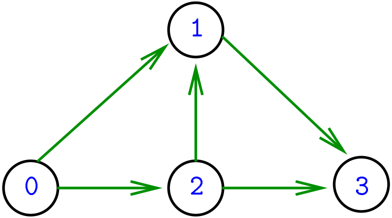
# Melhores momentos

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:



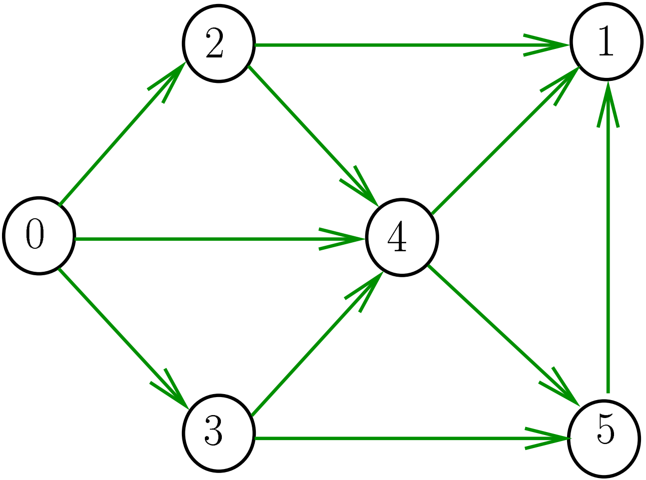

```
0:  1,  2
1:   3
2:  1,  3
3:
```

Consumo de espaço: $\Theta(V + A)$ (linear)

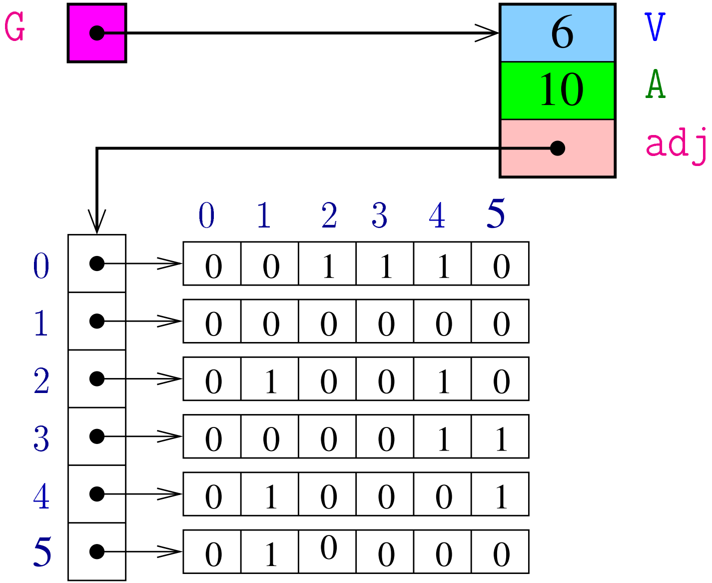
Manipulação eficiente

# Digrafo

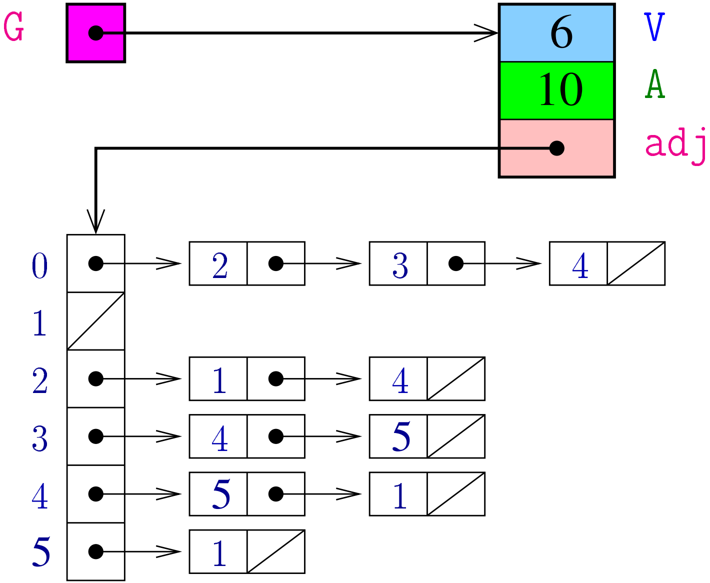
`Digraph G`



# Matriz de adjacência



# Listas de adjacência



# AULA 6

# Vetor de listas de adjacência (continuação)

S 17.4

# DIGRAPHpath

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;
    for (v = 0; v < G->V; v++) {
        lbl[v] = -1;
        parnt[v] = -1;
    }
    parnt[s] = s;
    pathR(G,s)
    if (lbl[t] == -1) return 0;
    else return 1;
}
```

# pathR

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

# pathR

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

# 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 | $= \mathtt{V} + 1$ | $= \Theta(\mathtt{V})$ |
| 2 | $= \mathtt{V}$ | $= \Theta(\mathtt{V})$ |
| 3 | $= 1$ | $=$ ???? |
| 4 | $= 1$ | $= \Theta(1)$ |
| 5 | $= 1$ | $= \Theta(1)$ |

total $= 2\,\Theta(1) + 2\,\Theta(\mathtt{V}) +$ ???

$= \Theta(\mathtt{V}) +$ ????

# Conclusão

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

# Consumo de tempo

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

# Consumo de tempo

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

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

total $= 3\,\mathrm{O}(\mathtt{V}) + \mathrm{O}(\mathtt{A}) + \mathrm{O}(\mathtt{V} + \mathtt{A})$

$= \mathrm{O}(\mathtt{V} + \mathtt{A})$

# Conclusão

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

# 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 $O(V^2)$.

# Busca DFS

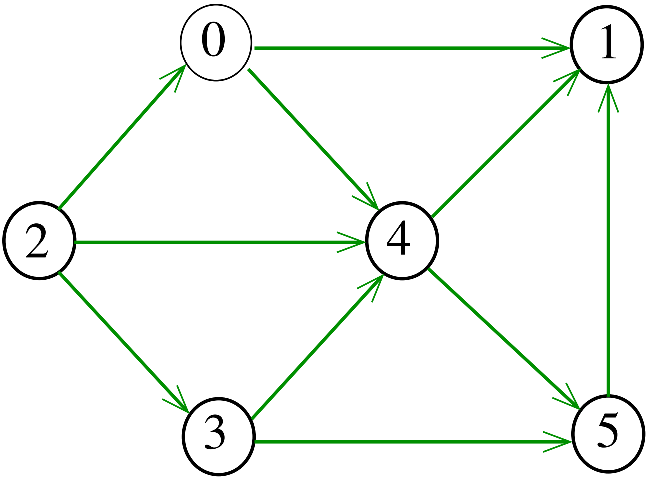
S 18.1 e 18.2

# Busca ou varredura

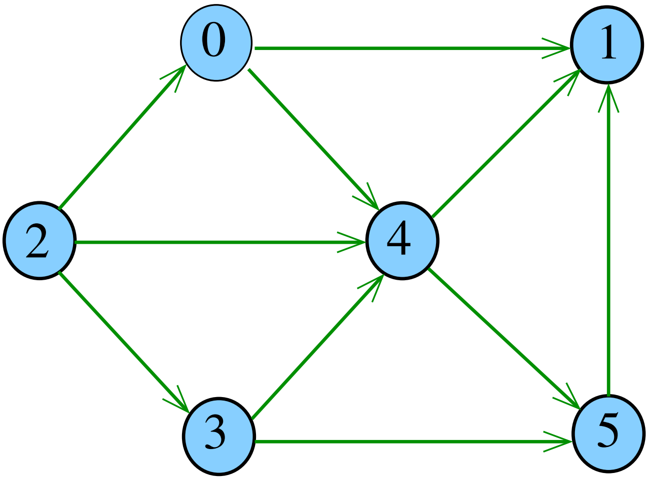
Um algoritimo de **busca** (ou **varredura**) examina, sistematicamente, todos os vértices e todos os arcos de um digrafo.

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

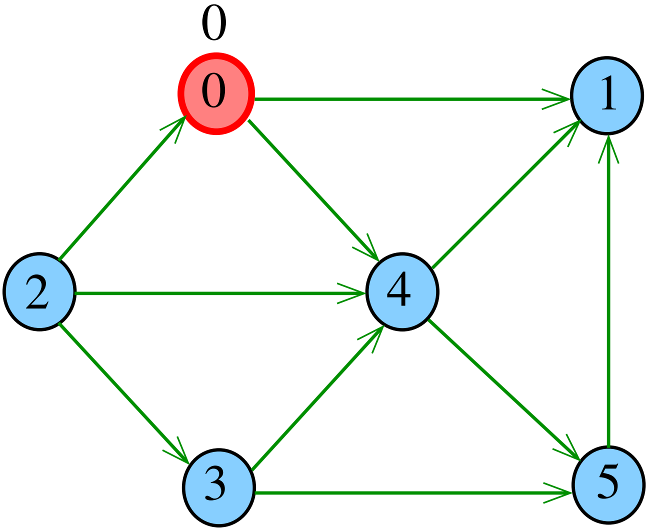
# DIGRAPHdfs(G)



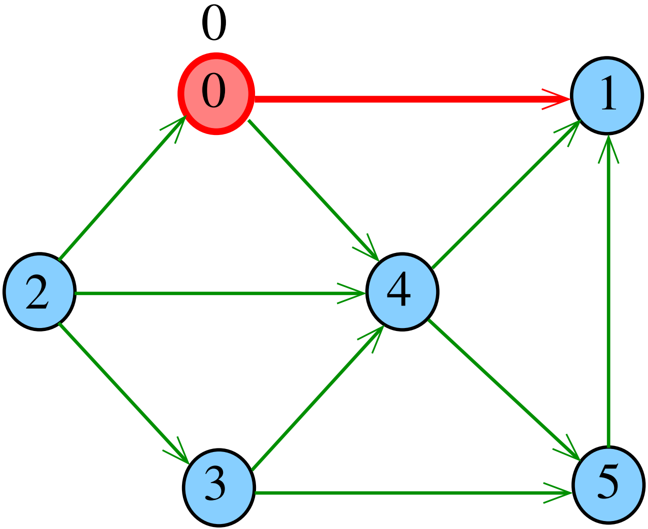
# DIGRAPHdfs(G)



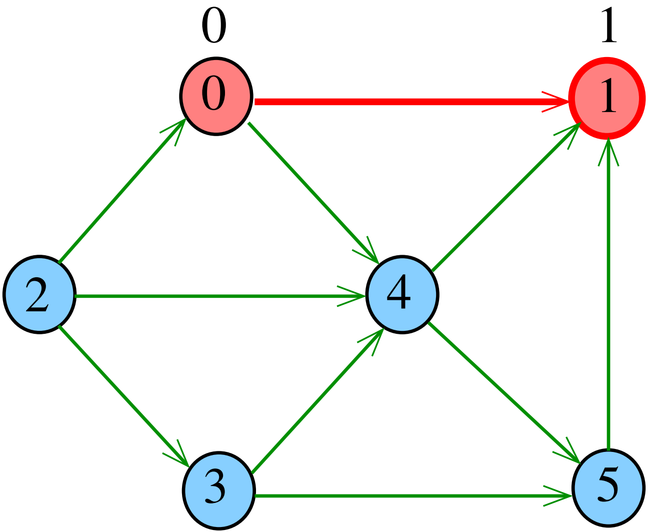
dfsR(G,0)



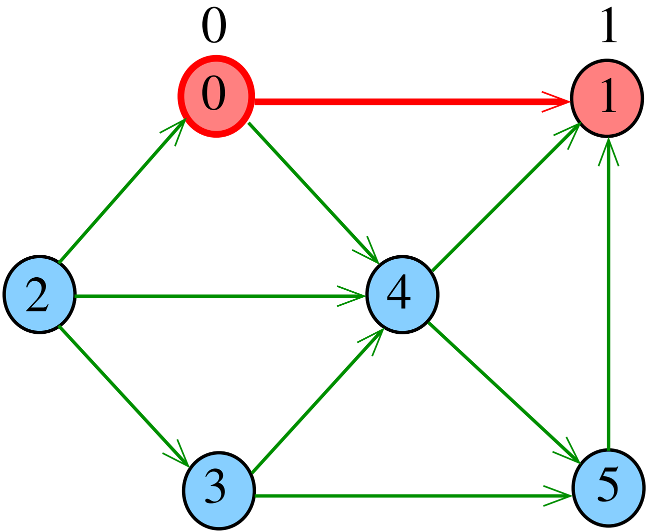
dfsR(G,0)



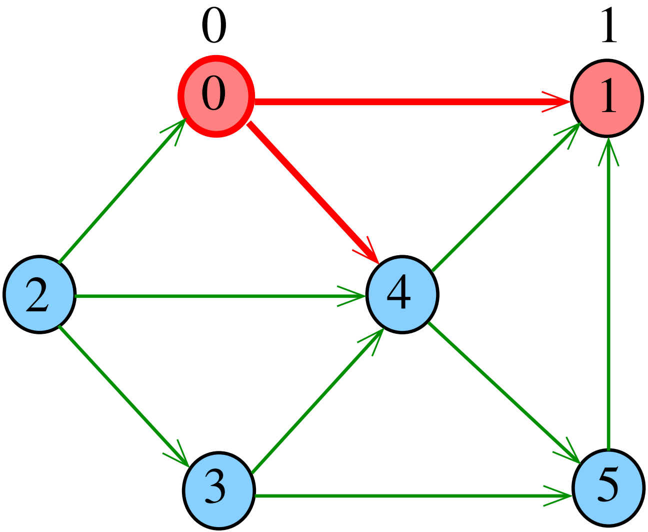
dfsR(G,1)



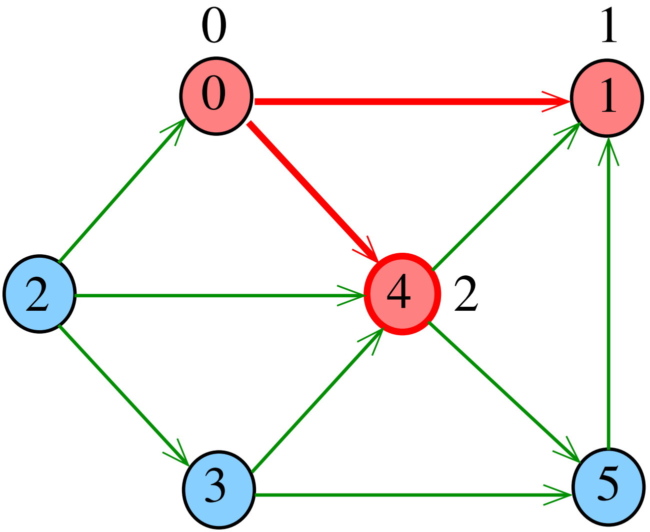
dfsR(G,0)



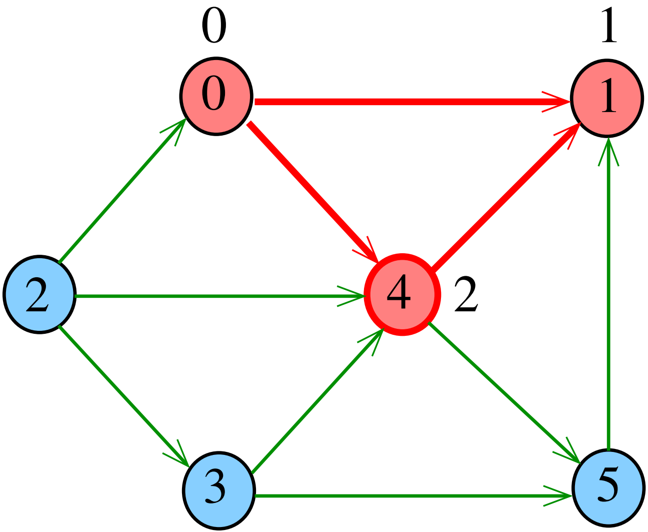
dfsR(G,0)



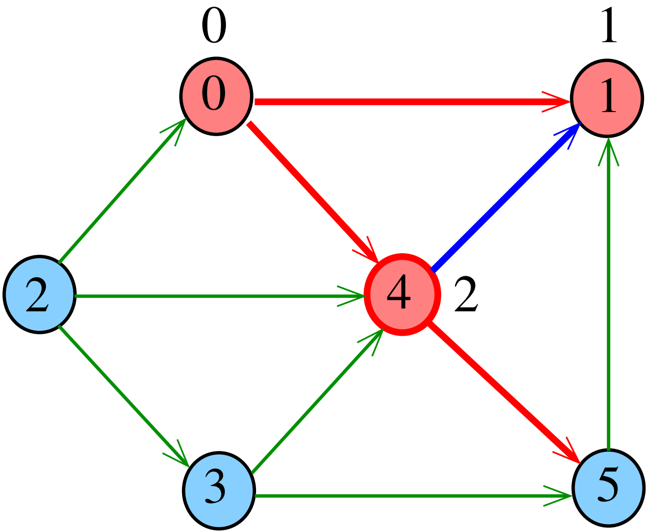
# dfsR(G,4)



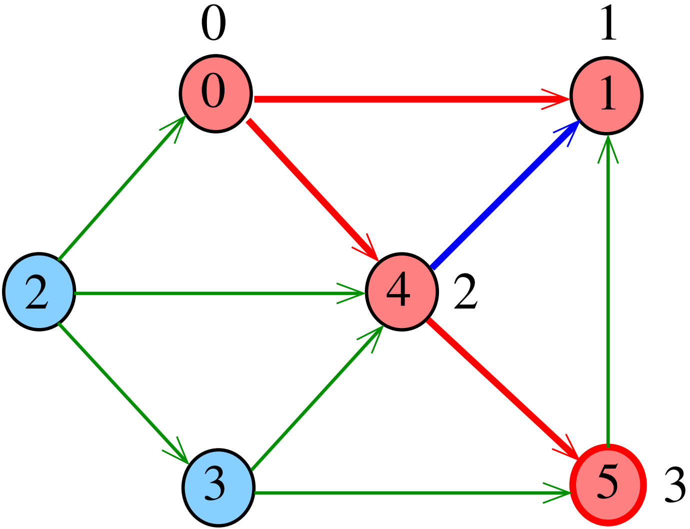
dfsR(G,4)



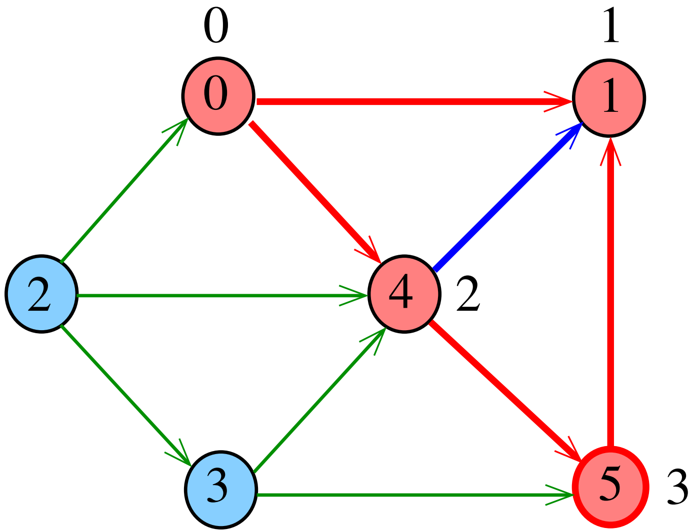
dfsR(G,4)



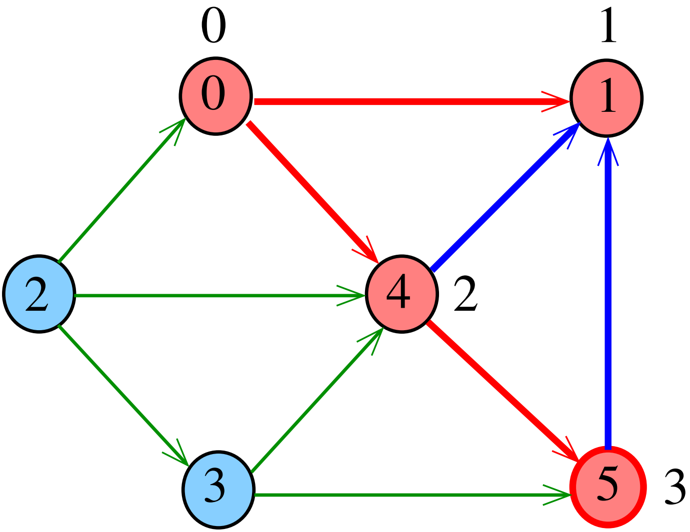
dfsR(G,5)



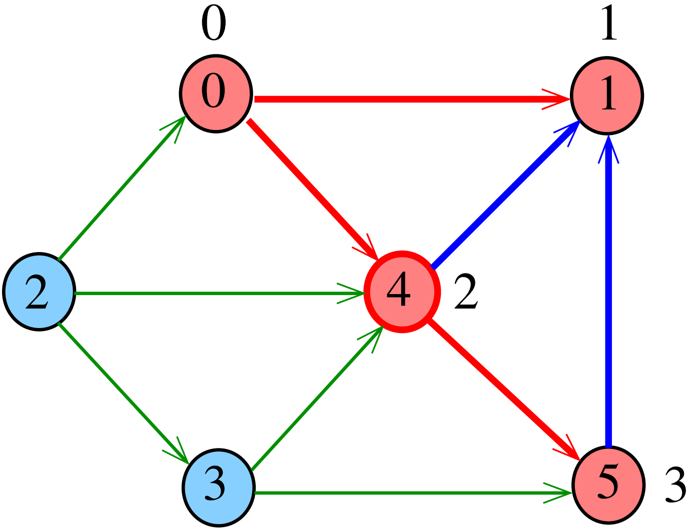
dfsR(G,5)



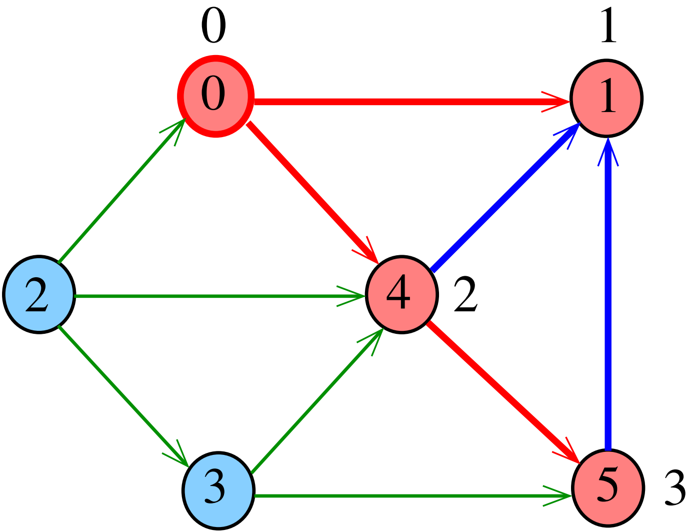
dfsR(G,5)



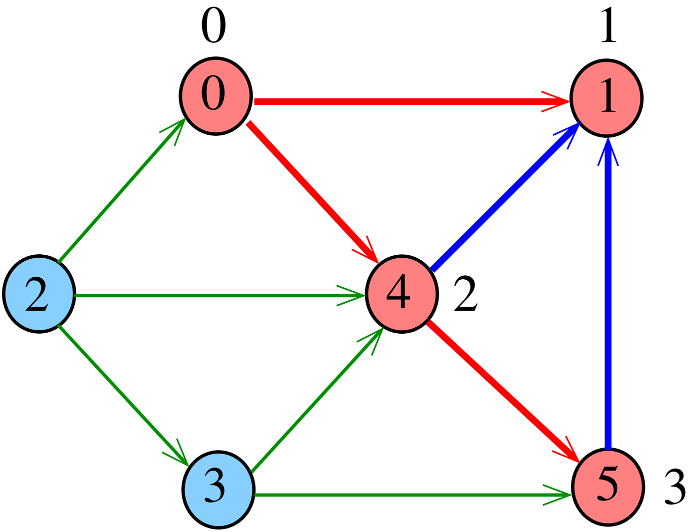
dfsR(G,4)



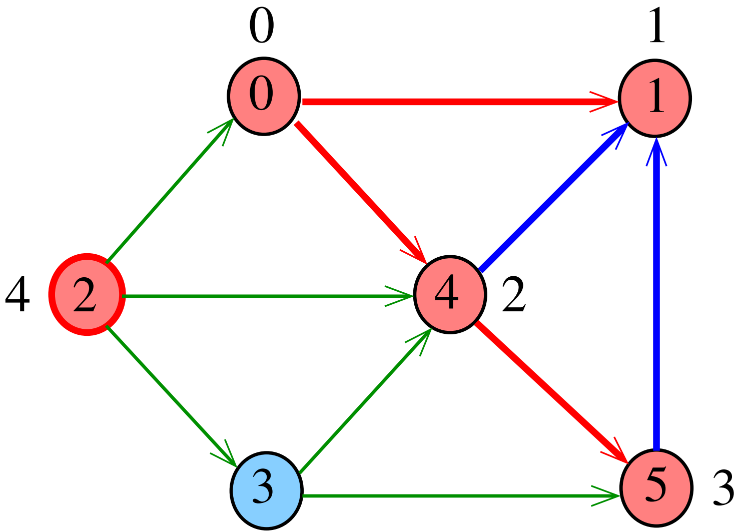
# dfsR(G,0)



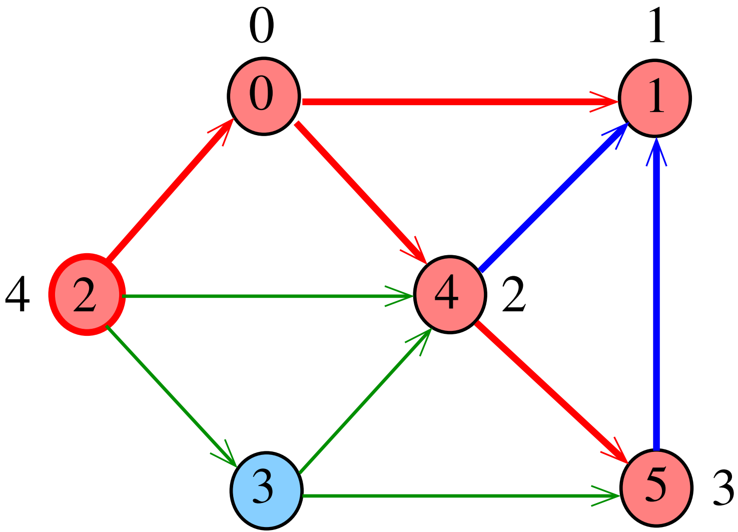
# DIGRAPHdfs(G)



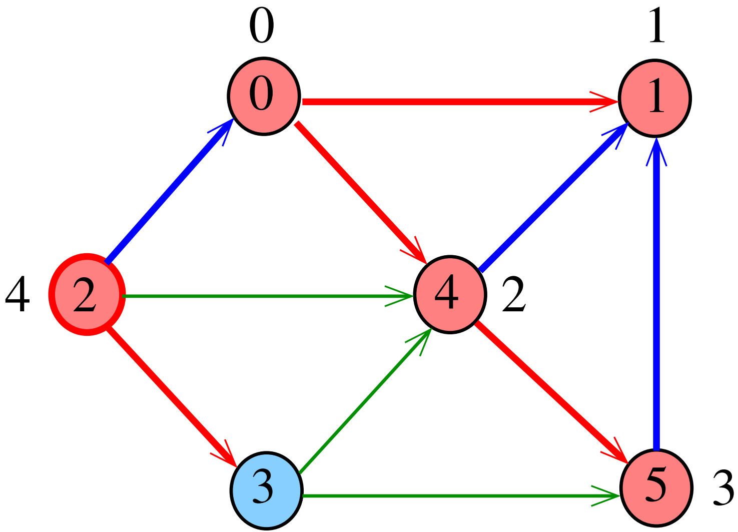
dfsR(G,2)



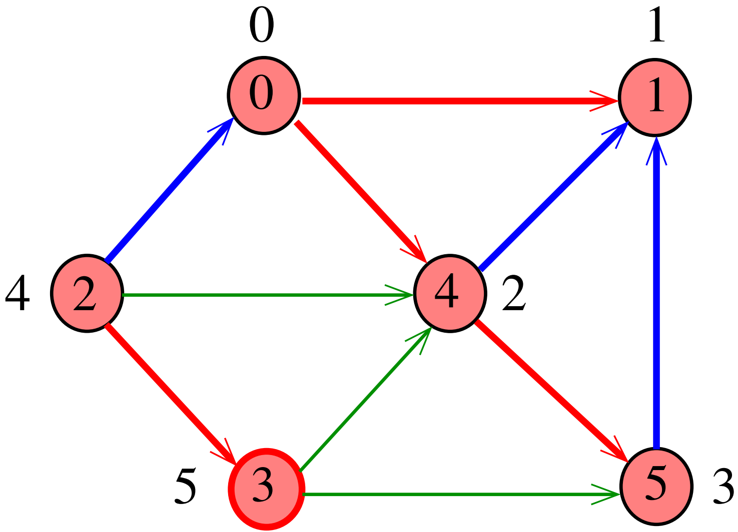
# dfsR(G,2)



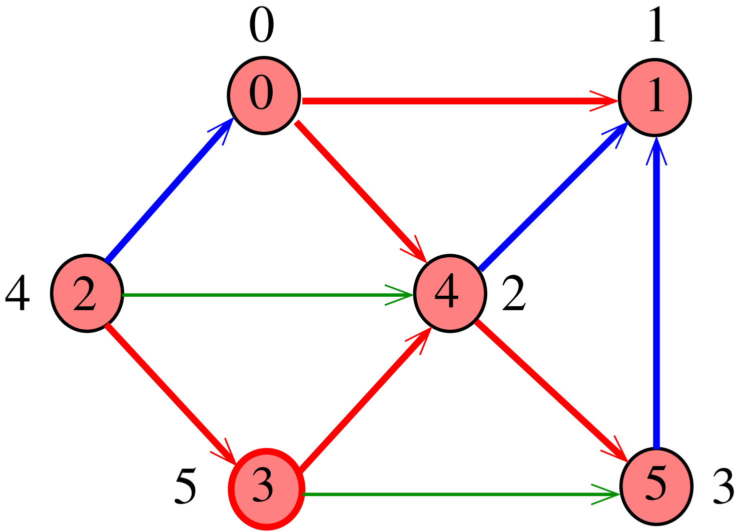
dfsR(G,2)



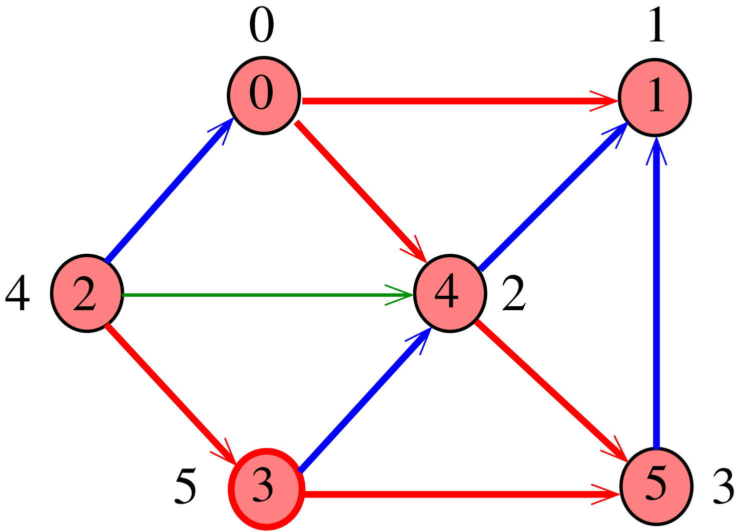
## dfsR(G,3)



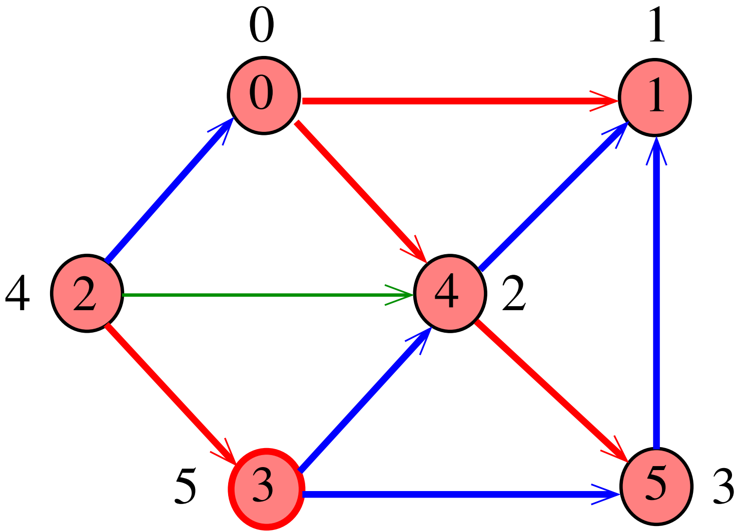
dfsR(G,3)



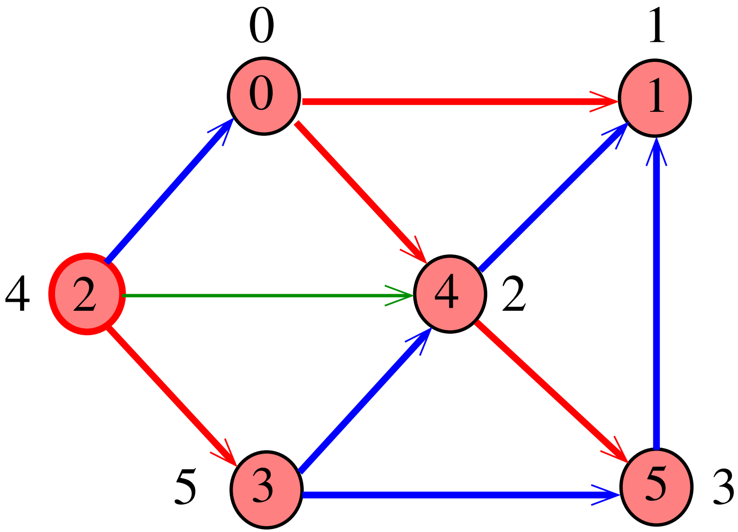
# dfsR(G,3)



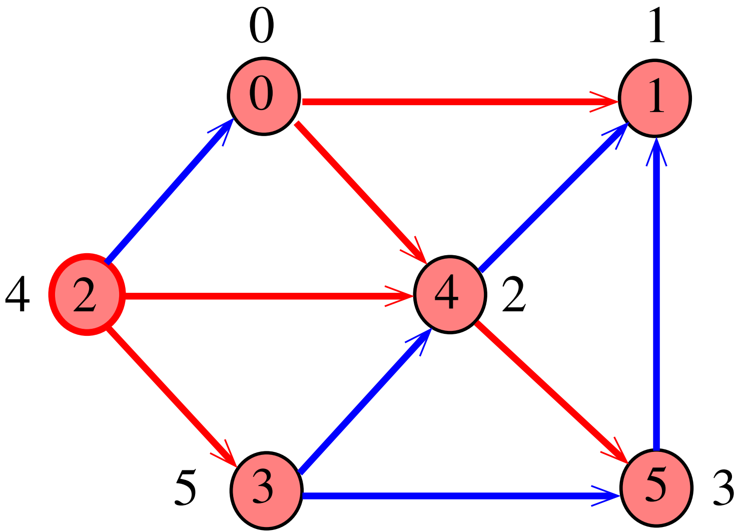
dfsR(G,3)



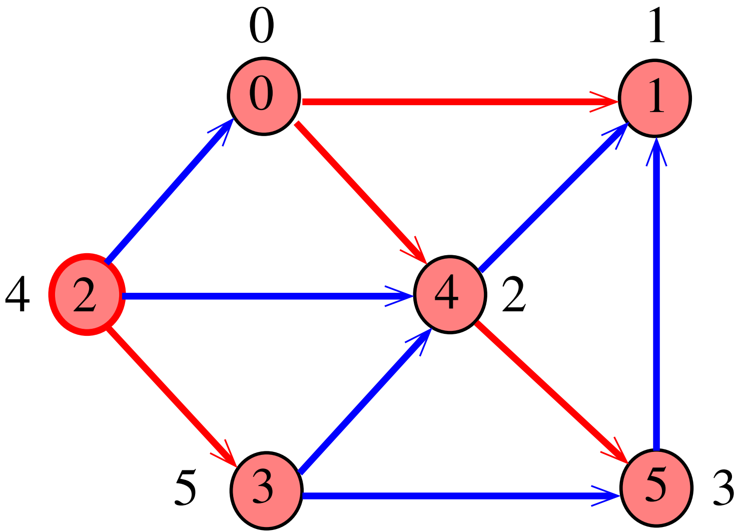
dfsR(G,2)



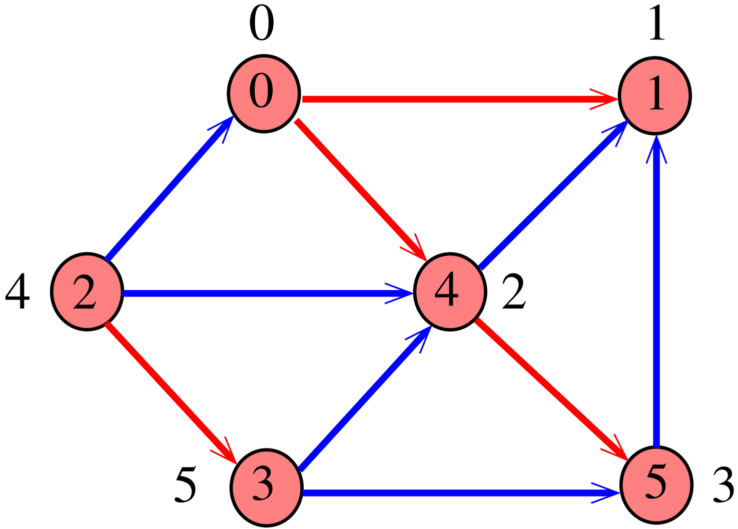
dfsR(G,2)



dfsR(G,2)



# DIGRAPHdfs(G)



# DIGRAPHdfs

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

# dfsR

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

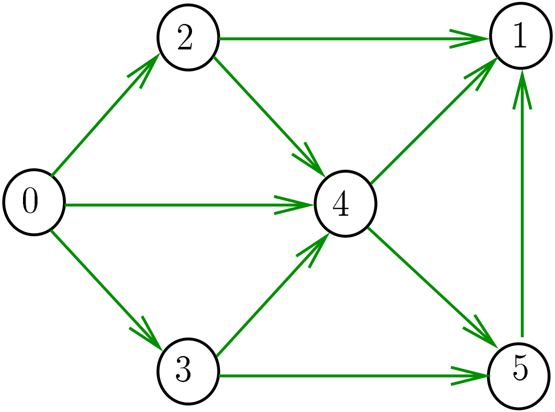
```
void dfsR (Digraph G, Vertex v) {
    Vertex w;
    lbl[v] = cnt++;
    for (w = 0; w < G->V; w++)
        if (G->adj[v][w] != 0)
            if (lbl[w] == -1)
                dfsR(G, w);
}
```

# dfsR

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

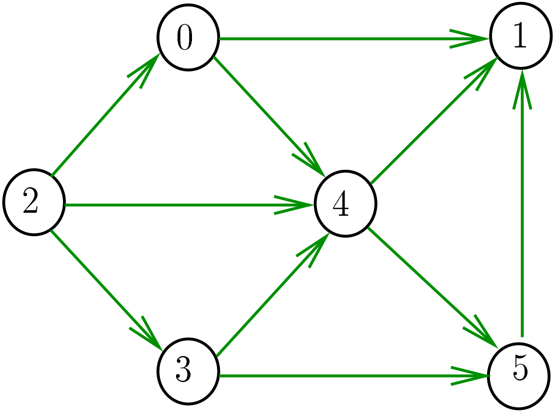
```
void dfsR (Digraph G, Vertex v) {
    link p;
    lbl[v] = cnt++;
    for (p = G->adj[v]; p != NULL; p= p->next)
        if (lbl[p->w] == -1)
            dfsR(G, p->w);
}
```

# 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 $\Theta(\mathtt{V} + \mathtt{A})$.

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

# Arborescência de busca em profundidade

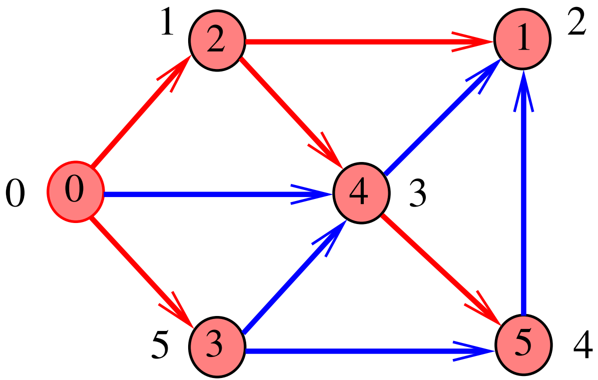
## Classificação dos arcos

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

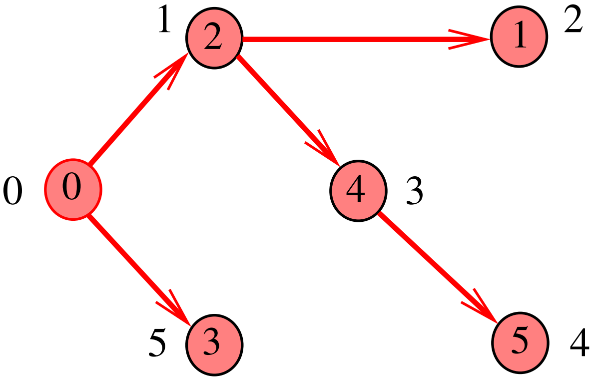
Exemplo: arcos em vermelho são arcos da arborescência



# Arcos da arborescência

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

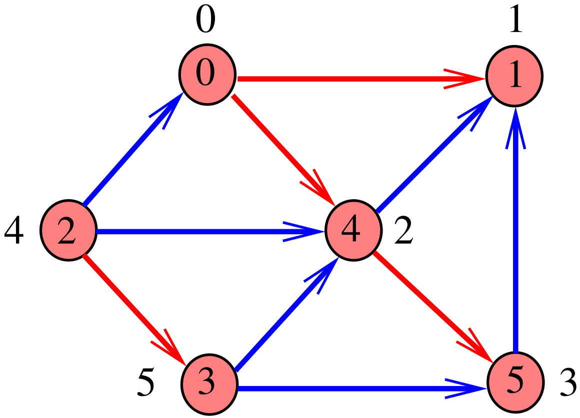
Exemplo: arcos em vermelho são arcos da arborescência



# Floresta DFS

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

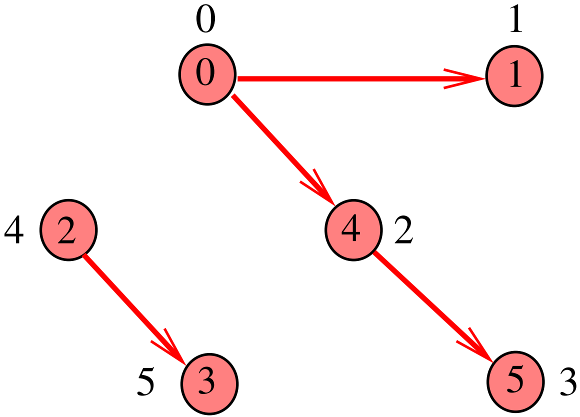
Exemplo: arcos em vermelho formam a floresta DFS



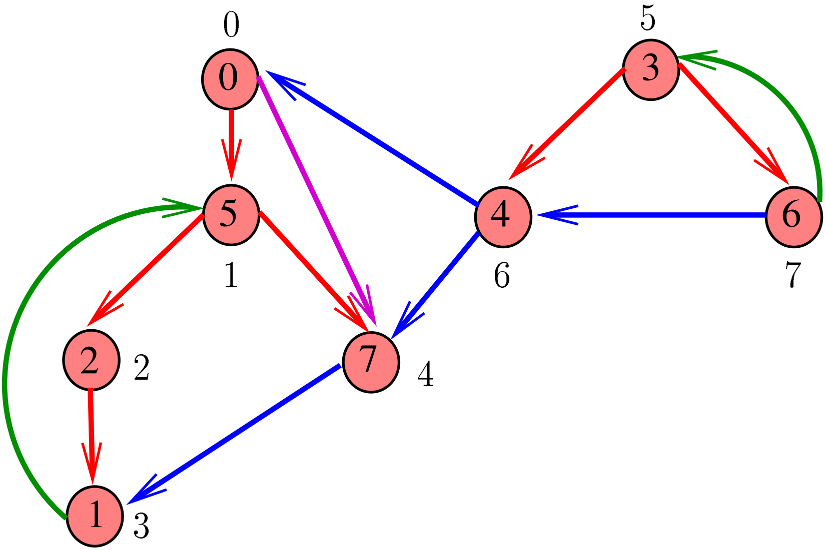
# Floresta DFS

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

Exemplo: arcos em vermelho formam a floresta DFS

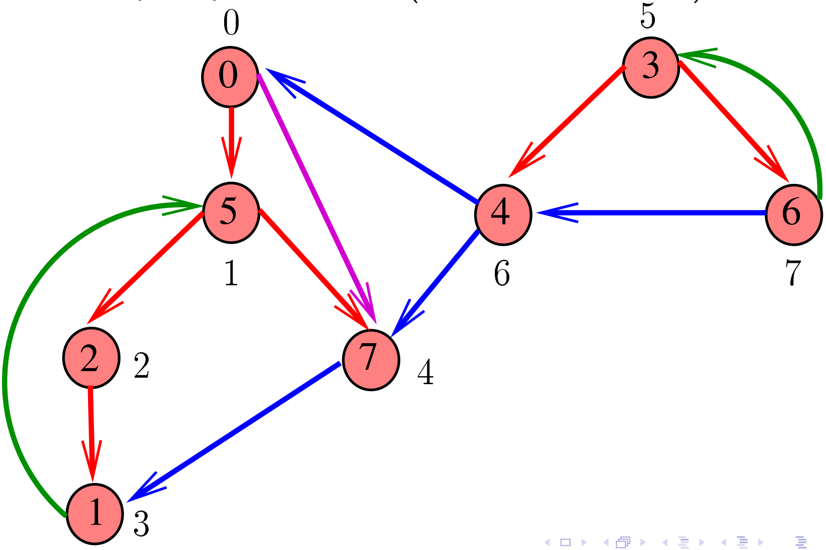


# Classificação dos arcos



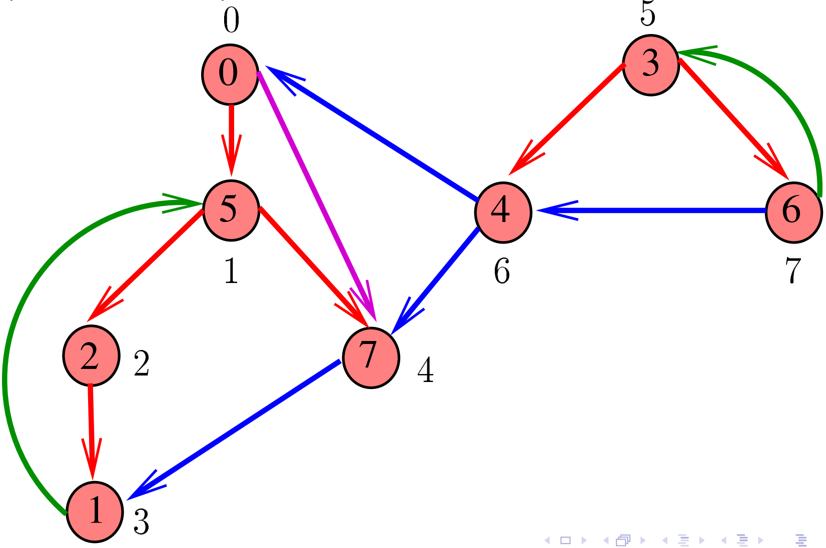
# Arcos de arborescência

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



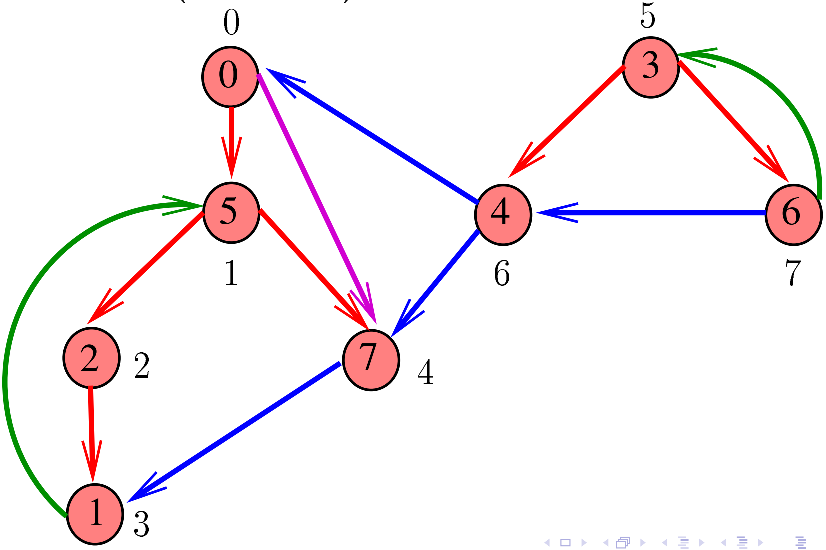
# Arcos de retorno

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



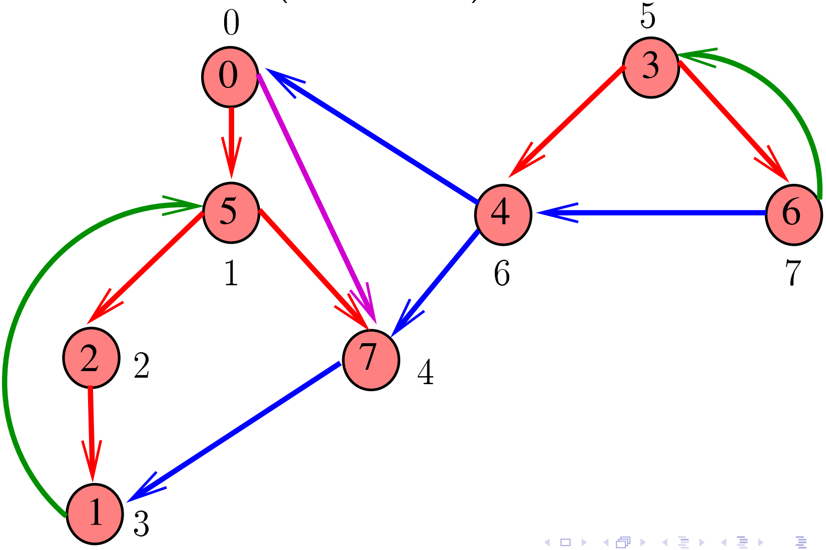
# Arcos descendentes

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**)



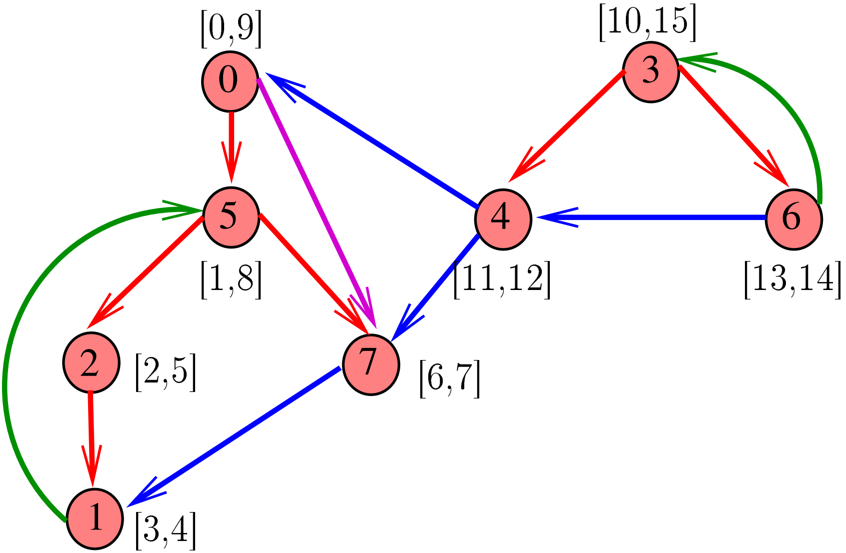
# Busca DFS (CLRS)

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)



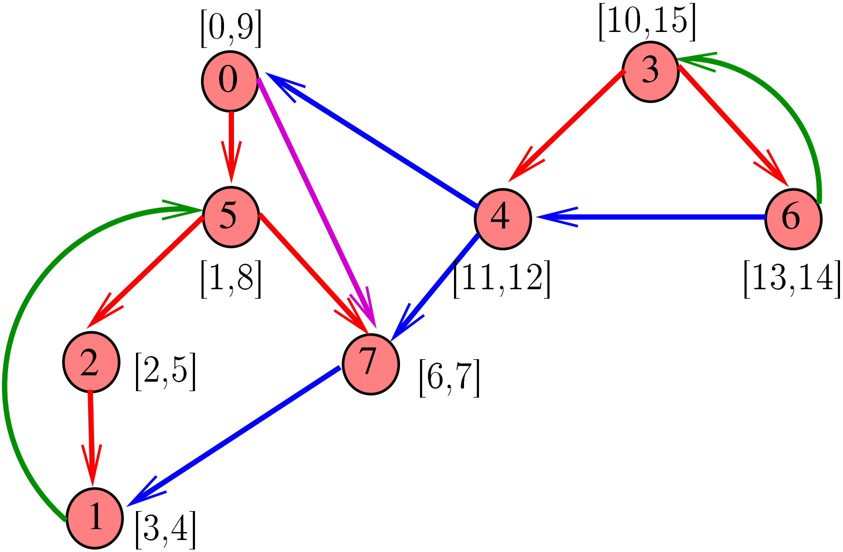
# DIGRAPHdfs

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

# dfsR

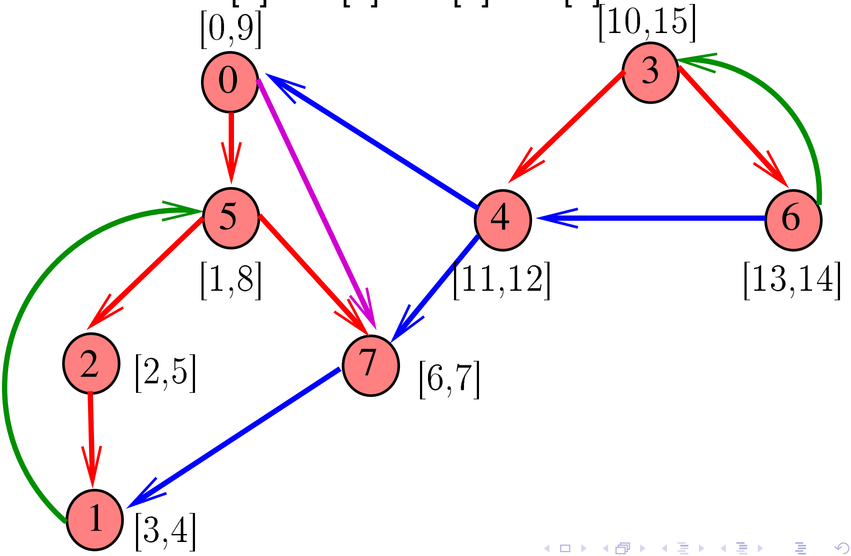
```
void dfsR (Digraph G, Vertex v) {
    link p;
    d[v] = time++;
    for (p = G->adj[v]; p != NULL; p= p->next)
        if (d[p->w] == -1) {
            parnt[w] = p->w;
            dfsR(G, p->w);
        }
    f[v] = time++;
}
```

# Classificação dos arcos



# Arcos de arborescência ou descendentes

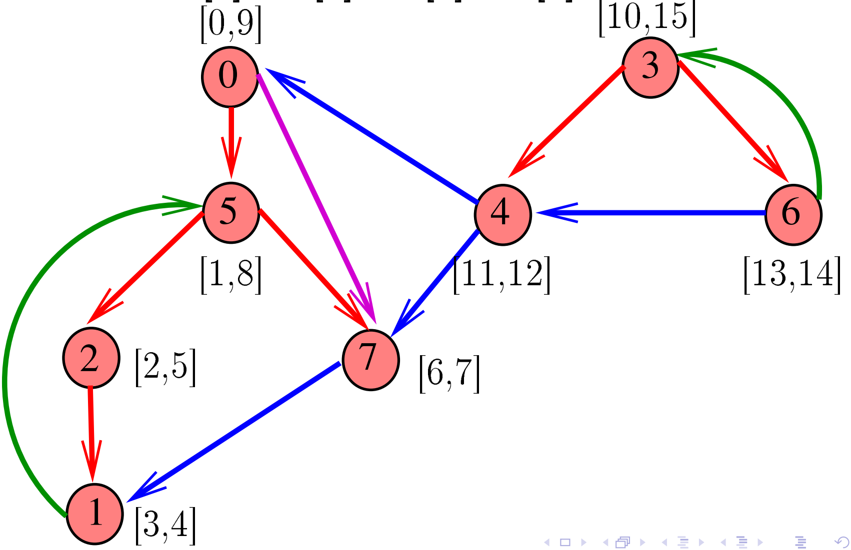
$\texttt{v-w}$ é **arco de arborescência** ou **descendente** se e somente se $\texttt{d[v]} < \texttt{d[w]} < \texttt{f[w]} < \texttt{f[v]}$



# Arcos de retorno

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

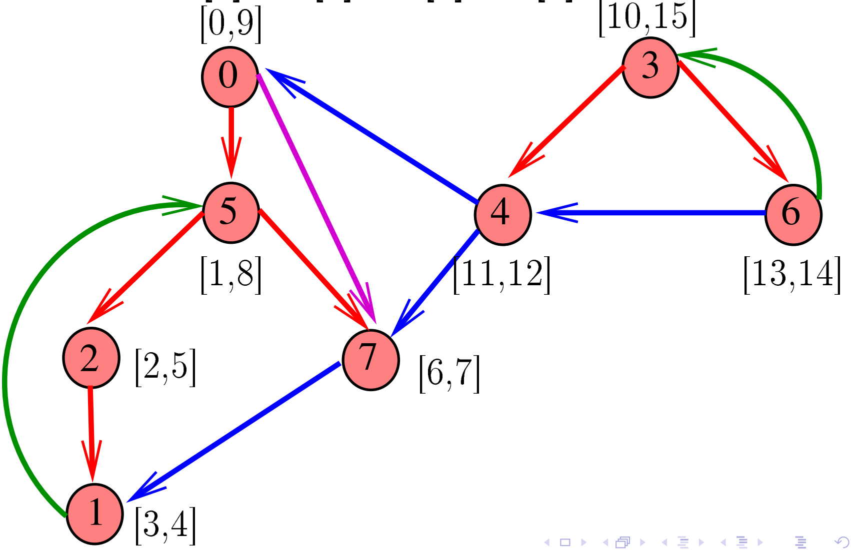
$$d[w] < d[v] < f[v] < f[w]$$



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