

Melhores momentos

AULA 6

Busca DFS (CLRS)

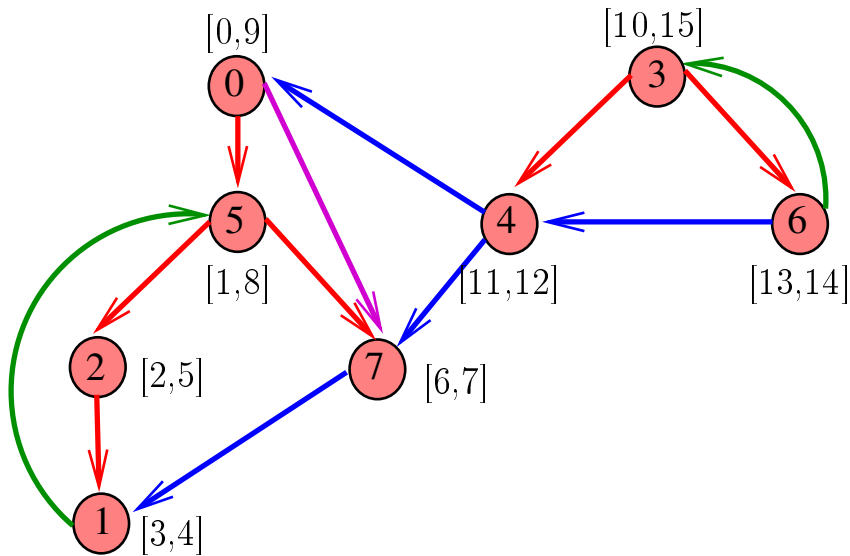
Vamos supor que nossos digrafos têm no máximo `maxV` vértices

```
#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)



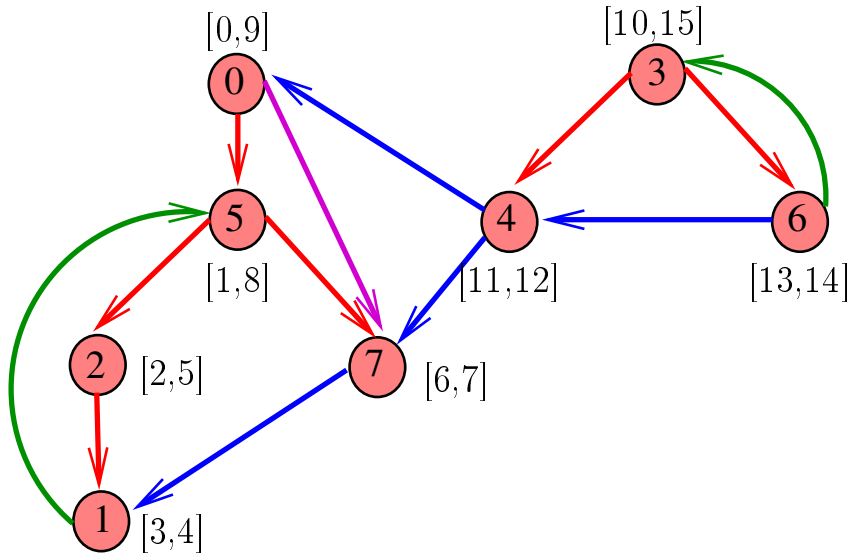
DIGRAPHdfs

```
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           parnt[v] = v;  
9           dfsR(G, v),  
        }  
}
```

dfsR

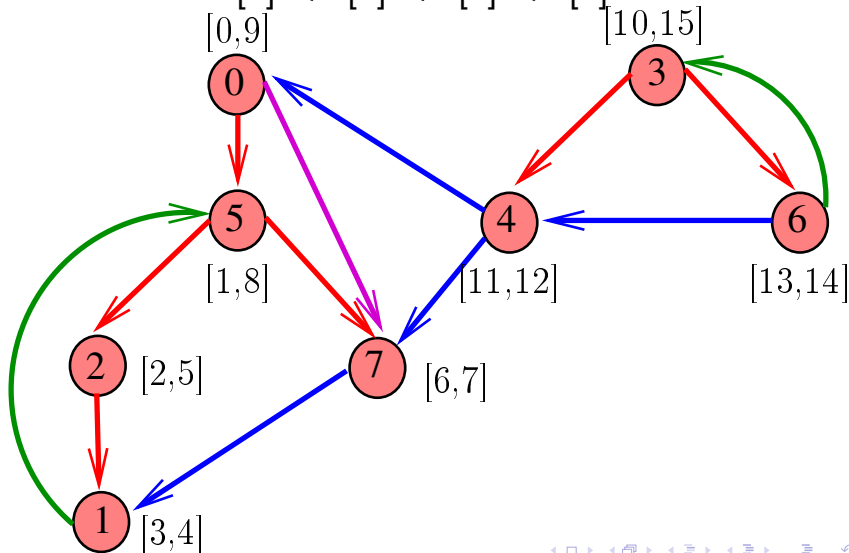
```
void dfsR (Digraph G, Vertex v) {  
    link p;  
    Vertex w;  
1    d[v] = time++;  
2    for (p = G->adj[v]; p != NULL; p = p->next)  
3        w = p->w;  
4        if (d[w] == -1) {  
5            parnt[w] = v;  
6            dfsR(G, w);  
7        }  
8    f[v] = time++;  
}
```

Classificação dos arcos



Arcos de arborescência ou descendentes

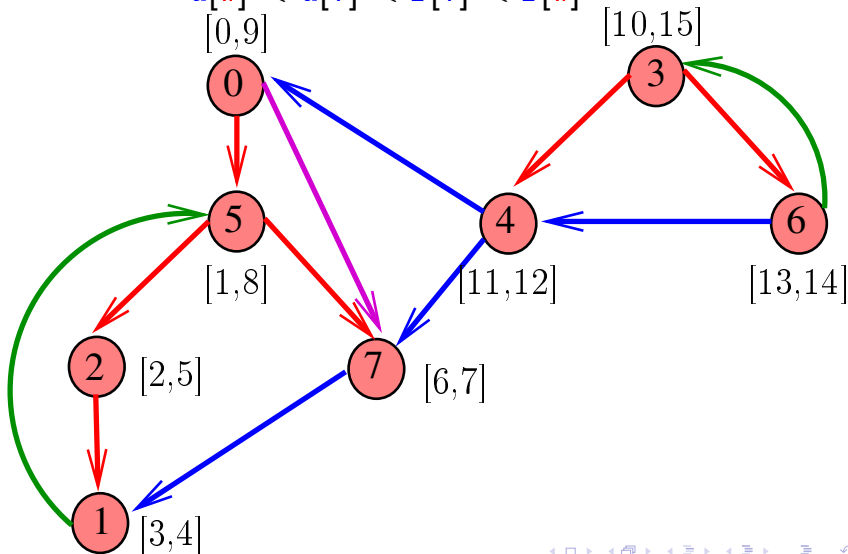
$v-w$ é **arco de arborescência** ou **descendente** se e somente se $d[v] < d[w] < f[w] < f[v]$



Arcos de retorno

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

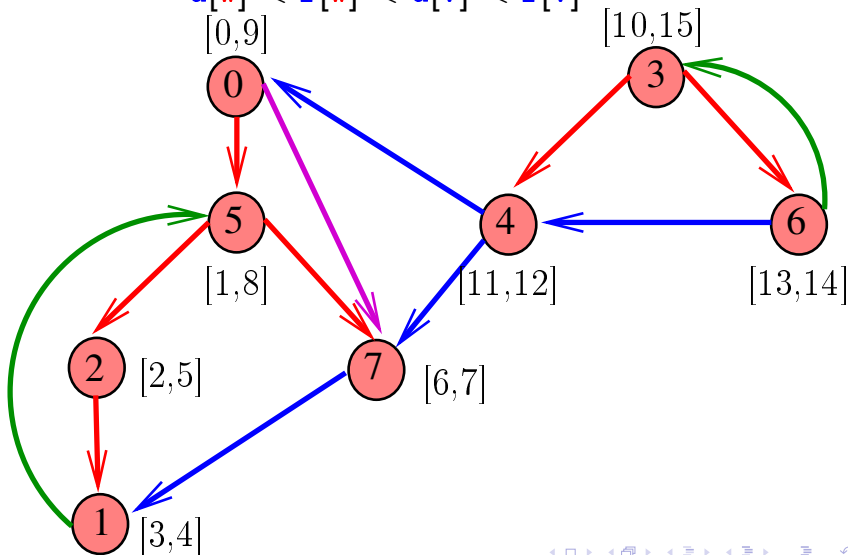
$$d[w] < d[v] < f[v] < f[w]$$



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

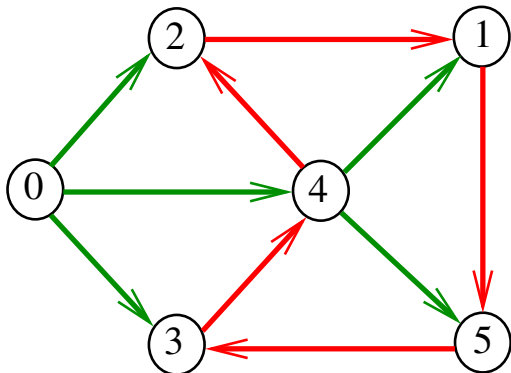
AULA 7

Ciclos em digrafos

Ciclos

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

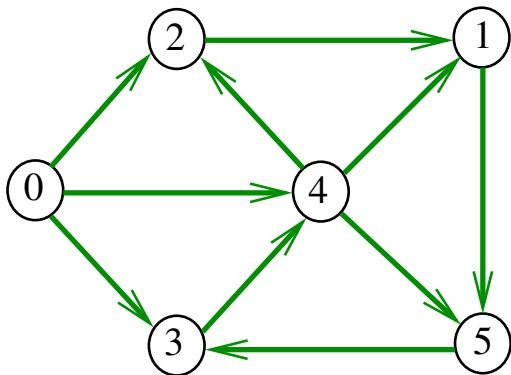
Exemplo: 2-1-5-3-4-2 é um ciclo



Procurando um ciclo

Problema: decidir se dado digrafo G possui um ciclo

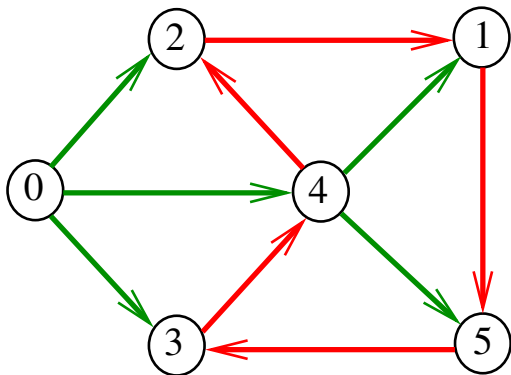
Exemplo: para o grafo a seguir a resposta é **SIM**



Procurando um ciclo

Problema: decidir se dado digrafo G possui um ciclo

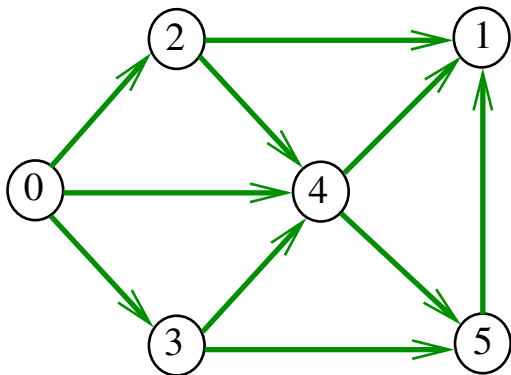
Exemplo: para o grafo a seguir a resposta é **SIM**



Procurando um ciclo

Problema: decidir se dado digrafo G possui um ciclo

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



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;  
    int output;  
1   for (v = 0; v < G->V; v++)  
2       for (p=G->adj[v]; p!= NULL; p=p->next)  
        {  
3           output = DIGRAPHpath(G, p->w, v);  
4           if (output == 1) 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 `DIGRAPHcycle1` para **matriz de adjacência** é $O(AV^2)$.

DIGRAPHcycle

Vamos supor que nossos digrafos têm no máximo **maxV** vértices

```
#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);
```

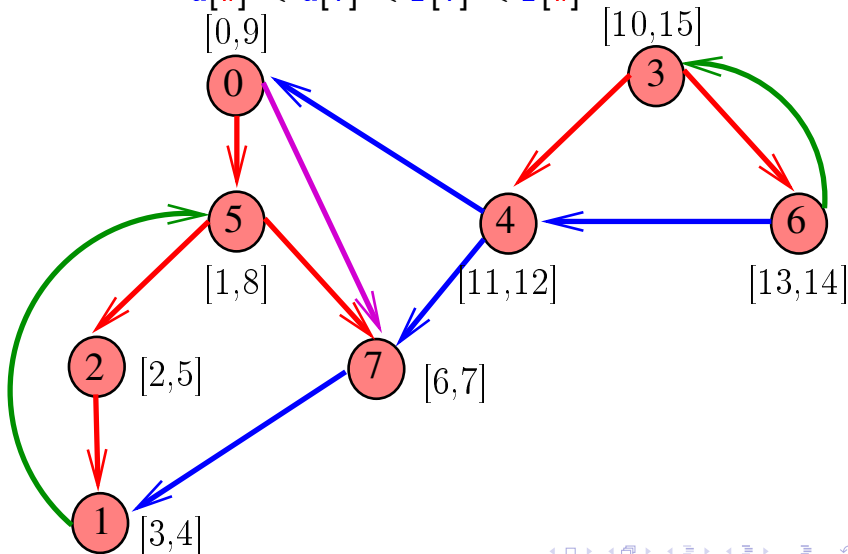
A função tem por base a seguinte observação: em relação a **qualquer** floresta de busca em profundidade,

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] = -1; parnt[v] = -1;  
4   }  
5   for (v = 0; v < G->V, v++)  
6       if (d[v] == -1) {  
7           parnt[v] = v;  
8           if (cycleR(G, v) == 1) return 1;  
        }  
9   return 0;  
}
```

cycleR

```
int cycleR (Digraph G, Vertex v) {  
    link p; Vertex w;  
1    d[v] = time++;  
2    for (p = G->adj[v]; p != NULL; p = p->next)  
3        w = p->w;  
4        if (d[w] == -1) {  
4            parnt[w] = v;  
5            if (cycleR(G, w) == 1) return 1;  
        }  
6        else if (f[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 $O(V^2)$.