*UFPA*

# *Grafos Direcionados (Dígrafos)*

January 9, 2022

# Resumindo - Relações e Grafos

## Relações e Grafos

| | |
|---|---|
| Relações Simétricas | Grafos não Direcionados |
| Relações Transitivas | Caminhos em Grafos |
| Relações de Equivalência | Caminhos em Grafos não Direcionados |
| Ordem Parcial | Caminhos em Dígrafos Acíclicos |

## Dígrafo Acíclico - DAG

- DAG - Tipo especial de dígrafo
- Ex. aplicação: escalonamento (agendamento)
  Ordenar um conjunto de atividades a serem concluídas de acordo com certas restrições (e.g. precedência).
- Como certificar se o dígrafo é ou não acíclico?

# *Dígrafo Acíclico - DAG*

- DAG - Tipo especial de dígrafo
- Ex. aplicação: escalonamento (agendamento)
  Ordenar um conjunto de atividades a serem concluídas de acordo com certas restrições (e.g. precedência).
- Como certificar se o dígrafo é ou não acíclico?
  $\rightarrow$ Algoritmo DFS

## *Dígrafo Acíclico - DAG*

*Mais conceitos:*

*Fonte -* é um vértice com grau de **entrada** $= 0$

*Sumidouro -* é um vértice com grau de **saída** $= 0$

## *Dígrafo Acíclico - DAG*

*Mais conceitos:*

*Fonte -* é um vértice com grau de **entrada** $= 0$

*Sumidouro -* é um vértice com grau de **saída** $= 0$

*Propriedade*

Todo DAG possui no mínimo uma *fonte* e no mínimo um *sumidouro*.

## *Dígrafo Acíclico - DAG*

*Mais conceitos:*

*Fonte* - é um vértice com grau de **entrada** $= 0$

*Sumidouro* - é um vértice com grau de **saída** $= 0$

*Propriedade*

Todo DAG possui no mínimo uma *fonte* e no mínimo um *sumidouro*.

**Prova:**

- Supondo um DAG sem *sumidouro*, pode-se construir um caminho longo de um vértice para qualquer outro.

# *Dígrafo Acíclico - DAG*

*Mais conceitos:*

*Fonte -* é um vértice com grau de **entrada** $= 0$

*Sumidouro -* é um vértice com grau de **saída** $= 0$

*Propriedade*

Todo DAG possui no mínimo uma *fonte* e no mínimo um *sumidouro*.

**Prova:**

- Supondo um DAG sem *sumidouro*, pode-se construir um caminho longo de um vértice para qualquer outro.
- Após $V + 1$ vértices, tem-se um ciclo. (Prin. das casas de pombo)

# *Dígrafo Acíclico - DAG*

*Mais conceitos:*

*Fonte* - é um vértice com grau de **entrada** $= 0$

*Sumidouro* - é um vértice com grau de **saída** $= 0$

*Propriedade*

Todo DAG possui no mínimo uma *fonte* e no mínimo um *sumidouro*.

**Prova:**

- Supondo um DAG sem *sumidouro*, pode-se construir um caminho longo de um vértice para qualquer outro.
- Após $V + 1$ vértices, tem-se um ciclo. (Prin. das casas de pombo)
- Ciclo é $\rightarrow$ contradição $\leftarrow$ com o conceito de DAG !!!!!

# *Dígrafo Acíclico - DAG*

*Mais conceitos:*

*Fonte -* é um vértice com grau de **entrada** $= 0$

*Sumidouro -* é um vértice com grau de **saída** $= 0$

*Propriedade*

Todo DAG possui no mínimo uma *fonte* e no mínimo um *sumidouro*.

**Prova:**

- Supondo um DAG sem *sumidouro*, pode-se construir um caminho longo de um vértice para qualquer outro.
- Após $V+1$ vértices, tem-se um ciclo. (Prin. das casas de pombo)
- Ciclo é $\rightarrow$ contradição $\leftarrow$ com o conceito de DAG !!!!!
- Como provar a *fonte* ?

## Dígrafo Acíclico - DAG

*Ordem Total* Somente uma forma de organizar os elementos de um conjunto;

*Ordem Parcial* Várias formas de organizar os elementos de um conjunto.

# *Dígrafo Acíclico - DAG*

*Ordem Total* Somente uma forma de organizar os elementos de um conjunto;

*Ordem Parcial* Várias formas de organizar os elementos de um conjunto.

- Relação de Ordem Parcial - é uma relação binária que possui as seguintes propriedades: **transitiva, reflexiva e antisimétrica.**
  Ex.: $\mathbb{Z}^+, xRy \Leftrightarrow x|y$

# *Dígrafo Acíclico - DAG*

*Ordem Total* Somente uma forma de organizar os elementos de um conjunto;

*Ordem Parcial* Várias formas de organizar os elementos de um conjunto.

- Relação de Ordem Parcial - é uma relação binária que possui as seguintes propriedades: **transitiva, reflexiva e antisimétrica.**
  Ex.: $\mathbb{Z}^+, xRy \Leftrightarrow x|y$

# *Dígrafo Acíclico - DAG*

*Ordenação Topológica (Cormen et. al)*

A ordenação topológica de um DAG $G = (V, E)$ é uma ordem linear de todos os vértices tal que se $G$ contém uma aresta $(u, v)$, então $u$ aparece antes de $v$ na ordem.

# *Dígrafo Acíclico - DAG*

*Ordenação Topológica (Cormen et. al)*

A ordenação topológica de um DAG $G = (V, E)$ é uma ordem linear de todos os vértices tal que se $G$ contém uma aresta $(u, v)$, então $u$ aparece antes de $v$ na ordem.

Assim, os vértices são ordenados sobre uma linha horizontal tal que todas as arestas são direcionadas da esquerda para a direita.

## *Ordenação Topológica*

Seja um conjunto de atividades a serem concluídas, com uma ordem parcial que especifica que certas atividades devem ser concluídas antes de outras iniciarem, como escalonar estas tarefas de forma que as mesmas sejam concluídas respeitando a ordem parcial ?

# *Ordenação Topológica*

Seja um conjunto de atividades a serem concluídas, com uma ordem parcial que especifica que certas atividades devem ser concluídas antes de outras iniciarem, como escalonar estas tarefas de forma que as mesmas sejam concluídas respeitando a ordem parcial ?

Solução: **Ordenação Topológica :-)**

# *Ordenação Topológica*

Seja um conjunto de atividades a serem concluídas, com uma ordem parcial que especifica que certas atividades devem ser concluídas antes de outras iniciarem, como escalonar estas tarefas de forma que as mesmas sejam concluídas respeitando a ordem parcial ?

Solução: **Ordenação Topológica :-)**

*Ordenação Topológica*

Dado um DAG, uma ordenação topológica é uma sequência de vértices obtida a partir do DAG.

## Um Algoritmo para Ordenação Topológica

**Entrada:** DAG $D = (V, A)$

**Saída:** Ordenação topológica $v_1, \ldots, v_n$ dos vértices de D

Executar o algo. DFS(G) para calcular todos os tempos de finalização $f[v]$ para cada vértice $v$;

Ao finalizar cada vértice, inserí-lo na frente de uma lista encadeada;

Retorne a lista encadeada.

## *Um Algoritmo para Ordenação Topológica*

**Entrada:** DAG $D = (V, A)$

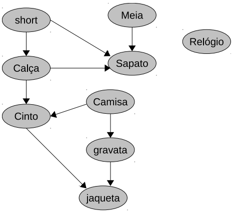
**Saída:** Ordenação topológica $v_1, \ldots, v_n$ dos vértices de D

Executar o algo. DFS(G) para calcular todos os tempos de finalização $f[v]$ para cada vértice $v$;

Ao finalizar cada vértice, inserí-lo na frente de uma lista encadeada;

Retorne a lista encadeada.

*Exemplo: Qual a ordem para se vestir ?*

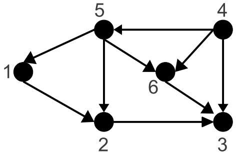



Cada aresta $(u, v)$ significa que $u$ deve ser colocado antes de $v$.

## *Um Algoritmo para Ordenação Topológica*

Ao finalizar o algoritmo, os vértices ordenados aparecem na ordem reversa de seus tempos de finalização.

*Exercício:*



# *Algoritmo de Ordenação Topológica*

## *Aspectos de Implementação*

- Pode existir múltiplas *fontes* no DAG $\rightarrow$ Utilizar uma **Fila de Fontes** (e.g. FIFO).

# *Algoritmo de Ordenação Topológica*

## *Aspectos de Implementação*

- Pode existir múltiplas *fontes* no DAG $\rightarrow$ Utilizar uma **Fila de Fontes** (e.g. FIFO).
- Identificar as fontes que continuam após remover uma dada fonte $\rightarrow$ Vetor de vértices contendo os graus de entrada de cada vértice.

# Algoritmo de Ordenação Topológica

## Aspectos de Implementação

- Pode existir múltiplas *fontes* no DAG $\rightarrow$ Utilizar uma **Fila de Fontes** (e.g. FIFO).
- Identificar as fontes que continuam após remover uma dada fonte $\rightarrow$ Vetor de vértices contendo os graus de entrada de cada vértice.

# *Algoritmo de Ordenação Topológica*

## *Aspectos de Implementação*

- Pode existir múltiplas *fontes* no DAG $\rightarrow$ Utilizar uma **Fila de Fontes** (e.g. FIFO).
- Identificar as fontes que continuam após remover uma dada fonte $\rightarrow$ Vetor de vértices contendo os graus de entrada de cada vértice.

## *Operações - até a Fila de Fontes ficar vazia*

- Remover a fonte e nomeá-lo;
- Nos vértices destino do vértice removido: decrementar as entradas no vetor contendo os graus de entrada;
- Se ao decrementar qualquer entrada ela se torne grau de entrada 0, inserir o vértice correspondente na fila de fontes.

# *A Transposta de um Grafo Direcionado*

*A Transposta de um Grafo Direcionado (Cormen et. al)*

A transposta de um dígrafo $G = (V, E)$ é um grafo $G^T = (V, E^T)$, onde $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$

Ou seja, o grafo $G^T$ equivale ao grafo $G$ com as arestas em ordem reversa.

# Componentes Fortemente Conectados

*Componentes Fortemente Conectados*

Um componente fortemente conectado de um grafo orientado $G = (V, E)$ é um conjunto máximal de vértices $C \subseteq V$ tal que, para todo par de vértices $u, v \in C$, os vértices $u$ e $v$ são acessíveis a partir do outro.

# Componentes Fortemente Conectados

## *Componentes Fortemente Conectados*

Um componente fortemente conectado de um grafo orientado $G = (V, E)$ é um conjunto máximal de vértices $C \subseteq V$ tal que, para todo par de vértices $u, v \in C$, os vértices $u$ e $v$ são acessíveis a partir do outro.

Dado um grafo direcionado, vamos decompó-lo em seus *componentes fortemente conectados*

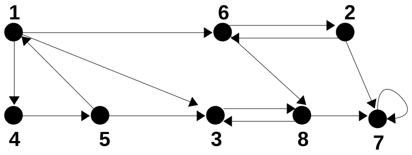
# Componentes Fortemente Conectados

## Componentes Fortemente Conectados

Um componente fortemente conectado de um grafo orientado $G = (V, E)$ é um conjunto máximal de vértices $C \subseteq V$ tal que, para todo par de vértices $u, v \in C$, os vértices $u$ e $v$ são acessíveis a partir do outro.

Dado um grafo direcionado, vamos decompó-lo em seus *componentes fortemente conectados*

*Exemplo:*



# Um Algoritmo Encontrar os Componentes Fortemente Conectados

Seja um dígrafo $G = (V, E)$, o algoritmo encontra os componentes fortemente conectados em tempo linear $\Theta(V + E)$. Para isso, ele emprega o DFS no dígrafo $G$ e o DFS no dígrafo $G^T$.

*Algoritmo [Cormen et. al]*

**Entrada:** Dígrafo $D = (V, A)$

**Saída:** Os componentes fortemente conectados

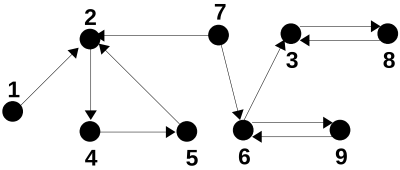
1. Executar o DFS($G$) e encontrar todos os tempos de finalização $f[u]$ para cada vértice $u$;
2. Calcular $G^T$;
3. Executar o DFS($G^T$), mas no laço principal do DFS, considerar os vértices em ordem decrescente de $f[u]$ (como calculado em 1.);
4. Os vértices de cada árvore na Floresta DFS formada em (3.) resultam nos componentes fortemente conectados. Cada componente conectado corresponde a uma árvore DFS.

# *Exemplo*

*Exemplo:*

# *Exercício*

*Utilize o algoritmo acima para encontrar os componentes fortemente conectados dos seguinte grafo:*



# *Referências*


- LEISERSON , Charles E.; STEIN, C.; RIVEST, Ronald L., CORMEN, Thomas H. **Algoritmos: Teoria e Prática**, 1ª.ed. Campus, 2002 (caps. 22 a 26);
- GERSTING, Judith L. **Fundamentos Matemáticos para a Ciência da Computação**. 5a. Edição. LTC Editora, 2004. 616p. ISBN-10: 8521614225. ISBN-13: 978-8521614227.
- ROSEN, Kenneth H. **Matemática Discreta e suas Aplicações**. Tradução da 6a. edição em inglês, McGrawHill, 2009, ISBN 978-85-77260-36-2
- GROSS, Jonthan L., YELLEN, Jay. **Graph Theory and Its Applications**, Second Edition, Chapman and Hall/CRC, 2005.
- SEDGEWICK, Robert. **Algorithms in Java, Part 5: Graph Algorithms**, 3rd Edition, Addison-Wesley Professional, 2003;