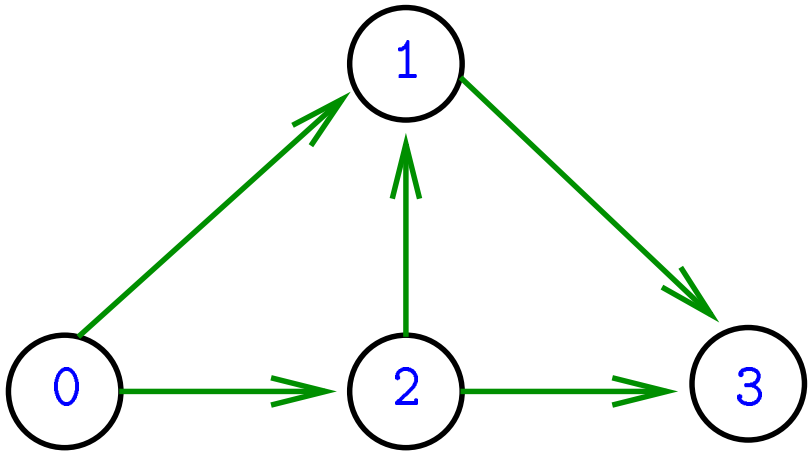
Vetor de listas de adjacência

S 17.4

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:
```

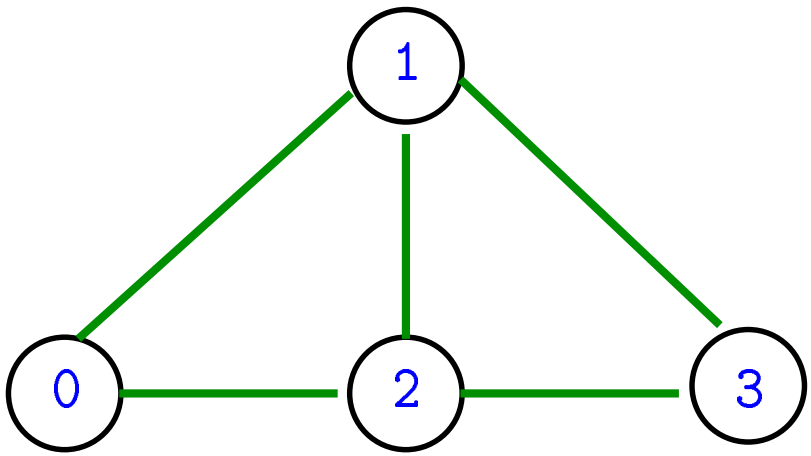
Consumo de espaço: $\Theta(V + A)$ (linear)

Manipulação eficiente

Vetor de lista de adjacência de grafos

Na representação de um grafo através de **listas de adjacência** tem-se, para cada vértice v, uma lista dos vértices que são pontas de arestas incidentes a v

Exemplo:



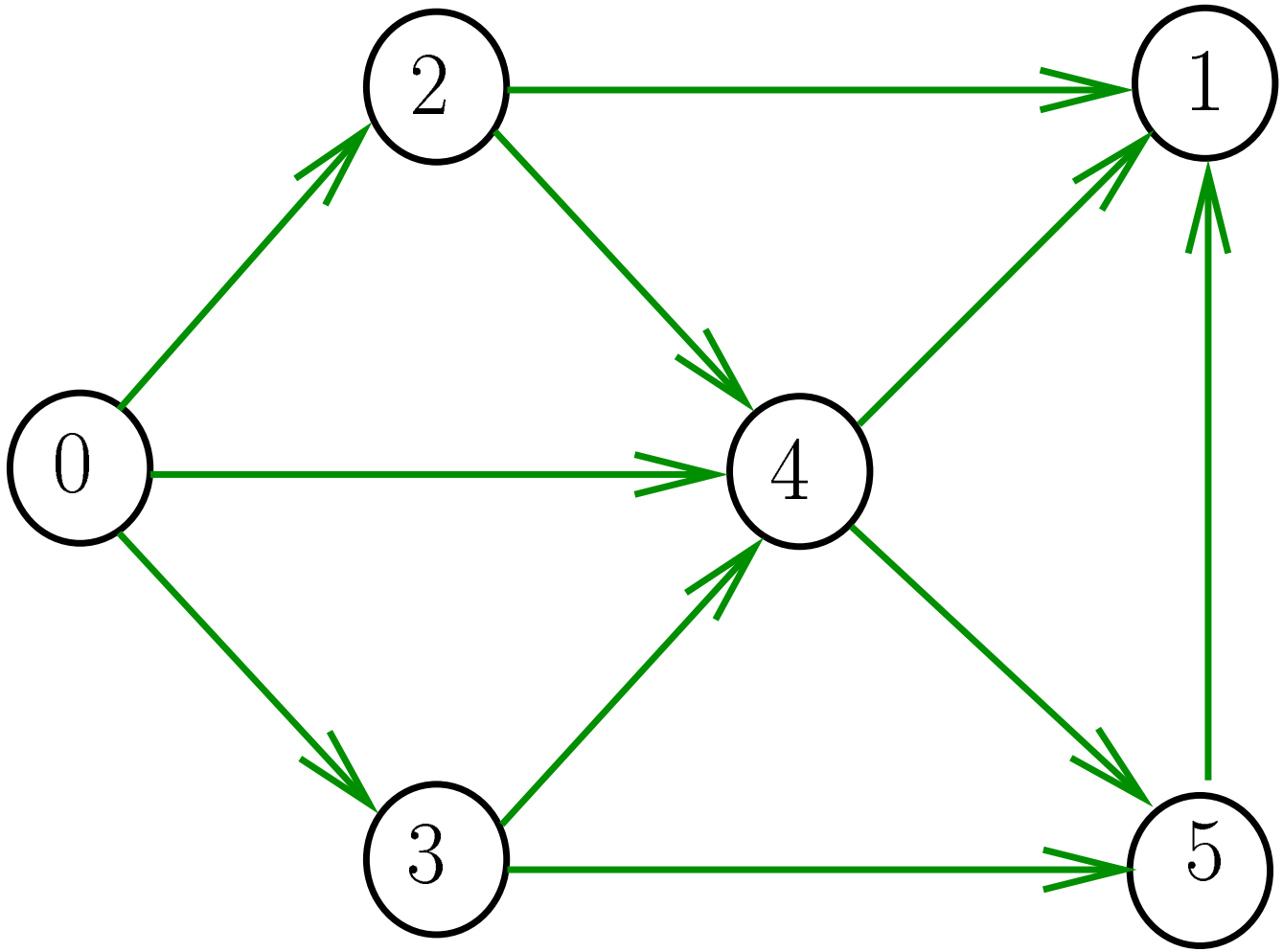

```
0:  1,  2
1:  3,  0,  2
2:  1,  3,  0
3:  1,  2
```

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

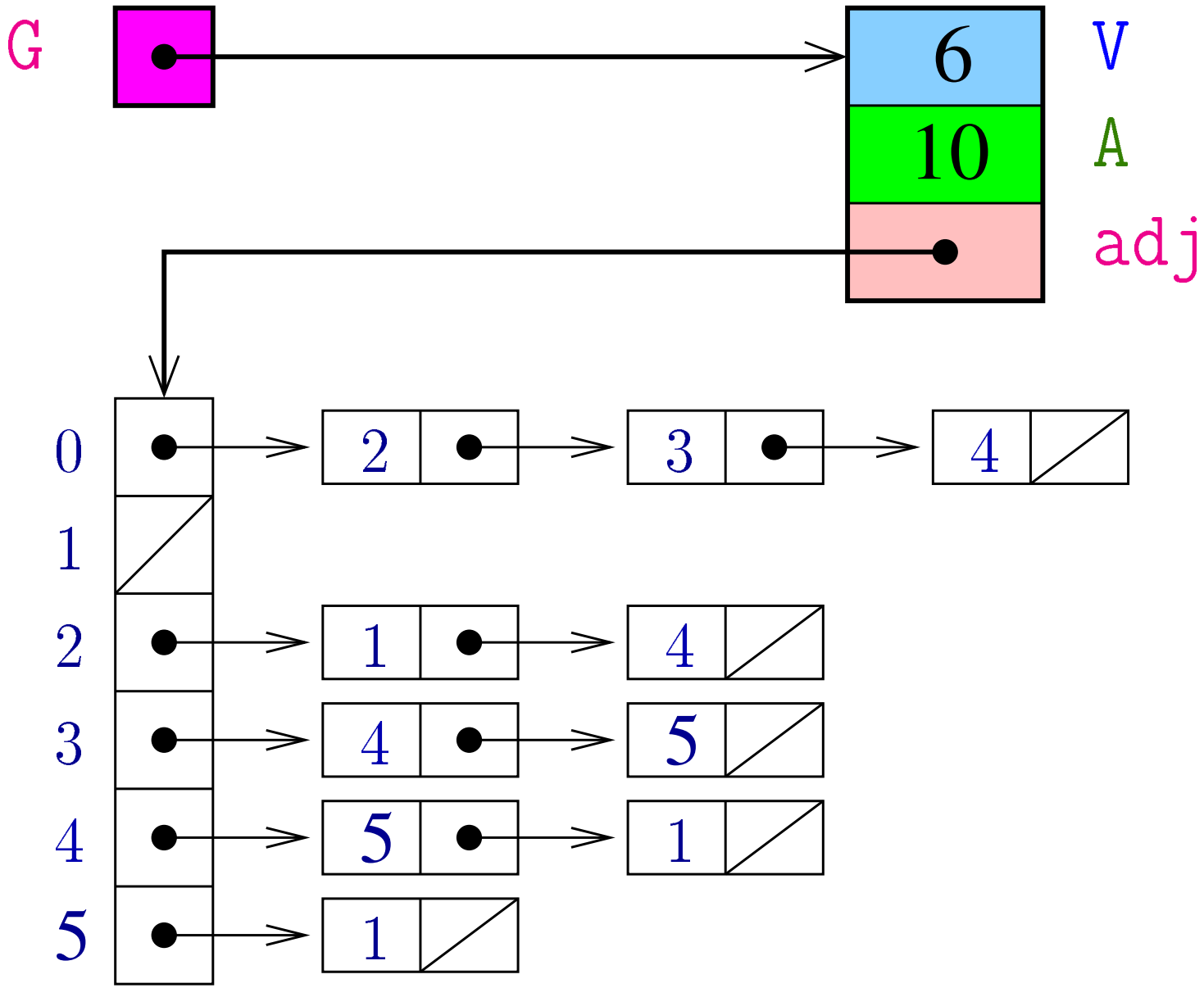
Manipulação eficiente

Digrafo

Digraph G



Estruturas de dados



Certificados

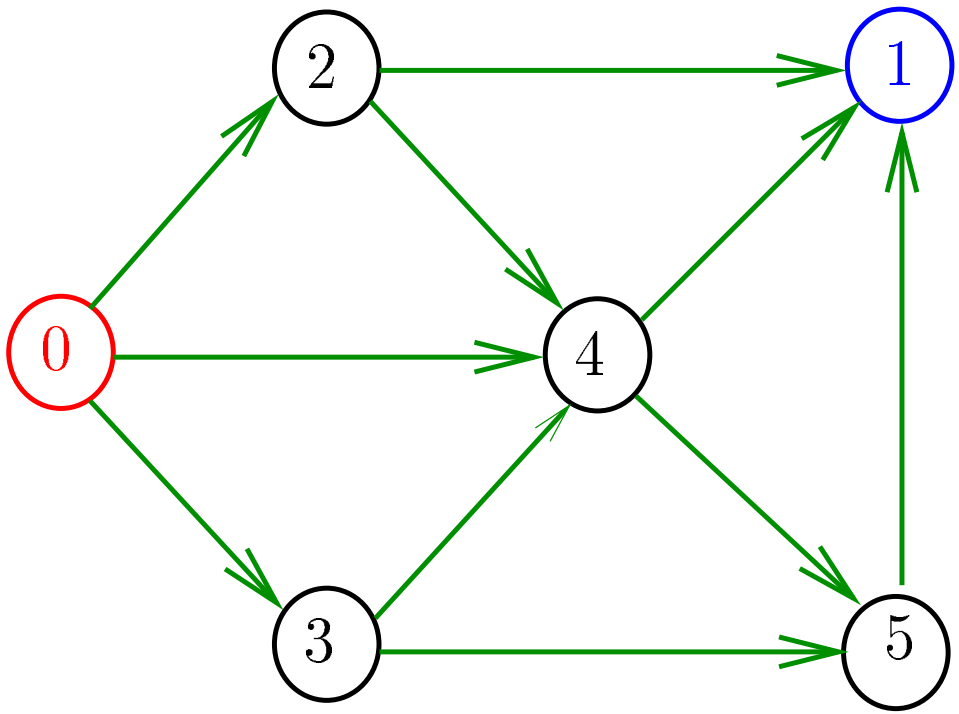
Cortes e arborescências

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?

Certificados

Como é possível 'verificar' a resposta?

Como é possível 'verificar' que existe caminho?

Como é possível 'verificar' que não existe caminho?

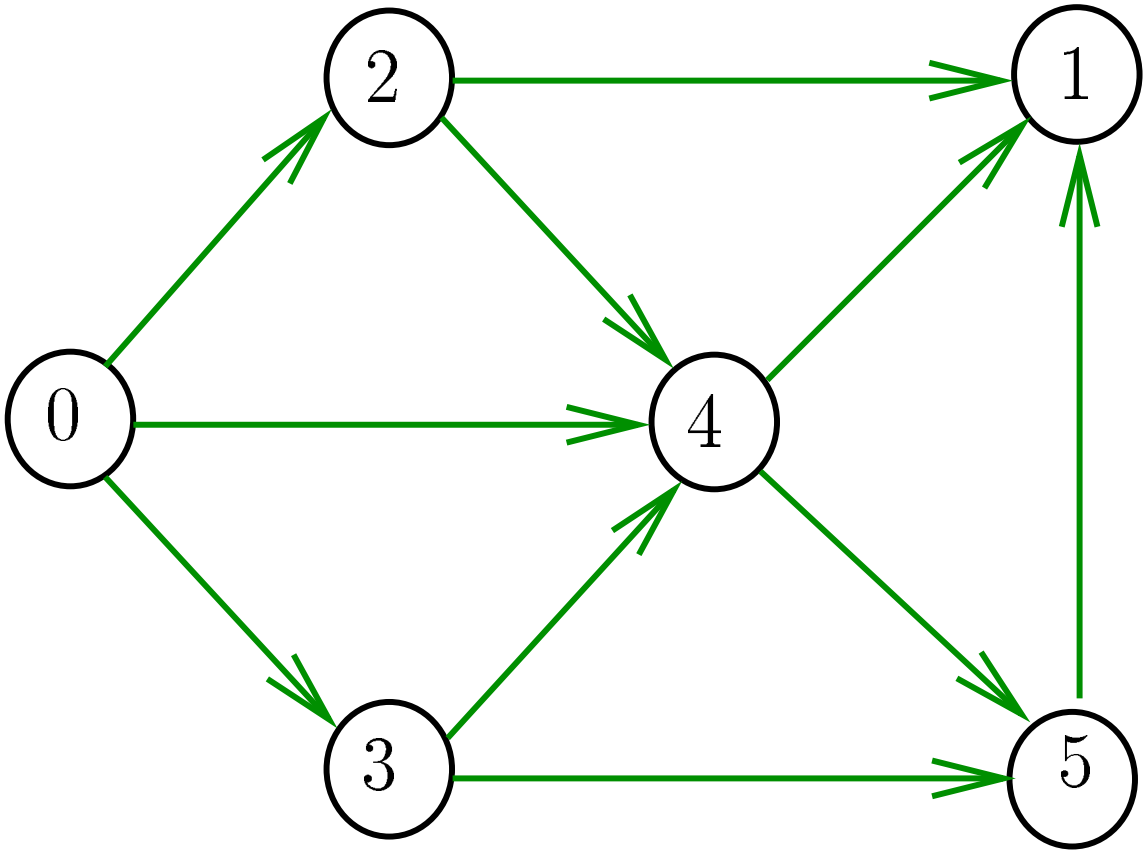
Veremos questões deste tipo freqüentemente

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

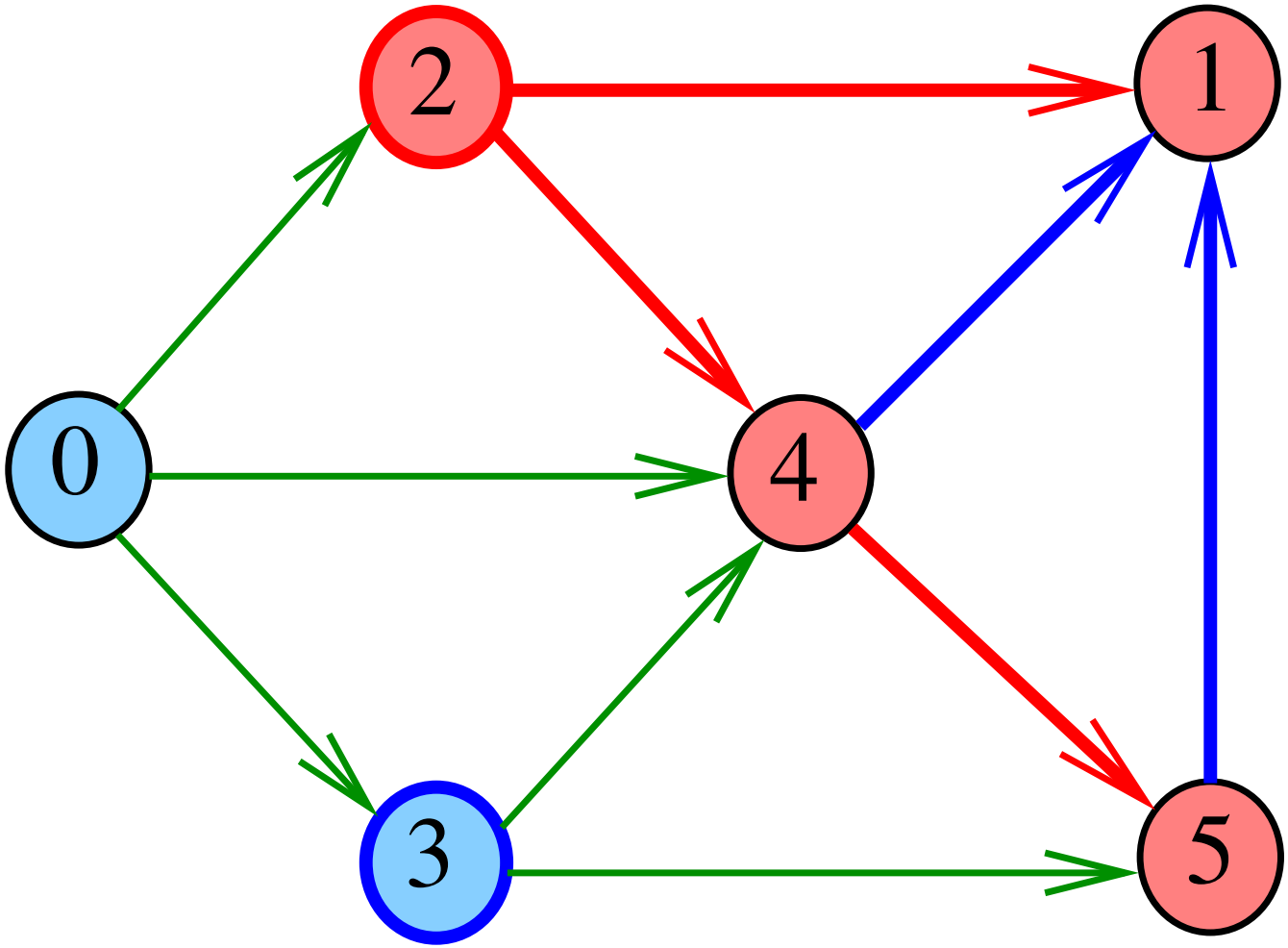
Certificado de inexistência

Como é possível demonstrar que o problema não tem solução?




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

DFSpath(G,2,3)

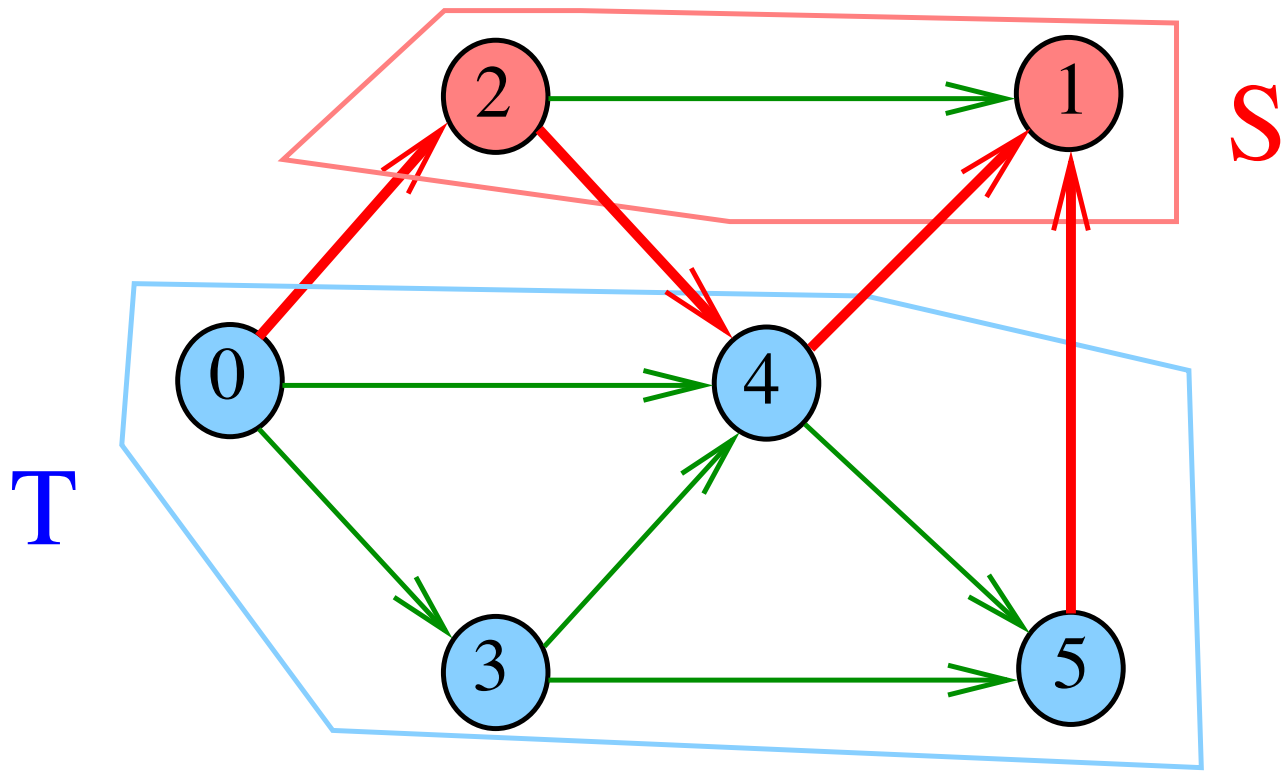


Cortes (= *cuts*)

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)

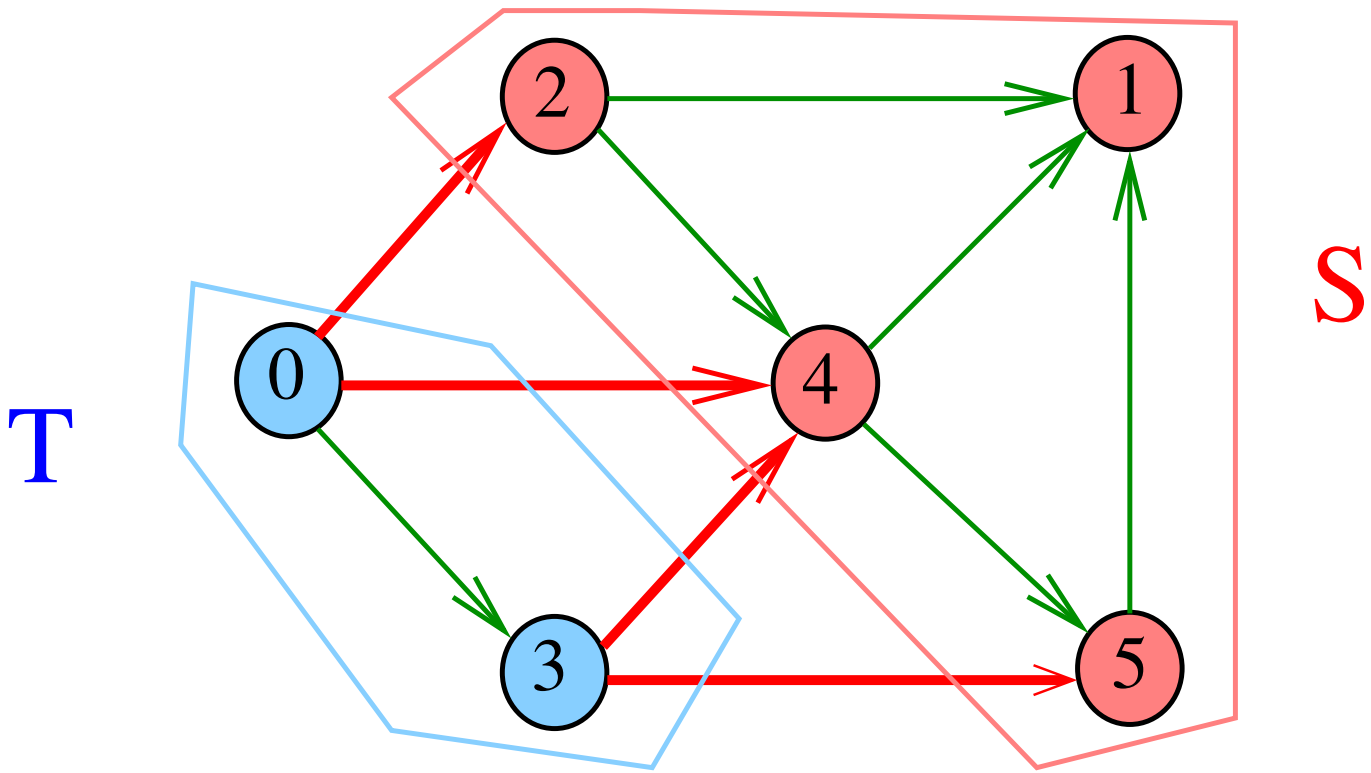


Cortes (= *cuts*)

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)

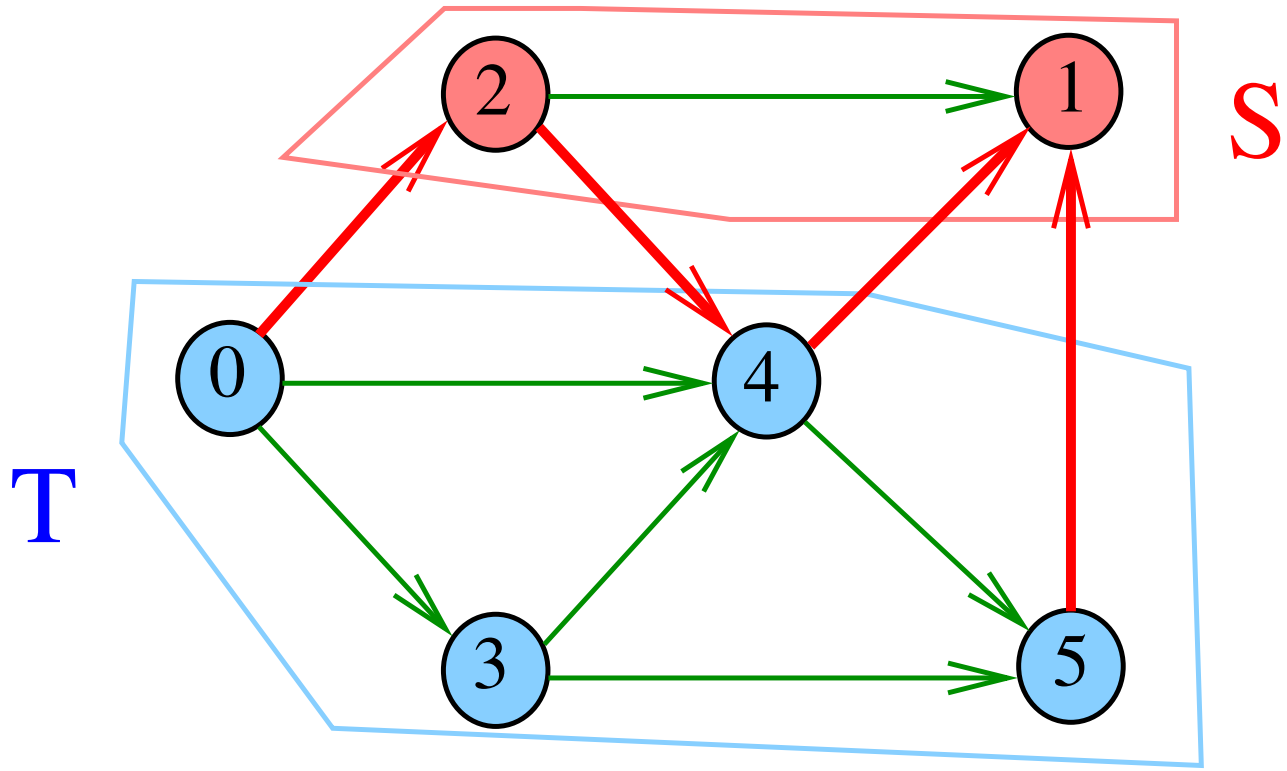


st-Cortes (= st-*cuts*)

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 . . .



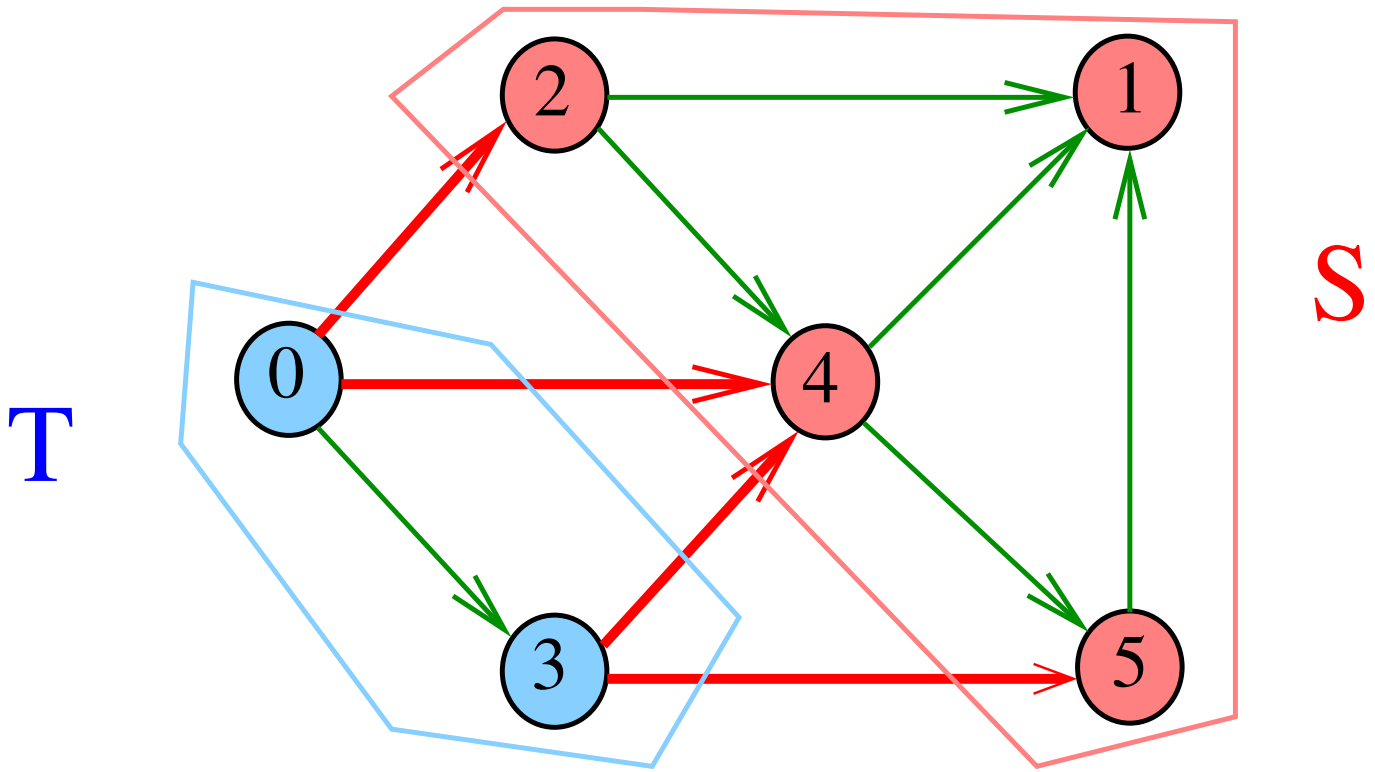
Certificado de inexistência

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



Conclusão

Para quaisquer vértices s e t de um digrafo, vale uma e apenas umas 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.

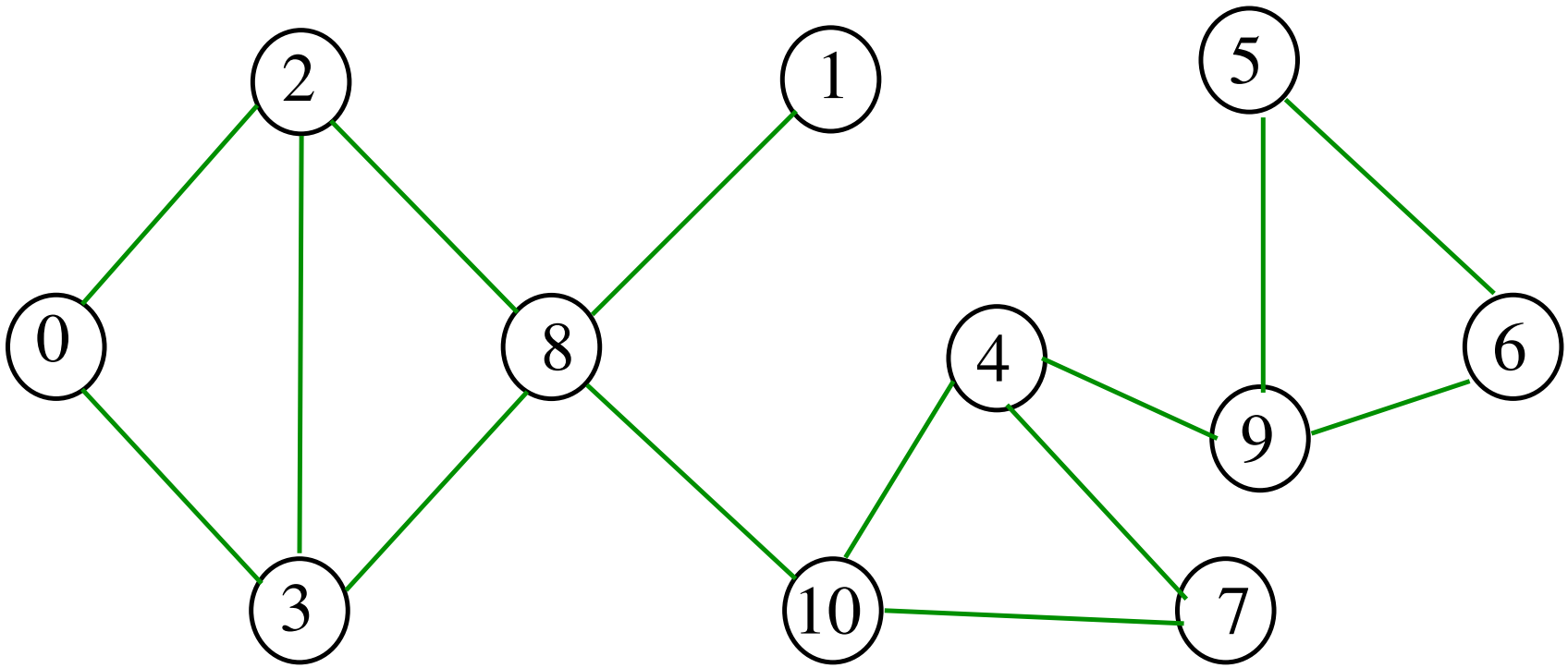
Componentes de grafos

S 18.5

Grafos conexos

Um grafo é **conexo** se e somente se, para cada par `(s,t)` de seus vértices, existe um caminho com origem `s` e término `t`

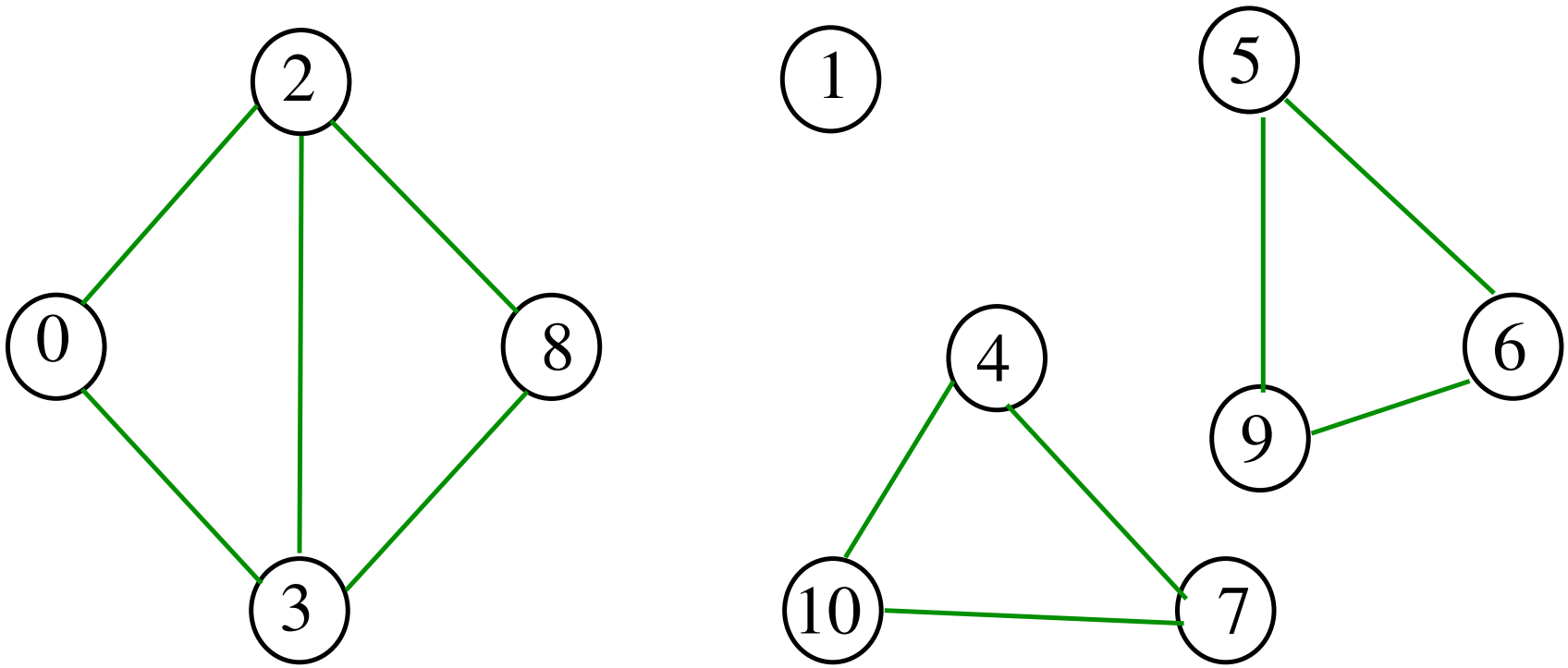
Exemplo: um grafo conexo



Componentes de grafos

Uma **componente** (= *component*) de um grafo é o subgrafo conexo maximal

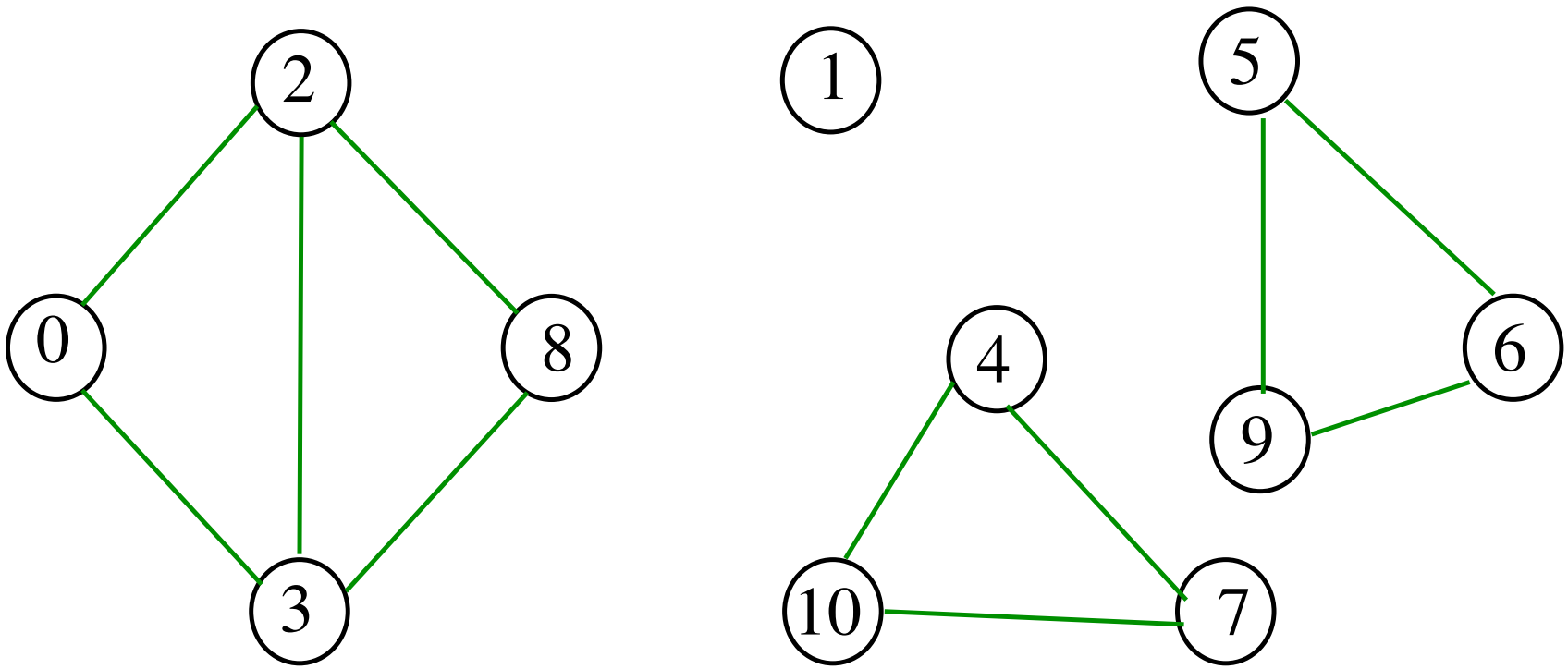
Exemplo: grafo com 4 componentes (conexos)



Contando componentes

Problema: calcular o número de componente

Exemplo: grafo com 4 componentes



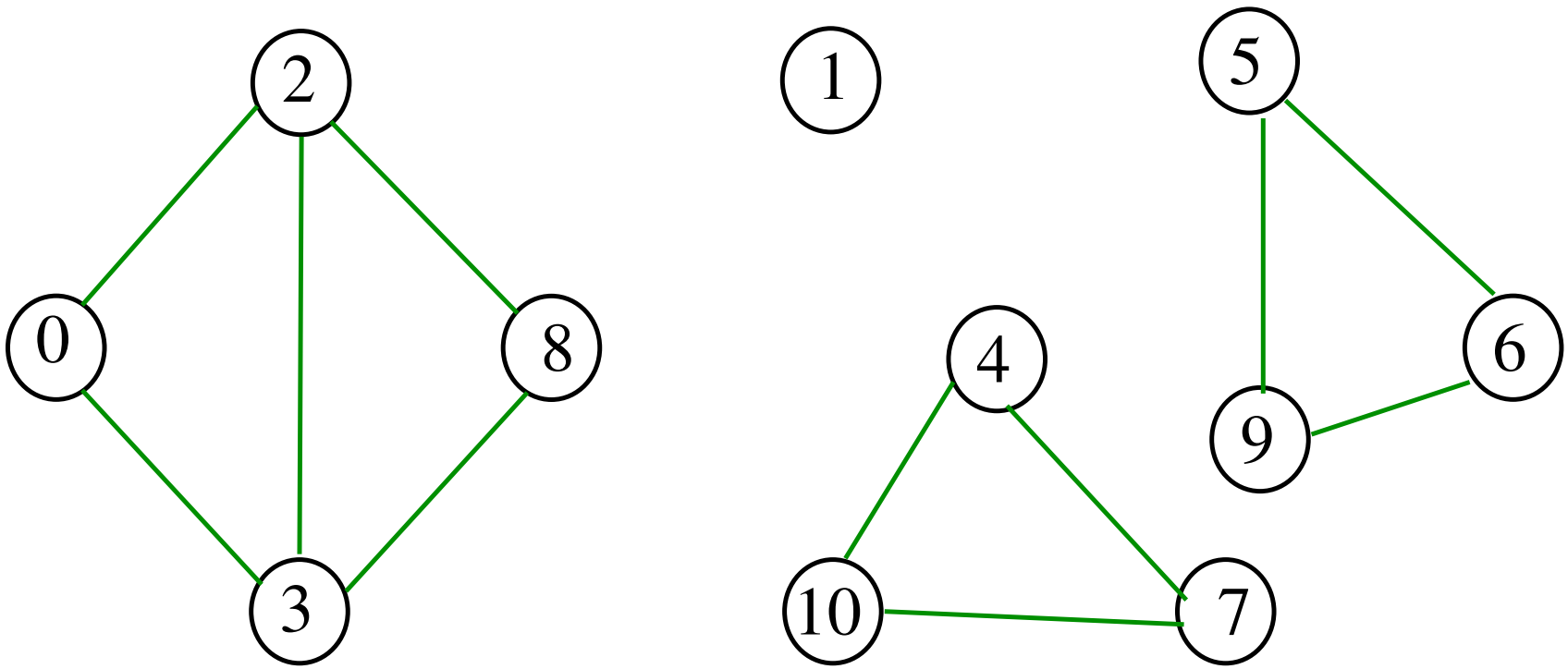
Cálculo das componentes de grafos

O classe CC calcula o número de componentes do grafo `G`.
Além disso, ela armazena no vetor `id` o número do componente a que o vértice pertence: se o vértice `v` pertence ao `k`-ésimo componente então `id[v] == k-1`

```
int GRAPHcc (Graph G)
```

Exemplo

v	0	1	2	3	4	5	6	7	8	9	10
id[v]	0	1	0	0	2	3	3	2	0	3	2



Consumo de tempo

O consumo de tempo da função `GRAPHcc` é $O(V + E)$.

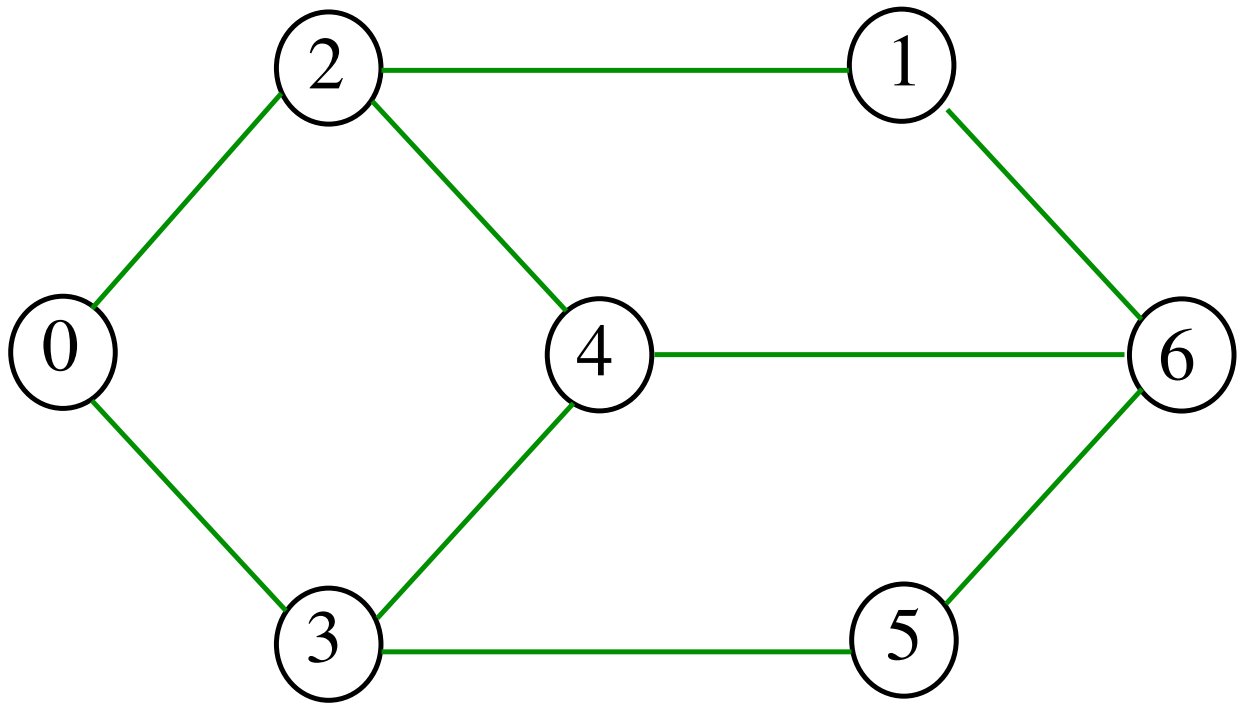
Grafos bipartidos e ciclos ímpares

S 18.5

Bipartição

Um grafo é **bipartido** (= *bipartite*) se existe uma bipartição do seu conjunto de vértices tal que cada aresta tem uma ponta em uma das partes da bipartição e a outra ponta na outra parte

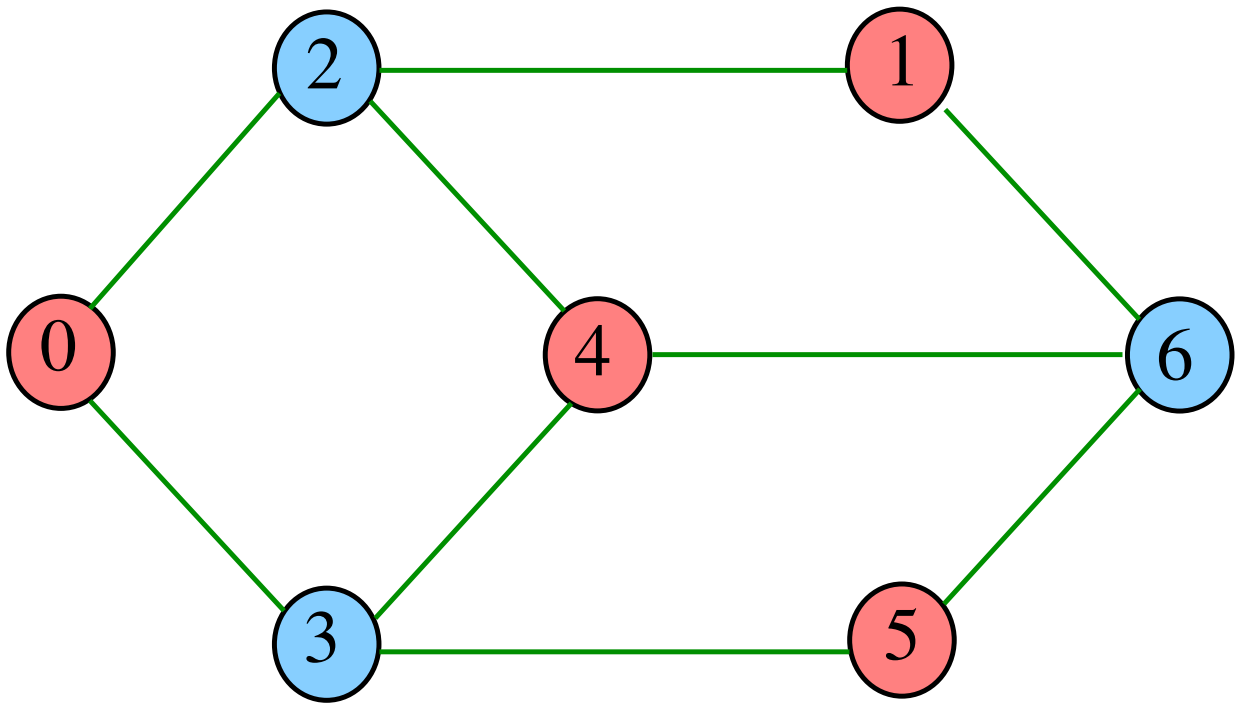
Exemplo:



Bipartição

Um grafo é **bipartido** (= *bipartite*) se existe uma bipartição do seu conjunto de vértices tal que cada aresta tem uma ponta em uma das partes da bipartição e a outra ponta na outra parte

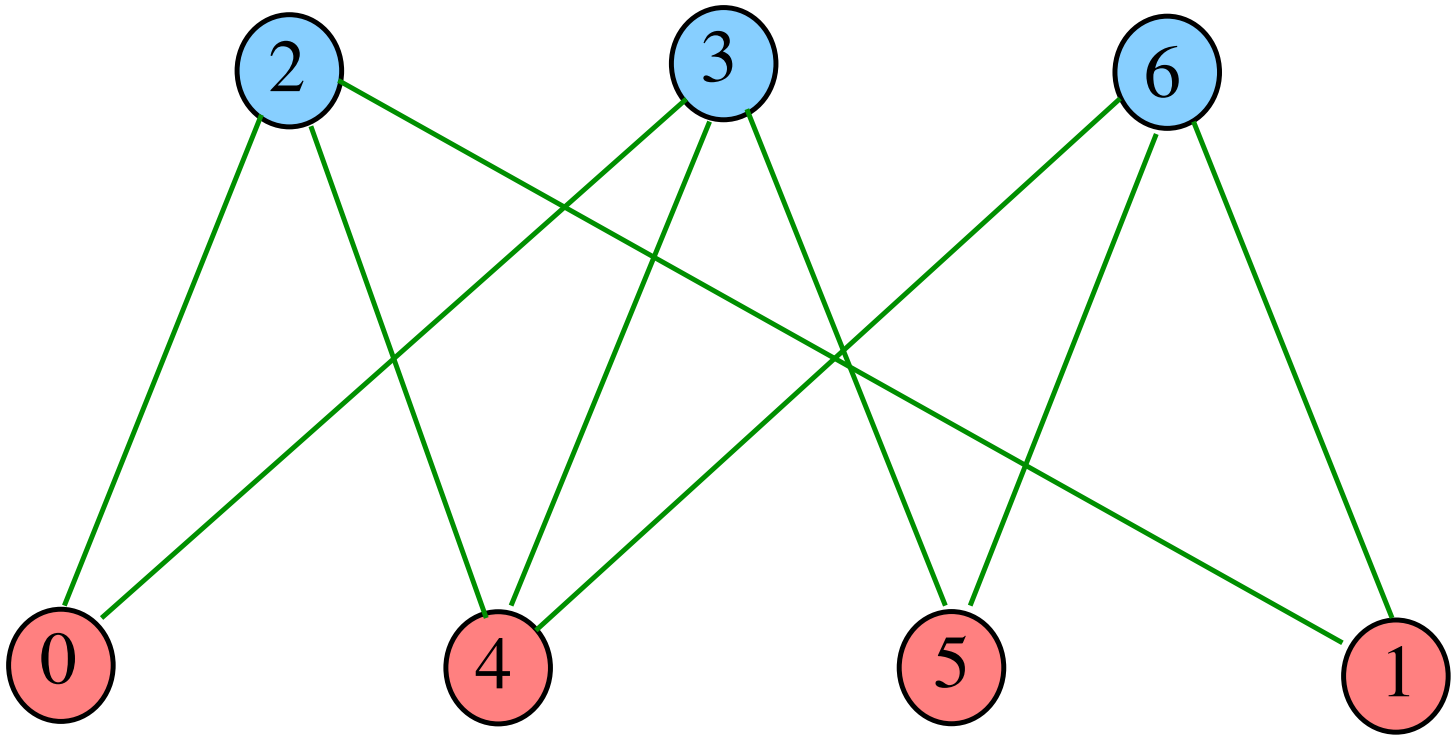
Exemplo:



Bipartição

Um grafo é **bipartido** (= *bipartite*) se existe uma bipartição do seu conjunto de vértices tal que cada aresta tem uma ponta em uma das partes da bipartição e a outra ponta na outra parte

Exemplo:



Class DFSBipartite

A classe decide se um dado grafo `G` é bipartido.

Nossos grafos têm `G.V()` vértices

```
private boolean color[G.V()];
```

Se `G` é bipartido, o método `dfs` atribui uma "cor" a cada vértice de `G` de tal forma que toda aresta tenha pontas de cores diferentes

As cores dos vértices, true e false, são registradas no vetor `color` indexado pelos vértices

```
public DFSBipartite (Graph G)
```

Consumo de tempo

A classe `DFSBipartite`, para vetor de listas de adjacência, consome tempo $O(V + E)$ para decidir se um grafo é bipartido.

Certificado

Para todo grafo G, vale uma e apenas umas das seguintes afirmações:

- G possui um ciclo ímpar
- G é bipartido