

# Grafos

*Pontes e Pontos de Articulação*

**Prof. Edson Alves**

**Faculdade UnB Gama**

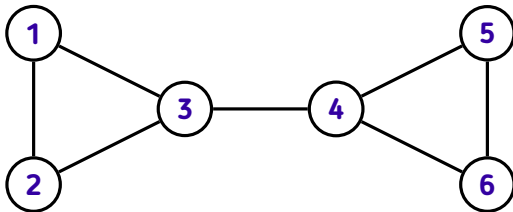
**Pontes**

# Pontes

Seja  $G(V, E)$  um grafo não-direcionado conectado. Uma aresta  $e \in E$  é uma ponte se a exclusão de  $e$  torna o grafo  $G$  desconectado.

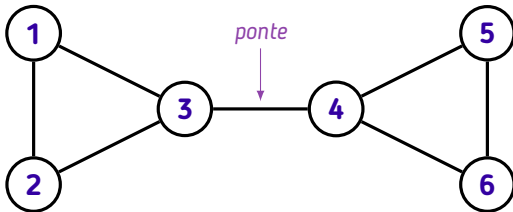
# Pontes

Seja  $G(V, E)$  um grafo não-direcionado conectado. Uma aresta  $e \in E$  é uma ponte se a exclusão de  $e$  torna o grafo  $G$  desconectado.



# Pontes

Seja  $G(V, E)$  um grafo não-direcionado conectado. Uma aresta  $e \in E$  é uma **ponte** se a exclusão de  $e$  torna o grafo  $G$  desconectado.



**Árvore gerada pela DFS**

# Árvore gerada pela DFS

- ★ Uma DFS em um grafo  $G$  gera uma árvore

# Árvore gerada pela DFS

- ★ Uma DFS em um grafo  $G$  gera uma árvore
- ★ Os vértices são os mesmos de  $G$



# Árvore gerada pela DFS

- ★ Uma DFS em um grafo  $G$  gera uma árvore
- ★ Os vértices são os mesmos de  $G$
- ★ As arestas dependem da ordem de descoberta dos vértices

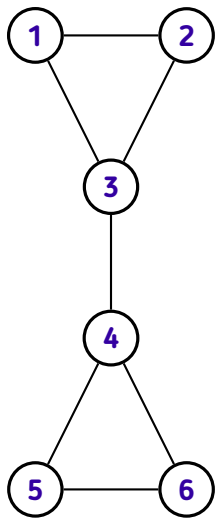
# Árvore gerada pela DFS

- ★ Uma DFS em um grafo  $G$  gera uma árvore
- ★ Os vértices são os mesmos de  $G$
- ★ As arestas dependem da ordem de descoberta dos vértices
- ★ Esta ordem também determina uma permutação dos vértices

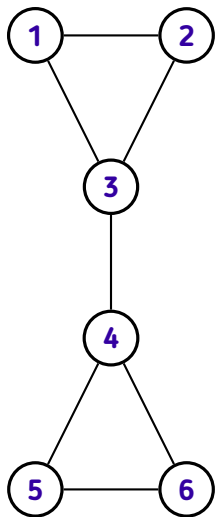
# Árvore gerada pela DFS

- ★ Uma DFS em um grafo  $G$  gera uma árvore
- ★ Os vértices são os mesmos de  $G$
- ★ As arestas dependem da ordem de descoberta dos vértices
- ★ Esta ordem também determina uma permutação dos vértices
- ★ O índice de cada vértice nesta permutação tem importantes propriedades

## Grafo

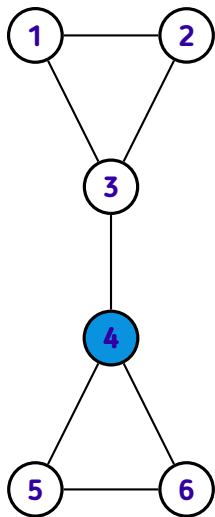


**Grafo**



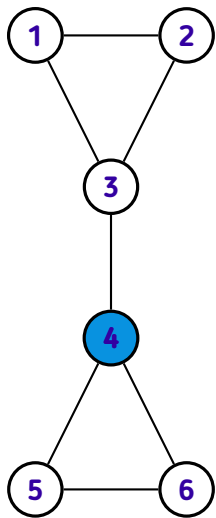
**Árvore gerada pelo DFS**

**Grafo**



**Árvore gerada pelo DFS**

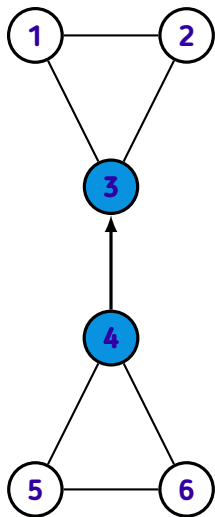
**Grafo**



**Árvore gerada pelo DFS**



**Grafo**

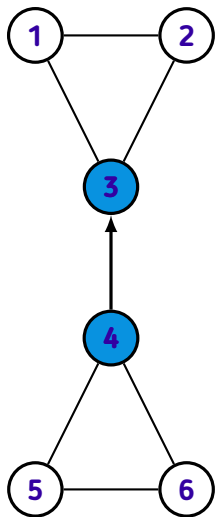


**Árvore gerada pelo DFS**

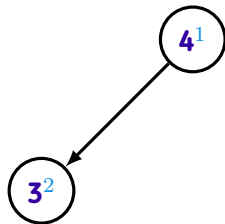




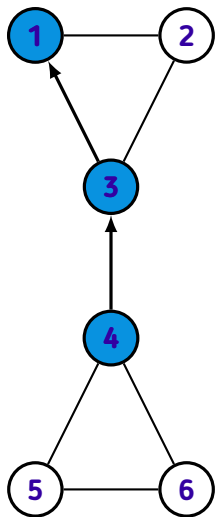
**Grafo**



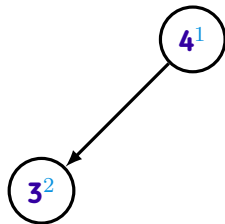
**Árvore gerada pelo DFS**



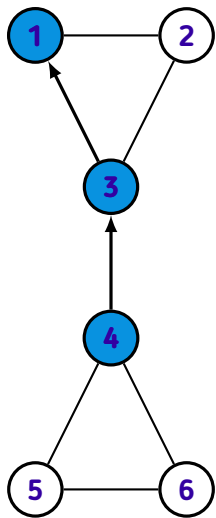
**Grafo**



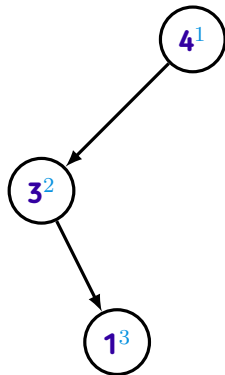
**Árvore gerada pelo DFS**



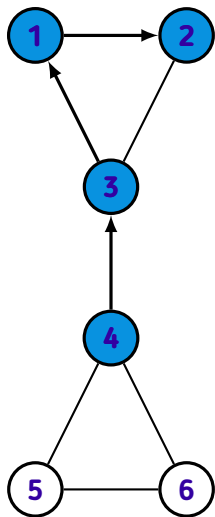
**Grafo**



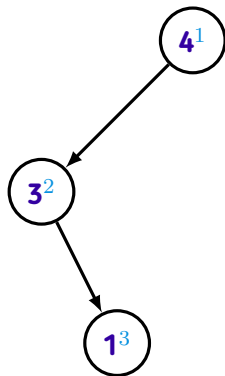
**Árvore gerada pelo DFS**



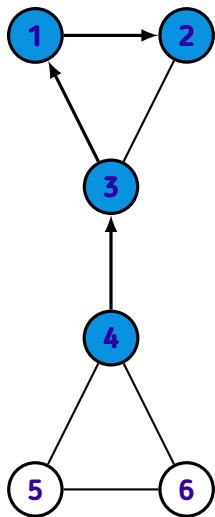
**Grafo**



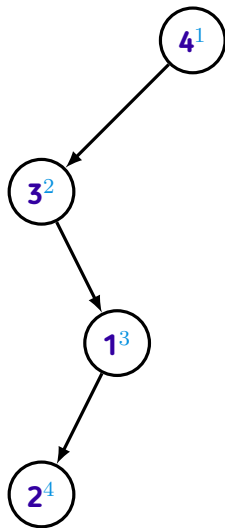
**Árvore gerada pelo DFS**



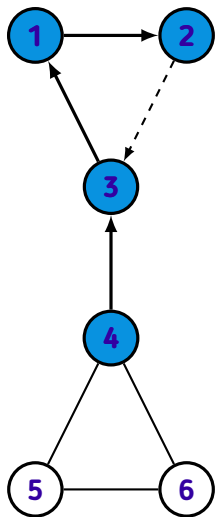
**Grafo**



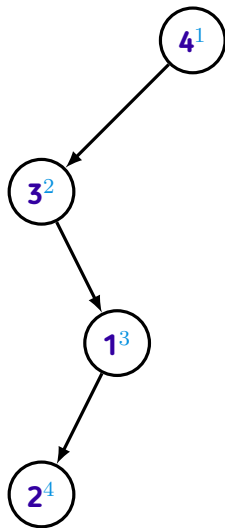
**Árvore gerada pelo DFS**



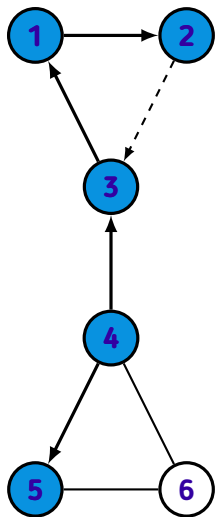
Grafo



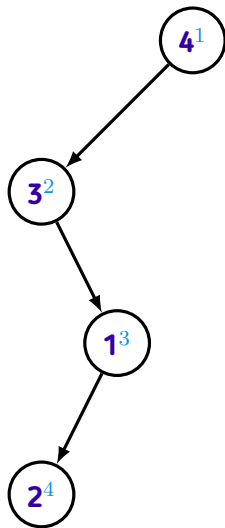
Árvore gerada pelo DFS



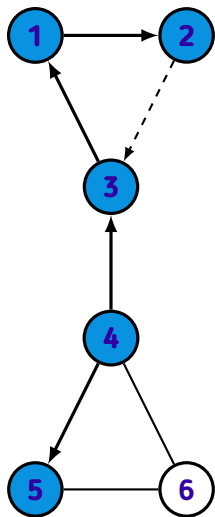
**Grafo**



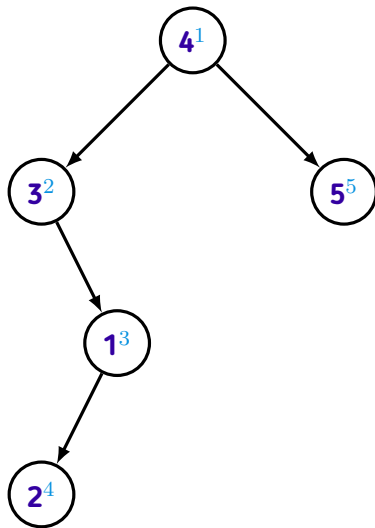
**Árvore gerada pelo DFS**



**Grafo**

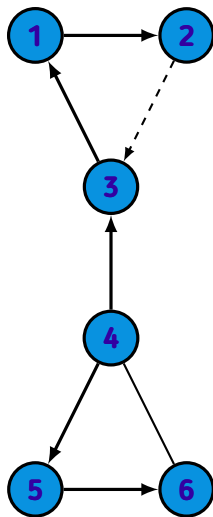


**Árvore gerada pelo DFS**

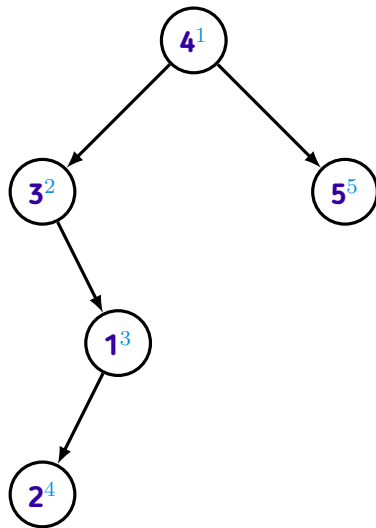




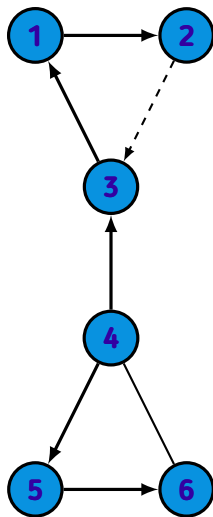
**Grafo**



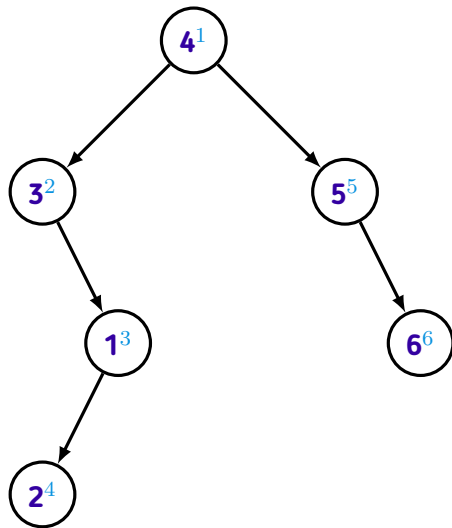
**Árvore gerada pelo DFS**



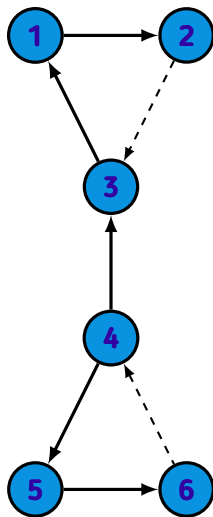
**Grafo**



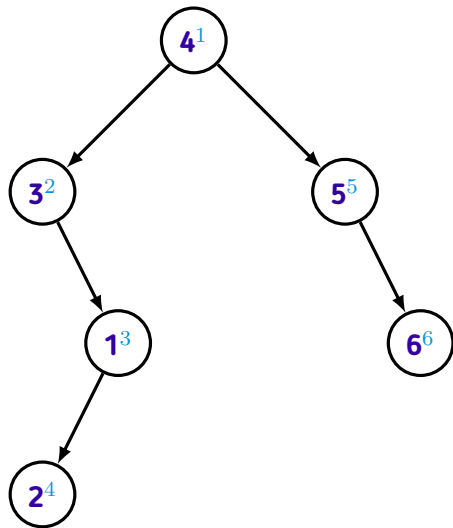
**Árvore gerada pelo DFS**



**Grafo**



**Árvore gerada pelo DFS**



## **Propriedades dos índices da permutação**

## Propriedades dos índices da permutação

★ Seja  $i_s(u)$  o índice do vértice  $u$  na permutação gerada pela DFS que tem  $s$  como vértice inicial

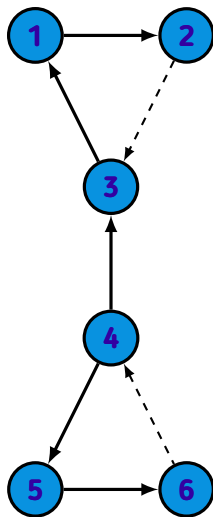
## Propriedades dos índices da permutação

- ★ Seja  $i_s(u)$  o índice do vértice  $u$  na permutação gerada pela DFS que tem  $s$  como vértice inicial
- ★ Se  $i_s(u) < i_s(v)$  então ou  $u$  é ancestral de  $v$  ou  $u$  está em uma subárvore de  $s$  distinta da subárvore que contém  $v$

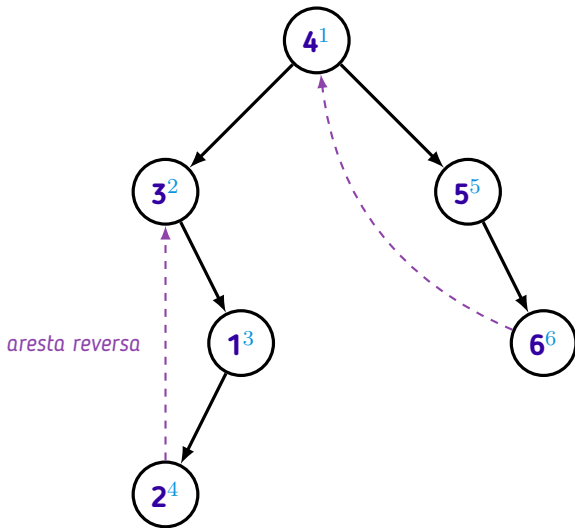
## Propriedades dos índices da permutação

- ★ Seja  $i_s(u)$  o índice do vértice  $u$  na permutação gerada pela DFS que tem  $s$  como vértice inicial
- ★ Se  $i_s(u) < i_s(v)$  então ou  $u$  é ancestral de  $v$  ou  $u$  está em uma subárvore de  $s$  distinta da subárvore que contém  $v$
- ★ Se  $(u, v) \in E$  e  $i_s(v) < i_s(u)$ , então  $(u, v)$  é uma aresta reversa

Grafo



Árvore gerada pelo DFS





**Menor ancestral alcançável**

## Menor ancestral alcançável

★ Seja  $\mu_s(u)$  o menor índice dentre todos os vértices atingíveis a partir da subárvore (ou subgrafo) cuja raiz é  $u$

## Menor ancestral alcançável

- ★ Seja  $\mu_s(u)$  o menor índice dentre todos os vértices atingíveis a partir da subárvore (ou subgrafo) cuja raiz é  $u$
- ★ A DFS a partir de  $u$  gera uma subárvore se não houverem arestas reversas

## Menor ancestral alcançável

- ★ Seja  $\mu_s(u)$  o menor índice dentre todos os vértices atingíveis a partir da subárvore (ou subgrafo) cuja raiz é  $u$
- ★ A DFS a partir de  $u$  gera uma subárvore se não houverem arestas reversas
- ★ Neste caso,  $i_s(w) = \mu_s(w)$  para todo vértice  $w$  nesta subárvore

## Menor ancestral alcançável

- ★ Seja  $\mu_s(u)$  o menor índice dentre todos os vértices atingíveis a partir da subárvore (ou subgrafo) cuja raiz é  $u$
- ★ A DFS a partir de  $u$  gera uma subárvore se não houverem arestas reversas
- ★ Neste caso,  $i_s(w) = \mu_s(w)$  para todo vértice  $w$  nesta subárvore
- ★ As arestas reversas impactam nos valores de  $\mu_s(u)$

## Menor ancestral alcançável

- ★ Seja  $\mu_s(u)$  o menor índice dentre todos os vértices atingíveis a partir da subárvore (ou subgrafo) cuja raiz é  $u$
- ★ A DFS a partir de  $u$  gera uma subárvore se não houverem arestas reversas
- ★ Neste caso,  $i_s(w) = \mu_s(w)$  para todo vértice  $w$  nesta subárvore
- ★ As arestas reversas impactam nos valores de  $\mu_s(u)$
- ★ Se  $(u, v)$  é aresta reversa, então  $\mu_s(u) = \min\{\mu_s(u), i_s(v)\}$

## **Identificação de pontes**

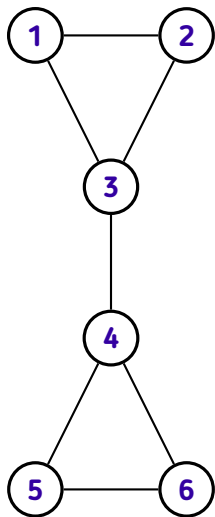
## Identificação de pontes

Seja  $G(V, E)$  um grafo conectado e  $s \in V$  o vértice de partida de uma DFS.

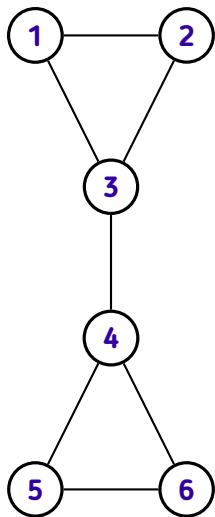
A aresta  $(u, v) \in E$  é uma ponte se  $\mu_s(v) > i_s(u)$ .



## Grafo

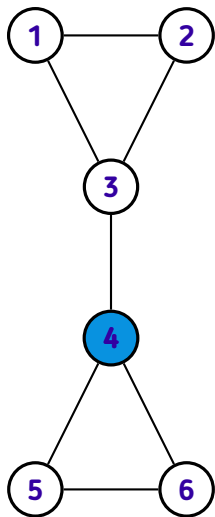


**Grafo**



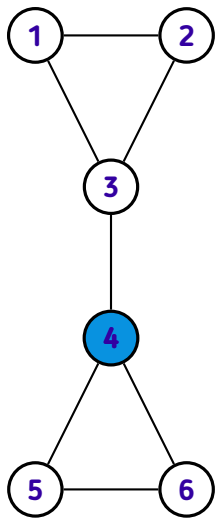
**Árvore gerada pelo DFS**

**Grafo**



**Árvore gerada pelo DFS**

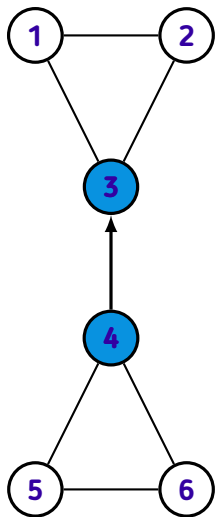
**Grafo**



**Árvore gerada pelo DFS**



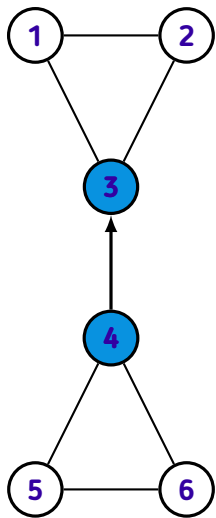
**Grafo**



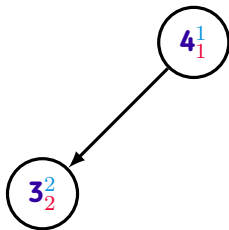
**Árvore gerada pelo DFS**



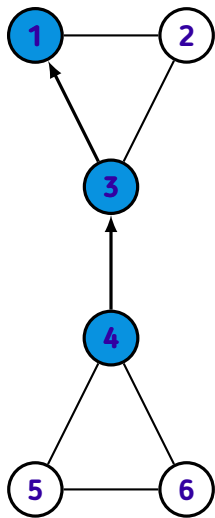
**Grafo**



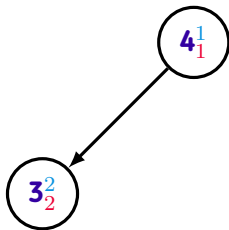
**Árvore gerada pelo DFS**



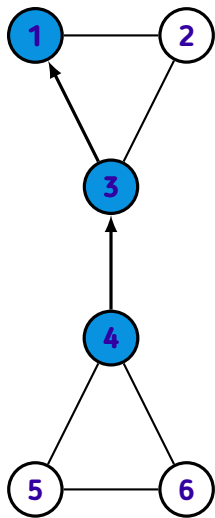
Grafo



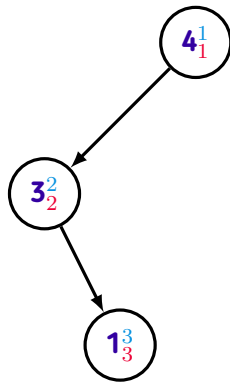
Árvore gerada pelo DFS



Grafo

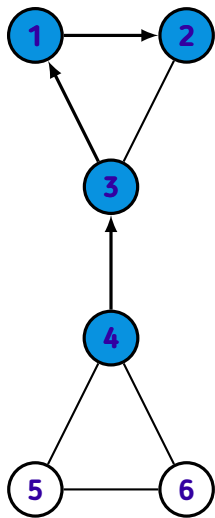


Árvore gerada pelo DFS

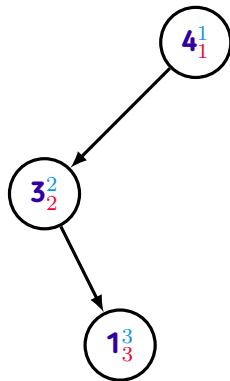




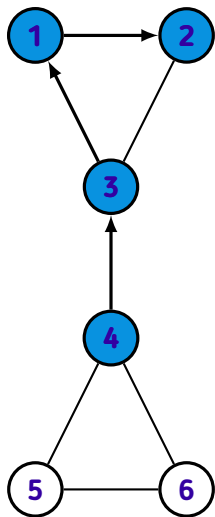
Grafo



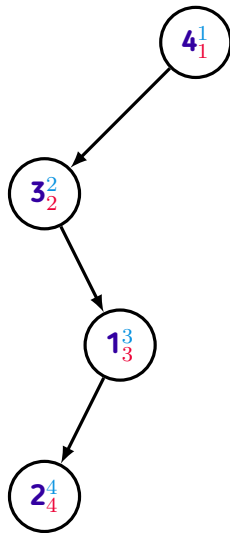
Árvore gerada pelo DFS



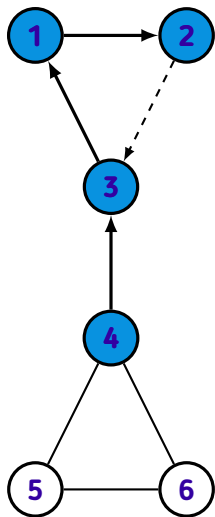
Grafo



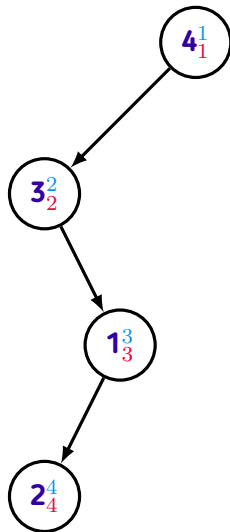
Árvore gerada pelo DFS



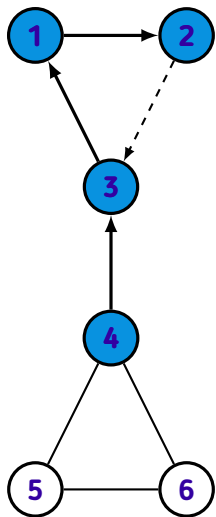
Grafo



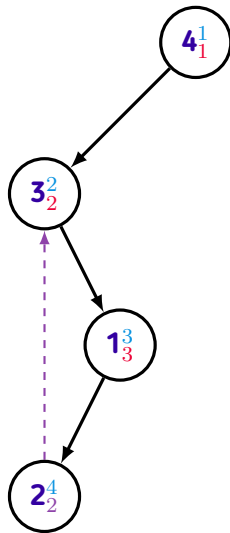
Árvore gerada pelo DFS



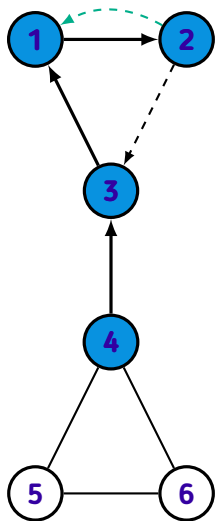
Grafo



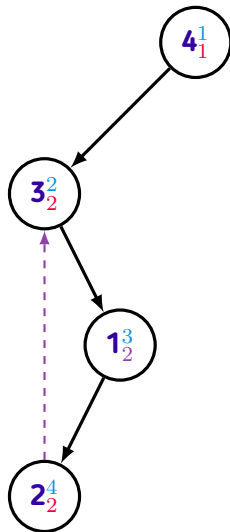
Árvore gerada pelo DFS



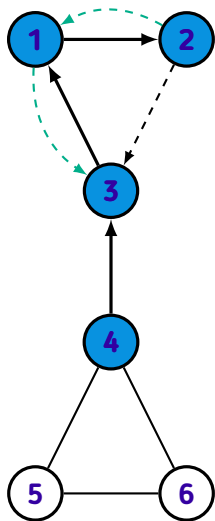
Grafo



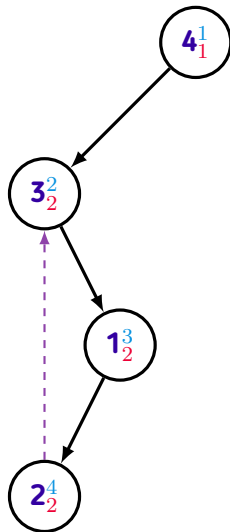
Árvore gerada pelo DFS



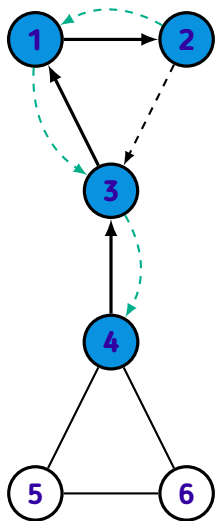
Grafo



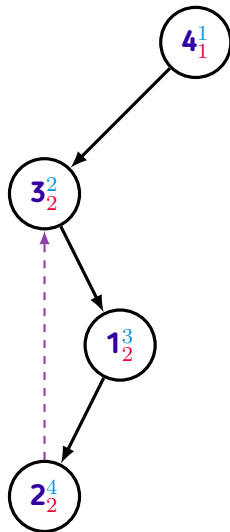
Árvore gerada pelo DFS



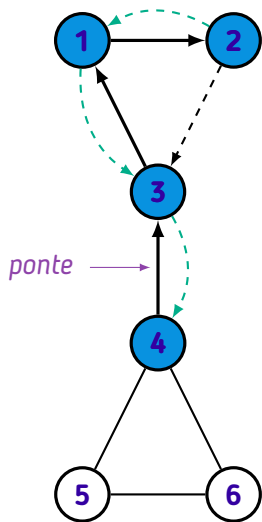
Grafo



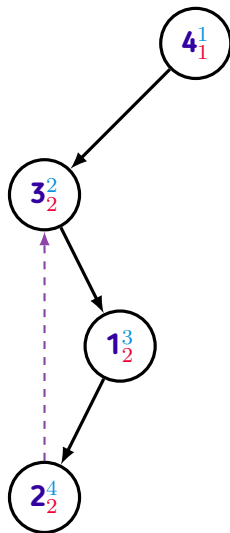
Árvore gerada pelo DFS



Grafo

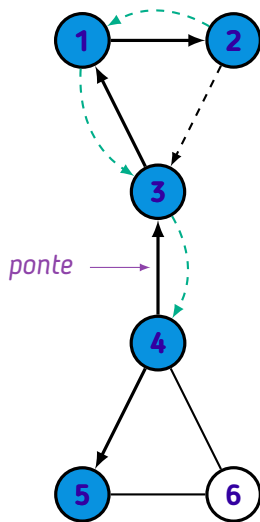


Árvore gerada pelo DFS

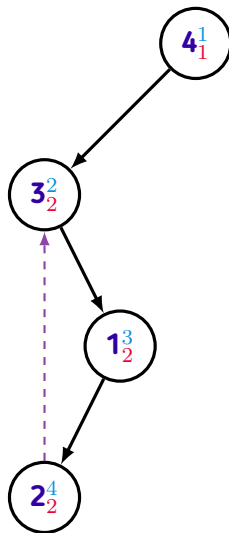




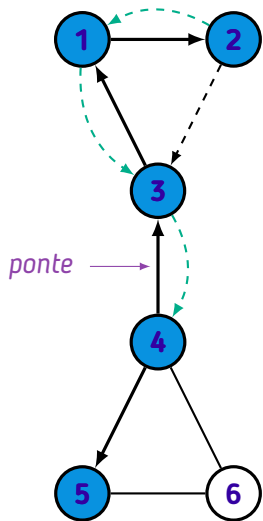
Grafo



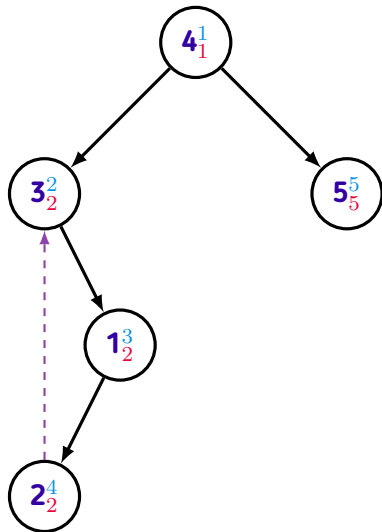
Árvore gerada pelo DFS



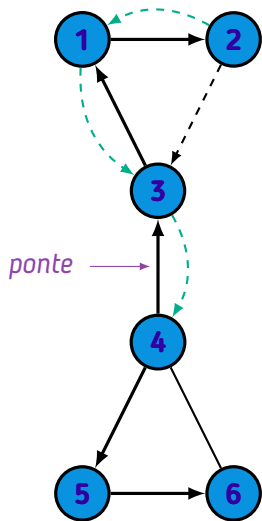
Grafo



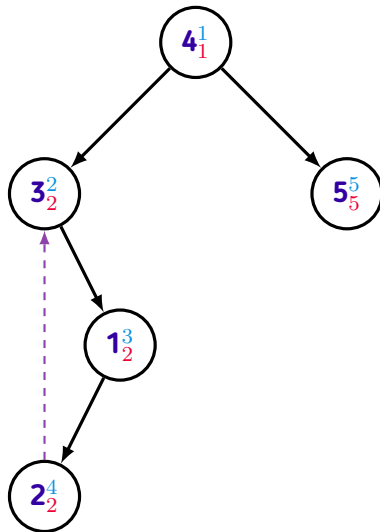
Árvore gerada pelo DFS



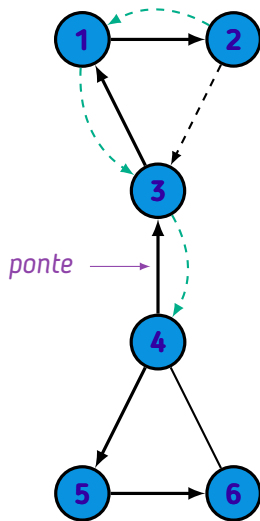
Grafo



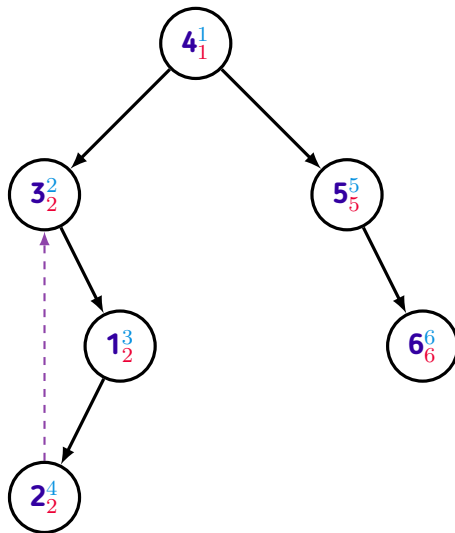
Árvore gerada pelo DFS



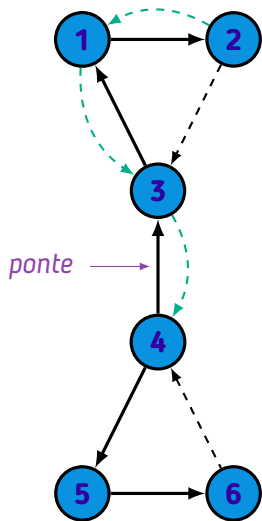
Grafo



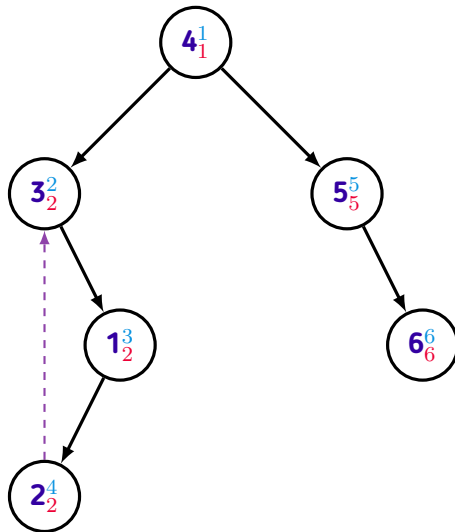
Árvore gerada pelo DFS



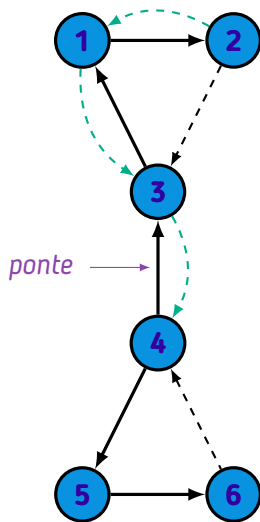
Grafo



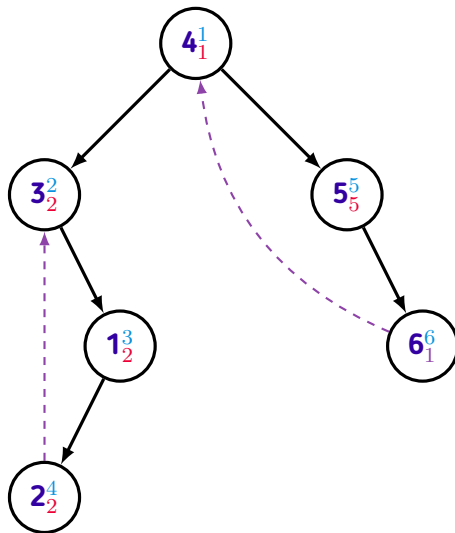
Árvore gerada pelo DFS



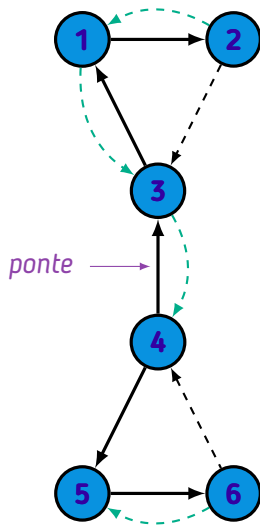
Grafo



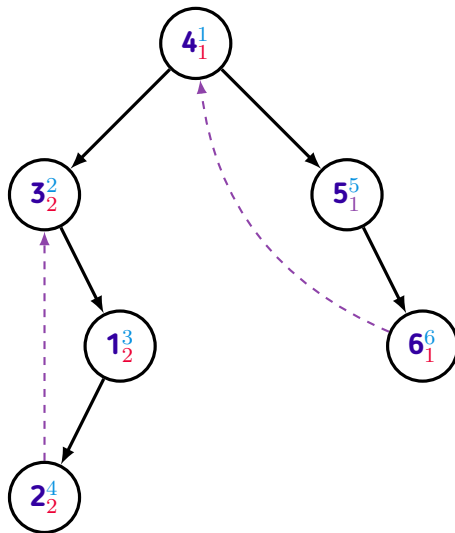
Árvore gerada pelo DFS



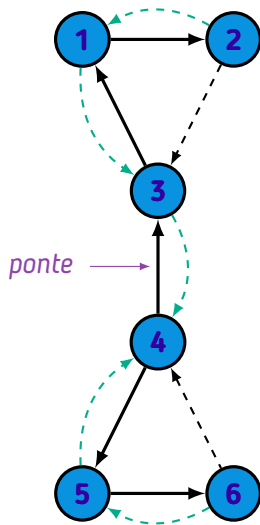
Grafo



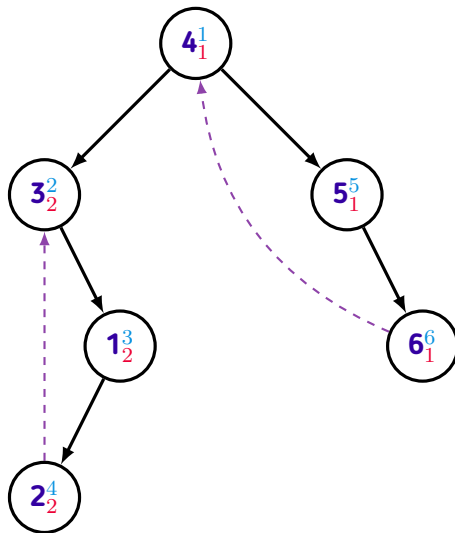
Árvore gerada pelo DFS



Grafo



Árvore gerada pelo DFS





```
void dfs_bridge(int u, int p, int& next, vector<edge>& bridges)
{
    dfs_low[u] = dfs_num[u] = next++;

    for (auto v : adj[u])
        if (not dfs_num[v]) {

            dfs_bridge(v, u, next, bridges);

            if (dfs_low[v] > dfs_num[u])
                bridges.emplace_back(u, v);

            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        } else if (v != p)
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
}
```

```
vector<edge> bridges(int N)
{
    memset(dfs_num, 0, (N + 1)*sizeof(int));
    memset(dfs_low, 0, (N + 1)*sizeof(int));

    vector<edge> bridges;

    for (int u = 1, next = 1; u <= N; ++u)
        if (not dfs_num[u])
            dfs_bridge(u, u, next, bridges);

    return bridges;
}
```

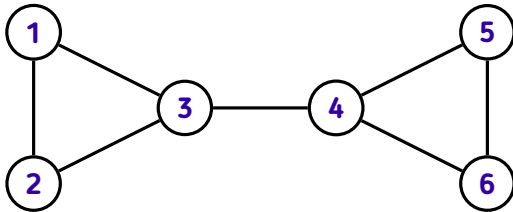
## **Pontos de articulação**

## Pontos de articulação

Seja  $G(V, E)$  um grafo não-direcionado conectado. Um vértice  $u \in V$  é um ponto de articulação se a exclusão de  $u$  e de todas as arestas que incidem em  $u$  torna o grafo desconectado.

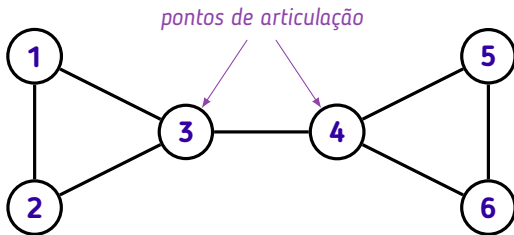
## Pontos de articulação

Seja  $G(V, E)$  um grafo não-direcionado conectado. Um vértice  $u \in V$  é um ponto de articulação se a exclusão de  $u$  e de todas as arestas que incidem em  $u$  torna o grafo desconectado.



## Pontos de articulação

Seja  $G(V, E)$  um grafo não-direcionado conectado. Um vértice  $u \in V$  é um ponto de articulação se a exclusão de  $u$  e de todas as arestas que incidem em  $u$  torna o grafo desconectado.



## **Identificação de pontos de articulação**

## Identificação de pontos de articulação

Seja  $G(V, E)$  um grafo conectado e  $s \in V$  o vértice de partida de uma DFS.

A aresta  $(v, u) \in E$  identifica o ponto de articulação  $u$  se  $\mu_s(v) \geq i_s(u)$ .



## Identificação de pontos de articulação

Seja  $G(V, E)$  um grafo conectado e  $s \in V$  o vértice de partida de uma DFS.

A aresta  $(v, u) \in E$  identifica o ponto de articulação  $u$  se  $\mu_s(v) \geq i_s(u)$ .

Caso especial:  $s$  só é ponto de articulação se ele tem, no mínimo, dois filhos.

## Identificação de pontos de articulação

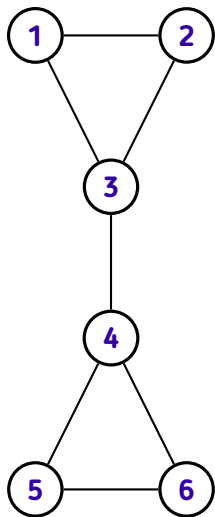
Seja  $G(V, E)$  um grafo conectado e  $s \in V$  o vértice de partida de uma DFS.

A aresta  $(v, u) \in E$  identifica o ponto de articulação  $u$  se  $\mu_s(v) \geq i_s(u)$ .

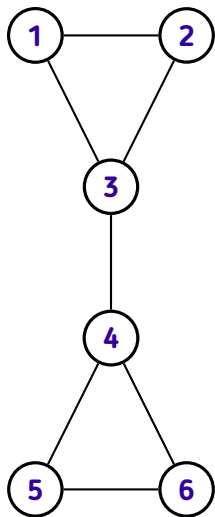
Caso especial:  $s$  só é ponto de articulação se ele tem, no mínimo, dois filhos.

Definição: Se  $G$  não tem pontos de articulação ele é denominado biconectado.

## Grafo

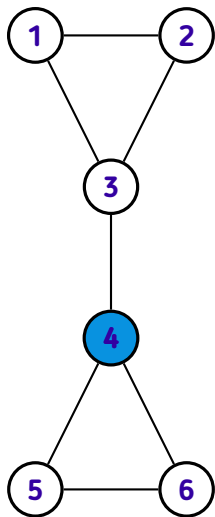


**Grafo**



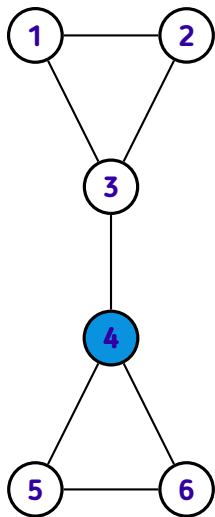
**Árvore gerada pelo DFS**

**Grafo**



**Árvore gerada pelo DFS**

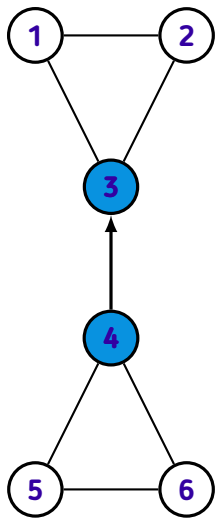
**Grafo**



**Árvore gerada pelo DFS**



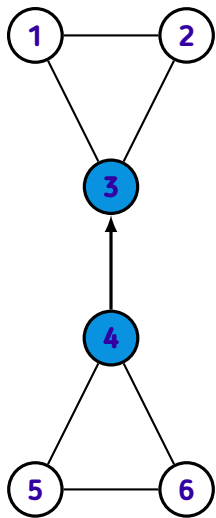
**Grafo**



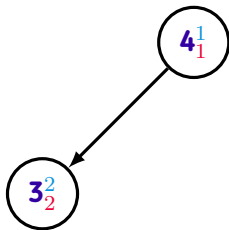
**Árvore gerada pelo DFS**



Grafo

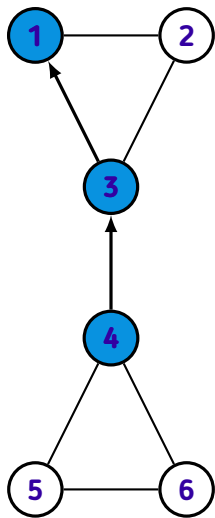


Árvore gerada pelo DFS

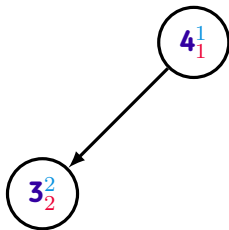




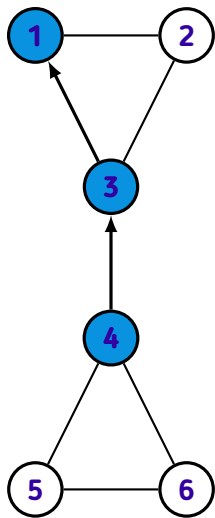
**Grafo**



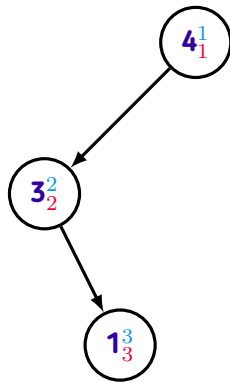
**Árvore gerada pelo DFS**



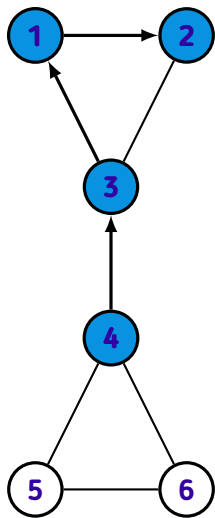
Grafo



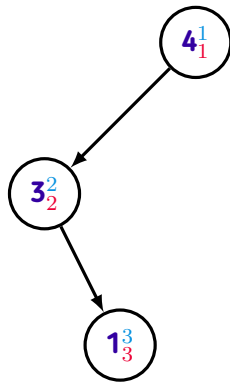
Árvore gerada pelo DFS



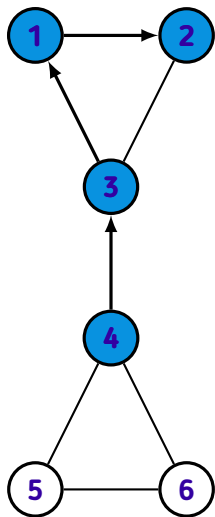
**Grafo**



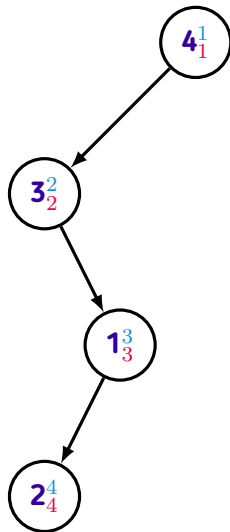
**Árvore gerada pelo DFS**



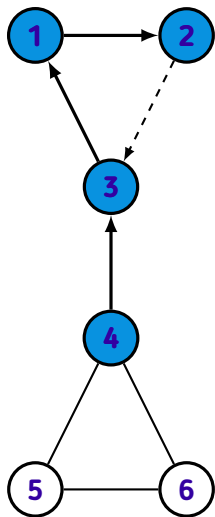
Grafo



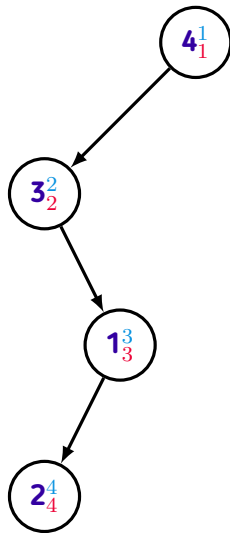
Árvore gerada pelo DFS



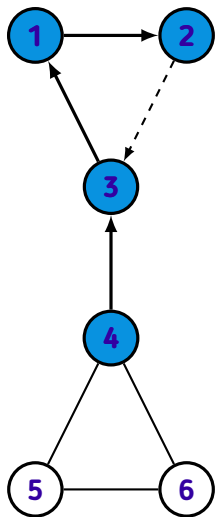
Grafo



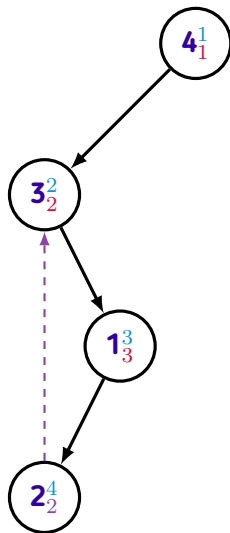
Árvore gerada pelo DFS



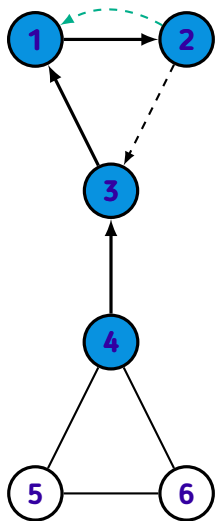
Grafo



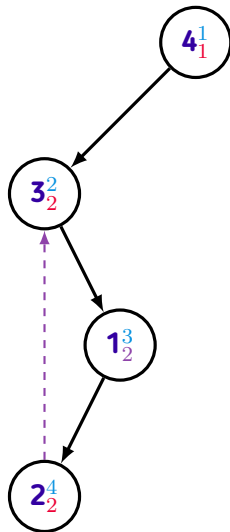
Árvore gerada pelo DFS



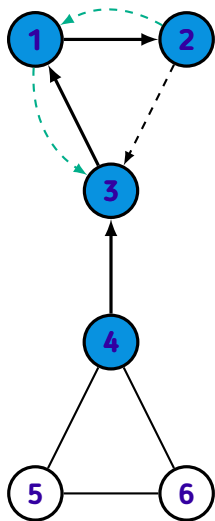
Grafo



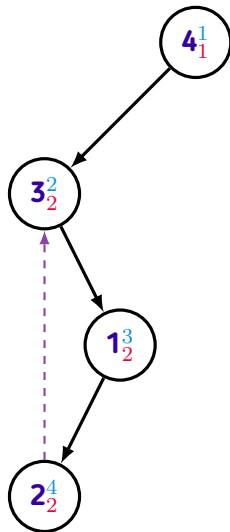
Árvore gerada pelo DFS



Grafo

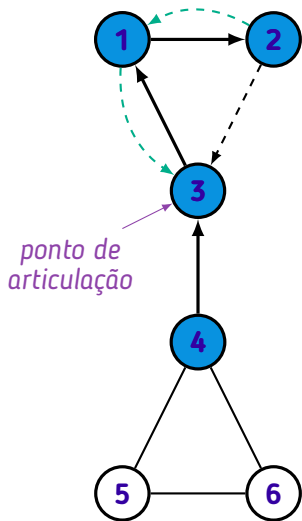


Árvore gerada pelo DFS

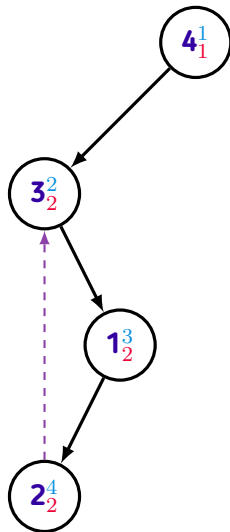




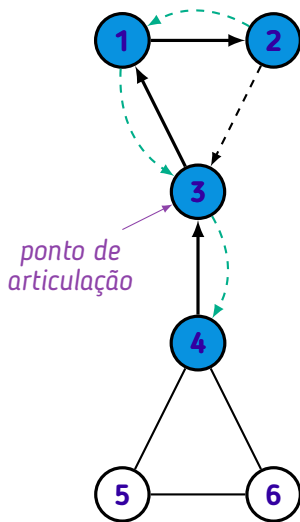
Grafo



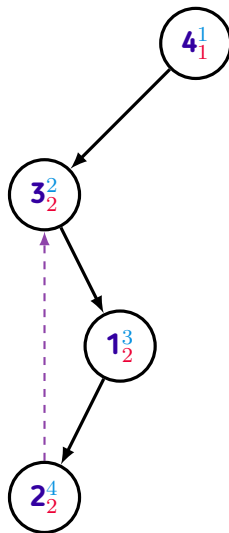
Árvore gerada pelo DFS



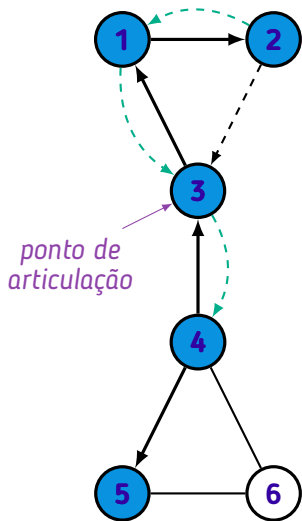
Grafo



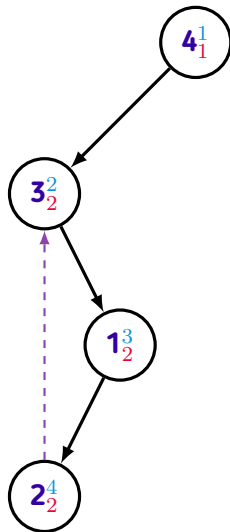
Árvore gerada pelo DFS



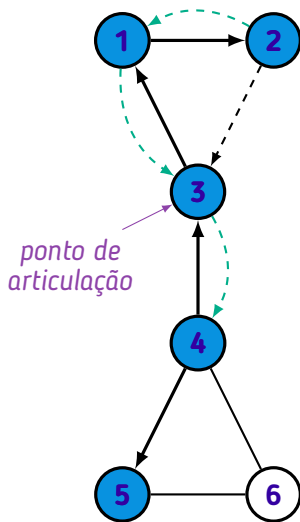
Grafo



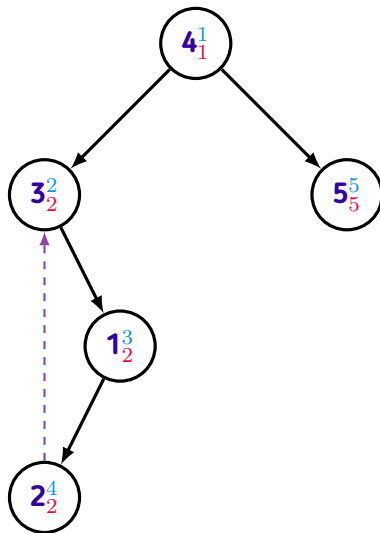
Árvore gerada pelo DFS



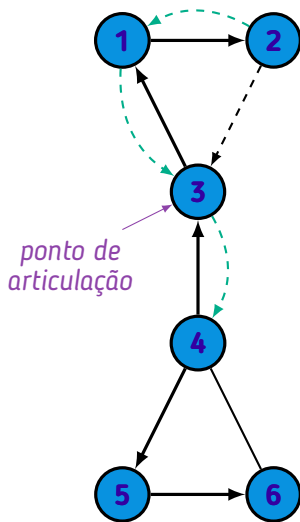
Grafo



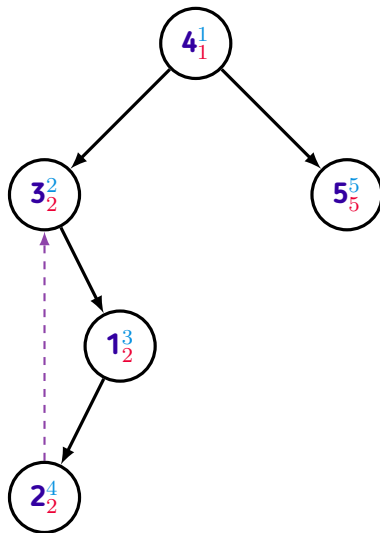
Árvore gerada pelo DFS



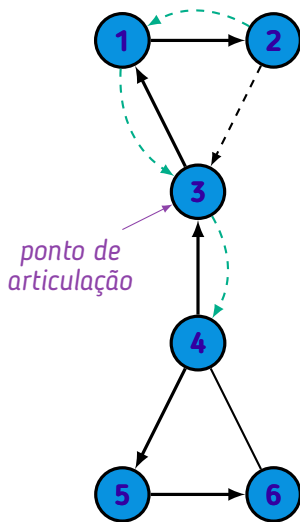
Grafo



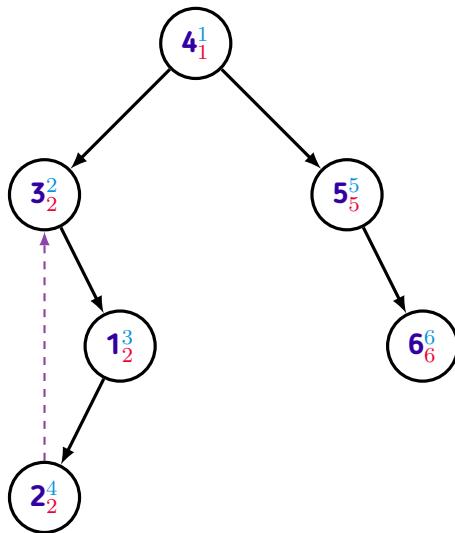
Árvore gerada pelo DFS



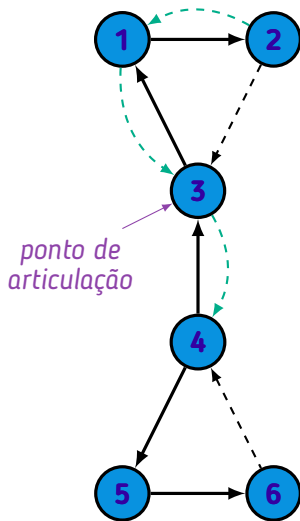
Grafo



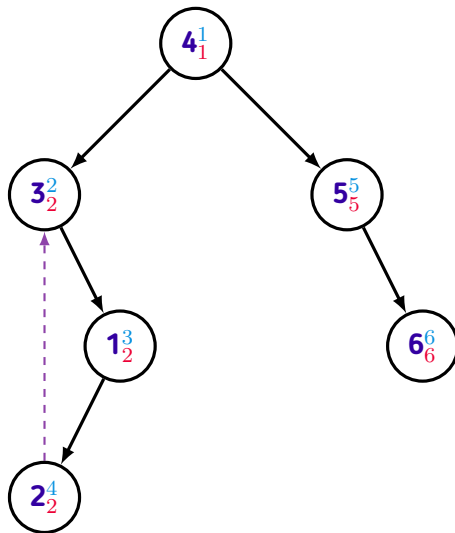
Árvore gerada pelo DFS



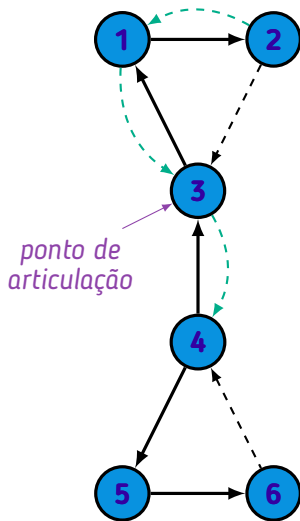
Grafo



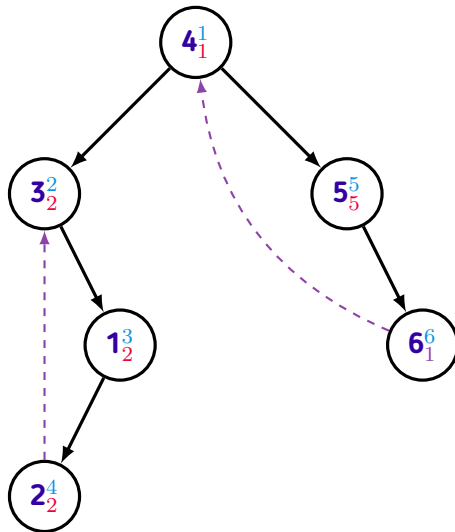
Árvore gerada pelo DFS



Grafo

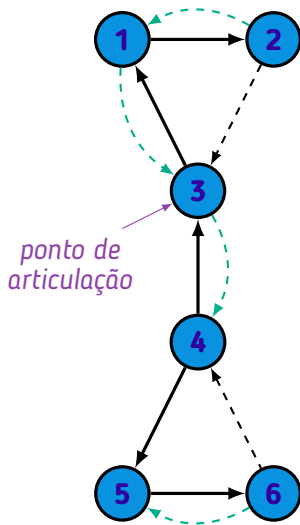


Árvore gerada pelo DFS

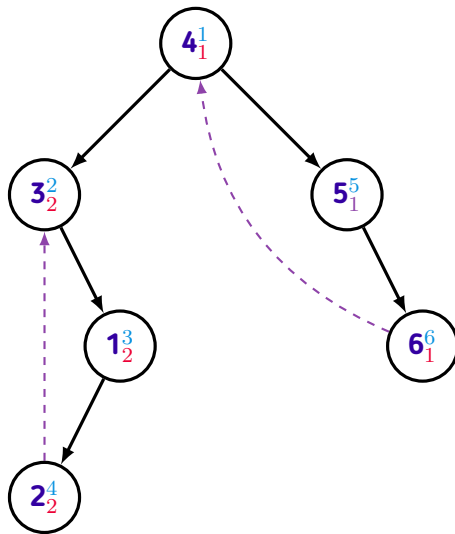




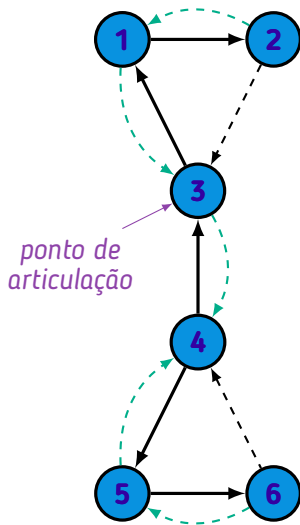
Grafo



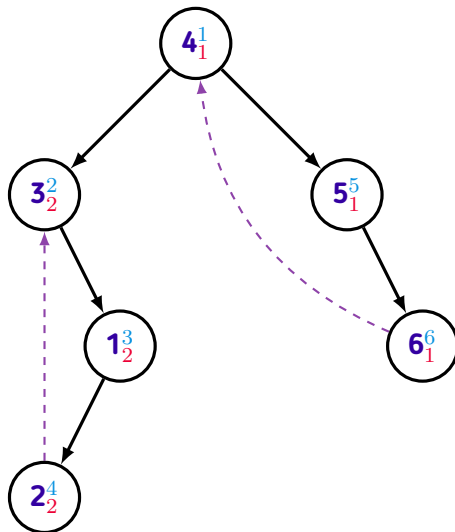
Árvore gerada pelo DFS



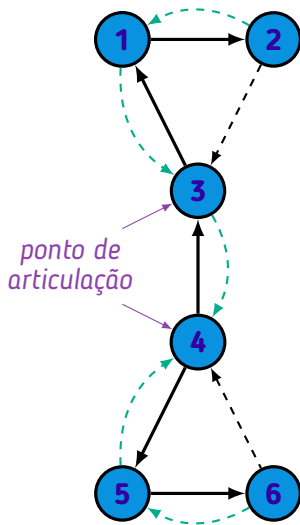
Grafo



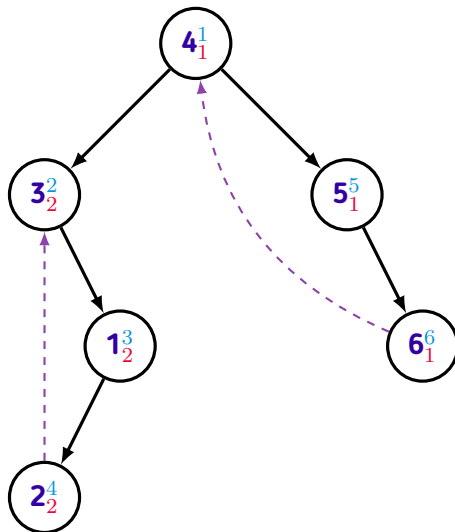
Árvore gerada pelo DFS



**Grafo**



**Árvore gerada pelo DFS**



```
int dfs_articulation_points(int u, int p, int& next, set<int>& points)
{
    int children = 0;
    dfs_low[u] = dfs_num[u] = next++;

    for (auto v : adj[u])
        if (not dfs_num[v]) {
            ++children;

            dfs_articulation_points(v, u, next, points);

            if (dfs_low[v] >= dfs_num[u])
                points.insert(u);

            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        } else if (v != p)
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);

    return children;
}
```

```
set<int> articulation_points(int N)
{
    memset(dfs_num, 0, (N + 1)*sizeof(int));
    memset(dfs_low, 0, (N + 1)*sizeof(int));

    set<int> points;

    for (int u = 1, next = 1; u <= N; ++u)
        if (not dfs_num[u])
        {
            auto children = dfs_articulation_points(u, u, next, points);

            if (children == 1)
                points.erase(u);
        }

    return points;
}
```

## Problemas sugeridos

1. [AtCoder Beginner Contest 075 – Problem C: Bridge](#)
2. [OJ 315 – Network](#)
3. [OJ 610 – Street Directions](#)
4. [SPOJ SUBMERGE – Submerging Islands](#)

## Referências

1. HALIM, Felix; HALIM, Steve. *Competitive Programming 3*, 2010.
2. LAAKSONEN, Antti. *Competitive Programmer's Handbook*, 2018.
3. SKIENA, Steven; REVILLA, Miguel. *Programming Challenges*, 2003.