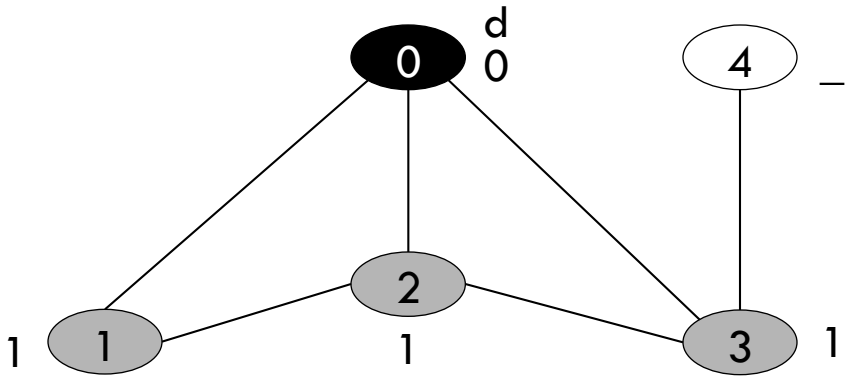
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 - 3

a = NULL

v = 3



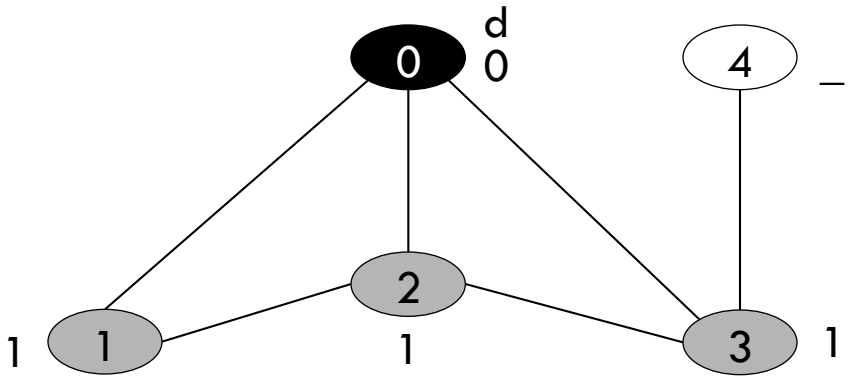
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 0

Fila: 1 - 2 - 3

a = NULL

v = 3



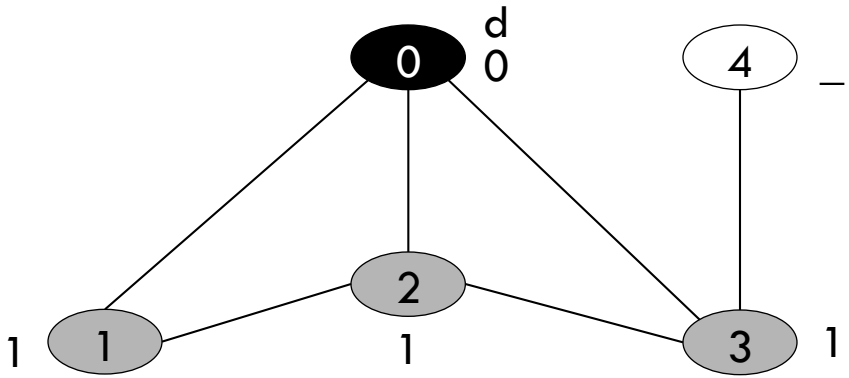
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 1

Fila: 2 - 3

a = NULL

v = 3



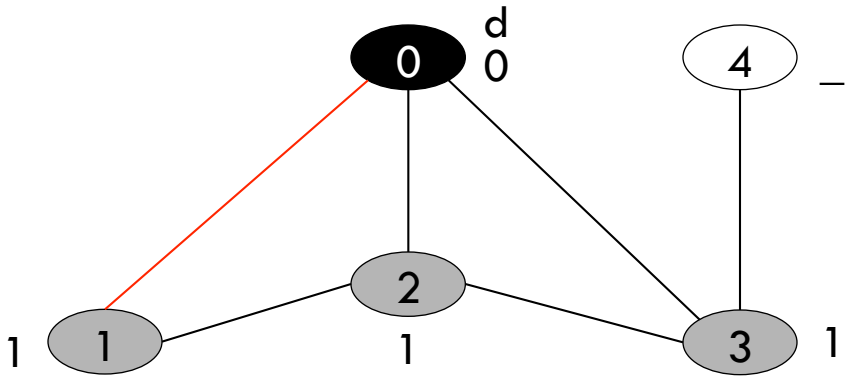
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 1

Fila: 2 - 3

a = 1 - 0

v = 0



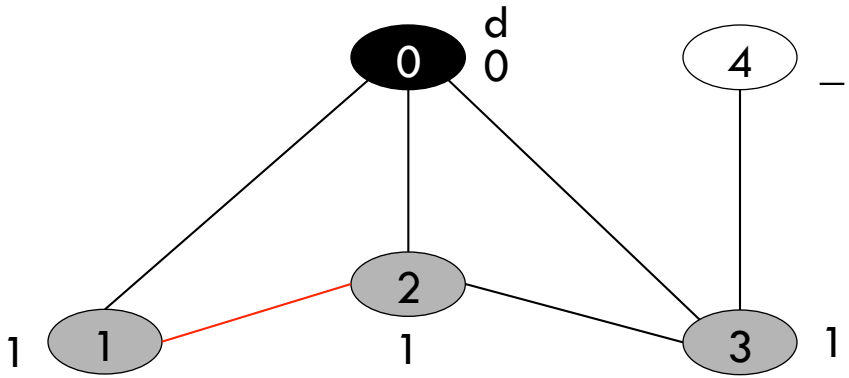
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 1

Fila: 2 - 3

a = 1 - 2

v = 0



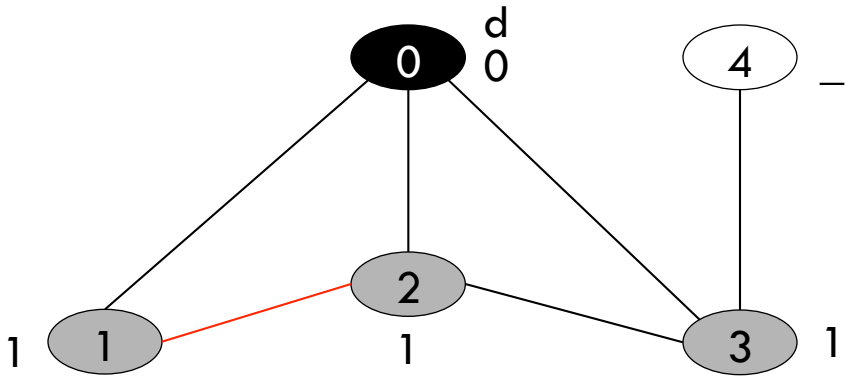
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 1

Fila: 2 - 3

a = 1 - 2

v = 2



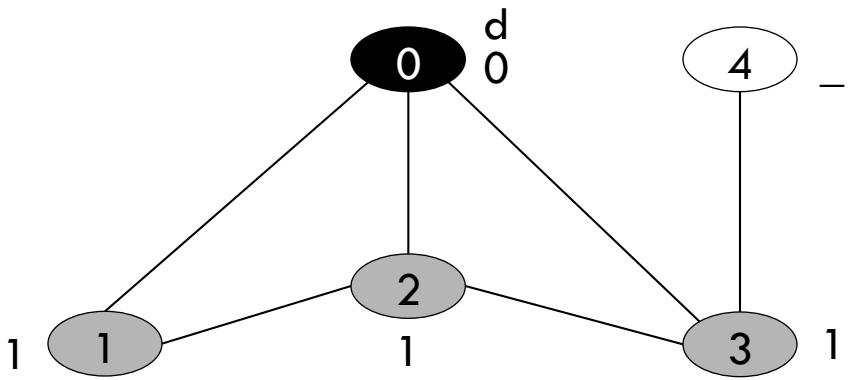
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 1

Fila: 2 - 3

a = NULL

v = 2



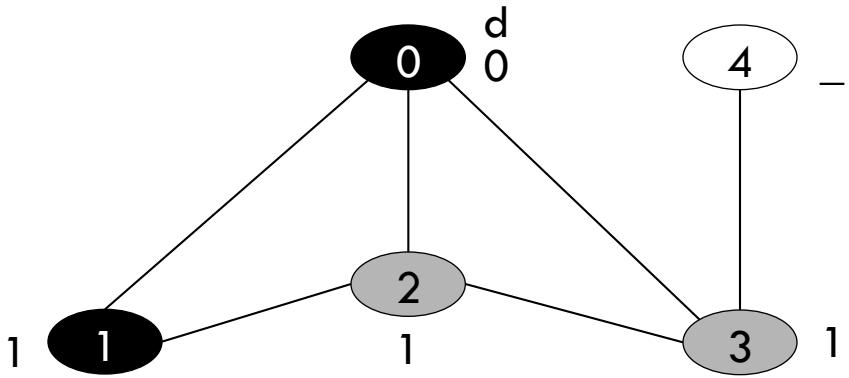
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 1

Fila: 2 - 3

a = NULL

v = 2



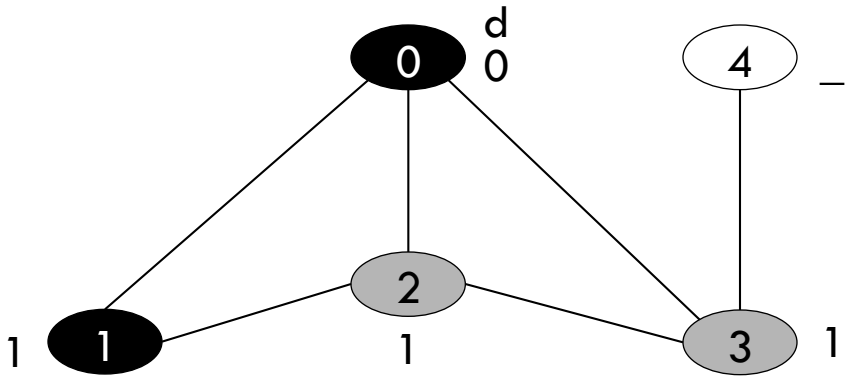
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 2

Fila: 3

a = NULL

v = 2



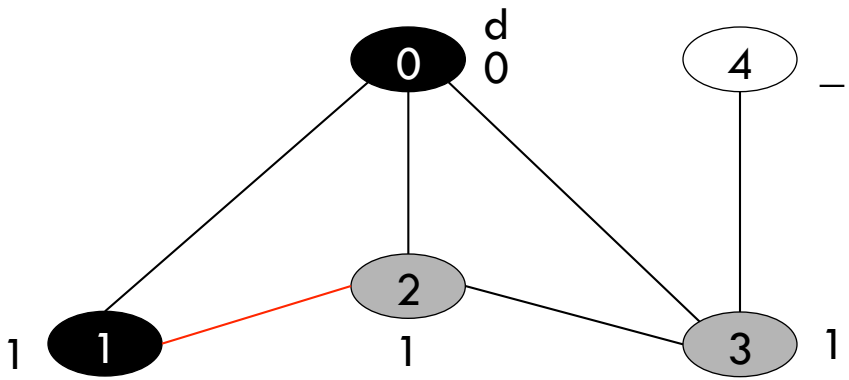
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 2

Fila: 3

a = 2 - 1

v = 1



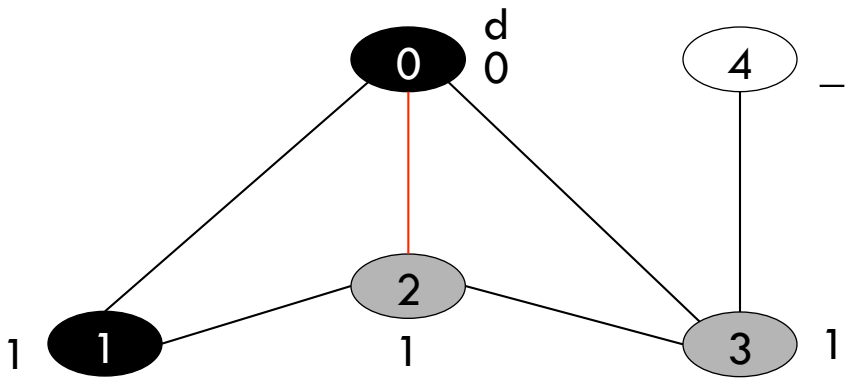
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 2

Fila: 3

a = 2 - 0

v = 0



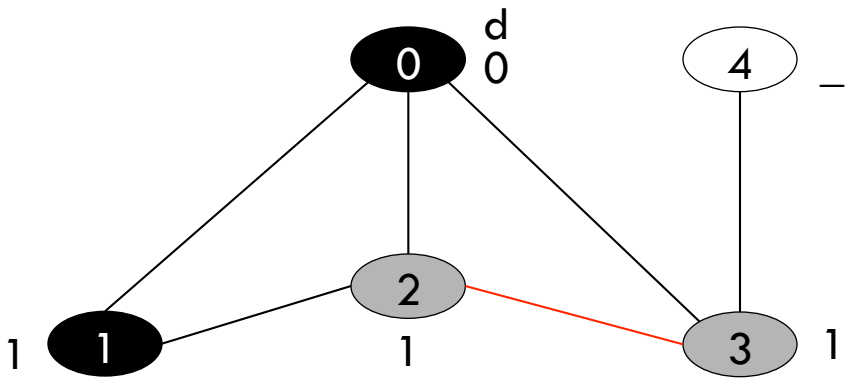
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 2

Fila: 3

a = 2 - 3

v = 3



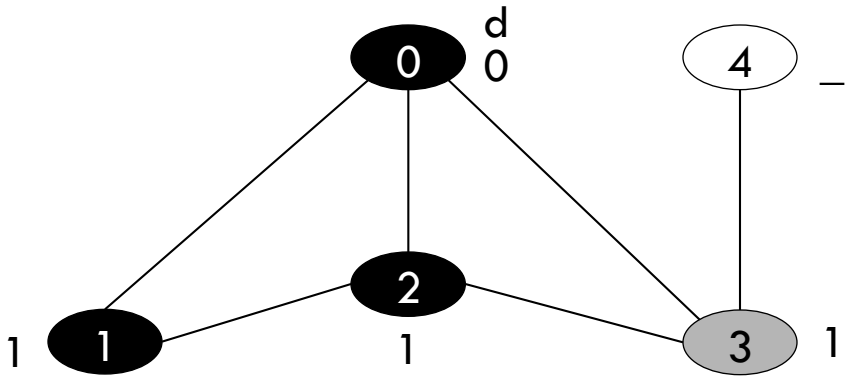
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 2

Fila: 3

a = NULL

v = 3



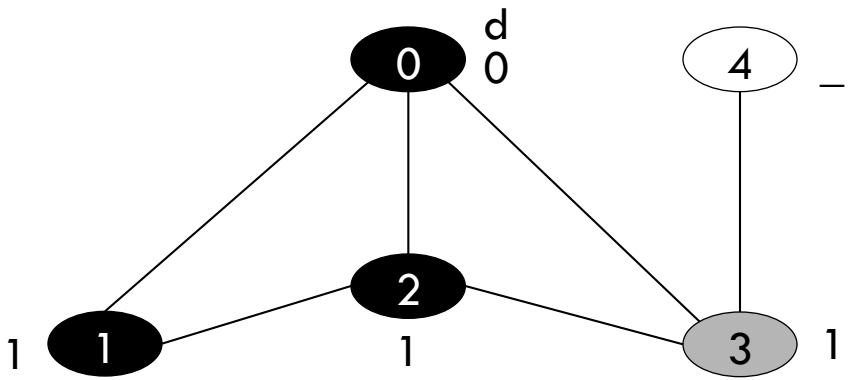
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 3

Fila:

a = NULL

v = 3



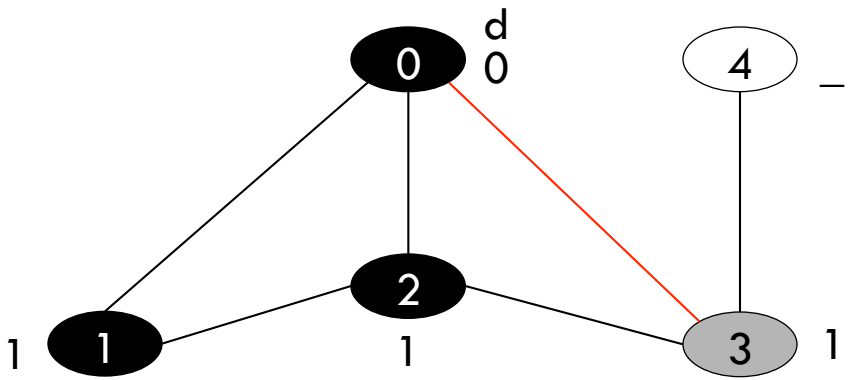
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 3

Fila:

a = 3 - 0

v = 0



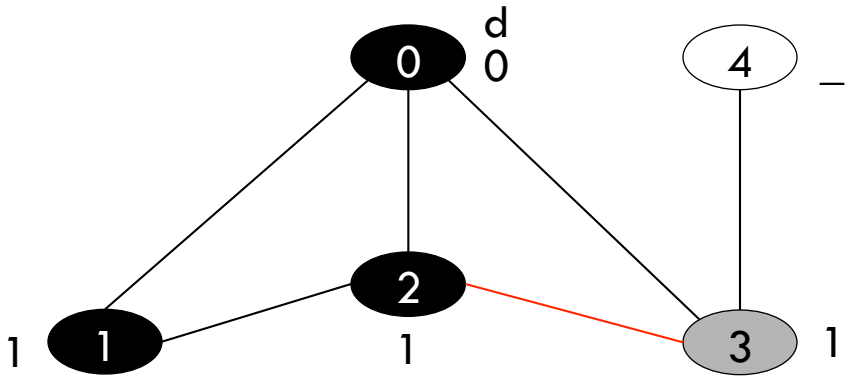
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 3

Fila:

a = 3 - 2

v = 2



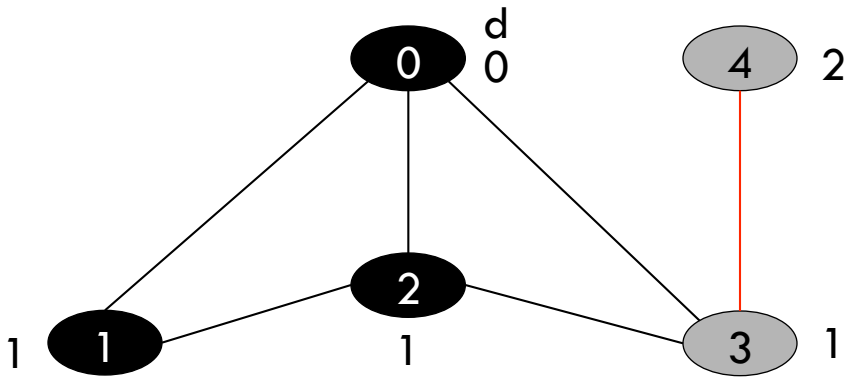
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 3

Fila: 4

a = 3 - 4

v = 4



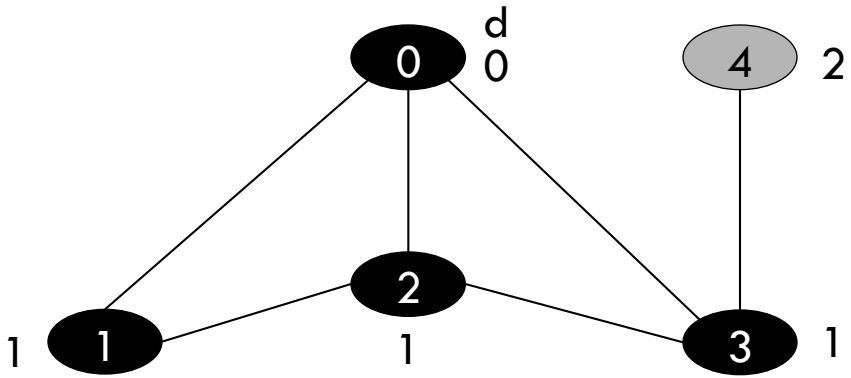
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 3

Fila: 4

a = NULL

v = 4



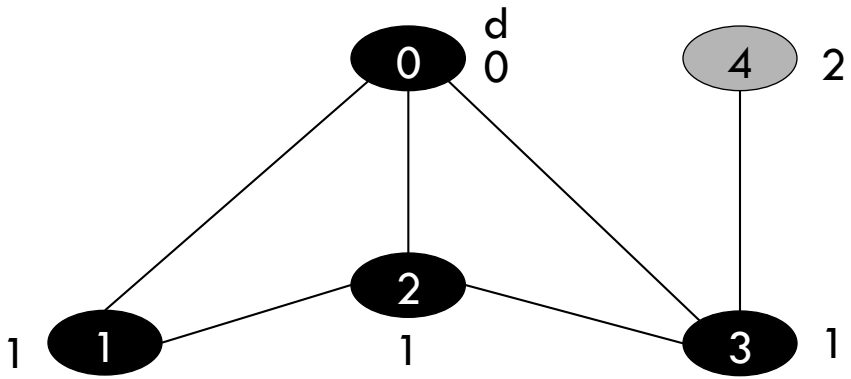
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 4

Fila:

a = NULL

v = 4



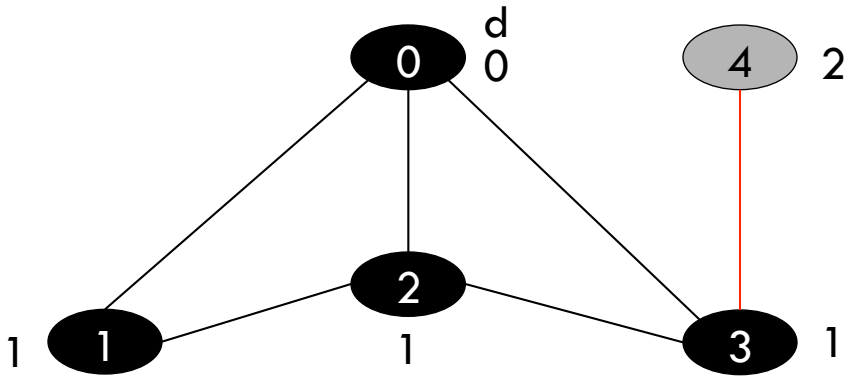
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 4

Fila:

a = 3 - 4

v = 4



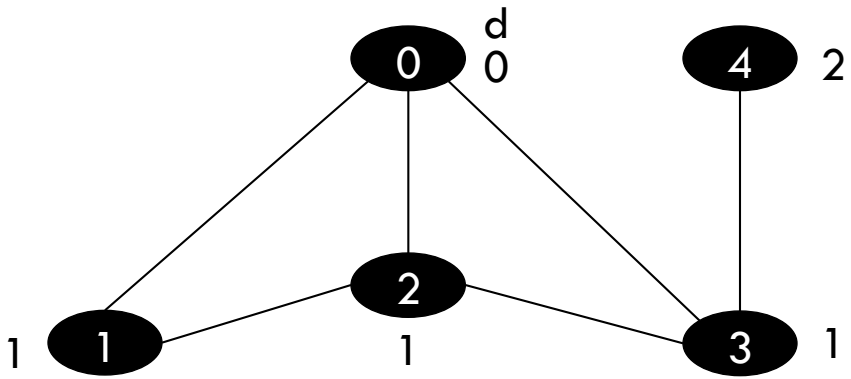
```
private void visitaBfs (int u, int []cor) {
        cor[u] = cinza; this.d[u] = 0;
        FilaRef fila = new FilaRef (); fila.enfileira (u);
        while (!fila.vazia ())
        {
            int aux = (int)fila.desenfileira (); u = aux;
            if (!this.grafo.listaAdjVazia (u))
            {
                Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
                while (a != null)
                {
                    int v = a.get_v2();
                    if (cor[v] == branco) {
                        cor[v] = cinza; this.d[v] = this.d[u] + 1;
                        this.antecessor[v] = u; fila.enfileira (v);
                    }
                    a = this.grafo.proxAdj (u);
                }
            }
            cor[u] = preto;
        }
}
```

vértice atual → u = 4

Fila:

a = NULL

v = 4



Busca em Largura - Análise (para listas de adjacência)

- O custo de inicialização do primeiro anel no método $buscaEmLargura$ é $O(|V|)$.
- O custo do segundo anel é também $O(|V|)$.
- Método $visitaBfs$: enfileirar e $desenfileirar$ têm custo $O(1)$, logo, o custo total com a fila é $O(|V|)$.
- Cada lista de adjacentes é percorrida no máximo uma vez, quando o vértice é $desenfileirado$.
- Desde que a soma de todas as listas de adjacentes é $O(|A|)$, o tempo total gasto com as listas de adjacentes é $O(|A|)$.
- Complexidade total: é $O(|V| + |A|)$.

Caminho mais curto

- A busca em largura encontra o caminho mais curto entre dois vértice u e v.
- O caminho entre dois vértices quaisquer fica armazenado no vetor antecessor.
- O programa abaixo imprime os vértices do caminho mais curto entre o vértice origem (do bfs) e outro vértice qualquer do grafo (a distância aqui não considera peso de arestas).

```
public void imprimeCaminho (int origem, int v) {
        if (origem == v)
            Console.WriteLine (origem);
        else if (this.antecessor[v] == -1)
            Console.WriteLine ("Nao existe caminho de " + origem + " ate " + v);
        else {
            imprimeCaminho (origem, this.antecessor[v]);
            Console.WriteLine (v);
        }
```

Busca em profundidade

- Em inglês, *Depth First Search* (DFS)
- A partir de um vértice de origem, busca recursivamente um vértice adjacente, até que não existam mais vértices a visitar
- Pode gerar várias árvores de profundidade (floresta de busca)

Busca em profundidade

- Mantidas as propriedades de estado
- Nova propriedade: *timestamps* (tempo da busca)
  - *Timestamp* de descoberta
  - *Timestamp* de término

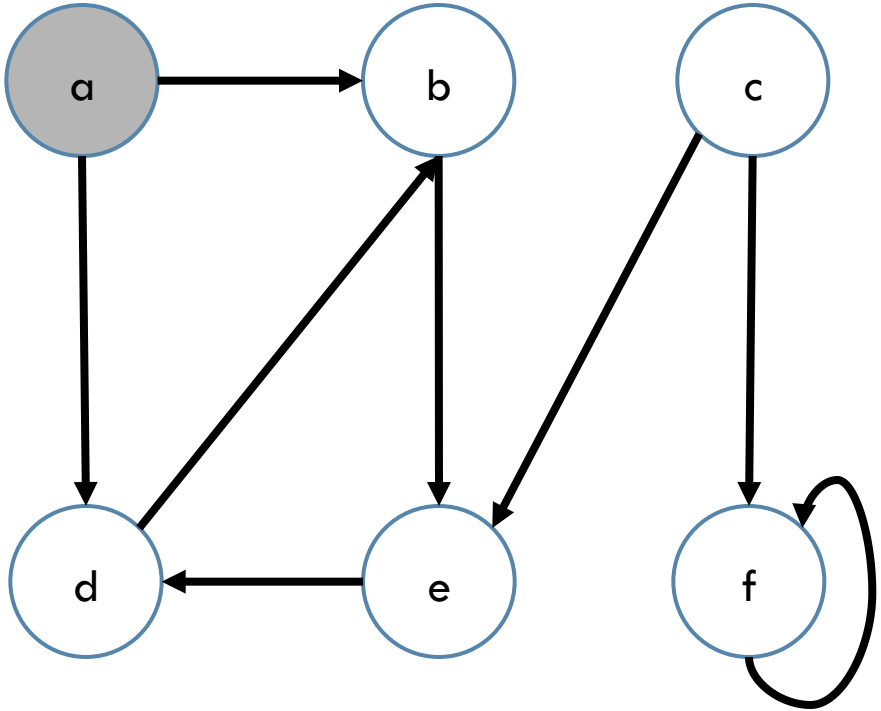
Algoritmo DFS - inicialização

```
Para cada vértice u faça
        u.cor = branco;
        u.pai = null;
Fim para
timestamp = 0
Para cada vértice u faça
        se u.cor == branco
                Visitar(u);
        Fim se
Fim Para
```

Algoritmo DFS – principal (visita)

```
timestamp = timestamp + 1;
u.descoberta = timestamp;
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                v.pai = u;
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
timestamp = timestamp+1;
u.término = timestamp;
```

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	-	-	-	-	-

Finalização

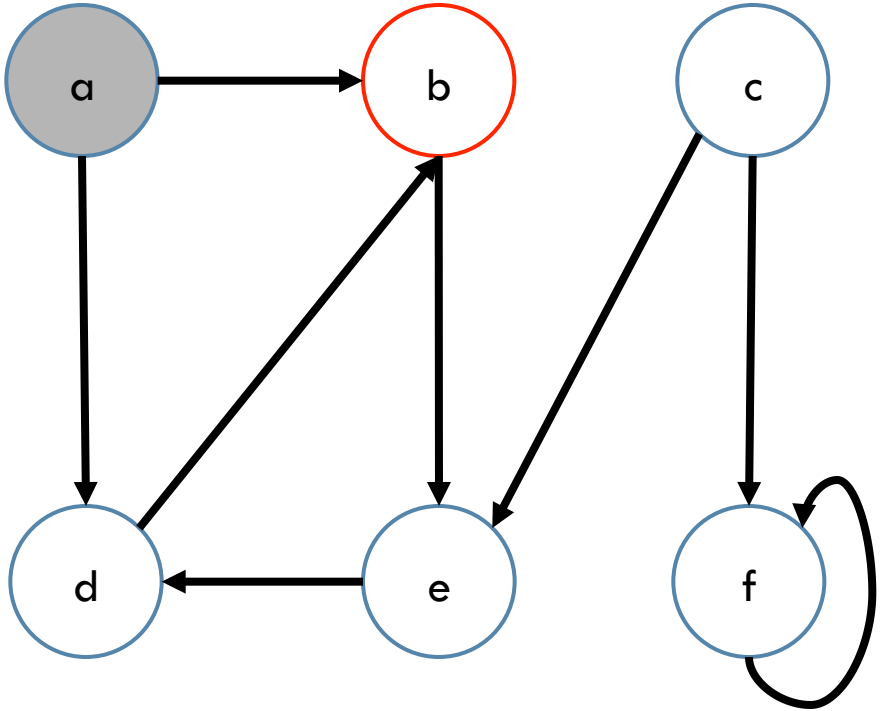
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	-	-	-	-	-

Timestamp: 1

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	-	-	-	-	-

Finalização

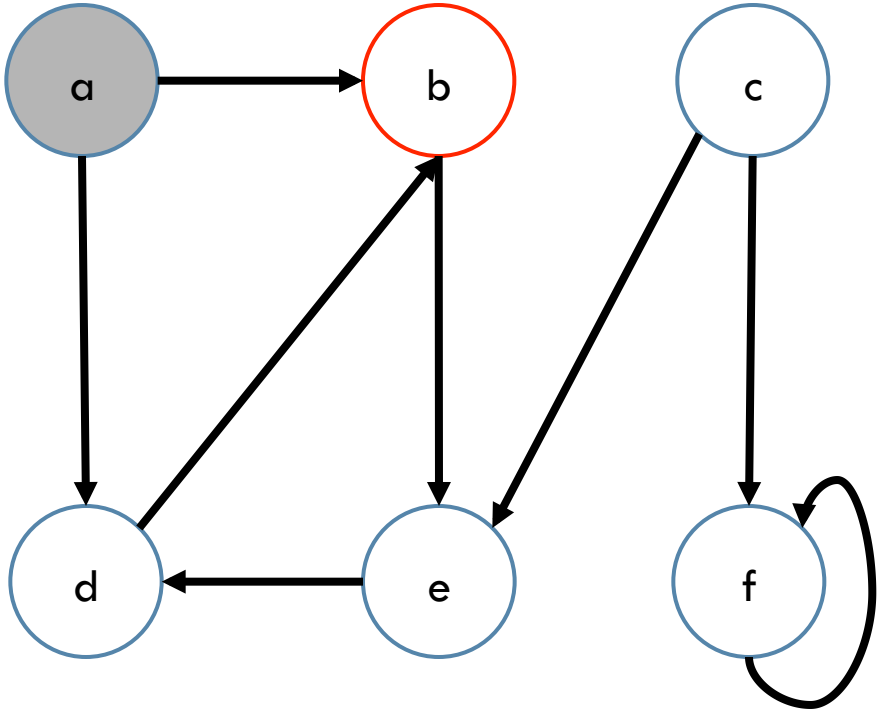
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	-	-	-	-	-

Timestamp: 1

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	-	-	-	-	-

Finalização

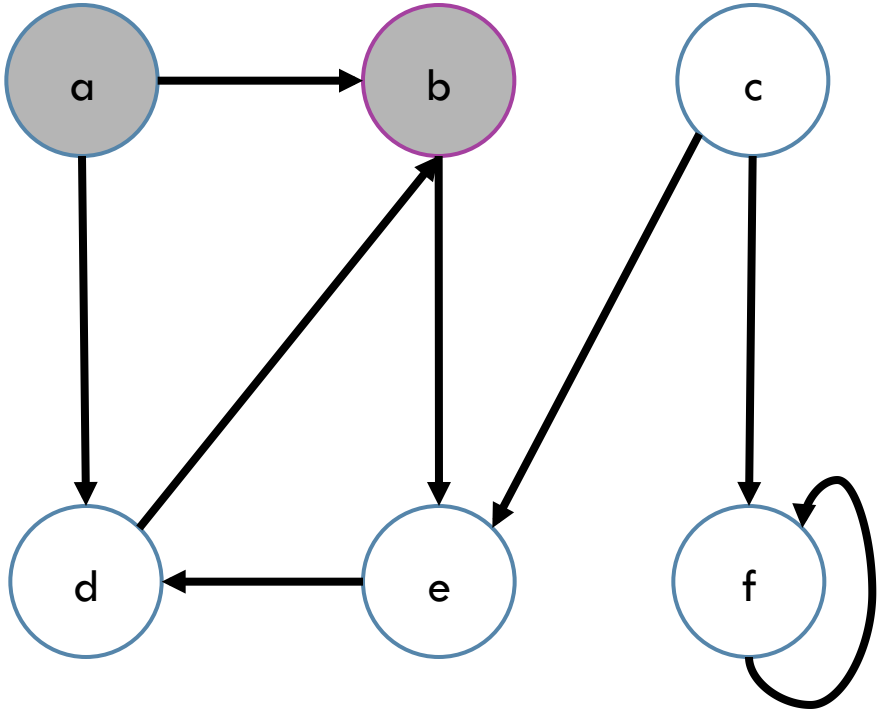
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	-	-	-

Timestamp: 1

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	-	-	-

Finalização

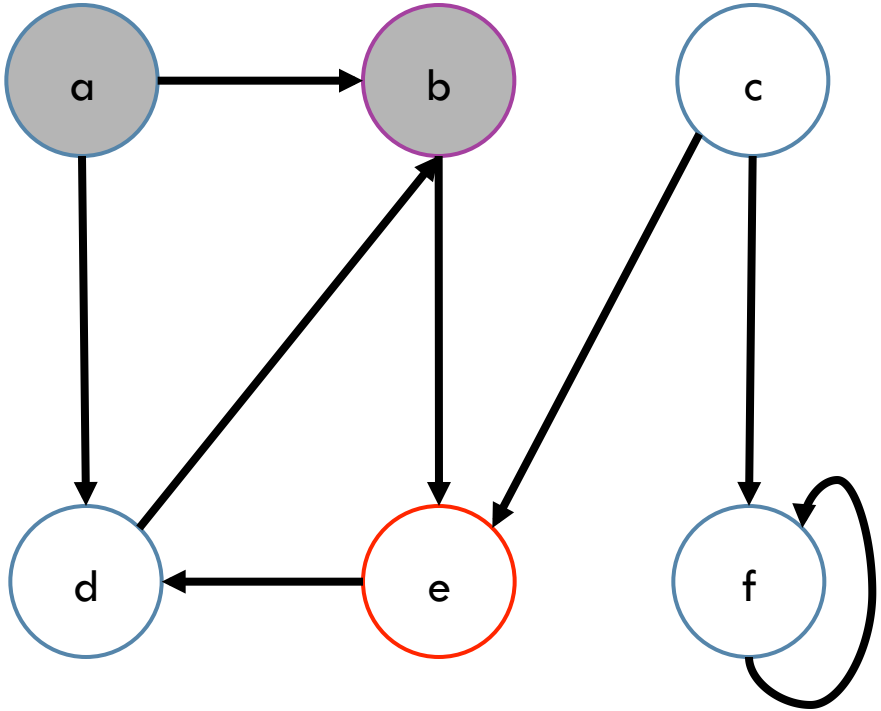
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	-	-	-

Timestamp: 2

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	-	-	-

Finalização

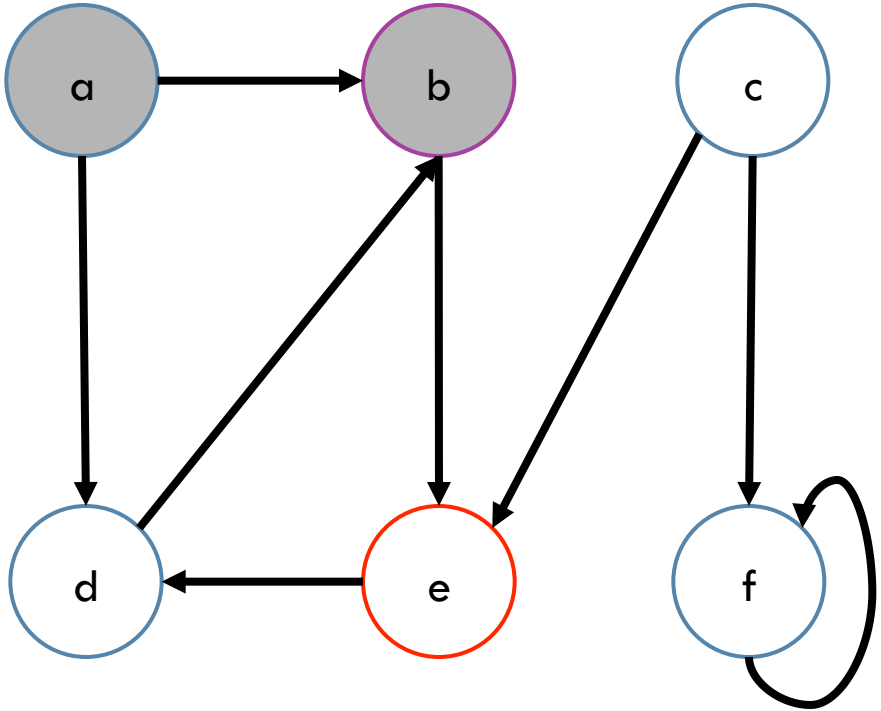
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	-	-	-

Timestamp: 2

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	-	-	-

Finalização

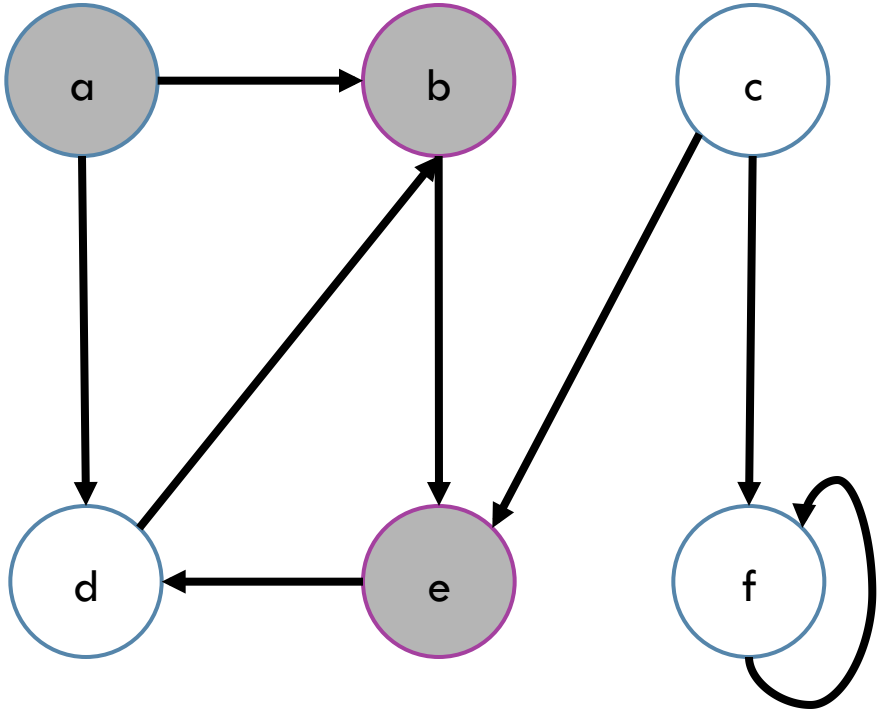
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	-	b	-

Timestamp: 2

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	-	3	-

Finalização

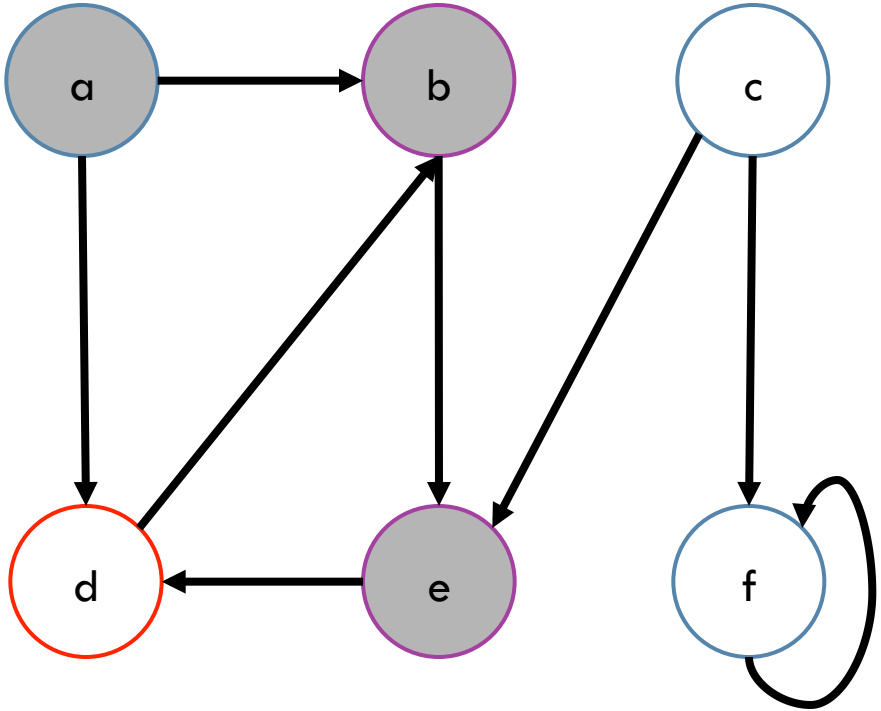
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	-	b	-

Timestamp: 3

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	-	3	-

Finalização

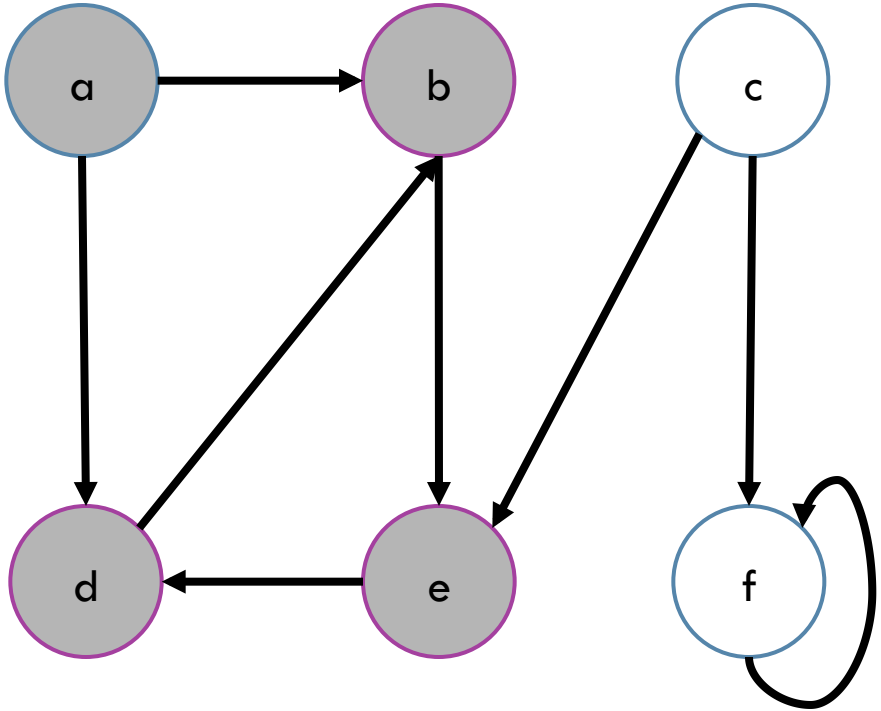
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 3

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

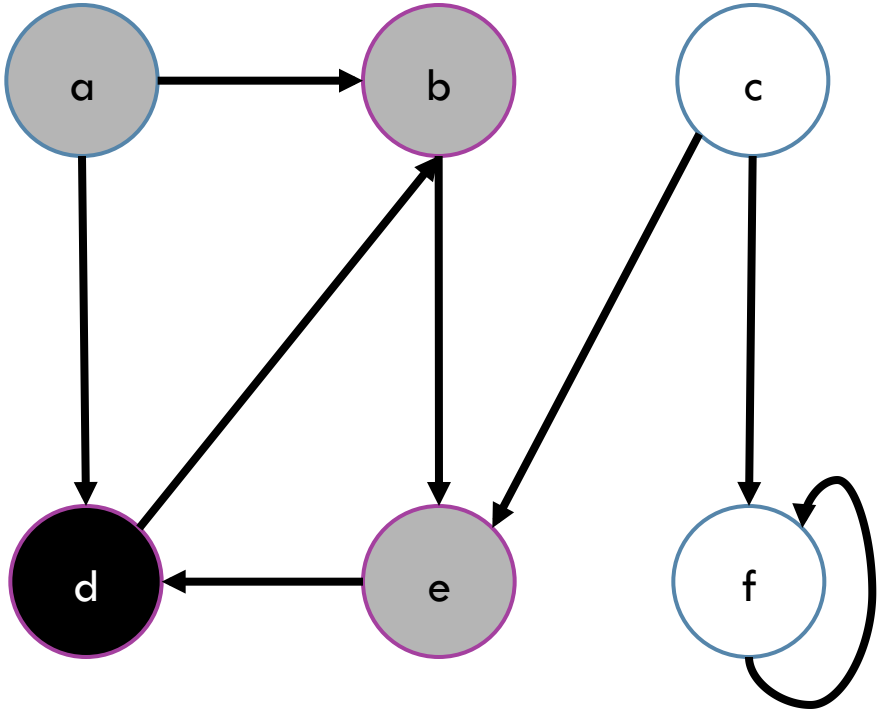
a	b	c	d	e	f
-	-	-	-	-	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 4

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

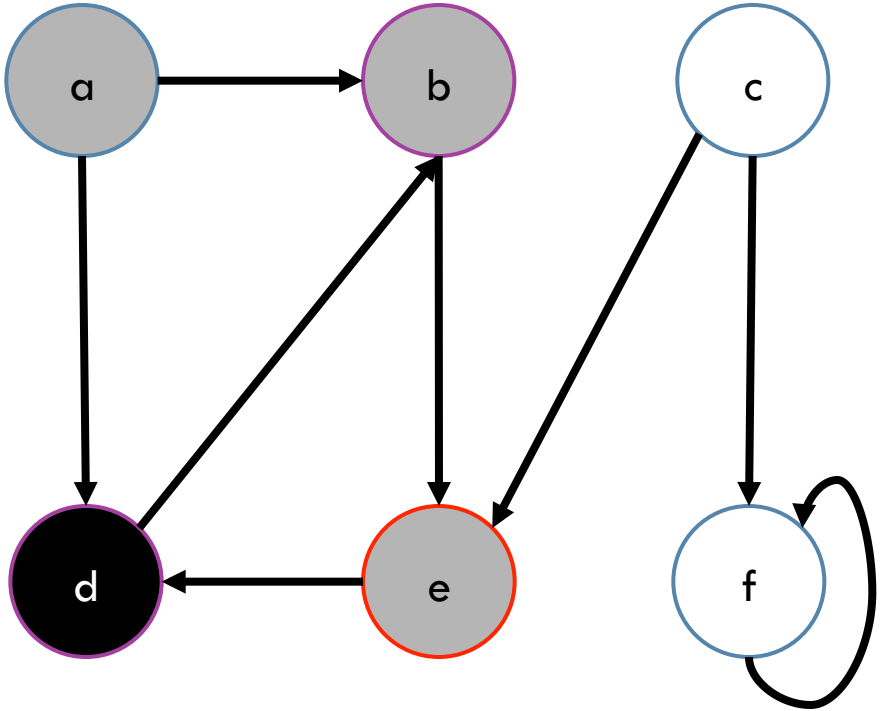
a	b	c	d	e	f
-	-	-	5	-	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 5

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

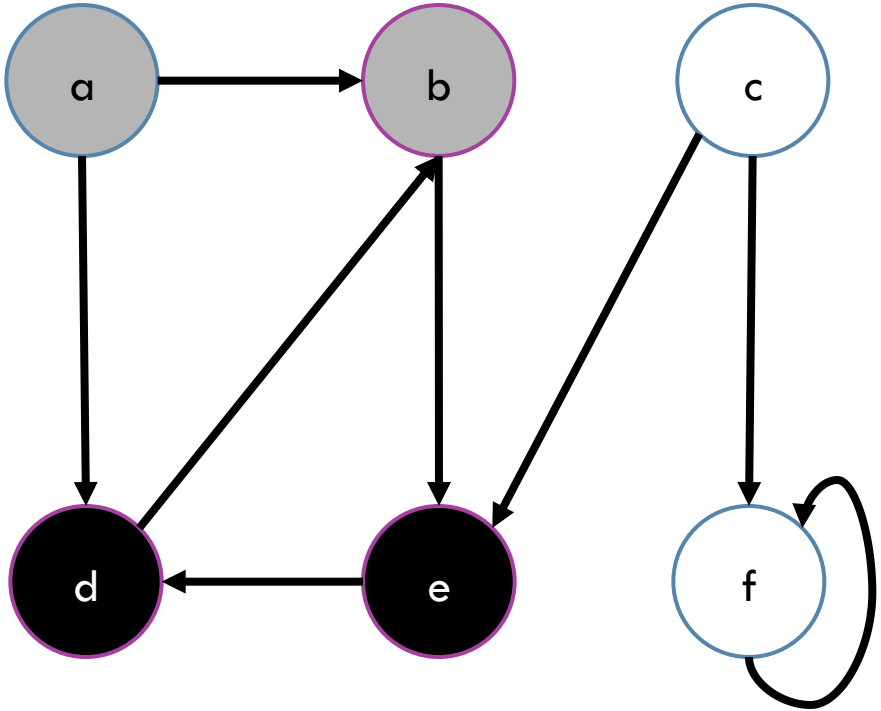
a	b	c	d	e	f
-	-	-	5	-	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 5

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

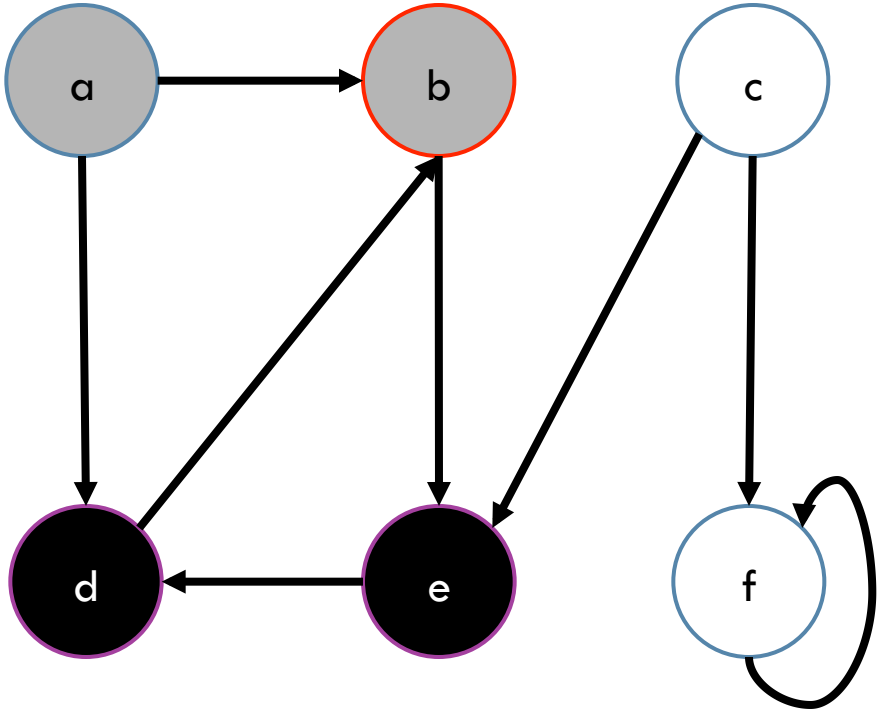
a	b	c	d	e	f
-	-	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 6

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

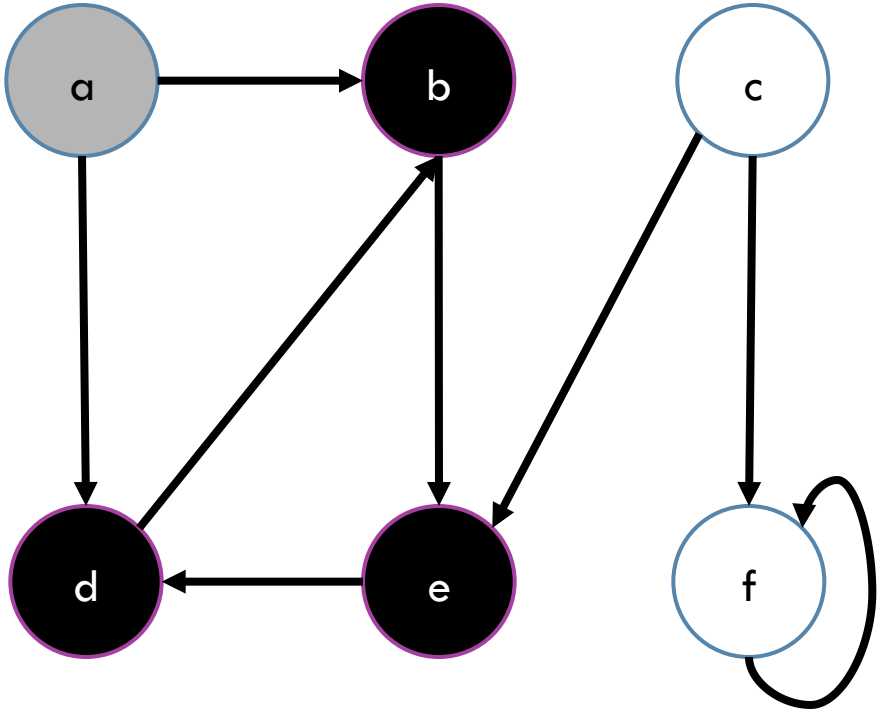
a	b	c	d	e	f
-	-	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 6

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

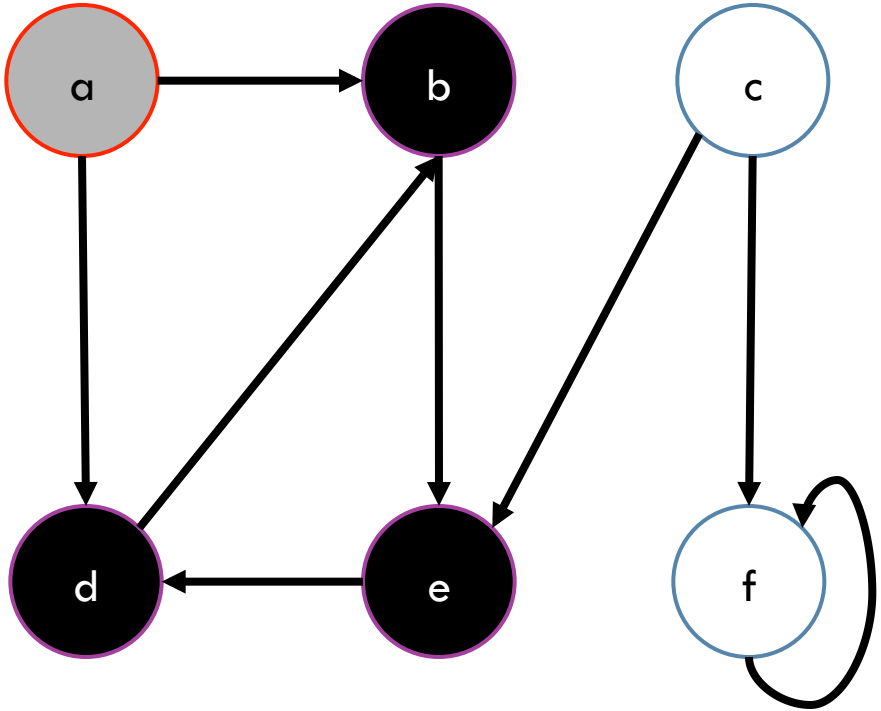
a	b	c	d	e	f
-	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 7

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

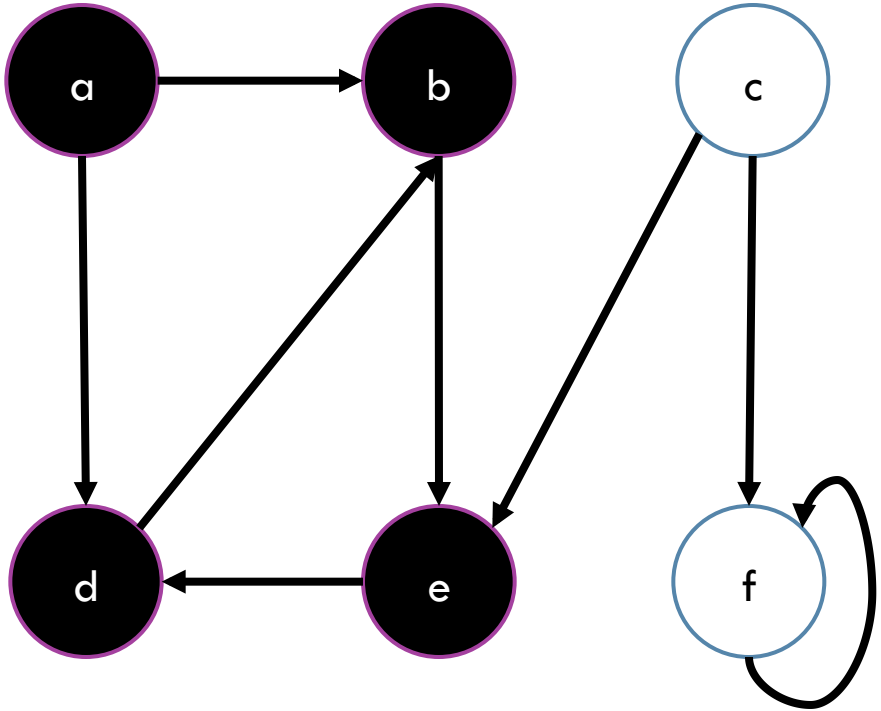
a	b	c	d	e	f
-	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 7

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

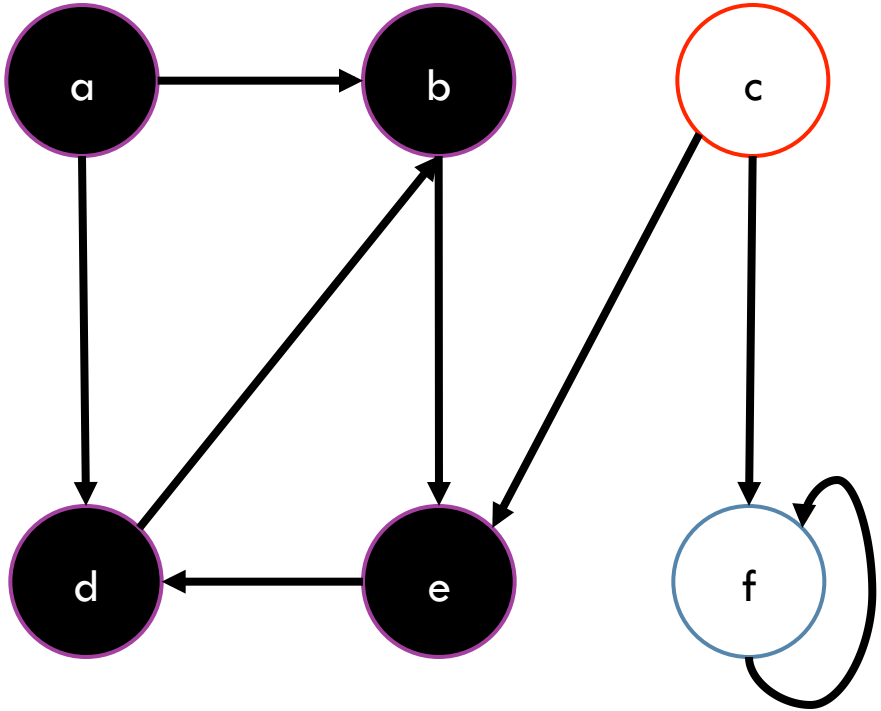
a	b	c	d	e	f
8	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 8

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	-	4	3	-

Finalização

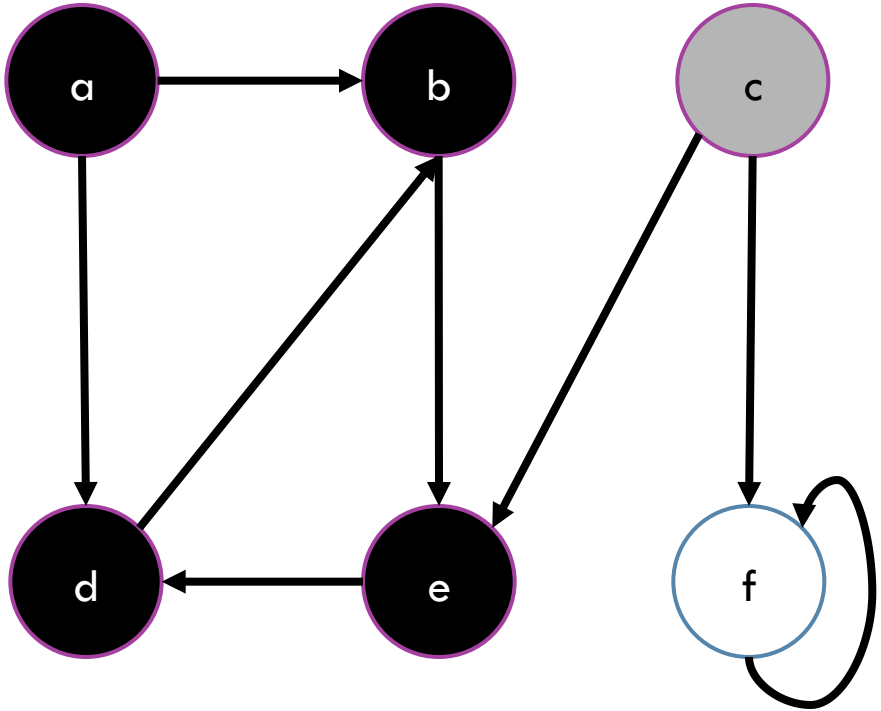
a	b	c	d	e	f
8	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 8

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	9	4	3	-

Finalização

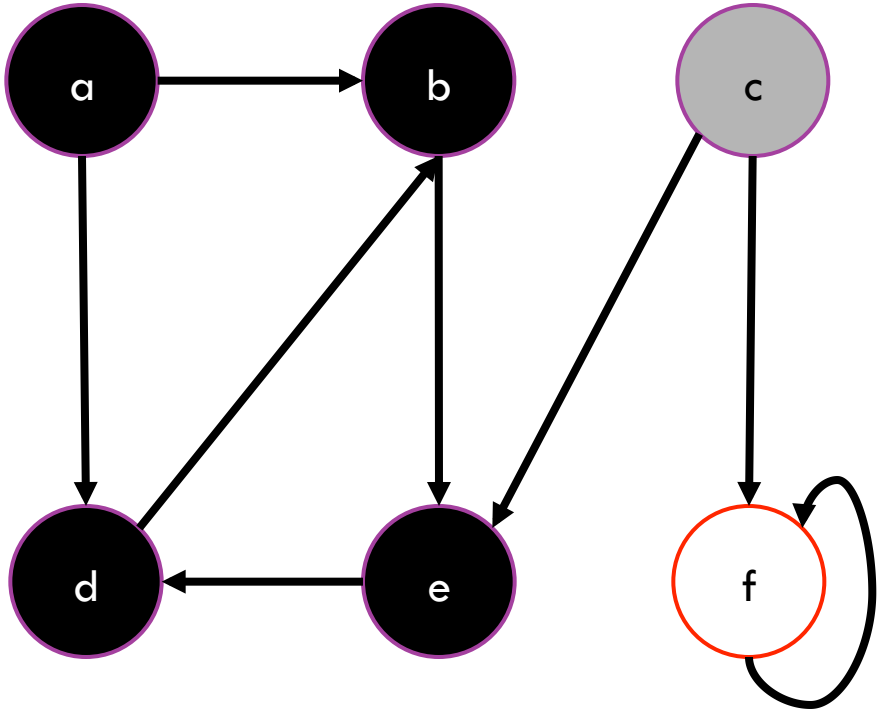
a	b	c	d	e	f
8	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	-

Timestamp: 9

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	9	4	3	-

Finalização

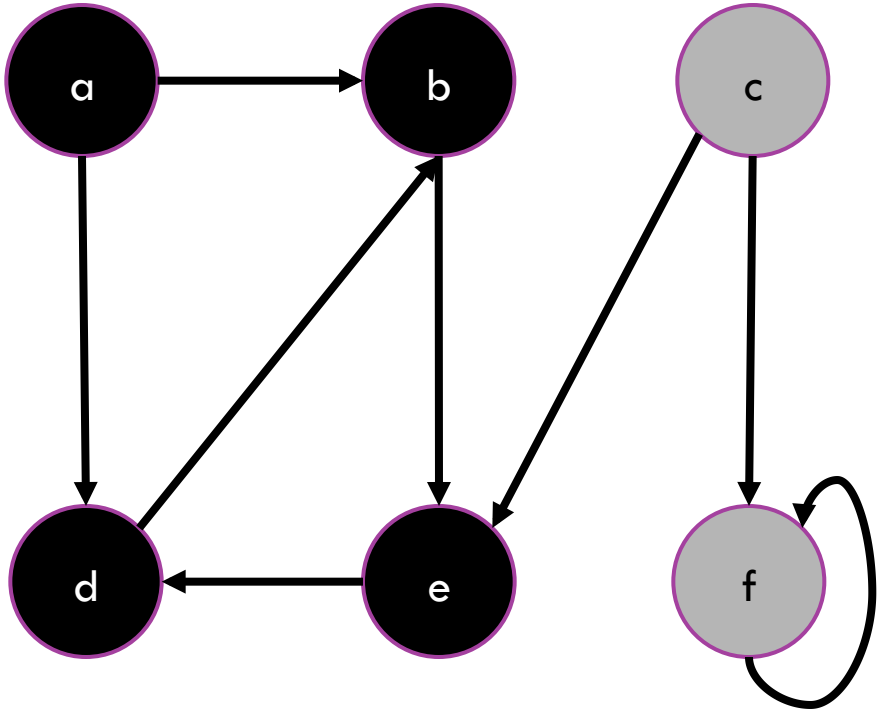
a	b	c	d	e	f
8	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	c

Timestamp: 9

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	9	4	3	10

Finalização

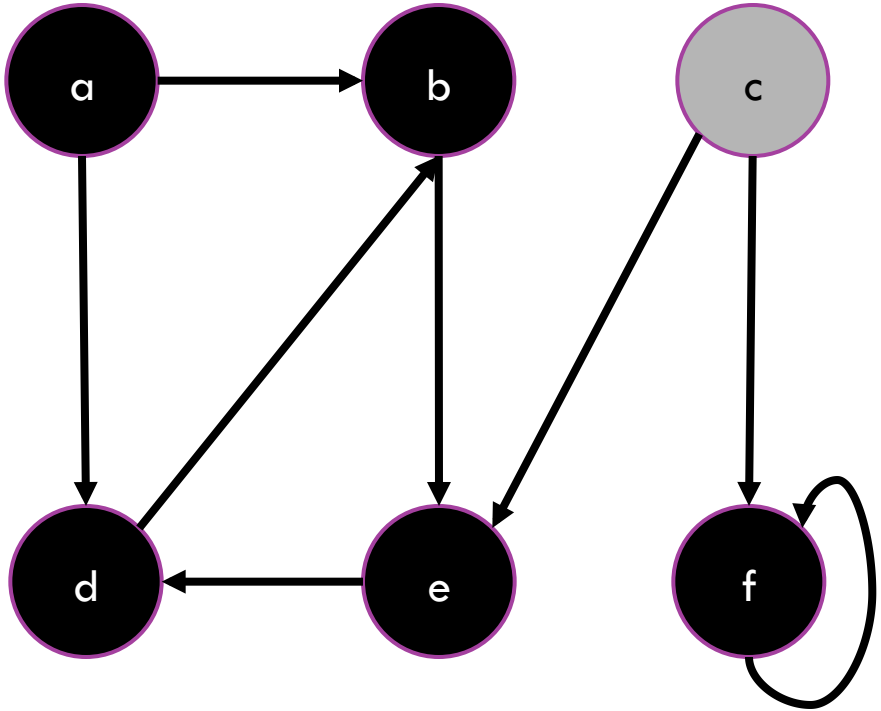
a	b	c	d	e	f
8	7	-	5	6	-

Pais

a	b	c	d	e	f
-	a	-	e	b	c

Timestamp: 10

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	9	4	3	10

Finalização

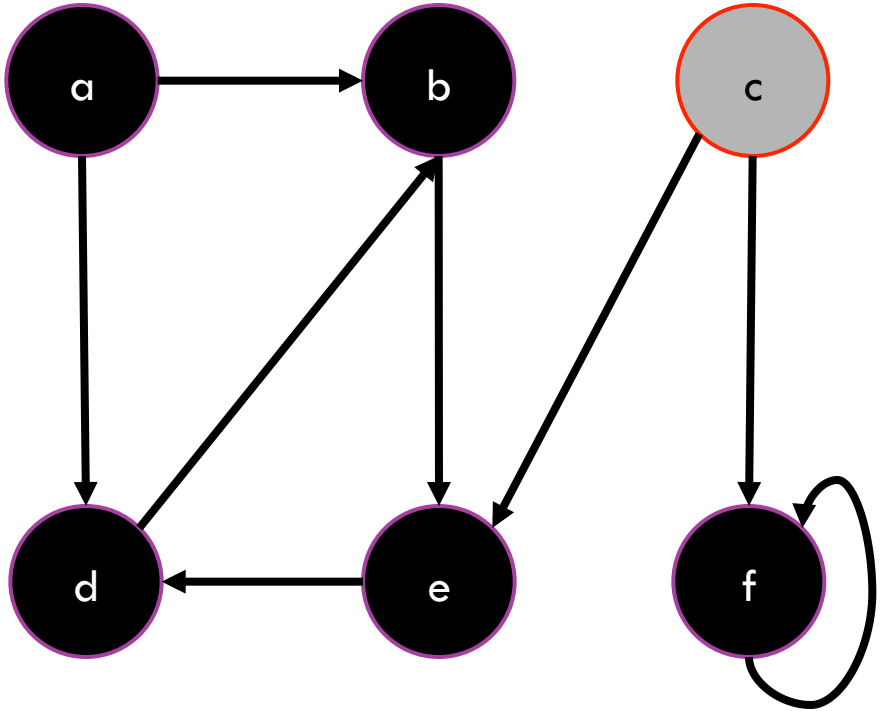
a	b	c	d	e	f
8	7	-	5	6	11

Pais

a	b	c	d	e	f
-	a	-	e	b	c

Timestamp: 11

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	9	4	3	10

Finalização

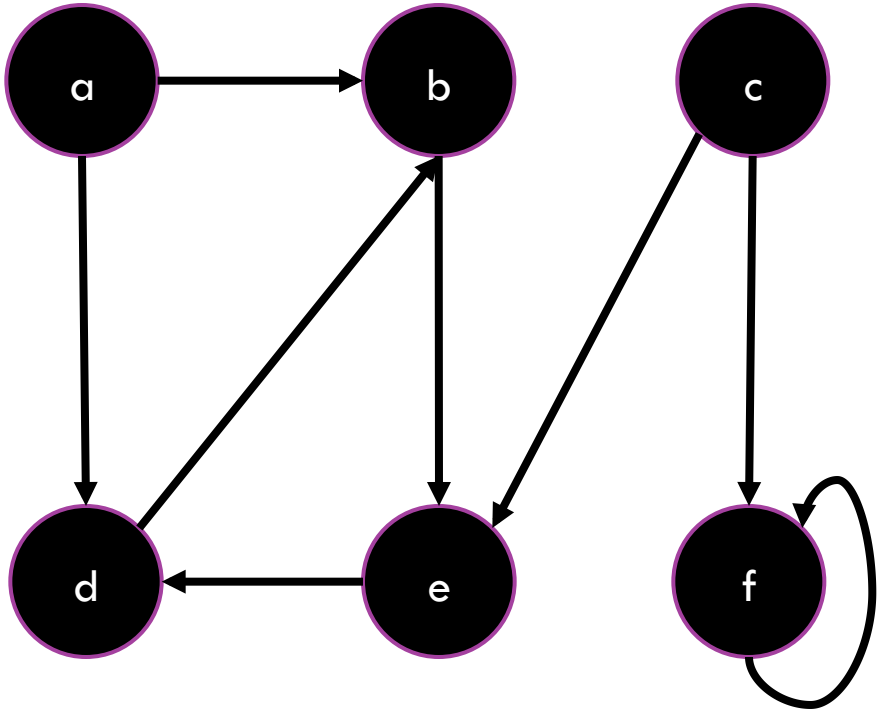
a	b	c	d	e	f
8	7	-	5	6	11

Pais

a	b	c	d	e	f
-	a	-	e	b	c

Timestamp: 11

Algoritmo DFS



Descoberta

a	b	c	d	e	f
1	2	9	4	3	10

Finalização

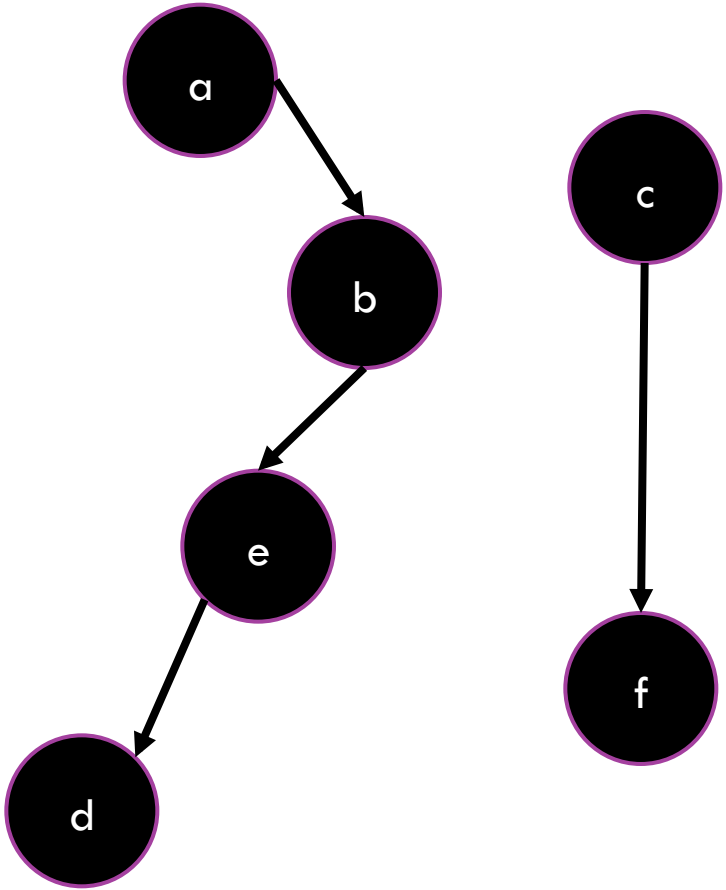
a	b	c	d	e	f
8	7	12	5	6	11

Pais

a	b	c	d	e	f
-	a	-	e	b	c

Timestamp: 12

Floresta DFS

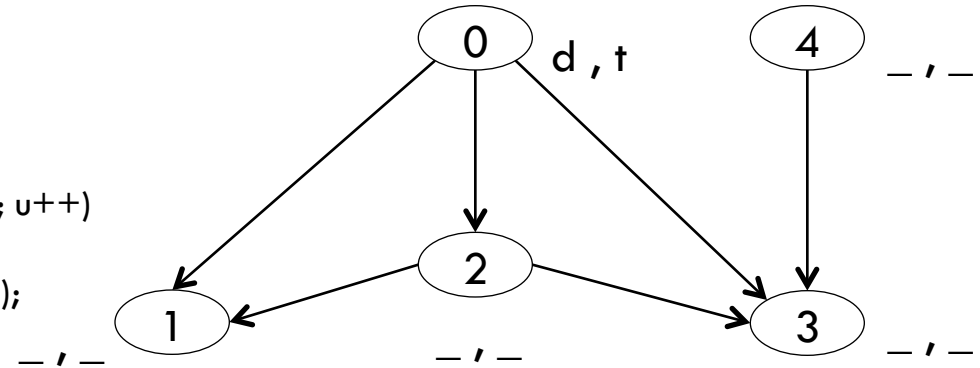


Busca em Profundidade

- O tempo de descoberta $d[v]$ é o momento em que o vértice v foi visitado pela primeira vez
- O tempo de término do exame da lista de adjacentes $t[v]$ é o momento em que a visita a toda lista de vértices adjacentes a v foi concluída.
- Estes registros são inteiros entre 1 e $2|V|$ pois existe um evento de descoberta e um evento de término

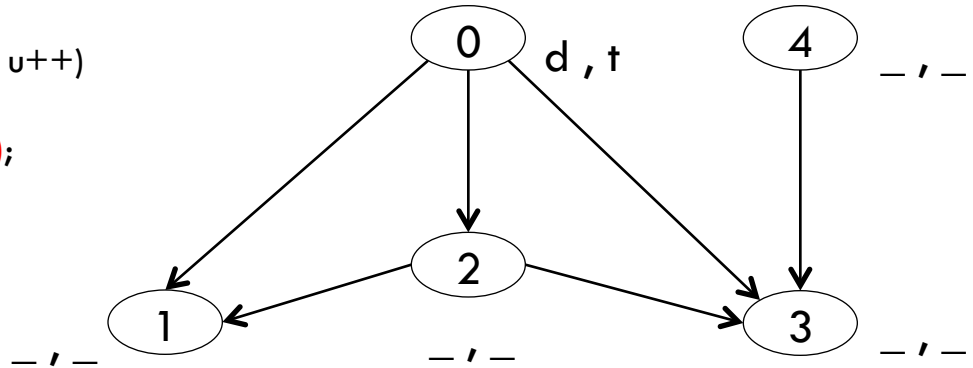
```
public class BuscaEmProfundidade
   {
      public static const byte branco = 0;
      public static const byte cinza  = 1;
      public static const byte preto  = 2;

      private int []d;
      private int []t;
      private int []antecessor;
      private Grafo grafo;

      public BuscaEmProfundidade (Grafo grafo)
      {
         this.grafo = grafo; int n = this.grafo.get_numVertices();
         d = new int[n]; t = new int[n]; antecessor = new int[n];
      }

      public int d (int v) { return this.d[v]; }
      public int t (int v) { return this.t[v]; }
      public int antecessor (int v) { return this.antecessor[v]; }
```

```
public void buscaEmProfundidade ()
    {
        int tempo = 0; int []cor = new int[this.grafo. get_numVertices ()];

        for (int u = 0; u < grafo. get_numVertices (); u++)
        {
            cor[u] = branco; this.antecessor[u] = -1;
        }

        for (int u = 0; u < grafo. get_numVertices (); u++)
            if (cor[u] == branco)
                tempo = this.visitaDfs (u, tempo, cor);
    }
```



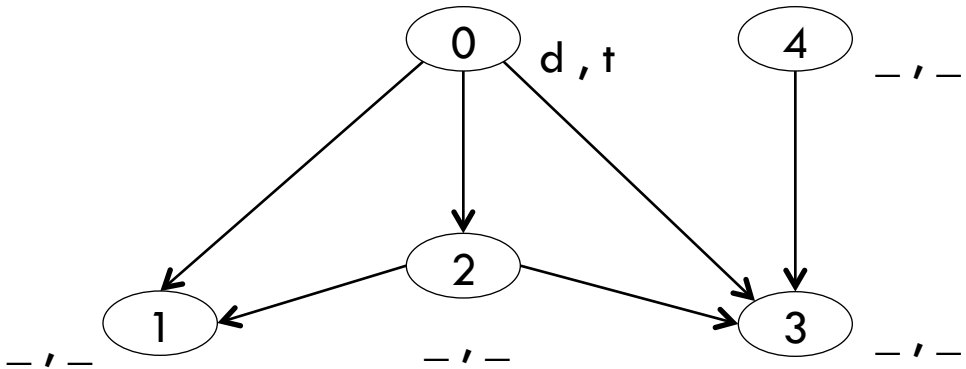
```
public void buscaEmProfundidade ()
    {
        int tempo = 0; int []cor = new int[this.grafo. get_numVertices ()];

        for (int u = 0; u < grafo. get_numVertices (); u++)
        {
            cor[u] = branco; this.antecessor[u] = -1;
        }

        for (int u = 0; u < grafo. get_numVertices (); u++)
            if (cor[u] == branco)
                    tempo = this.visitaDfs (u, tempo, cor);
    }
```

Início da chamada da função visitaDfs



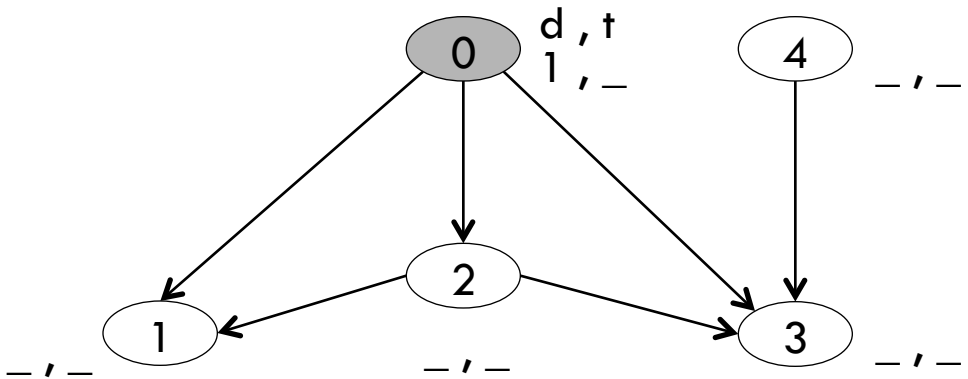
```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

vértice atual → u = 0
tempo = 0
cor [u] = branco



```
private int visitaDfs (int u, int tempo, int []cor)  {
     cor[u] = cinza; this.d[u] = ++tempo;

     if (!this.grafo.listaAdjVazia (u))
     {
          Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
          while (a != null)
          {
               int v = a. get_v2 ();
               if (cor[v] == branco)
               {
                   this.antecessor[v] = u;
                   tempo = this.visitaDfs (v, tempo, cor);
               }
               a = this.grafo.proxAdj (u);
          }
     }
     cor[u] = preto; this.t[u] = ++tempo;
     return tempo;
}
```

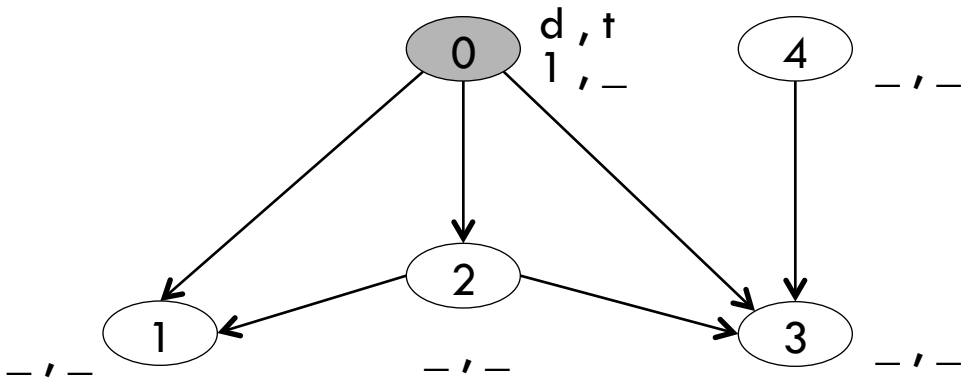
u = 0
tempo = 1
cor [u] = cinza



```
private int visitaDfs (int u, int tempo, int []cor)  {
     cor[u] = cinza; this.d[u] = ++tempo;

     if (!this.grafo.listaAdjVazia (u))
     {
          Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
           while (a != null)
           {
                int v = a. get_v2 ();
                if (cor[v] == branco)
                {
                    this.antecessor[v] = u;
                    tempo = this.visitaDfs (v, tempo, cor);
                }
                a = this.grafo.proxAdj (u);
           }
      }
      cor[u] = preto; this.t[u] = ++tempo;
      return tempo;
}
```

Verifica se lista de adj do vértice u está vazia

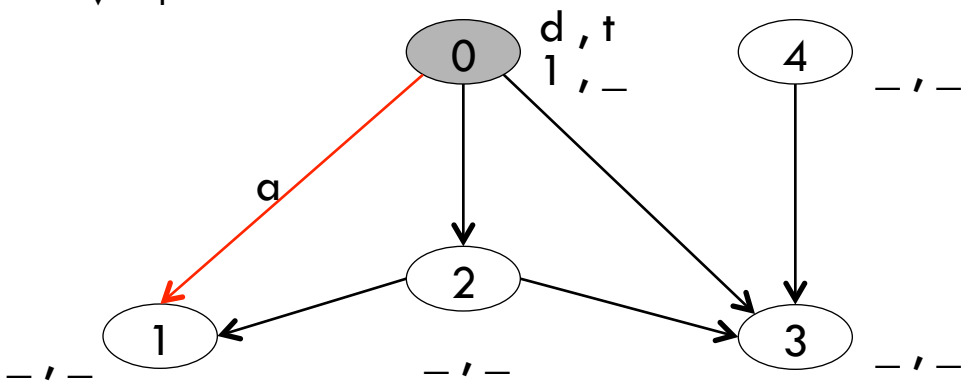
u = 0
tempo = 1
cor [u] = cinza



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
            {
                  int v = a. get_v2 ();
                  if (cor[v] == branco)
                  {
                      this.antecessor[v] = u;
                      tempo = this.visitaDfs (v, tempo, cor);
                  }
                  a = this.grafo.proxAdj (u);
            }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 1
cor [u] = cinza

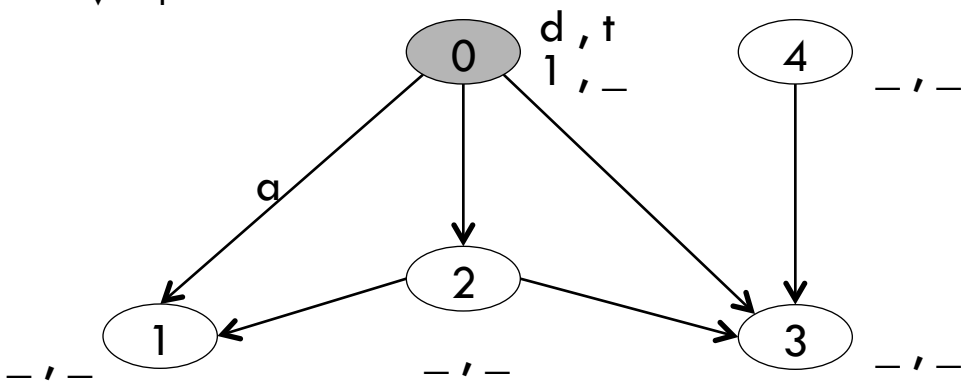
a = aresta (0 -> 1)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 1
cor [u] = cinza

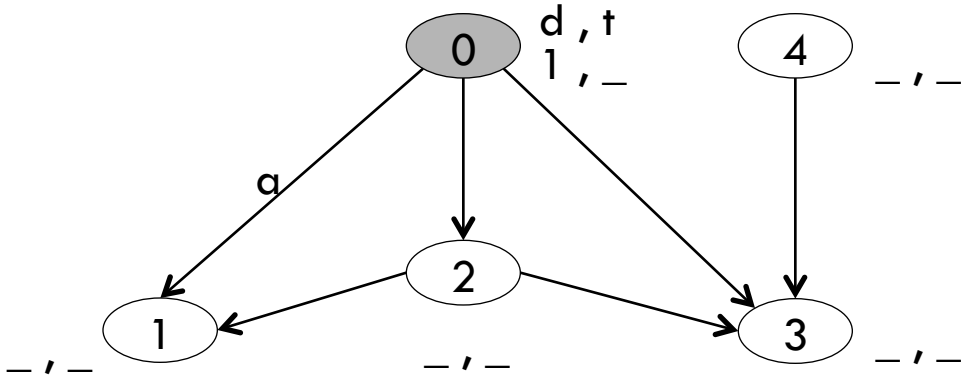
a = aresta (0 -> 1)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 1
cor [u] = cinza

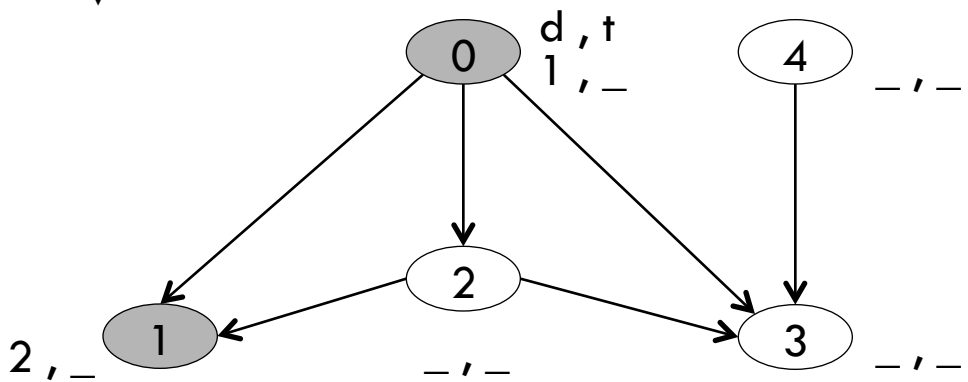
a = aresta (0 -> 1)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 1
tempo = 2
cor [u] = cinza

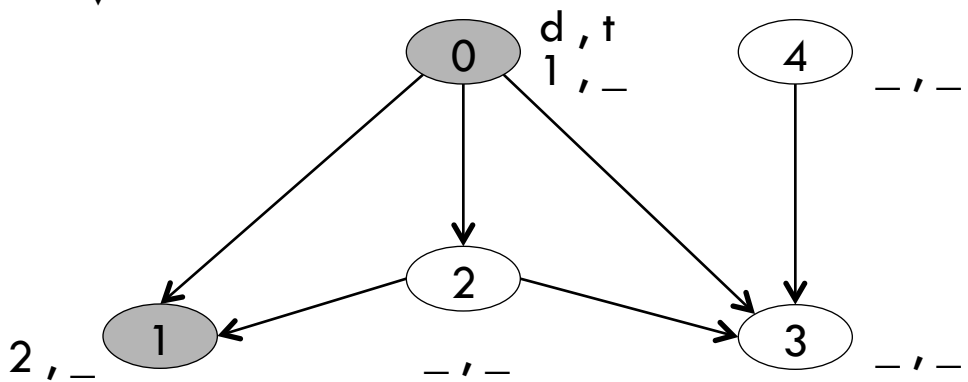
a =
v =



```
private int visitaDfs (int u, int tempo, int []cor)  {
     cor[u] = cinza; this.d[u] = ++tempo;

     if (!this.grafo.listaAdjVazia (u))
     {
          Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
           while (a != null)
           {
                int v = a. get_v2 ();
                if (cor[v] == branco)
                {
                    this.antecessor[v] = u;
                    tempo = this.visitaDfs (v, tempo, cor);
                }
                a = this.grafo.proxAdj (u);
           }
      }
      cor[u] = preto; this.t[u] = ++tempo;
      return tempo;
}
```

u = 1
tempo = 2
cor [u] = cinza

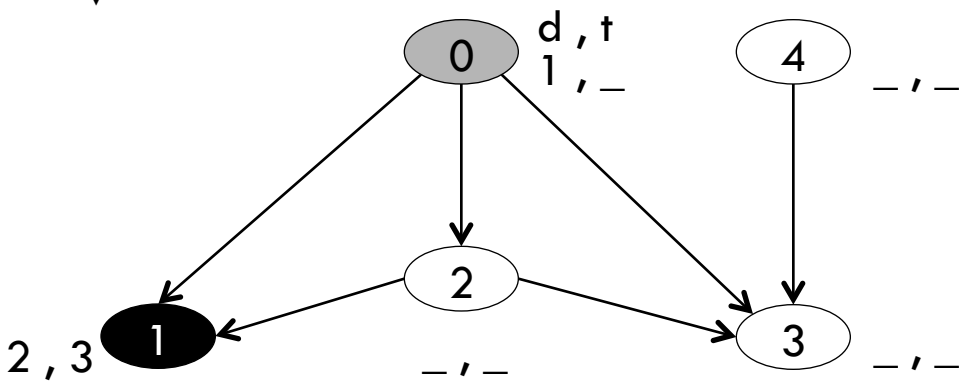
a =
v =



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 1
tempo = 3
cor [u] = preto

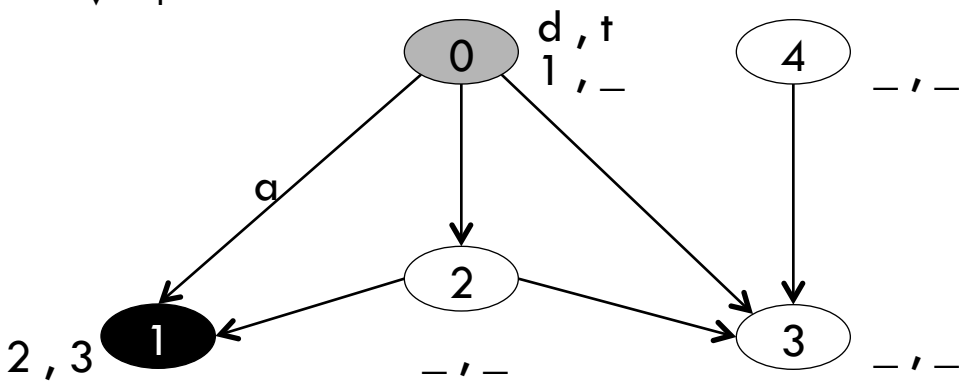
a =
v =



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 3
cor [u] = cinza

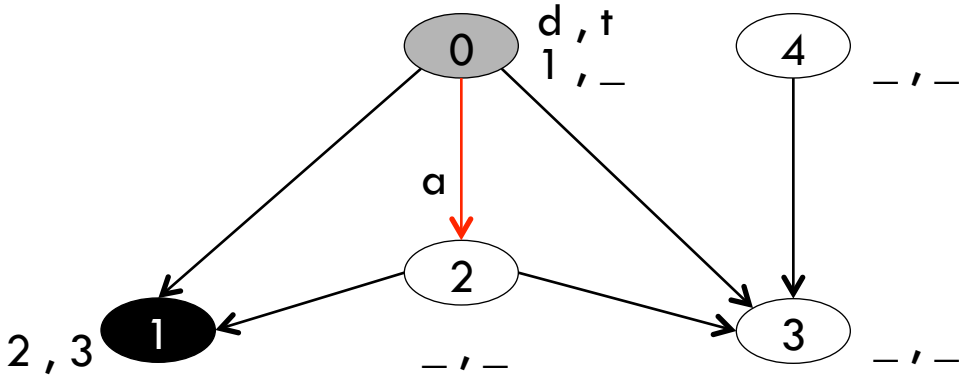
a = aresta (0 -> 1)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 3
cor [u] = cinza

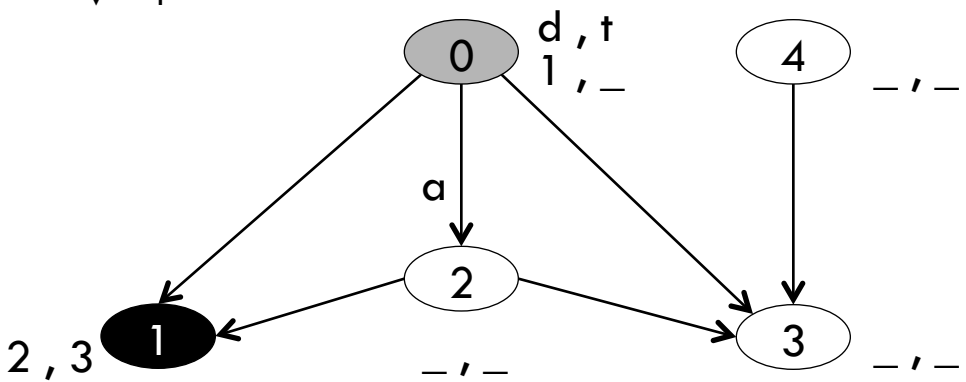
a = aresta (0 -> 2)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                      this.antecessor[v] = u;
                      tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
            }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 3
cor [u] = cinza

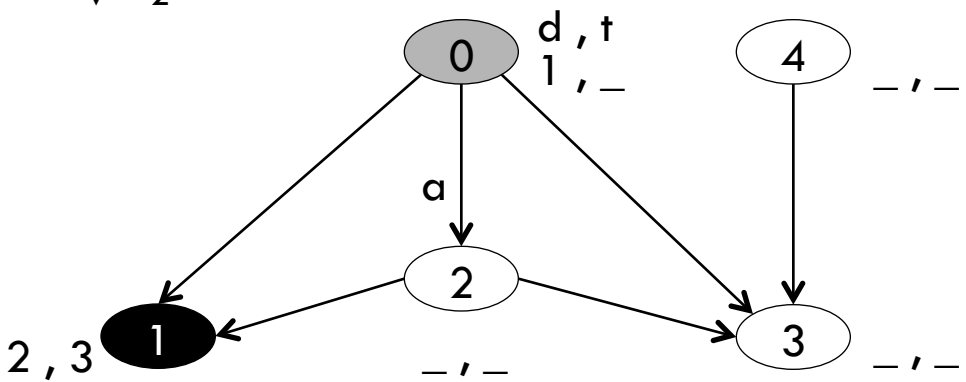
a = aresta (0 -> 2)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 3
cor [u] = cinza

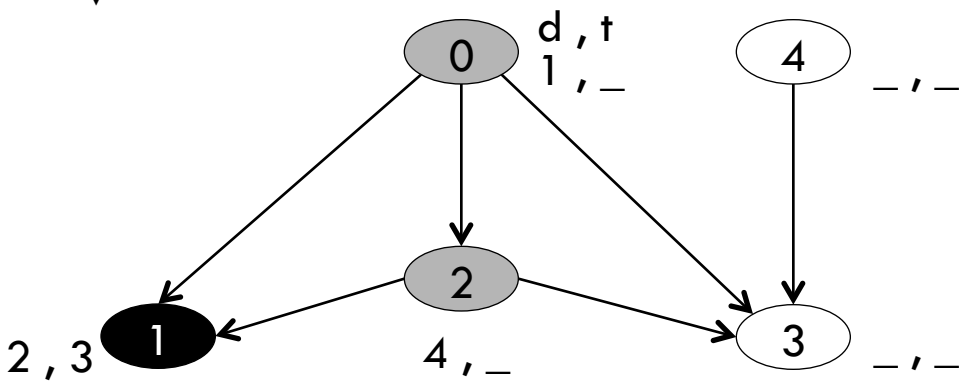
a = aresta (0 -> 2)
v = 2



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 4
cor [u] = cinza

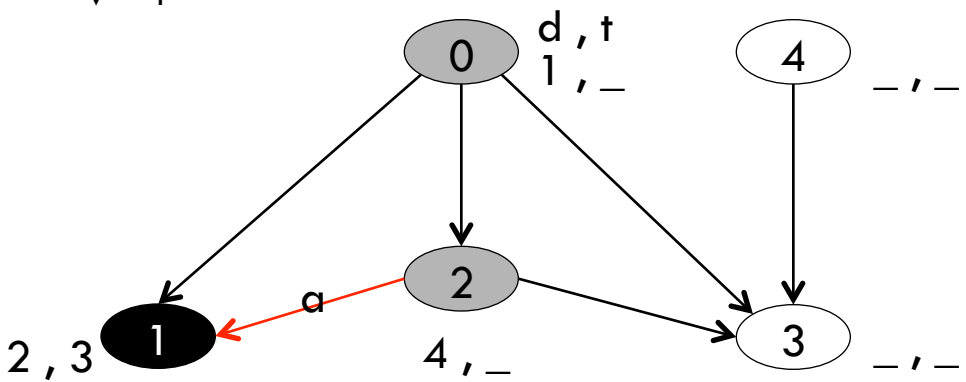
a =
v =



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 4
cor [u] = cinza

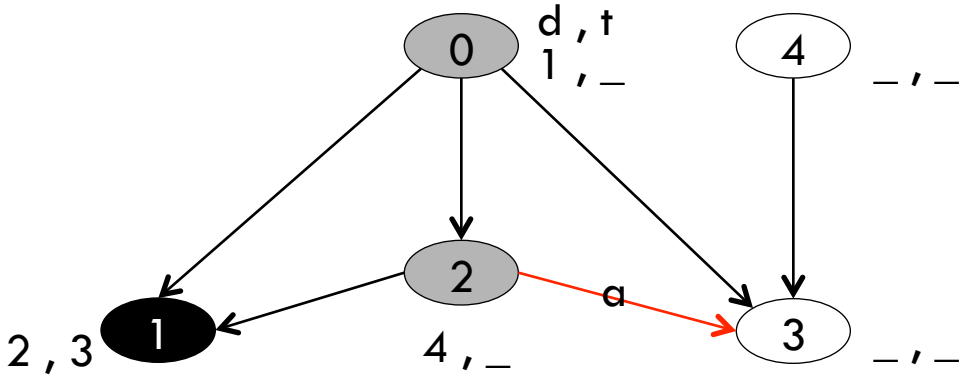
a = aresta (2 -> 1)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 4
cor [u] = cinza

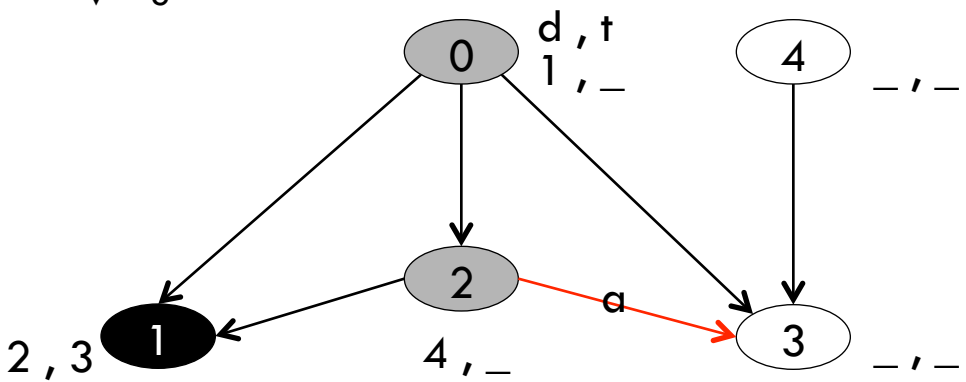
a = aresta (2 -> 3)
v = 1



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 4
cor [u] = cinza

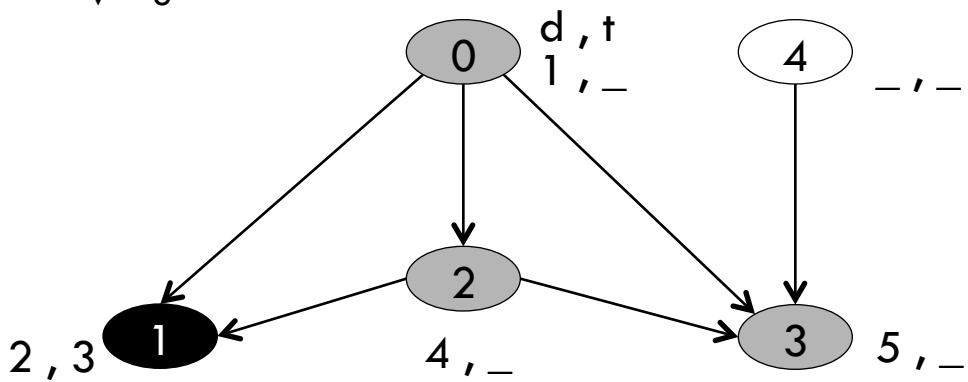
a = aresta (2 -> 3)
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
     cor[u] = cinza; this.d[u] = ++tempo;

     if (!this.grafo.listaAdjVazia (u))
     {
          Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
           while (a != null)
           {
                int v = a. get_v2 ();
                if (cor[v] == branco)
                {
                    this.antecessor[v] = u;
                    tempo = this.visitaDfs (v, tempo, cor);
                }
                a = this.grafo.proxAdj (u);
           }
      }
      cor[u] = preto; this.t[u] = ++tempo;
      return tempo;
}
```

u = 3
tempo = 5
cor [u] = cinza

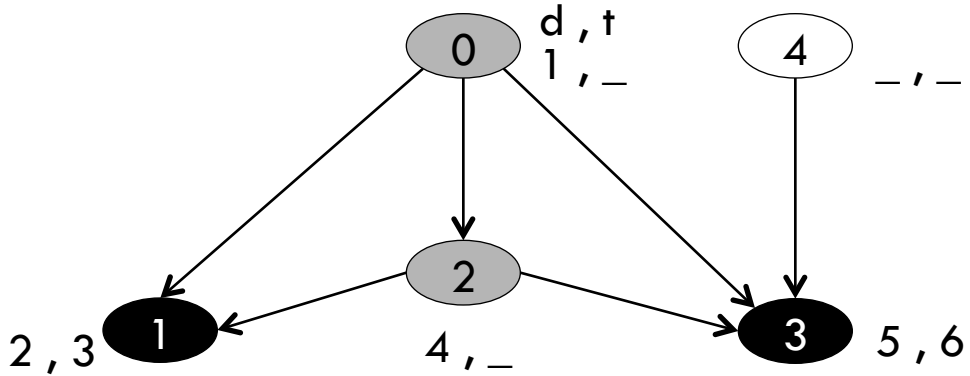
a = aresta (2 -> 3)
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                      this.antecessor[v] = u;
                      tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 3
tempo = 6
cor [u] = preto

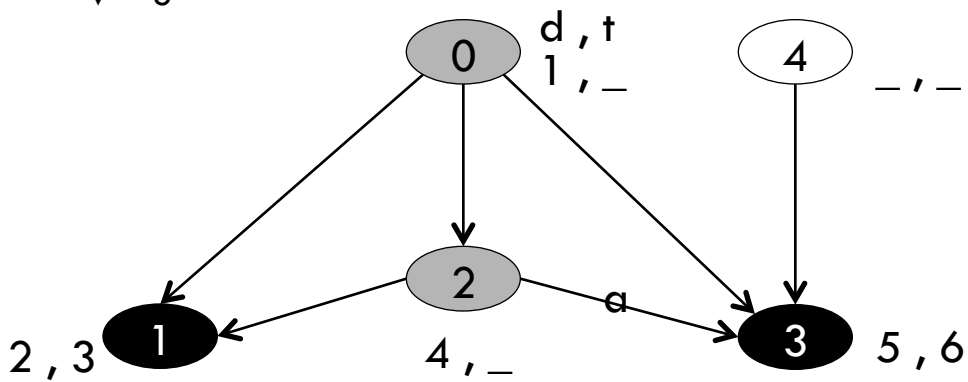
a = aresta (2 -> 3)
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 6
cor [u] = cinza

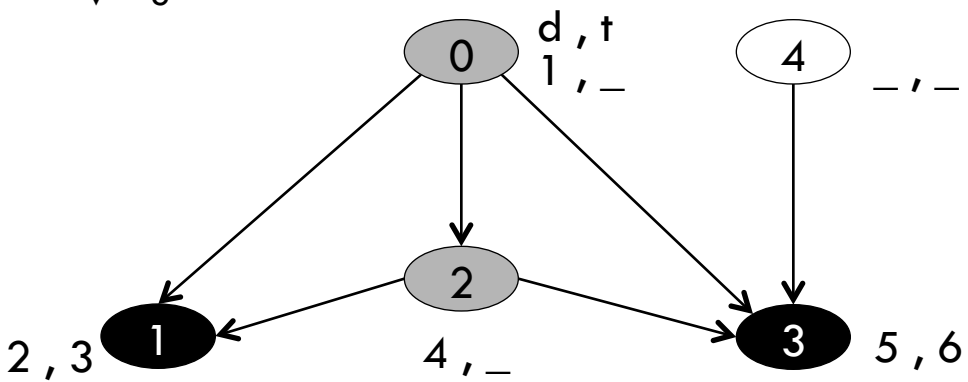
a = aresta (2 -> 3)
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 6
cor [u] = cinza

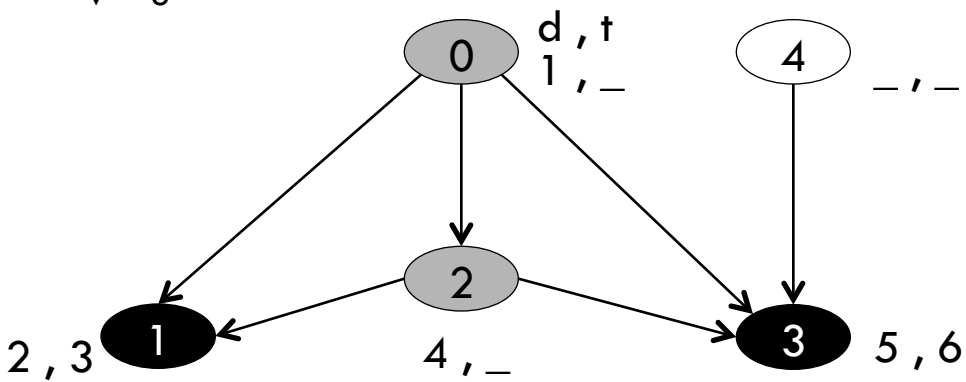
a = NULL
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                      this.antecessor[v] = u;
                      tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 6
cor [u] = cinza

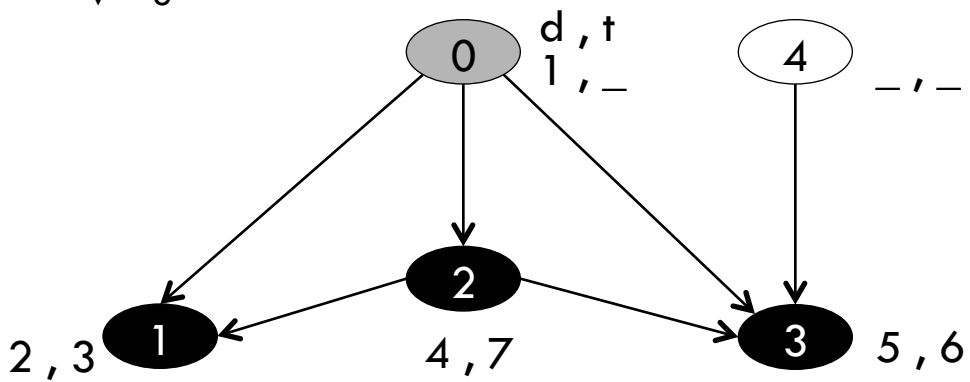
a = NULL
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                      this.antecessor[v] = u;
                      tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 2
tempo = 7
cor [u] = preto

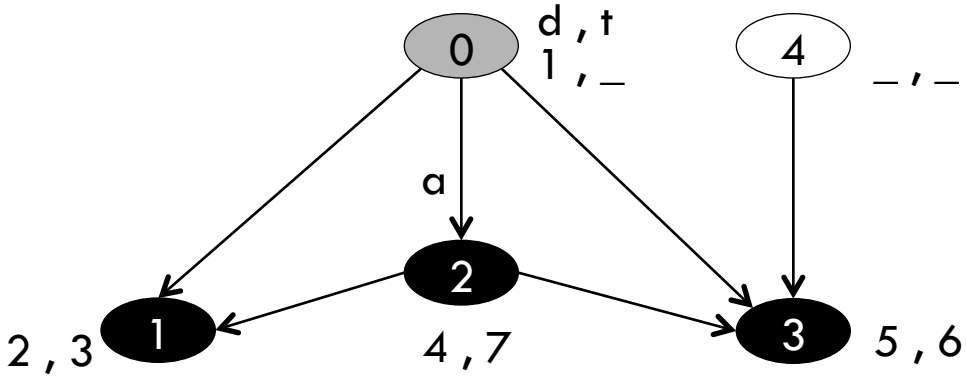
a = NULL
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 7
cor [u] = cinza

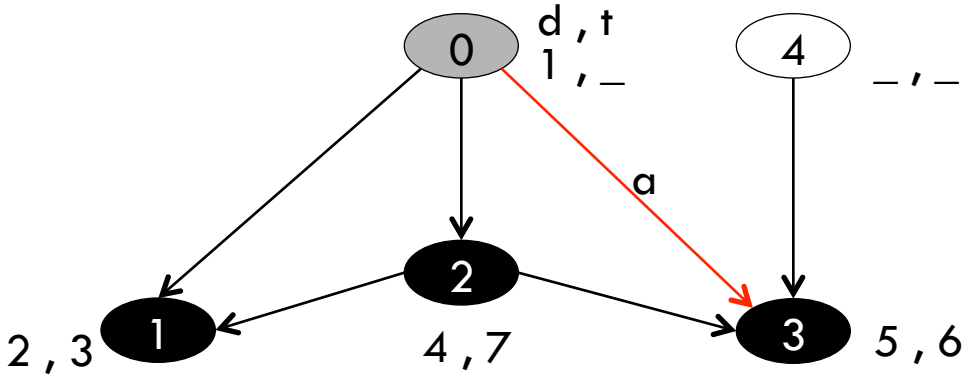
a = aresta (0 -> 2)
v = 2



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 7
cor [u] = cinza

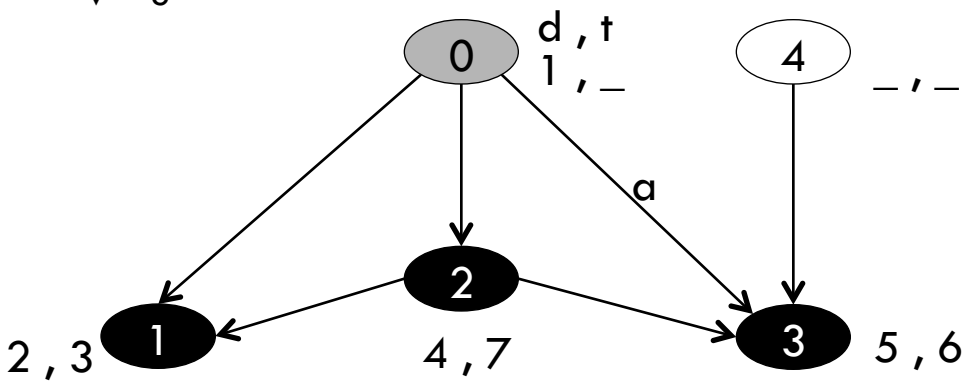
a = aresta (0 -> 3)
v = 2



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                      this.antecessor[v] = u;
                      tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 7
cor [u] = cinza

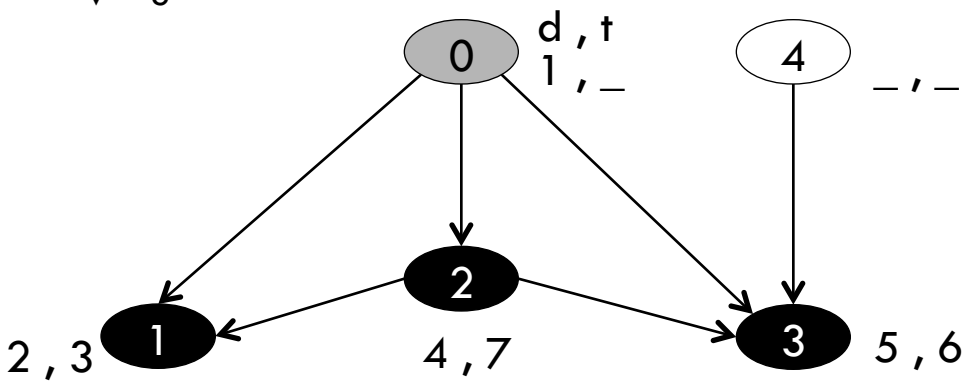
a = aresta (0 -> 3)
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 7
cor [u] = cinza

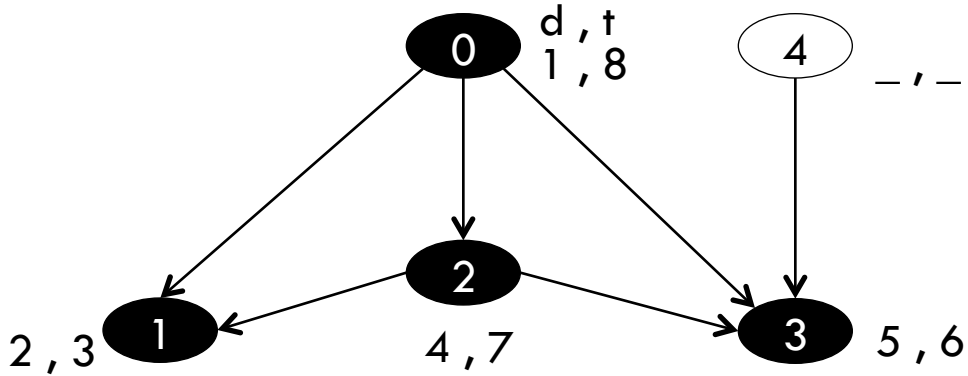
a = NULL
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
      cor[u] = cinza; this.d[u] = ++tempo;

      if (!this.grafo.listaAdjVazia (u))
      {
            Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
             while (a != null)
             {
                   int v = a. get_v2 ();
                   if (cor[v] == branco)
                   {
                       this.antecessor[v] = u;
                       tempo = this.visitaDfs (v, tempo, cor);
                   }
                   a = this.grafo.proxAdj (u);
             }
       }
       cor[u] = preto; this.t[u] = ++tempo;
       return tempo;
}
```

u = 0
tempo = 8
cor [u] = preto

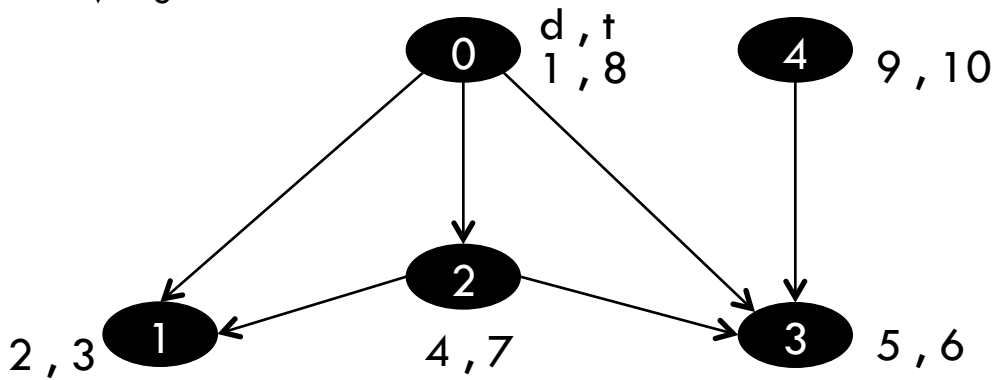
a = NULL
v = 3



```
private int visitaDfs (int u, int tempo, int []cor)  {
     cor[u] = cinza; this.d[u] = ++tempo;

     if (!this.grafo.listaAdjVazia (u))
     {
          Grafo.Aresta a = this.grafo.primeiroListaAdj (u);
           while (a != null)
           {
                int v = a. get_v2 ();
                if (cor[v] == branco)
                {
                    this.antecessor[v] = u;
                    tempo = this.visitaDfs (v, tempo, cor);
                }
                a = this.grafo.proxAdj (u);
           }
      }
      cor[u] = preto; this.t[u] = ++tempo;
      return tempo;
}
```

u = 0
tempo = 8
cor [u] = preto

a = NULL
v = 3



Neste passo, o mesmo procedimento é repetido para o vértice 4, até que a função seja finalizada.

Classificação das arestas no DFS

- **Arestas de árvore:** são arestas de uma árvore de busca em profundidade. A aresta (u, v) é uma aresta de árvore se v foi descoberto pela primeira vez ao percorrer a aresta (u, v). Ou seja, quando encontra um vértice branco
- **Arestas de retorno**: conectam um vértice u com um antecessor v em uma árvore de busca em profundidade (inclui *self-loops*). De um descendente para um ancestral (Encontra um vértice cinza – sendo cinza para cinza)

Classificação das arestas no DFS

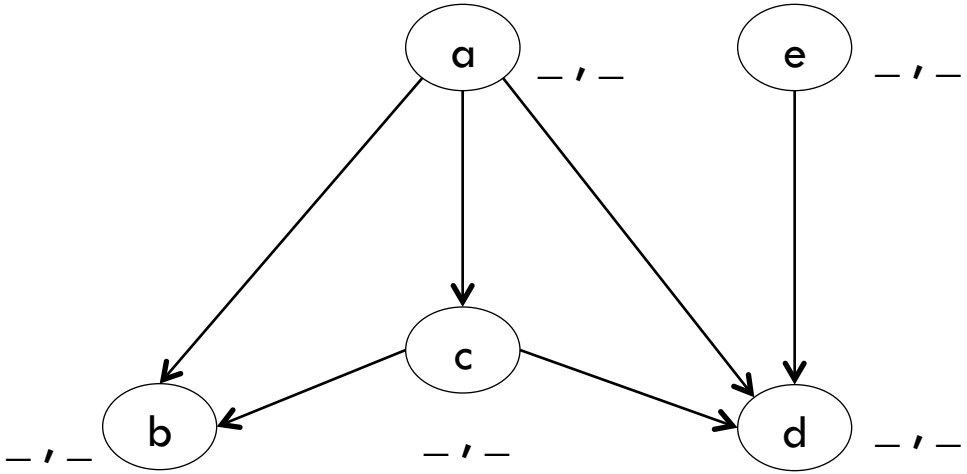
- **Arestas de avanço**: ***não pertencem à árvore de busca em profundidade*** mas conectam um vértice a um descendente que pertence à árvore de busca em profundidade (de um ancestral para um descendente, quando encontra um vértice preto)
- **Arestas de cruzamento**: todas as outras arestas que podem conectar vértices na mesma árvore de busca em profundidade, ou em duas árvores diferentes (de um vértice cinza para um vértice preto)

Classificação das arestas no DFS

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

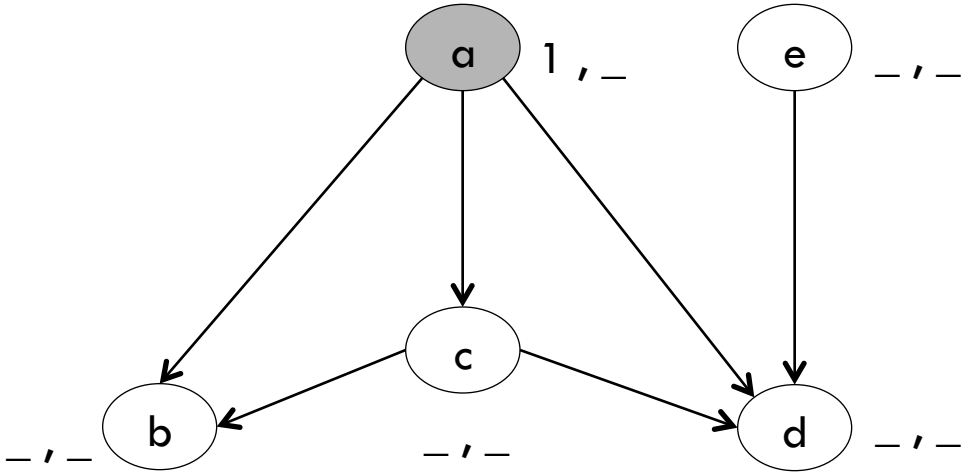
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



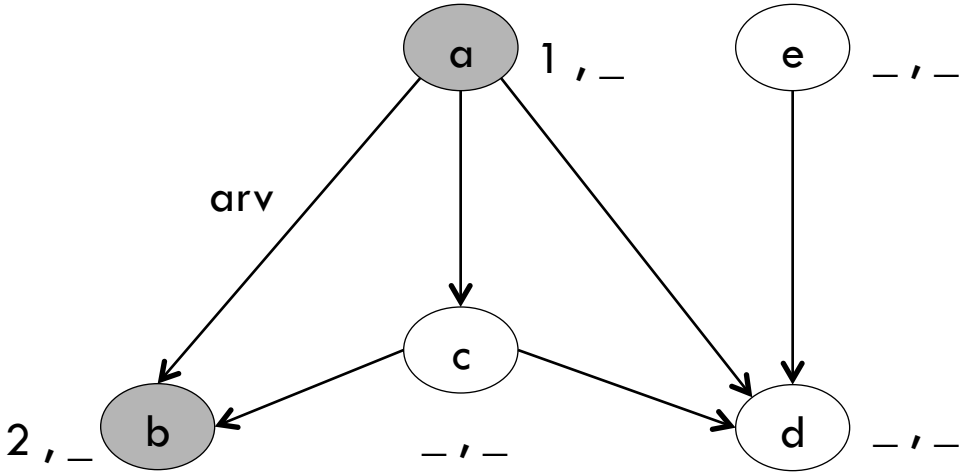
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



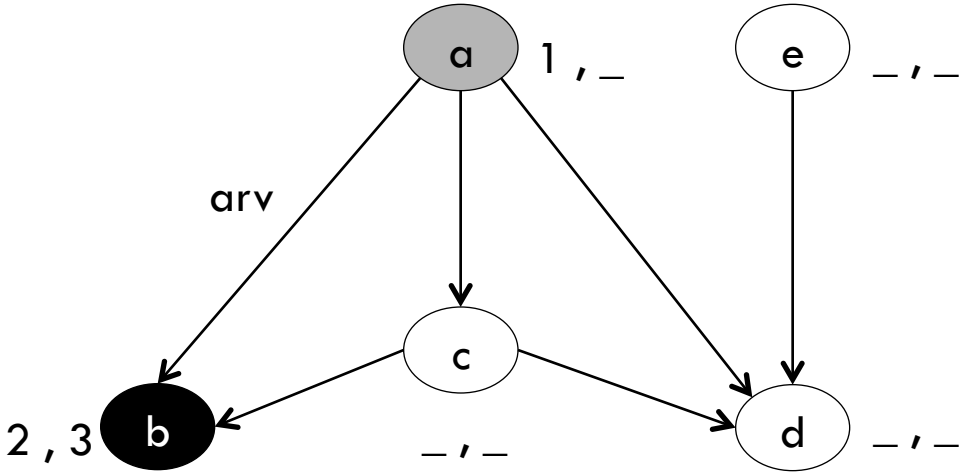
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



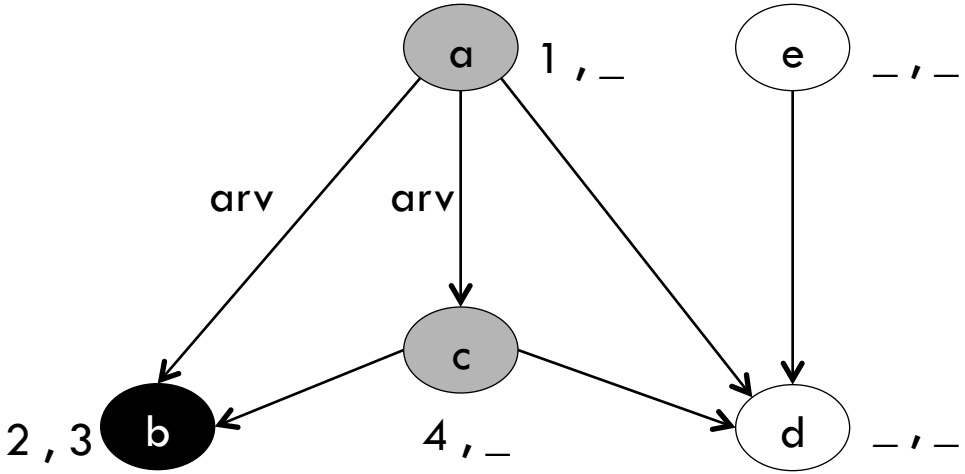
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



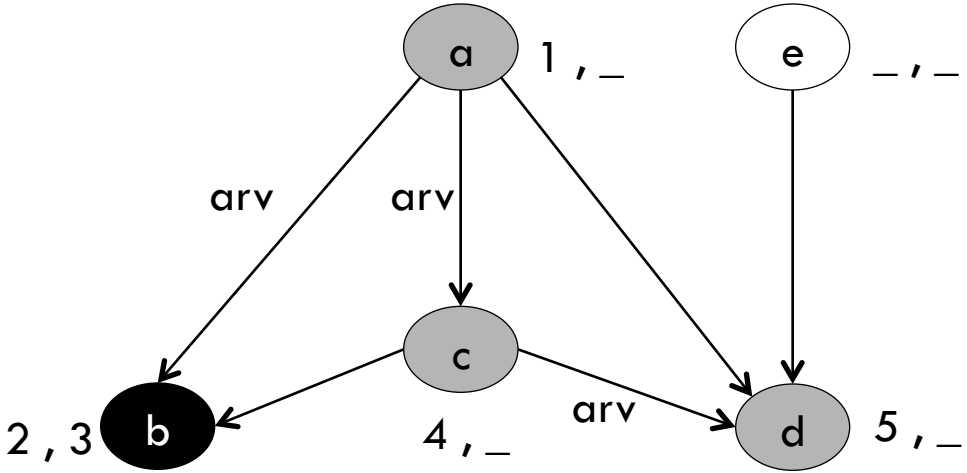
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



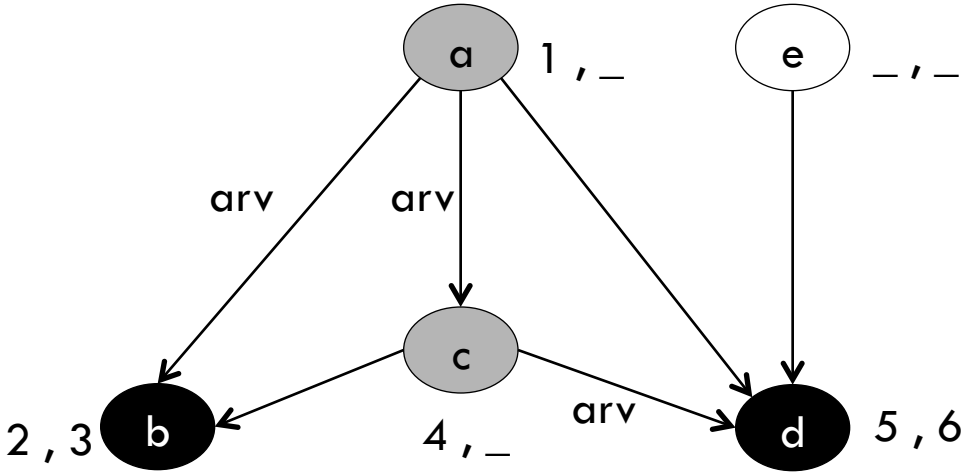
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



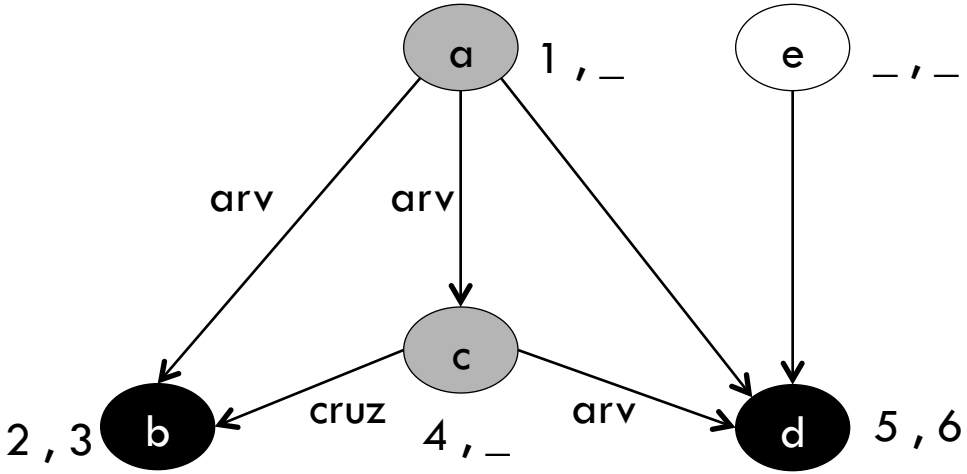
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



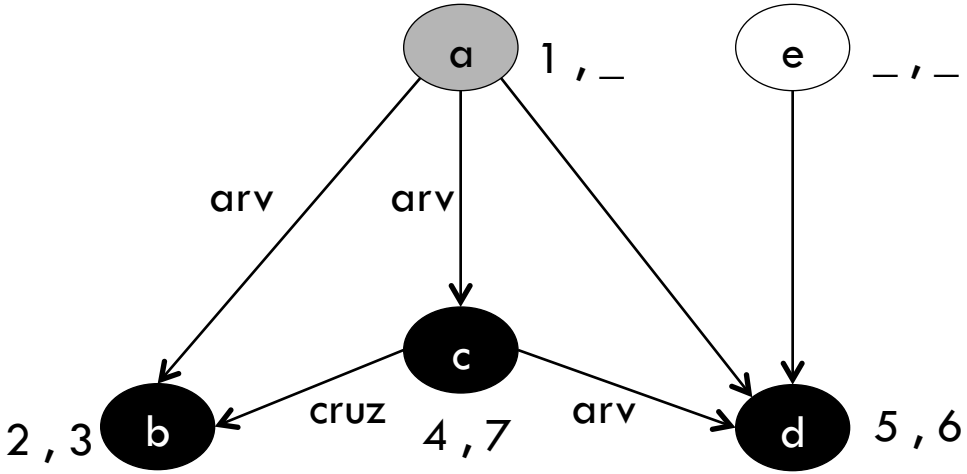
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



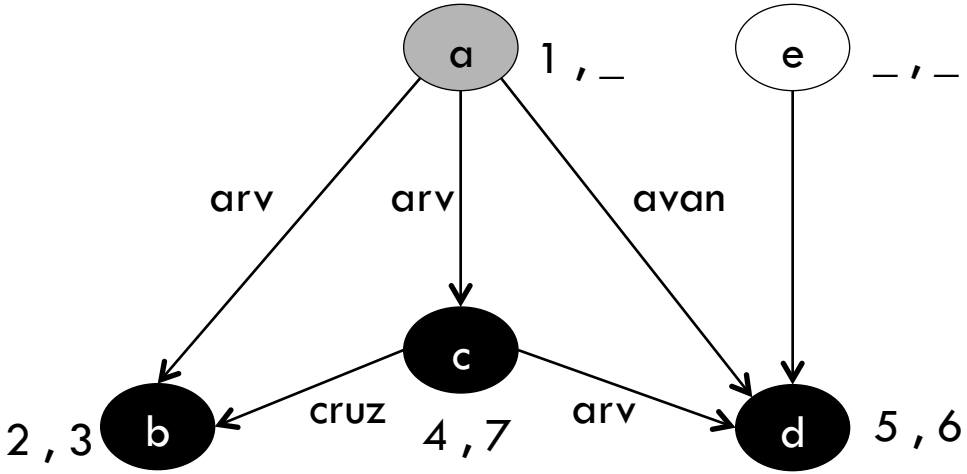
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



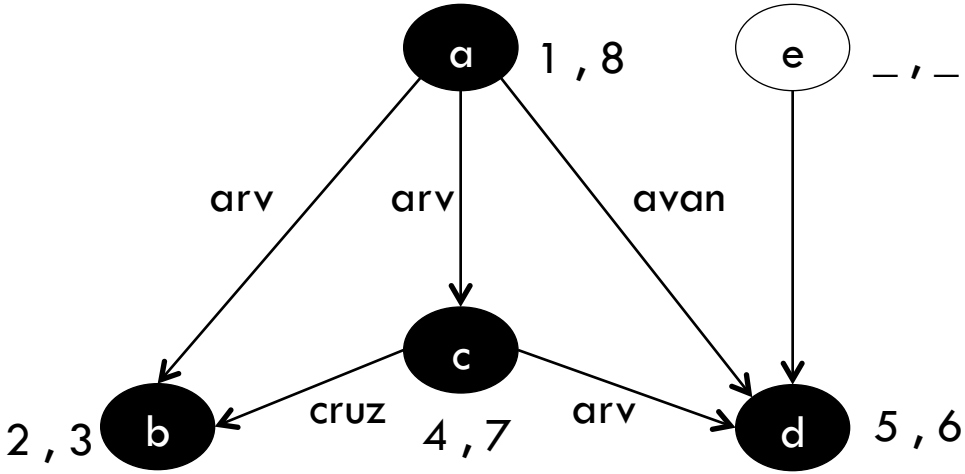
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



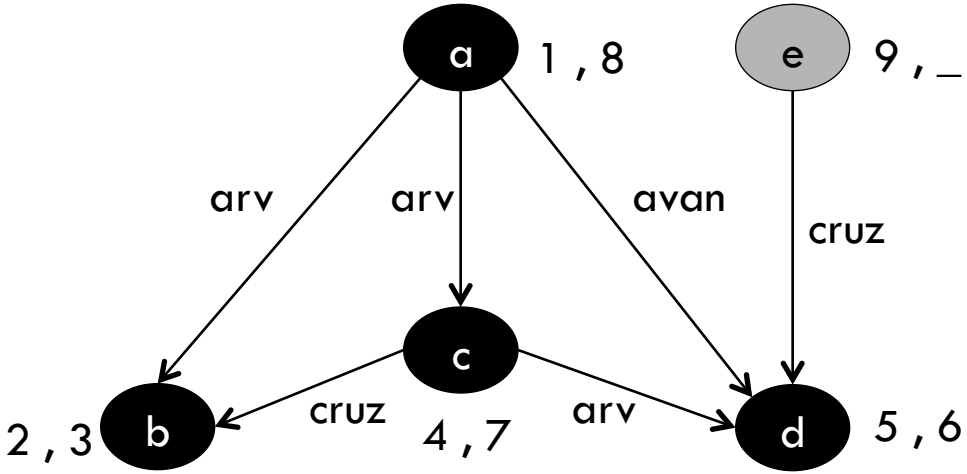
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



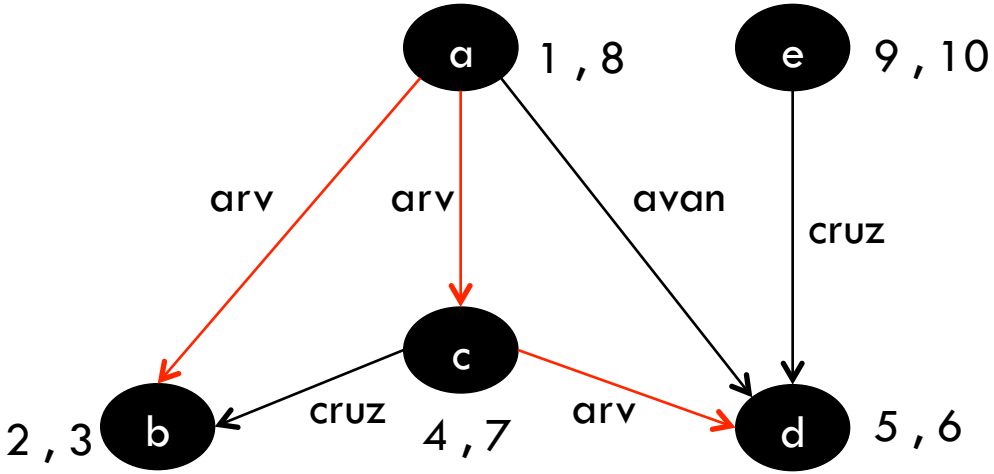
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



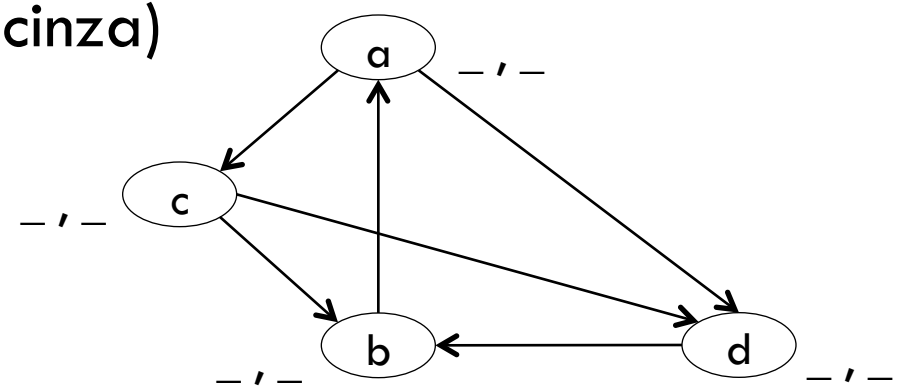
Classificação das arestas no DFS

- **Arestas de árvore:** encontra um vértice branco
- **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)
- **Arestas de avanço**: encontra um vértice preto (u é descoberto antes de v)
- **Arestas de cruzamento**: de um vértice cinza para um vértice preto



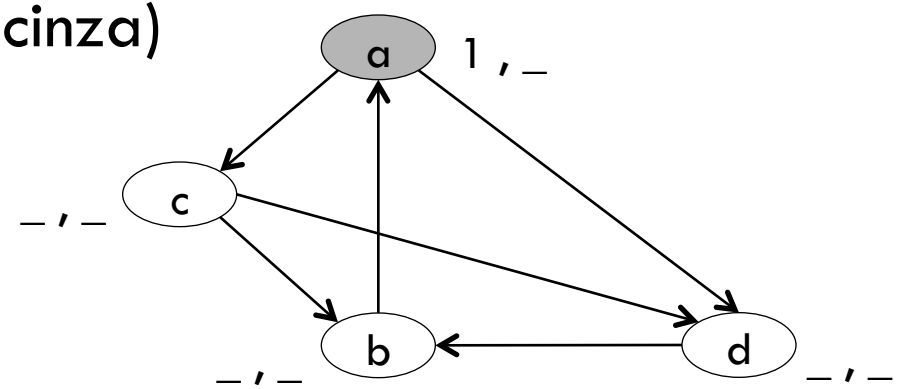
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)



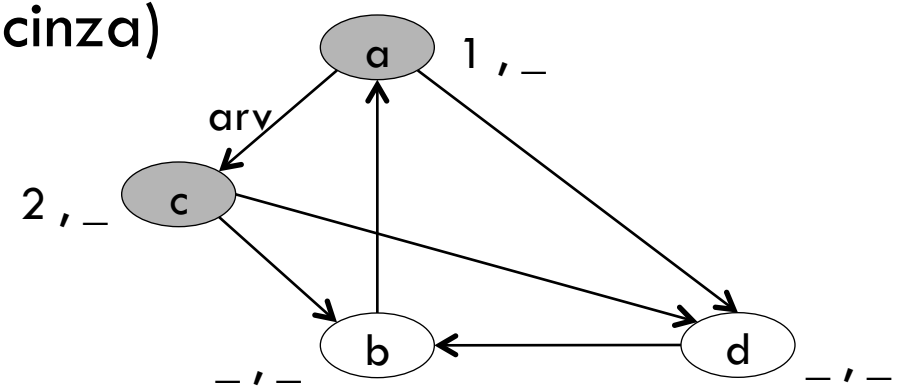
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)



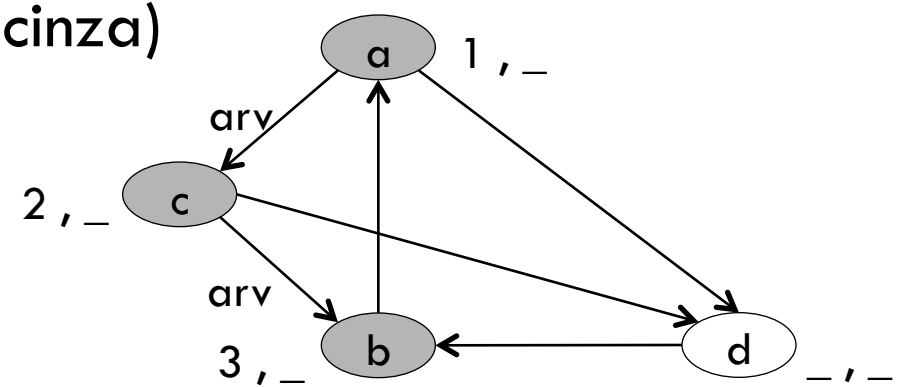
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



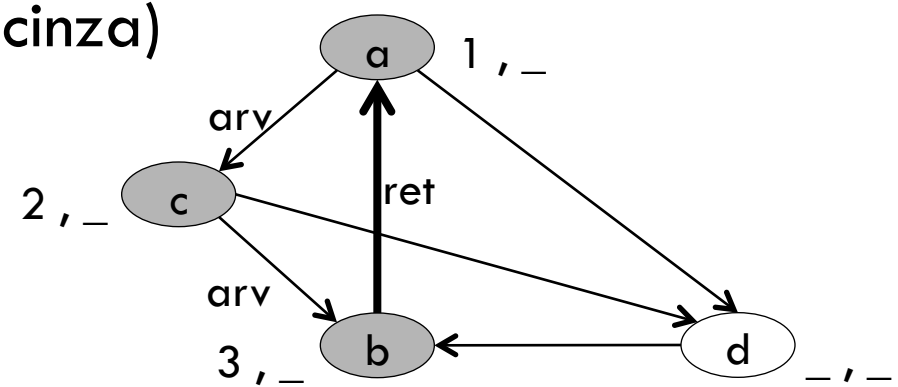
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)



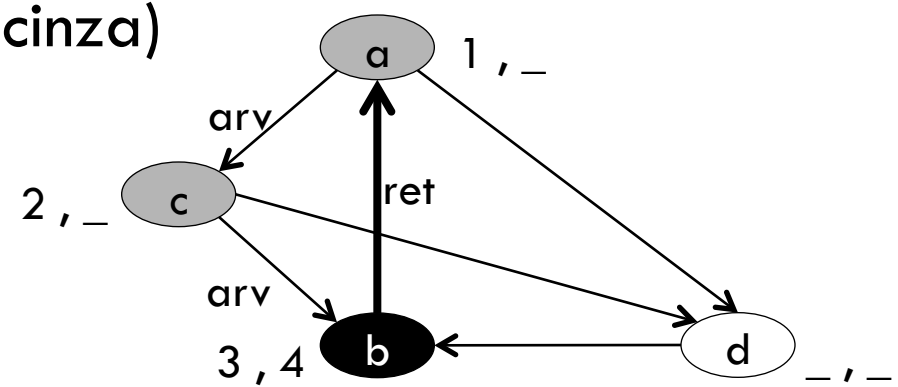
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



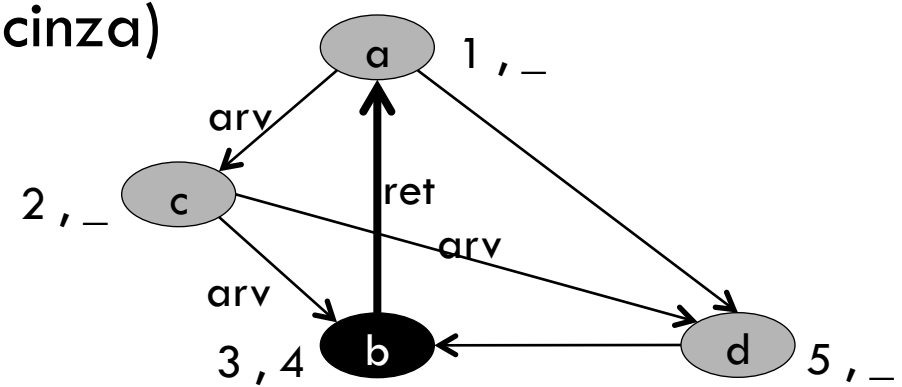
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



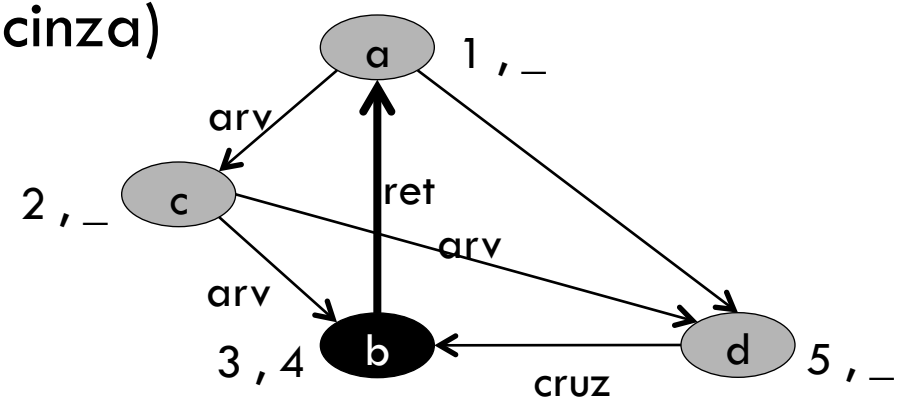
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



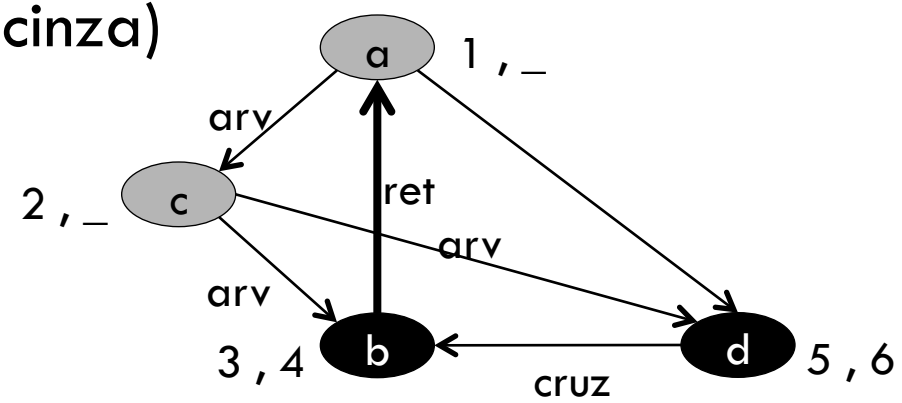
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



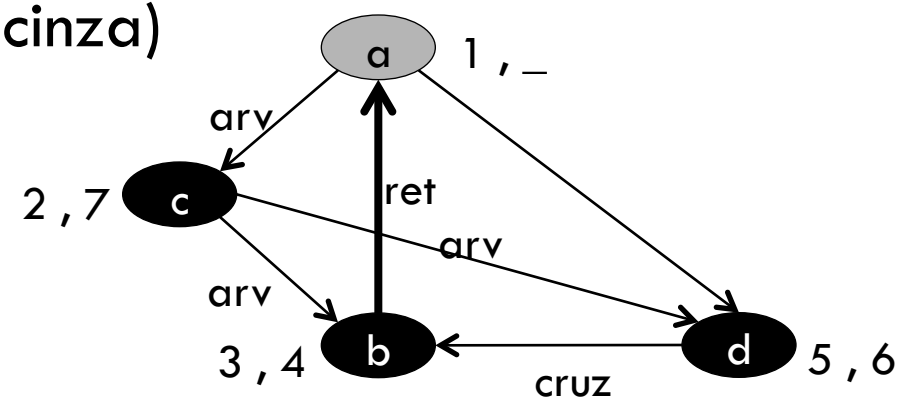
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



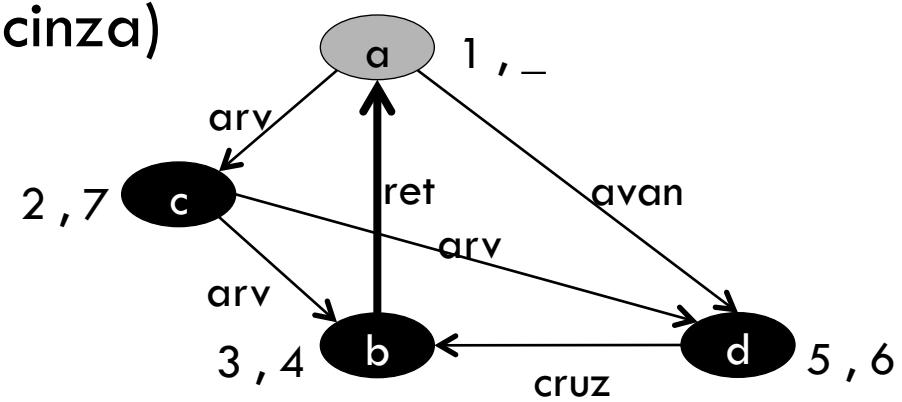
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno**: Encontra um vértice cinza (cinza para cinza)



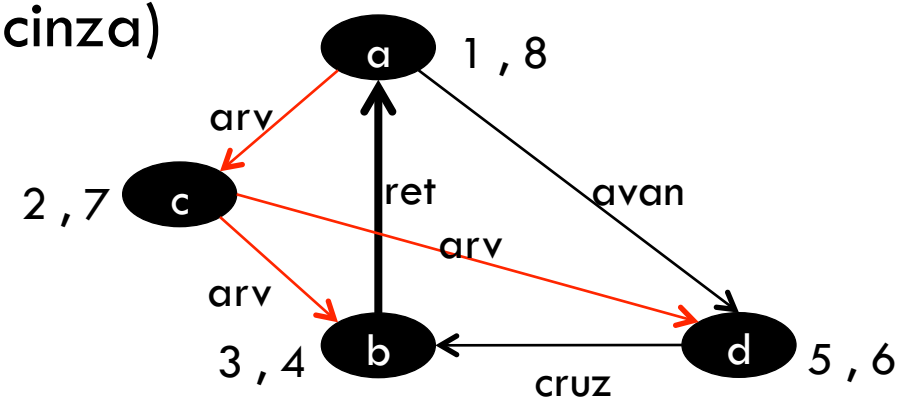
Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



Busca em Profundidade

- Exemplo de uma aresta de retorno:
  - **Arestas de retorno:** Encontra um vértice cinza (cinza para cinza)



Busca em Profundidade

- Teste de circuito
  - Se uma **aresta de retorno** é encontrada na busca em profundidade então o grafo possui um **ciclo**
  - Um grafo é acíclico se e somente se na busca em profundidade não for encontrada nenhuma **aresta de retorno**

Aplicação

- Busca em profundidade: verificar se um grafo é acíclico
- Grafo acíclico: Rede Bayesiana
- Redes de opinião, causais, baseadas na incerteza
  - Vértices: variáveis
  - Arcos: conexões

Aplicação

- Rede Bayesiana – principal característica
  - Adaptabilidade
  - A partir de novas informações e com base em informações verdadeiras, gerar alterações nas dependências e nos seus conceitos

Aplicação

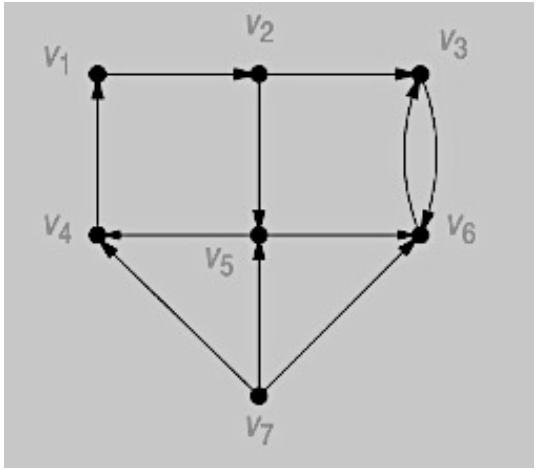
- Jogos de simulação: fins educacionais, treinamento e entretenimento

Fecho transitivo direto

- **Fecho transitivo direto** (FTD) de um vértice **v**:
  - conjunto de todos os **vértices que podem ser atingidos** por algum **caminho iniciado em v**

Fecho transitivo direto

- Exemplo:
  - o FTD do vértice v_5 do grafo abaixo é:
    - $\{v_1, v_2, v_3, v_4, v_5, v_6\}$.

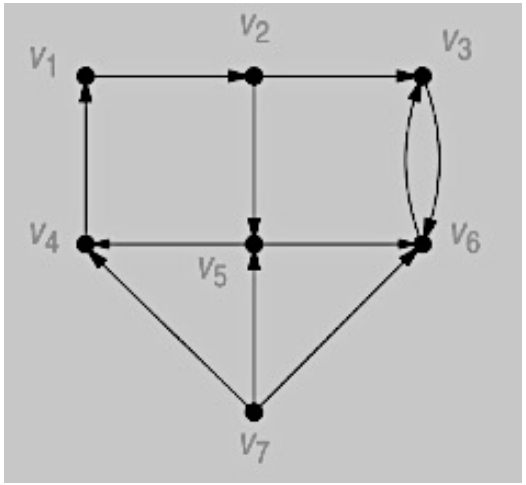


Fecho transitivo inverso

- **Fecho transitivo inverso** (FTI) de um vértice **v:**
  - conjunto de todos os **vértices a partir dos quais se pode atingir v por algum caminho**

Fecho transitivo inverso

- Exemplo:
  - o FTI do vértice v_5 do grafo abaixo é:
    - $\{v_1, v_2, v_4, v_5, v_7\}$.



Ordenação topológica

- Grafos acíclicos direcionados podem ser usados para indicar precedência de eventos: ordenação topológica
- É uma ordenação linear de seus vértices, na qual cada vértice aparece antes de seus descendentes

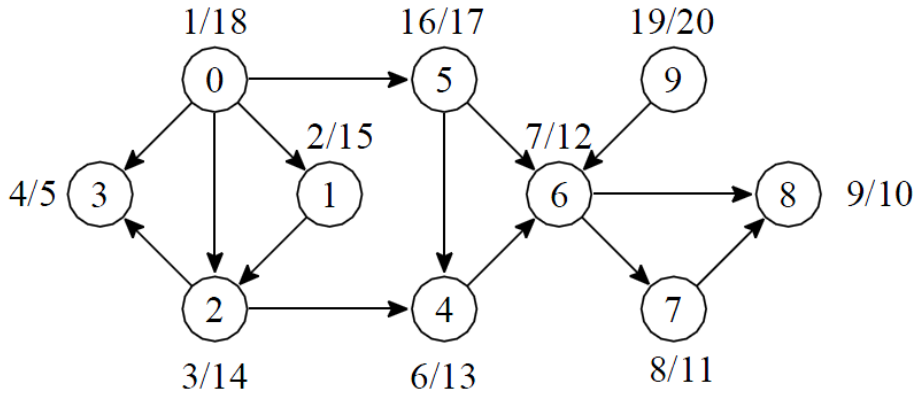
Ordenação topológica

- Cada GAD possui uma ou mais ordenações topológicas
- Caso um grafo possua ciclos, não é possível estabelecer uma relação de precedência entre os vértices, e portanto, é impossível estabelecer uma ordenação topológica

Ordenação topológica

- Os vértices ordenados topologicamente aparecem em **ordem inversa** aos seus **tempos de término do exame da lista de vértices adjacentes**, t[u]
  - na **travessia em profundidade**

Ordenação topológica



[Ziviani, 2006]

Ordenação topológica

- Algoritmo para ordenar topologicamente um grafo direcionado acíclico $G = (V, A)$.
  - Aplicar a busca em profundidade no grafo G para obter os tempos de término $t[u]$ para cada vértice u.
  - Ao término de cada vértice, insira-o na frente de uma lista linear encadeada.
  - Retornar a lista encadeada de vértices.

- A Custo $O(|V| + |A|)$, uma vez que a busca em profundidade tem complexidade de tempo $O(|V| + |A|)$ e o custo para inserir cada um dos $|V|$ vértices na frente da lista linear encadeada custa $O(1)$.

Ordenação topológica – aplicações

- Dependência entre tarefas:
  - vértices: tarefas a serem realizadas
  - arestas: restrições de dependência entre as tarefas
  - ordenação topológica: sequência válida para execução das tarefas

Ordenação topológica – aplicações

- Pré-requisitos de disciplinas:
  - vértices: disciplinas
  - arestas: pré-requisitos entre as disciplinas
  - ordenação topológica: sequência válida para se cursar as disciplinas

Ordenação topológica

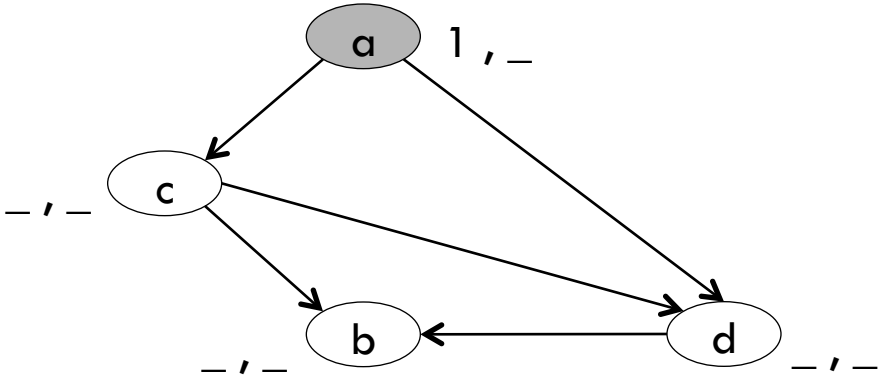
- Deve-se inserir uma chamada ao método $inserePrimeiro$ no método $buscaDfs$, logo após o momento em que o tempo de término $t[u]$ é obtido e o vértice é pintado de preto
- Ao final, basta retornar a lista obtida

```
// Insere antes do primeiro item da lista
public void inserePrimeiro (int v) {
        Celula item = new Celula(v, 0);

        CelulaRef aux = this.primeiro.prox;
        this.primeiro.prox = new CelulaRef();
        this.primeiro.prox.item = item;
        this.primeiro.prox.prox = aux;
}
```

Ordenação topológica

- Exemplo (grafo deve ser orientado e acíclico):

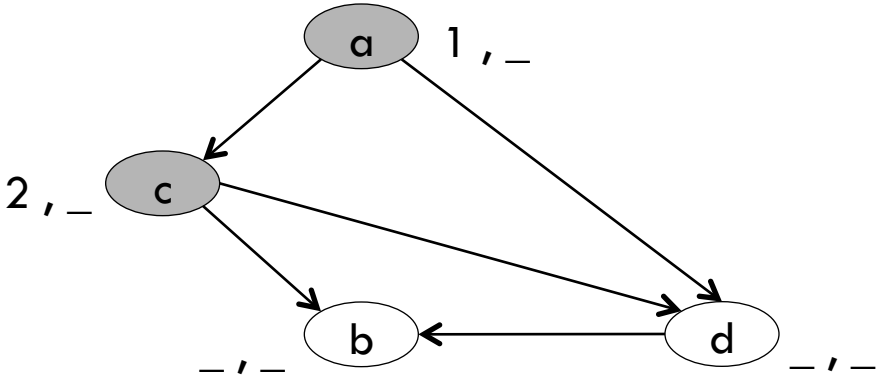
Lista: null



Ordenação topológica

- Exemplo (grafo deve ser orientado e acíclico):

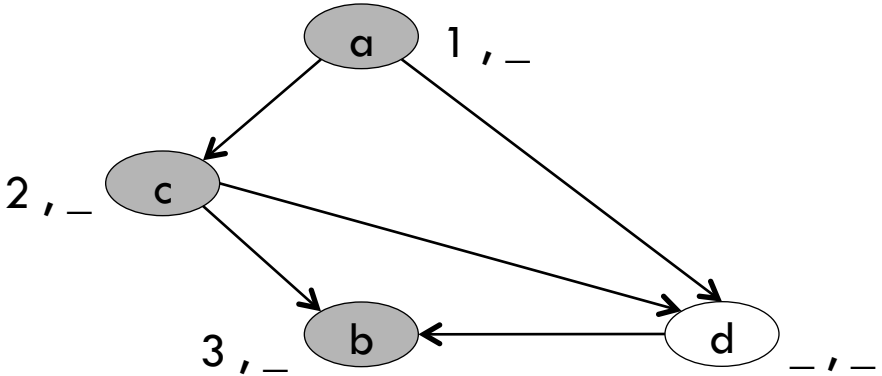
Lista: null



Ordenação topológica

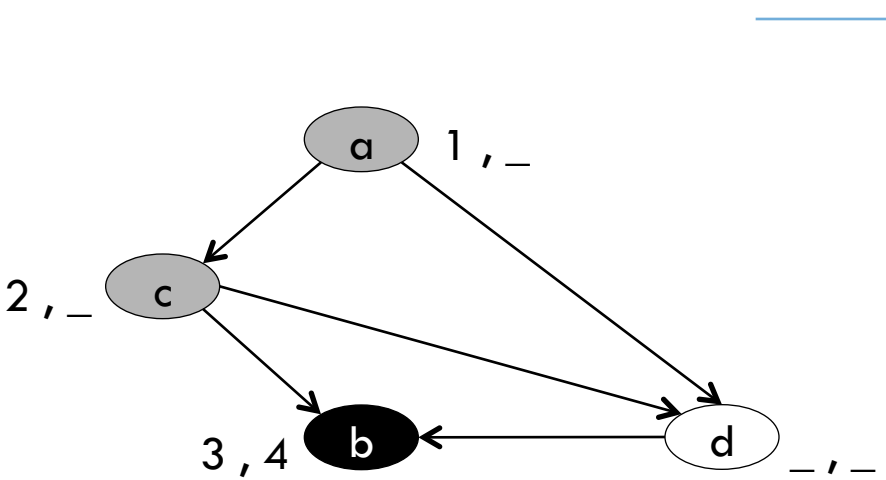
- Exemplo (grafo deve ser orientado e acíclico):

Lista: null



Ordenação topológica

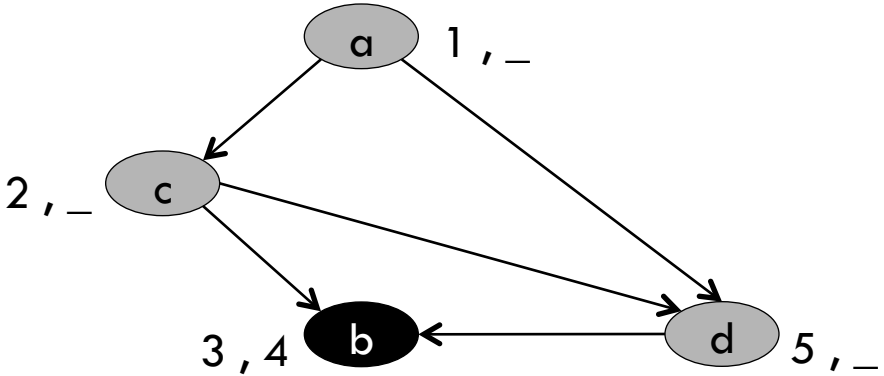
- Exemplo (grafo deve ser orientado e acíclico):



Ordenação topológica

- Exemplo (grafo deve ser orientado e acíclico):

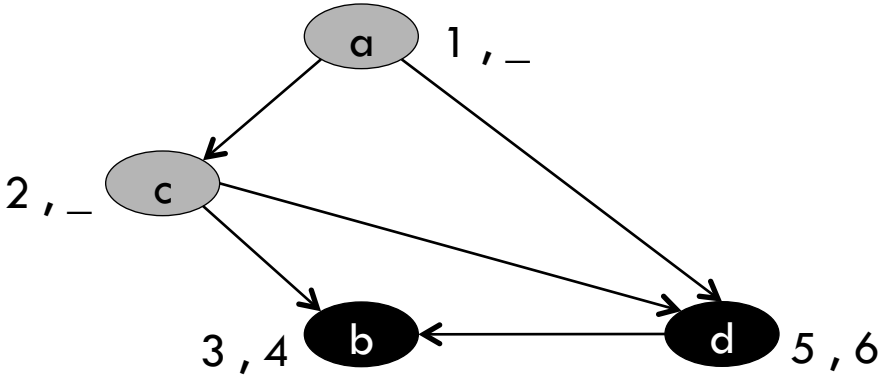
Lista: b -> null



Ordenação topológica

- Exemplo (grafo deve ser orientado e acíclico):

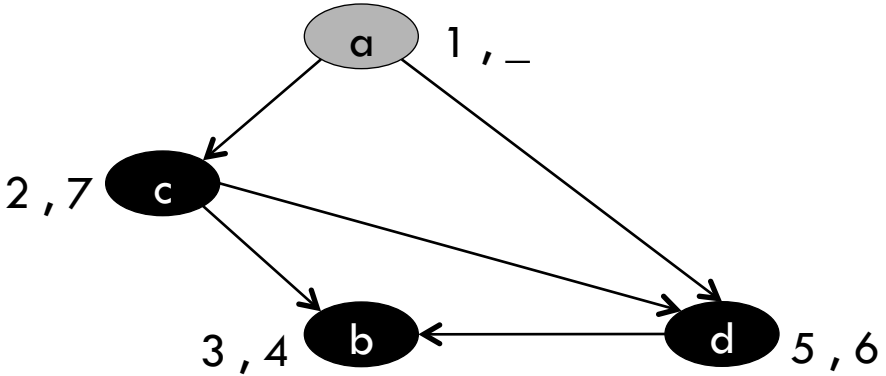
Lista: d -> b -> null



Ordenação topológica

- Exemplo (grafo deve ser orientado e acíclico):

Lista: c -> d -> b -> null



Ordenação topológica

- Exemplo (grafo deve ser orientado e acíclico):

Lista: a -> c -> d -> b -> null