

Caminhos em digrafos (continuação)

S 17.1

DIGRAPHpath

Esta versão para assim que encontra **t**

```
static int lbl[maxV];
```

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

```
{
```

```
    Vertex v;
```

```
1  for (v = 0; v < G->V; v++)
```

```
2      lbl[v] = 0;
```

```
3  return pathR(G, s, t);
```

```
}
```

DIGRAPHpath

Esta versão para assim que encontra t

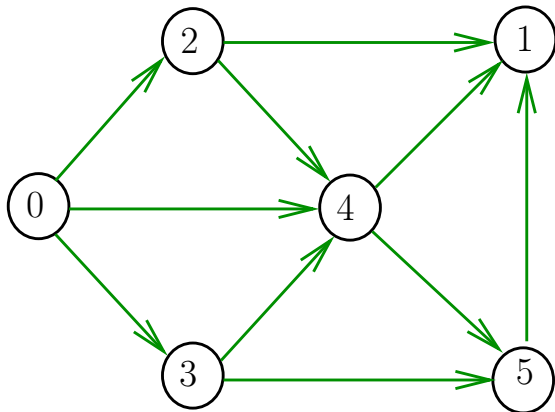
```
static int lbl[maxV];  
int DIGRAPHpath (Digraph G, Vertex s, Vertex t)  
{  
    Vertex v;  
1   for (v = 0; v < G->V; v++)  
2       lbl[v] = 0;  
3   return pathR(G, s, t);  
}
```

pathR

Para assim que encontra **t**

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

DIGRAPHpath($G,0,1$)



0-2 pathR($G,2$)

2-1 pathR($G,1$)

2-4 pathR($G,4$)

4-1

4-5 pathR($G,5$)

5-1

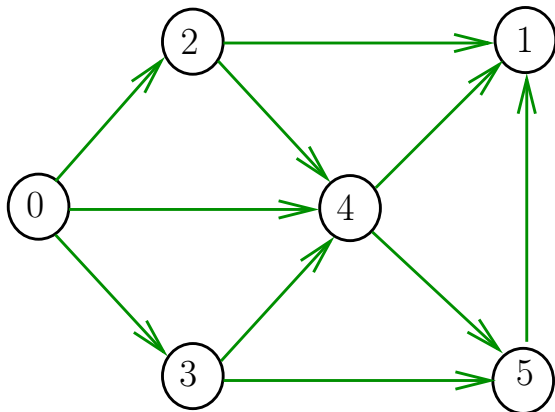
0-3 pathR($G,3$)

3-4

0-4

existe caminho

DIGRAPHpath(G,2,3)



2-1 pathR(G,1)

2-4 pathR(G,4)

4-1

4-5 pathR(G,5)

5-1

nao existe caminho

Problema

Como alterar `DIGRAPHpath` para devolver um caminho, quando existir?

Qual a ED para devolver o caminho?

Seria o caso de alterar `DIGRAPHpath` para computar um pouco mais de informação?

Problema

Como alterar `DIGRAPHpath` para devolver um caminho, quando existir?

Qual a ED para devolver o caminho?

Seria o caso de alterar `DIGRAPHpath` para computar um pouco mais de informação?

Problema

Como alterar `DIGRAPHpath` para devolver um caminho, quando existir?

Qual a ED para devolver o caminho?

Seria o caso de alterar `DIGRAPHpath` para computar um pouco mais de informação?

Certificados

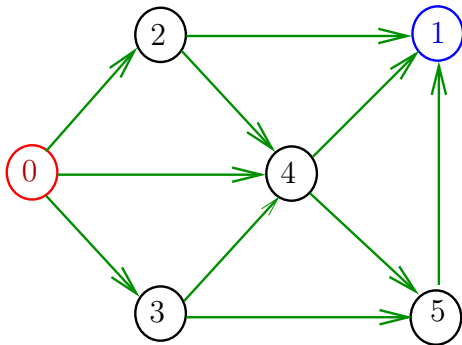
Cortes e arborescências

S páginas 84,91,92, 373

Procurando um caminho

Problema: dados um digrafo G e dois vértices s e t decidir se existe um caminho de s a t

Exemplo: para $s = 0$ e $t = 1$ a resposta é SIM



Certificados

Como é possível 'verificar' a resposta?

Como é possível 'verificar' que **existe** caminho?

Como é possível 'verificar' que **não existe** caminho?

Veremos questões deste tipo frequentemente

Elas terão um papel **suuupeer** importante no final de **MAC0338 Análise de Algoritmos**

Elas estão relacionadas com o **Teorema da Dualidade** visto em **MAC0315 Programação Linear**

Certificados

Como é possível 'verificar' a resposta?

Como é possível 'verificar' que **existe** caminho?

Como é possível 'verificar' que **não existe** caminho?

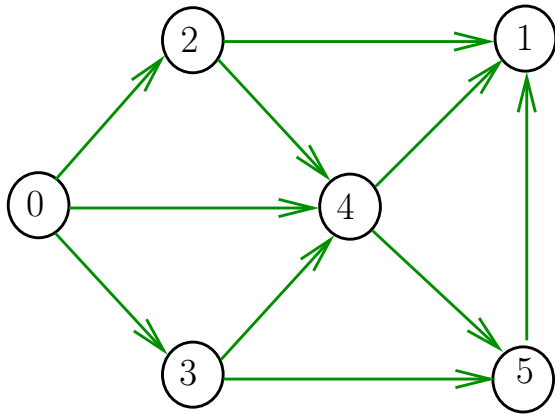
Veremos questões deste tipo freqüentemente

Elas terão um papel **suuupeer** importante no final de **MAC0338 Análise de Algoritmos**

Elas estão relacionadas com o **Teorema da Dualidade** visto em **MAC0315 Programação Linear**

Certificado de inexistência

Como é possível demonstrar que o problema não tem solução?



$\text{pathR}(G, 2)$

2-1 $\text{pathR}(G, 1)$

2-4 $\text{pathR}(G, 4)$

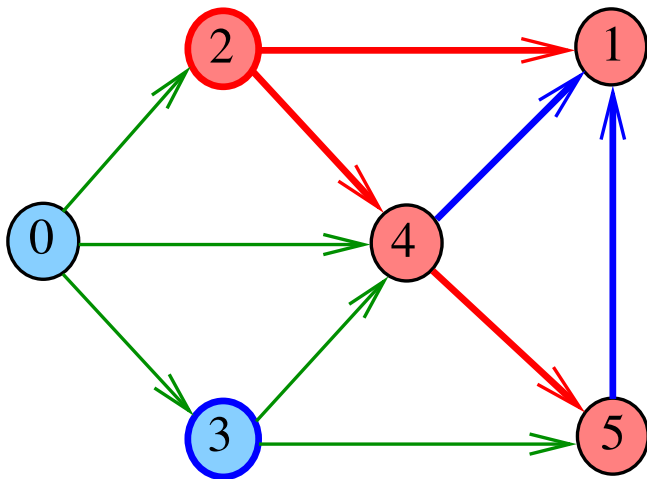
4-1

4-5 $\text{pathR}(G, 5)$

5-1

nao existe caminho

DIGRAPHpath(G, 2, 3)

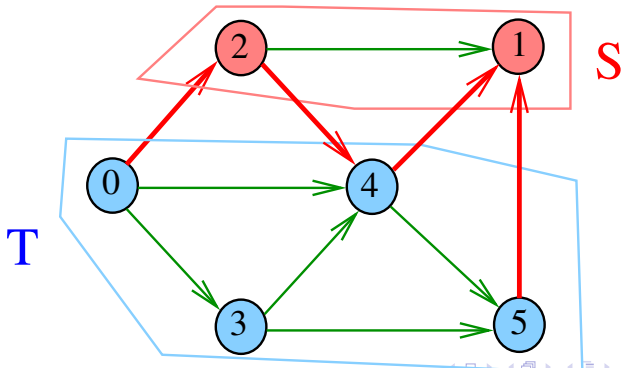


Cortes (= cuts)

Um **corte** é uma bipartição do conjunto de vértices

Um arco **pertence** ou **atravessa** um corte (S, T) se tiver uma ponta em S e outra em T

Exemplo 1: arcos em **vermelho** estão no corte (S, T)

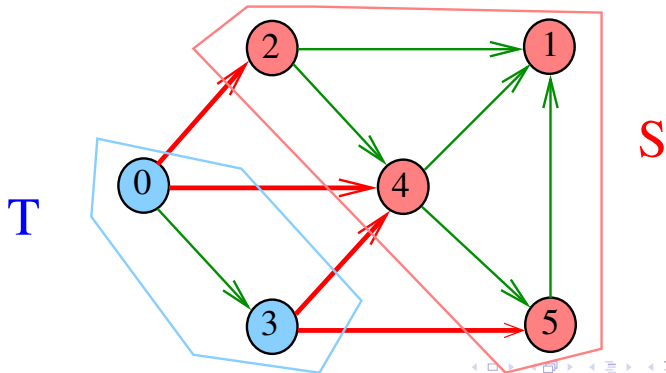


Cortes (= cuts)

Um **corte** é uma bipartição do conjunto de vértices

Um arco **pertence** ou **atravessa** um corte (S, T) se tiver uma ponta em S e outra em T

Exemplo 2: arcos em **vermelho** estão no corte (S, T)

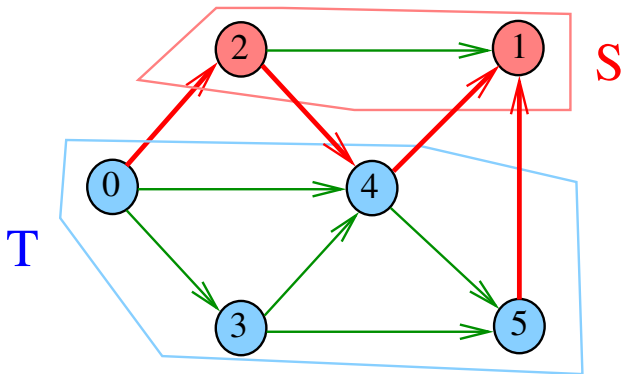


st-Cortes (= st-cuts)

Um corte (S, T) é um **st-corte** se

s está em S e t está em T

Exemplo: (S, T) é um 1-3-corte um 2-5-corte ...

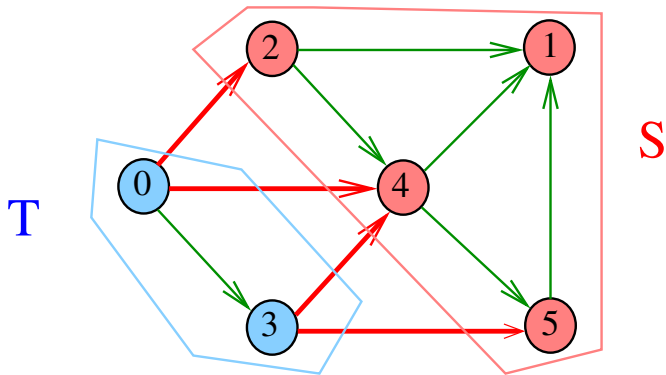


Certificado de inexistência

Para demonstrarmos que **não existe** um caminho de **s** a **t** basta exibirmos um **st**-corte (S, T) em que *todo arco* no corte tem ponta inicial em T e ponta final em S

Certificado de inexistência

Exemplo: certificado de que não há caminho de 2 a 3



st_corte

Recebe um digrafo G e vértices s e t , além do vetor $lb1$ computado pela chamada

`DIGRAPHpath(G, s, t);`

A função devolve **1** se

$$S = \{v : lb1[v] = 1\}$$

$$T = \{v : lb1[v] = 0\}$$

formam st -corte (S, T) em que todo arco no corte tem ponta inicial em T e ponta final em S ou devolve **0** em caso contrário

`int st_corte (Digraph G, Vertex s, Vertex t);`

st_corte

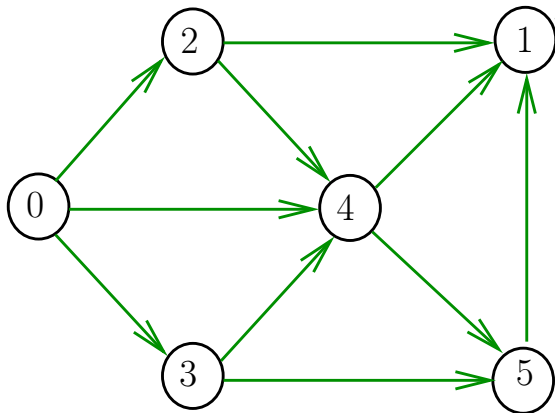
int

```
st_corte (Digraph G, Vertex s, Vertex t) {  
    Vertex v, w;  
1   if (!lbl[s] || lbl[t])  
2       return 0;  
3   for (v = 0; v < G->V; v++)  
4       for (w = 0; w < G->V; w++)  
5           if (G->adj[v][w] &&  
6               lbl[v] && !lbl[w] )  
7               return 0;  
8   return 1;  
}
```

Consumo de tempo

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

Certificado de existência



$\text{pathR}(G, 0)$

0-2 $\text{pathR}(G, 2)$

2-1 $\text{pathR}(G, 1)$

2-4 $\text{pathR}(G, 4)$

4-1

4-5 $\text{pathR}(G, 5)$

5-1

0-3 $\text{pathR}(G, 3)$

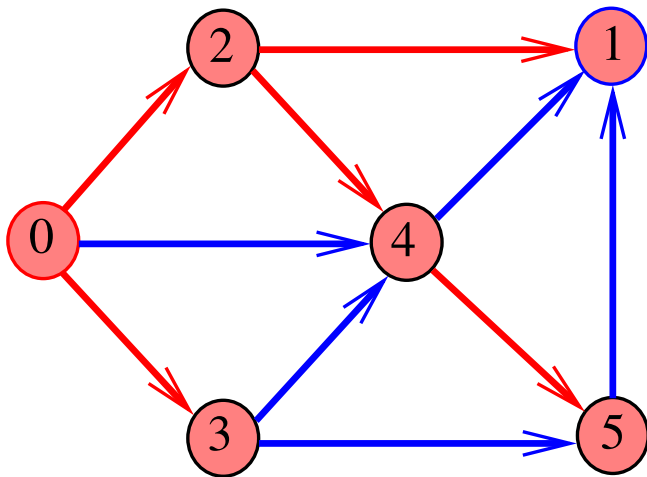
3-4

3-5

0-4

existe caminho

DIGRAPHpath($G, 0, 1$)



Caminhos no computador

Como representar **caminhos** no computador?

Caminhos no computador

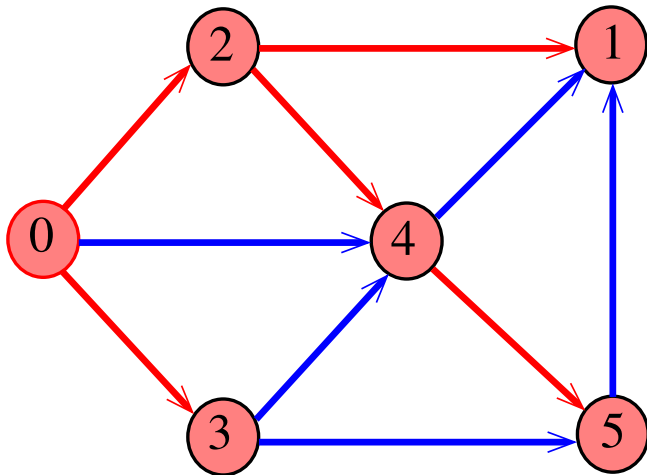
Uma maneira **compacta** de representar caminhos de um vértice a outros é uma arborescência

Uma **arborescência** é um digrafo em que

- existe exatamente um vértice com grau de entrada 0, a **raiz** da arborescência
- não existem vértices com grau de entrada maior que 1,
- cada um dos vértices é término de um caminho com origem no vértice **raiz**.

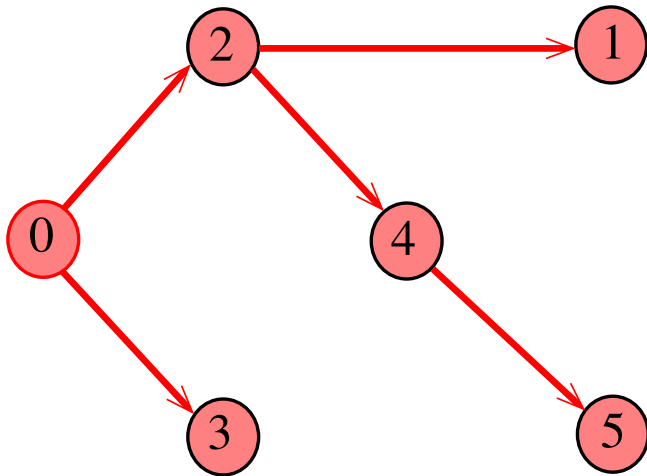
Arborescências

Exemplo: a raiz da arborescência é 0



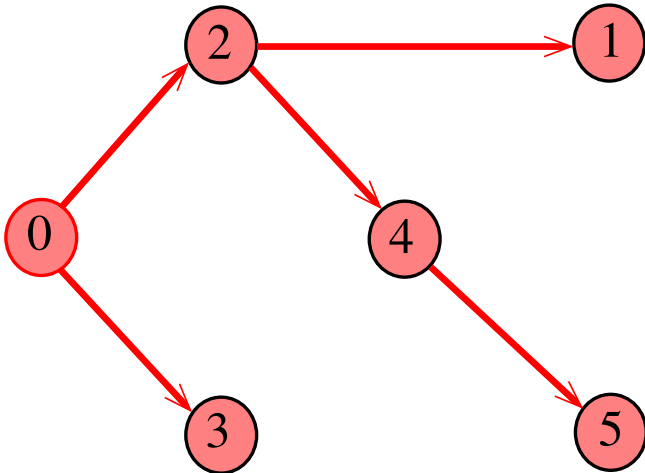
Arborescências

Exemplo: a raiz da arborescência é 0



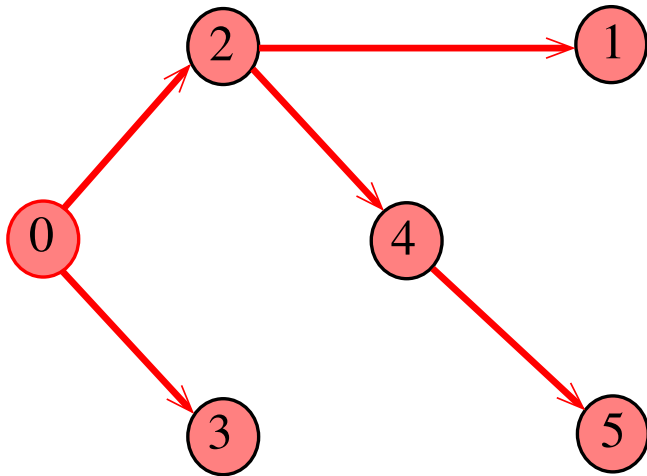
Arborescências

Propriedade: para todo vértice v , existe exatamente um caminho da raiz a v



Arborescências

Todo vértice w , exceto a raiz, tem uma **pai**: o único vértice v tal que $v-w$ é um arco

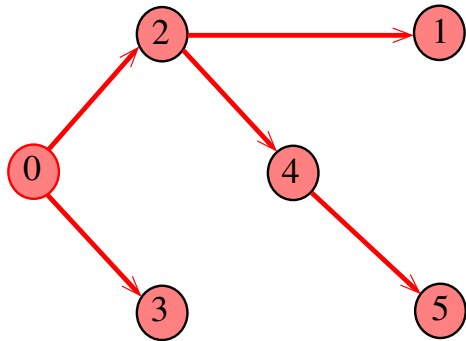


Arborescências no computador

Um arborescência pode ser representada através de um

vetor de pais: $\text{parnt}[w]$ é o pai de w

Se r é a raiz, então $\text{parnt}[r]=r$



vértice	parnt
0	0
1	2
2	0
3	0
4	2
5	4

Caminho

Dado o vetor de pais, `parnt`, de uma arborescência, é fácil determinar o caminho que leva da `raiz` a um dado vértice `v`: `basta inverter` a seqüência impressa pelo seguinte fragmento de código:

```
Vertex x;  
1  for (x = v; parnt[x] != x; x = parnt[x])  
2      printf("%d-", x);  
3  printf("%d", x);
```

Caminho

Dado o vetor de pais, `parnt`, de uma arborescência, é fácil determinar o caminho que leva da `raiz` a um dado vértice `v`: `basta inverter` a seqüência impressa pelo seguinte fragmento de código:

```
Vertex x;  
1  for (x = v; parnt[x] != x; x = parnt[x])  
2      printf("%d-", x);  
3  printf("%d", x);
```

DIGRAPHpath

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

pathR

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

st_caminho

Recebe um digrafo G e vértices s e t , além do vetor `parnt` computado pela chamada

```
DIGRAPHpath(G, s, t);
```

A função devolve **1** se

t -`parnt`[t]-`parnt`[`parnt`[t]]-...

é o reverso de um caminho de s a t em G ou devolve **0** em caso contrário

```
int st_caminho (Digraph G, Vertex s, Vertex t);
```

st_caminho

int

```
st_caminho (Digraph G, Vertex s, Vertex t) {  
    Vertex v, w;  
1    if (parnt[t] == -1 || parnt[s] != s)  
2        return 0;  
3    for (w = t; w != s; w = v) {  
4        v = parnt[w];  
5        if (!G->adj[v][w]) return 0;  
    }  
6    return 1;  
}
```

Consumo de tempo

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

linha	número de execuções da linha	
1	$= 1$	$= \Theta(1)$
2	≤ 1	$= O(1)$
3	$\leq V$	$= O(V)$
4	$\leq V$	$= O(V)$
5	$\leq V$	$= O(V)$
6	≤ 1	$= O(1)$

$$\begin{aligned} \text{total} &= \Theta(1) + 2 O(1) + 3 O(V) \\ &= O(V) \end{aligned}$$

Consumo de tempo

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

linha	número de execuções da linha	
1	$= 1$	$= \Theta(1)$
2	≤ 1	$= O(1)$
3	$\leq V$	$= O(V)$
4	$\leq V$	$= O(V)$
5	$\leq V$	$= O(V)$
6	≤ 1	$= O(1)$

$$\begin{aligned} \text{total} &= \Theta(1) + 2 O(1) + 3 O(V) \\ &= O(V) \end{aligned}$$

Consumo de tempo

O consumo de tempo da função `st_caminho` é $O(V)$.

Conclusão

Para quaisquer vértices s e t de um digrafo, vale uma e apenas uma das seguintes afirmações:

- existe um caminho de s a t
- existe st -corte (S, T) em que todo arco no corte tem ponta inicial em T e ponta final em S .