# Componentes fortemente conexos

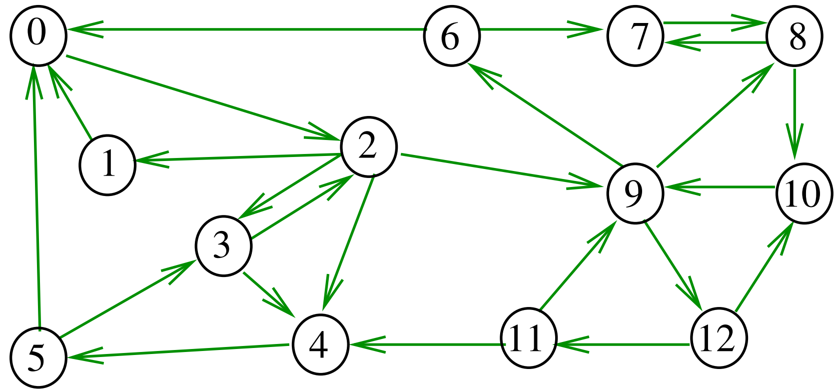
S 19.8
CLRS 22.5



# 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

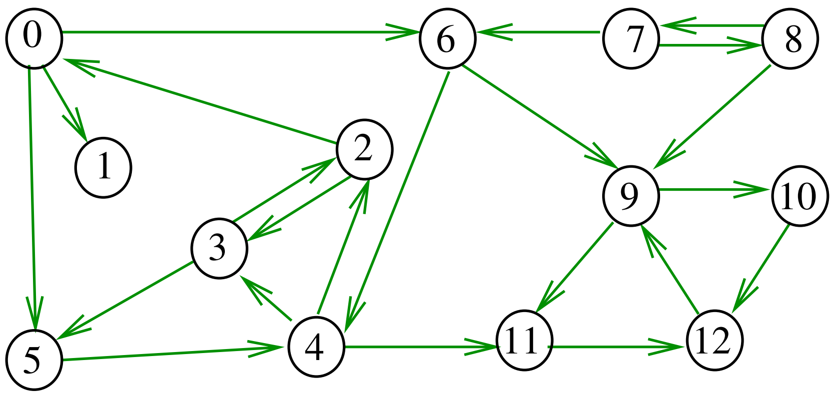





# Componentes fortemente conexos

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



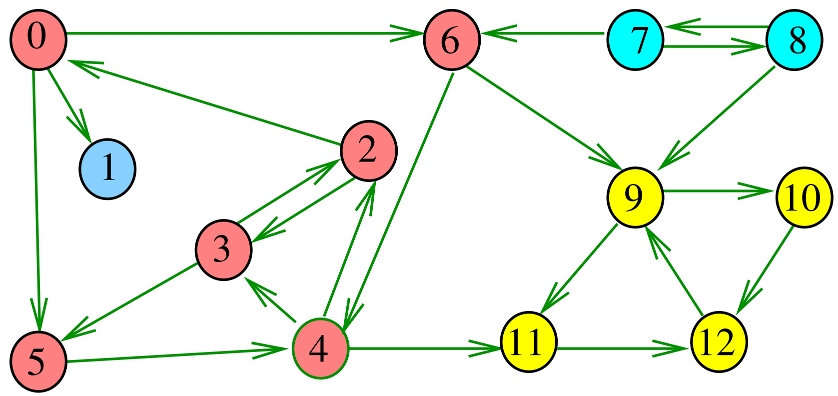

Exemplo: 4 componentes fortemente conexos



# Determinando componentes f.c.

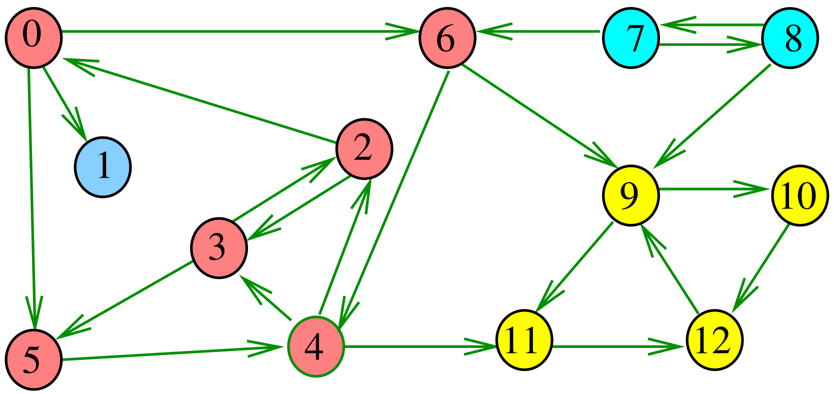
Problema: determinar os componentes fortemente conexos

Exemplo: 4 componentes fortemente conexos



## Exemplo

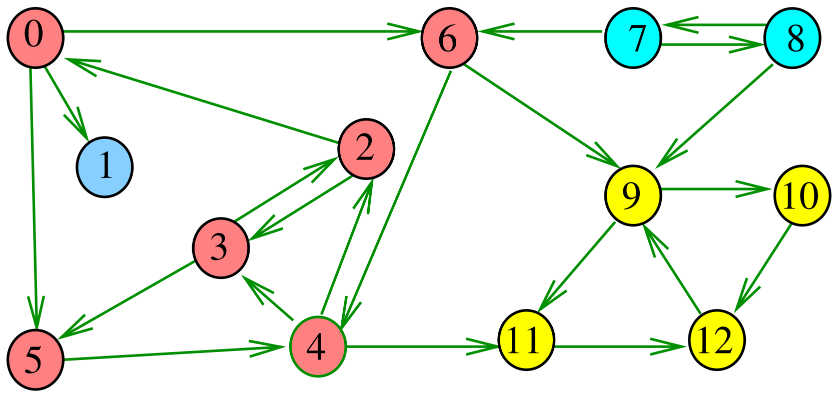
| v | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sc[v] | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 0 | 0 |






## 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);
  for (v = 0; v < G->V; v++)
      for (v = w+1; v < G->V; v++)
          if (DIGRAPHpath(G,v,w)==1
              && DIGRAPHpath(G,w,v)==1)
              GRAPHinsertE(H,v,w);
  n = GRAPHcc(H);
  for (v = 0; v < G->V; v++) sc[v]=cc[v];
  return n;
}
```

## Consumo de tempo

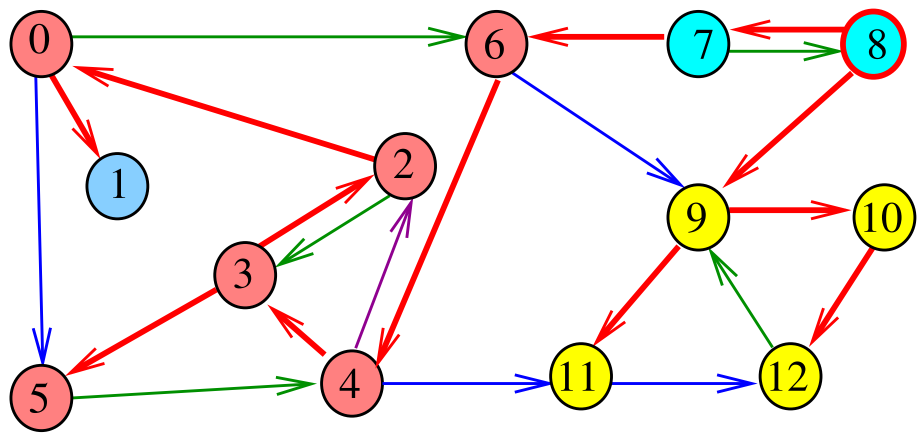
O consumo de tempo da função `DIGRAPHsc1` para vetor de listas de adjacência é $O(V^2(V + A))$.

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



## Propriedade

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

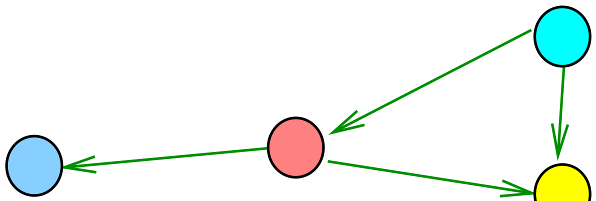





## Digrafos dos componentes

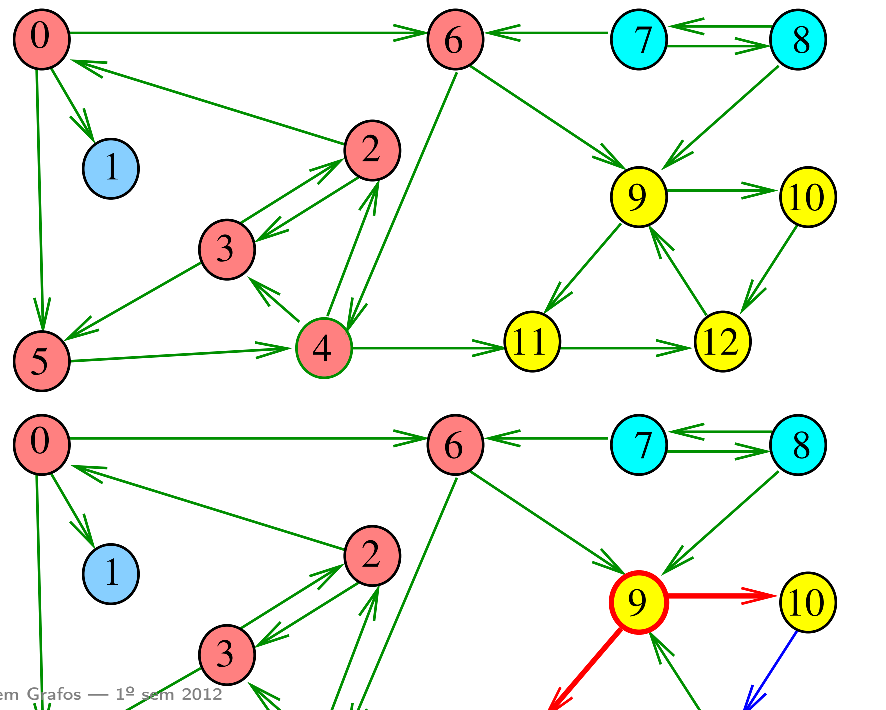
O **digrafo dos componentes** de G tem um vértice para cada componente fortemente conexo e um arco U-W se G possui um arco com ponta inicial em U e ponta final em W

Digrafo dos componente é um DAG





## Idéia ... G e DFS

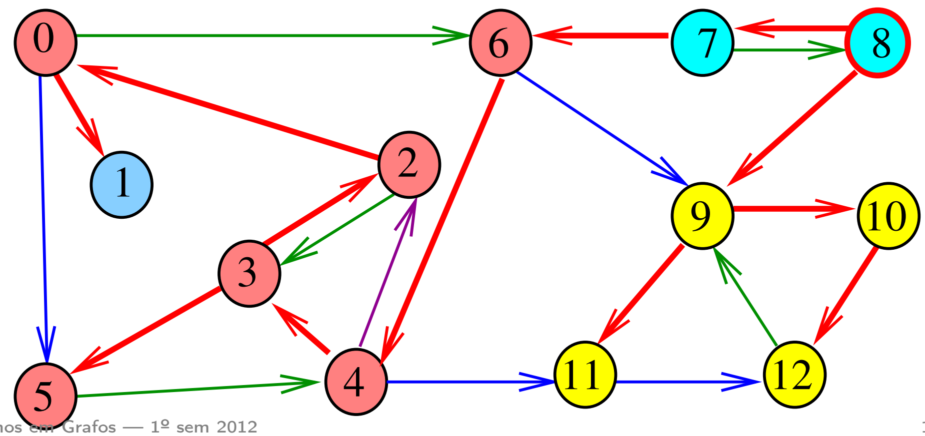


# Numeração pós-ordem

pos[v] = numeração pós-ordem de v

sop[i] = vértice de numeração pós-ordem i

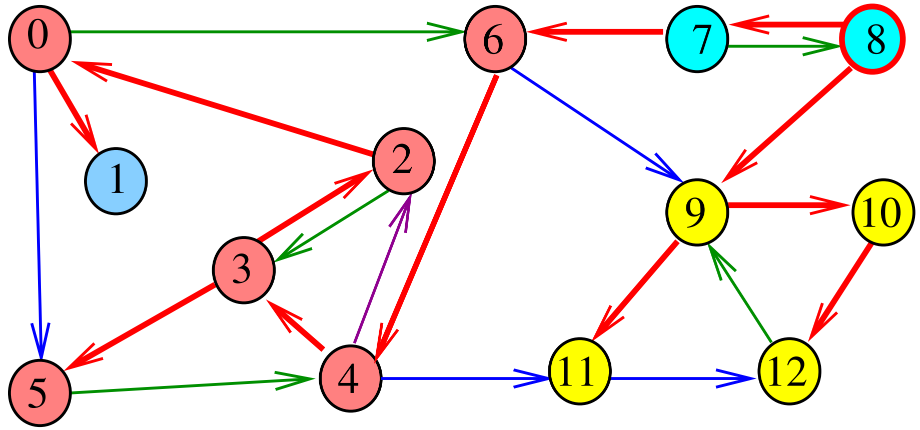
pos[W] = maior numeração pós-ordem de um vértice em W






# Exemplo

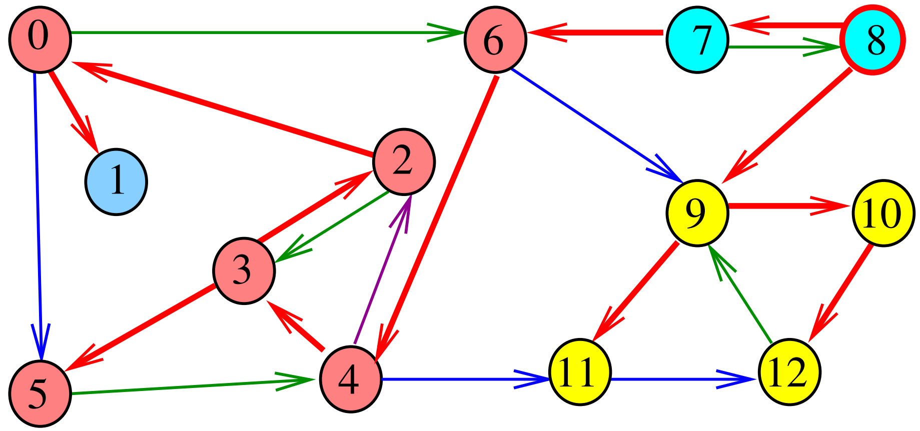
| v | 0 | **1** | 2 | 3 | 4 | 5 | **6** | 7 | **8** | **9** | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pos[v] | 6 | 5 | 7 | 8 | 9 | 4 | 10 | 11 | 12 | 3 | 1 | 2 | 0 |






# Exemplo

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





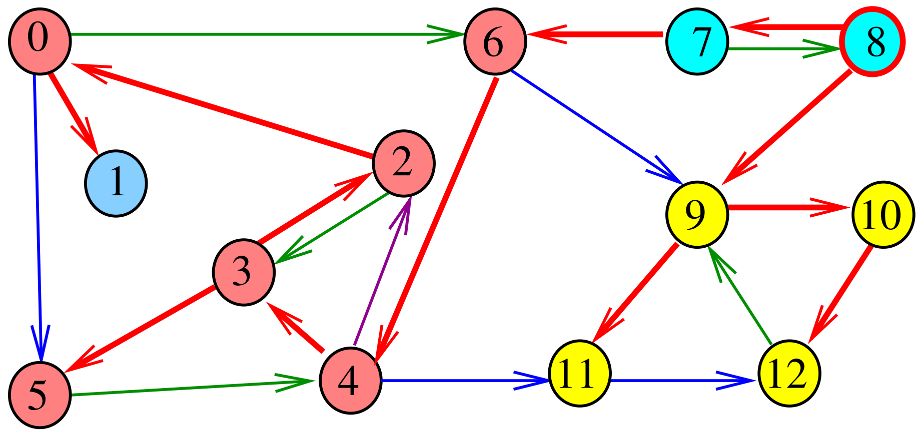

# Exemplo

pos[{7, **8**}] = 12

pos[{0, 2, 3, 4, 5, **6**}] = 10

pos[{**1**}] = 5

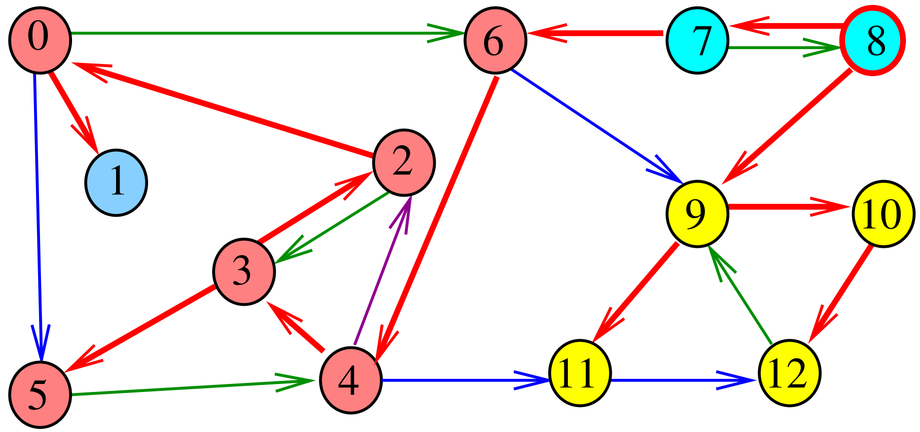
pos[{**9**, 10, 11, 12}] = 3



## 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

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

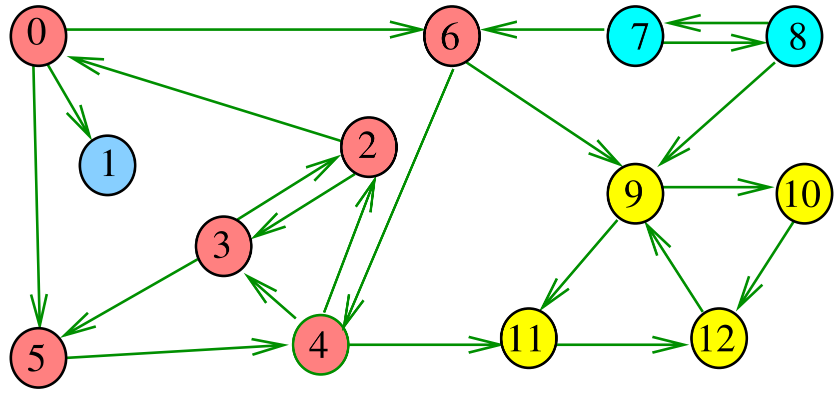




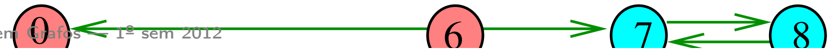
## Propriedade

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

Exemplo: Digrafo G



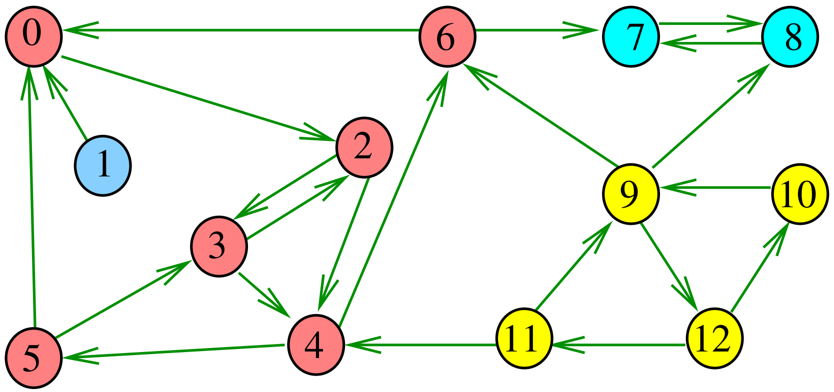
Exemplo: Digrafo reverso R de G





## Digrafo reverso R

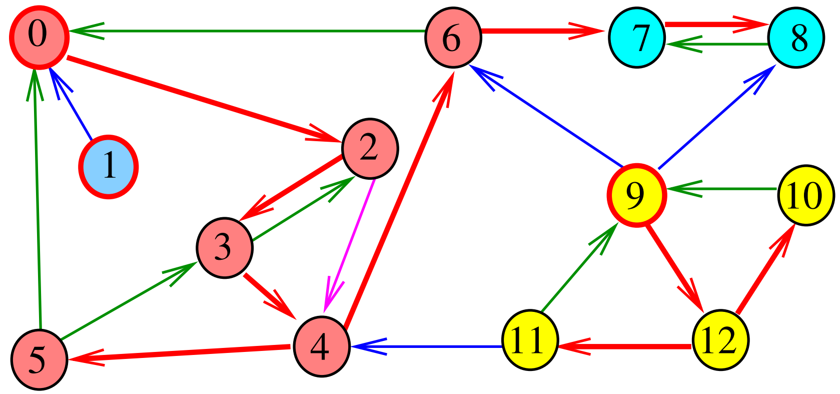
| v | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sc[v] | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 0 | 0 |





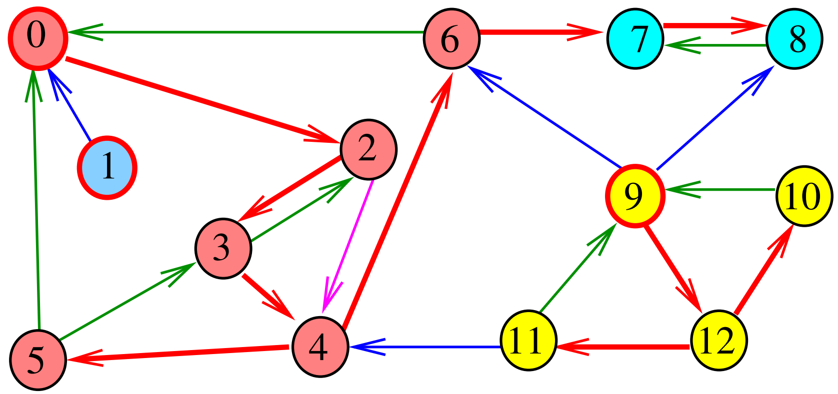
## Digrafo reverso R e DFS

| 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

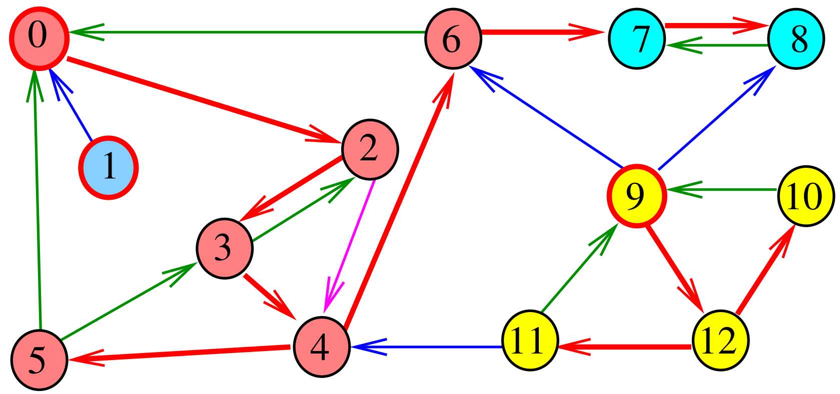
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sop[i] | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 | 1 | 10 | 11 | 12 | 9 |






# Digrafo reverso R e DFS

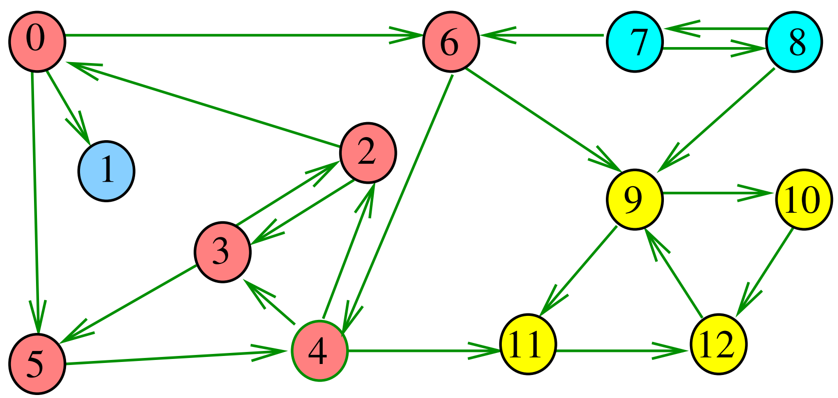
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sop[i] | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 | 1 | 10 | 11 | 12 | 9 |






# Digrafo G e DFS

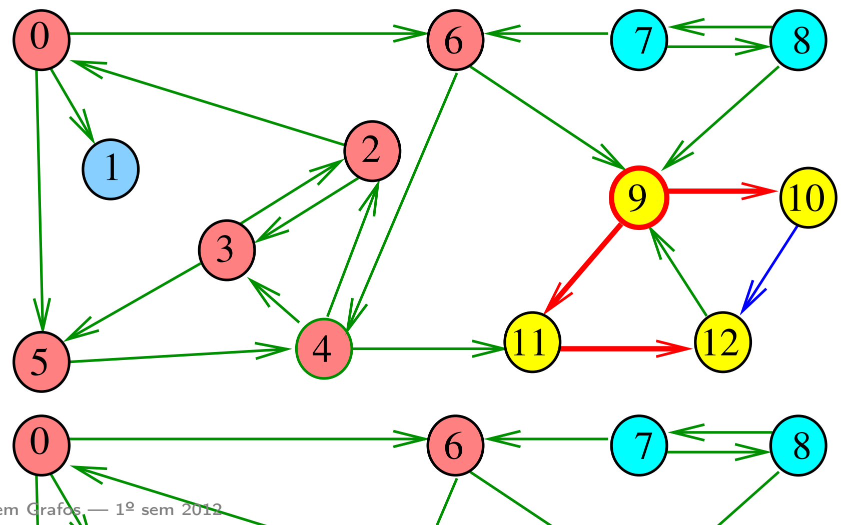
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sop[i] | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 | 1 | 10 | 11 | 12 | 9 |






# 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 |



# Algoritmo de Kosaraju

A função devolve o número de componentes fortemente conexos 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 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;
 Digraph R = DIGRAPHreverse(G);
 cnt = 0;
 for (v = 0; v < R->V; v++) sc[v] = -1;
 for (v = 0; v < R->V; v++)
     if (sc[v] == -1)
         dfsRsc(R, v, 0);
```



# DIGRAPHsc

```
 for (i = 0; i < G->V; i++)
     sopR[i] = sop[i];
 cnt = id = 0;
 for (v = 0; v < G->V; v++) sc[v] = -1;
 for (i = G->V-1; i > 0; i--)
     if (sc[sopR[i]] == -1)
         dfsRsc(G, sopR[i], id++);
 DIGRAPHdestroy(R);
 return id;
}
```



# dfsRsc

```
void dfsRsc(Digraph G,Vertex v,int id){
 link p;
 sc[v] = id;
 for (p=G->adj[v];p!=NULL;p=p->next)
     if (sc[p->w] == -1)
         dfsRsc(G, p->w, id);
 pos[v] = cnt; /* não precisa */
 sop[cnt++] = v;
}
```

# DIGRAPHreverse

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



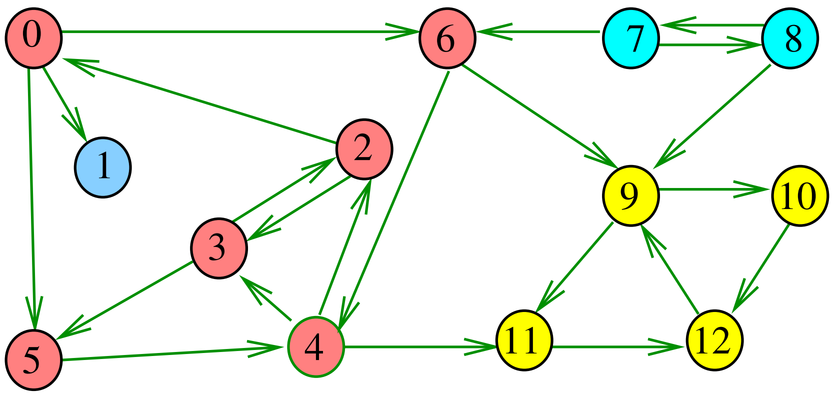
# Consumo de tempo

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



# Algoritmo de Tarjan

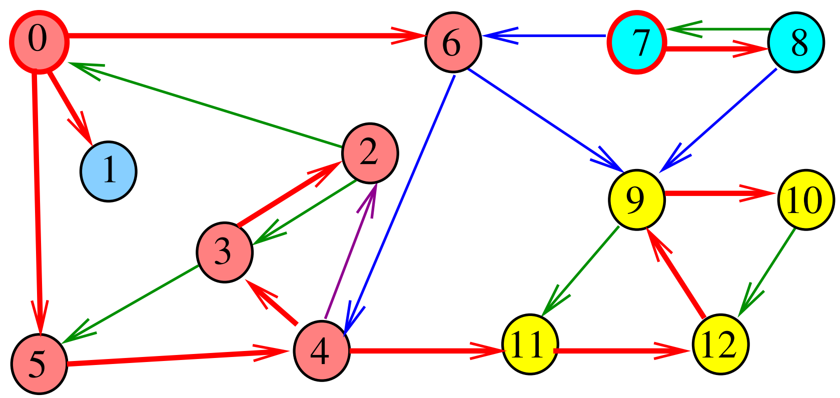
O menor **número de pré-ordem** de um vértice "ativo" que pode ser alcançado por v utilizando arcos da arborescência e **até um** arco de retorno será denotado por low[v]






# Exemplo

| v | **0** | **1** | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | **11** | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pre[v] | 0 | 9 | 4 | 3 | 2 | 1 | 10 | 11 | 12 | 7 | 8 | 5 | 6 |
| low[v] | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 11 | 11 | 5 | 6 | 5 | 5 |



## DIGRAPHsc

```
void DIGRAPHsc (Graph G) {
Vertex v;
cnt = id = t = 0;
for (v = 0; v < G->V; v++)
    pre[v] = -1;
for (v = 0; v < G->V; v++)
    if (pre[v] == -1)
        dfsRsc(G, v);
}
```



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



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