

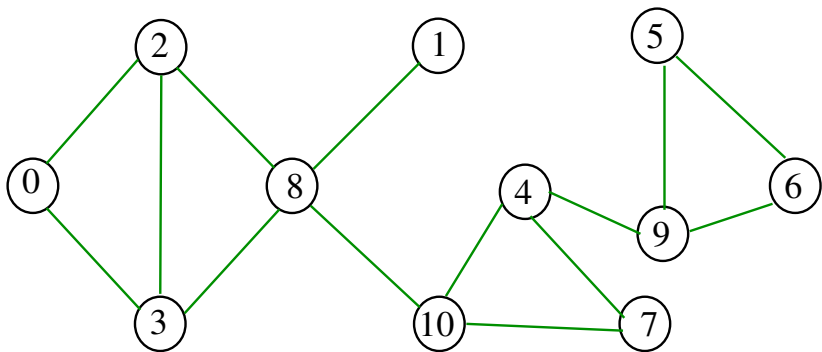
Melhores momentos

AULA 10

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.

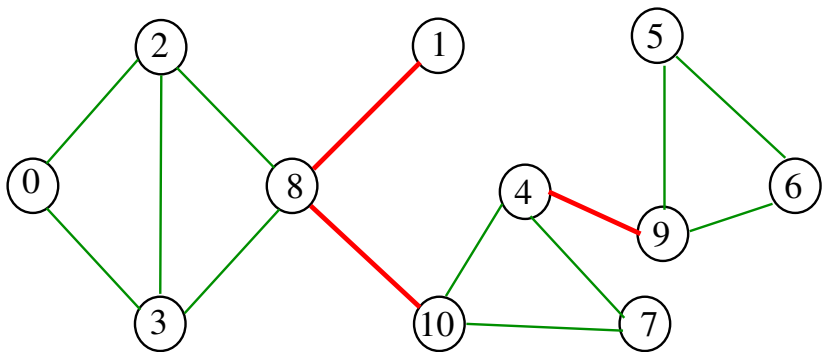
Exemplo:



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.

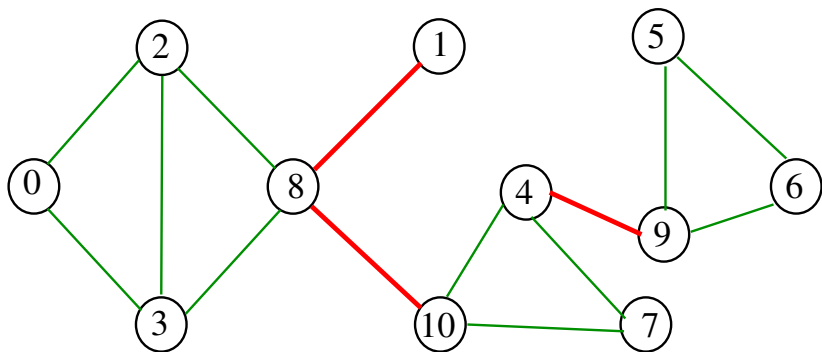
Exemplo: as arestas em **vermelho** são pontes



Procurando pontes

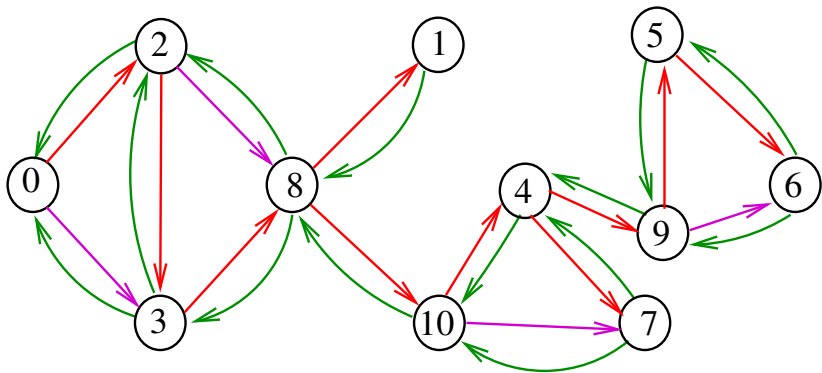
Problema: encontrar as pontes de um grafo dado

Exemplo: as arestas em **vermelho** são pontes



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



Aresta-biconexão

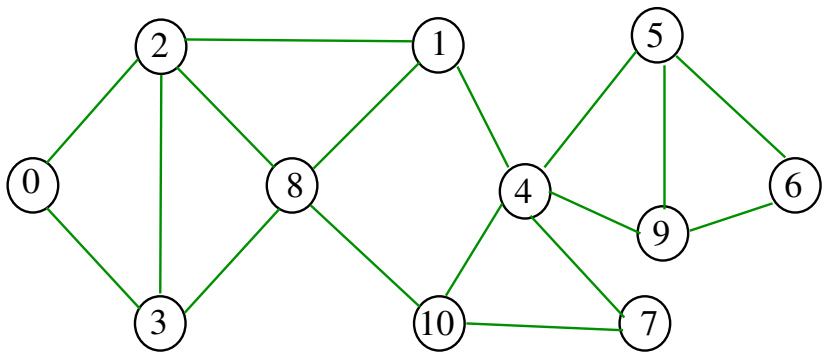
Um grafo é **aresta-biconexo** (= 2-edge-connected) ou **2-aresta-conexo** se for conexo e não tiver pontes.

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 **pele menos duas** arestas de um grafo aresta-biconexo para que ele deixe de ser conexo



AULA 11

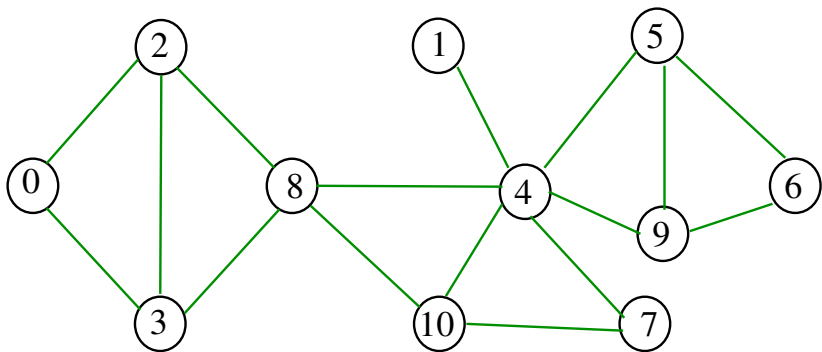
Articulações e biconexão

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

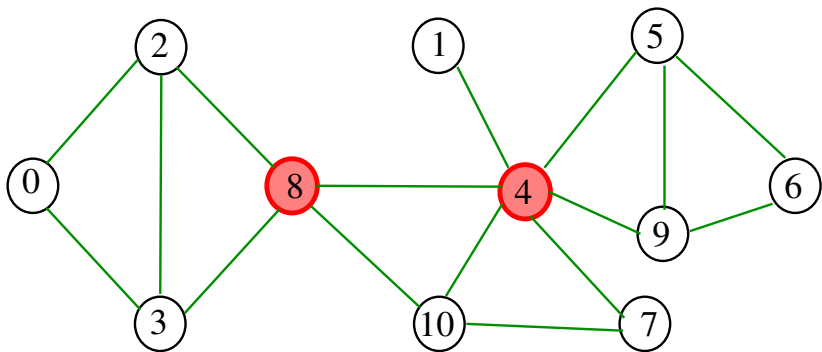
Exemplo:



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

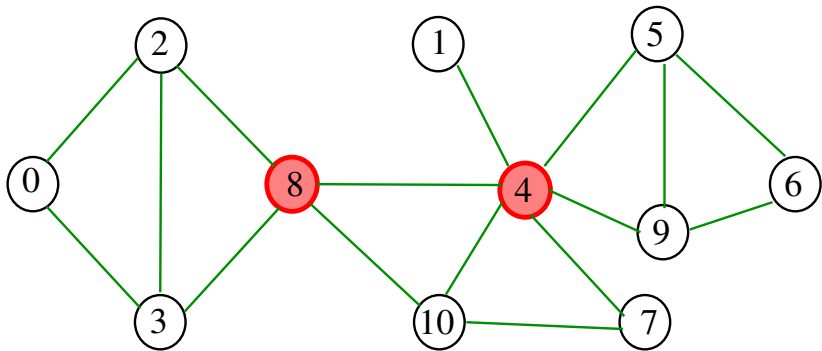
Exemplo: os vértices em **vermelho** são articulações



Articulações e busca em profundidade

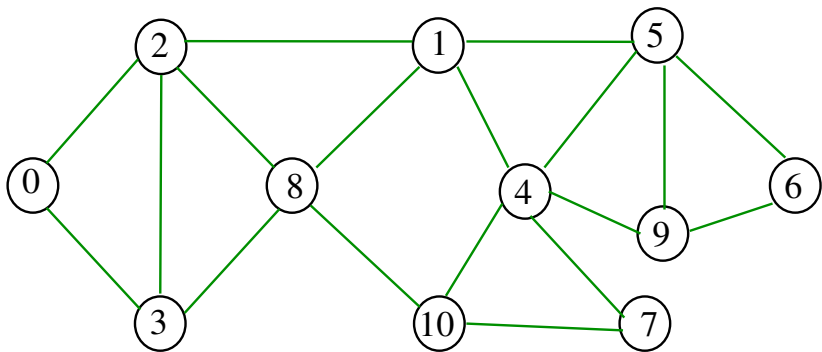
É possível encontrar todas as articulações de um grafo através de uma variante da função `bridgeR`.

Exemplo: os vértices em **vermelho** são articulações



Biconexão

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



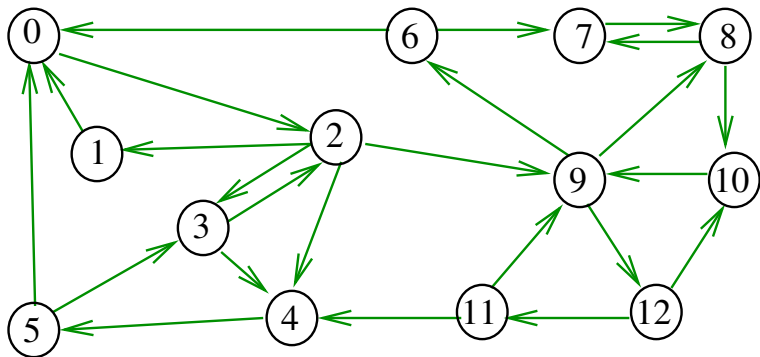
Componentes fortemente conexos

S 19.8

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

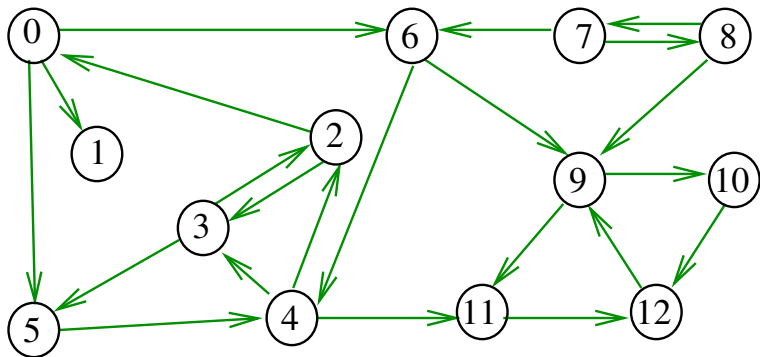
Exemplo: um digrafo fortemente conexo



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

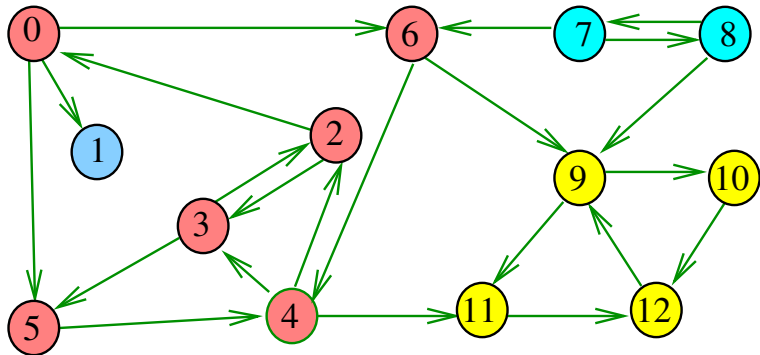
Exemplo: 4 componentes fortemente conexos



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

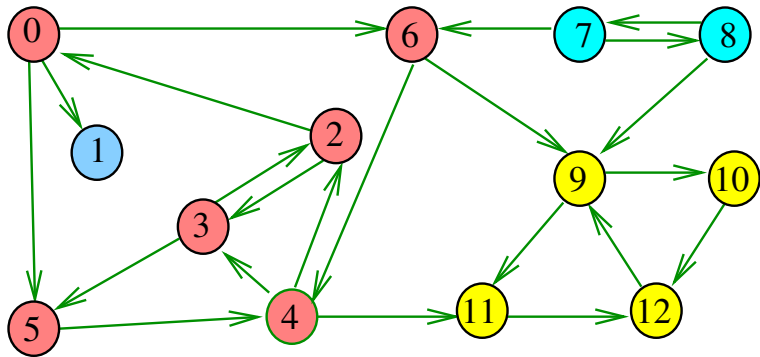
Exemplo: 4 componentes fortemente conexos



Determinando componentes f.c.

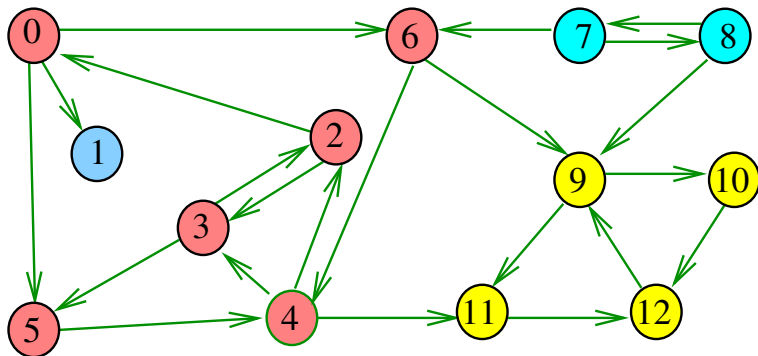
Problema: determinar os componentes fortemente conexos

Exemplo: 4 componentes fortemente conexos



Exemplo

v	0	1	2	3	4	5	6	7	8	9	10	11	12
$sc[v]$	2	1	2	2	2	2	2	3	3	0	0	0	0



strongreach

int

strongreach(Digraph G, Vertex s, Vertex t)

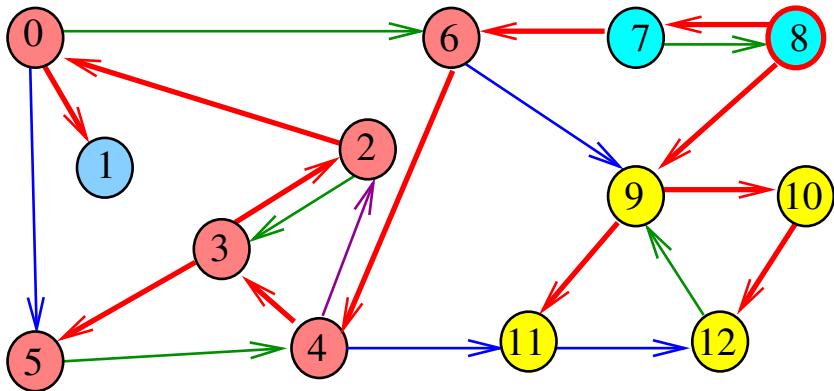
{

return sc[s]==sc[t];

}

Propriedade

Vértices de de um componente fortemente conexo é uma **subarborescência** em uma floresta DFS



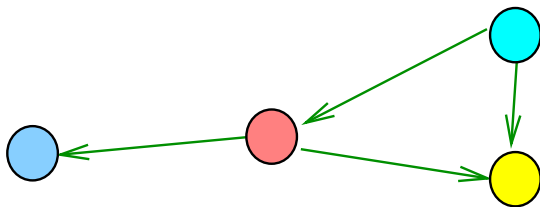
Digrafos dos componentes

O **digrafo dos componentes** de G tem um vértice para cada componente fortemente conexo e um arco $U-W$ se G possui um arco com ponta inicial em U e ponta final em W

Digrafos dos componentes

O **digrafo dos componentes** de G tem um vértice para cada componente fortemente conexo e um arco $U-W$ se G possui um arco com ponta inicial em U e ponta final em W

Digrafo dos componente é um DAG

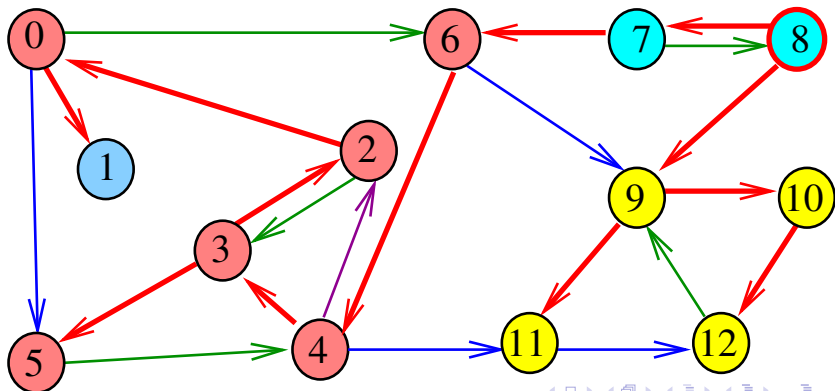


Numeração pós-ordem

$\text{pos}[v]$ = numeração pós-ordem de v

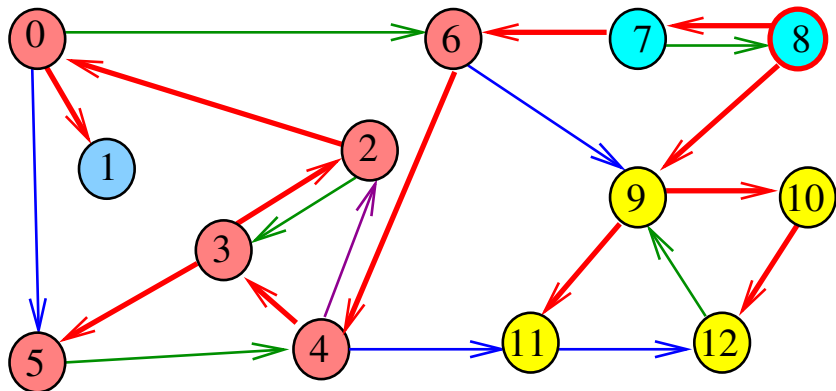
$\text{sop}[i]$ = vértice de numeração pós-ordem i

$\text{pos}[W]$ = maior numeração pós-ordem de um vértice em W



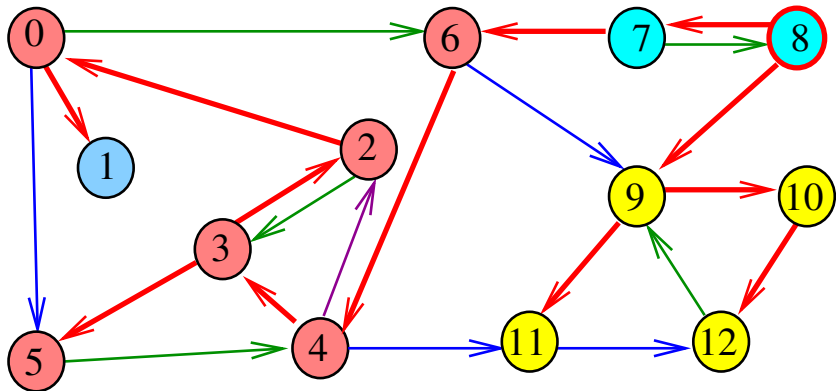
Exemplo

v	0	1	2	3	4	5	6	7	8	9	10	11	12
pos[v]	6	5	7	8	9	4	10	11	12	3	1	2	0



Exemplo

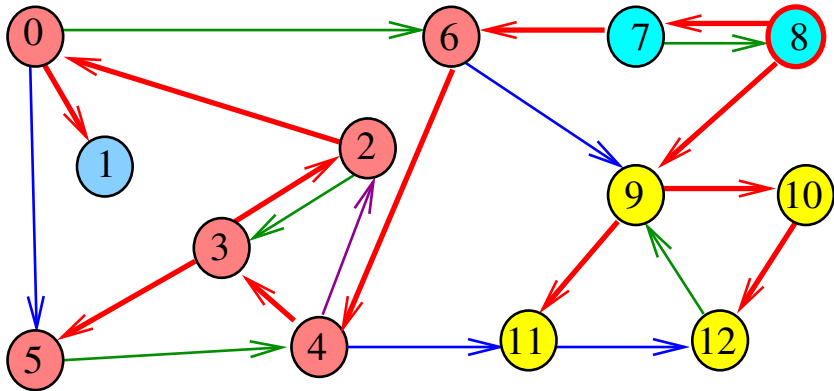
i	0	1	2	3	4	5	6	7	8	9	10	11	12
$\text{sop}[i]$	12	10	11	9	5	1	0	2	3	4	6	7	8



Numeração pós-ordem e componentes f.c.

Se U e W são componentes f.c. e existe arco com ponta inicial em U e ponta final em W , então

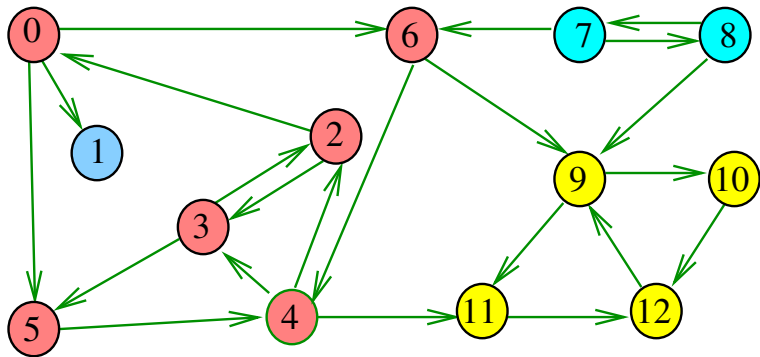
$$\text{pos}[U] > \text{pos}[W]$$



Propriedade

Um digrafo G e seu digrafo reverso R têm os **mesmos** componente fortemente conexos

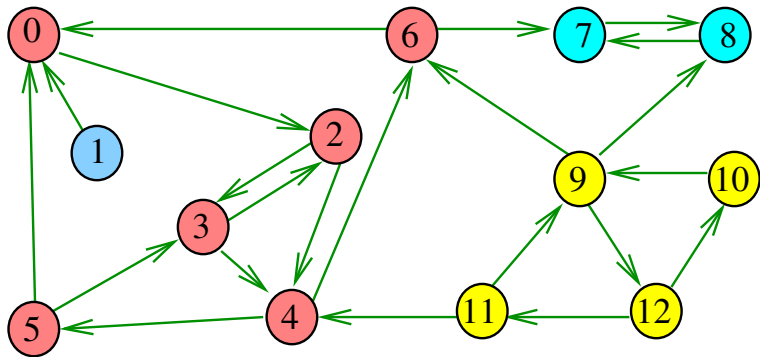
Exemplo: Digrafo G



Propriedade

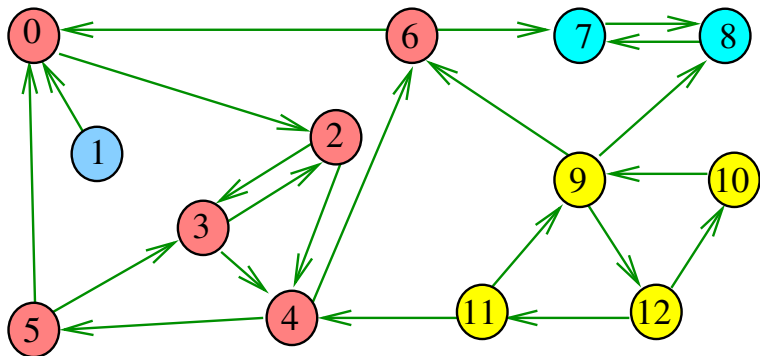
Um digrafo G e seu digrafo reverso R têm os **mesmos** componente fortemente conexos

Exemplo: Digrafo reverso R de G



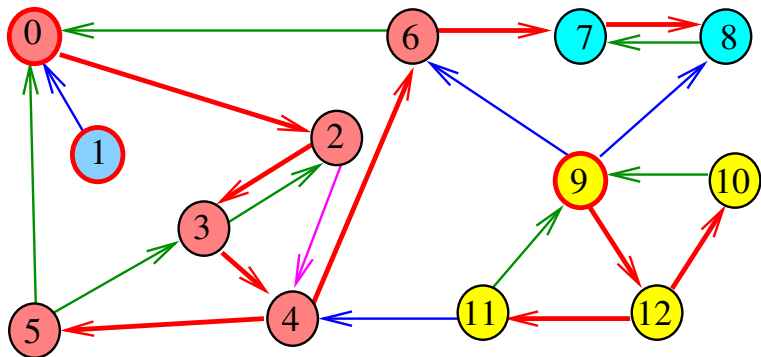
Digrafo reverso

v	0	1	2	3	4	5	6	7	8	9	10	11	12
$sc[v]$	2	1	2	2	2	2	2	3	3	0	0	0	0



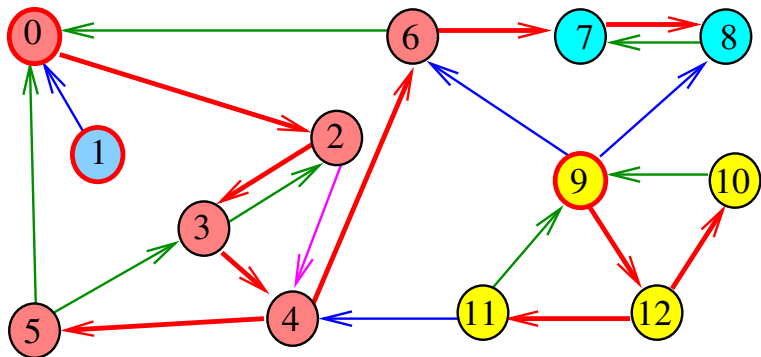
Digrafo reverso e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
sop[i]	8	7	6	5	4	3	2	0	1	10	11	12	9



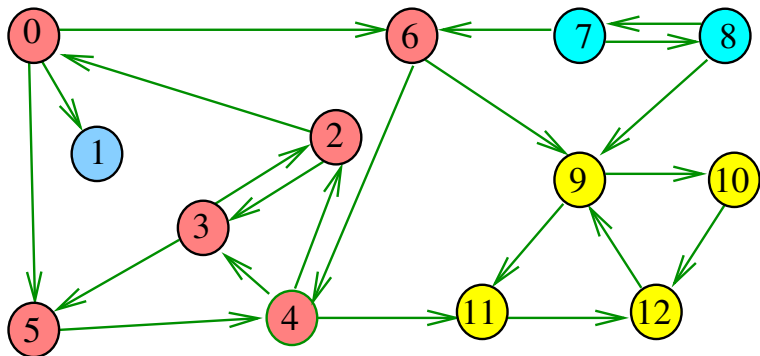
Digrafo reverso e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$sop[i]$	8	7	6	5	4	3	2	0	1	10	11	12	9



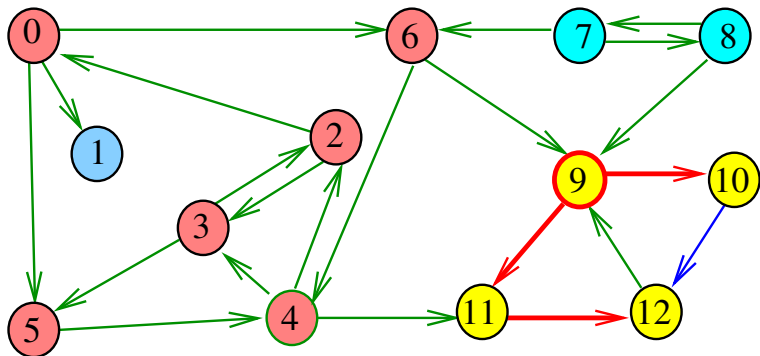
Digrafo e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
sop[i]	8	7	6	5	4	3	2	0	1	10	11	12	9



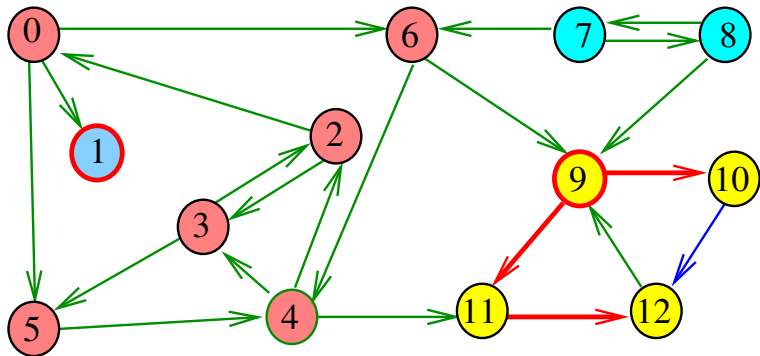
Digrafo e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
sop[i]	8	7	6	5	4	3	2	0	1	10	11	12	9



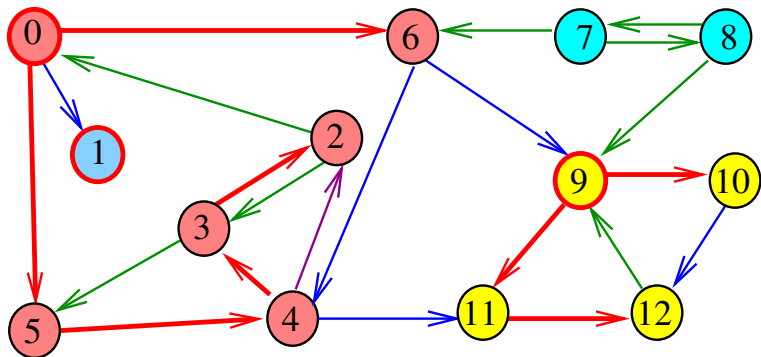
Digrafo e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
sop[i]	8	7	6	5	4	3	2	0	1	10	11	12	9



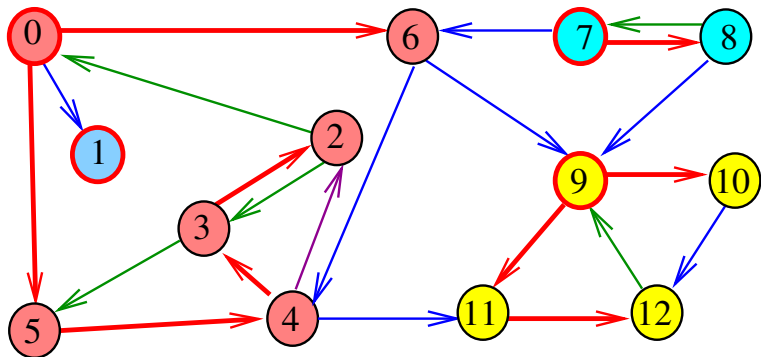
Digrafo e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$\text{sop}[i]$	8	7	6	5	4	3	2	0	1	10	11	12	9



Digrafo e DFS

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$sop[i]$	8	7	6	5	4	3	2	0	1	10	11	12	9



Algoritmo de Kosaraju

A função devolve o número de componentes fortemente conexos do digrafo **G**

```
static int sc[maxV] , sop[maxV] , sopR[maxV] ;  
static int cnt, id;
```

Além disso, ela armazena no vetor **sc** o número do componente a que o vértice pertence: se o vértice **v** pertence ao **k**-ésimo componente então **sc[v] == k-1**

```
int DIGRAPHsc (Graph G)
```


DIGRAPHsc

```
int DIGRAPHsc (Digraph G) {  
    Vertex v;  
    int id, i  
1    cnt = 0;  
2    R = DIGRAPHreverse(G);  
3    for (v = 0; v < G->V; v++) sc[v] = -1;  
4    for (v = 0; v < G->V; v++)  
5        if (sc[v] == -1)  
6            dfsRsc(R, v, 0);  
}
```

DIGRAPHsc

```
7  id = cnt = 0;
8  for (v = 0; v < G->V; v++) sc[v] = -1;
9  for (i = 0; i < G->V; i++)
10     sopR[i] = sop[i]
11  for (i=G->V-1; i>0; i--)
12     if (sc[sopR[i]] == -1)
13         dfsRsc(R, sopR[i], id++);
14  DIGRAPHdestroy(R);
15  return id;
}
```

dfsRsc

```
void dfsRsc(Digraph G, Vertex v, int id){  
    link p;  
1   sc[v] = id;  
2   for (p=G->adj[v]; p!=NULL; p=p->next)  
3       if (sc[p->w] == -1)  
4           dfsRsc(G, p->w, id);  
5   sop[cnt++] = v;  
}
```

DIGRAPHreverse

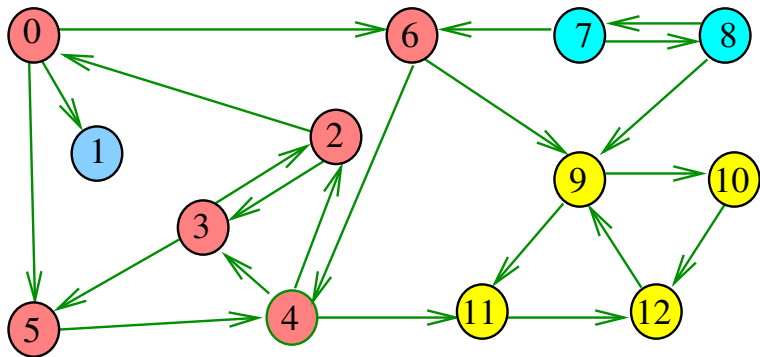
```
Digraph DIGRAPHreverse (Digraph G) {  
1  Vertex v; link p;  
2  Digraph R= DIGRAPHinit(G->V);  
3  for (v=0; v<G->V; v++)  
4      for (p=G->adj[v]; p!=NULL; p=p->next)  
5          DIGRAPHinsertA(R, p->w, v);  
6  return R;  
}
```

Consumo de tempo

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

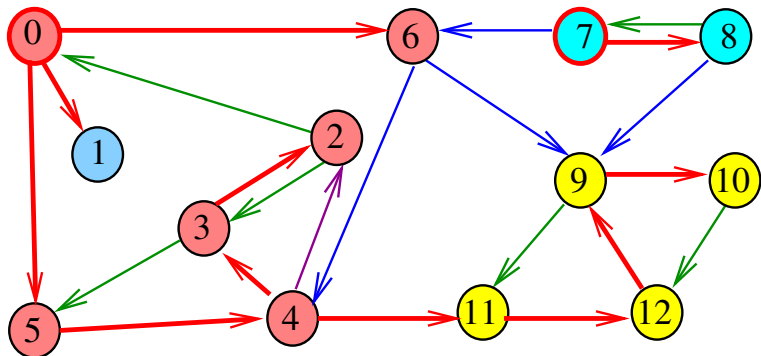
Algoritmo de Tarjan

O menor **número de pré-ordem** de um vértice “ativo” que pode ser alcançado por v utilizando arcos da **arborescência** e **até um** arco de **retorno** será denotado por $low[v]$



Exemplo

v	0	1	2	3	4	5	6	7	8	9	10	11	12
$\text{pre}[v]$	0	9	4	3	2	1	10	11	12	7	8	5	6
$\text{low}[v]$	0	9	0	0	0	0	0	11	11	5	6	5	5



DIGRAPHsc

```
void DIGRAPHsc (Graph G) {  
    Vertex v;  
1    cnt = id = t = 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            dfsRsc(G, v);  
}
```



```

void dfsRsc(Digraph G, Vertex v){
    link p; Vertex w; int min;
1   pre[v] = cnt++; low[v] = pre[v];
2   min = low[v]; s[t++] = v;
3   for (p=G->adj[v]; p!=NULL; p=p->next){
4       if (pre[w=p->w]==-1) dfsRsc(G, w);
6       if (low[w] < min) min=low[w];
    }
7   if (min<low[v]) {low[v]=min; return;}
8   do {
9       sc[w=s[--t]]=id; low[w]=G->V;
10  } while (s[t] != v);
11  id++;
    }

```

```

void dfsRsc(Digraph G, Vertex v){
    link p; Vertex w;
1   pre[v] = cnt++; low[v] = pre[v];
2   s[t++] = v;
3   for (p=G->adj[v]; p!=NULL; p=p->next){
4       if (pre[w=p->w]==-1) dfsRsc(G, w);
6       if (low[w] < low[v]) low[v]=low[w];
    }
7   if (low[v]<pre[v]) return;
8   do {
9       sc[w=s[--t]]=id; low[w]=G->V;
10  } while (s[t] != v);
11  id++;
    }

```