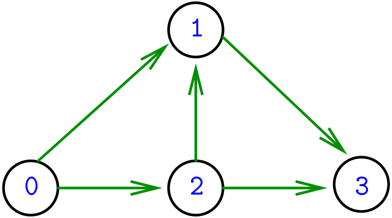
Melhores momentos

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:



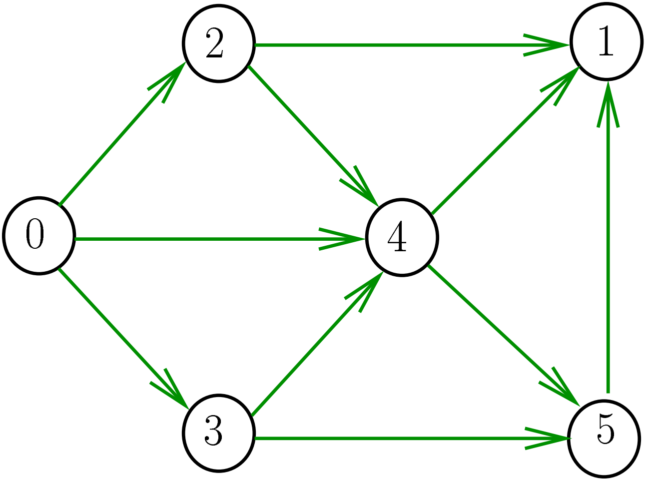

```
0:  1,  2
1:  3
2:  1,  3
3:
```

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

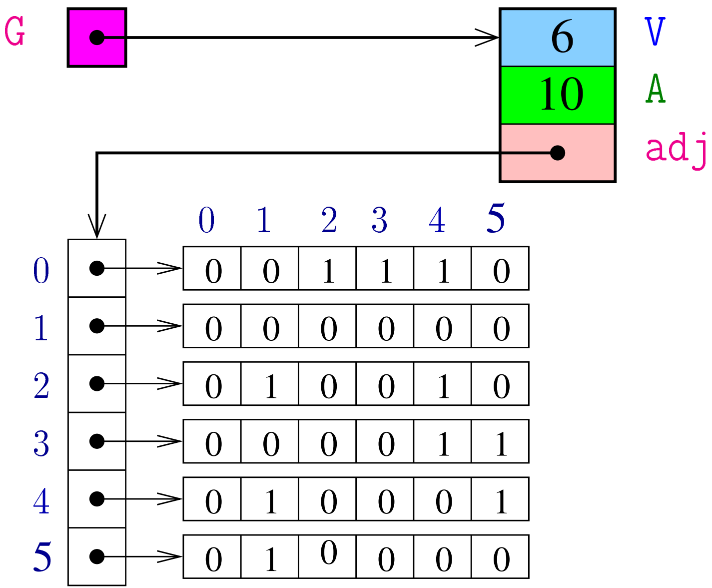
Manipulação eficiente

Digrafo

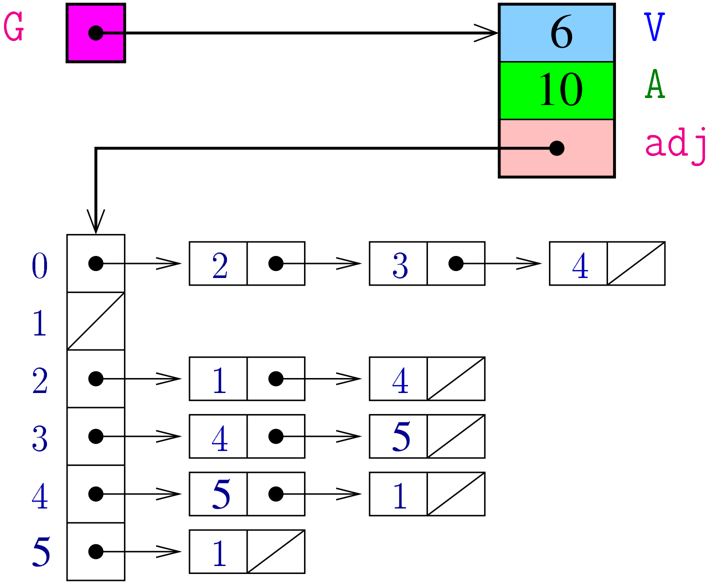
`Digraph G`



Matriz de adjacência



Listas de adjacência



DIGRAPHshow

Para cada vértice `v` de `G`, imprime, em uma linha, os vértices adjacentes a `v`

```
void DIGRAPHshow (Digraph G) {
    Vertex v, w;
    for (v = 0; v < G->V; v++) {
        printf("%2d:", v);
        for (w = 0;w< G->V;w++)
            if (G->adj[v][w] == 1)
                printf("%2d", w);
        printf("\n");
    }
}
```

DIGRAPHshow

```
void DIGRAPHshow (Digraph G) {
    Vertex v;
    link p;;
    for (v = 0; v < G->V; v++) {
        printf("%2d:", v);
        for (p=G->adj[v];p!= NULL;p=p->next)
            printf("%2d";p->w);
        printf("\n");
    }
}
```

Conclusão

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

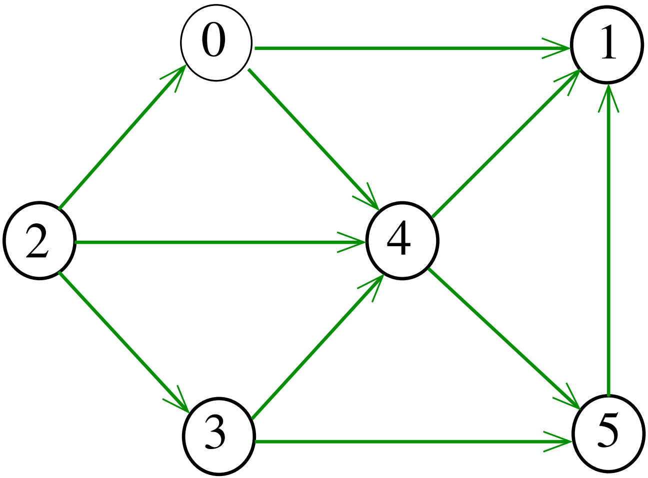
O consumo de tempo da função `DigraphShow` para matriz adjacência é $\Theta(\mathtt{V}^2)$.

Busca ou varredura

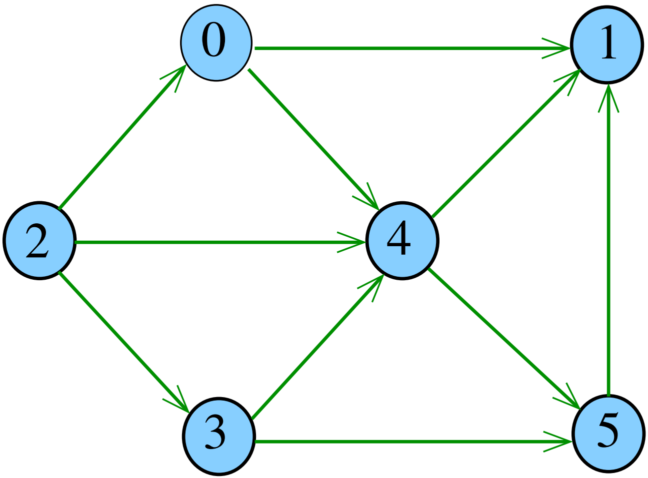
Um algoritimo de **busca** (ou **varredura**) examina, sistematicamente, todos os vértices e todos os arcos de um digrafo.

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

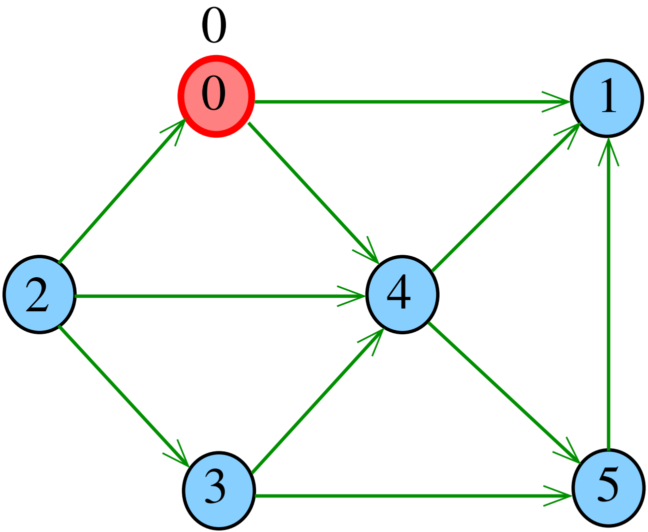
DIGRAPHdfs(G)



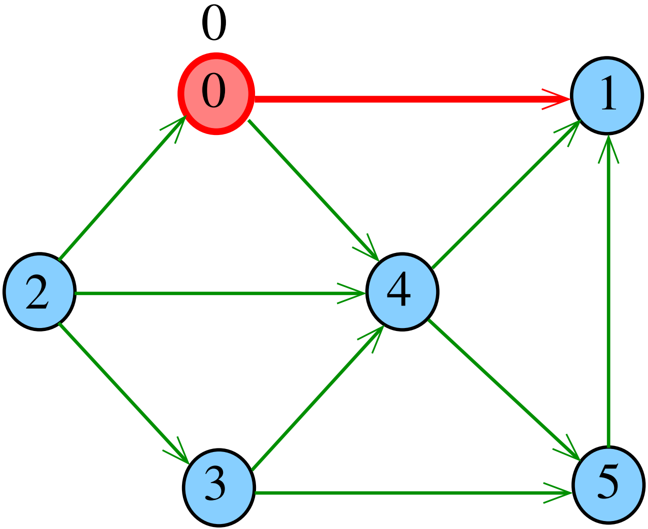
DIGRAPHdfs(G)



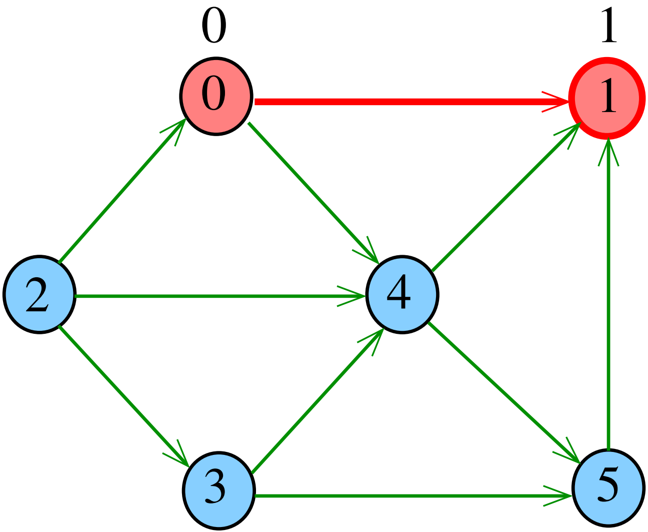
dfsR(G,0)



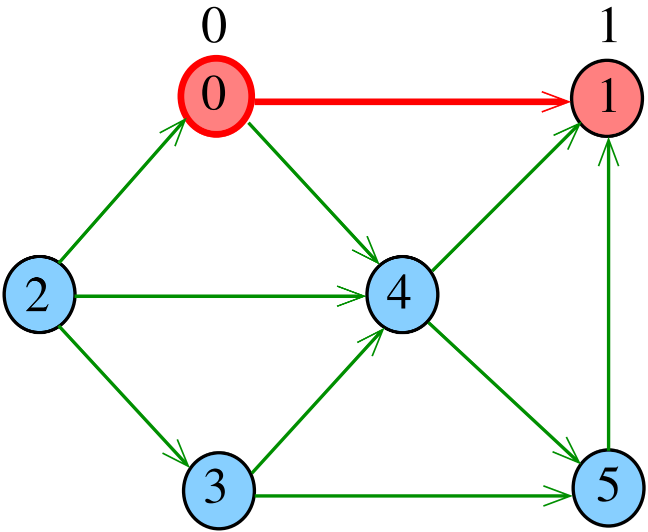
dfsR(G,0)



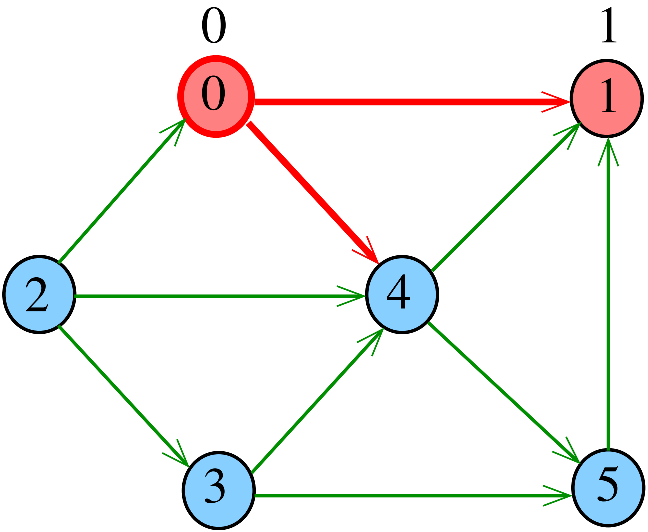
dfsR(G,1)



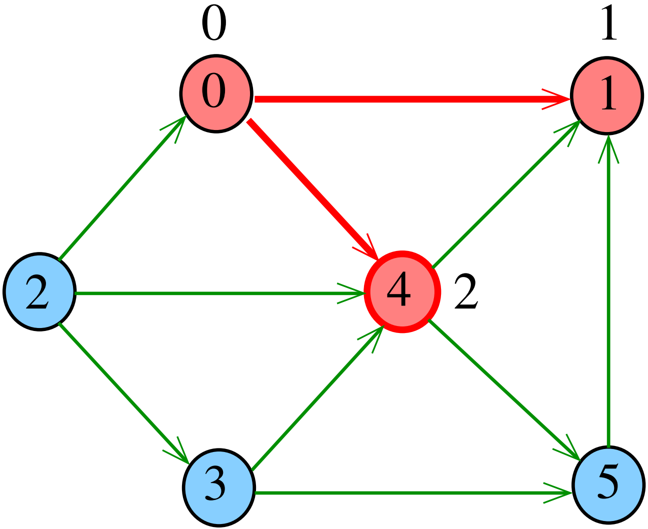
dfsR(G,0)



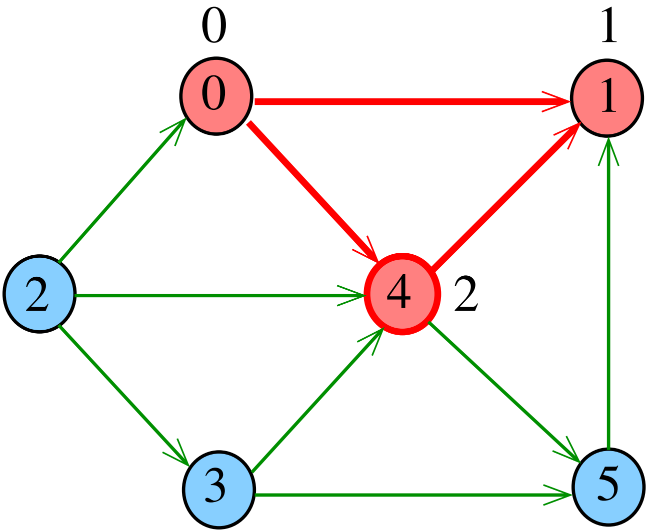
dfsR(G,0)



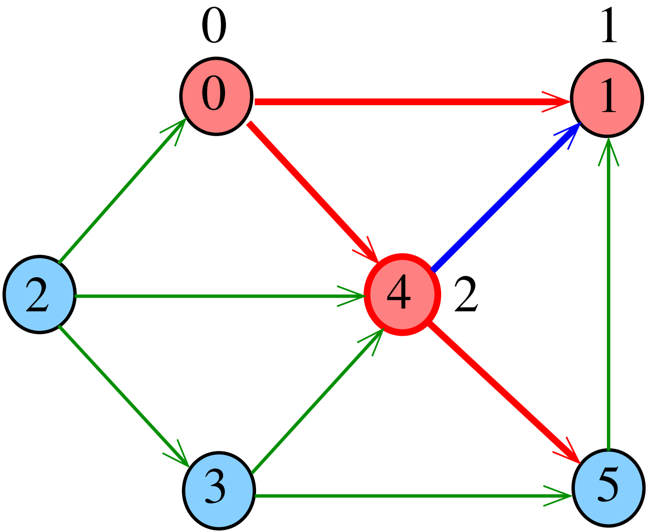
dfsR(G,4)



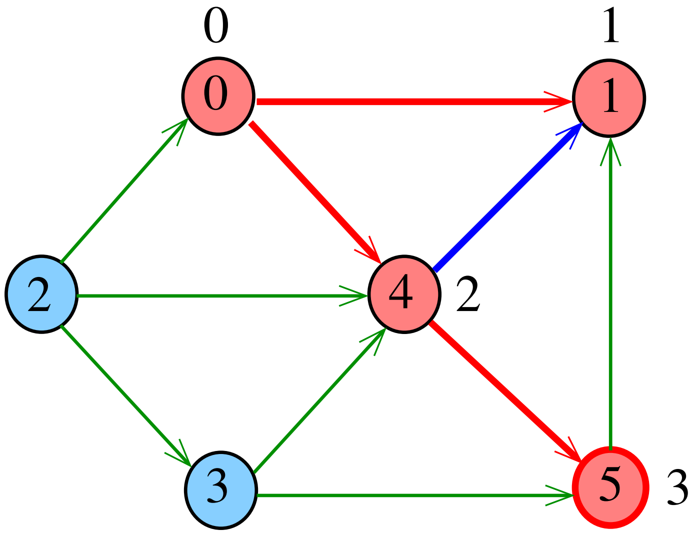
dfsR(G,4)



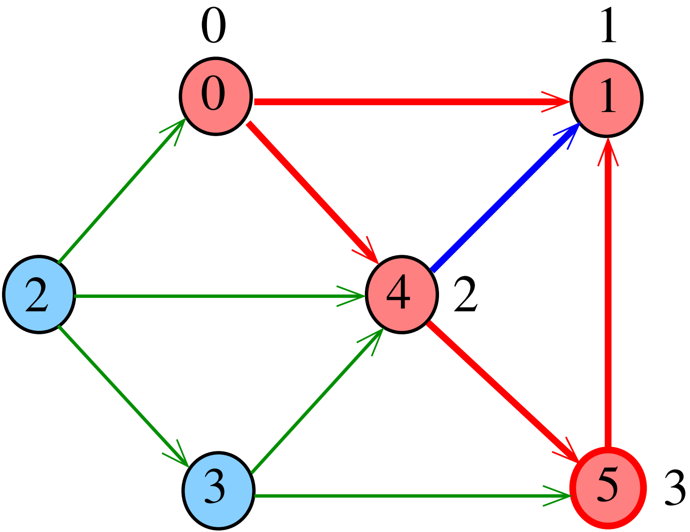
dfsR(G,4)



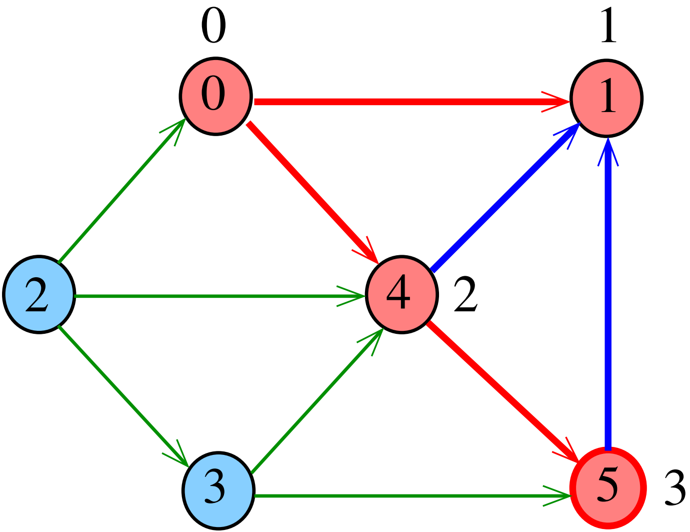
dfsR(G,5)



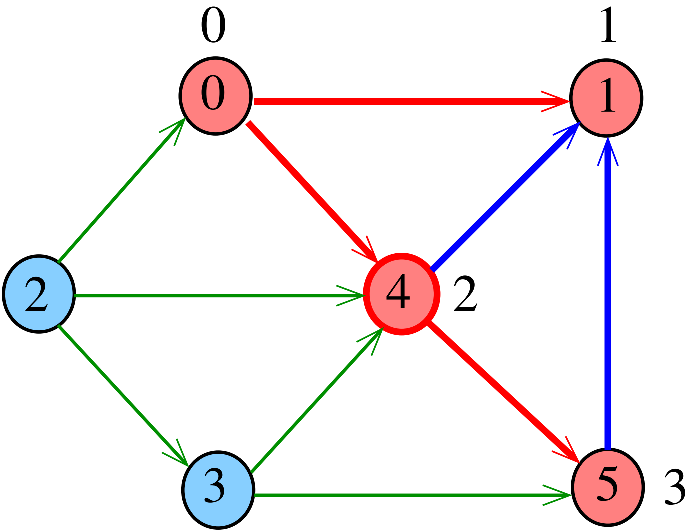
dfsR(G,5)



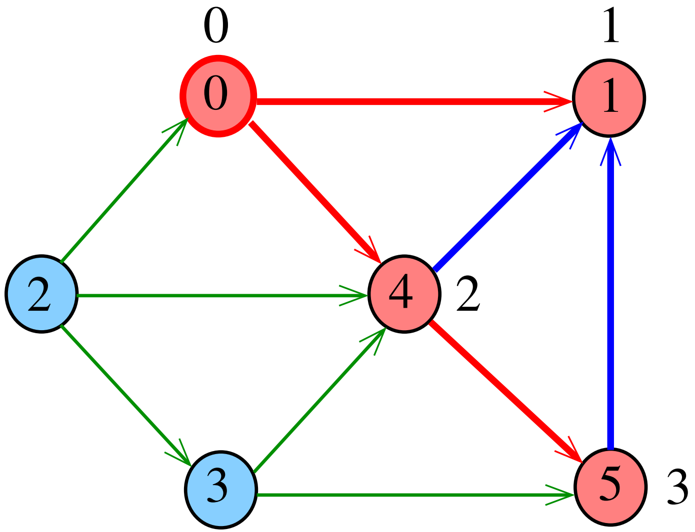
dfsR(G,5)



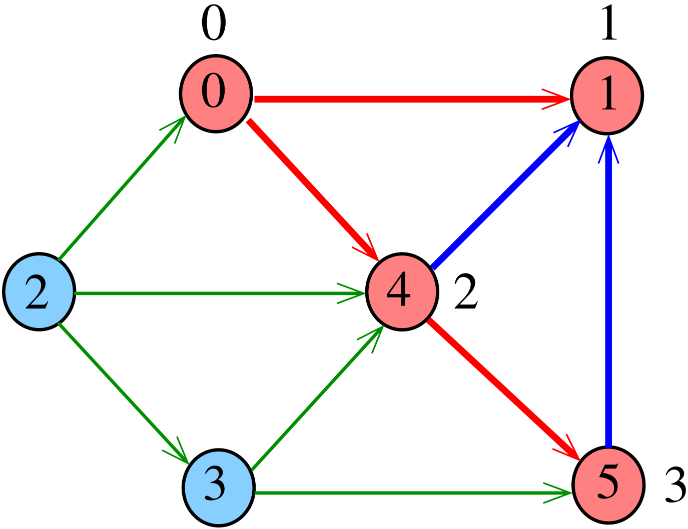
dfsR(G,4)



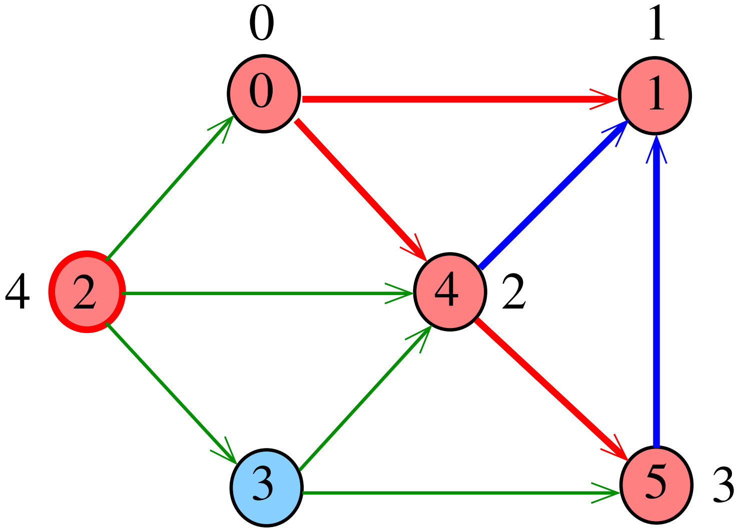
dfsR(G,0)



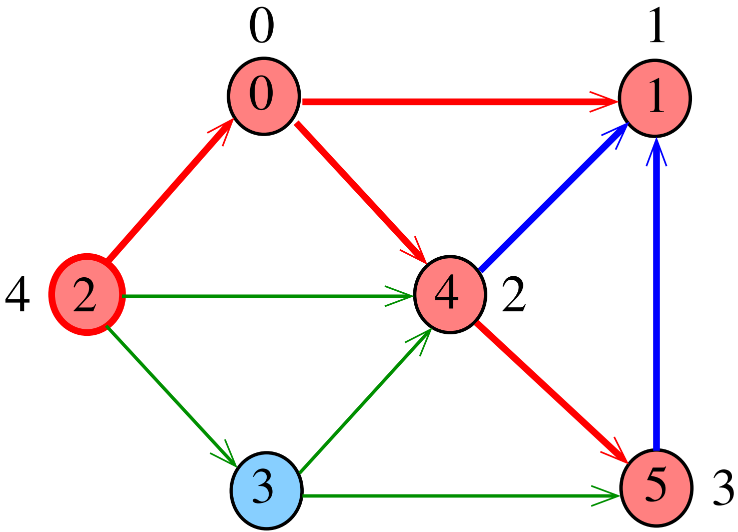
DIGRAPHdfs(G)



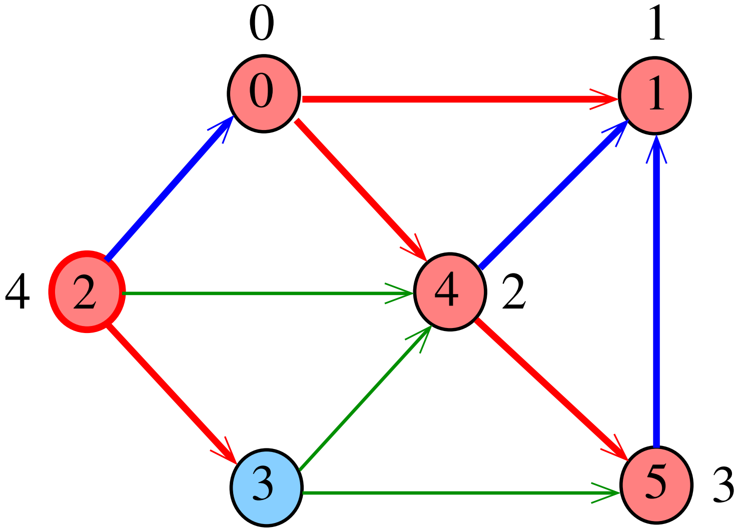
dfsR(G,2)



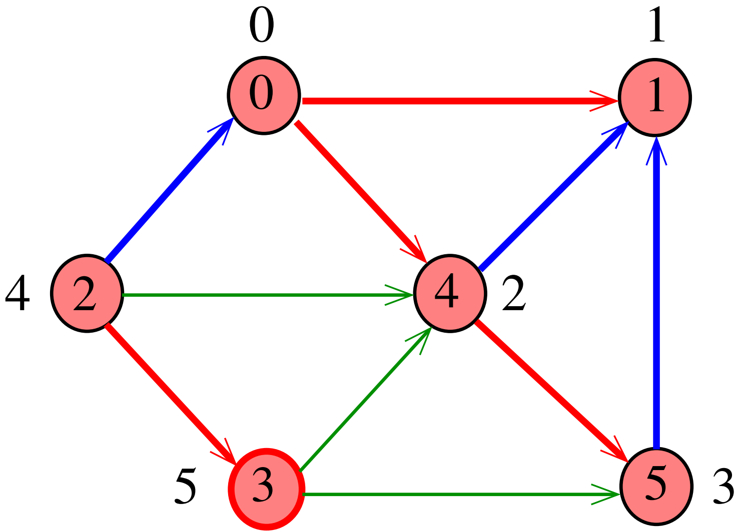
dfsR(G,2)



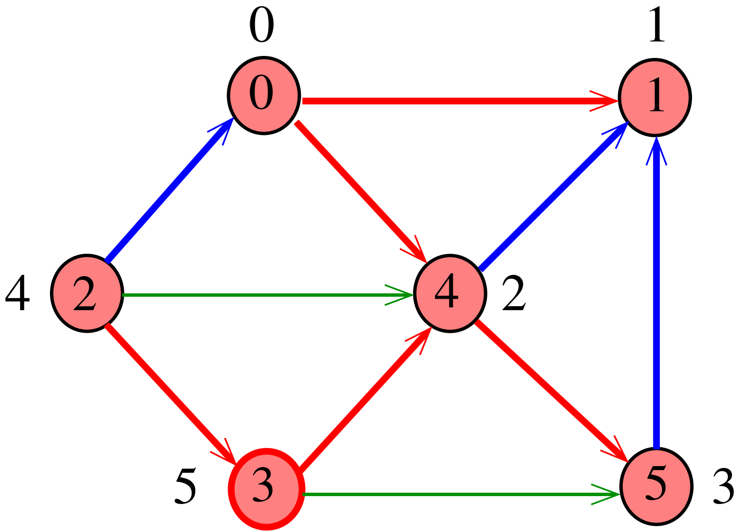
dfsR(G,2)



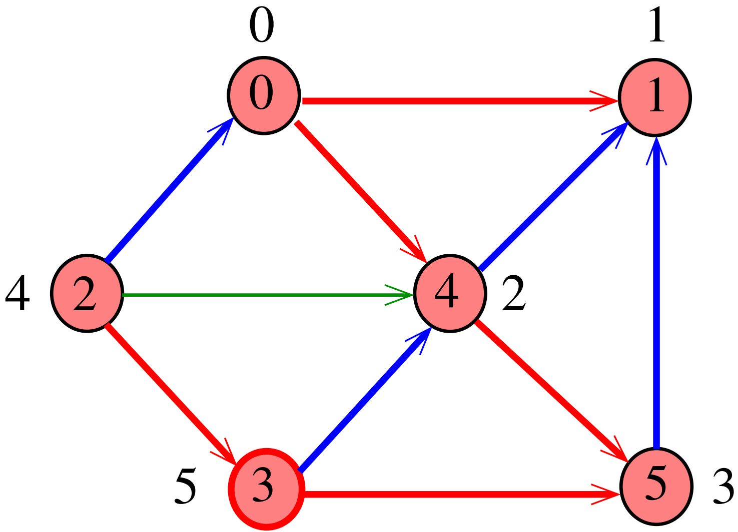
dfsR(G,3)



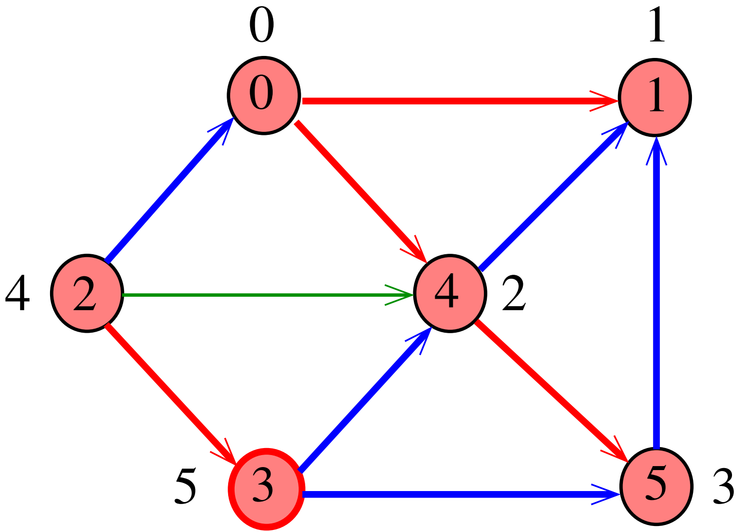
dfsR(G,3)



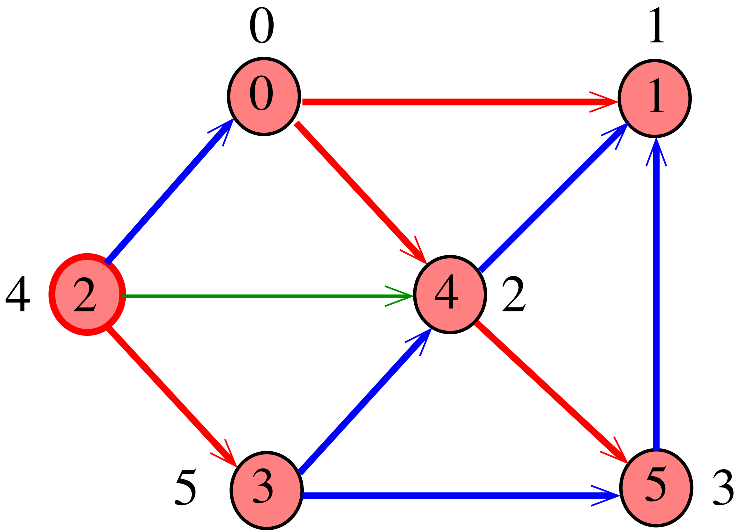
dfsR(G,3)



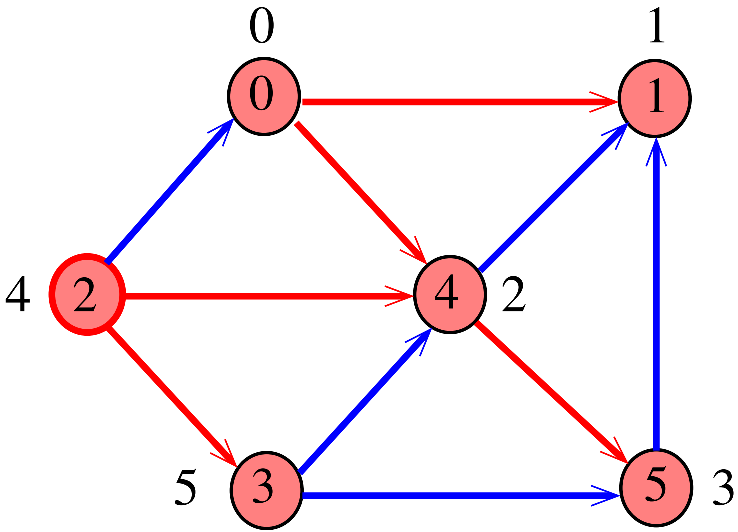
dfsR(G,3)



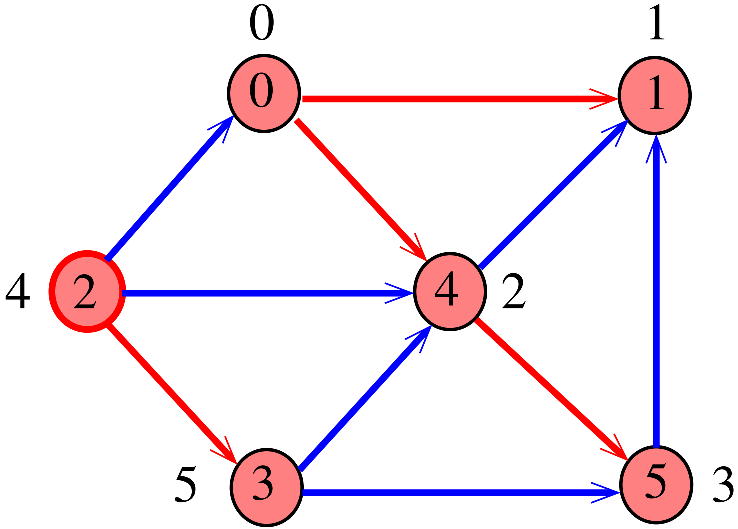
dfsR(G,2)



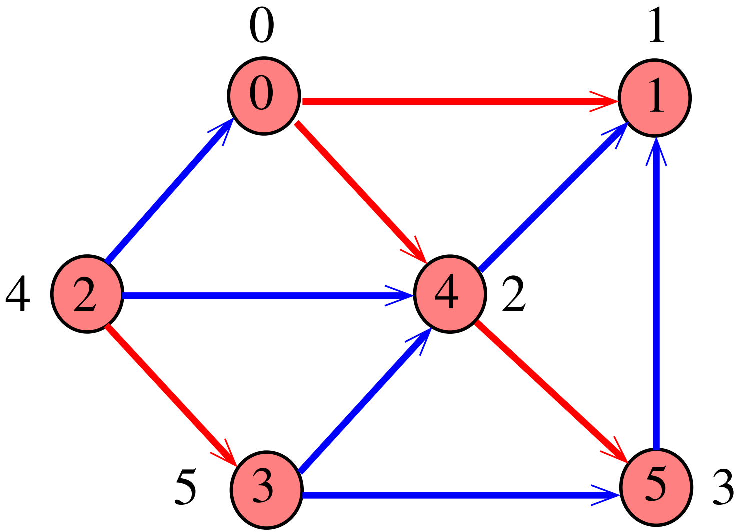
dfsR(G,2)



dfsR(G,2)



DIGRAPHdfs(G)



DIGRAPHdfs

```
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

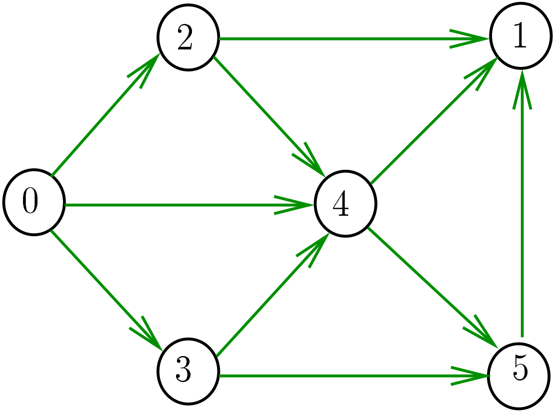
```
void dfsR (Digraph G, Vertex v) {
    Vertex w;
    lbl[v] = cnt++;
    for (w = 0; w < G->V; w++)
        if (G->adj[v][w] == 1)
            if (lbl[w] == -1)
                dfsR(G, w);
}
```

dfsR

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

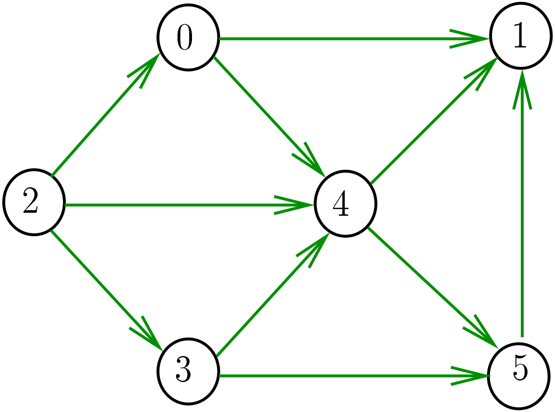
```
void dfsR (Digraph G, Vertex v) {
    link p;
    lbl[v] = cnt++;
    for (p = G->adj[v]; p != NULL; p= p->next)
        if (lbl[p->w] == -1)
            dfsR(G, p->w);
}
```

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 $\Theta(\mathtt{V}^2)$.

AULA 6

Arborescência de busca em profundidade

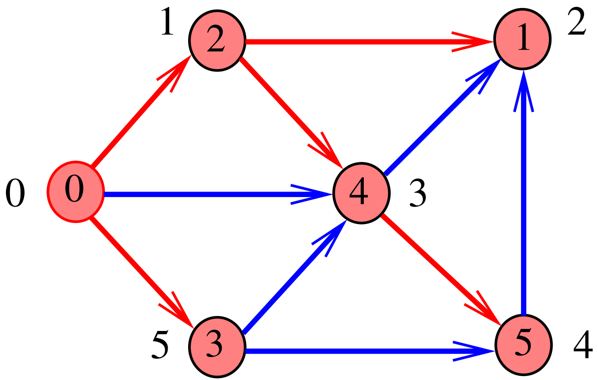
Classificação dos arcos

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

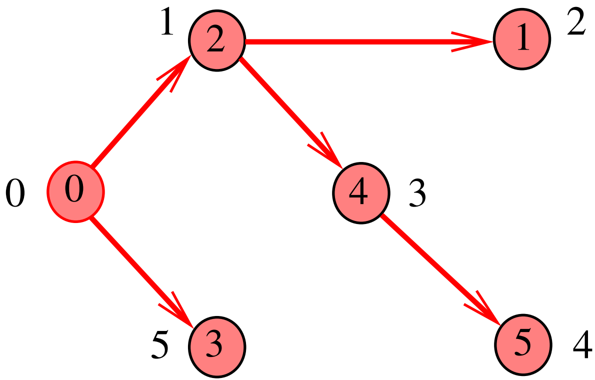


Arcos da arborescência

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

Exemplo: arcos em vermelho são arcos da

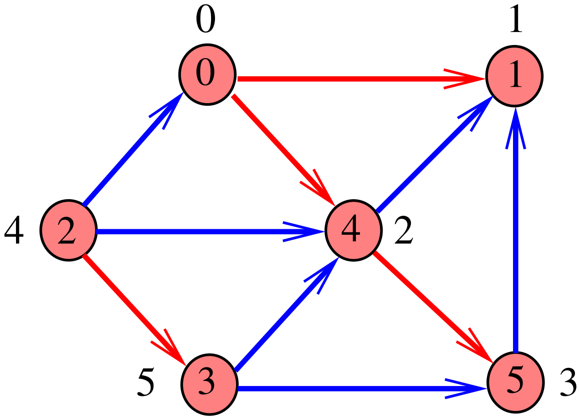
arborescência



Floresta DFS

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

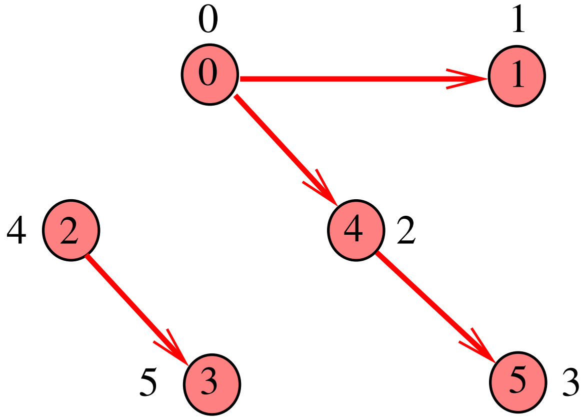
Exemplo: arcos em vermelho formam a floresta DFS



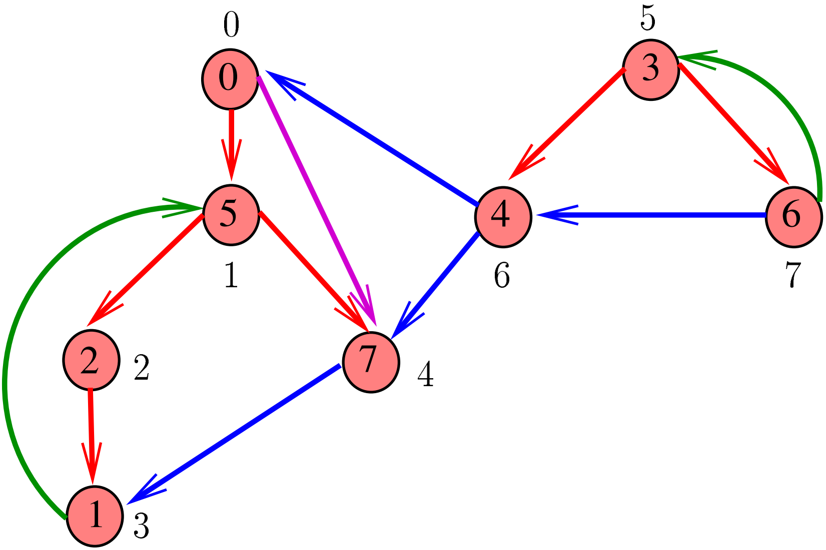
Floresta DFS

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

Exemplo: arcos em vermelho formam a floresta DFS

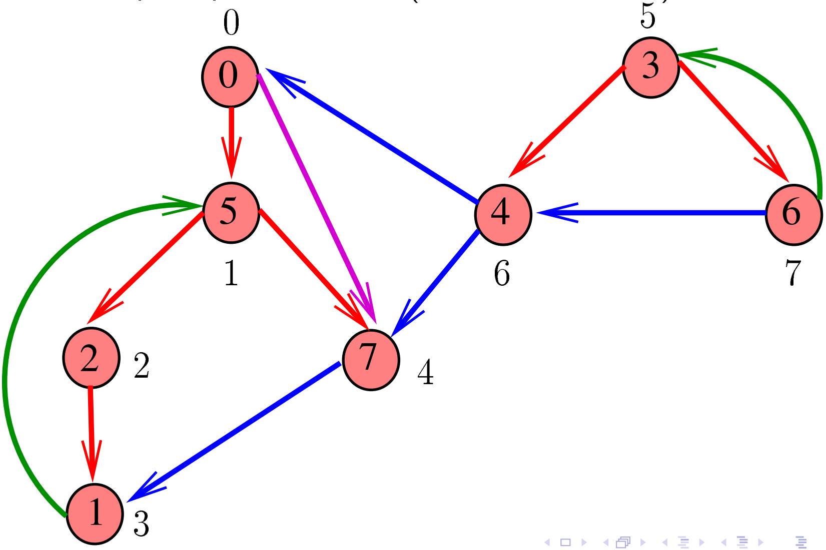


Classificação dos arcos



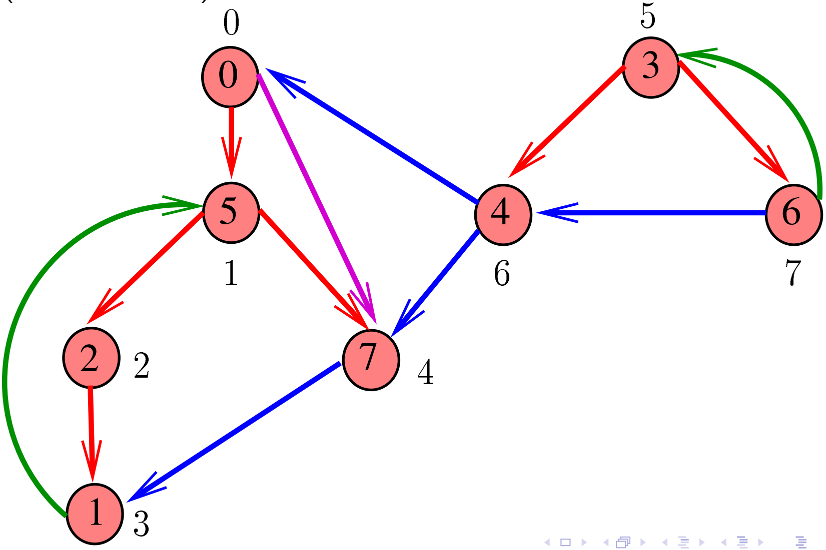
Arcos de arborescência

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



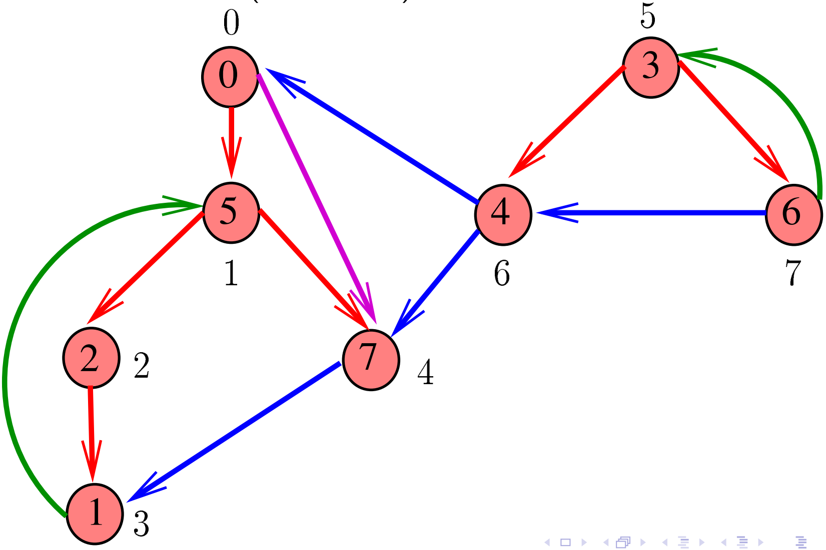
Arcos de retorno

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



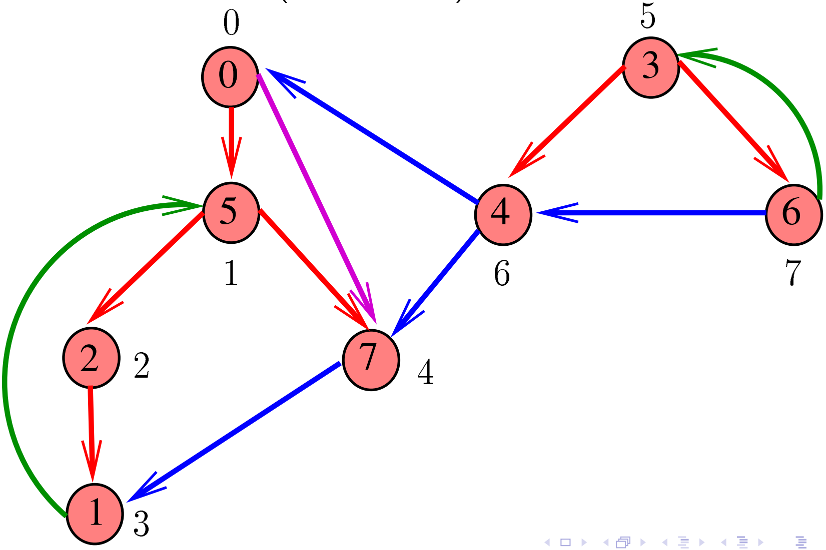
Arcos descendentes

`v-w` é **arco 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)



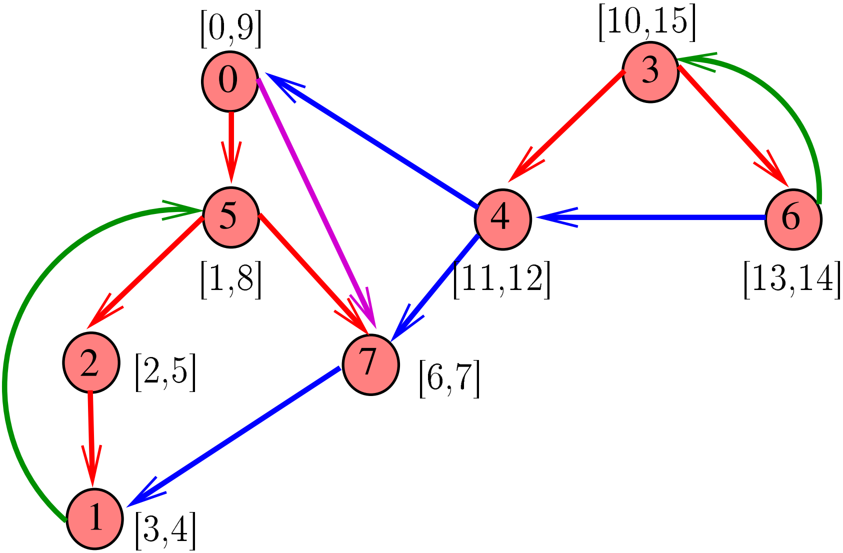
Busca DFS (CLRS)

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)



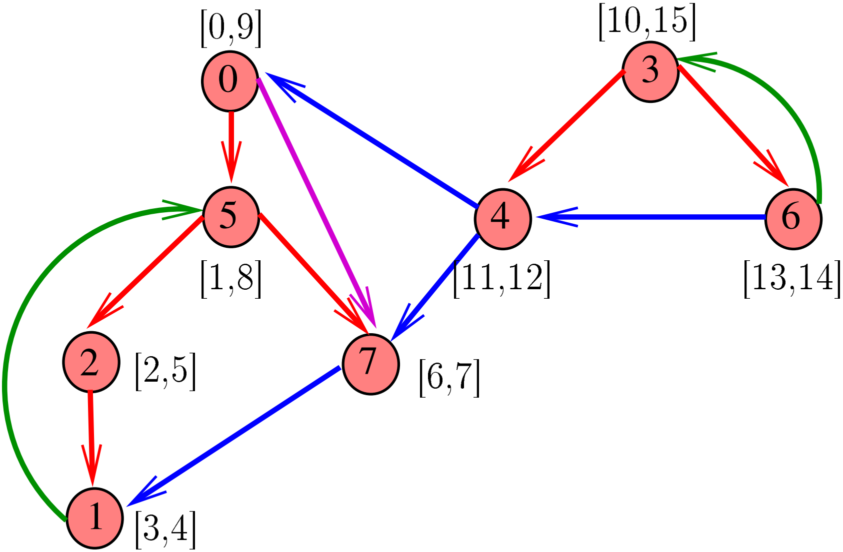
DIGRAPHdfs

```
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

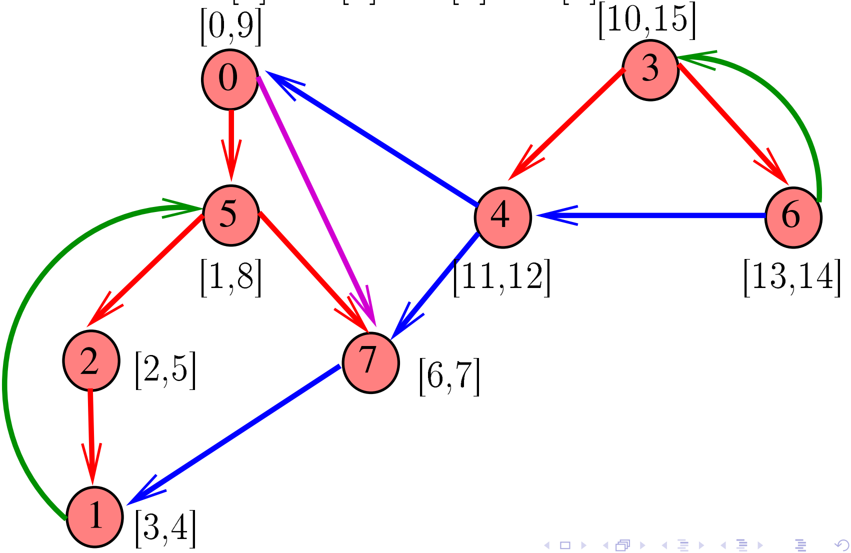
```
void dfsR (Digraph G, Vertex v) {
    link p;
    d[v] = time++;
    for (p = G->adj[v]; p != NULL; p= p->next)
        if (d[p->w] == -1) {
            parnt[w] = p->w;
            dfsR(G, p->w);
        }
    f[v] = time++;
}
```

Classificação dos arcos



Arcos de arborescência ou descendentes

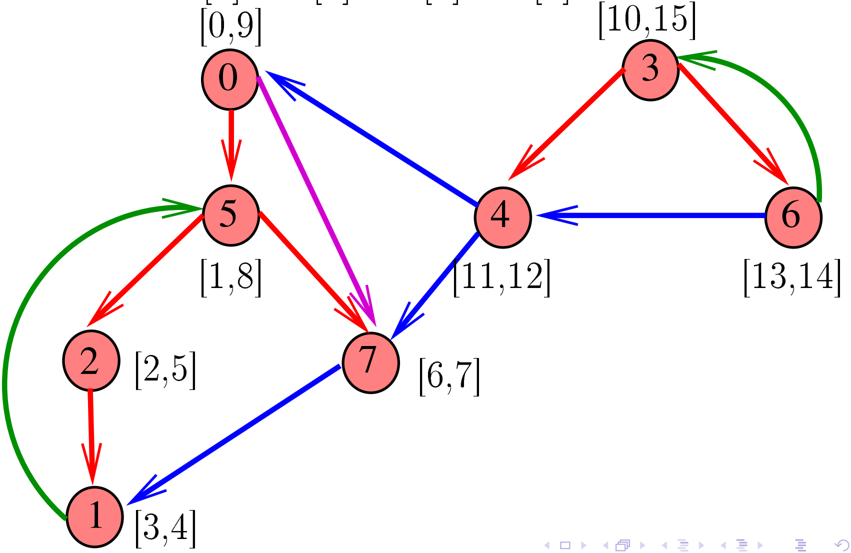
$\mathtt{v}$-$\mathtt{w}$ é **arco de arborescência** ou **descendente** se e somente se $\mathtt{d}[\mathtt{v}] < \mathtt{d}[\mathtt{w}] < \mathtt{f}[\mathtt{w}] < \mathtt{f}[\mathtt{v}]$



Arcos de retorno

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

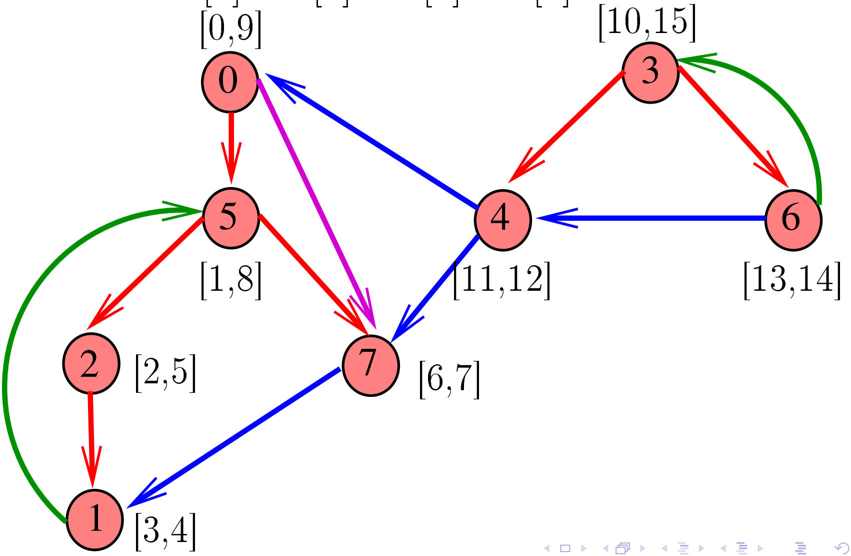
$$d[w] < d[v] < f[v] < f[w]$$



Arcos cruzados

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

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



Conclusões

v-w é:

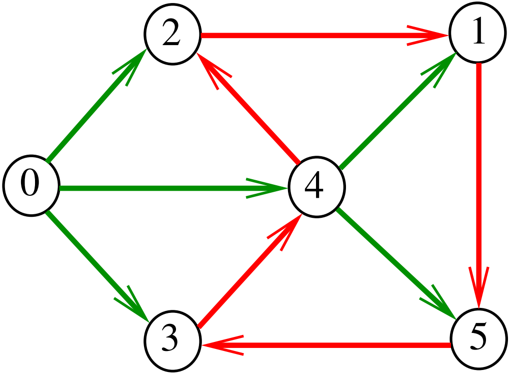
- **arco de arborescência** se e somente se
  $d[v] < d[w] < f[w] < f[v]$ e $parnt[w] = v$;
- **arco descendente** se e somente se
  $d[v] < d[w] < f[w] < f[v]$ e $parnt[w] \neq v$;
- **arco de retorno** se e somente se
  $d[w] < d[v] < f[v] < f[w]$;
- **arco cruzado** se e somente se
  $d[w] < f[w]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

Ciclos em digrafos

Ciclos

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

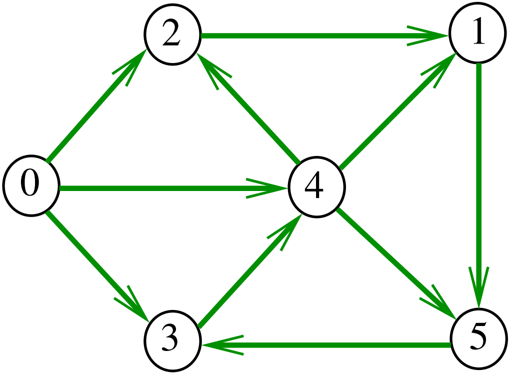
Exemplo: 2-1-5-3-4-2 é um ciclo



Procurando um ciclo

Problema: decidir se dado digrafo `G` possui um ciclo

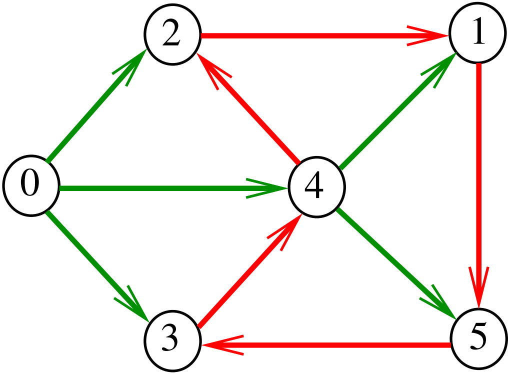
Exemplo: para o grafo a seguir a resposta é SIM



Procurando um ciclo

Problema: decidir se dado digrafo `G` possui um ciclo

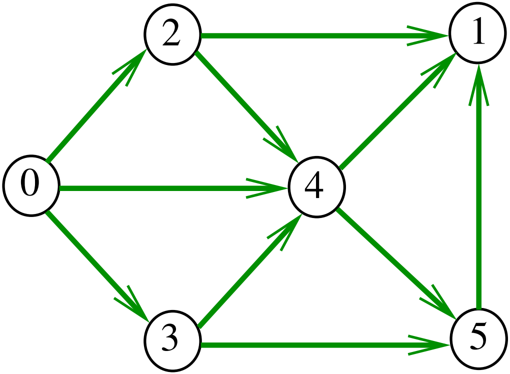
Exemplo: para o grafo a seguir a resposta é SIM



Procurando um ciclo

Problema: decidir se dado digrafo G possui um ciclo

Exemplo: para o grafo a seguir a resposta é NÃO



digraphcycle

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 digraphcycle (Digraph G);
```

Primeiro algoritmo

```
int digraphcycle (Digraph G) {
    Vertex v;
    link p;
    int output;
    for (v = 0; v< G->V; v++)
        for (p=G->adj[v];p!= NULL;p=p->next)
        {
            output = DIGRAPHpath(G, p->w, v);
            if (output == 1) return 1;
        }
    return 0;
}
```

Consumo de tempo

O consumo de tempo da função `digraphcycle` é $\mathtt{A}$ vezes o consumo de tempo da função `DIGRAPHpath`.

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

O consumo de tempo da função `digracycle` para matriz de adjacência é $\mathrm{O}(\mathtt{AV}^2)$.