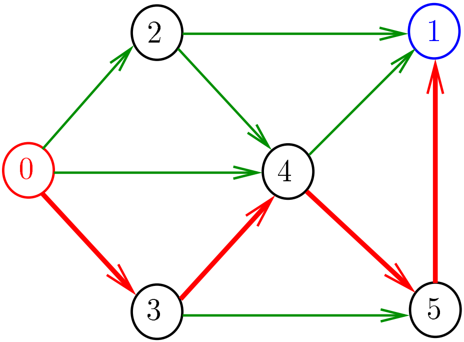
Melhores momentos

AULA 3

Procurando um caminho

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

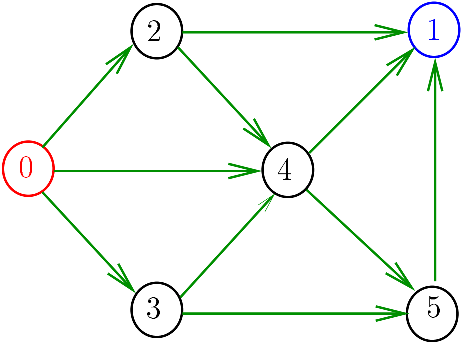
Exemplo: para s = 0 e t = 1 a resposta é SIM



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 $= 5$ e t $= 4$ a resposta é NÃO



DIGRAPHpath

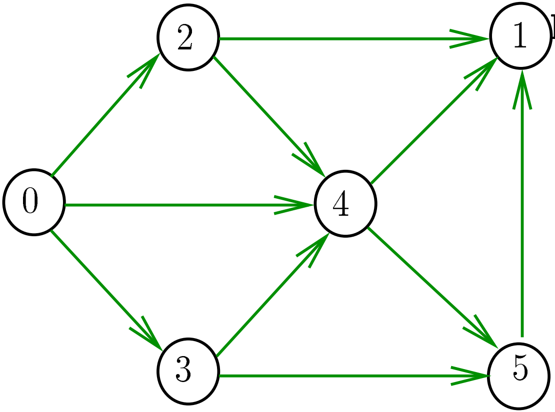
```
static int lbl[maxV];
int DIGRAPHpath (Digraph G, Vertex s, Vertex t)
{
    Vertex v;
    for (v = 0; v < G->V; v++)
        lbl[v] = -1;
    pathR(G,s);
    if (lbl[t] == -1) return 0;
    else return 1;
}
```

pathR

Visita todos os vértices que podem ser atingidos a partir de v

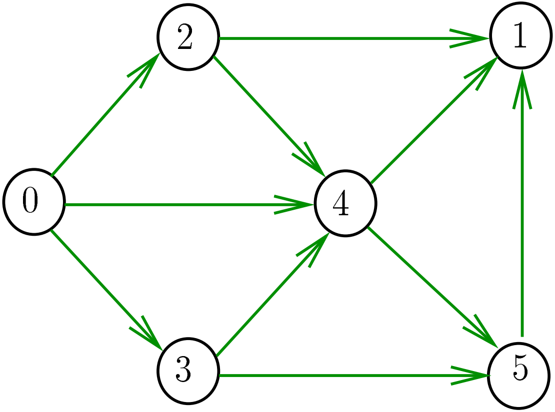
```
void pathR (Digraph G, Vertex v)
{
    Vertex w;
    lbl[v] = 0;
    for (w = 0; w < G->V; w++)
        if (G->adj[v][w] == 1)
            if (lbl[w] == -1)
                pathR(G, w);
}
```

DIGRAPHpath(G,0,1)




```
pathR(G,0)
  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)



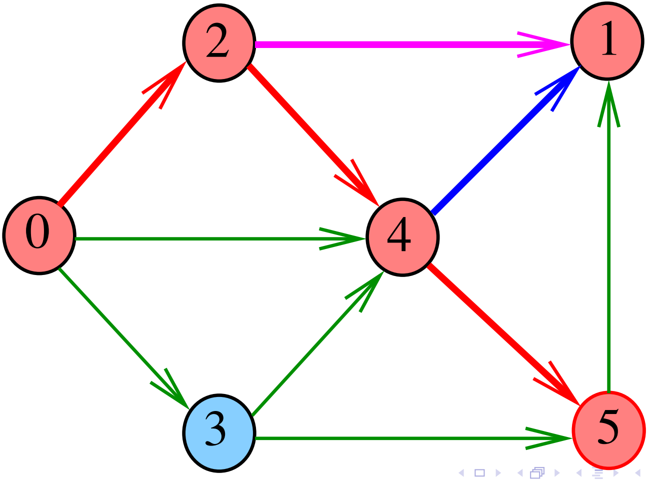

```
pathR(G,2)
  2-1 pathR(G,1)
  2-4 pathR(G,4)
    4-1
    4-5 pathR(G,5)
      5-1
nao existe caminho
```

DIGRAPHpath (versão iterativa)

Relação invariante chave: no início de cada iteração

`caminho[0]-caminho[1]-...-caminho[k-1]`

é um caminho de `s` a `v`.



Certificados

Como é possível 'verificar' a resposta?

Como é possível 'verificar' que existe caminho?

Como é possível 'verificar' que não existe caminho?

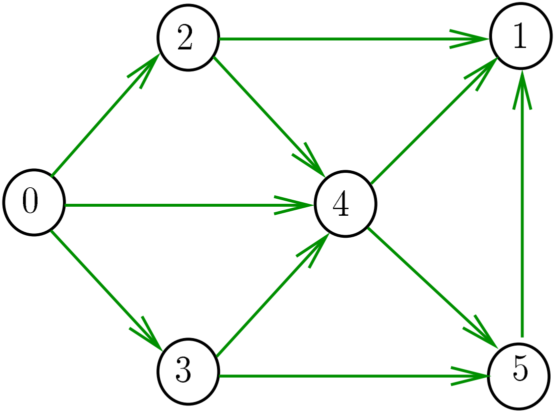
Veremos questões deste tipo freqüentemente

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

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

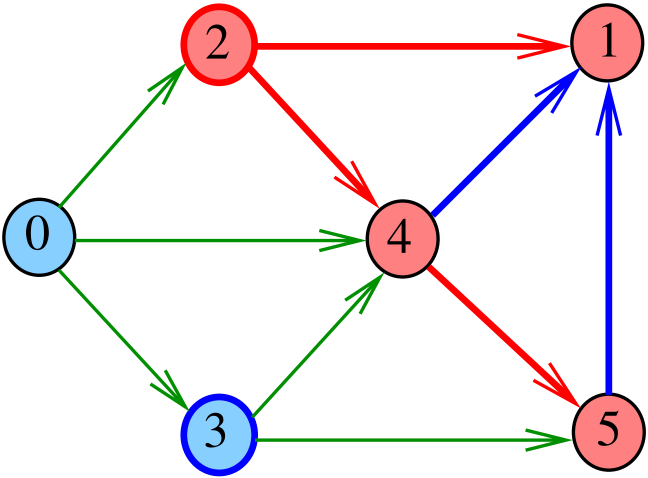
Certificado de inexistênncia

Como é possível demonstrar que **não existe** caminho de 2 a 3?




```
pathR(G,2)
  2-1 pathR(G,1)
  2-4 pathR(G,4)
    4-1
    4-5 pathR(G,5)
      5-1
nao existe caminho
```

DIGRAPHpath(G,2,3)

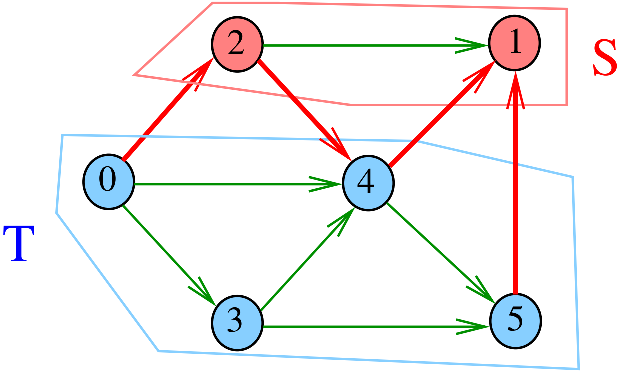


Cortes (= *cuts*)

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)

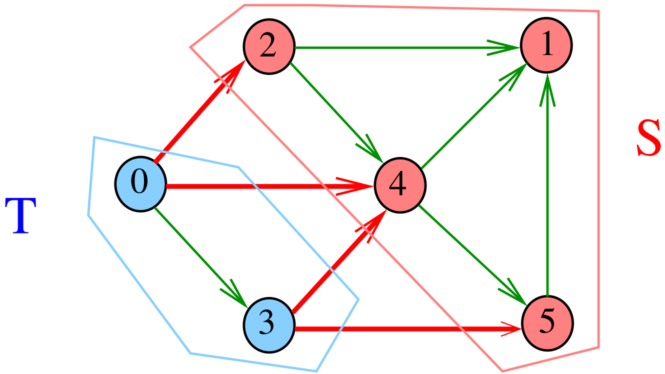


Cortes (= *cuts*)

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)

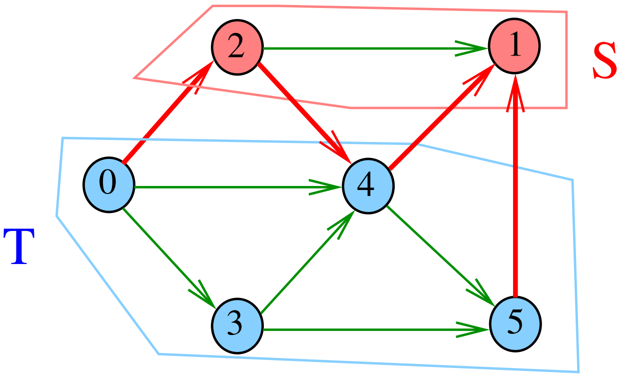


st-Cortes (= st-*cuts*)

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 $\mathtt{s}$ a $\mathtt{t}$ basta exibirmos um $\mathtt{st}$-corte $(\mathtt{S}, \mathtt{T})$ em que

> ***todo arco*** *no corte tem ponta inicial em* T *e ponta final em* S

AULA 4

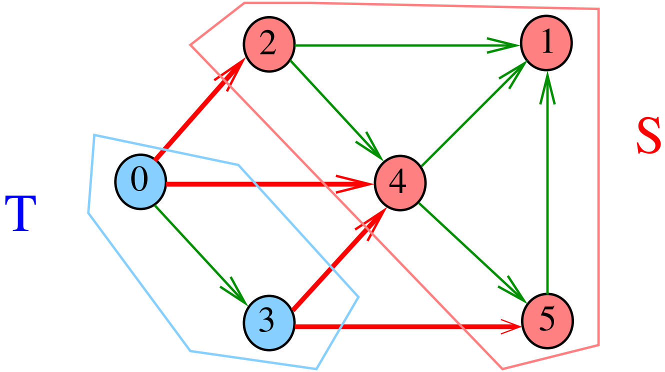
Certificados (continuação)

Cortes e arborescências

S páginas 84,91,92, 373

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 lbl computado pela chamada

DIGRAPHpath(G, s,t);

A função devolve **1** se

$$S = \{v : \texttt{lbl}[v] = 0\}$$
$$T = \{v : \texttt{lbl}[v] = -1\}$$

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);
```

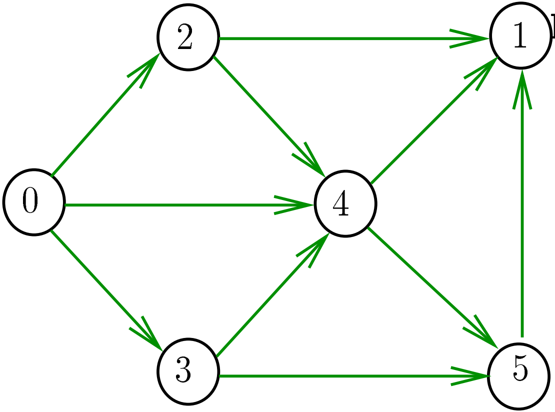
st_corte

```
int
st_corte (Digraph G, Vertex s, Vertex t) {
    Vertex v, w;
    if (lbl[s] == -1 || lbl[t] == 0)
        return 0;
    for (v = 0; v < G->V; v++)
        for (w = 0; w < G->V; w++)
            if (G->adj[v][w] == 1 &&
                (lbl[v] == 0 && lbl[w] == -1 ))
                return 0;
    return 1;
}
```

Consumo de tempo

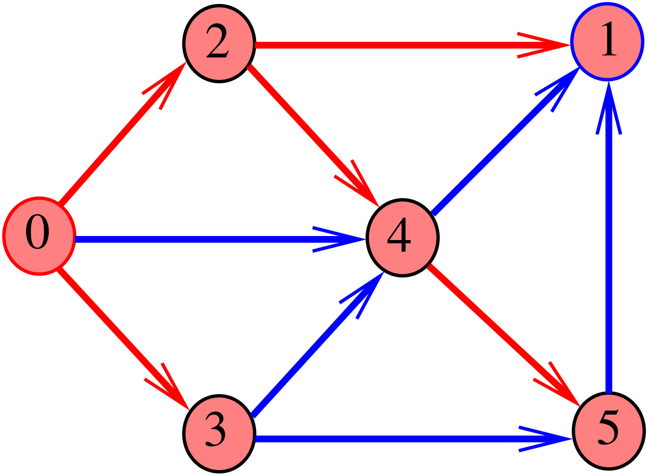
O consumo de tempo da função `st_corte` para matriz de adjacência é $O(V^2)$.

Certificado de existência




```
pathR(G,0)
  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
    3-5
  0-4
existe caminho
```

DIGRAPHpath(G,0,1)



Caminhos no computador

Como representar caminhos no computador?

Caminhos no computador

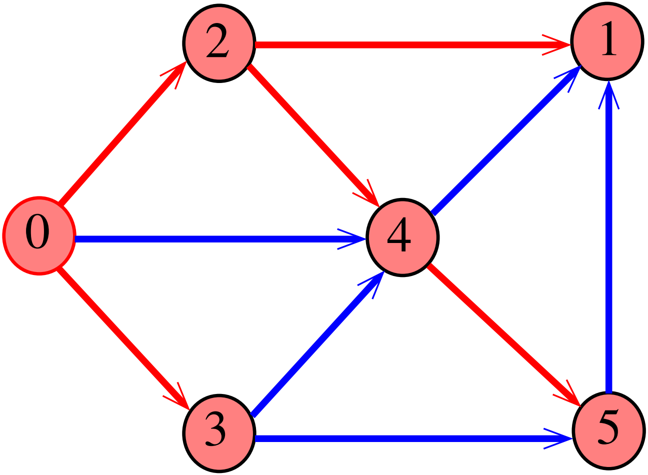
Uma maneira **compacta** de representar caminhos de um vértice a outros é uma arborescência

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**.

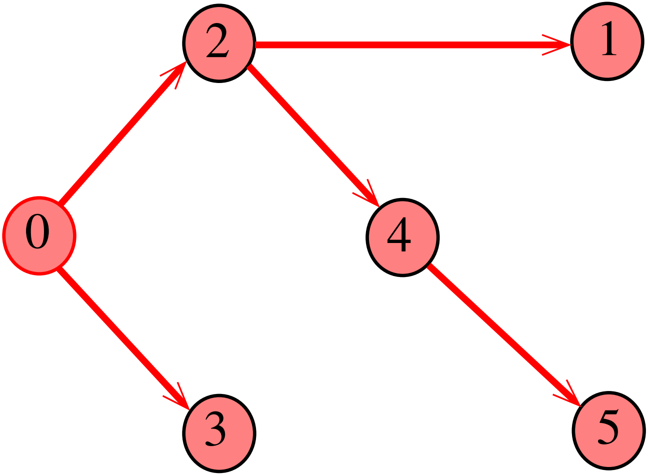
Arborescências

Exemplo: a raiz da arborescência é 0



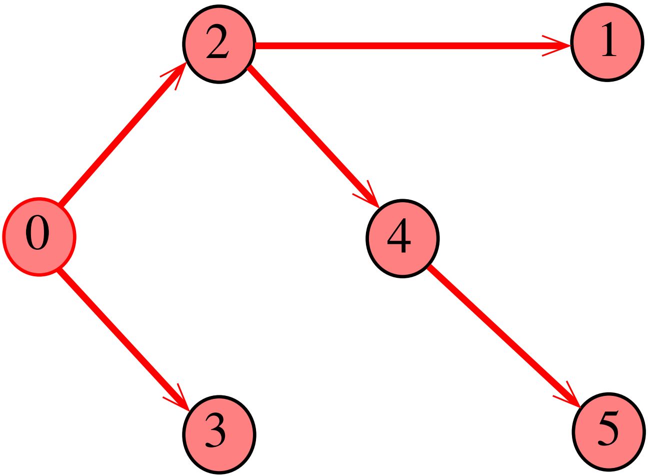
Arborescências

Exemplo: a raiz da arborescência é 0



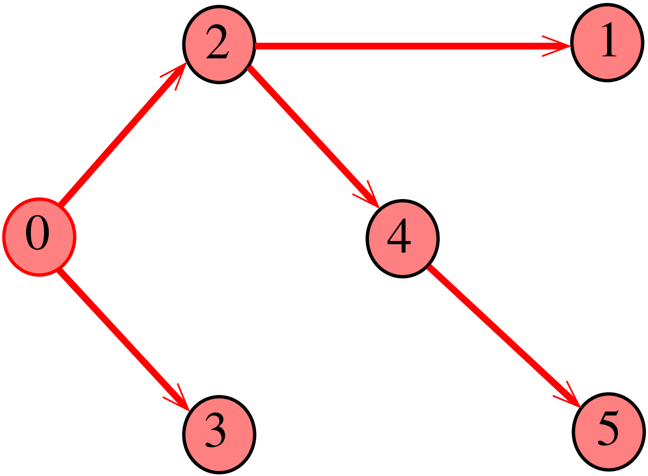
Arborescências

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



Arborescências

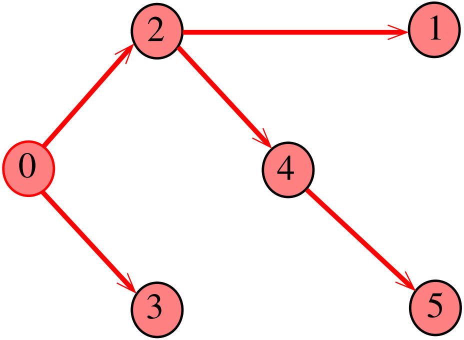
Todo vértice `w`, exceto a raiz, tem uma **pai**: o único vértice `v` tal que `v-w` é um arco



Arborescências no computador

Um arborência pode ser representada através de um **vetor de pais**: parnt[w] é o pai de w

Se r é a raiz, então 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:

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

DIGRAPHpath

```
static int lbl[maxV], 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);
            }
}
```

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;
    if (parnt[t] == -1) return 0;
    for (w = t; w != parnt[w]; w= v) {
        v= parnt[w];
        if (G->adj[v][w] != 1) return 0;
    }
    if (w != s) return 0;;
    return 1;
}
```

Consumo de tempo

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

Consumo de tempo

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

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

total $= \Theta(1) + 2\,\mathrm{O}(1) + 3\,\mathrm{O}(\mathtt{V})$
$= \mathrm{O}(\mathtt{V})$

Consumo de tempo

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

Conclusão

Para quaisquer vértices $\mathtt{s}$ e $\mathtt{t}$ de um digrafo, vale uma e apenas umas das seguintes afirmações:

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