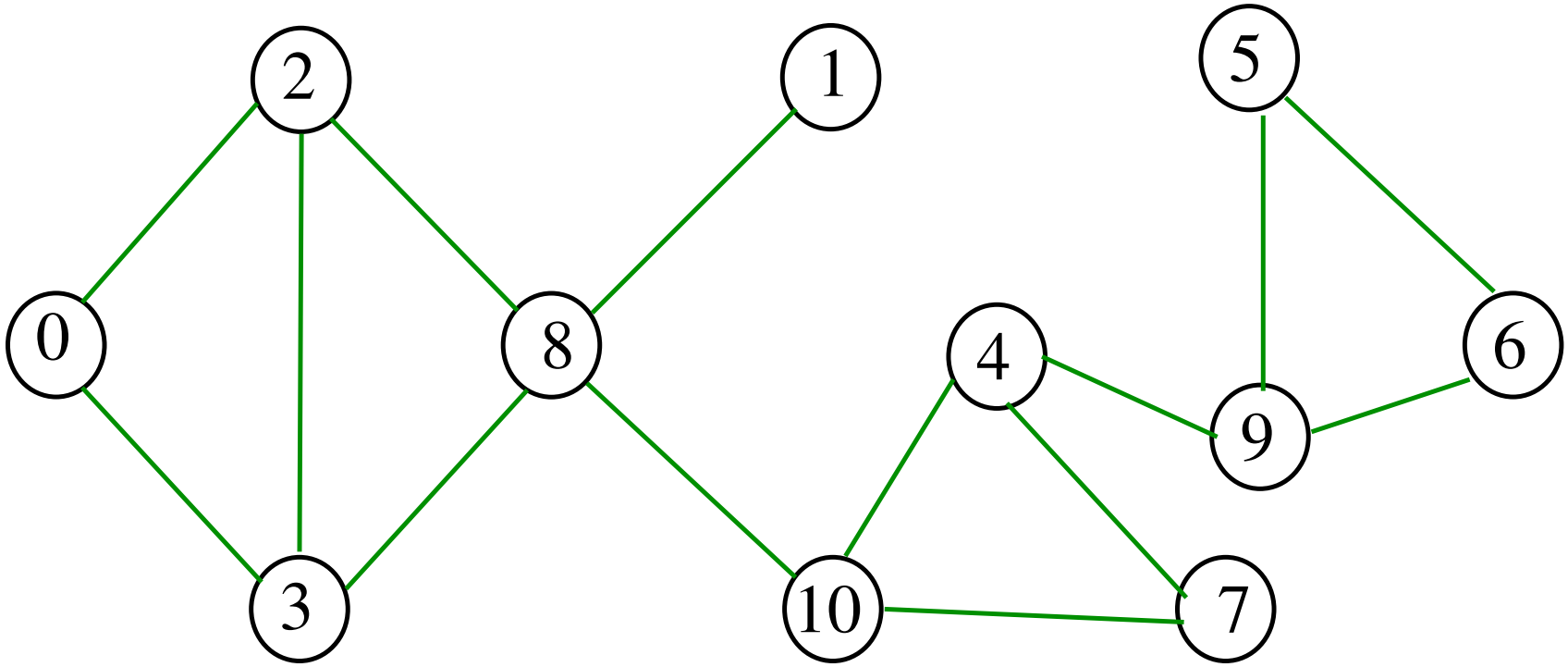
Pontes em grafos e aresta-biconexão

S 18.6

Pontes em grafos

Uma aresta de um grafo é uma **ponte** (= *bridge* = *separation edge*) se ela é a única aresta que atravessa algum corte do grafo.

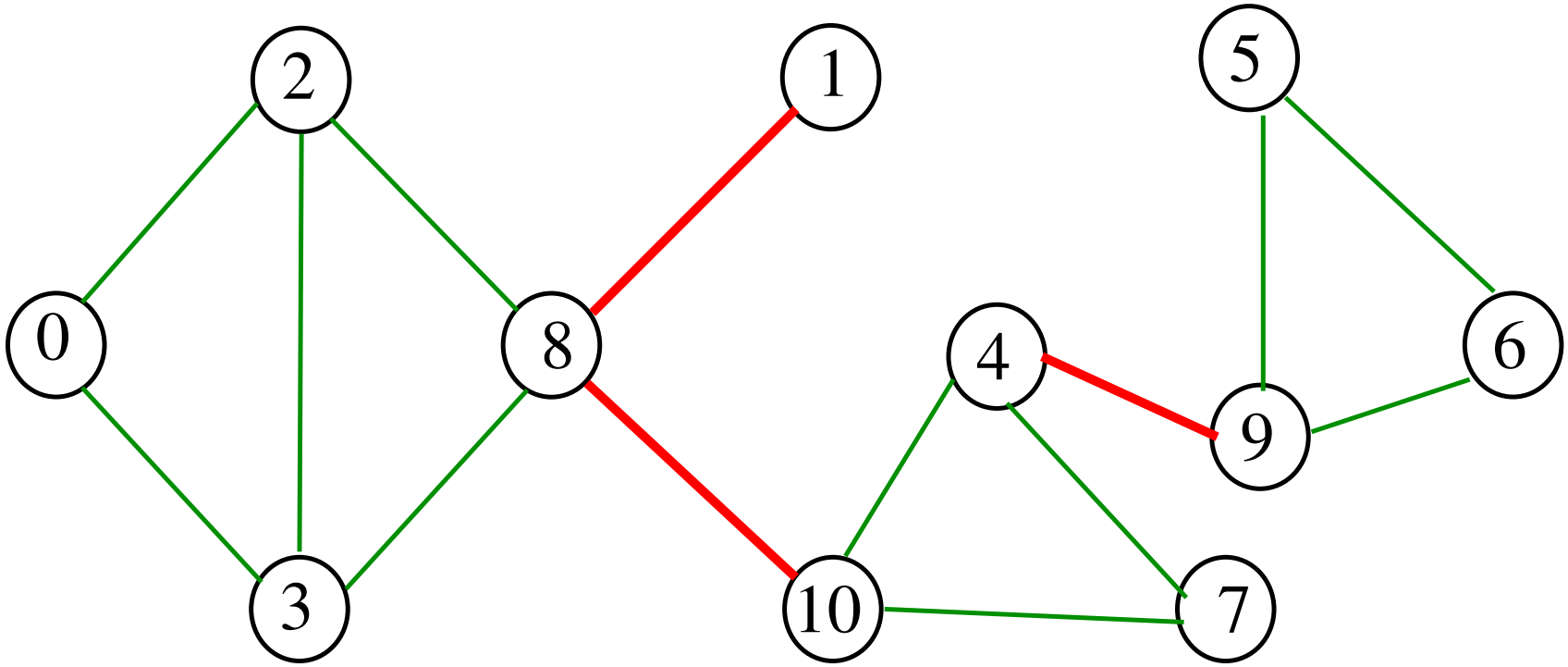
Exemplo:



Pontes em grafos

Uma aresta de um grafo é uma **ponte** (= *bridge* = *separation edge*) se ela é a única aresta que atravessa algum corte do grafo.

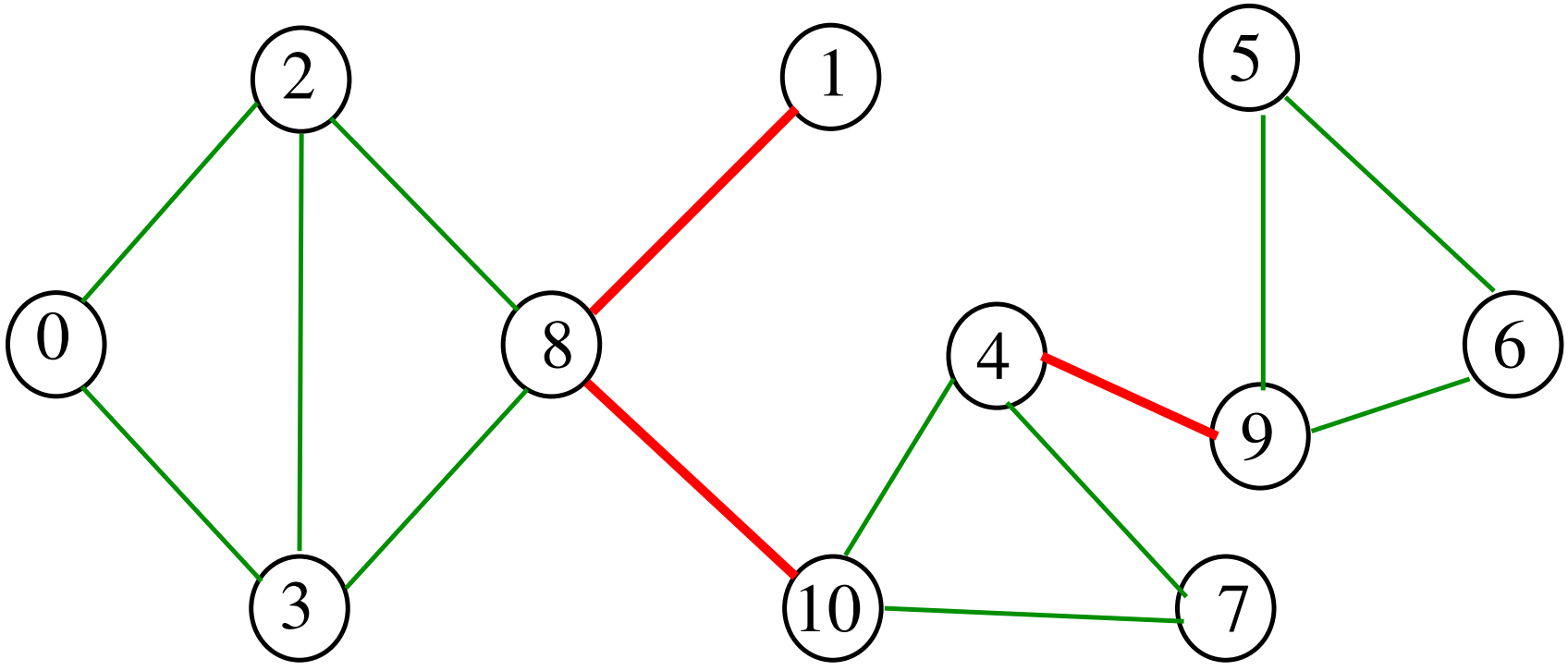
Exemplo: as arestas em vermelho são pontes



Procurando pontes

Problema: encontrar as pontes de um grafo dado

Exemplo: as arestas em vermelho são pontes



all_bridges1

Recebe um grafo G e calcula o número bcnt de pontes do grafo G e imprime todas as pontes.

void all_bridges1 (Graph G);

Primeiro algoritmo

```
void all_bridges1 (Graph G) {
    Vertex v, w; link p; int ligados;
 1   for (v = 0; v < G->V; v++)
 2       for(p=G->adj[v];p!=NULL;p=p->next){
 3           w = p->w;
 4           if (v < w) {
 5               GRAPHremoveA(G,w,v);
 6               ligados = DIGRAPHpath(G,w,v);
 7               GRAPHinsertA(G,w,v);
 8               if (!ligados) {
 9                   bcnt++;
10                   output(v, w);
                 }
             }
         }
 }
```

Consumo de tempo

O consumo de tempo da função `all_bridges1` é $A/2$ vezes o consumo de tempo da função `DIGRAPHpath`.

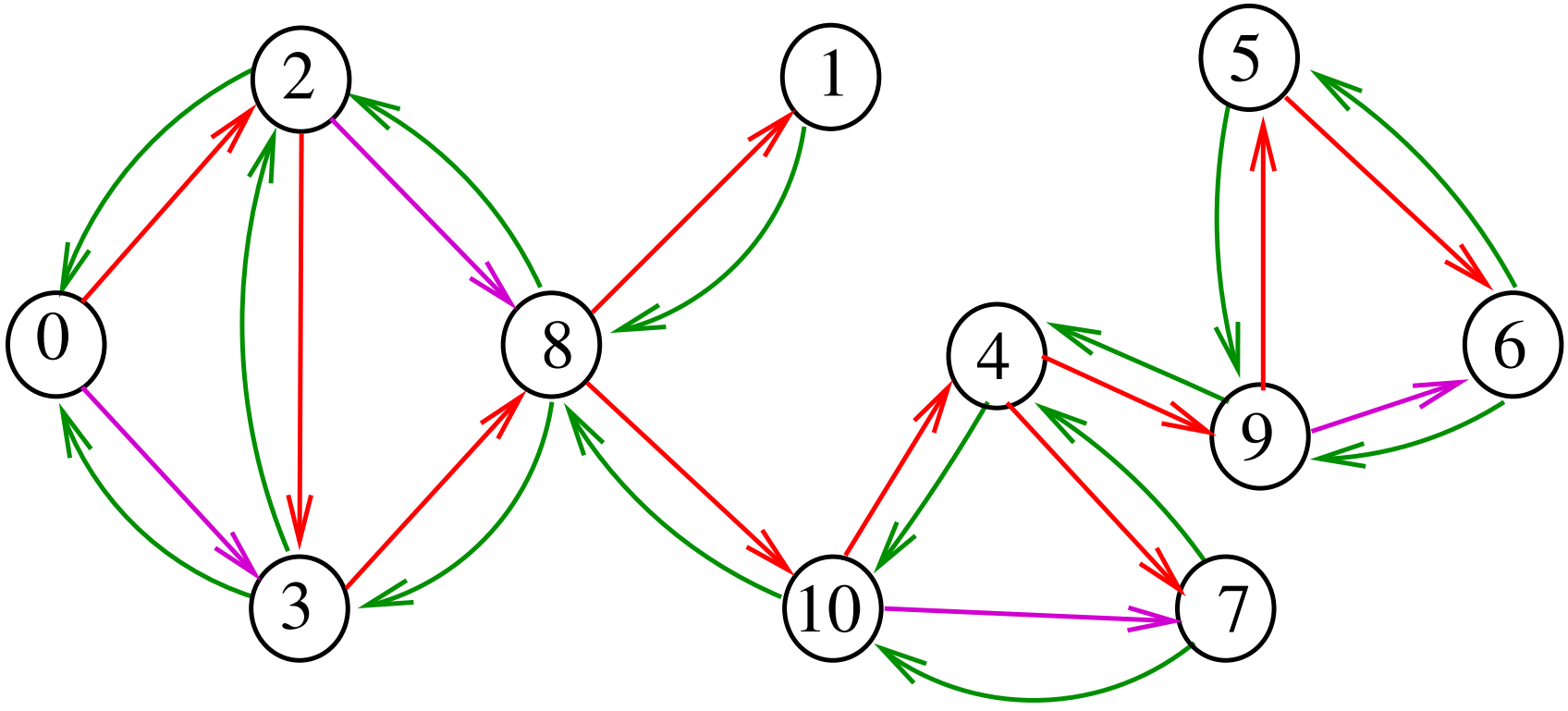
O consumo de tempo da função `all_bridges1` para vetor de listas de adjacência é $O(A(V + A))$.

O consumo de tempo da função `all_bridges1` para matriz de adjacência é $O(AV^2)$.

Pontes e busca em profundidade

Em uma floresta DFS, um dos dois arcos de cada ponte será um arco da arborescência

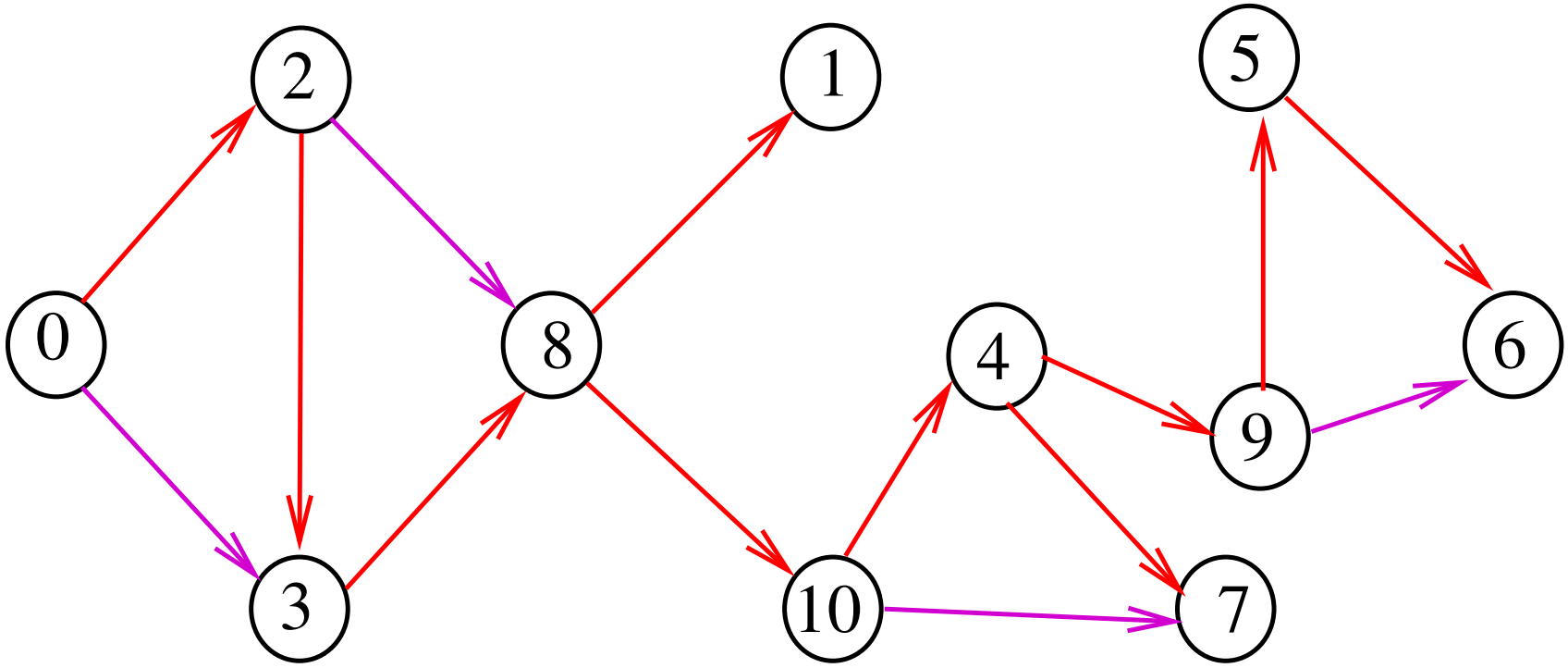
Exemplo: arcos em vermelho são da arborescência



Pontes e busca em profundidade

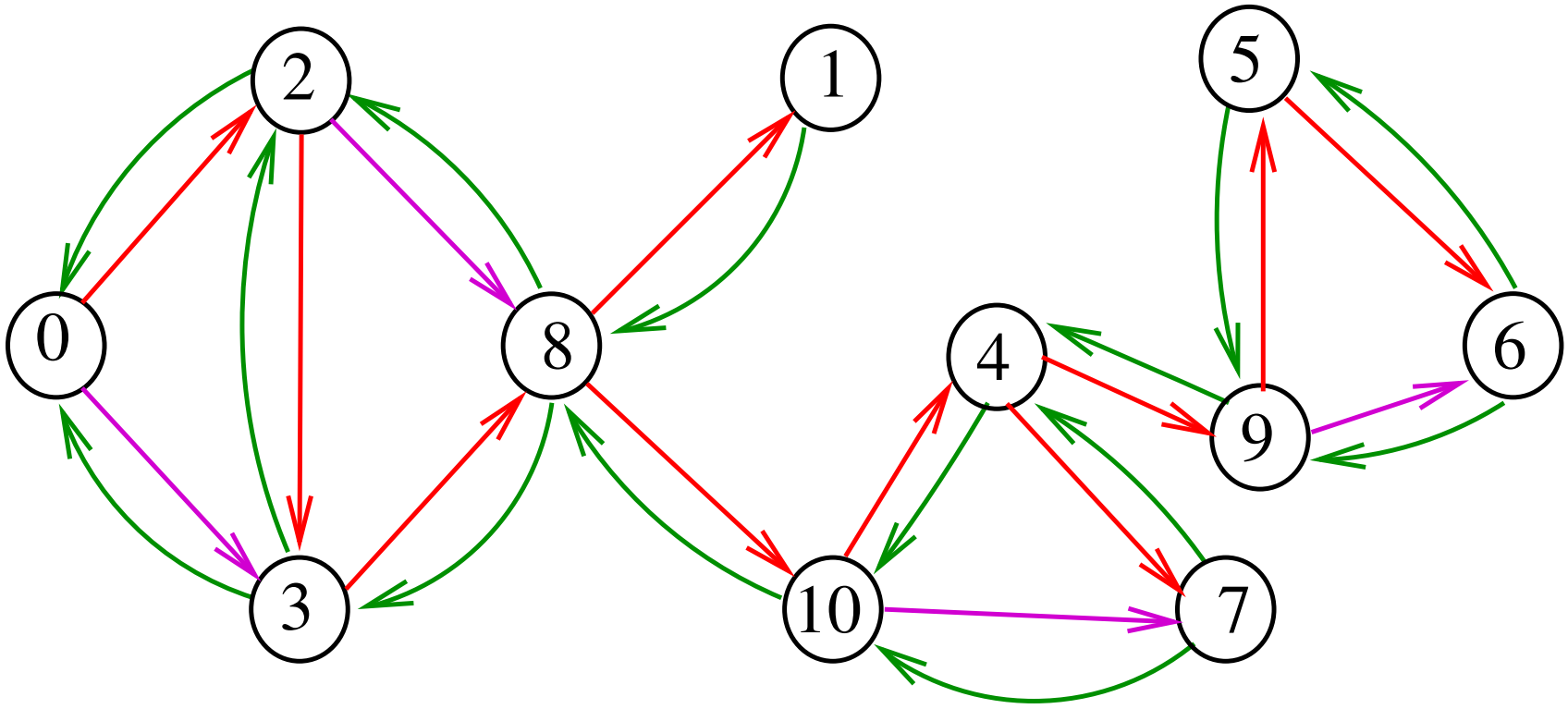
Em uma floresta DFS, um dos dois arcos de cada ponte será um arco da arborescência

Exemplo: arcos em vermelho são da arborescência



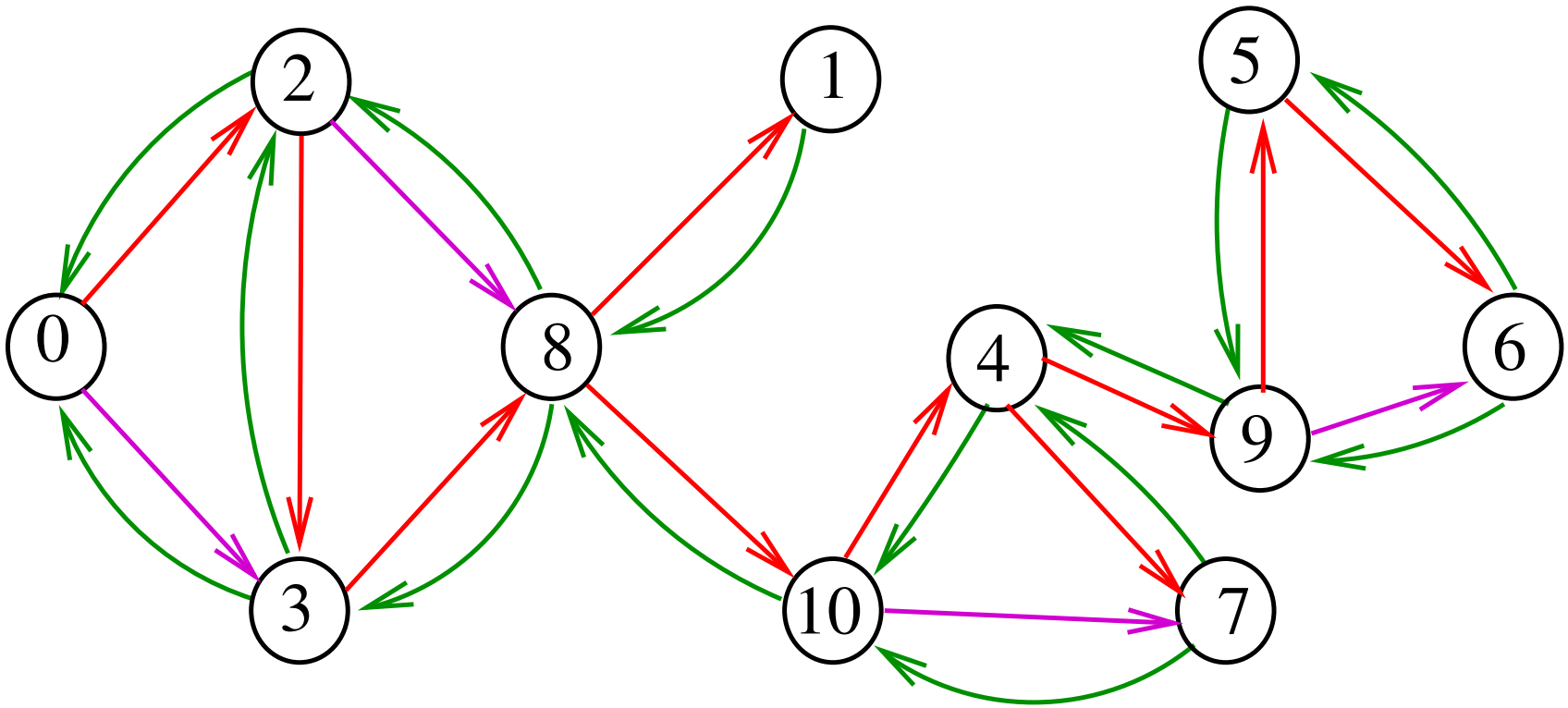
Propriedade

Um arco v-w da floresta DFS faz parte (juntamente com w-v) de uma ponte se e somente se não existe arco de retorno que ligue um descendente de w a um ancestral de v



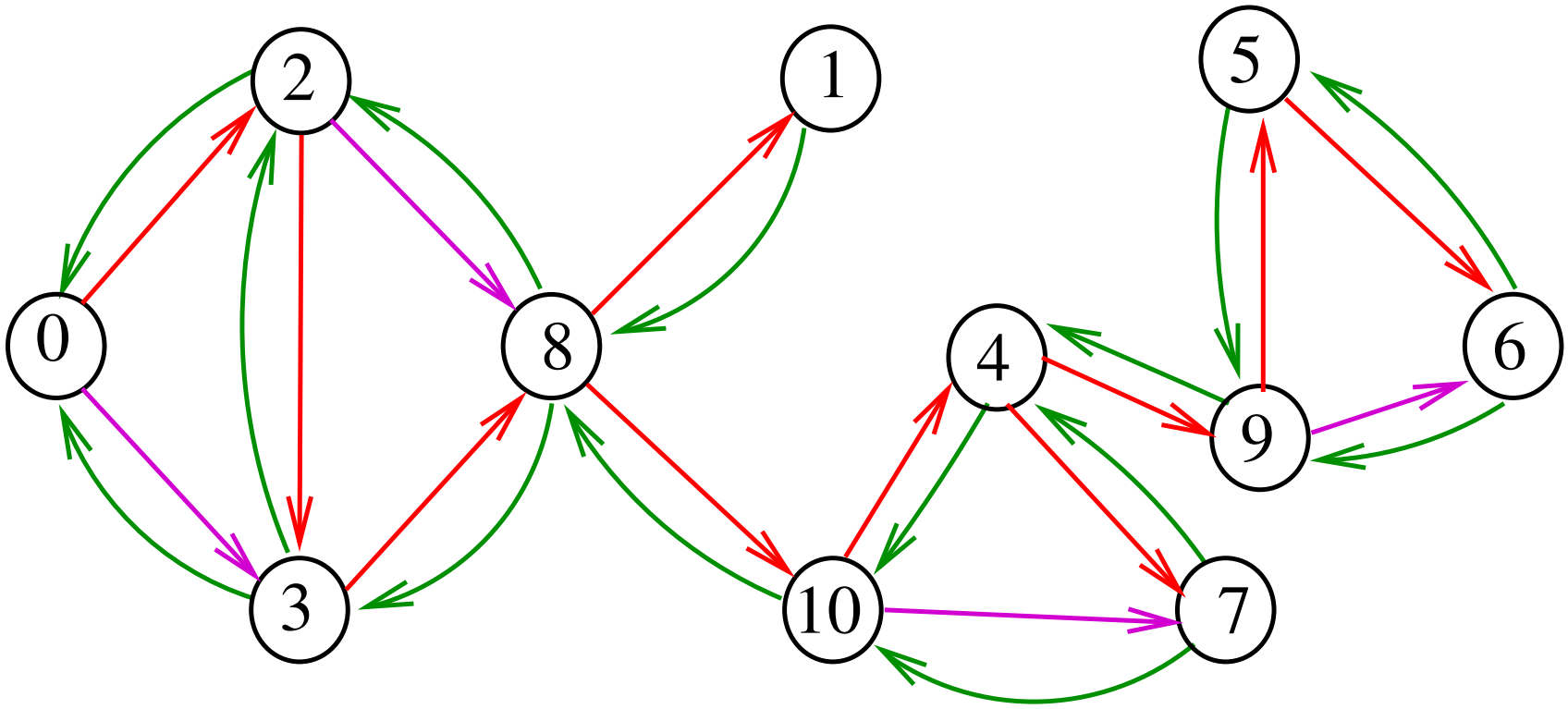
Numeração em ordem de descoberta (pré-ordem)

v	0	1	2	3	4	5	6	7	8	9	10
pre[v]	0	**4**	1	2	6	8	9	10	3	**7**	**5**



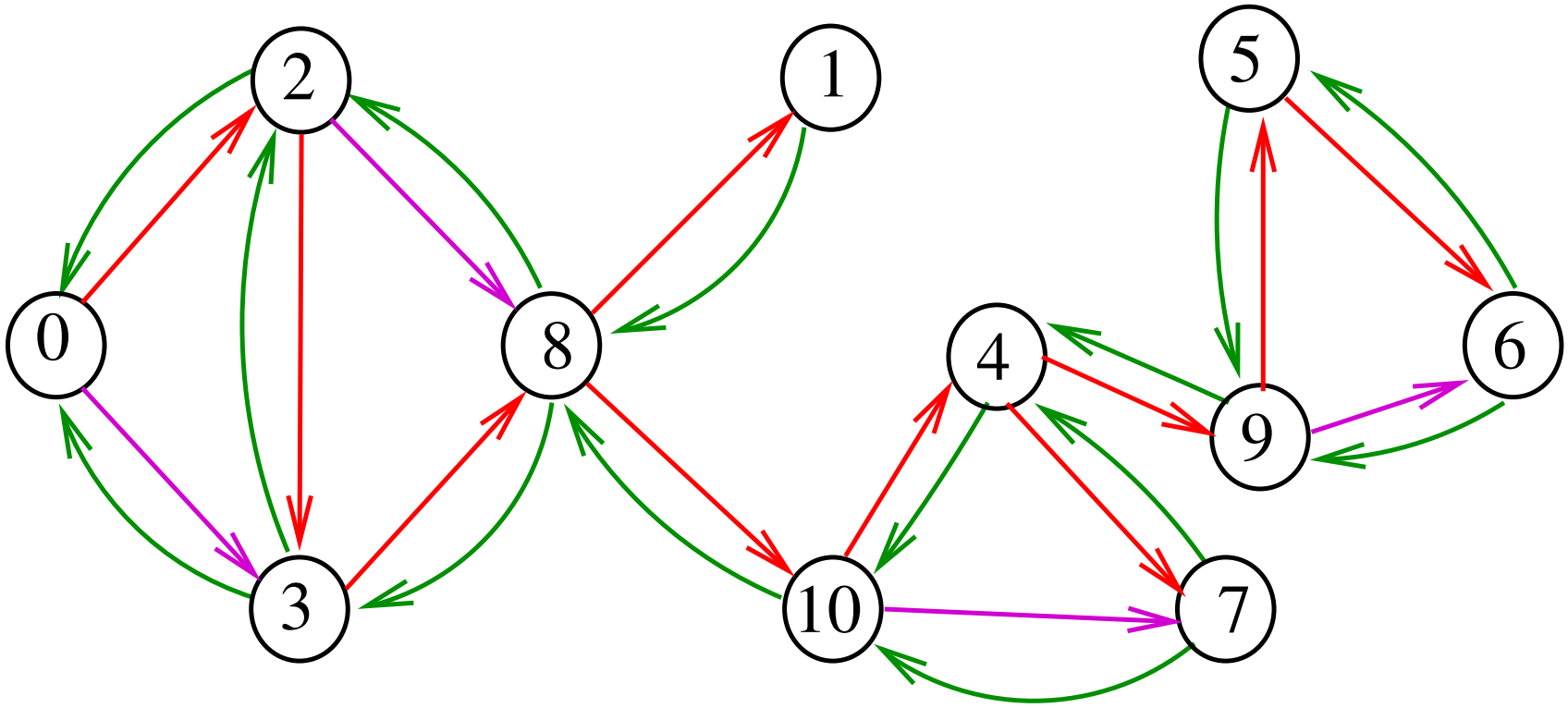
Lowest Preorder Number

O menor **número de pré-ordem** que pode ser alcançado por `v` utilizando arcos da arborescência e **até um** arco de retorno ("arco-pai" não vale) será denotado por `low[v]`



Exemplo

v	0	1	2	3	4	5	6	7	8	9	10
pre[v]	0	**4**	1	2	6	8	9	10	3	**7**	**5**
low[v]	0	**4**	0	0	5	7	7	5	1	**7**	**5**



Observações

Para todo vértice v,

$$\texttt{low}[\texttt{v}] \leq \texttt{pre}[\texttt{v}]$$

Para todo arco v-w do grafo

- se v-w é um arco de **arborescência** então

$$\texttt{low}[\texttt{v}] \leq \texttt{low}[\texttt{w}];$$

- se v-w é uma arco de **retorno**, então

$$\texttt{low}[\texttt{v}] \leq \texttt{pre}[\texttt{w}].$$

Observações

Para todo vértice v,

$$\texttt{low}[\texttt{v}] \leq \texttt{pre}[\texttt{v}]$$

Para todo arco v-w do grafo

- se v-w é um arco de **arborescência** então

$$\texttt{low}[\texttt{v}] \leq \texttt{low}[\texttt{w}];$$

- se v-w é uma arco de **retorno**, então

$$\texttt{low}[\texttt{v}] \leq \texttt{pre}[\texttt{w}].$$

Cálculo de low[v]

Para todo vértice v,

$\texttt{low}[v] = \min\{\texttt{pre}[w] |$ v-w é um arco de **retorno** **ou**
v-w é um arco de **arborescência**$\}$

Algoritmo das pontes

Em qualquer floresta de busca em profundidade de um grafo, um arco de arborescência v-w faz parte de uma ponte se e somente se low[w] == pre[w]

```
static int cnt, pre[maxV], bcnt, low[maxV];
static int parnt[maxV];
```

A função abaixo calcula o número bcnt de pontes do grafo G e imprime todas as pontes

```
void all_bridges (Graph G);
```

all_bridges

```
    void all_bridges (Graph G) {
    Vertex v;
1   cnt = bcnt = 0;
2   for (v = 0; v < G->V; v++)
3       pre[v] = -1;
4   for (v = 0; v < G->V; v++)
5       if (pre[v] == -1) {
6           parnt[v] = v;
7           bridgeR(G, v);
        }
    }
```

bridgeR

```
    void bridgeR (Graph G, Vertex v) {
    link p; Vertex w;
 1   pre[v] = cnt++;
 2   low[v] = pre[v];
 3   for (p=G->adj[v];p!=NULL;p=p->next)
 4       if (pre[w=p->w] == -1) {
 5           parnt[w] = v;
 6           bridgeR(G, w);
 7           if (low[v] > low[w]) low[v]=low[w];
 9           if (low[w] == pre[w]) {
10               bcnt++;
11               printf("%d-%d\n", v, w);
             }
         }
12       else if (w!=parnt[v] && low[v]>pre[w])
13           low[v] = pre[w];
     }
```

Consumo de tempo

O consumo de tempo da função `all_bridges` é $O(V + A)$.

Aresta-biconexão

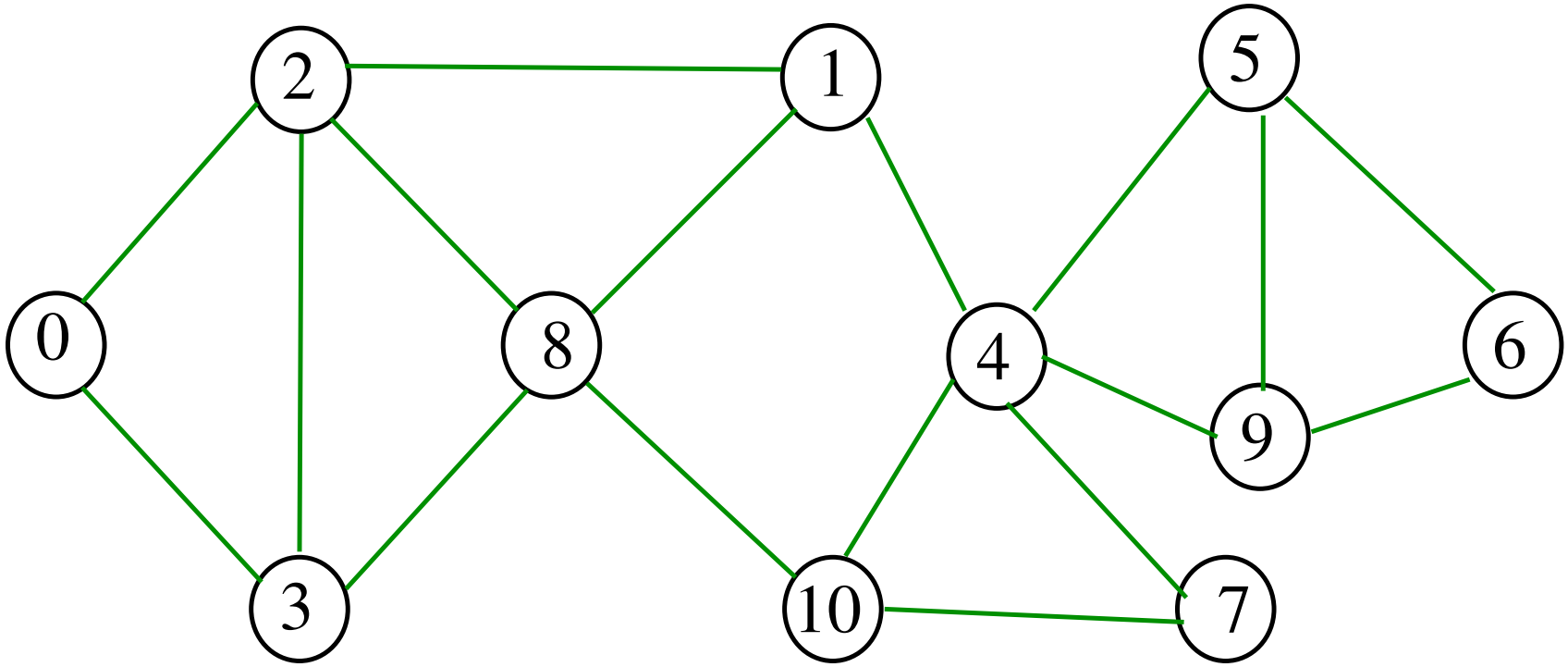
Um grafo é **aresta-biconexo** (= 2-edge-connected) ou **2-aresta-conexo** se for conexo e não tiver pontes.

Fato básico importante:

> Um grafo é aresta-biconexo se e somente se, para cada par (s,t) de seus vértices, existem (pelo menos) dois caminhos de s a t sem arestas em comum.

Exemplo

É preciso remover pelo menos duas arestas de um grafo aresta-biconexo para que ele deixe de ser conexo



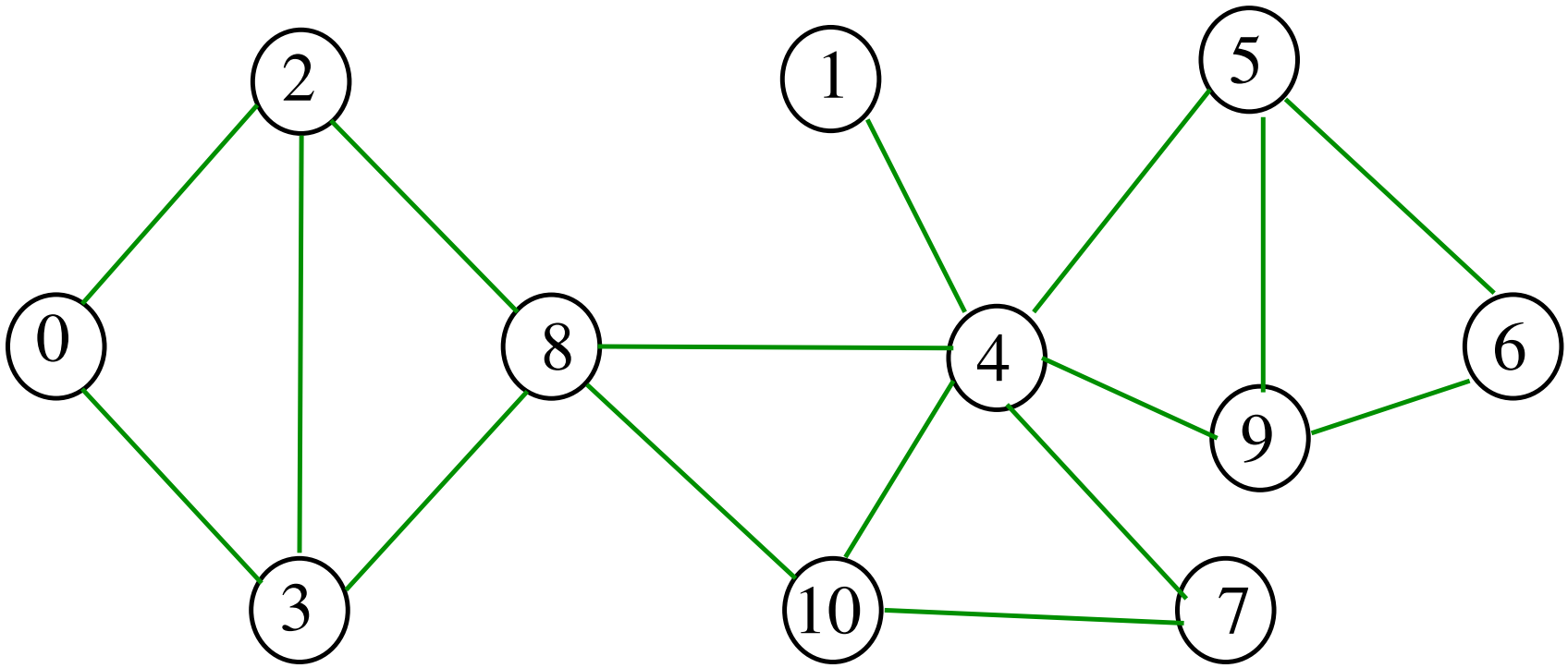
Articulações e biconexão

S 18.6

Articulações em grafos

Uma **articulação** (= *articulation point*) ou **vértice de corte** (= *cut vertex*) de um grafo é um vértice cuja remoção aumenta o número de componentes

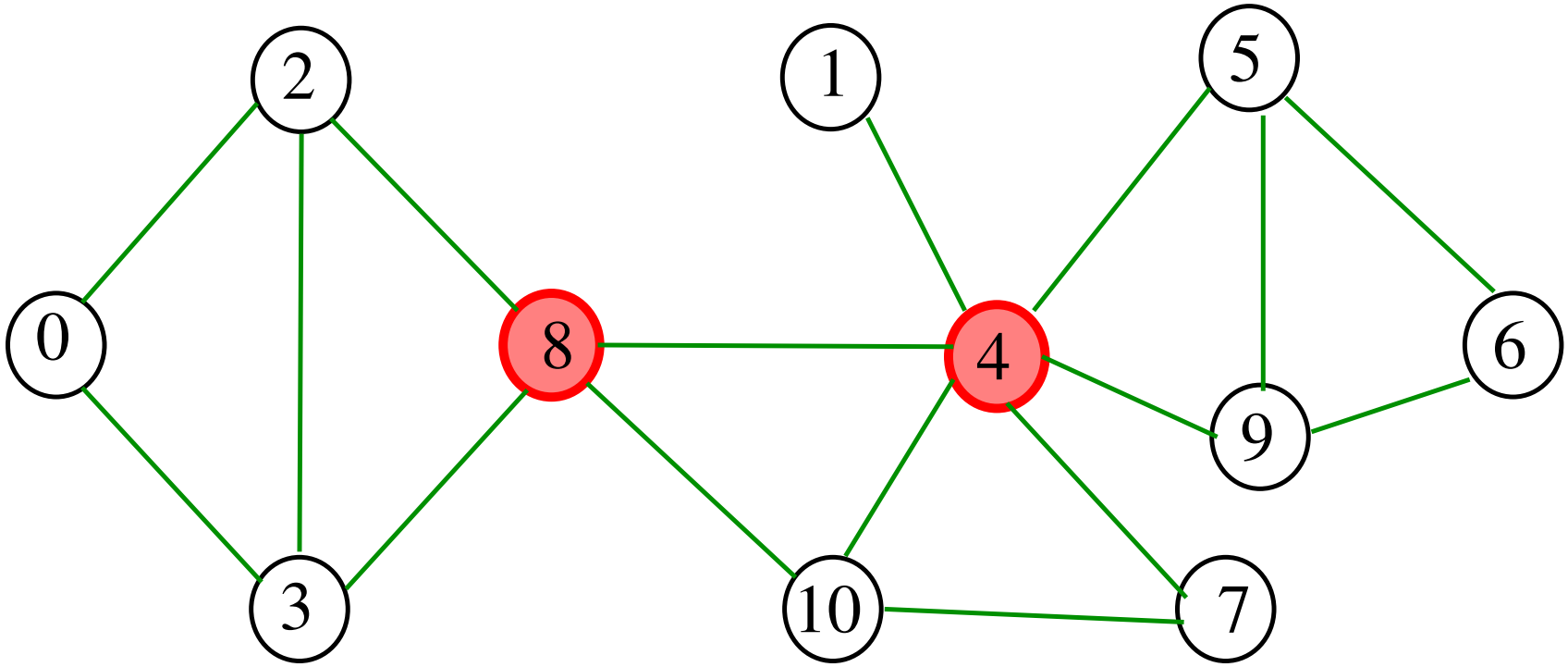
Exemplo:



Articulações em grafos

Uma **articulação** (= *articulation point*) ou **vértice de corte** (= *cut vertex*) de um grafo é um vértice cuja remoção aumenta o número de componentes

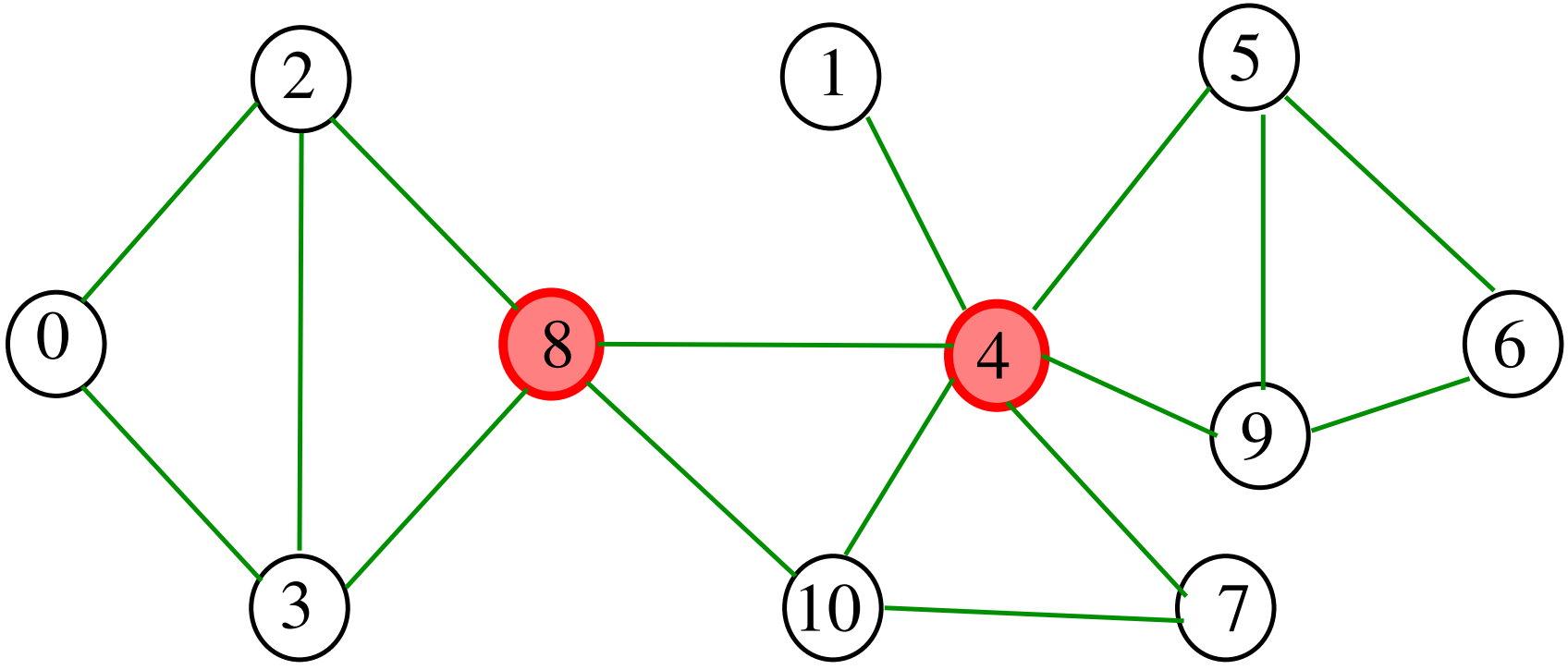
Exemplo: os vértices em vermelho são articulações



Procurando articulações

Problema: encontrar as articulações de um grafo

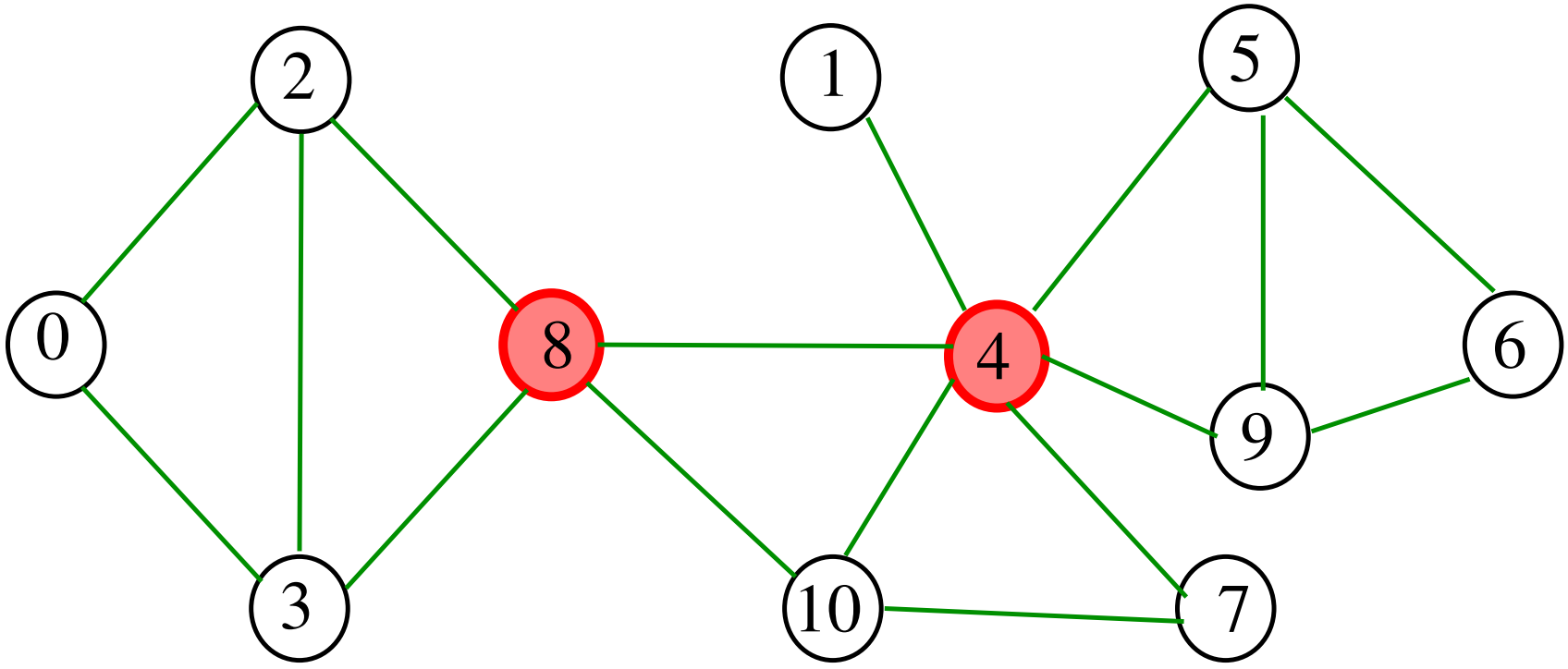
Exemplo: os vértices em vermelho são articulações



Articulações e busca em profundidade

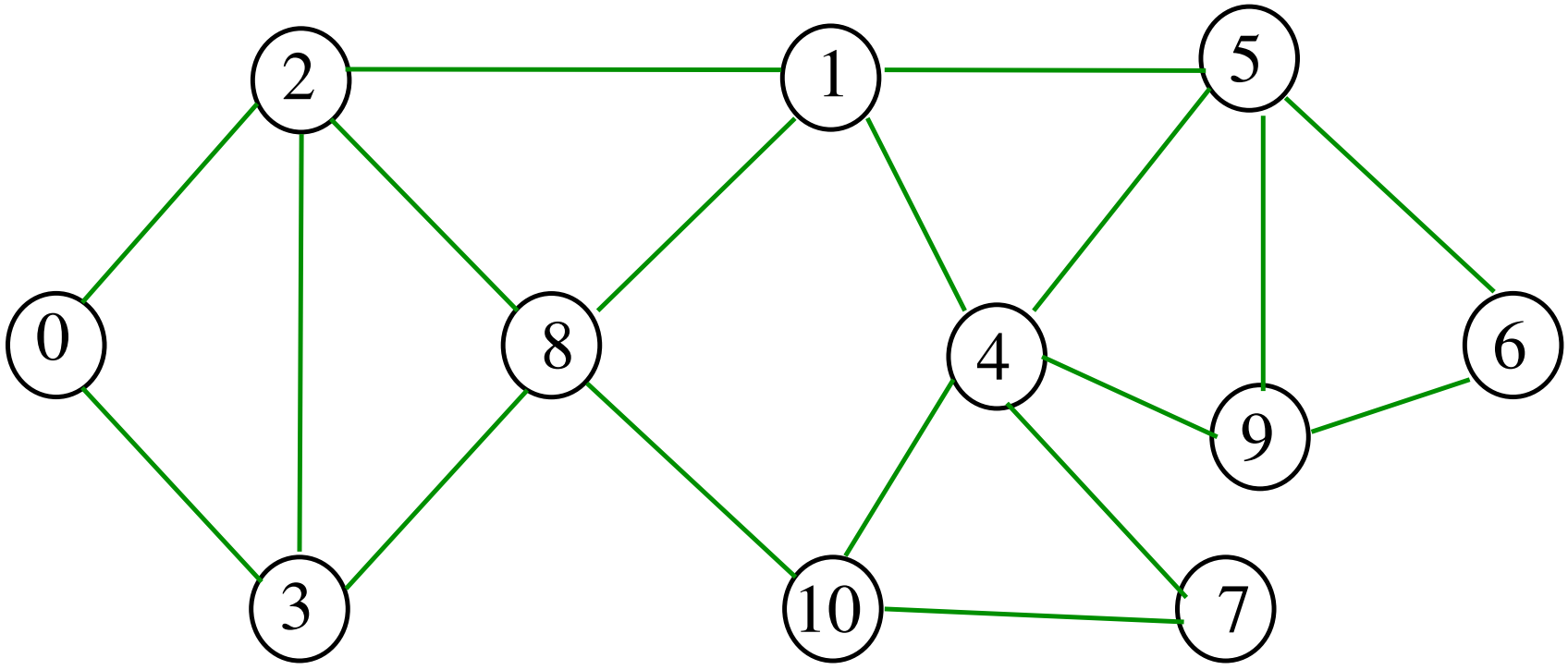
É possível encontrar todas as articulações de um grafo através de uma variante da função `bridgeR`

Exemplo: os vértices em vermelho são articulações



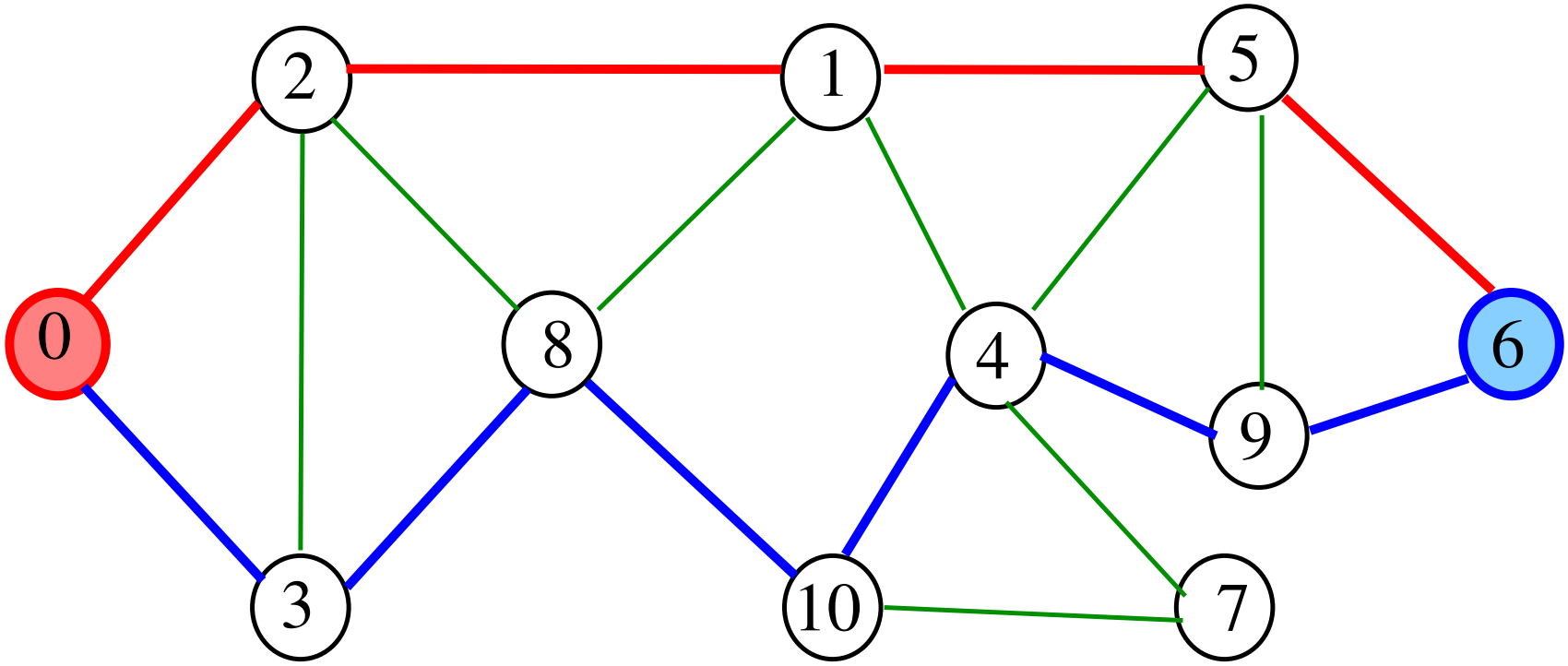
Biconexão

Um grafo é **biconexo** (= *biconnected*) ou
2-conexo se é conexo e não tem articulações



Fato básico

Um grafo é biconexo se e somente se, para cada par (s,t) de vértices, existem (pelo menos) dois caminhos de s a t sem vértices internos em comum



Componentes fortemente conexos

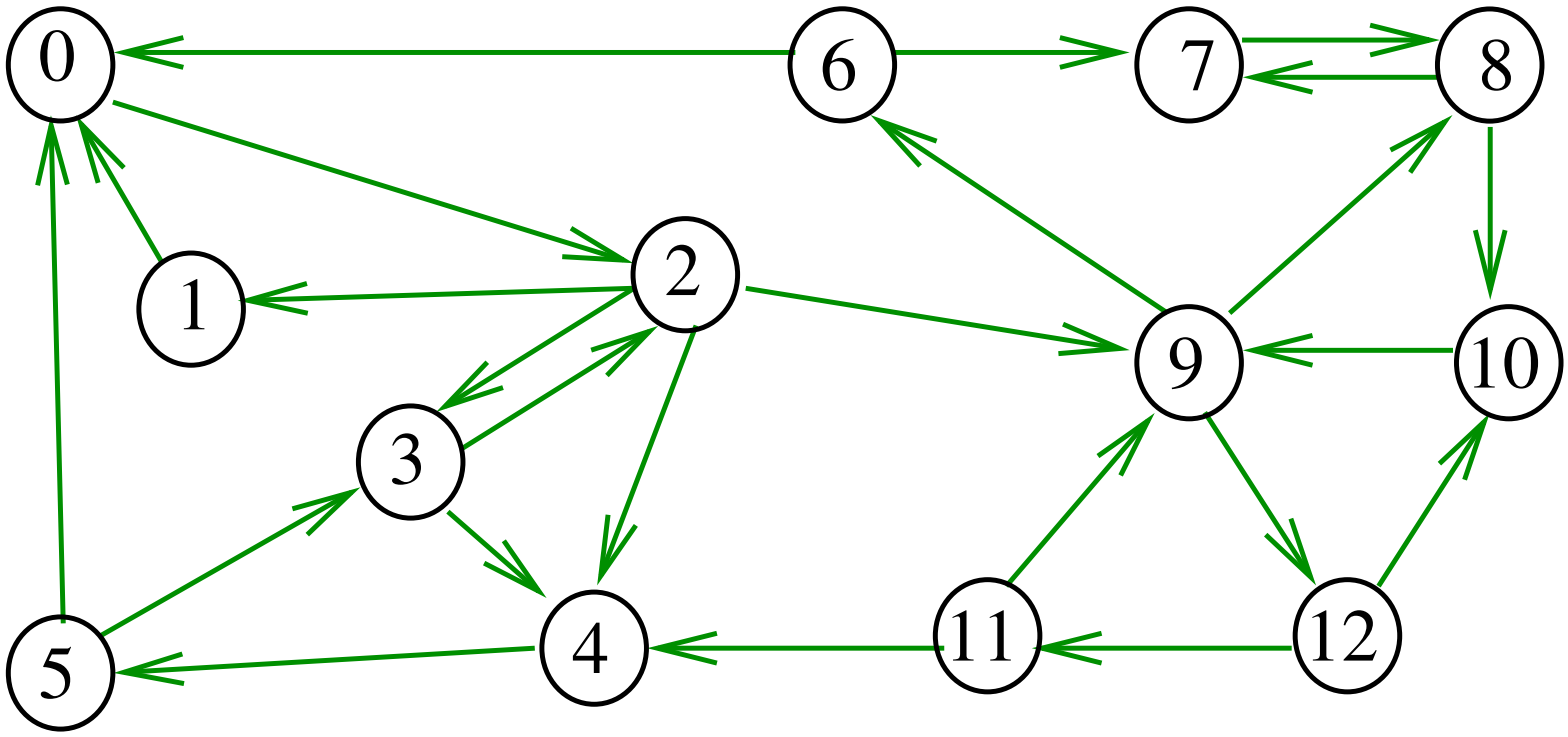
S 19.8

CLRS 22.5

Digrafos fortemente conexos

Um digrafo é **fortemente conexo** se e somente se para cada par {s,t} de seus vértices, existem caminhos de s a t e de t a s

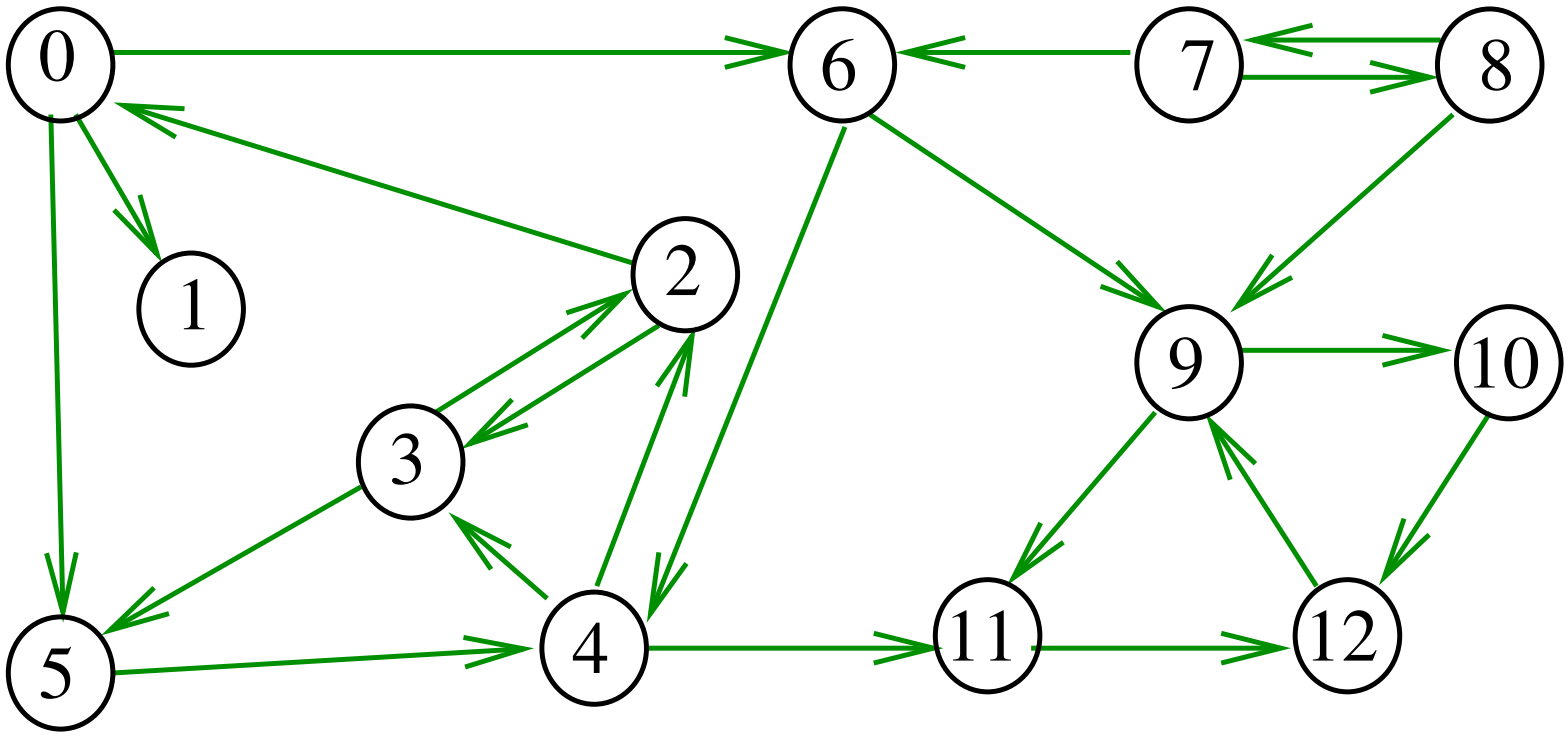
Exemplo: um digrafo fortemente conexo



Componentes fortemente conexos

Um componente **fortemente conexo** (= *strongly connected*) é um conjunto maximal de vértices `W` tal que digrafo induzido por `W` é fortemente conexo

Exemplo: 4 componentes fortemente conexos



Componentes fortemente conexos

Um componente **fortemente conexo** (= *strongly connected*) é um conjunto maximal de vértices `W` tal que digrafo induzido por `W` é fortemente conexo

Exemplo: 4 componentes fortemente conexos