

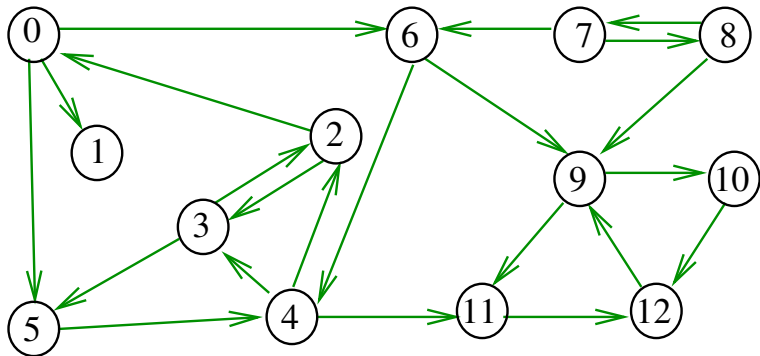
Componentes fortemente conexas

S 19.8
CLRS 22.5

Componentes fortemente conexas

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

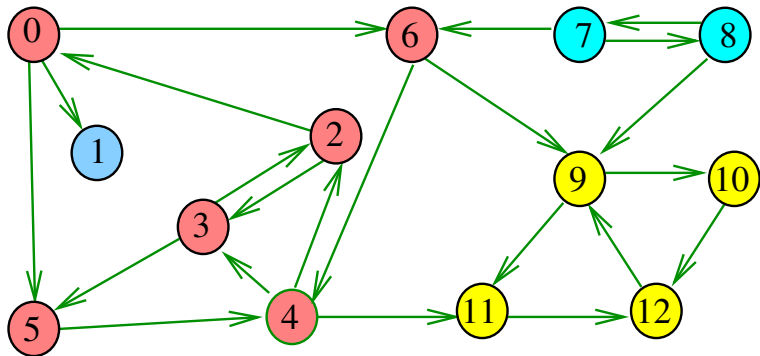
Exemplo: 4 componentes fortemente conexas



Componentes fortemente conexas

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

Exemplo: 4 componentes fortemente conexas



Relações de equivalência

Dada uma relação binária \Rightarrow no conjunto \mathcal{C} , seja \Rightarrow^* seu fecho reflexivo e transitivo. Defina, para $a, b \in \mathcal{C}$

$$a \Leftrightarrow b \quad \text{sse} \quad a \Rightarrow^* b \quad \text{e} \quad b \Rightarrow^* a$$

claro que \Leftrightarrow é um **relação de equivalência**.

Caso típico: \Rightarrow é alguma transformação de dados.

Problema: *Determinar as classes de equivalência.*

Mesma coisa que achar as componentes fortemente conexas do

grafo com vértices \mathcal{C} e arcos dados por \Rightarrow .

Relações de equivalência

Dada uma relação binária \Rightarrow no conjunto \mathcal{C} , seja \Rightarrow^* seu fecho reflexivo e transitivo. Defina, para $a, b \in \mathcal{C}$

$$a \Leftrightarrow b \quad \text{sse} \quad a \Rightarrow^* b \quad \text{e} \quad b \Rightarrow^* a$$

claro que \Leftrightarrow é um **relação de equivalência**.

Caso típico: \Rightarrow é alguma transformação de dados.

Problema: *Determinar as classes de equivalência.*

Mesma coisa que achar as componentes fortemente conexas do

grafo com vértices \mathcal{C} e arcos dados por \Rightarrow .

Relações de equivalência

Dada uma relação binária \Rightarrow no conjunto \mathcal{C} , seja \Rightarrow^* seu fecho reflexivo e transitivo. Defina, para $a, b \in \mathcal{C}$

$$a \Leftrightarrow b \quad \text{sse} \quad a \Rightarrow^* b \quad \text{e} \quad b \Rightarrow^* a$$

claro que \Leftrightarrow é um **relação de equivalência**.

Caso típico: \Rightarrow é alguma transformação de dados.

Problema: *Determinar as classes de equivalência.*

Mesma coisa que achar as componentes fortemente conexas do

grafo com vértices \mathcal{C} e arcos dados por \Rightarrow .

Relações de equivalência

Dada uma relação binária \Rightarrow no conjunto \mathcal{C} , seja \Rightarrow^* seu fecho reflexivo e transitivo. Defina, para $a, b \in \mathcal{C}$

$$a \Leftrightarrow b \quad \text{sse} \quad a \Rightarrow^* b \quad \text{e} \quad b \Rightarrow^* a$$

claro que \Leftrightarrow é um **relação de equivalência**.

Caso típico: \Rightarrow é alguma transformação de dados.

Problema: *Determinar as classes de equivalência.*

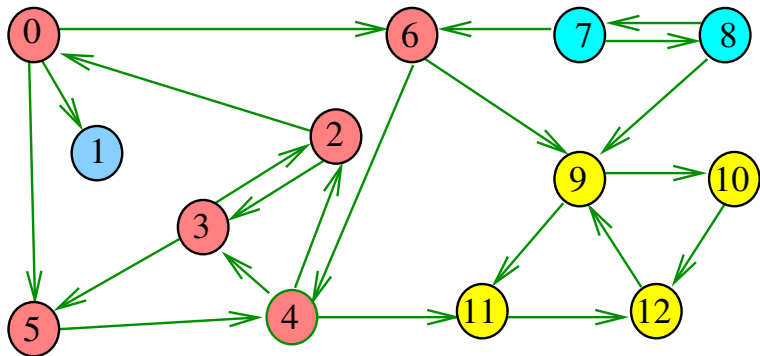
Mesma coisa que achar as componentes fortemente conexas do

grafo com vértices \mathcal{C} e arcos dados por \Rightarrow .

Determinando componentes f.c.

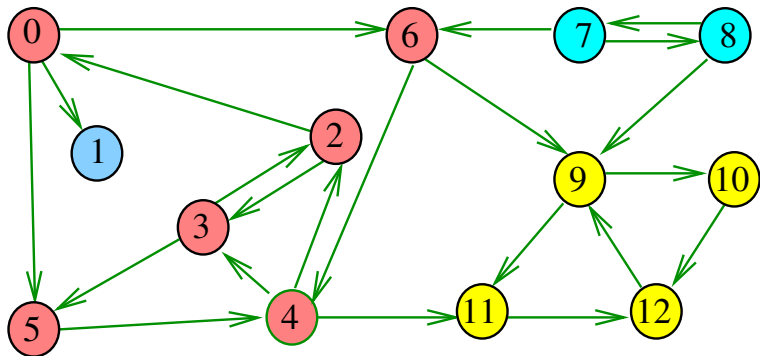
Problema: determinar as componentes fortemente conexas

Exemplo: 4 componentes fortemente conexas



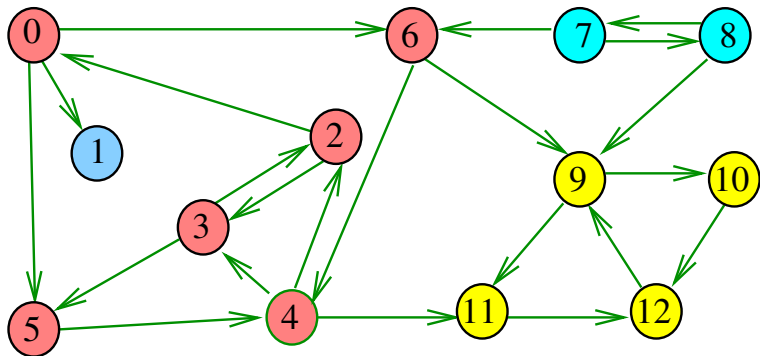
Exemplo

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



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] ;  
}
```

Força Bruta

```
int DIGRAPHsc1 (Digraph G) {  
    Vertex v , w ; int n ;  
    Graph H = GRAPHinit(G->V);  
1   for (v = 0; v < G->V; v++)  
2       for (w = v+1; w < G->V; w++)  
3           if (DIGRAPHpath(G,v,w)==1  
                && DIGRAPHpath(G,w,v)==1)  
4               GRAPHinsertE(H,v,w);  
5   n = GRAPHcc(H);  
6   for (v = 0; v < G->V; v++)  
7       sc[v]=cc[v];  
8   return n ;  
}
```

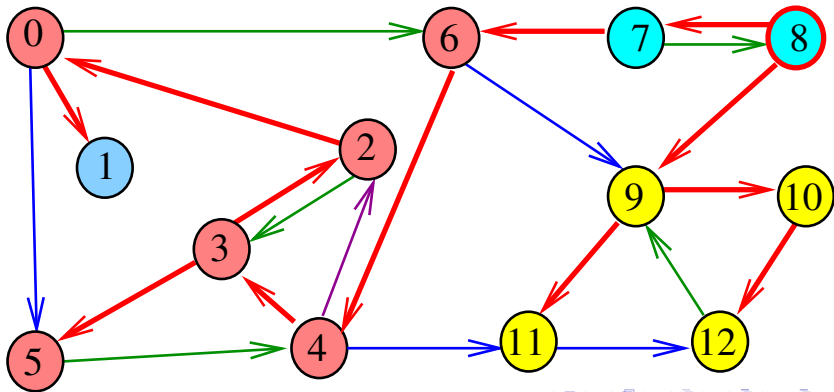
Consumo de tempo

O consumo de tempo da função `DIGRAPHsc1` para **vetor de listas de adjacência** é $O(V^2(V + A))$.

O consumo de tempo da função `DIGRAPHsc1` para **matriz de adjacência** é $O(V^4)$.

Propriedade

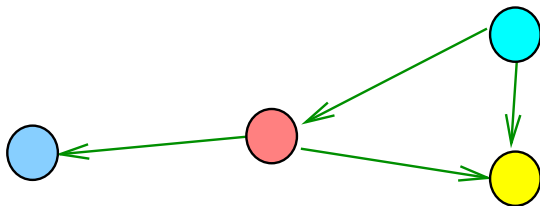
Os vértices de uma componente fortemente conexa induzem uma **subarborescência** em uma floresta DFS



Digrafo das componentes

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

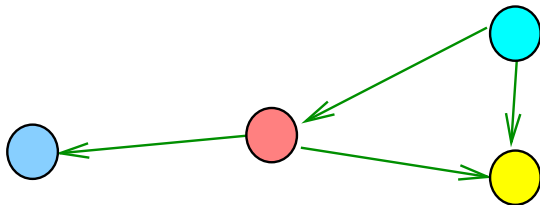
Digrafo das componente é um DAG



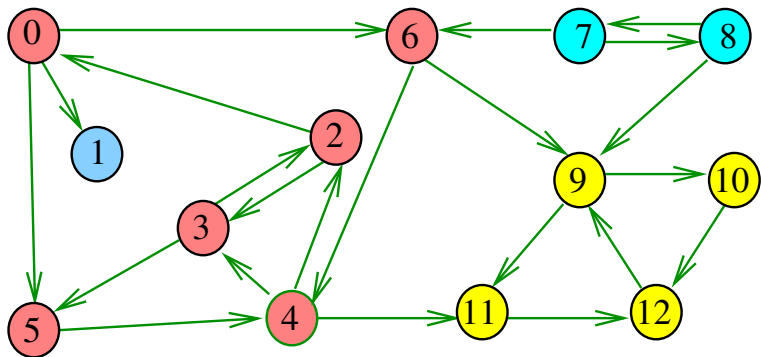
Digrafo das componentes

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

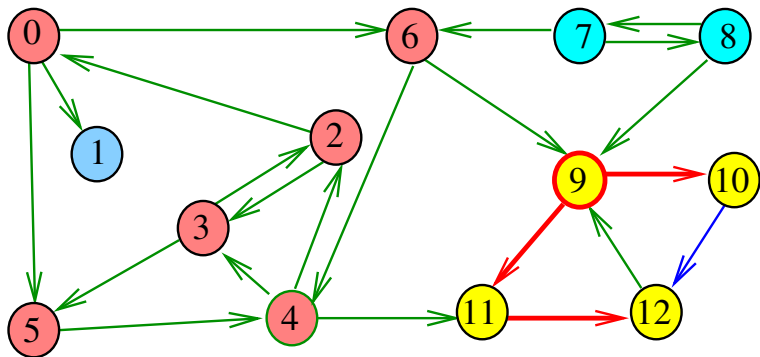
Digrafo das componente é um DAG



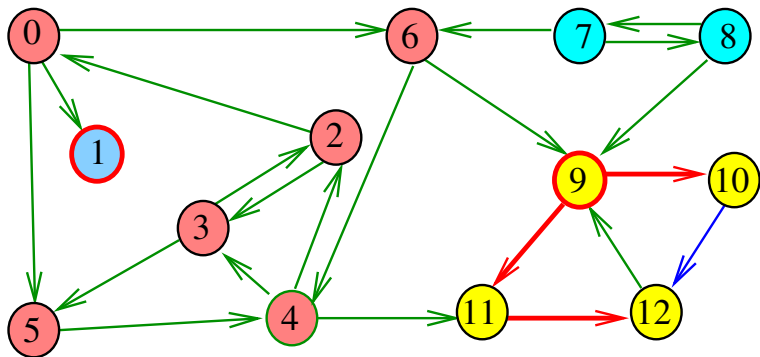
Idéia ... G e DFS



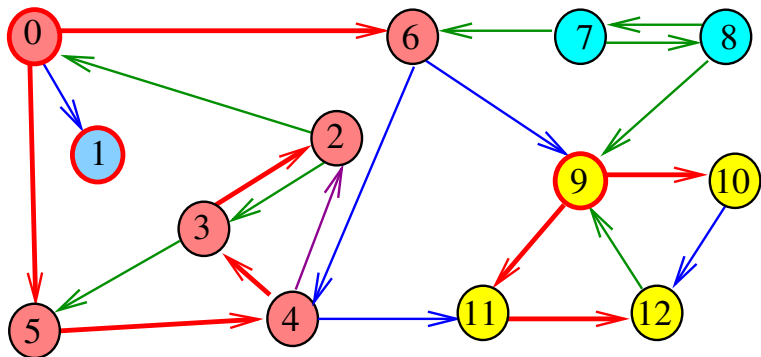
Idéia ... G e DFS



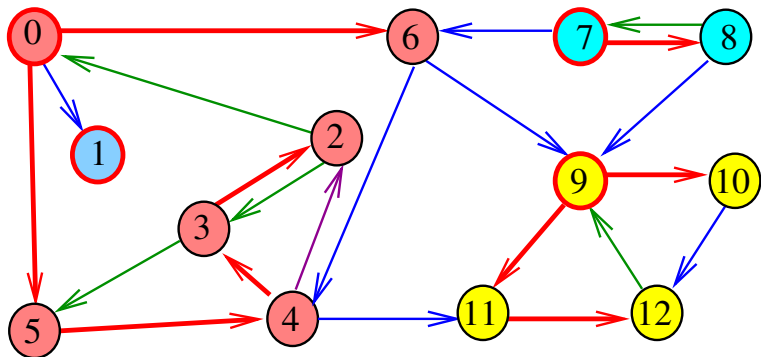
Idéia ... G e DFS



Idéia ... G e DFS



Idéia ... G e DFS

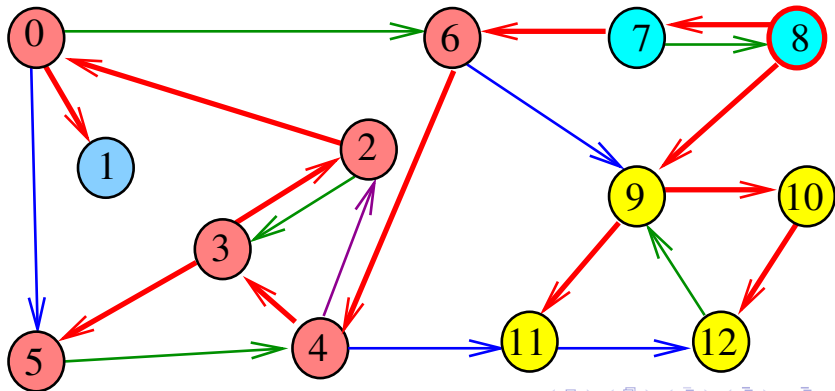


Numeração pós-ordem (finalização)

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

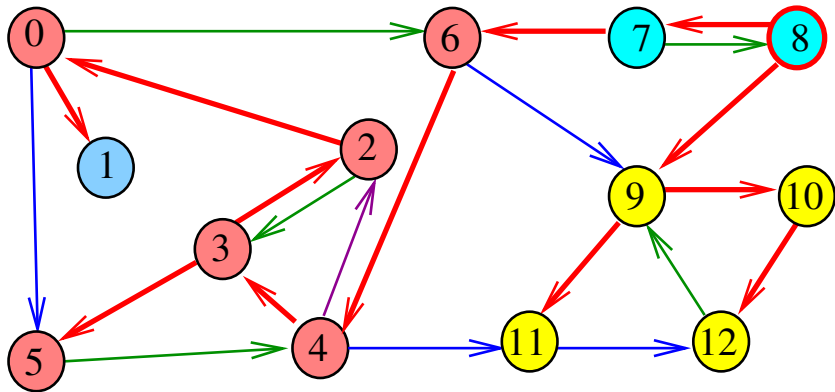
$\text{sop}[i]$ = vértice de numeração pós-ordem i

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



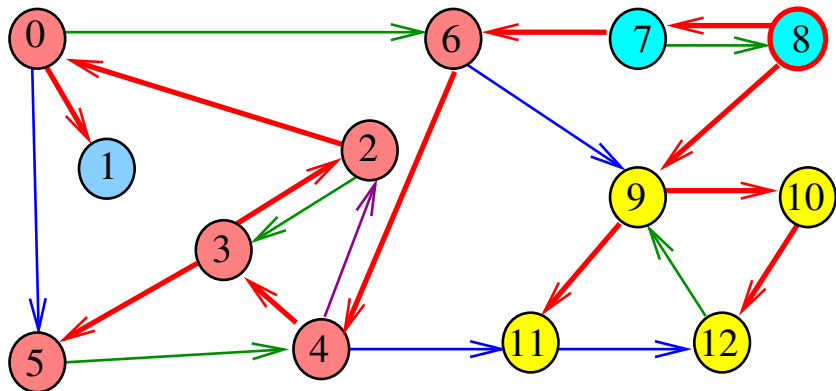
Exemplo

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



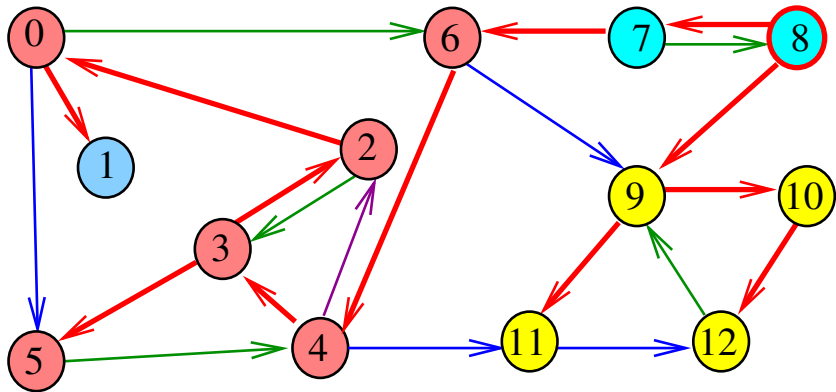
Exemplo

$\text{pos}[\{7, \mathbf{8}\}] = 12$

$\text{pos}[\{0, 2, 3, 4, 5, \mathbf{6}\}] = 10$

$\text{pos}[\{\mathbf{1}\}] = 5$

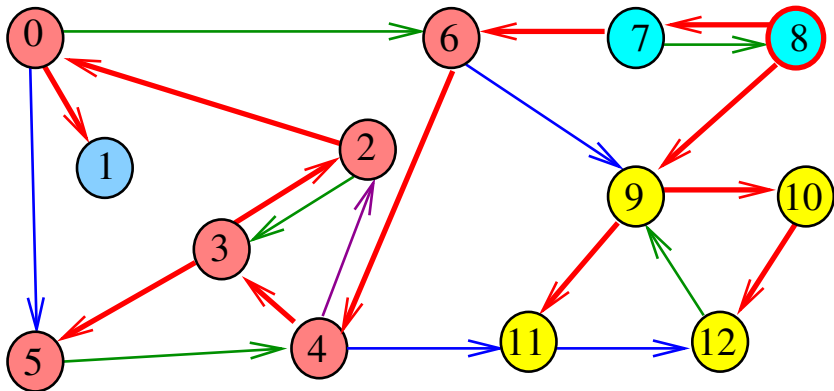
$\text{pos}[\{\mathbf{9}, 10, 11, 12\}] = 3$



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

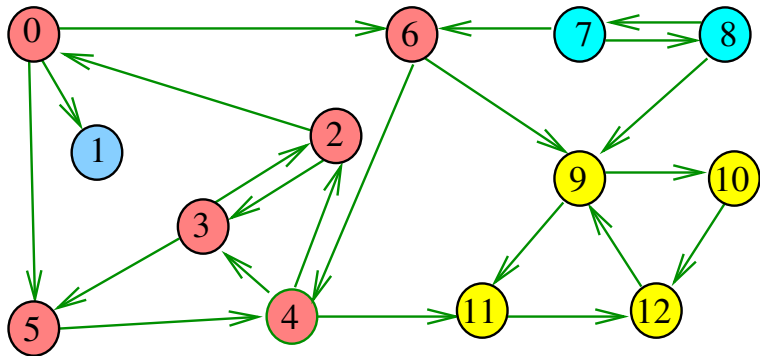
$$\text{pos}[U] > \text{pos}[W]$$



Propriedade

Um digrafo G e seu digrafo reverso R têm as **mesmas** componentes fortemente conexas

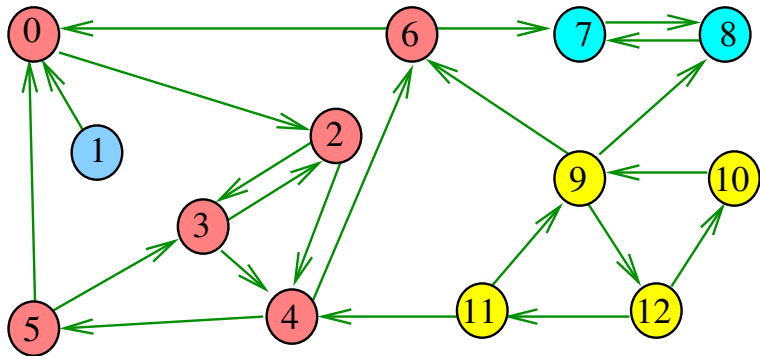
Exemplo: Digrafo G



Propriedade

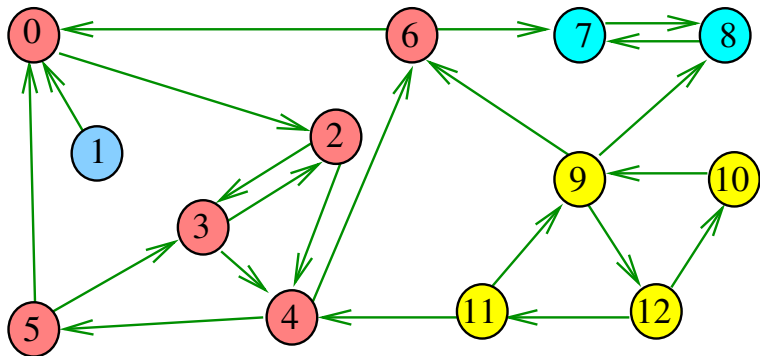
Um digrafo G e seu digrafo reverso R têm as **mesmas** componentes fortemente conexas

Exemplo: Digrafo reverso R de G



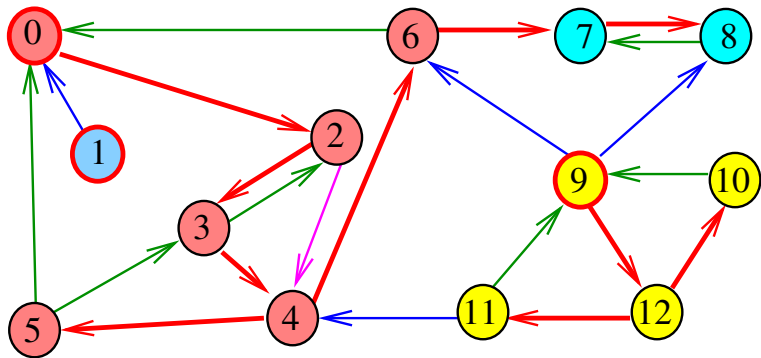
Digrafo reverso R

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



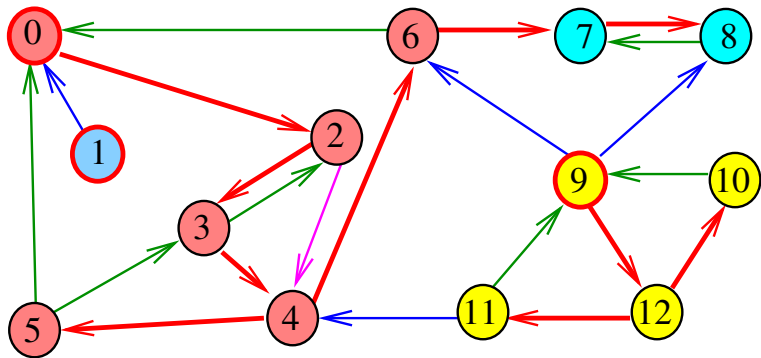
Digrafo reverso R e DFS

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



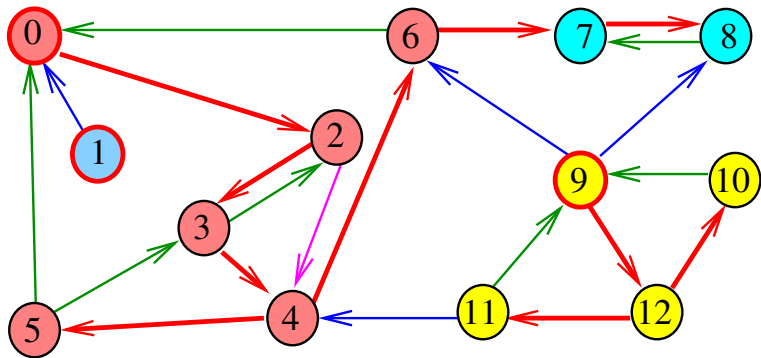
Digrafo reverso R e DFS

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



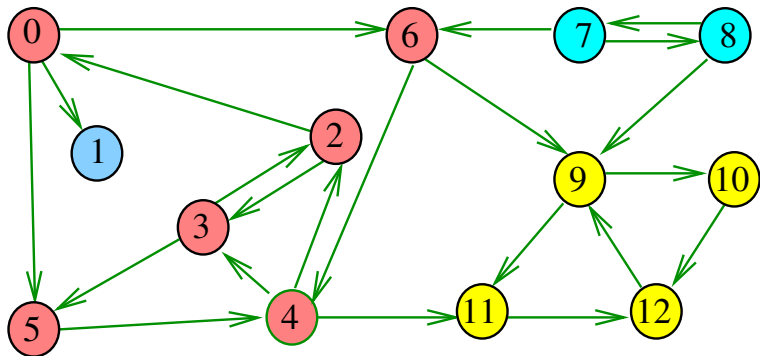
Digrafo reverso R e DFS

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



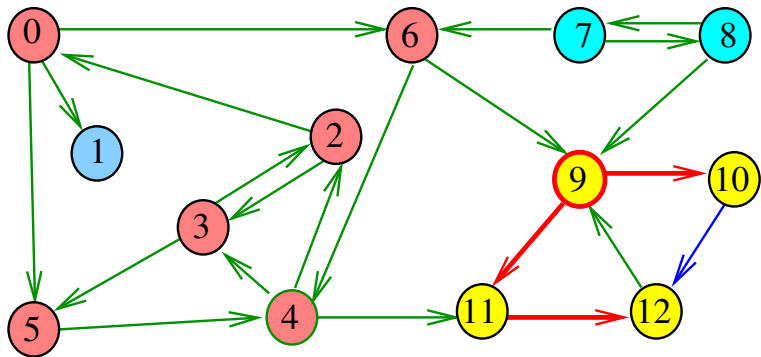
Digrafo G e DFS

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



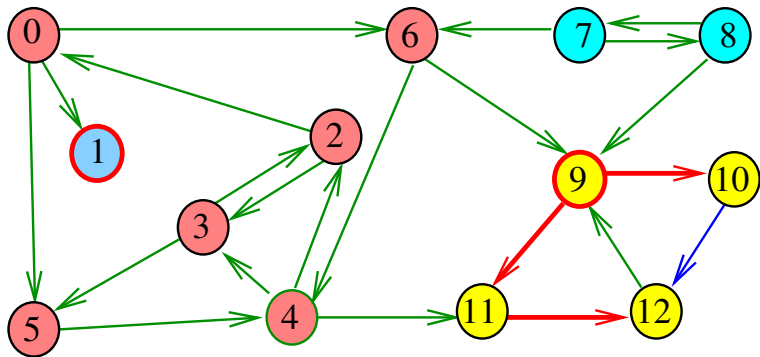
Digrafo G e DFS

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



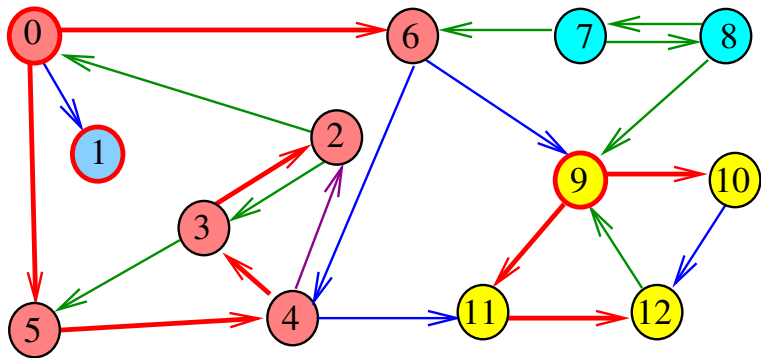
Digrafo G e DFS

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



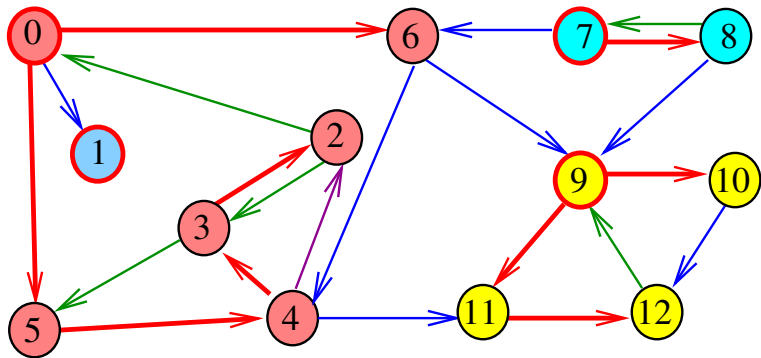
Digrafo G e DFS

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 G 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



Algoritmo de Kosaraju

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

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

Além disso, ela armazena no vetor *sc* o número da componente a que o vértice pertence: se o vértice **v** pertence à **k**-ésima componente então $sc[v] == k-1$

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

DIGRAPHsc

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


DIGRAPHsc

```
7  for (i = 0; i < G->V; i++)
8      sopR[i] = sop[i];
9  cnt = id = 0;
10 for (v = 0; v < G->V; v++) sc[v]=-1;
11 for (i = G->V-1; i > 0; i--)
12     if (sc[sopR[i]] == -1)
13         dfsRsc(G, 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; p = p->next)
3       if (sc[p->w] == -1)
4           dfsRsc(G, p->w, id);
5   pos[v] = cnt; /* não precisa */
6   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; p = p->next)  
5          DIGRAPHinsertA(G,p->w,v);  
6  return R;  
}
```

Consumo de tempo

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