

# Algoritmos

Pedro Hokama

- [cirs] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
  - [timr] Algorithms Illuminated Series, Tim Roughgarden
  - Desmistificando Algoritmos, Thomas H. Cormen.
  - Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
  - Stanford Algorithms  
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623ckIH7V>  
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZ17A5HEK6>
  - Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
  - Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420
- Qualquer erro é de minha responsabilidade.

## Componentes Conexas (em grafos não direcionados)

- Seja  $G = (V, E)$  um grafo **não direcionado**.
- As componentes conexas de um grafo são as partes do grafo que são isoladas.

### Definição

Uma **Componente Conexa** é uma classe de equivalência da relação  $u \sim v \iff$  existe um  $u - v$  caminho em  $G$ . ( $\sim$  é uma relação de equivalência)

- Desejamos encontrar todas as componente conexas de um grafo:
  - ▶ Verificar se a rede está desconectada
  - ▶ Visualização de Grafo (e como subrotina para outros algoritmos)
  - ▶ Clusterização (de páginas web, imagens, pessoas)

## BFS para encontrar Componentes Conexas

**Algoritmo 1:** Busca em Largura (BFS)

**Entrada:** Um Grafo  $G$ , e um vértice fonte  $s$

- 1 Marcar  $s$  como visitado;
- 2 Seja  $F$  uma Fila inicializada com  $s$ ;
- 3 **enquanto**  $F \neq \emptyset$  **faça**
- 4      $u = F.pop()$ ;
- 5     **para cada aresta**  $\{u, v\}$  **faça**
- 6         **se**  $v$  **está não-visitado** **então**
- 7             Marcar  $v$  como visitado;
- 8              $F.push(v)$ .

**Algoritmo 2:** Componentes-Conexas

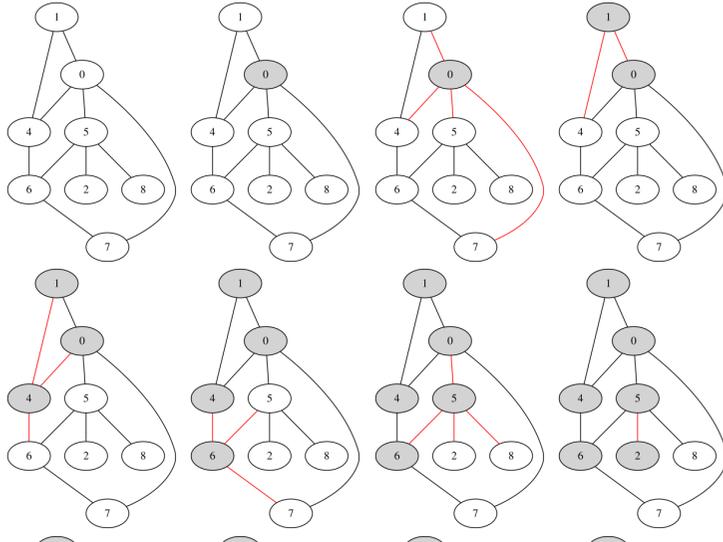
**Entrada:** Um Grafo  $G$

- 1 Marcar  $v \in V$  como não-visitado;
- 2 **para cada vértice**  $v$  de  $|V|$  **faça**
- 3     **se**  $v$  **está não-visitado** **então**
- 4         BFS( $G, i$ );

- Vértices marcados como visitado na mesma chamada de BFS

# Busca em Profundidade (Depth-First Search - DFS)

- Explora mais agressivamente, tentando ir o mais longe possível. Retrocede (backtrack) somente quando necessário. Exemplo:



5 / 12

- Por que ver outro algoritmo de Busca em Grafo, se a BFS já é linear?
- Outras aplicações: Ordenação Topológica de Grafos Direcionados Acíclicos, Componentes Fortemente Conexas em grafos direcionados.
- Tempo de execução:  $O(|V| + |E|)$

6 / 12

## DFS

- A ideia é que para decidir qual o próximo vértice a ser visitado utilizamos uma PILHA (Last In First Out - LIFO).
- Podemos fazer explicitamente, ou podemos fazer usando recursão.

---

### Algoritmo 3: DFS

---

**Entrada:** Um Grafo  $G$ , e um vértice fonte  $s$

- 1 Marcar  $s$  como visitado ;
  - 2 **para toda aresta**  $\{s, v\}$  **faça**
  - 3     **se**  $v$  **é não visitado** **então**
  - 4          $DFS(G, v)$ ;
- 

## Propriedades da DFS

### Teorema

Ao final do DFS,  $v$  é visitado  $\iff$  existem um caminho de  $s$  até  $v$  em  $G$

- O DFS é só um caso especial da busca genérica, e a prova desse Teorema é a mesma.
- O Tempo de execução é  $O(n_s + m_s)$  onde  $n_s$  é o número de vértices alcançáveis por  $s$  e  $m_s$  o número de arestas alcançáveis por  $s$ . Basta notar que no máximo é feito uma chamada recursiva de DFS para cada vértice alcançável, e cada aresta é analisada no máximo duas vezes.

7 / 12

8 / 12

## Ordenação Topológica

### Definição

Uma **Ordenação Topológica** de um grafo direcionado  $G = (V, A)$  é uma ordenação  $f$  dos vértices de  $V$ , tal que:

- 1  $f$  rotula os vértices com valores  $\{1, \dots, n\}$
- 2  $(u, v) \in A \implies f(u) < f(v)$

- Pode ser usado para resolver problemas de precedência. Suponha um Grafo onde os vértices são disciplinas, e as arestas indicam pré-requisitos. Ou a produção de um produto que tem várias etapas.
- É útil como pre-processamento em vários algoritmos.
- Se o Grafo tiver um ciclo direcionado então não existe uma ordenação topológica.
- Ordenação topológica só funciona em **Grafo Direcionado Acíclico** (Directed Acyclic Graph - DAG) !!!!

9 / 12

## Ordenação Topológica - intuição

- Se eu tenho um DAG eu posso encontrar uma Ordenação topológica
- Todo DAG tem um Sorvedouro (*Sink*), um vértice sem nenhuma aresta saindo. Por que?
- Encontrar o Sorvedouro  $v$ , atribuir  $f(v) = n$ , fazer a recursão em  $V - \{v\}$ .

10 / 12

## Propriedades da Ordenação Topológica

- Tempo de Execução Linear  $O(|V| + |E|)$ , já que cada vértice é visitado uma vez.
- O Algoritmo funciona corretamente, ou seja, se  $(u, v) \in A$  então  $f(u) < f(v)$ .
  - 1 Caso 1:  $u$  é visitado pela DFS antes de  $v$ . Então a chamada recursiva de  $v$  vai terminar antes, e portanto  $v$  será rotulado antes e  $f(v)$  será maior.
  - 2 Caso 2:  $v$  é visitado pela DFS antes de  $u$ . Como não existe um caminho direcionado de  $v$  para  $u$  (já que é um DAG) então  $u$  não é alcançado e logo a chamada recursiva de  $v$  vai terminar antes, e portanto  $v$  será rotulado antes e  $f(v)$  será maior.

---

### Algoritmo 4: DFS

**Entrada:** Um Grafo  $G$ , e um vértice fonte  $s$

- 1 Marcar  $s$  como visitado ;
- 2 **para** toda aresta  $\{s, v\}$  **faça**
- 3     **se**  $v$  é não visitado **então**
- 4          $DFS(G, v)$ ;
- 5  $f(s) = rotulo\_atual$ ;
- 6  $rotulo\_atual - -$ ;

---

### Algoritmo 5: Top-Sort

**Entrada:** Um Grafo  $G$

- 1 Marcar  $v \in V$  como não-visitado;
- 2  $rotulo\_atual = |V|$ ;
- 3 **para** cada aresta  $v \in V$  **faça**
- 4     **se**  $v$  está não-visitado **então**
- 5          $DFS(G, v)$ ;

11 / 12

12 / 12