# Problema da Ordenação Topológica
# Grafos Direcionados Acíclicos

**Paulo Mateus Luza Alves**

Departamento de Informática – Universidade Federal do Paraná (UFPR)

pmla20@inf.ufpr.br (GRR20203945)

## 1. O PROBLEMA DA ORDENAÇÃO TOPOLÓGICA

Uma ordenação topológica de um grafo direcionado G é uma permutação $(v_1, \ldots, v_n)$ de V(G) que "respeita a direção dos arcos" de G, ou seja, "os filhos nunca vêm antes dos pais". Logo,

$$i < j, \; para\; todo\; (v_i, v_j) \in A(G)$$

Portanto, podemos afirmar que um mesmo grafo pode ter mais de uma ordenação topológica.

Uma maneira de obter uma das ordenações topológicas de um grafo direcionado acíclico G é apresentar o reverso da pós-ordem de uma floresta direcionada resultante de uma busca em profundidade iniciada em qualquer vértice deste grafo.

## 2. A IMPLEMENTAÇÃO

A seção da implementação será dividida em 3 partes, sendo elas:

- Tratamento dos arquivos de entrada;
- Verificação de coerência (se o grafo enviado é um grafo direcionado acíclico);
- Construção e apresentação da ordenação topológica.

Com estes 3 tópicos abrangemos toda a construção e lógica por trás da implementação da solução da busca de uma ordenação topológica de um grafo direcionado acíclico.

### 2.1 TRATAMENTO DOS ARQUIVOS DE ENTRADA

Como informado na especificação do trabalho, a linguagem de descrição dos grafos de entrada é a DOT que é implementada pelo pacote de software GraphViz.

Por este motivo, para a leitura do arquivo de entrada e também para estrutura de dados utilizada dentro do algoritmo, foi utilizada a biblioteca *cgraph*, a qual faz uso do pacote de software GraphViz, possuindo um *parser* para arquivos .dot além das estruturas de dados necessárias para armazenar e manipular o grafo dentro do algoritmo.

## 2.2 VERIFICAÇÃO DE COERÊNCIA

Após a leitura do arquivo, é necessário confirmar se o grafo enviado se trata de um grafo direcionado acíclico, pois caso contrário, a ordenação topológica não pode ser computada. Desta forma, esta verificação foi separada em duas partes, sendo a primeira verificar se o grafo é de fato direcionado e a segunda verificar se ele é acíclico.

Para a primeira verificação, foi utilizada a função *agisdirect* da biblioteca *cgraph.* Esta função consiste basicamente em pegar o tipo do grafo, que é informado no arquivo .dot , e verificar se ele é *graph* ou *digraph*, e caso o tipo seja o segundo o grafo é direcionado.

Já a segunda verificação foi feita dentro da função *get_topology* que também é responsável por retornar a ordenação topológica do grafo. Como esta função é baseada em uma DFS, caso algum vértice no estado 1 seja encontrado durante as chamadas recursivas, podemos confirmar a caracterização de uma aresta de retorno, pois nesta situação, estamos retornando para um vértice que ainda possui filhos em processamento, ou seja, estamos voltando para um ancestral na árvore, logo, uma aresta de retorno.

Portando, durante a execução da *get_topology* caso esta situação seja encontrada, a execução do *loop* de recursões é encerrada, e a função retorna um erro indicando que o grafo possui ciclo.

## 2.3 CONSTRUÇÃO E APRESENTAÇÃO DA ORDENAÇÃO TOPOLÓGICA

Como comentado na seção anterior, a função *get_topology*, além de verificar a existência de um ciclo, tem a responsabilidade de computar e retornar a ordenação topológica do grafo. Isto é feito, como já mencionado, por meio de uma DFS. Logo, a cada momento que um vértice é colocado no estado 2, ou seja, encerra seu processamento, ele também é adicionado em um vetor que armazena a ordenação atual.

Como essa adição é feita no final do vetor, podemos dizer que ele armazena a ordenação de trás para frente. Além disso, vale ressaltar, que o algoritmo de busca em profundidade implementado inicia a busca com o o primeiro vértice apresentado no arquivo .dot, logo, de acordo com o seguinte exemplo:

```
strict digraph digraph {
        a -> b
        a -> c
        b -> d
        c -> d
}
```

A busca irá iniciar no vértice *a*, e o primeiro vértice de sua vizinhança que será chamado é o vértice *b*, e posteriormente o vértice *c*.

Além disso, como a função armazena a ordenação topológica de trás para frente, a função de escrita desta ordenação realiza o *print* a partir da última posição do *array* até a primeira.

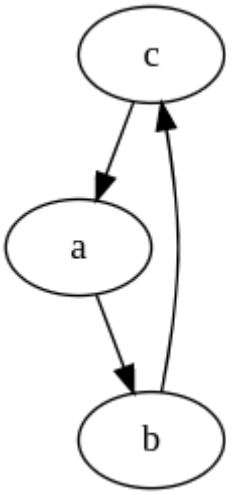
## 3. EXEMPLOS

Dentro da pasta *examples* do projeto podemos encontrar os exemplos utilizados para testar o algoritmo, vou apresentar 2 aqui.

### 3.1 EXEMPLO DE GRAFO COM CICLO

Seja o arquivo .dot referenciado a baixo com a respectiva representação visual

```
strict digraph digraph {
        c -> a
        a -> b
        b -> c
}
```




Como é bem visível, o grafo possui um ciclo, logo ao executar o algoritmo nele, o retorno deve ser um erro. Segue abaixo o retorno do algoritmo

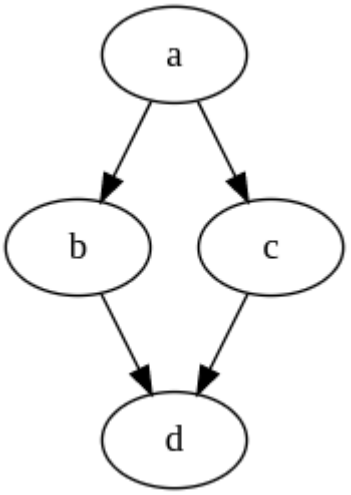
```
The given graph is not a DAG. Please, send a DAG (Directed Acyclic graph) instead.
```

Como comentado, o retorno foi um erro, finalizando corretamente de acordo com a especificação.

## 3.2 EXEMPLO DE GRAFO SEM CICLO

Seja o arquivo .dot referenciado a baixo com a respectiva representação visual

```
strict digraph digraph {
        a -> b
        a -> c
        b -> d
        c -> d
}
```




Como é visível, o grafo não possui ciclo e portanto deve retornar uma ordenação topológica. Segue abaixo o retorno do algoritmo

```
a c b d
```

Como comentado, o retorno foi uma ordenação topológica partindo do vértice *a*, caminhando para o vértice *b* primeiramente.