Busca DFS

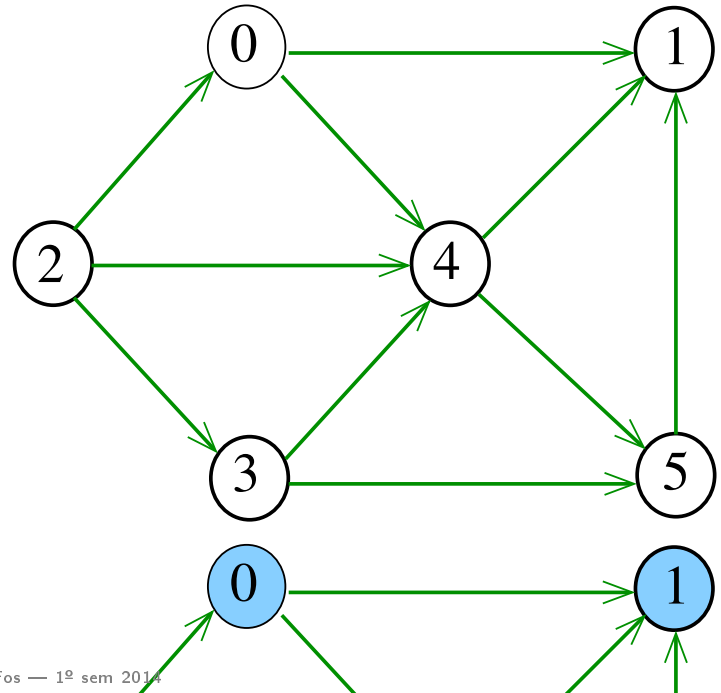
S 18.1 e 18.2

Busca ou varredura

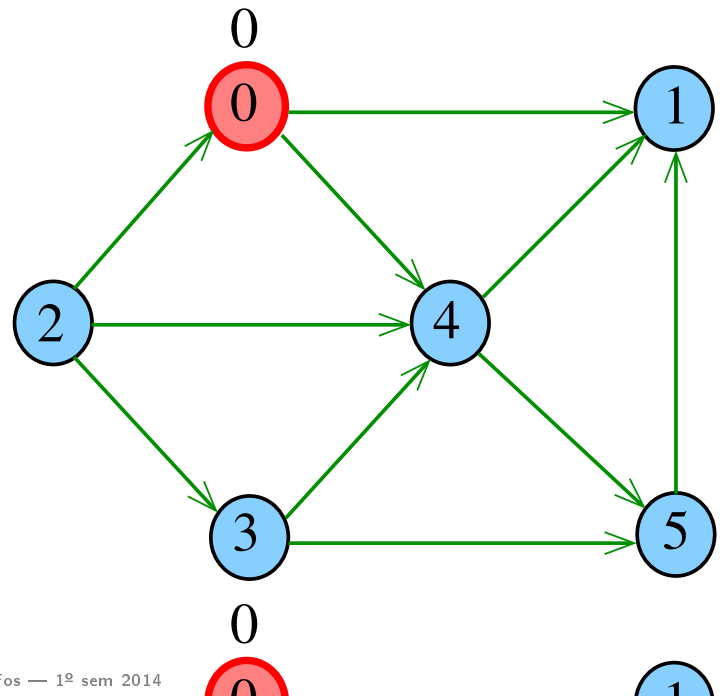
Um algoritmo de **busca** (ou **varredura**) examina, sistematicamente, todos os vértices e todos os arcos de um digrafo.

Cada arco é examinado **uma só vez**.
Despois de visitar sua ponta inicial o algoritmo percorre o arco e visita sua ponta final.

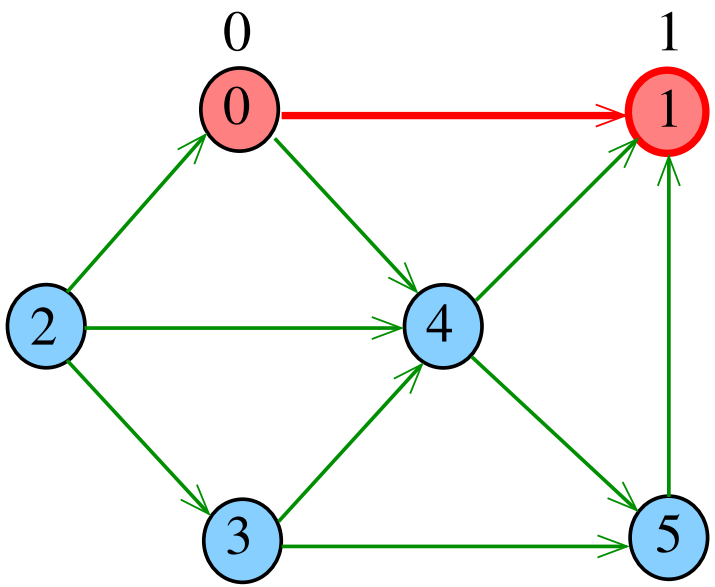
DIGRAPHdfs(G)



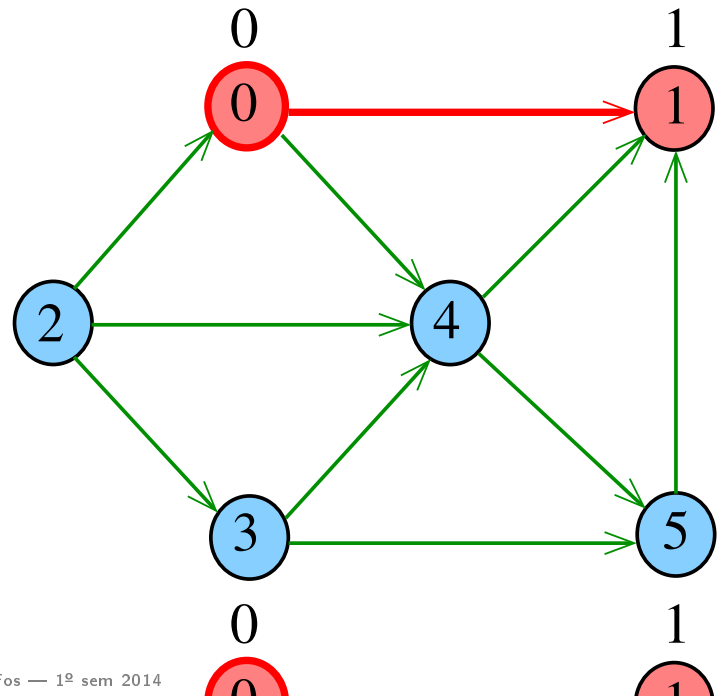
dfsR(G,0)



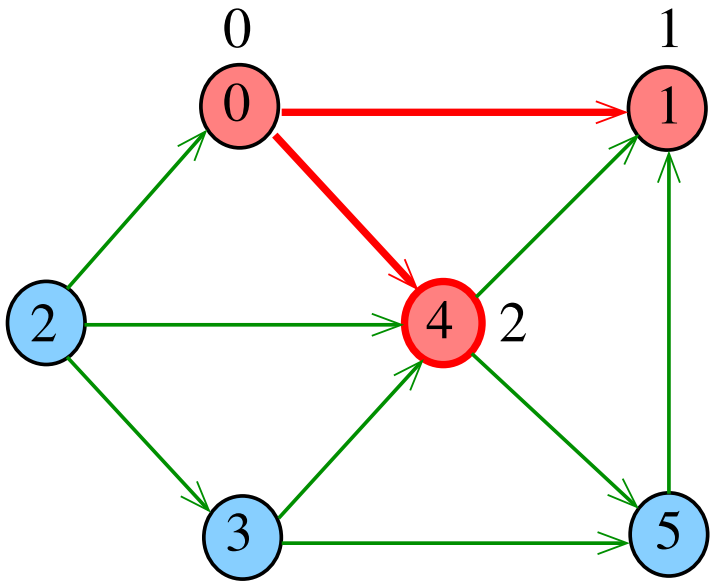
dfsR(G,1)



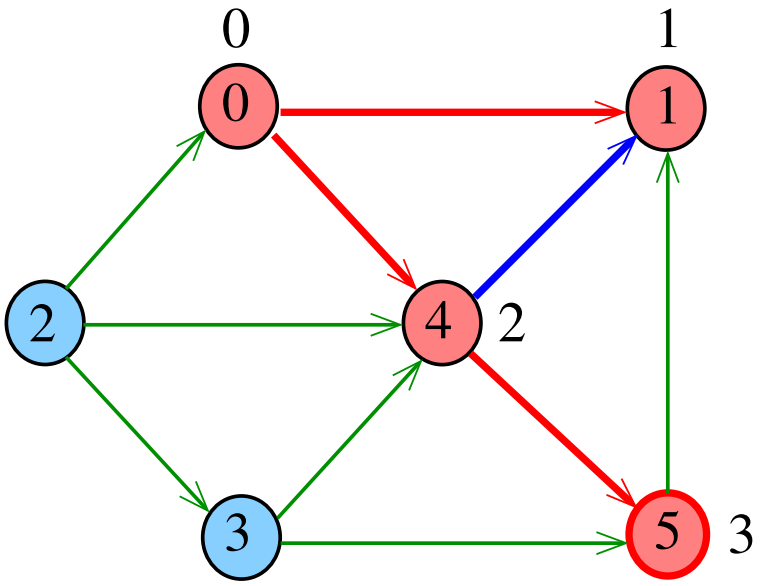
dfsR(G,0)



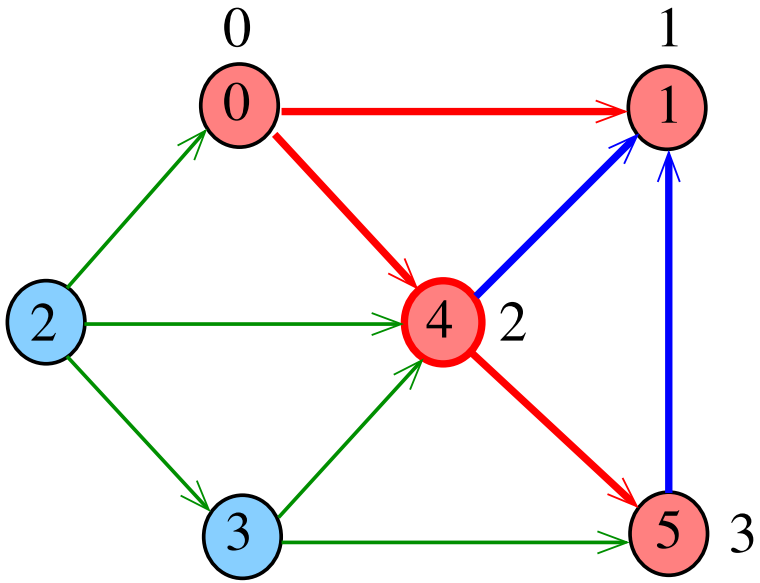
dfsR(G,4)



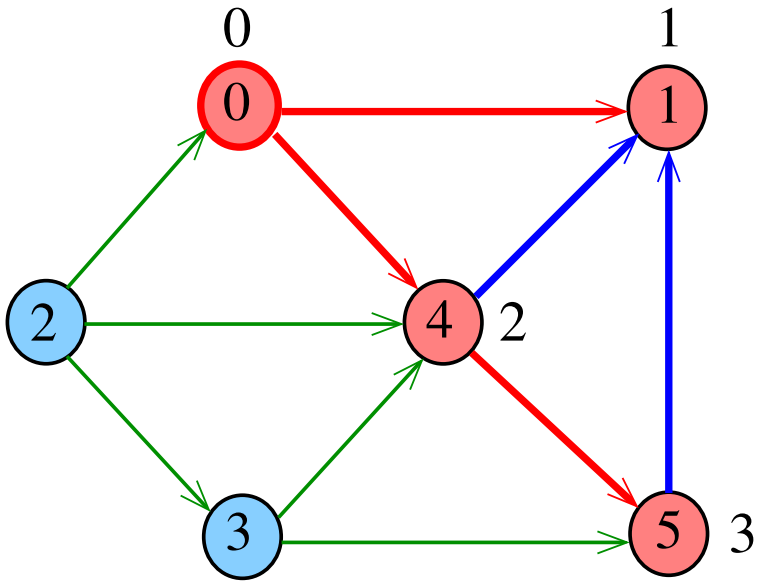
dfsR(G,5)



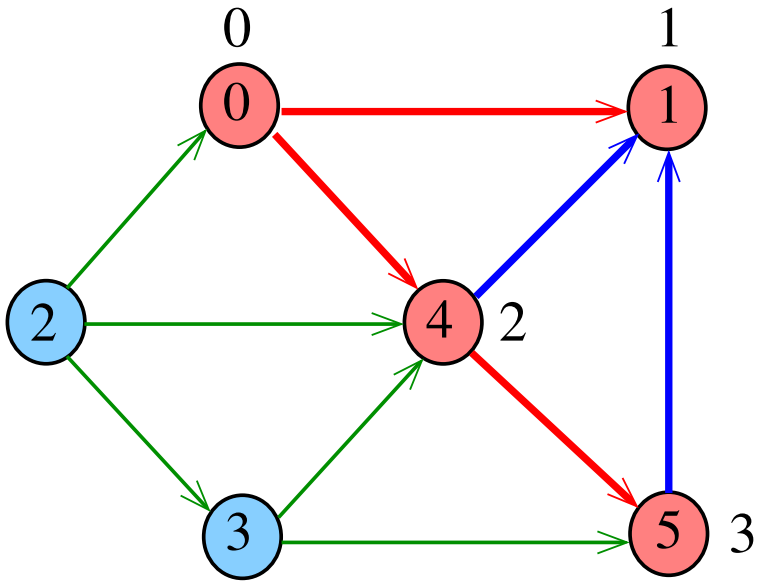
dfsR(G,4)



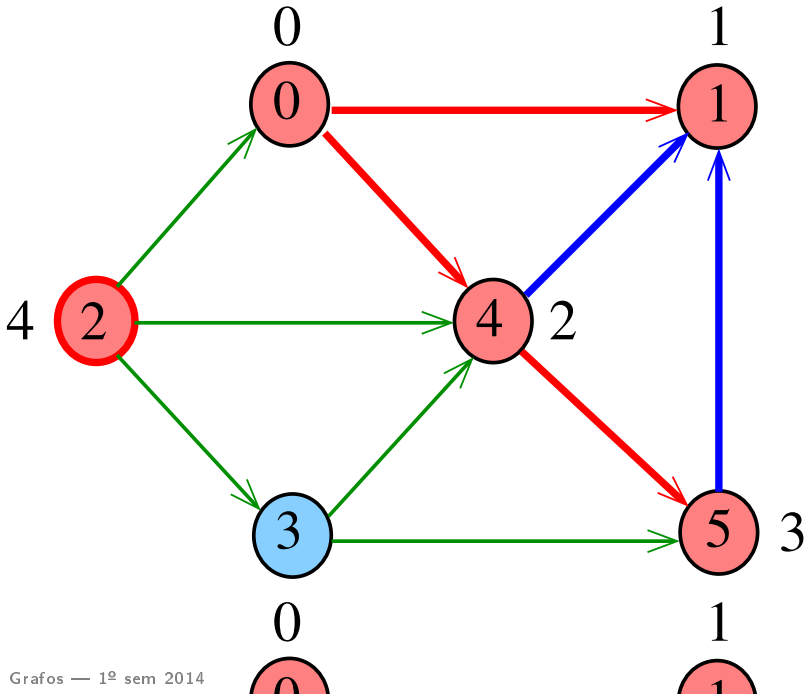
dfsR(G,0)



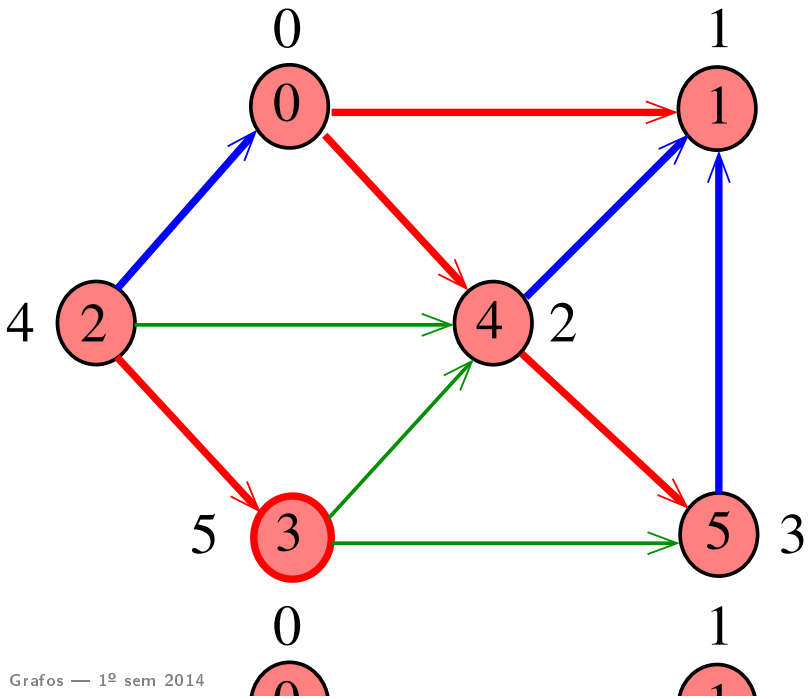
DIGRAPHdfs(G)



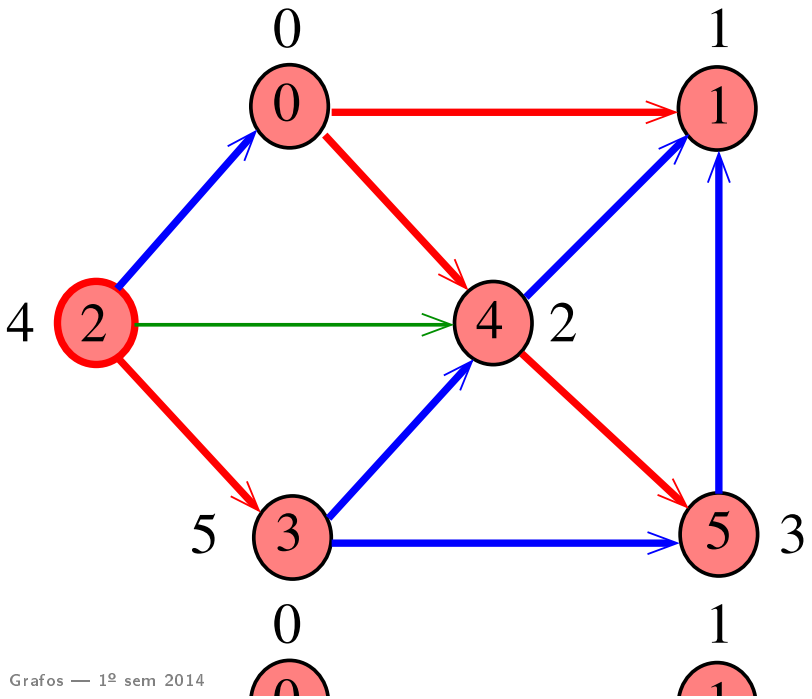
dfsR(G,2)



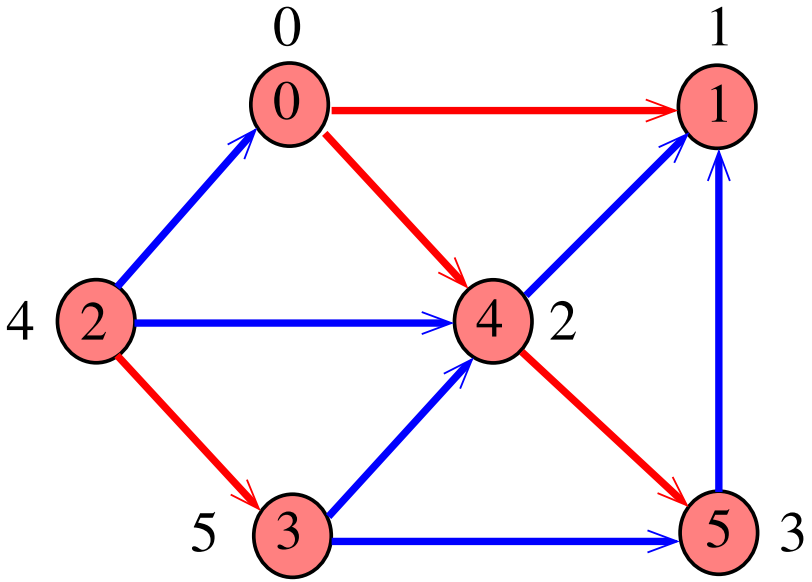
dfsR(G,3)



dfsR(G,2)



DIGRAPHdfs(G)



DIGRAPHdfs

```
static int cnt, lbl[maxV];
void DIGRAPHdfs (Digraph G) {
    Vertex v;
1   cnt = 0;
2   for (v = 0; v < G->V; v++)
3       lbl[v] = -1;
4   for (v= 0; v < G->V; v++)
5       if (lbl[v] == -1)
6           dfsR(G, v);
}
```

dfsR

dfsR supõe que o digrafo G é representado por uma matriz de adjacência

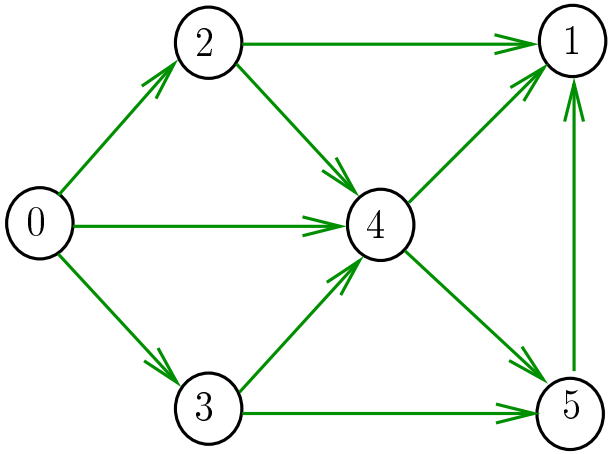
```
void dfsR (DigraphG, Vertex v) {
    Vertex w;
1   lbl[v] = cnt++;
2   for (w = 0; w < G->V; w++)
3       if (G->adj[v][w] != 0)
4           if (lbl[w] == -1)
5               dfsR(G, w);
}
```


dfsR

dfsR supõe que o digrafo G é representado por listas de adjacência

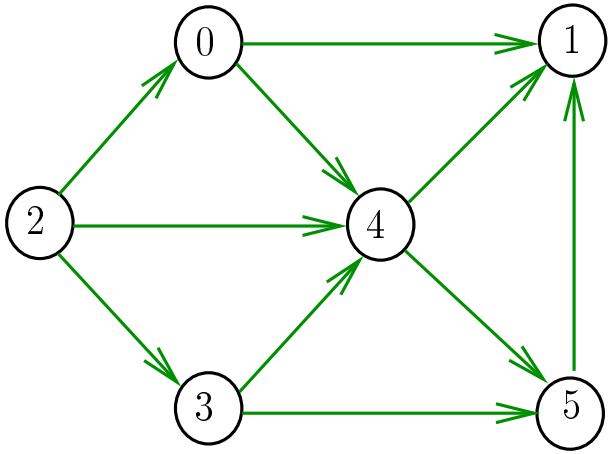
```
void dfsR (Digraph G, Vertex v) {
    link p;
1   lbl[v] = cnt++;
2   for (p = G->adj[v]; p != NULL; p= p->next)
3       if (lbl[p->w] == -1)
4           dfsR(G, p->w);
}
```


DIGRAPHdfs(G)




```
dfsR(G, 0)
  0-2 dfsR(G,2)
    2-1 dfsR(G,1)
    2-4 dfsR(G,4)
      4-1
      4-5 dfsR(G,5)
        5-1
  0-3 dfsR(G,3)
    3-4
    3-5
  0-4
```


DIGRAPHdfs(G)




```
dfsR(G, 0)
  0-1 dfsR(G, 1)
  0-4 dfsR(G, 4)
    4-1
    4-5 dfsR(G, 5)
      5-1
dfsR(G, 2)
  2-0
  2-3 dfsR(G, 3)
    3-4
    3-5
  2-4
```

Consumo de tempo

O consumo de tempo da função DIGRAPHdfs para vetor de listas de adjacência é $\Theta(V + A)$.

O consumo de tempo da função DIGRAPHdfs para matriz de adjacência é $\Theta(V^2)$.

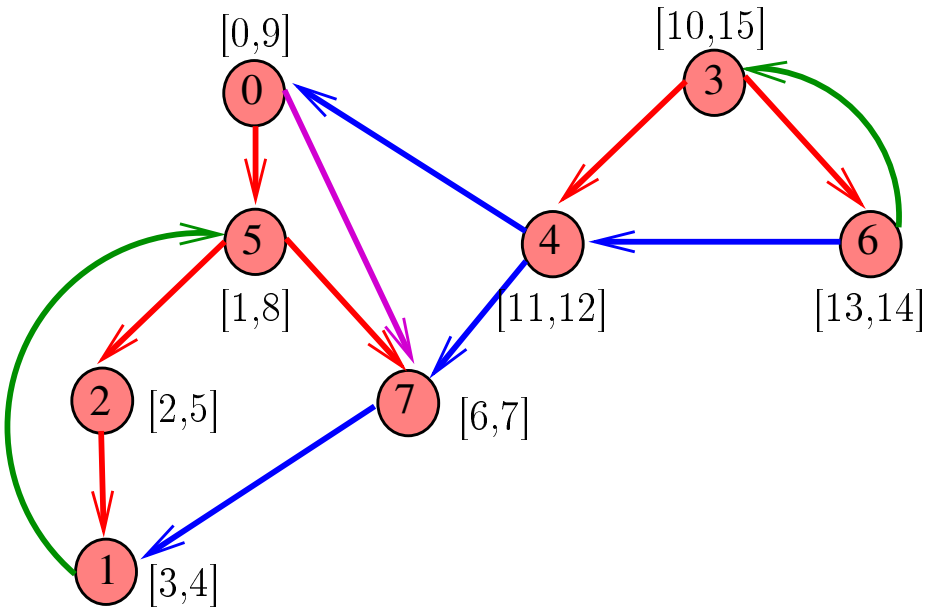
Busca DFS (CLRS)

Vamos supor que nossos digrafos têm no máximo maxV vértices

```
#define maxV 10000
static int time, parnt[maxV], d[maxV],
f[maxV];
```

DIGRAPHdfs visita todos os vértices e arcos do digrafo G.
A função registra em d[v] o 'momento' em que v foi descoberto e em f[v] o momento em que ele foi completamente examinado

Busca DFS (CLRS)



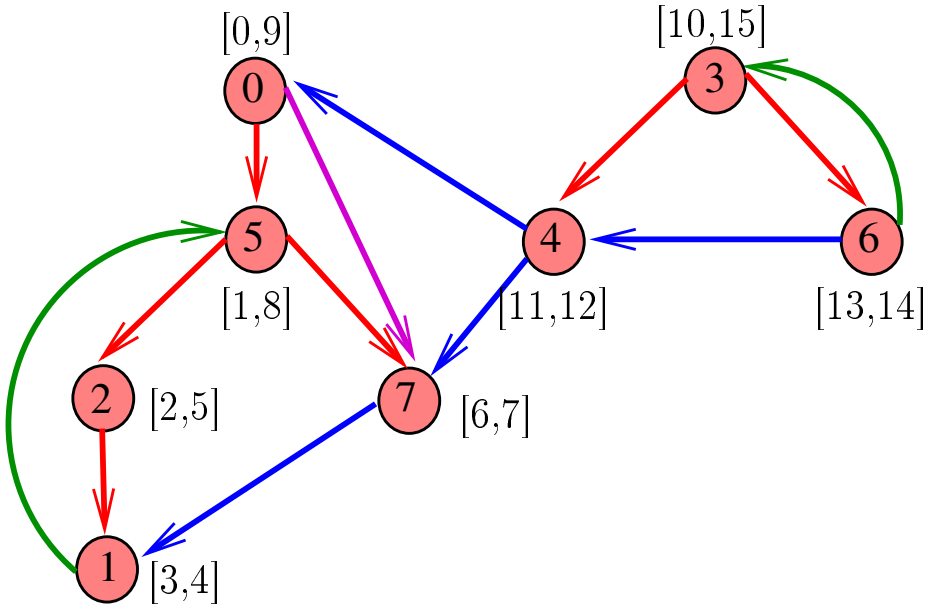
DIGRAPHdfs

```
void DIGRAPHdfs (Digraph G) {
    Vertex v;
1   time = 0;
2   for (v = 0; v < G->V; v++)
3       d[v] = f[v] = parnt[v] = -1;
4   for (v= 0; v < G->V; v++)
5       if (d[v] == -1)
6           dfsR(G, v);
}
```

dfsR

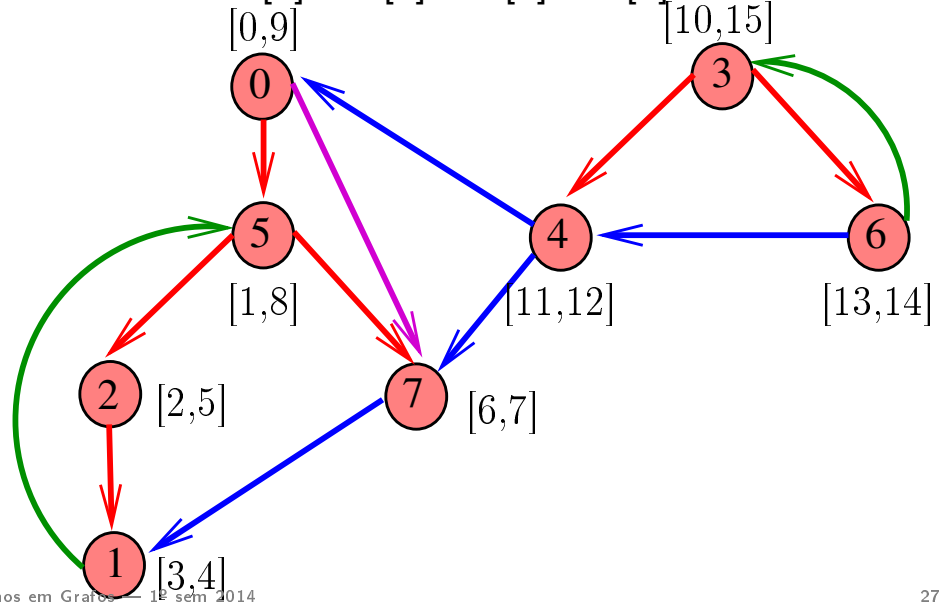
```
void dfsR (Digraph G, Vertex v) {
    link p;
1   d[v] = time++;
2   for (p = G->adj[v]; p; p= p->next)
3       if (d[p->w] == -1) {
4           parnt[p->w] = v;
5           dfsR(G, p->w);
6       }
7   f[v] = time++;
}
```


Classificação dos arcos



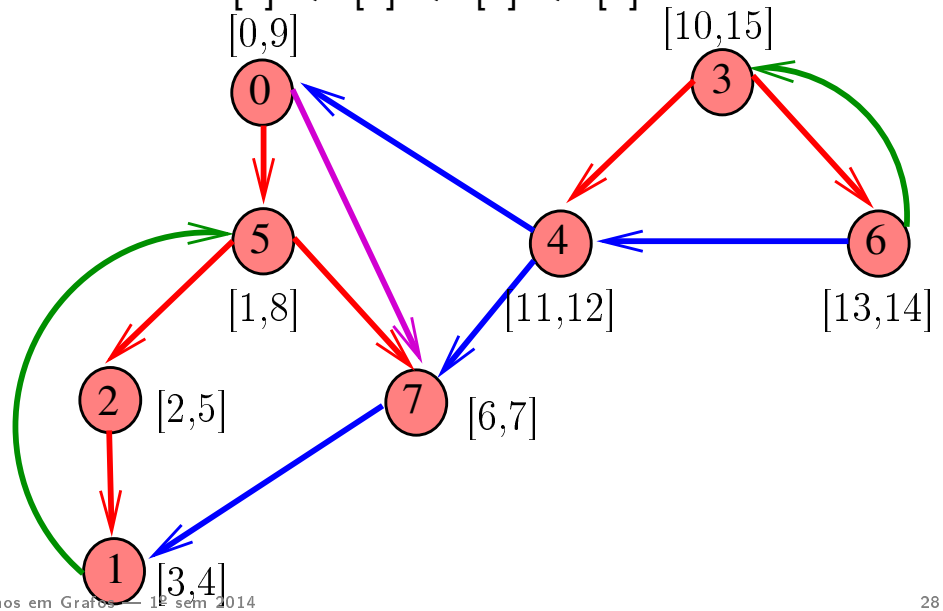
Arcos de arborescência ou descendentes

v-w é **arco de arborescência** ou **descendente** se e somente se $d[v] < d[w] < f[w] < f[v]$



Arcos de retorno

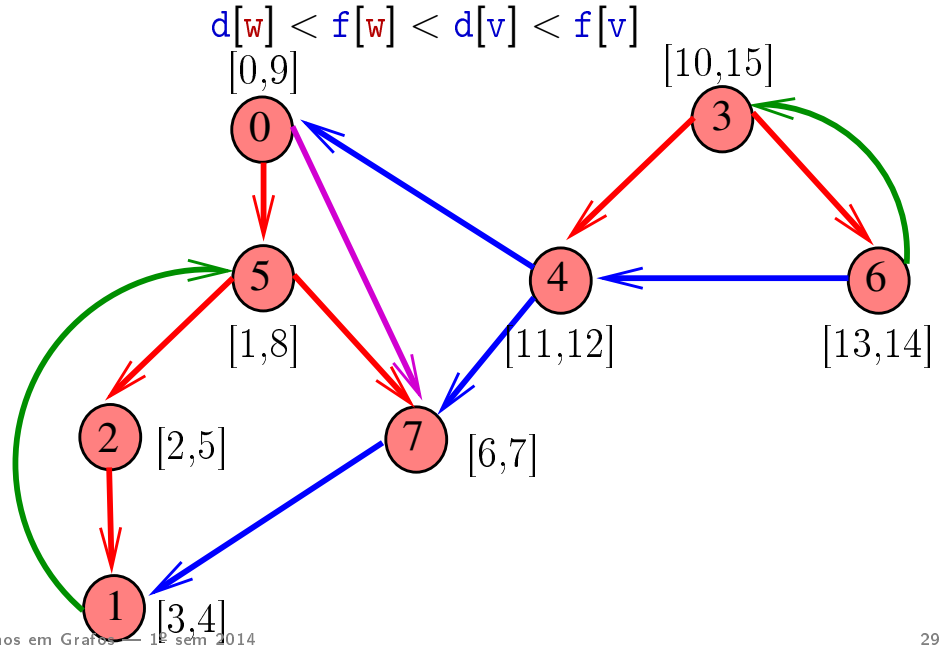
v-w é **arco de retorno** se e somente se $d[w] < d[v] < f[v] < f[w]$



Arcos cruzados

v-w é arco **cruzado** se e somente se

$d[w] < f[w] < d[v] < f[v]$



Conclusões

v-w é:

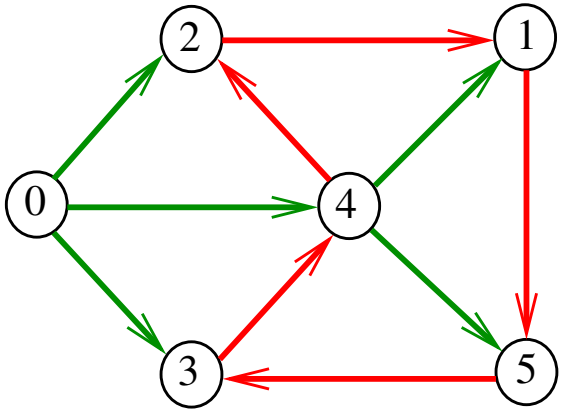
- **arco de arborescência** se e somente se
  $d[v] < d[w] < f[w] < f[v]$ e $\texttt{parnt}[w] = v$;
- **arco descendente** se e somente se
  $d[v] < d[w] < f[w] < f[v]$ e $\texttt{parnt}[w] \neq v$;
- **arco de retorno** se e somente se
  $d[w] < d[v] < f[v] < f[w]$;
- **arco cruzado** se e somente se
  $d[w] < f[w] < d[v] < f[v]$;

Ciclos em digrafos

Ciclos

Um **ciclo** num digrafo é qualquer seqüência da forma v_0–v_1–v_2–...–v_{k-1}–v_p, onde v_{k-1}-v_k é um arco para $k = 1, \ldots, p$ e $v_0 = v_p$.

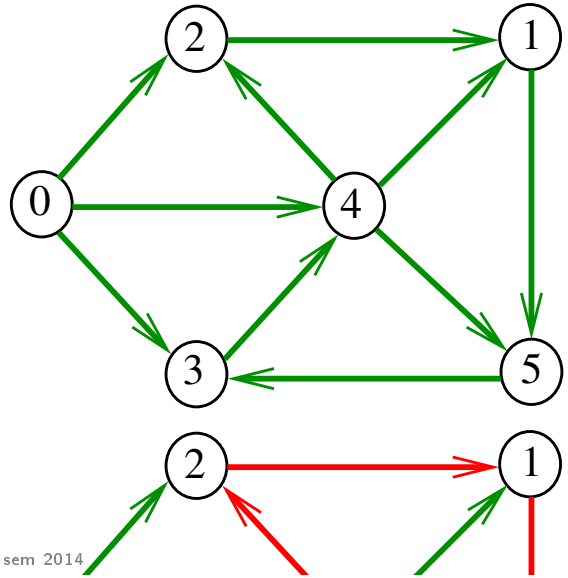
Exemplo: 2-1-5-3-4-2 é um ciclo



Procurando um ciclo

Problema: decidir se dado digrafo G possui um ciclo

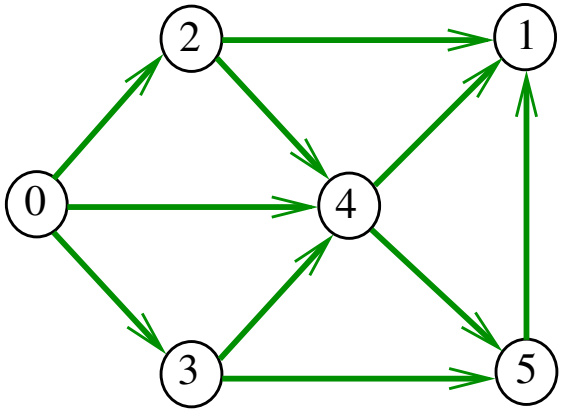
Exemplo: para o grafo a seguir a resposta é **SIM**



Procurando um ciclo

Problema: decidir se dado digrafo G possui um ciclo

Exemplo: para o grafo a seguir a resposta é **NÃO**



DIGRAPHcycle1

Recebe um digrafo G e devolve **1** se existe um ciclo em G e devolve **0** em caso contrário
Supõe que o digrafo tem no máximo maxV vértices.

```
int DIGRAPHcycle1 (Digraph G);
```


Primeiro algoritmo

```
int DIGRAPHcycle1 (Digraph G) {
    Vertex v;
    link p;
1   for (v = 0; v < G->V; v++)
2       for (p=G->adj[v]; p ; p=p->next)
3           if (DIGRAPHpath(G, p->w, v))
4               return 1;
5   return 0;
}
```

Consumo de tempo

O consumo de tempo da função DIGRAPHcycle1 é A vezes o consumo de tempo da função DIGRAPHpath.

O consumo de tempo da função DIGRAPHcycle1 para vetor de listas de adjacência é $O(A(V + A))$.

O consumo de tempo da função DIGRAPHcicle1 para matriz de adjacência é $O(AV^2)$.

DIGRAPHcycle

Vamos supor que nossos digrafos têm no máximo maxV vértices

Exercício: Mantendo a ideia de um vetor de listas, modificar o código para permitir um número ilimitado de vértices (dentro da disponibilidade da memória)

```
#define maxV 10000
static int time, d[maxV], f[maxV];
static Vertex parnt[maxV];
```


DIGRAPHcycle

Recebe um digrafo G e devolve **1** se existe um ciclo em G e devolve **0** em caso contrário

int DIGRAPHcycle (Digraph G);

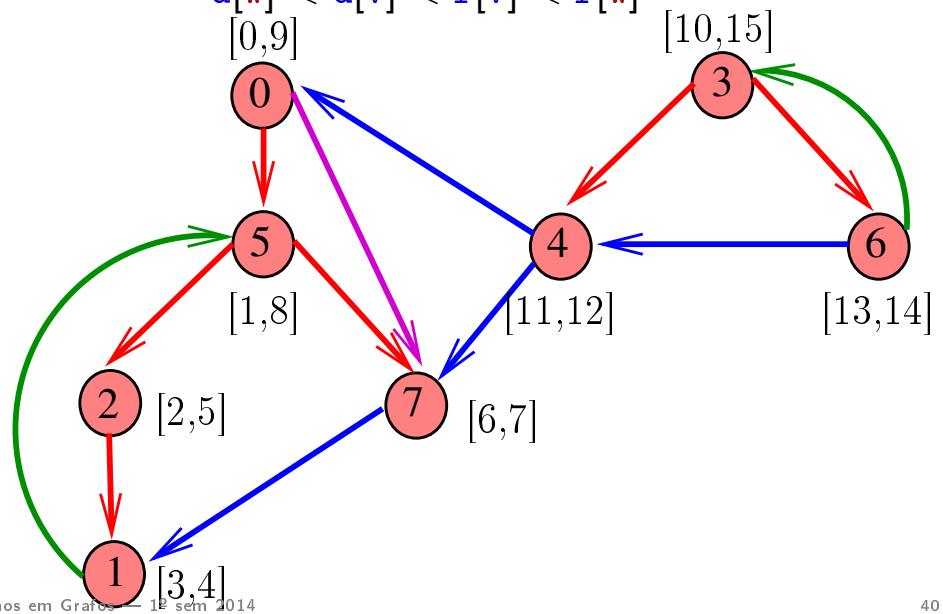
A função tem por base a seguinte observação: em relação a **qualquer** floresta de busca em profundidade,

todo *arco de* ***retorno*** *pertence a um ciclo e* ***todo*** *ciclo tem um arco de* ***retorno***

Arcos de retorno

v-w é **arco de retorno** se e somente se

$d[w] < d[v] < f[v] < f[w]$



DIGRAPHcycle

```
int DIGRAPHcycle (Digraph G) {
    Vertex v;
1   time = 0;
2   for (v = 0; v < G->V; v++)
3       d[v] = f[v] = parnt[v] = -1;
4   for (v= 0; v < G->V; v++)
5       if (d[v] == -1) {
6           parnt[v] = v;
7           if (cycleR(G, v) == 1) return 1;
        }
8   return 0;
}
```


dfsR

```
void dfsR (Digraph G, Vertex v) {
    link p;
1   d[v] = time++;
2   for (p = G->adj[v]; p; p= p->next)
3       if (d[p->w] == -1) {
4           parnt[p->w] = v;
5           dfsR(G, p->w);
        }
6
7   f[v] = time++;
8
}
```


cycleR

```
int cycleR (Digraph G, Vertex v) {
    link p;
1   d[v] = time++;
2   for (p = G->adj[v]; p; p= p->next) {
3       if (d[p->w] == -1) { /* arborescência */
4           parnt[p->w] = v;
5           if (cycleR(G,p->w)) return 1;
        } /* else: arco de retorno */
6       else if (f[p->w] == -1) return 1;
7   f[v] = time++;
8   return 0;
}
```


Consumo de tempo

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

O consumo de tempo da função DIGRAPHcycle para matriz de adjacência é $O(V^2)$.

Certificados

Como é possível verificar a resposta?

Melhorando a resposta:

devolve o arco de retorno, em vez de 1

Como é possível verificar que existe ciclo?

Usando arco de retorno e `parnt`

Como é possível verificar que não existe ciclo?

Próxima aula!

Certificado de existência

Trecho de código que verifica se o arco `v-u` junto com alguns arcos da floresta DFS formam um ciclo

Supõe que o grafo está representado através de matriz de adjacência

```
[...]
if (G->adj[v][w] == 0)
    return ERRO;
if (st_caminho(G, w, v) == 0)
    return ERRO;
[...]
```