Projeto e Análise de Algoritmos II (MC558)
Busca em grafos: aplicações, ordenação topológica

Prof. Dr. Ruben Interian

Resumo

Resumo

Revisão do conteúdo

Vimos 2 algoritmos de busca em grafos:

- Busca em largura, chamada de **BFS**.
- Busca em profundidade, chamada de **DFS**.

Cada algoritmo possui as suas próprias características e aplicações.

Objetivo

Vamos estudar algumas aplicações do algoritmo **DFS**:

- Componentes conexas em grafos (não direcionados);
- Ordenação topológica;
- ...

Resumo

Busca em profundidade: lembrando o algoritmo

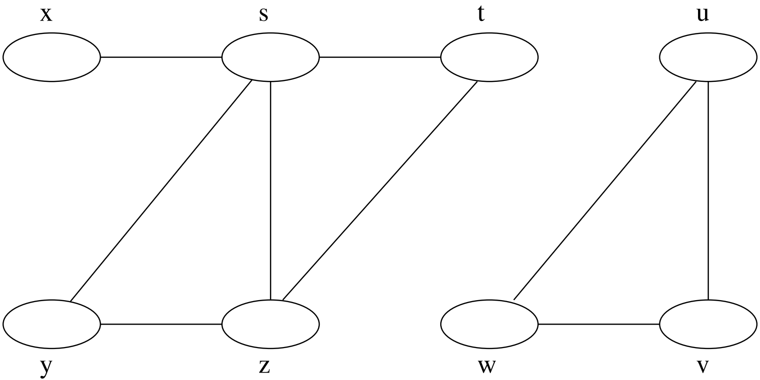
DFS (G)

▷ Inicialização:
para cada $u \in V$ **faça**
    cor$[u]$ ← branco
    $\pi[u] \leftarrow$ NIL
tempo ← 0
▷ Execução:
para cada $u \in V$ **faça**
    se cor$[u]$ = branco **então**
        DFS-Visit(G, u)

DFS-Visit (G, u)

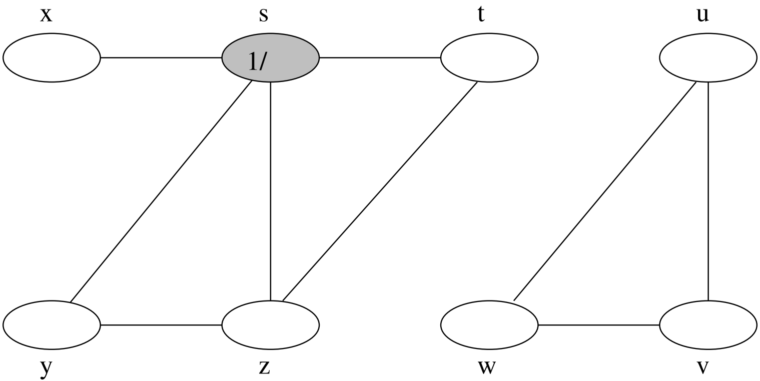
▷ Cria árvore de busca com origem u:
cor$[u]$ ← cinza
$d[u] \leftarrow$ ++tempo
para cada $v \in Adj[u]$ **faça**
    se cor$[v]$ = branco **então**
        $\pi[v] \leftarrow u$
        DFS-Visit (G, v)
cor$[u]$ ← preto
$f[u] \leftarrow$ ++tempo

Componentes conexas

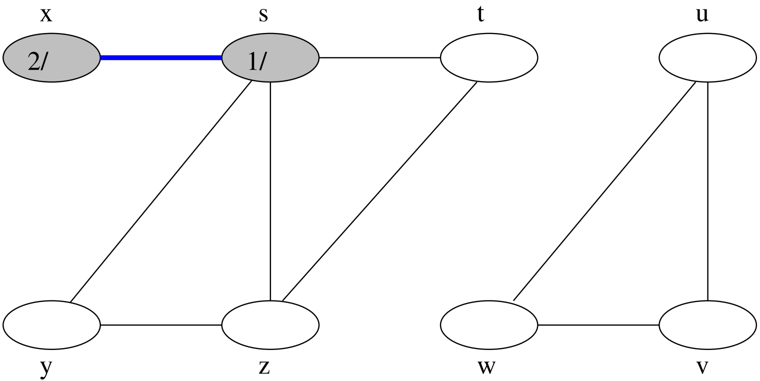


Problema: identificar as componentes conexas do grafo não direcionado.

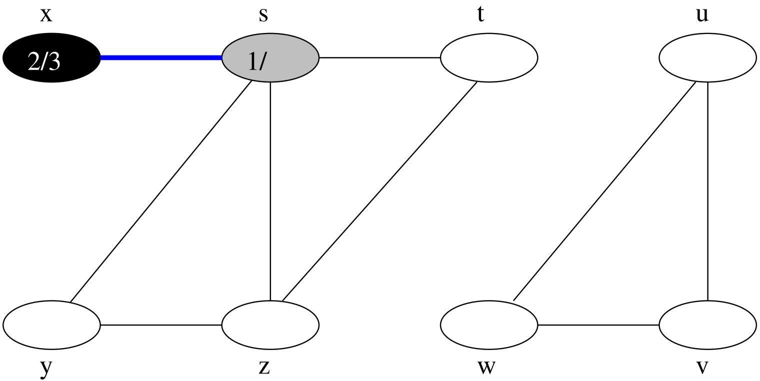
Componentes conexas



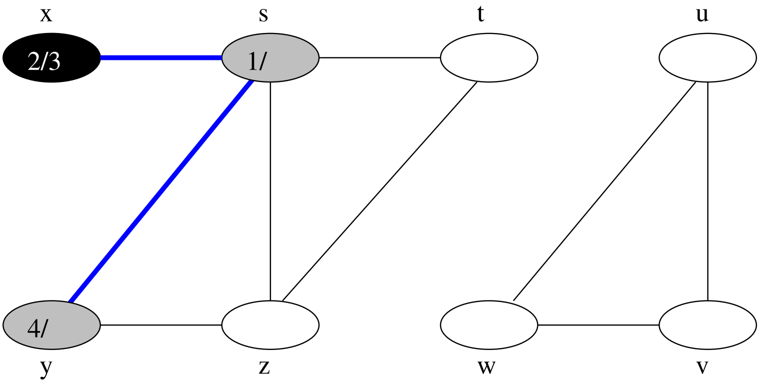
Componentes conexas



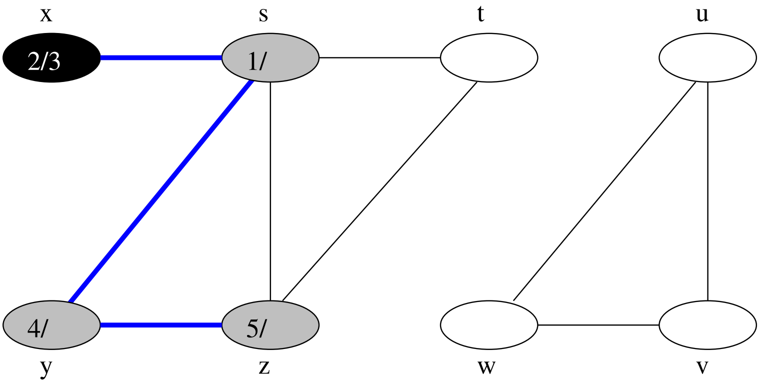
Componentes conexas



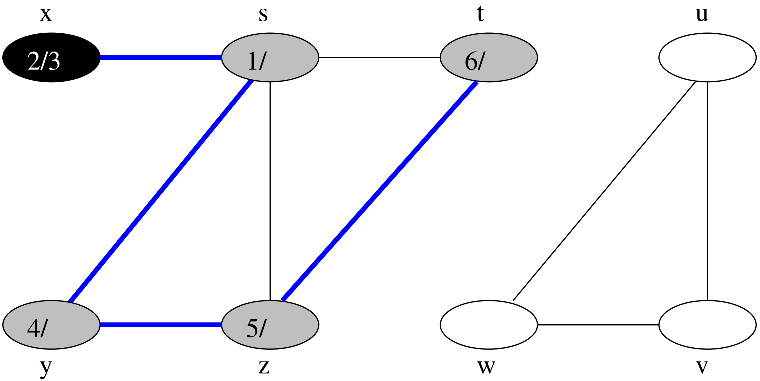
Componentes conexas



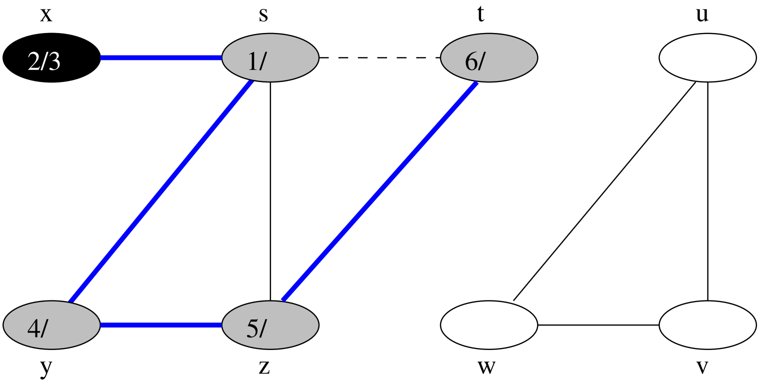
Componentes conexas



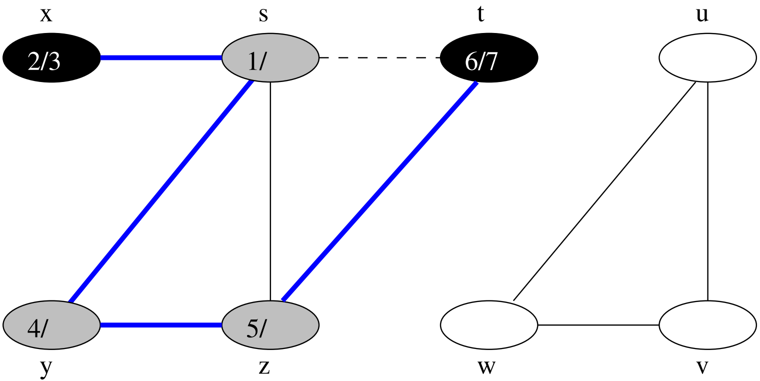
Componentes conexas



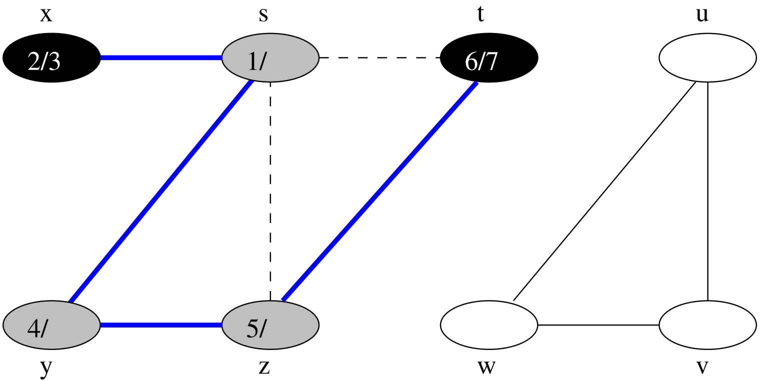
Componentes conexas



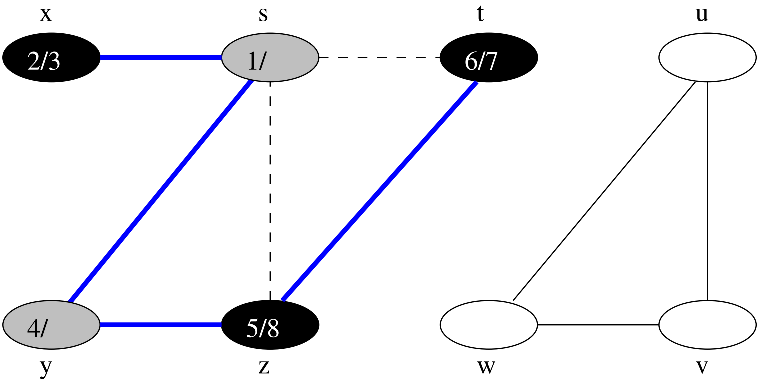
Componentes conexas



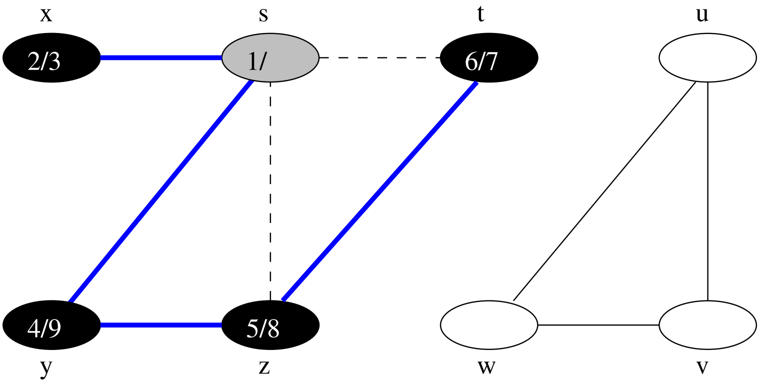
Componentes conexas



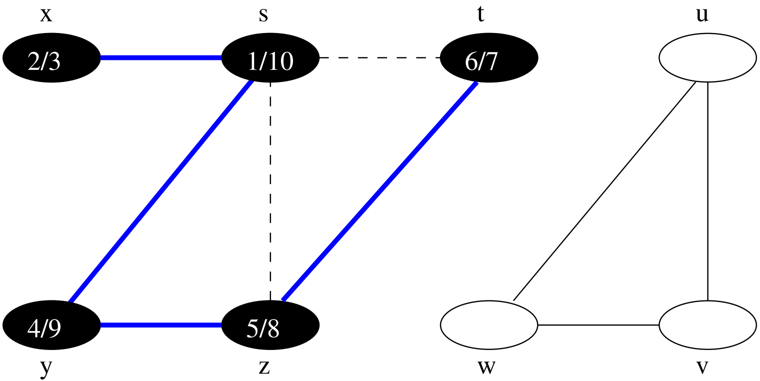
Componentes conexas



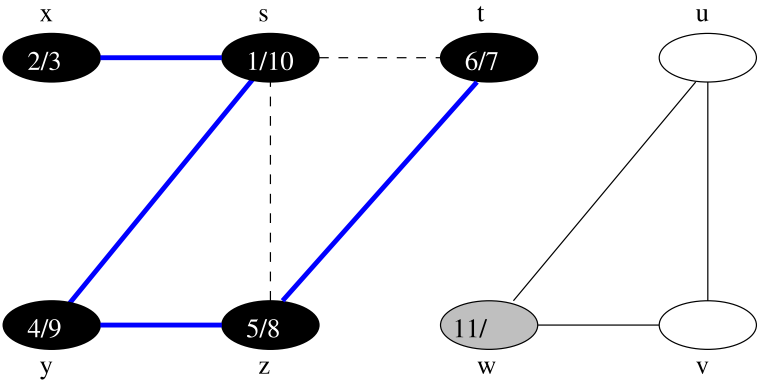
Componentes conexas



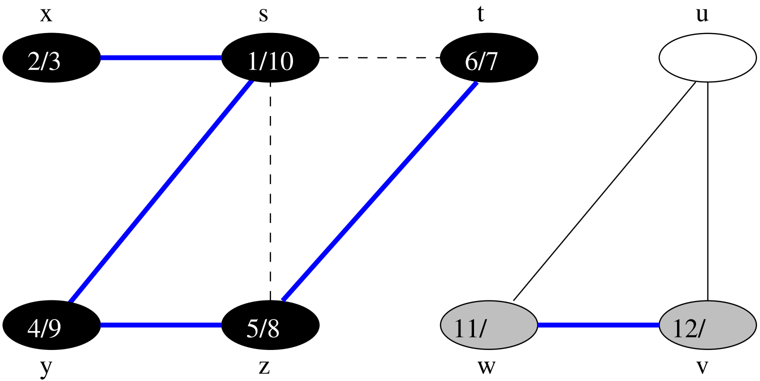
Componentes conexas



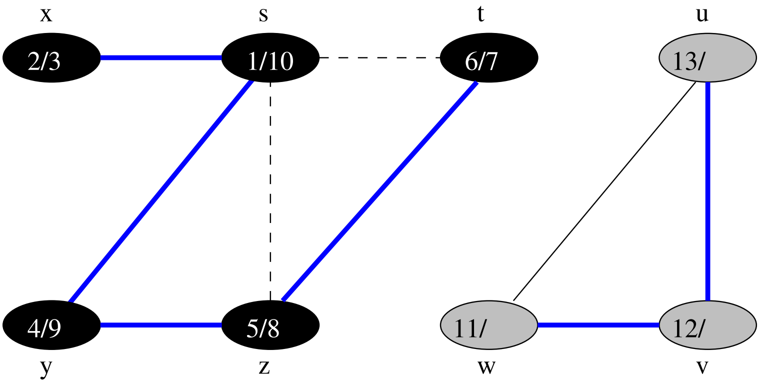
Componentes conexas



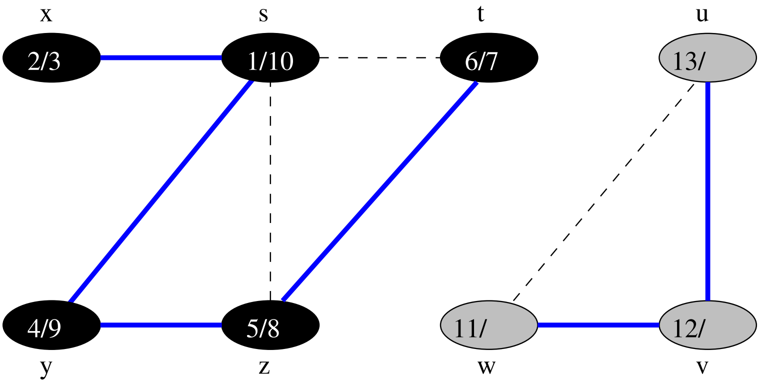
Componentes conexas



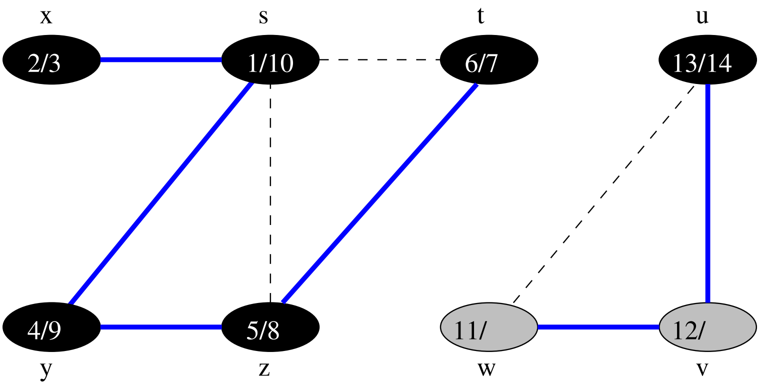
Componentes conexas



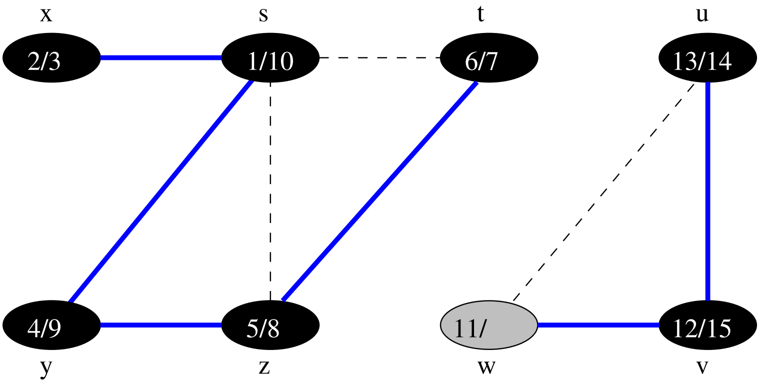
Componentes conexas



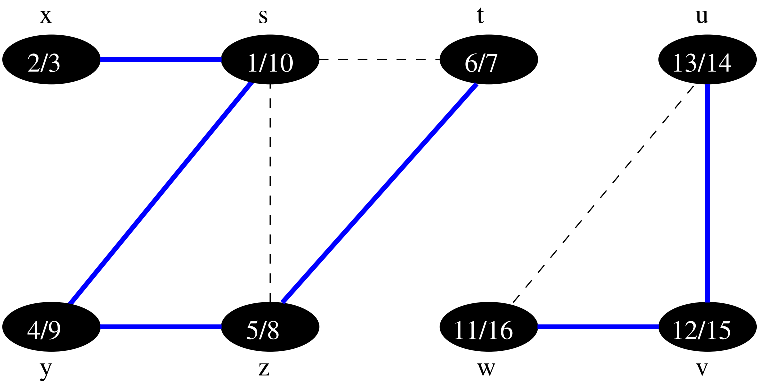
Componentes conexas



Componentes conexas



Componentes conexas



Componentes conexas

Contando o número de componentes de um grafo:

- Cada componente corresponde a uma árvore na floresta de busca;
- Número de componentes é o número de **chamadas** a **DFS-Visit** a partir do **DFS**.

Vamos modificar o **DFS** para computar as componentes do grafo:

- Identificamos cada componente usando um número inteiro.
- Denotaremos por comp[v] a componente do vértice v.

Busca em profundidade: o algoritmo modificado

DFS (G)

▷ Inicialização:
para cada $u \in V$ **faça**
    $\text{cor}[u] \leftarrow$ branco
    $\cancel{\pi[u] \leftarrow \text{NIL}}$
$\ell \leftarrow 0$ ▷ contador de chamadas
▷ Execução:
para cada $u \in V$ **faça**
    se $\text{cor}[u] =$ branco **então**
        $\ell{+}{+}$
        DFS-Visit(G, u)

DFS-Visit (G, u)

▷ Identifica os vértices da componente:
$\text{cor}[u] \leftarrow$ cinza
$\cancel{d[u] \leftarrow {+}{+}\text{tempo}}$
para cada $v \in Adj[u]$ **faça**
    se $\text{cor}[v] =$ branco **então**
        $\cancel{\pi[v] \leftarrow u}$
        DFS-Visit (G, v)
$\text{cor}[u] \leftarrow$ preto
$\cancel{f[u] \leftarrow {+}{+}\text{tempo}}$
$\text{comp}[v] \leftarrow \ell$

Resumo

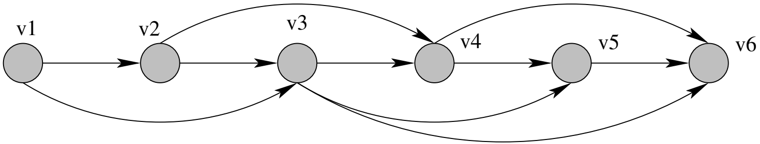
Ordenação topológica

Ordenação topológica

Uma ordenação topológica de um grafo direcionado $G = (V, A)$ é um arranjo linear dos vértices de G:

$$v_1, v_2, \ldots, v_{n-1}, v_n$$

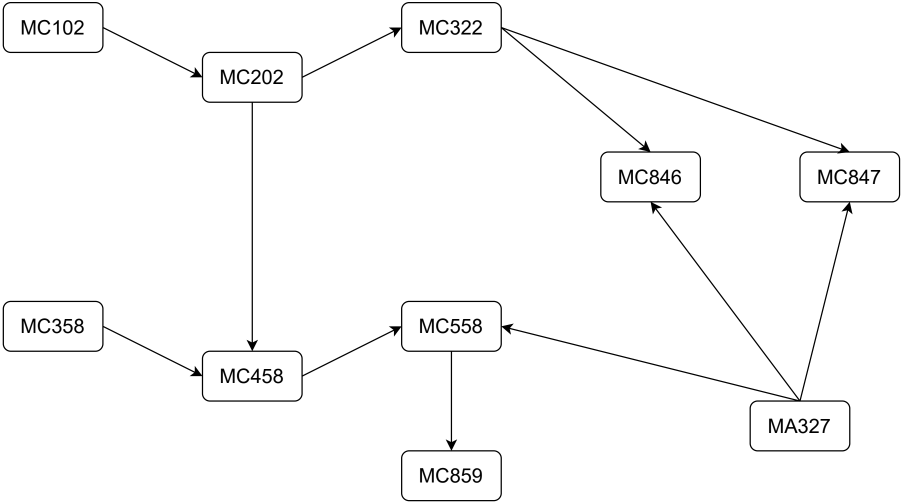
tal que se (v_i, v_j) é um arco de G, então $i < j$.



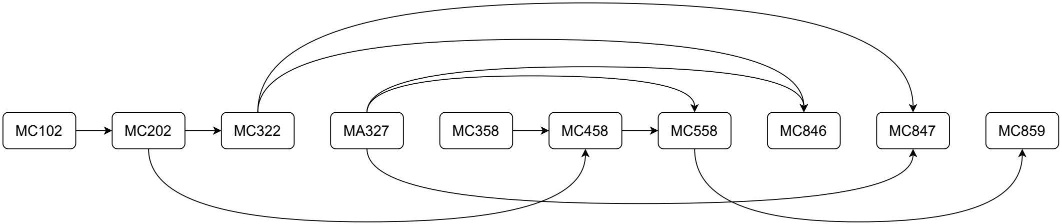
Ordenação topológica: aplicações reais

- Priorização (ordem de execução) de tarefas, respeitando precedências entre elas;
- Instalação de pacotes (por exemplo, em Python);
- Compiladores: geração de código dos segmentos do programa na ordem correta;
- Sistemas operacionais: prevenção de deadlock (quando dois ou mais processos estão bloqueados, esperando uns pelos outros).

Ordenação topológica: exemplo



Ordenação topológica: exemplo



Ordenação topológica

Todo grafo direcionado possui ordenação topológica?

Ordenação topológica

Todo grafo direcionado possui ordenação topológica?

- Não. Contraexemplo: um ciclo direcionado.

Ordenação topológica

Todo grafo direcionado possui ordenação topológica?

- Não. Contraexemplo: um ciclo direcionado.
- Nenhum grafo que contém um ciclo direcionado possui uma ordenação topológica.

Ordenação topológica

Todo grafo direcionado possui ordenação topológica?

- Não. Contraexemplo: um ciclo direcionado.
- Nenhum grafo que contém um ciclo direcionado possui uma ordenação topológica.

Um grafo direcionado é acíclico se não contiver um ciclo direcionado.

Ordenação topológica

Uma **condição necessária** para um grafo direcionado ter uma ordenação topológica é ser acíclico.

Essa condição é **suficiente**?

Ordenação topológica

Uma **condição necessária** para um grafo direcionado ter uma ordenação topológica é ser acíclico.

Essa condição é **suficiente**? SIM.

Ordenação topológica

Teorema

Um grafo direcionado $G = (V, A)$ possui uma ordenação topológica se e somente se G é acíclico.

Ordenação topológica

Teorema

Um grafo direcionado $G = (V, A)$ possui uma ordenação topológica se e somente se G é acíclico.

Para provar o **Teorema**, vamos precisar de duas definições e de um lema.

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Ordenação topológica

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Ordenação topológica

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração:

- Seja $v_1, \ldots, v_k$ um caminho mais longo no grafo.
- Veja que v_1 é uma fonte e v_k é um sorvedouro (Por quê?).

Ordenação topológica

Teorema

Um grafo direcionado $G = (V, A)$ possui uma ordenação topológica se e somente se G é acíclico.

Ordenação topológica

Teorema

Um grafo direcionado $G = (V, A)$ possui uma ordenação topológica se e somente se G é acíclico.

Prova.

($\Rightarrow$) Se G possui uma ordenação topológica, então G é acíclico, pois já vimos que se G possui um ciclo direcionado, não pode ter uma ordenação topológica.

($\Leftarrow$) Se G é acíclico, então G possui uma ordenação topológica.

Usaremos indução em $|V| = n$.

Ordenação topológica

G é acíclico $\Rightarrow$ G possui uma ordenação topológica

Base: se $n = 1$, temos apenas um vértice, então a sequência v_1 é uma ordenação topológica de G.

Hipótese de indução: Vamos supor que todo grafo acíclico com $n - 1$ vértices possui uma ordenação topológica $v_1, \ldots, v_{n-1}$.

Passo de indução: G é um grafo com n vértices, e pelo Lema, G possui um sorvedouro. Vamos chamar esse sorvedouro v_n. Então $G - v_n$ é acíclico, e pela hipótese de indução possui uma ordenação topológica $v_1, \ldots, v_{n-1}$.

Logo, $v_1, v_2, \ldots, v_n$, é uma ordenação topológica de G.

■

Ordenação topológica

Veja que a **demonstração** anterior é **construtiva**. Ela sugere um algoritmo recursivo:

1. Encontre um sorvedouro x em G;
2. Recursivamente, obtenha uma ordenação de $G - x$;
3. Coloque o sorvedouro no fim da sequência obtida e devolva o resultado.

Ordenação topológica

Veja que a **demonstração** anterior é **construtiva**. Ela sugere um algoritmo recursivo:

1. Encontre um sorvedouro x em G;
2. Recursivamente, obtenha uma ordenação de $G - x$;
3. Coloque o sorvedouro no fim da sequência obtida e devolva o resultado.

A **complexidade** desse algoritmo é $O(V^2)$:

- Há $|V|$ chamadas recursivas.
- Encontrar um sorvedouro leva tempo $O(V)$.

Ordenação topológica: algoritmo linear

Vamos projetar um **algoritmo linear** no tamanho da entrada (ou seja, $O(V + E)$) para encontrar uma ordenação topológica de um grafo acíclico $G = (V, E)$.

Ordenação topológica: algoritmo linear

Vamos projetar um **algoritmo linear** no tamanho da entrada (ou seja, $O(V + E)$) para encontrar uma ordenação topológica de um grafo acíclico $G = (V, E)$.

Ideia para o algoritmo:

- O 1^{ro} vértice que foi finalizado (1^{ro} em virar preto) pelo **DFS** é um sorvedouro.

Ordenação topológica: algoritmo linear

Vamos projetar um **algoritmo linear** no tamanho da entrada (ou seja, $O(V+E)$) para encontrar uma ordenação topológica de um grafo acíclico $G=(V,E)$.

Ideia para o algoritmo:

- O 1^{ro} vértice que foi finalizado (1^{ro} em virar preto) pelo **DFS** é um sorvedouro.
  - Por quê?

Ordenação topológica: algoritmo linear

Vamos projetar um **algoritmo linear** no tamanho da entrada (ou seja, $O(V+E)$) para encontrar uma ordenação topológica de um grafo acíclico $G=(V,E)$.

Ideia para o algoritmo:

- O 1^{ro} vértice que foi finalizado (1^{ro} em virar preto) pelo **DFS** é um sorvedouro.
  - Por quê? – Veja que ele não pode ter um arco saindo dele, nem para vértices brancos (não estaria finalizado), nem para vértices cinzas (haveria um ciclo).

Ordenação topológica: algoritmo linear

Vamos projetar um **algoritmo linear** no tamanho da entrada (ou seja, $O(V + E)$) para encontrar uma ordenação topológica de um grafo acíclico $G = (V, E)$.

Ideia para o algoritmo:

- O 1^{ro} vértice que foi finalizado (1^{ro} em virar preto) pelo **DFS** é um sorvedouro.
  - Por quê? – Veja que ele não pode ter um arco saindo dele, nem para vértices brancos (não estaria finalizado), nem para vértices cinzas (haveria um ciclo).
  - Como não há ciclos, não há arestas de retorno.

Ordenação topológica: algoritmo linear

Vamos projetar um **algoritmo linear** no tamanho da entrada (ou seja, $O(V+E)$) para encontrar uma ordenação topológica de um grafo acíclico $G=(V,E)$.

Ideia para o algoritmo:

- O 1^{ro} vértice que foi finalizado (1^{ro} em virar preto) pelo **DFS** é um sorvedouro.
  - Por quê? – Veja que ele não pode ter um arco saindo dele, nem para vértices brancos (não estaria finalizado), nem para vértices cinzas (haveria um ciclo).
  - Como não há ciclos, não há arestas de retorno.
- O 2^{do} vértice finalizado só pode ter arestas para o 1^{ro};
- O 3^{ro} só pode ter arestas para os dois primeiros ...

Ordenação topológica

Colocar os vértices em ordem decrescente de tempo de finalização parece funcionar ...

Algoritmo **Topological-Sort**(G):

1. Execute **DFS**(G) para calcular $f[u]$ para cada vértice u.
2. À medida que cada vértice for finalizado, coloque-o no início de uma lista ligada.
3. Devolva a lista ligada resultante.

Ordenação topológica

Colocar os vértices em ordem decrescente de tempo de finalização parece funcionar . . .

Algoritmo **Topological-Sort**(G):

1. Execute **DFS**(G) para calcular $f[u]$ para cada vértice u.
2. À medida que cada vértice for finalizado, coloque-o no início de uma lista ligada.
3. Devolva a lista ligada resultante.

A **complexidade** de tempo é $O(V + E)$:

- Executamos **DFS** apenas uma vez.
- Inserir cada um dos $|V|$ vértices leva tempo $O(1)$.

Exercício

Exercício:

- Execute o algoritmo **Topological-Sort** no grafo das disciplinas (ver início da aula), e compare a ordenação obtida com a apresentada → aqui.

Ordenação topológica: correção do algoritmo

Falta mostrar que **Topological-Sort** funciona.

Seja $G = (V, A)$ um grafo direcionado e acíclico. **Topological-Sort** devolve uma ordenação topológica de G.

Prova:

Ordenação topológica: correção do algoritmo

Falta mostrar que **Topological-Sort** funciona.

Seja $G = (V, A)$ um grafo direcionado e acíclico. **Topological-Sort** devolve uma ordenação topológica de G.

Prova: Lembre que a lista devolvida está em ordem decrescente de $f[v]$. Considere um arco (u, v). Vamos mostrar que $f[u] > f[v]$:

- Considere o instante em que (u, v) foi examinado.
- Como (u, v) não é aresta de retorno, v não pode ser cinza.
- Se v for branco, ele será descendente de u e $f[u] > f[v]$.
- Se v for preto, então ele já foi finalizado e $f[u] > f[v]$. ■

Ordenação topológica

Propriedade **importante** de grafos acíclicos:

Um grafo direcionado G é acíclico se, e somente se, em uma busca em profundidade em G, não há arestas de retorno.

Prova:

Ordenação topológica

Propriedade **importante** de grafos acíclicos:

Um grafo direcionado G é acíclico se, e somente se, em uma busca em profundidade em G, não há arestas de retorno.

Prova:
($\Rightarrow$) G é acíclico, então não há arestas de retorno. – Já vimos que quando tem uma aresta de retorno, tem um ciclo.

Ordenação topológica

Propriedade **importante** de grafos acíclicos:

Um grafo direcionado G é acíclico se, e somente se, em uma busca em profundidade em G, não há arestas de retorno.

Prova:

($\Rightarrow$) G é acíclico, então não há arestas de retorno. – Já vimos que quando tem uma aresta de retorno, tem um ciclo.

($\Leftarrow$) Exercício. (Dica: usar o Teorema do Caminho Branco; resposta no Cormen.)

Ordenação topológica: Exercício Conta-Caminhos

Exercício (CLRS 22.4-2). Descreva um algoritmo linear que recebe um grafo direcionado acíclico G e dois vértices s e t, e devolve o **número** de caminhos de s a t.

Note que precisamos apenas contar os caminhos, não exibi-los.

Ordenação topológica: Exercício Conta-Caminhos

Calcular o número de caminhos de s a t

Vamos usar ordenação topológica e programação dinâmica.

Seja G um grafo direcionado acíclico. Seja $p(v) =$ número de caminhos de v a t.

Queremos determinar $p(s)$.

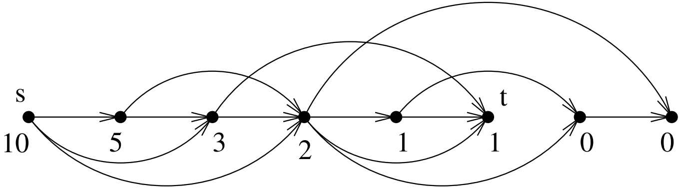
Ordenação topológica: Exercício Conta-Caminhos

Calcular o número de caminhos de s a t

Vamos usar ordenação topológica e programação dinâmica.

Seja G um grafo direcionado acíclico. Seja $p(v) =$ número de caminhos de v a t.

Queremos determinar $p(s)$.



Ordenação topológica: Exercício Conta-Caminhos

Qual é a relação entre $p(s)$ e cada vértice $v \in Adj[s]$?

$$p(s) =$$

Ordenação topológica: Exercício Conta-Caminhos

Qual é a relação entre $p(s)$ e cada vértice $v \in Adj[s]$?

$$p(s) = \sum_{v \in Adj[s]} p(v).$$

Isto vale para qualquer vértice u, não apenas s:

$$p(u) = \sum_{v \in Adj[u]} p(v).$$

Ordenação topológica: Exercício Conta-Caminhos

Seja uma ordenação topológica O.T. $= v_1, v_2, \ldots, v_n$ de G.

O valor $p(u)$ depende apenas de valores $p(v)$ onde $v \in Adj[u]$ (posteriores a u).

$$p(u) = \begin{cases} 1 & \text{se } u = t, \\ 0 & \text{se } u \text{ é posterior a } t \text{ na O.T.,} \\ \sum_{v \in Adj[u]} p(v) & \text{caso contrário.} \end{cases}$$

Note que se u é um sorvedouro, e é distinto de t, então a terceira regra diz que $p(u) = 0$.

Ordenação topológica: Exercício Conta-Caminhos

Conta-Caminhos (G, s, t)

1: **para cada** $v \in V \setminus \{s\}$ **faça**
2: $\quad p[v] \leftarrow 0$
3: $p[t] \leftarrow 1$
4: $v_1, v_2, \ldots, v_n \leftarrow$ ordenação topológica de G
5: **para cada** $i \in [n-1, n-2, \ldots, 1]$ **faça**
6: $\quad$ **para cada** $v \in Adj[v_i]$ **faça**
7: $\quad\quad p[v_i] \leftarrow p[v_i] + p[v]$

Complexidade: $O(V + E)$.
Corretude: no início da linha 5 vale a invariante $p[v_i] = p(v_i)$.

Resumo

Síntese

- Vimos algumas aplicações do algoritmo **DFS**:
  - Componentes conexas em grafos não direcionados;
  - Ordenação topológica;
  - Número de caminhos entre dois vértices.

Material bibliográfico e exercícios

T. Cormen et al. Algoritmos - Teoria e Prática (3a ed.). – **Cap. 22** (22.4)

Exercícios: ver exercícios no final do capítulo 22.4.

Dúvidas

Dúvidas?