

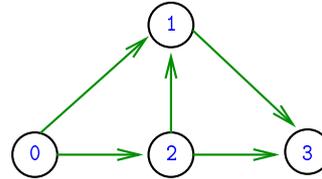
Melhores momentos

AULA 5

Vetor de listas de adjacência de digrafos

Na representação de um digrafo através de **listas de adjacência** tem-se, para cada vértice v , uma lista dos vértices que são vizinhos v .

Exemplo:



0: 1, 2
 1: 3
 2: 1, 3
 3:

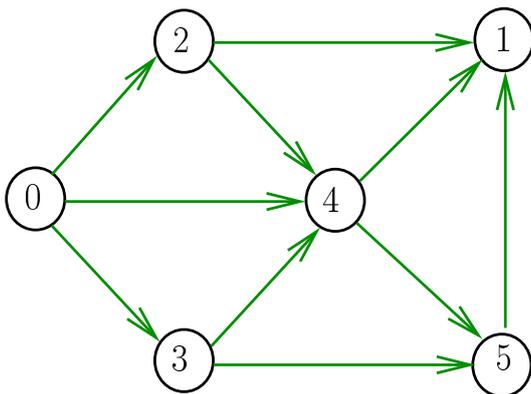
Consumo de espaço: $\Theta(V + A)$

(linear)

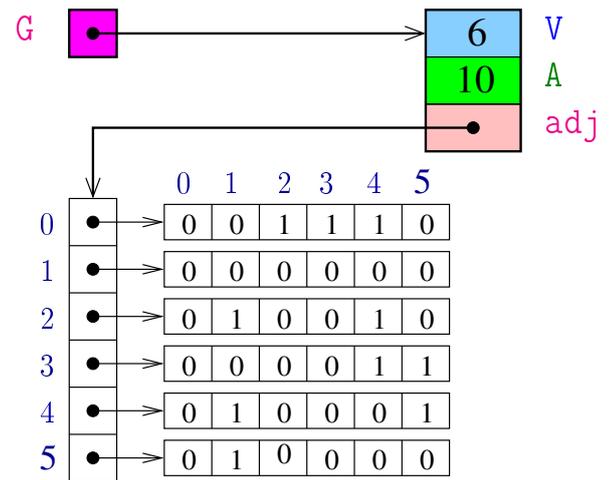
Manipulação eficiente

Digrafo

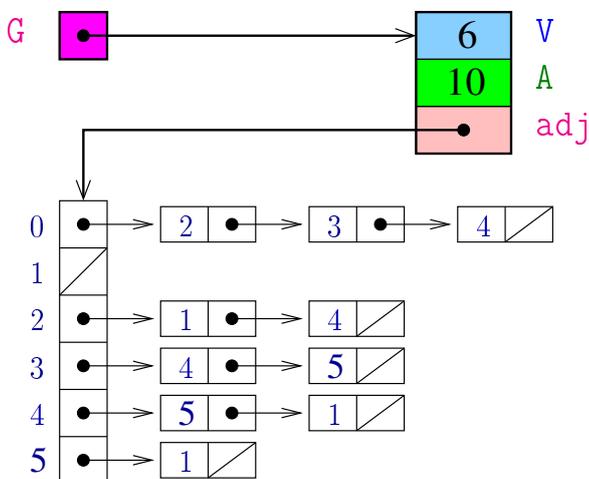
Digraph G



Matriz de adjacência



Listas de adjacência



AULA 6

Vetor de listas de adjacência (continuação)

DIGRAPHpath

S 17.4

Recebe um digrafo **G** e vértices **s** e **t** e devolve **1** se existe um caminho de **s** a **t** ou devolve **0** em caso contrário

Supõe que o digrafo tem no máximo **maxV** vértices.

```
int DIGRAPHpath (Digraph G, Vertex s, Vertex t)
```

DIGRAPHpath

```
static int lbl[maxV], static Vertex parnt[maxV];
int DIGRAPHpath (Digraph G, Vertex s, Vertex t)
{
    Vertex v;
1   for (v = 0; v < G->V; v++) {
2       lbl[v] = -1;
3       parnt[v] = -1;
4   }
5   parnt[s] = s;
6   pathR(G, s)
7   if (lbl[t] == -1) return 0;
8   else return 1;
}
```

pathR

```
void pathR (Digraph G, Vertex v)
{
    link p;
1   lbl[v] = 0;
2   for (p=G->adj[v]; p != NULL; p=p->next)
3       if (lbl[p->w] == -1) {
4           parnt[p->w] = v;
5           pathR(G, p->w);
6       }
}
```

pathR

```
void pathR (Digraph G, Vertex v)
{
    Vertex w;
1   lbl[v] = 0;
2   for (w = 0; w < G->V; w++)
3       if (G->adj[v][w] == 1)
4           if (lbl[w] == -1) {
5               parnt[w] = v;
6               pathR(G, w);
7           }
}
```

Consumo de tempo

Qual é o consumo de tempo da função

DIGRAPHpath?

Consumo de tempo

Qual é o consumo de tempo da função `DIGRAPHpath`?

linha	número de execuções da linha
1	$= V + 1 = \Theta(V)$
2	$= V = \Theta(V)$
3	$= 1 = ????$
4	$= 1 = \Theta(1)$
5	$= 1 = \Theta(1)$

$$\begin{aligned} \text{total} &= 2 \Theta(1) + 2 \Theta(V) + ??? \\ &= \Theta(V) + ??? \end{aligned}$$

Navigation icons

Consumo de tempo

Qual é o consumo de tempo da função `PathR`?

linha	número de execuções da linha
1	$\leq V = O(V)$
2	$\leq V + A = O(V + A)$
3	$\leq A = O(A)$
4	$\leq V - 1 = O(V)$
5	$\leq V - 1 = O(V)$

$$\begin{aligned} \text{total} &= 3O(V) + O(A) + O(V + A) \\ &= O(V + A) \end{aligned}$$

Navigation icons

Conclusão

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

Navigation icons

Conclusão

O consumo de tempo da função `DIGRAPHpath` é $\Theta(V)$ mais o consumo de tempo da função `PathR`.

Consumo de tempo

Qual é o consumo de tempo da função `PathR`?

Conclusão

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

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

Navigation icons

Busca DFS

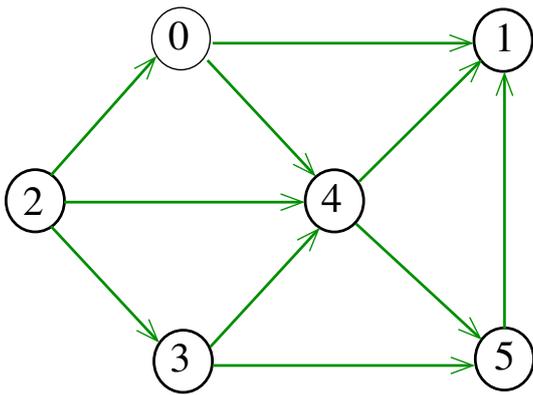
Busca ou varredura

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

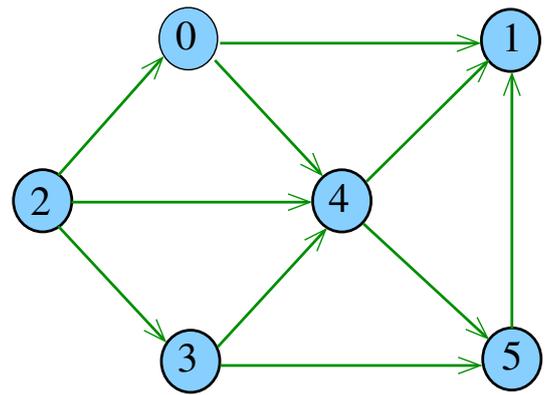
Cada arco é examinado **uma só vez**. Depois de visitar sua ponta inicial o algoritmo percorre o arco e visita sua ponta final.

S 18.1 e 18.2

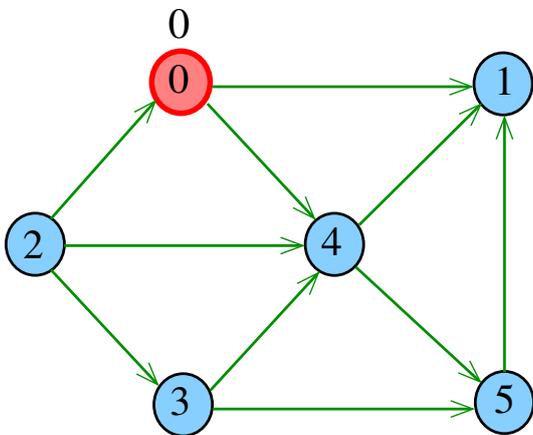
DIGRAPHdfs(G)



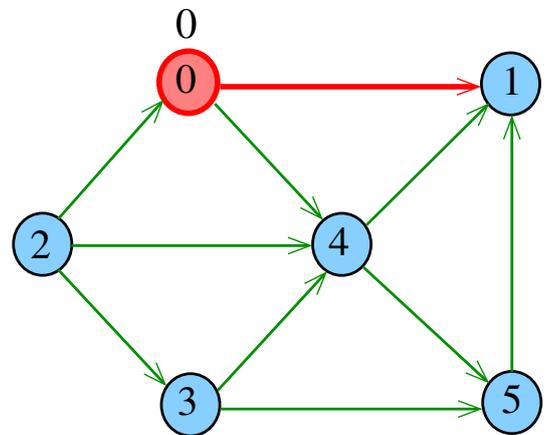
DIGRAPHdfs(G)



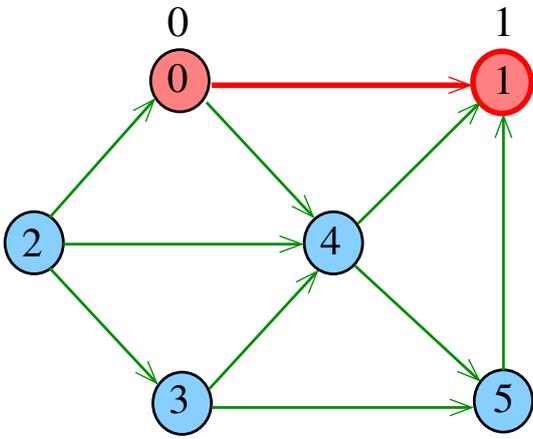
dfsR(G,0)



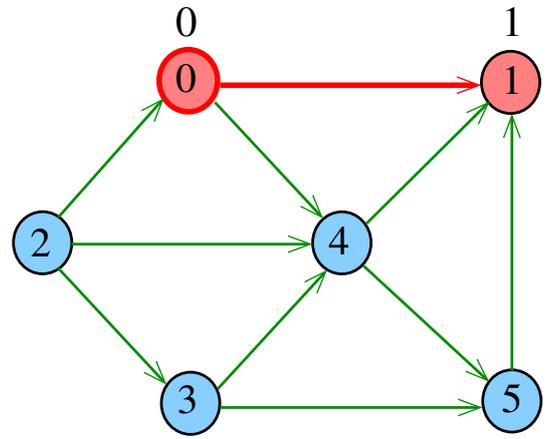
dfsR(G,0)



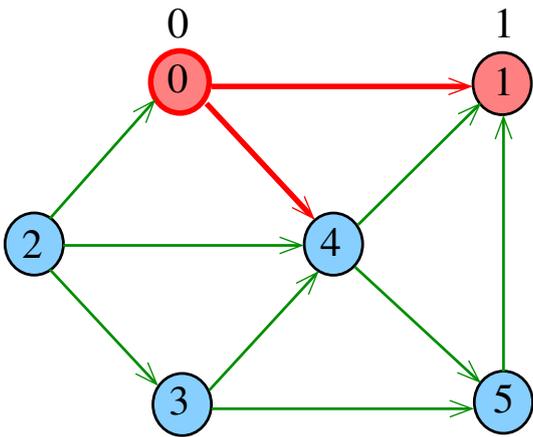
dfsR(G,1)



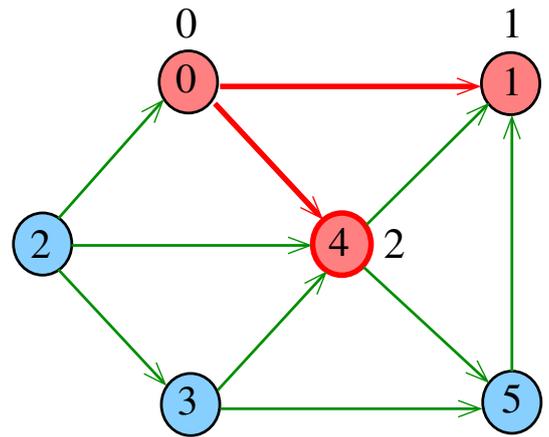
dfsR(G,0)



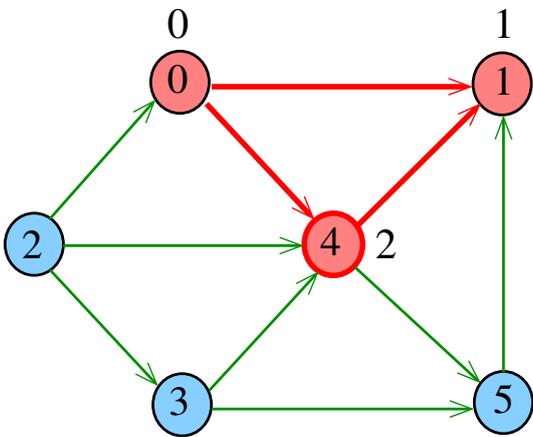
dfsR(G,0)



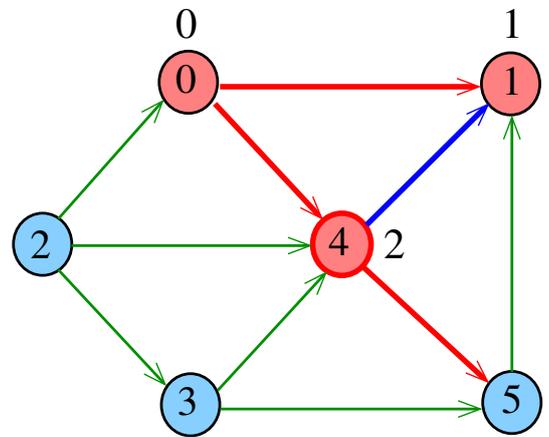
dfsR(G,4)



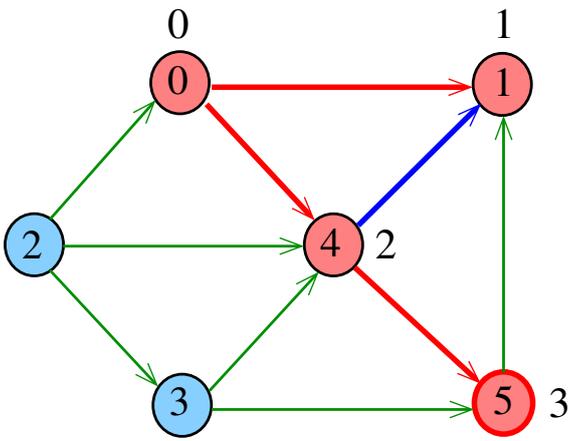
dfsR(G,4)



dfsR(G,4)

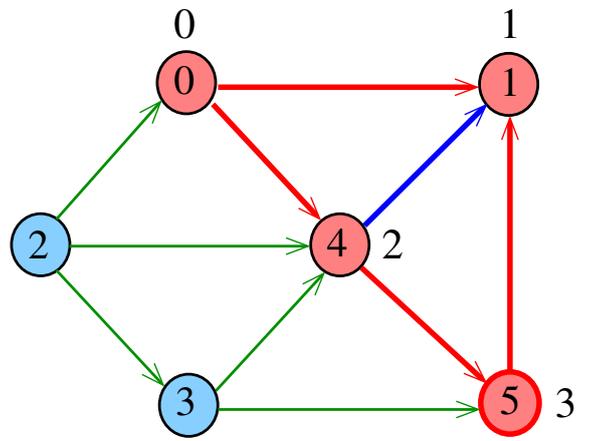


dfsR(G,5)



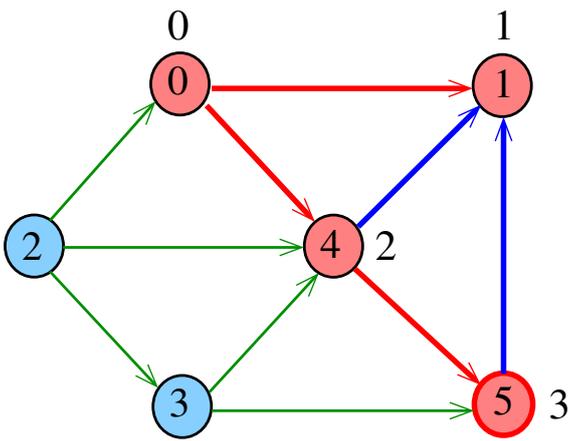
Navigation icons

dfsR(G,5)



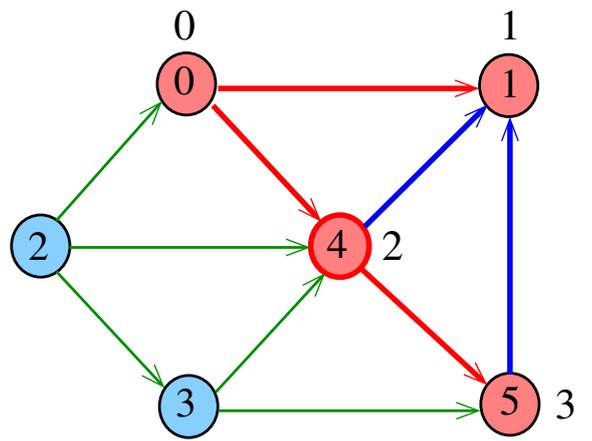
Navigation icons

dfsR(G,5)



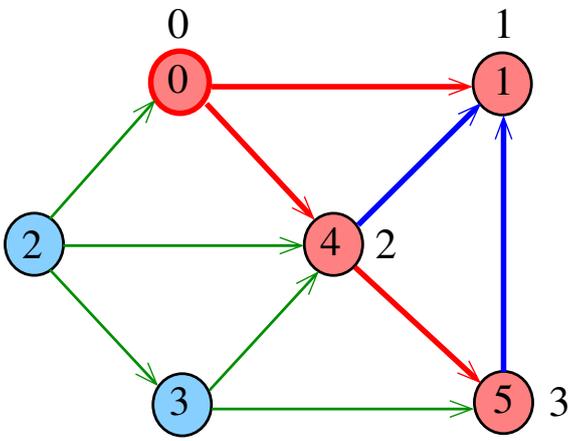
Navigation icons

dfsR(G,4)



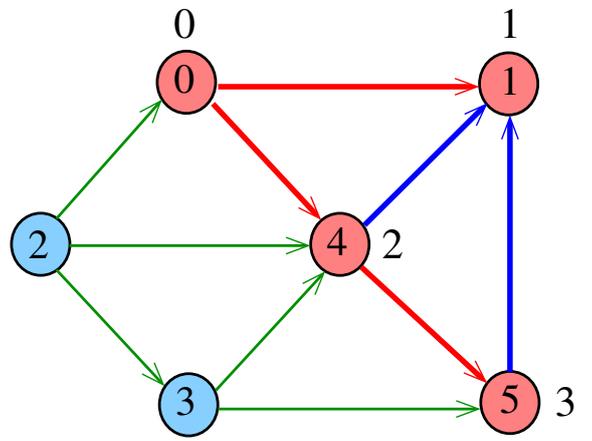
Navigation icons

dfsR(G,0)



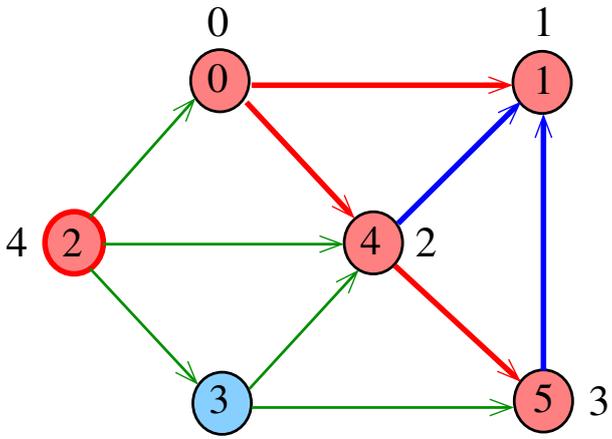
Navigation icons

DIGRAPHdfs(G)



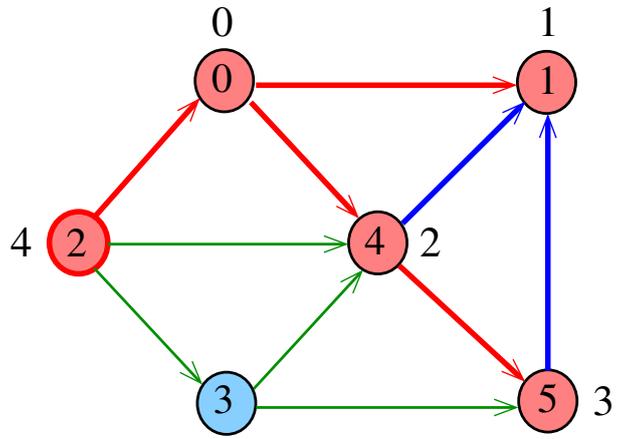
Navigation icons

dfsR(G,2)



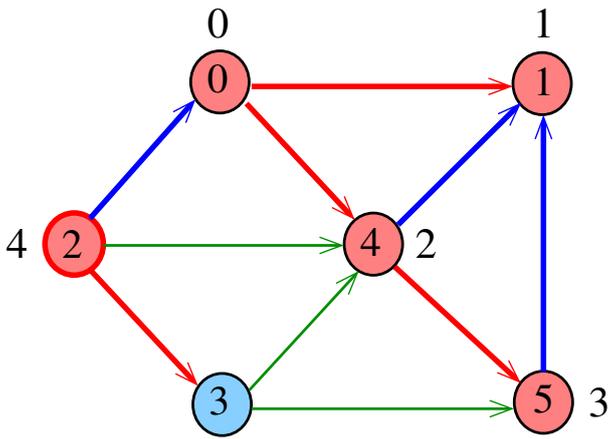
Navigation icons

dfsR(G,2)



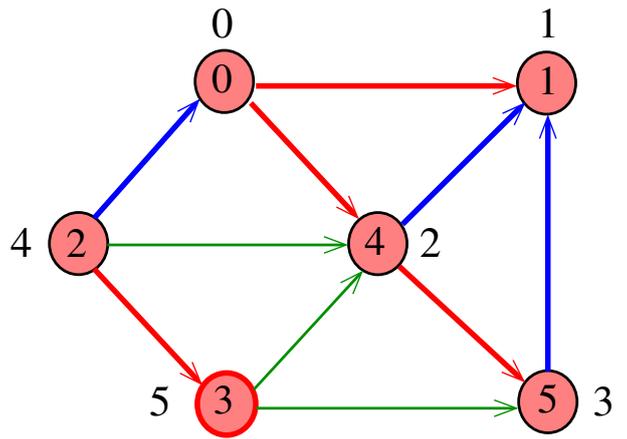
Navigation icons

dfsR(G,2)



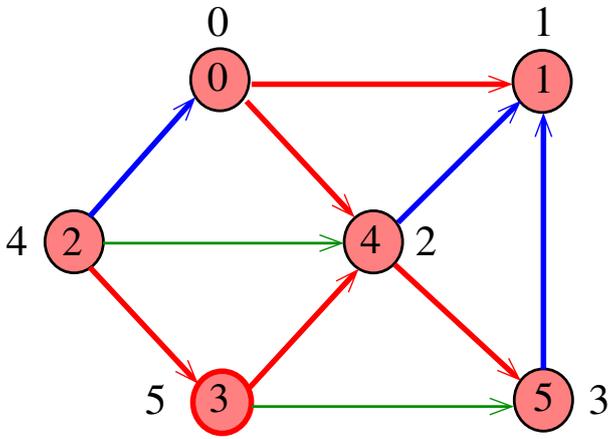
Navigation icons

dfsR(G,3)



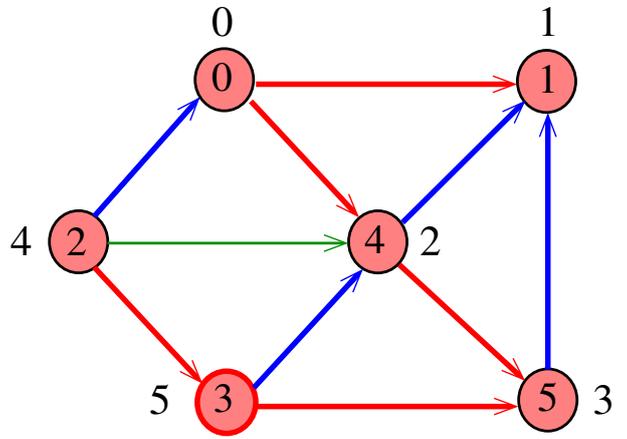
Navigation icons

dfsR(G,3)



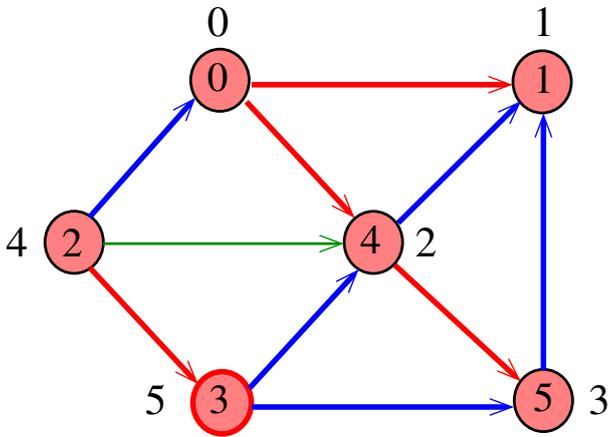
Navigation icons

dfsR(G,3)



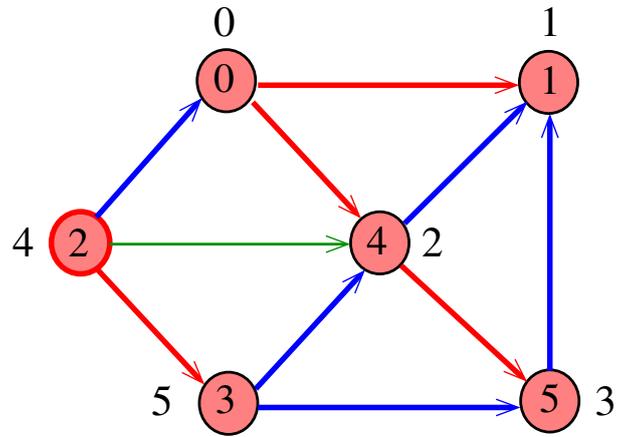
Navigation icons

dfsR(G,3)



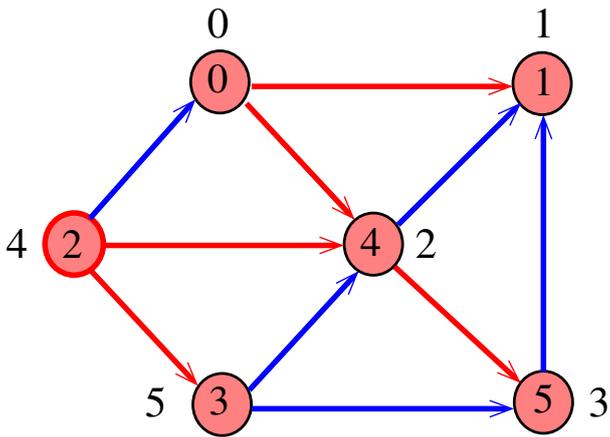
Navigation icons: back, forward, search, etc.

dfsR(G,2)



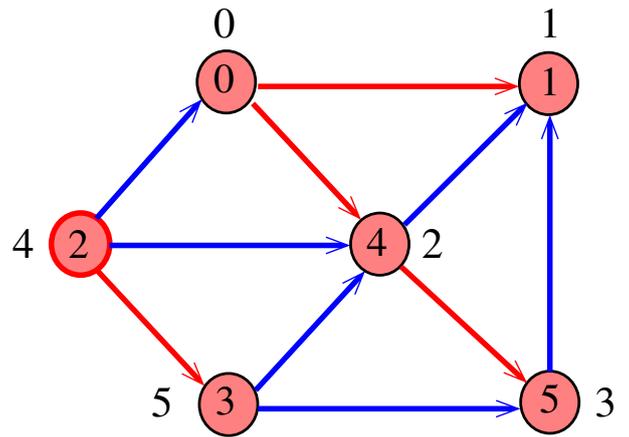
Navigation icons: back, forward, search, etc.

dfsR(G,2)



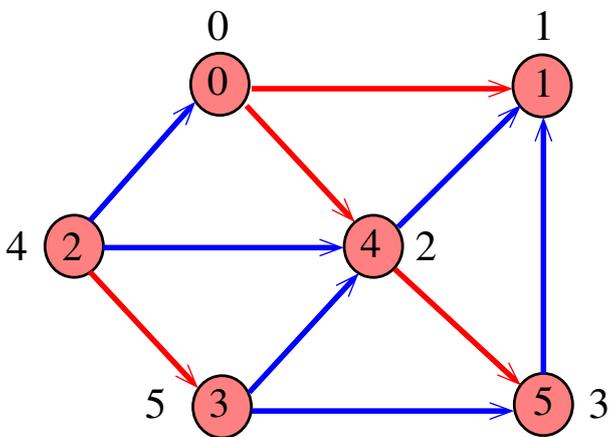
Navigation icons: back, forward, search, etc.

dfsR(G,2)



Navigation icons: back, forward, search, etc.

DIGRAPHdfs(G)



Navigation icons: back, forward, search, etc.

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);
}

```

Navigation icons: back, forward, search, etc.

dfsR

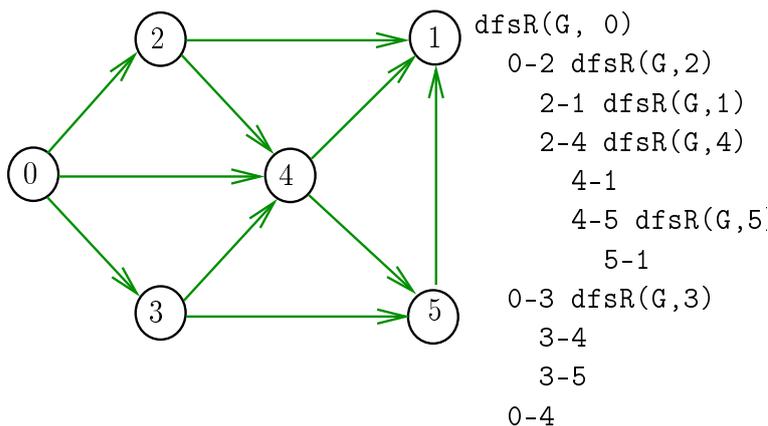
dfsR supõe que o digrafo G é representado por uma matriz de adjacência

```

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);
}

```

DIGRAPHdfs(G)



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 $\Theta(V^2)$.

dfsR

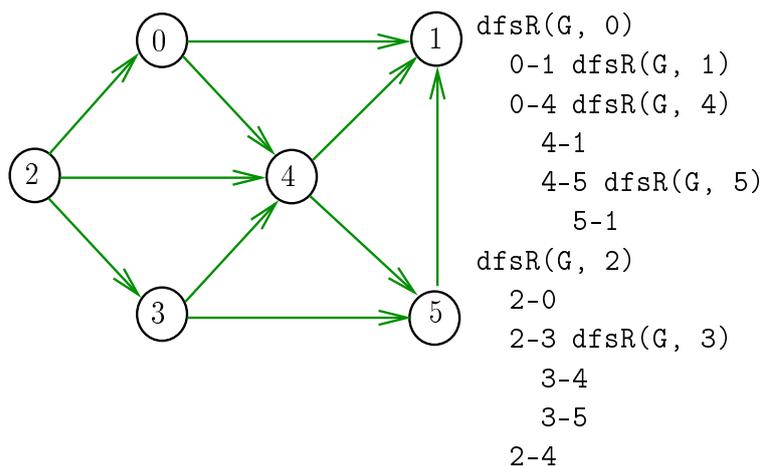
dfsR supõe que o digrafo G é representado por listas de adjacência

```

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)



Arborescência de busca em profundidade

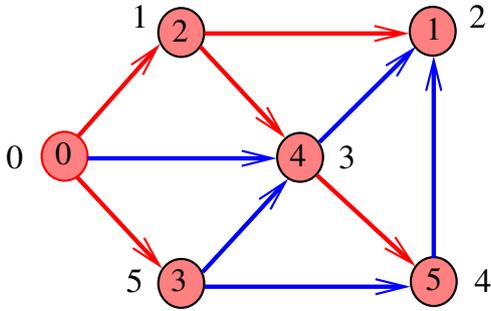
Classificação dos arcos

S 18.4 e 19.2
CLRS 22

Arcos da arborescência

Arcos da arborescência são os arcos $v-w$ que dfsR percorre para visitar w pela primeira vez

Exemplo: arcos em **vermelho** são arcos da arborescência

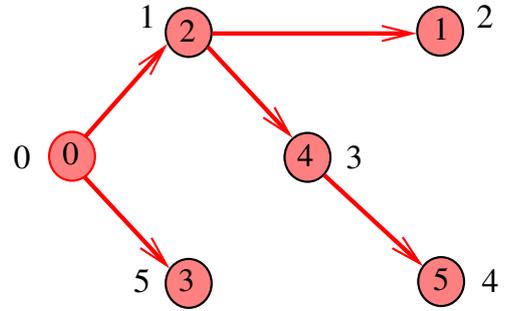


Navigation icons

Arcos da arborescência

Arcos da arborescência são os arcos $v-w$ que dfsR percorre para visitar w pela primeira vez

Exemplo: arcos em **vermelho** são arcos da arborescência

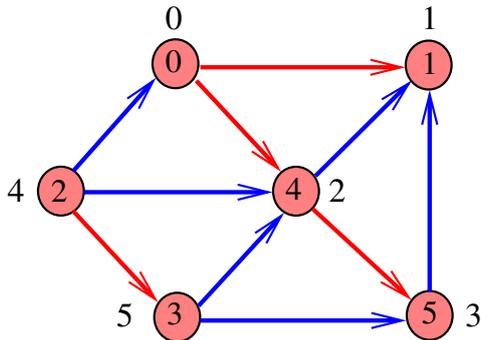


Navigation icons

Floresta DFS

Conjunto de arborescências é a **floresta da busca em profundidade** (= DFS forest)

Exemplo: arcos em **vermelho** formam a floresta DFS

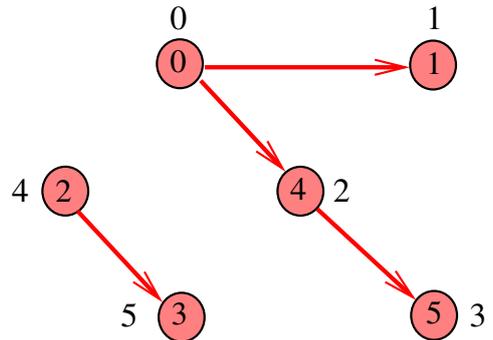


Navigation icons

Floresta DFS

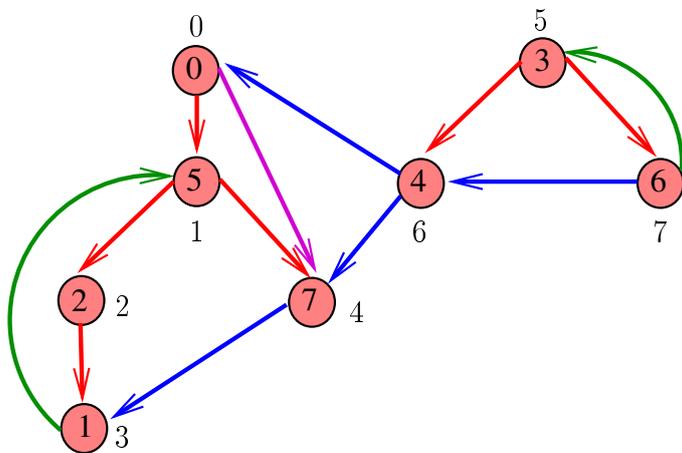
Conjunto de arborescências é a **floresta da busca em profundidade** (= DFS forest)

Exemplo: arcos em **vermelho** formam a floresta DFS



Navigation icons

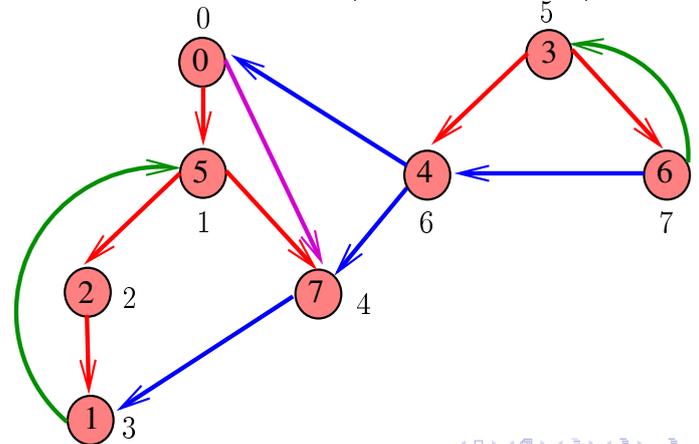
Classificação dos arcos



Navigation icons

Arcos de arborescência

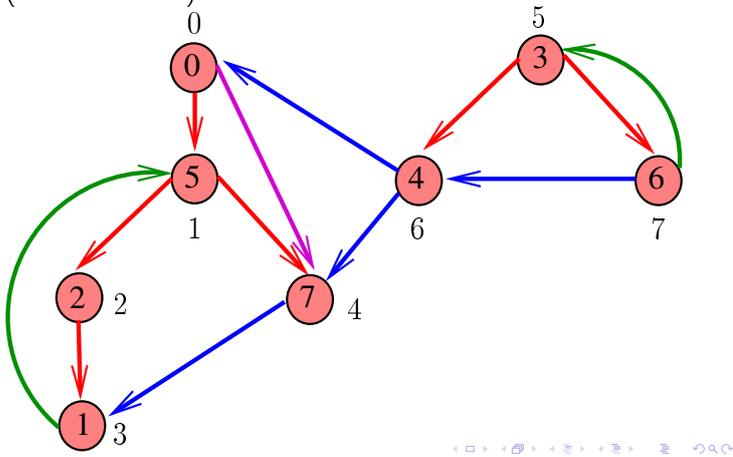
$v-w$ é **arco de arborescência** se foi usado para visitar w pela primeira vez (arcos **vermelhos**)



Navigation icons

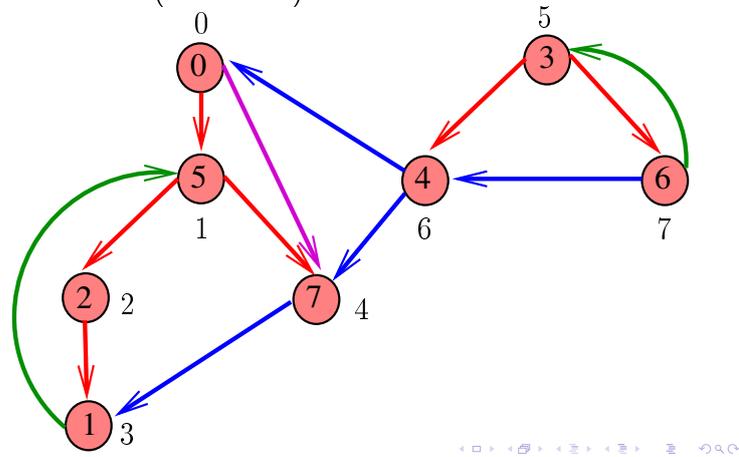
Arcos de retorno

$v-w$ é **arco de retorno** se w é ancestral de v (arcos **verdes**)



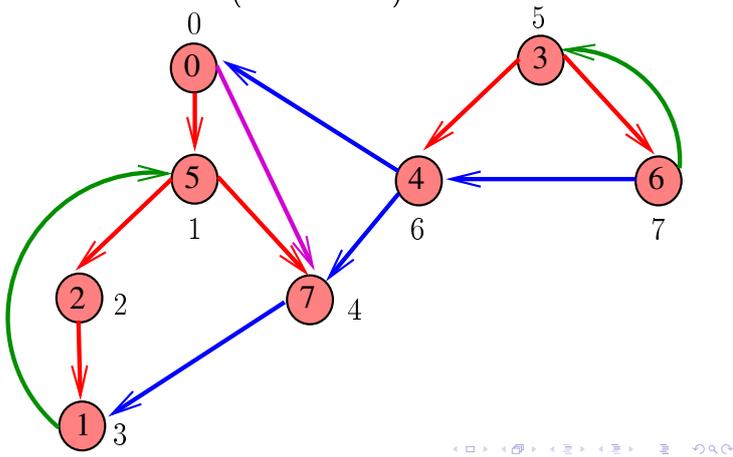
Arcos descendentes

$v-w$ é **descendente** se w é descendente de v , mas não é filho (arco **roxo**)



Arcos cruzados

$v-w$ é **arco cruzado** se w não é ancestral nem descendente de v (arcos **azuis**)



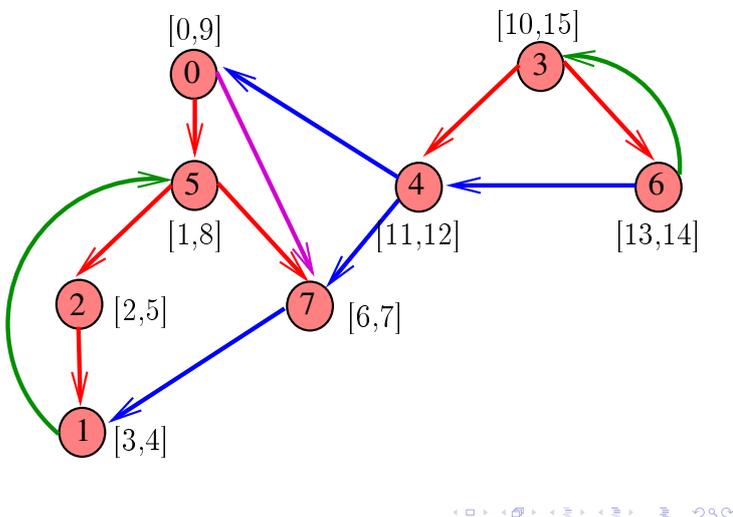
Busca DFS (CLRS)

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

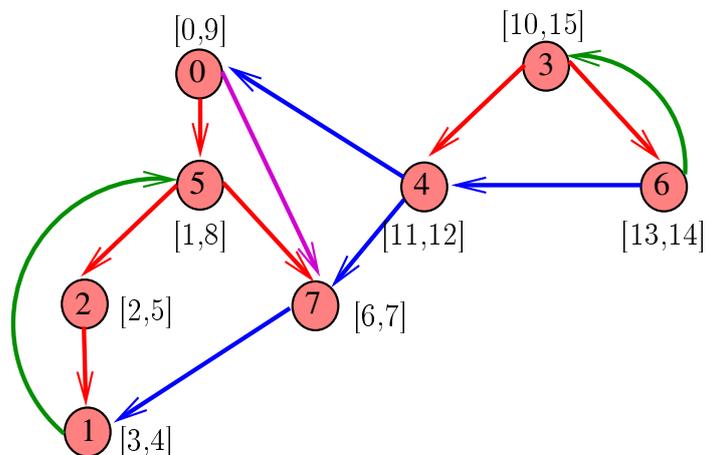
```
void DIGRAPHdfs (Digraph G) {
    Vertex v;
    1 time = 0;
    2 for (v = 0; v < G->V; v++) {
    3     d[v] = f[v] = -1;
    4     parnt[v] = -1;
    5 }
    6 for (v = 0; v < G->V; v++)
    7     if (d[v] == -1)
    8         dfsR(G, v);
}
```

dfsR

```

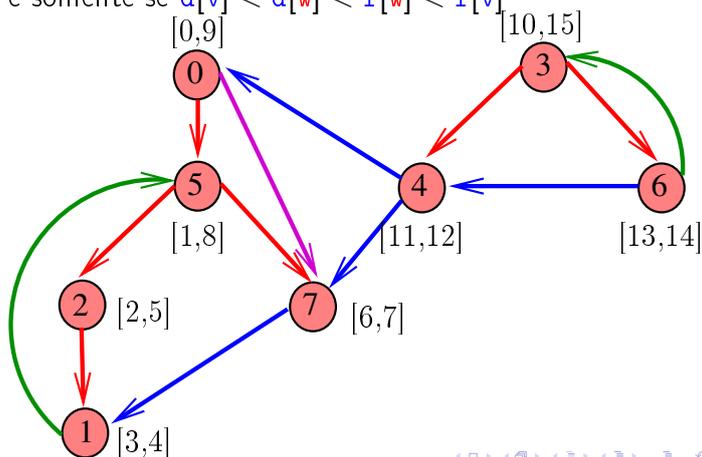
void dfsR (Digraph G, Vertex v) {
    link p;
1   d[v] = time++;
2   for (p = G->adj[v]; p != NULL; p= p->next)
3       if (d[p->w] == -1) {
4           parnt[w] = p->w;
5           dfsR(G, p->w);
6       }
7   f[v] = time++;
}
    
```

Classificação dos arcos



Arcos de arborescência ou descendentes

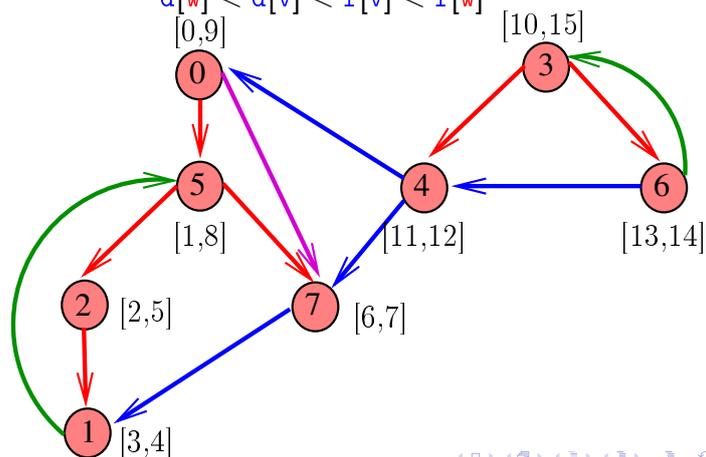
$v-w$ é **arco de arborescência** ou **descendente** se e somente se $d[v] < d[w] < f[w] < f[v]$



Arcos de retorno

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

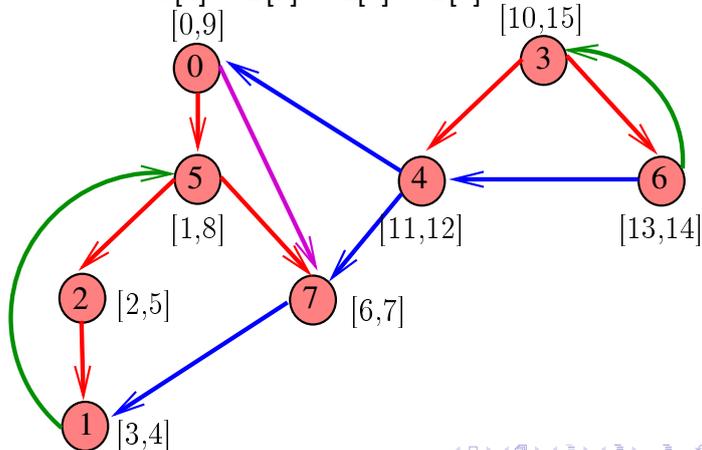
$$d[w] < d[v] < f[v] < f[w]$$



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