DAGs

DAGs

Um digrafo é acíclico se não tem ciclos

DAGs

Um digrafo é acíclico se não tem ciclos

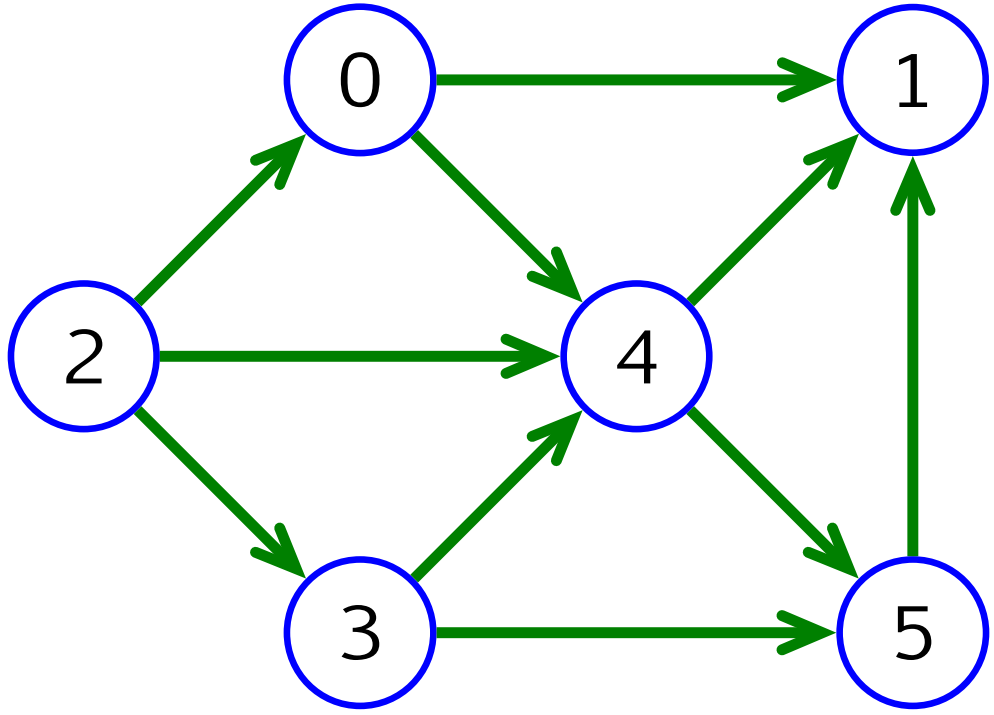
DAG = *directed acyclic graph*

DAGs

Um digrafo é acíclico se não tem ciclos

DAG = *directed acyclic graph*

Exemplo:



Acessibilidade

Acessibilidade

Num grafo acíclico, acessibilidade é *anti-simétrica*.

Acessibilidade

Num grafo acíclico, acessibilidade é *anti-simétrica*.

Afinal, se $u \hookrightarrow v$ e $v \hookrightarrow u$, então conseguimos um caminho fechado e daí um ciclo.

Acessibilidade

Num grafo acíclico, acessibilidade é *anti-simétrica*.

Afinal, se $u \hookrightarrow v$ e $v \hookrightarrow u$, então conseguimos um caminho fechado e daí um ciclo.

Num grafo acíclico, acessibilidade é uma *relação de ordem*

Acessibilidade

Num grafo acíclico, acessibilidade é *anti-simétrica*.

Afinal, se $u \hookrightarrow v$ e $v \hookrightarrow u$, então conseguimos um caminho fechado e daí um ciclo.

Num grafo acíclico, acessibilidade é uma *relação de ordem*

Qualquer relação de ordem é a de acessibilidade num grafo (mesmo infinito)

Acessibilidade

Num grafo acíclico, acessibilidade é *anti-simétrica*.

Afinal, se $u \hookrightarrow v$ e $v \hookrightarrow u$, então conseguimos um caminho fechado e daí um ciclo.

Num grafo acíclico, acessibilidade é uma *relação de ordem*

Qualquer relação de ordem é a de acessibilidade num grafo (mesmo infinito)

(Lembrando) A ordem é total se para todos $u \neq v$, $u \hookrightarrow v$ ou $v \hookrightarrow u$.

Extensão da ordem

Extensão da ordem

Isso é especialmente difícil para grafos infinitos:

Extensão da ordem

Isso é especialmente difícil para grafos infinitos:

Teorema

(Szpilrajn, 1930) Todo grafo acíclico pode ser estendido a um totalmente ordenado.

Extensão da ordem

Isso é especialmente difícil para grafos infinitos:

Teorema

(Szpilrajn, 1930) Todo grafo acíclico pode ser estendido a um totalmente ordenado.

Extensão da ordem

Isso é especialmente difícil para grafos infinitos:

Teorema

(Szpilrajn, 1930) Todo grafo acíclico pode ser estendido a um totalmente ordenado.

AVISO

Nunca processe um grafo infinito em casa.
A leitura dos dados acaba com a memória do computador.

Extensão da ordem

Isso é especialmente difícil para grafos infinitos:

Teorema

(Szpilrajn, 1930) Todo grafo acíclico pode ser estendido a um totalmente ordenado.

AVISO

Nunca processe um grafo infinito em casa.
A leitura dos dados acaba com a memória do computador.

Extensão da ordem

Isso é especialmente difícil para grafos infinitos:

Teorema

(Szpilrajn, 1930) Todo grafo acíclico pode ser estendido a um totalmente ordenado.

AVISO

Nunca processe um grafo infinito em casa.
A leitura dos dados acaba com a memória do computador.
E seu tempo.

Finito é bem mais simples

Finito é bem mais simples

Conjuntos infinitos podem ser ordenados de formas muito diferentes.
Ex: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$,...

Finito é bem mais simples

Conjuntos infinitos podem ser ordenados de formas muito diferentes.
Ex: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$,...

Num conjunto finito, todas ordenações têm a mesma cara, de fileirinha.

Finito é bem mais simples

Conjuntos infinitos podem ser ordenados de formas muito diferentes.
Ex: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$,...

Num conjunto finito, todas ordenações têm a mesma cara, de fileirinha.

Uma ordenação pode ser especificada numerando-se os elementos.

Ordenação topológica

Ordenação topológica

Uma ordenação topológica de um grafo G é uma bijeção t de uma sequência de n inteiros a $G.V$ tal que para todos i, j, se $t(i) \to t(j)$, então $i < j$.

Ordenação topológica

Uma ordenação topológica de um grafo G é uma bijeção t de uma sequência de n inteiros a $G.V$ tal que para todos i, j, se $t(i) \to t(j)$, então $i < j$.

Normalmente se usa $t : \{1, \ldots, n\} \to G.V$, mas é prático armazenar t como vetor ou lista, e usar $t : \{0, \ldots, n-1\} \to G.V$.

Ordenação topológica

Uma ordenação topológica de um grafo G é uma bijeção t de uma sequência de n inteiros a $G.V$ tal que para todos i, j, se $t(i) \to t(j)$, então $i < j$.

Normalmente se usa $t : \{1, \ldots, n\} \to G.V$, mas é prático armazenar t como vetor ou lista, e usar $t : \{0, \ldots, n-1\} \to G.V$.

Às vezes é melhor esquecer da numeração e só mencionar a ordenação.

Uma DFS num digrafo acíclico

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$
- descendente se
  $u.d < v.d < v.f < u.f$ e $v.sob \neq u$
- de retorno se
  $v.d < u.d < u.f < v.f$
- cruzado se
  $v.d < v.f < u.d < u.f$

Uma DFS num digrafo acíclico

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$
- descendente se
  $u.d < v.d < v.f < u.f$ e $v.sob \neq u$

- cruzado se
  $v.d < v.f < u.d < u.f$

Como encontrar

Como encontrar

Proposição

Se G é acíclico, então a ordem reversa de finalização de uma DFS em G é uma ordenação topológica.

Como encontrar

Proposição

Se G é acíclico, então a ordem reversa de finalização de uma DFS em G é uma ordenação topológica.

Como encontrar

Proposição

Se G é acíclico, então a ordem reversa de finalização de uma DFS em G é uma ordenação topológica.

Um algoritmo: durante a DFS, podemos produzir o vetor $t[\,]$ no caminho.

CLRS

TOPOLOGICAL-SORT(G)

1 call DFS(G) to compute finishing times $v.f$ for each vertex v

2 as each vertex is finished, insert it onto the front of a linked list

3 return the linked list of vertices

CLRS

TOPOLOGICAL-SORT(G)
1 call DFS(G) to compute finishing times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

O que tem de errado nisso?

Uma aplicação

Uma aplicação

PROBLEMA: Dado um digrafo, encontrar um caminho simples mais longo.

Uma aplicação

PROBLEMA: Dado um digrafo, encontrar um caminho simples mais longo.

Este problema é NP-completo!

Uma aplicação

PROBLEMA: Dado um digrafo, encontrar um caminho simples mais longo.

Este problema é NP-completo!

PROBLEMA: Dado um digrafo acíclico, encontrar um caminho mais longo.

Uma aplicação

PROBLEMA: Dado um digrafo, encontrar um caminho simples mais longo.

Este problema é NP-completo!

PROBLEMA: Dado um digrafo acíclico, encontrar um caminho mais longo.

Agora dá!

Caminho mais longo

Caminho mais longo

PROBLEMA: Dado um digrafo acíclico G, encontrar um caminho mais longo.

Caminho mais longo

PROBLEMA: Dado um digrafo acíclico G, encontrar um caminho mais longo.

ESQUELETO:

1. Construa uma ordenação topológica para G.

Caminho mais longo

PROBLEMA: Dado um digrafo acíclico G, encontrar um caminho mais longo.

ESQUELETO:

1. Construa uma ordenação topológica para G.
2. Associe a cada vértice v a variável $v.m$.

Caminho mais longo

PROBLEMA: Dado um digrafo acíclico G, encontrar um caminho mais longo.

ESQUELETO:

1. Construa uma ordenação topológica para G.
2. Associe a cada vértice v a variável $v.m$.
3. Processando os vértices em ordem topológica, mantenha o seguinte invariante

Caminho mais longo

PROBLEMA: Dado um digrafo acíclico G, encontrar um caminho mais longo.

ESQUELETO:

1. Construa uma ordenação topológica para G.
2. Associe a cada vértice v a variável $v.m$.
3. Processando os vértices em ordem topológica, mantenha o seguinte invariante

   para cada vértice v, $v.m$ é o comprimento do maior caminho que termina em v, cujos vértices anteriores já foram processados

Caminho mais longo

Caminho mais longo

Falta colocar músculo no esqueleto:

1. Como inicializar $v.m$?

Caminho mais longo

Falta colocar músculo no esqueleto:

1. Como inicializar $v.m$?
2. Como obter no final o *comprimento* do caminho mais longo?

Caminho mais longo

Falta colocar músculo no esqueleto:

1. Como inicializar $v.m$?
2. Como obter no final o *comprimento* do caminho mais longo?
3. Como adicionar informação no processo, de forma que no final se possa obter um caminho mais longo, como lista de vértices, em tempo linear?

Caminho mais longo

Falta colocar músculo no esqueleto:

1. Como inicializar $v.m$?
2. Como obter no final o *comprimento* do caminho mais longo?
3. Como adicionar informação no processo, de forma que no final se possa obter um caminho mais longo, como lista de vértices, em tempo linear?
4. Como adaptar para o caso de cada vértice e aresta ter um *peso* positivo? (PERT)

Grafos e subgrafos

Grafos e subgrafos

Se G e H são grafos, dizemos que H é subgrafo de G se $H.V \subseteq G.V$ e $H.A \subseteq G.A$.

Grafos e subgrafos

Se G e H são grafos, dizemos que H é subgrafo de G se $H.V \subseteq G.V$ e $H.A \subseteq G.A$.

Se $H.A$ consiste de todas as arestas de G entre seus vértices, H é um subgrafo induzido.

Grafos e subgrafos

Se G e H são grafos, dizemos que H é subgrafo de G se $H.V \subseteq G.V$ e $H.A \subseteq G.A$.

Se $H.A$ consiste de todas as arestas de G entre seus vértices, H é um subgrafo induzido.

Se $H.V = G.V$, H é um subgrafo gerador.

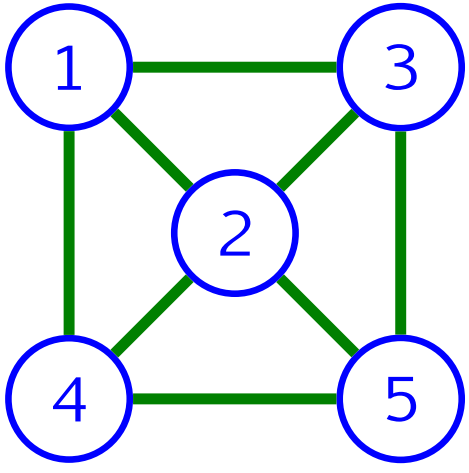
Grafos e subgrafos

Se G e H são grafos, dizemos que H é subgrafo de G se $H.V \subseteq G.V$ e $H.A \subseteq G.A$.

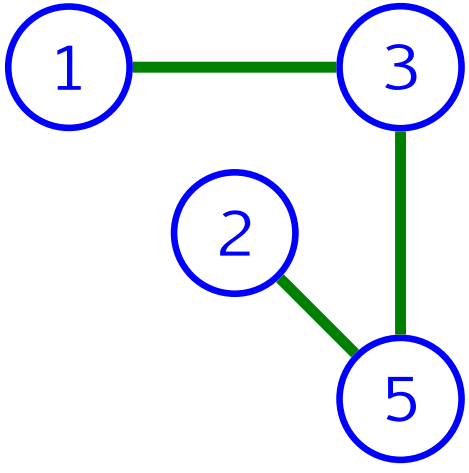
Se $H.A$ consiste de todas as arestas de G entre seus vértices, H é um subgrafo induzido.

Se $H.V = G.V$, H é um subgrafo gerador.

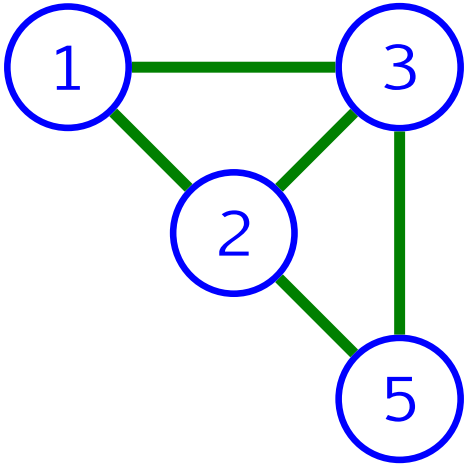
G



subgrafo



subgrafo induzido



Acessibilidade

Acessibilidade

Uma grande diferença entre grafos e digrafos é que a relação de *acessibilidade* é simétrica: se existe caminho de u a v, então existe caminho de v a u.

Acessibilidade

Uma grande diferença entre grafos e digrafos é que a relação de *acessibilidade* é simétrica: se existe caminho de u a v, então existe caminho de v a u.

Para grafos, essa relação é de *equivalência*.

Grafos conexos

Grafos conexos

Quando acessibilidade tem uma única classe, se diz que o grafo é conexo.

Grafos conexos

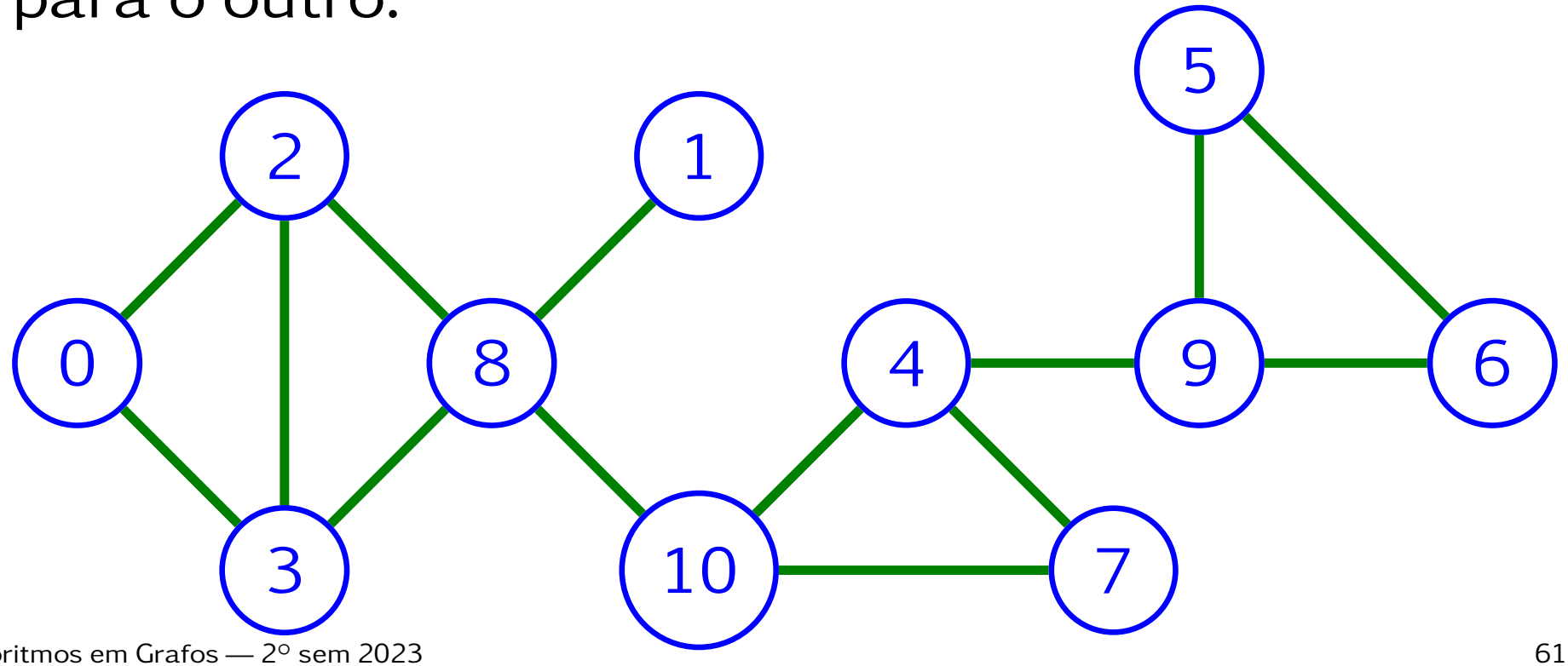
Quando acessibilidade tem uma única classe, se diz que o grafo é conexo.

Em outras palavras, um grafo é conexo se e somente se para cada dois vértices existe um caminho de um para o outro.

Grafos conexos

Quando acessibilidade tem uma única classe, se diz que o grafo é conexo.

Em outras palavras, um grafo é conexo se e somente se para cada dois vértices existe um caminho de um para o outro.



Componentes conexas

Componentes conexas

Uma componente conexa de G é um subgrafo induzido por uma classe de acessibilidade.

Componentes conexas

Uma componente conexa de G é um subgrafo induzido por uma classe de acessibilidade.

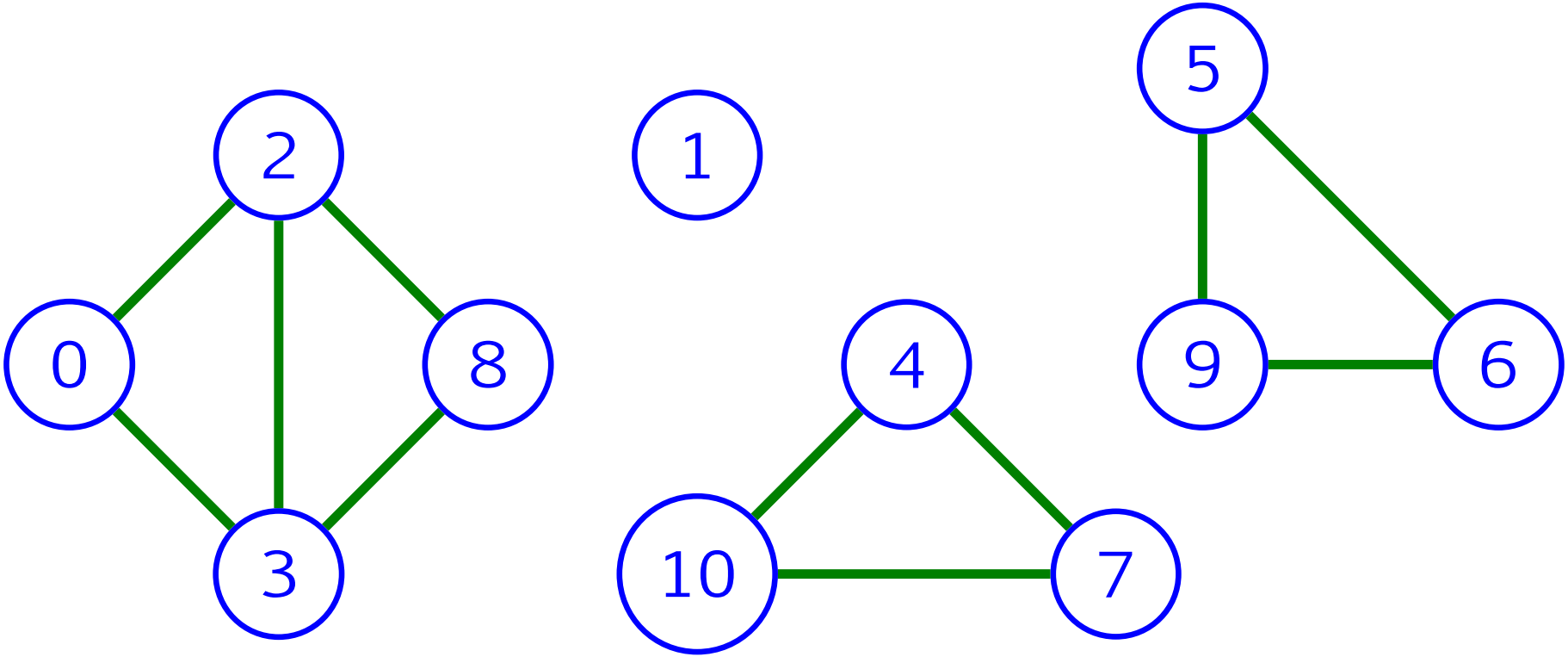
Ou seja, é um *subgrafo conexo maximal*.

Componentes conexas

Uma componente conexa de G é um subgrafo induzido por uma classe de acessibilidade.

Ou seja, é um *subgrafo conexo maximal*.

Exemplo: grafo com 4 componentes



Problema

Problema

Dado um grafo G, achar suas componentes.

DFS em grafos

DFS em grafos

O código para digrafos funciona exatamente para grafos, exceto que em vez de percorrer $v.out$ percorremos $v.adj$.

DFS em grafos

O código para digrafos funciona exatamente para grafos, exceto que em vez de percorrer $v.out$ percorremos $v.adj$.

Da mesma forma que antes, é produzida uma árvore de busca (na verdade, uma floresta).

DFS em grafos

O código para digrafos funciona exatamente para grafos, exceto que em vez de percorrer $v.out$ percorremos $v.adj$.

Da mesma forma que antes, é produzida uma árvore de busca (na verdade, uma floresta).

O que acontece com as outras arestas?

Relembrando o código

```
DFS-VISIT(G, u)
 1   u.state = descoberto
 2   u.d = ++time
 3   for each vertex v ∈ u.adj
 4         // processando o arco u → v
 5         if v.state == inicial
 6               // achou v
 7               v.sob = u
 8               DFS-VISIT(G, v)
 9         // else: revendo v
10   v.state = finalizado
11   u.f = time = time + 1
```

Classificação das arestas

Classificação das arestas

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$
- descendente se
  $u.d < v.d < v.f < u.f$ e $v.sob \neq u$
- de retorno se
  $v.d < u.d < u.f < v.f$
- cruzado se
  $v.d < v.f < u.d < u.f$

Classificação das arestas

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$

- de retorno se
  $v.d < u.d < u.f < v.f$
- cruzado se
  $v.d < v.f < u.d < u.f$

Classificação das arestas

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$

- de retorno se
  $v.d < u.d < u.f < v.f$

Achando componentes

```
DFS(G)
1   for each vertex u ∈ G.V
2         u.state = inicial
3         u.sob = NIL
4   time = 0
5   for each vertex u ∈ G.V
6         if u.state == inicial
7               //
8               DFS-VISIT(G, u)
9               //
```

Achando componentes

```
DFS(G)
1   for each vertex u ∈ G.V
2         u.state = inicial
3         u.sob = NIL
4   time = 0
5   for each vertex u ∈ G.V
6         if u.state == inicial
7               //
8               DFS-VISIT(G, u)
9               // Marcou a componente contendo u
```

Achando componentes

```
DFS(G)
1   for each vertex u ∈ G.V
2         u.state = inicial
3         u.sob = NIL
4   time = 0
5   for each vertex u ∈ G.V
6         if u.state == inicial
7               // Nova componente, associada a u
8               DFS-VISIT(G, u)
9               // Marcou a componente contendo u
```

E ciclos?

E ciclos?

Ciclos num grafo podem ser detectados por uma DFS, como no caso dirigido.

E ciclos?

Ciclos num grafo podem ser detectados por uma DFS, como no caso dirigido.

Como são grafos sem ciclos?

E ciclos?

Ciclos num grafo podem ser detectados por uma DFS, como no caso dirigido.

Como são grafos sem ciclos?

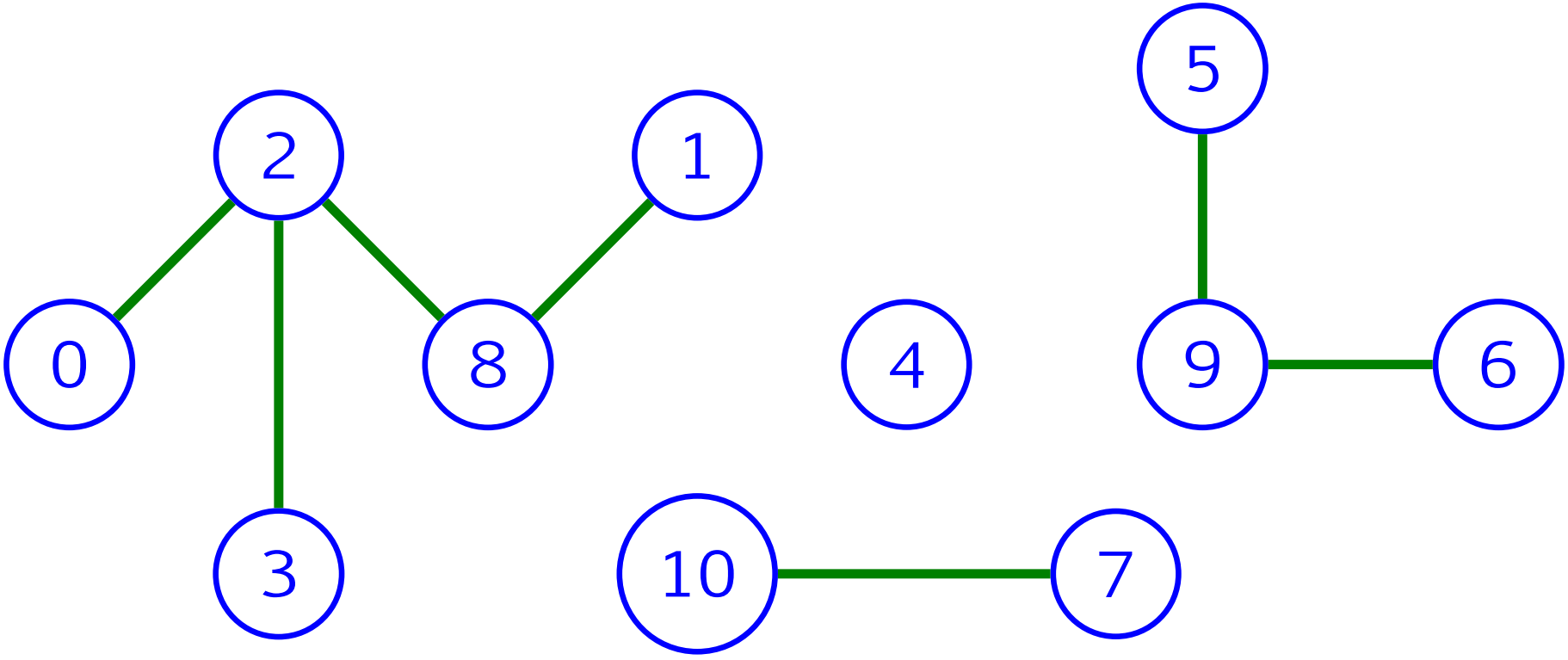
Se conexo, árvore (sem raiz), em geral floresta.

E ciclos?

Ciclos num grafo podem ser detectados por uma DFS, como no caso dirigido.

Como são grafos sem ciclos?

Se conexo, árvore (sem raiz), em geral floresta.



Algumas propriedades

Proposição

Para toda árvore:

Algumas propriedades

Proposição

Para toda árvore:

1. $m = n - 1$.

Algumas propriedades

Proposição

Para toda árvore:

1. $m = n - 1$.
2. *Existe um único caminho simples entre cada par de vértices.*

Algumas propriedades

Proposição

Para toda árvore:

1. $m = n - 1$.
2. *Existe um único caminho simples entre cada par de vértices.*
3. *Existem pelo menos dois vértices de grau 1 (folhas)*

Algumas propriedades

Proposição

Para toda árvore:

1. $m = n - 1$.
2. *Existe um único caminho simples entre cada par de vértices.*
3. *Existem pelo menos dois vértices de grau 1 (folhas)*

Proposição

Todo grafo conexo tem pelo menos uma árvore geradora (subgrafo gerador que é árvore).