ESCOLA DE PRIMAVERA DA MARATONA SBC DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG

UFMG

UFV

UNIFEI

USP Universidade de São Paulo

UFU

APOIO:

beecrowd

SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

ORDENAÇÃO TOPOLÓGICA

Por: Ulisses Andrade Carvalho

CONTEÚDOS

01 - PROBLEMA MOTIVADOR

Dado um conjunto de N tarefas (dependentes entre si), em que ordem podemos executar estas tarefas ?

01 - PROBLEMA MOTIVADOR

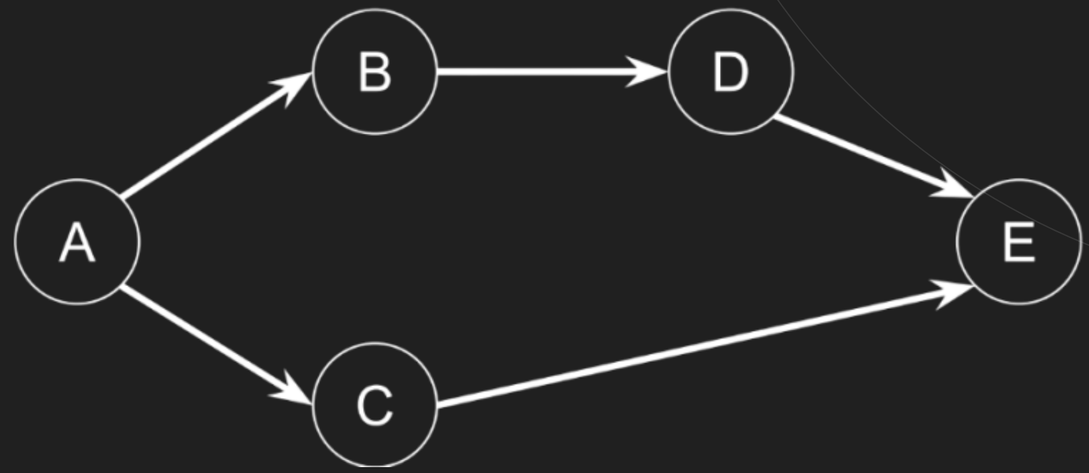
Dado um conjunto de N tarefas (dependentes entre si), em que ordem podemos executar estas tarefas ?

- As relações de dependência podem ser modeladas através de um digrafo (grafo direcionado);
- Sendo os vertices as tarefas e os arcos as relações de dependência entre as tarefas.

01 - PROBLEMA MOTIVADOR

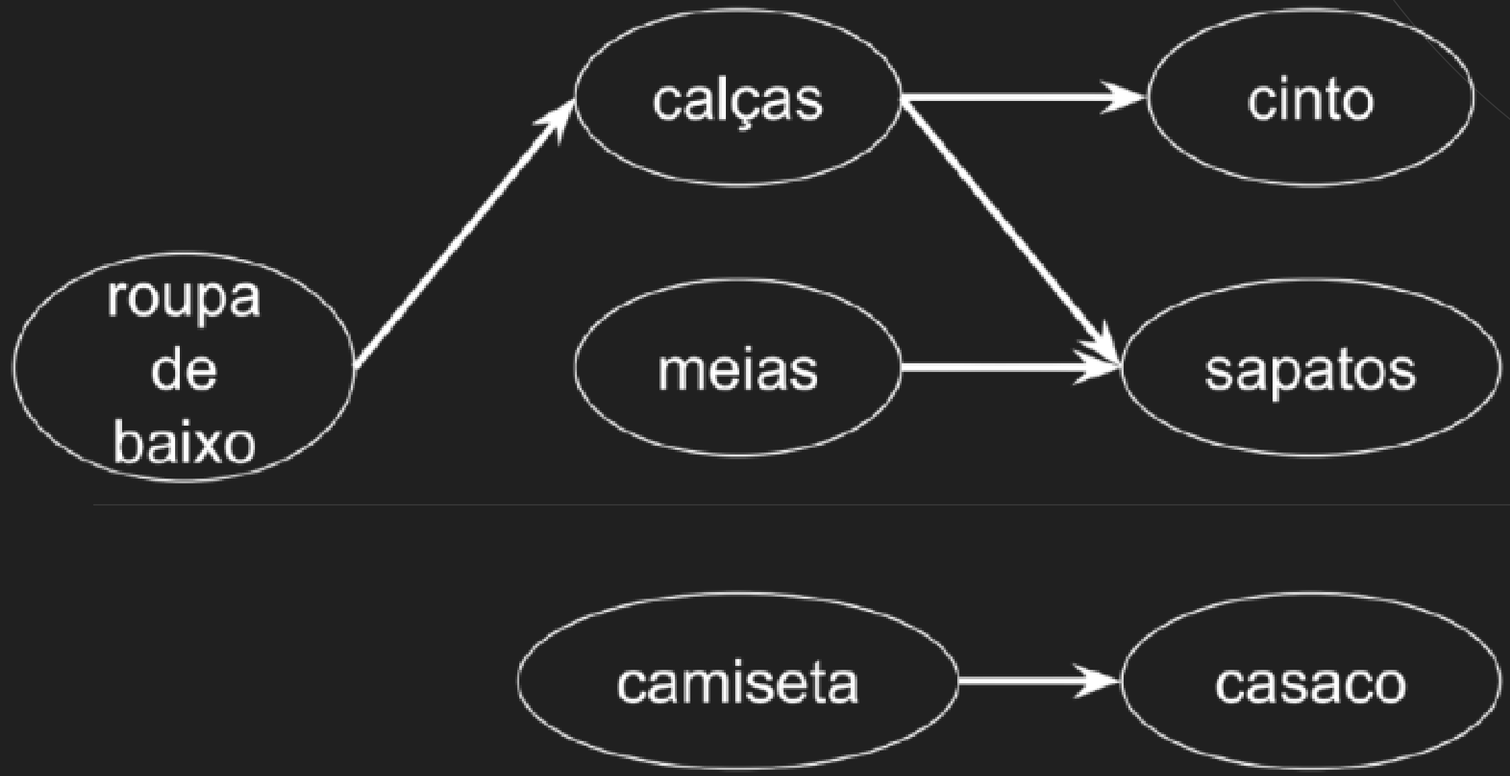
Exemplo:

- B depende de A
- C depende de A
- D depende de B
- E depende de C
- E depende de D



01 - PROBLEMA MOTIVADOR

- Exemplo do processo de se vestir:



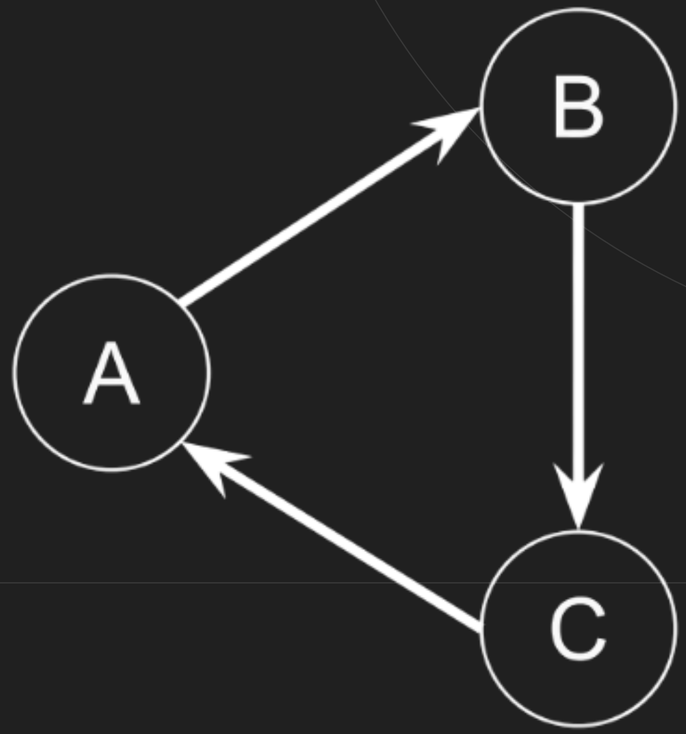
01 - PROBLEMA MOTIVADOR

Dado um conjunto de N tarefas (dependentes entre si), em que ordem podemos executar estas tarefas ?

02 - DEFINIÇÃO DO ALGORITMO

- A ordenação topológica nada mais é do que uma **permutação** dos vértices de um grafo direcionado que **respeita as relações de dependências** impostas pelos arcos;

- Caso um **grafo possua ciclos** ou seja não direcionado, não é possível estabelecer uma relação de precedência entre os vértices, e portanto, **é impossível estabelecer uma ordenação topológica**.



02 - DEFINIÇÃO DO ALGORITMO

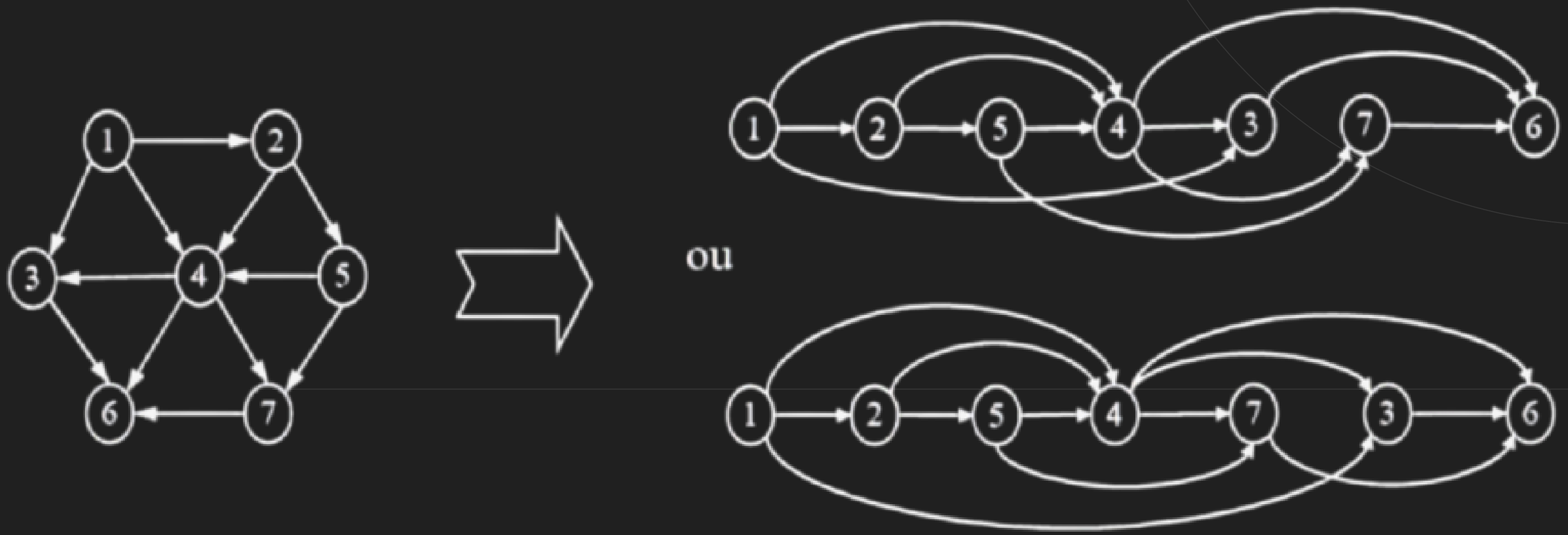
Definição:

- Ordenação linear dos vértices de um DAG (Grafo Acíclico Dirigido) tal que, se existe um arco (u, v) no grafo, então u aparece antes de v;

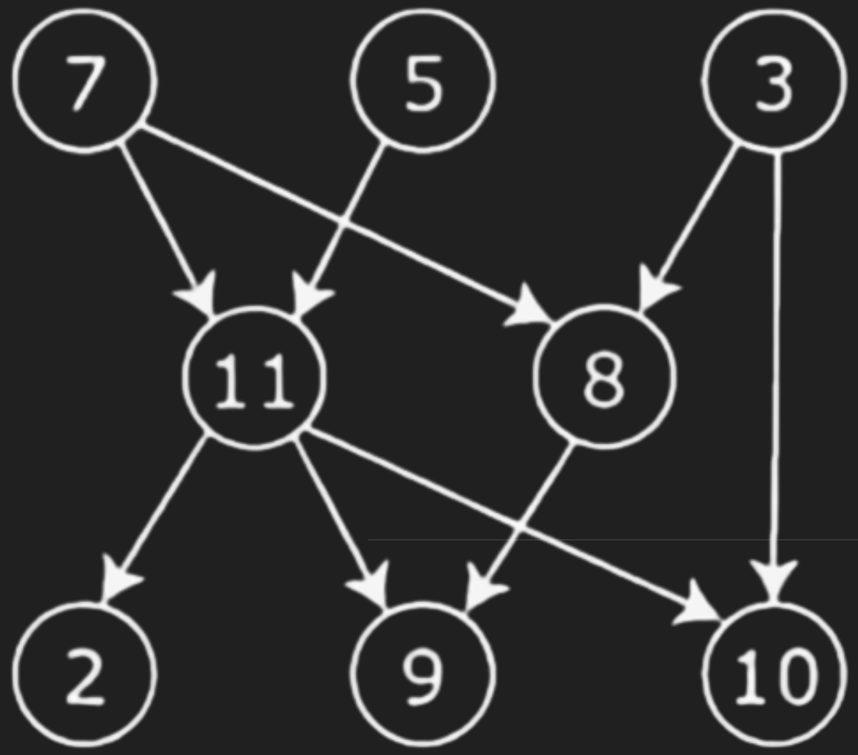
- Em outras palavras, é uma ordenação linear de vértices na qual cada vértice v precede os vértices que formam seu fecho transitivo direto (descendentes de v);

- Cada DAG possui uma ou mais ordenações topológicas.

02 - DEFINIÇÃO DO ALGORITMO



02 - DEFINIÇÃO DO ALGORITMO



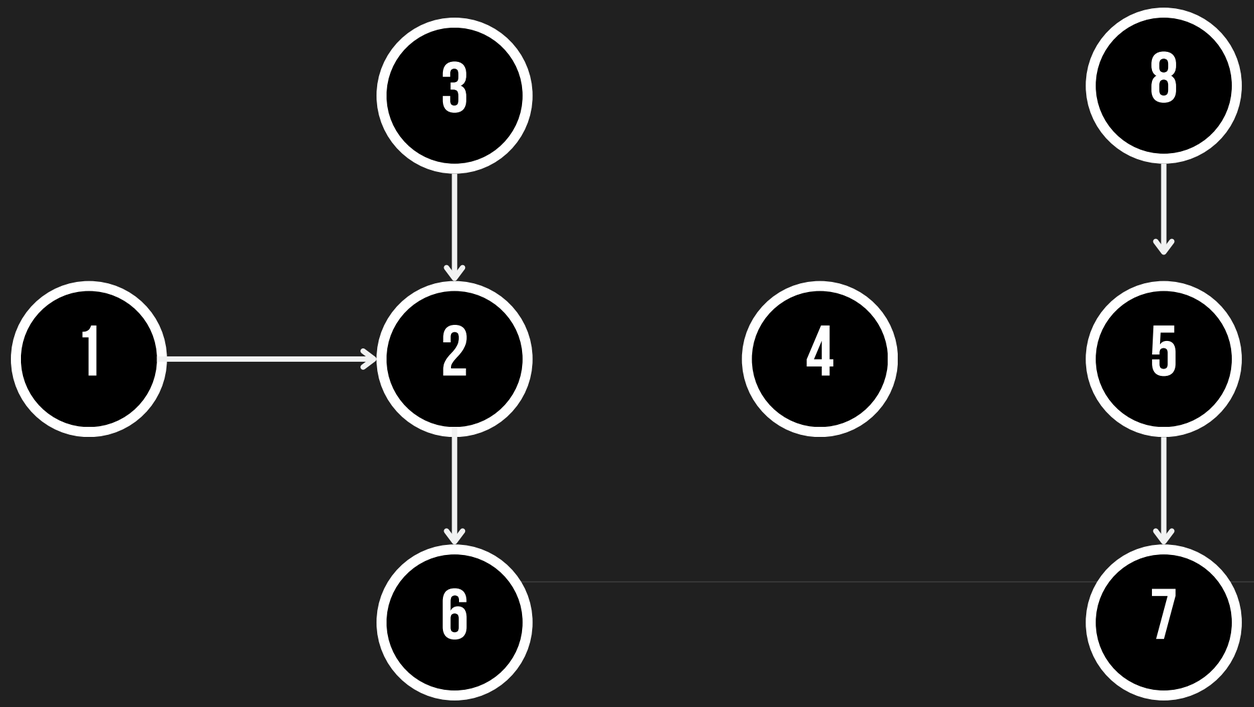
- 7, 5, 3, 11, 8, 2, 9, 10 (visual esquerda para direita e de cima para baixo);
- 3, 5, 7, 8, 11, 2, 9, 10 (vértice de menor número disponível primeiro);
- 5, 7, 3, 8, 11, 10, 9, 2 (menor número de arestas primeiro);
- 7, 5, 11, 3, 10, 8, 9, 2 (vértice de maior número disponível primeiro);

03 - FUNCIONAMENTO DO ALGORITMO

Algoritmo de Kahn:

- Esse algoritmo data de 1962 e é baseado na estratégia de eliminação das fontes, determinando a cada instante os vértices que não possuem arcos;
- A cada vértice inserido na solução, todos seus arcos correspondentes são removidos do grafo;
- Também detecta a existência de ciclos no grafo.

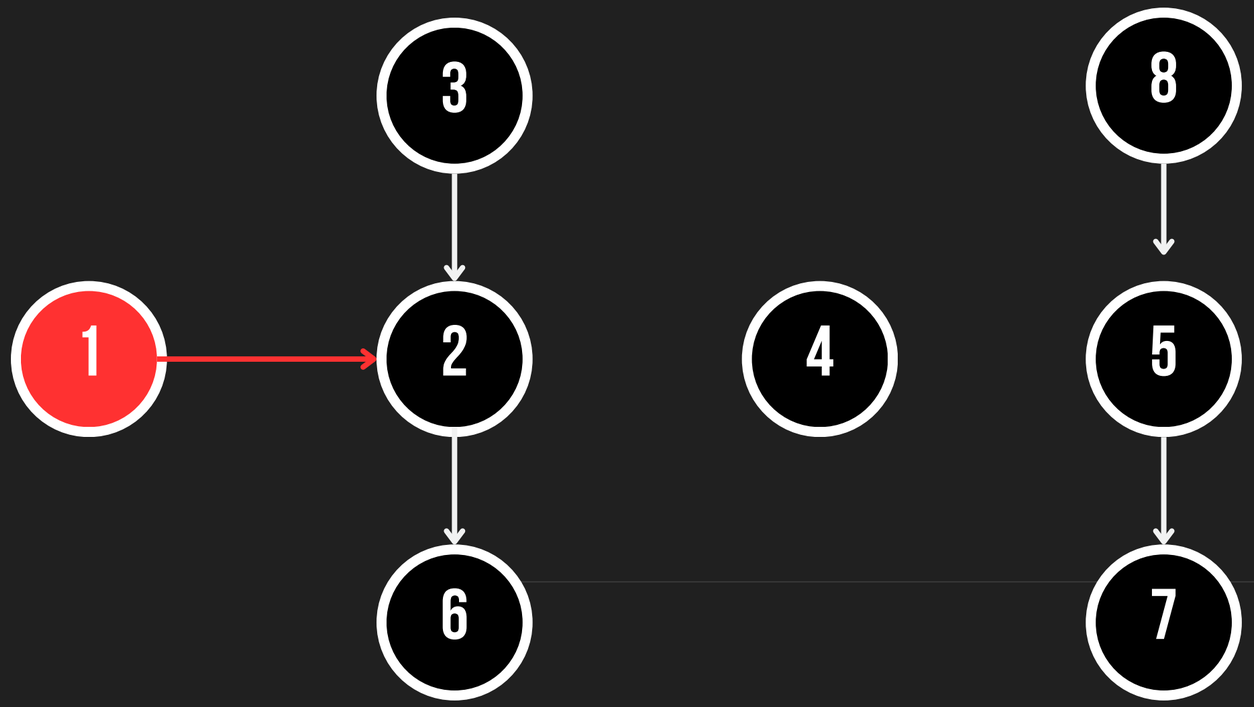
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	0
2	2
3	0
4	0
5	1
6	1
7	1
8	0

Ordem:

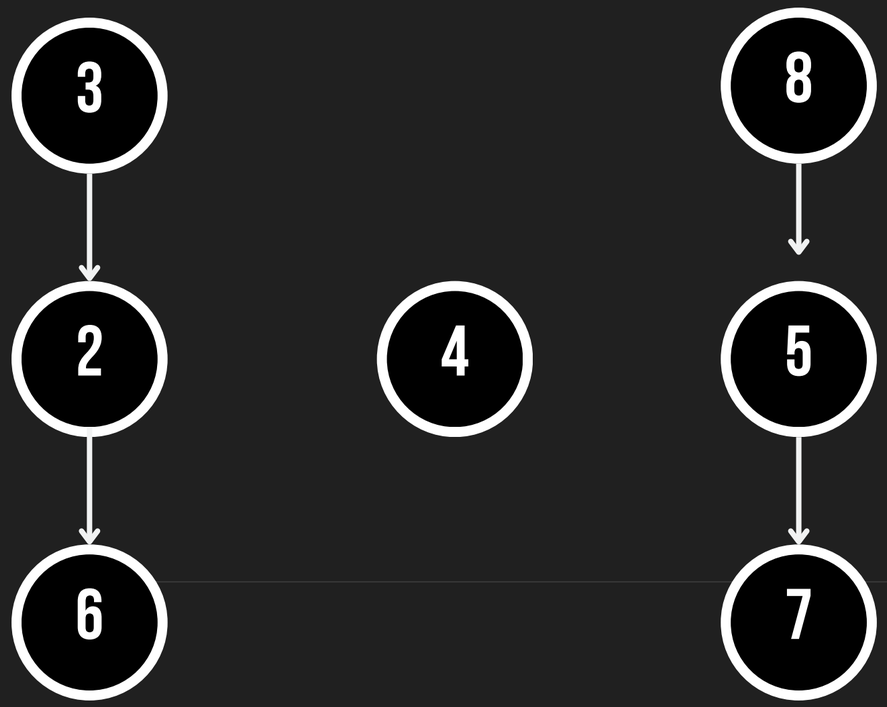
03 - FUNCIONAMENTO DO ALGORITMO



Ordem: 1

Vértice	Grau
1	0
2	1
3	0
4	0
5	1
6	1
7	1
8	0

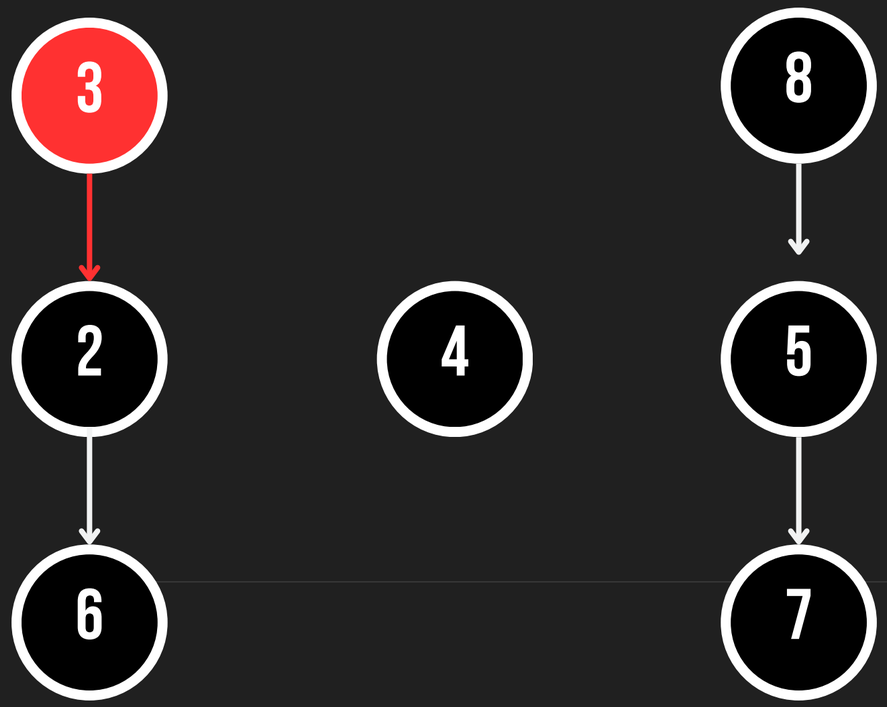
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	1
3	0
4	0
5	1
6	1
7	1
8	0

Ordem: 1

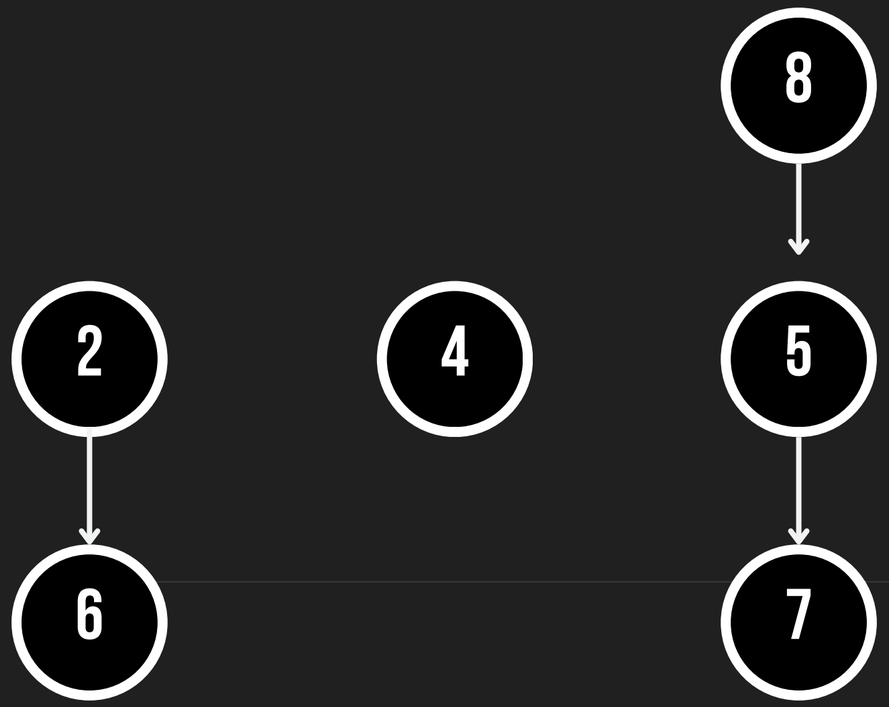
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	0
3	0
4	0
5	1
6	1
7	1
8	0

Ordem: 1 3

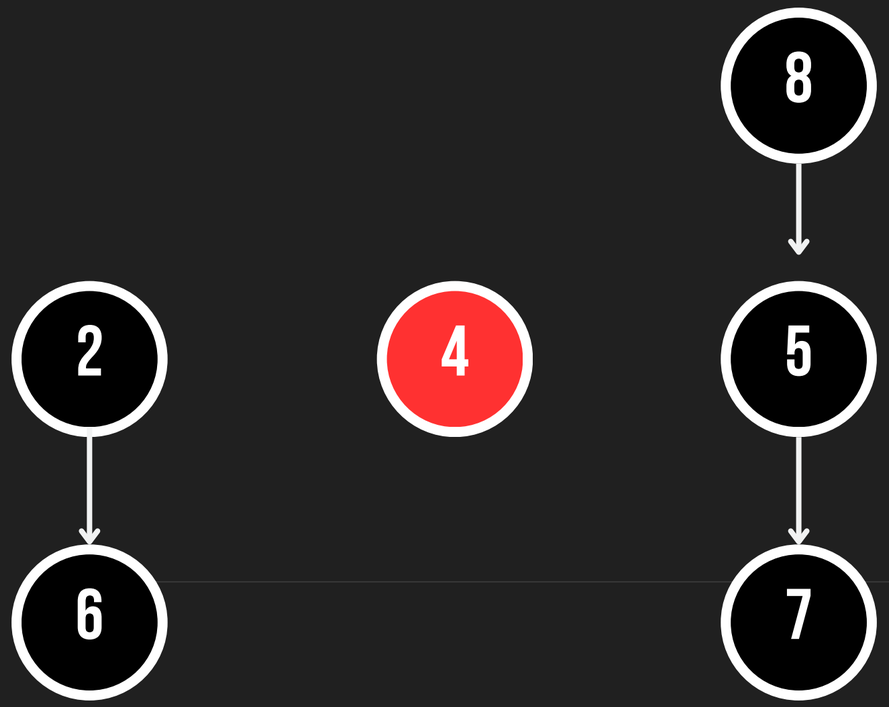
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	0
3	-
4	0
5	1
6	1
7	1
8	0

Ordem: 1 3

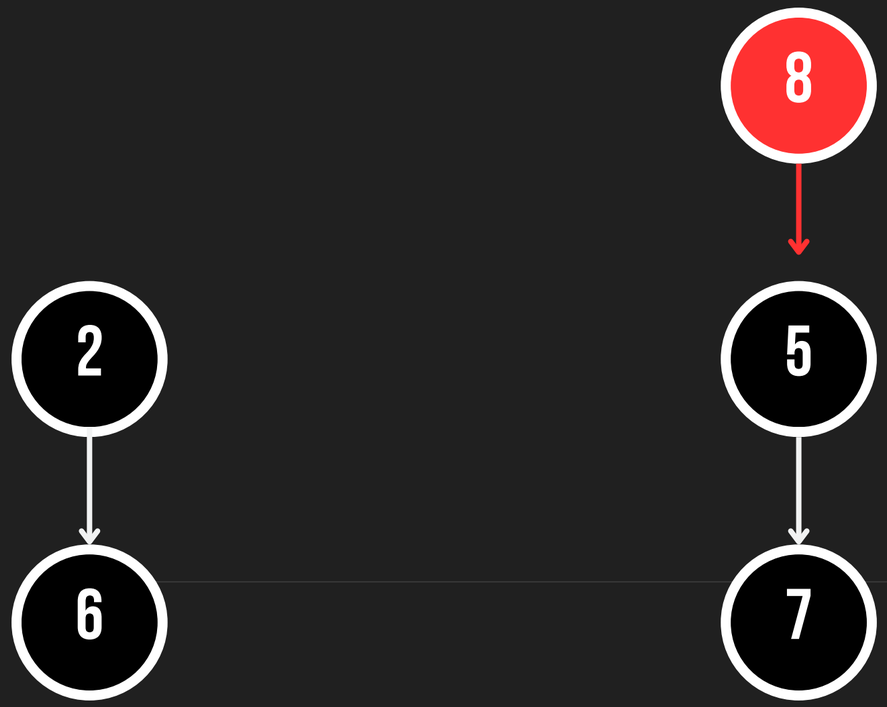
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	0
3	-
4	0
5	1
6	1
7	1
8	0

Ordem: 1 3 4

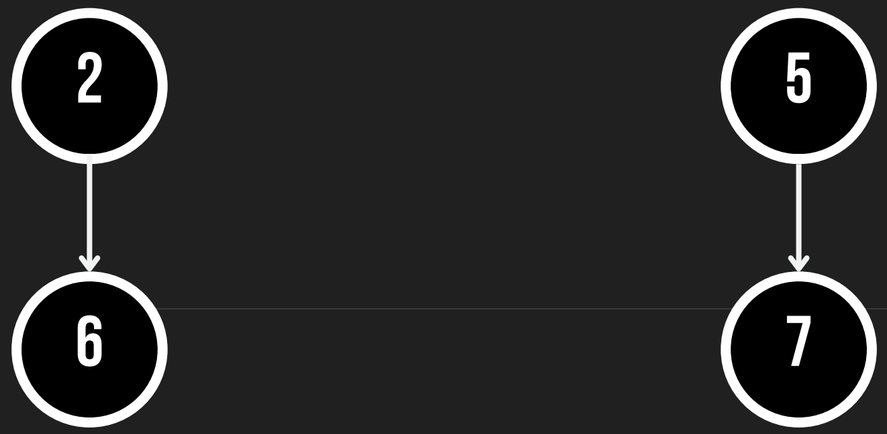
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	0
3	-
4	-
5	0
6	1
7	1
8	0

Ordem: 1 3 4 8

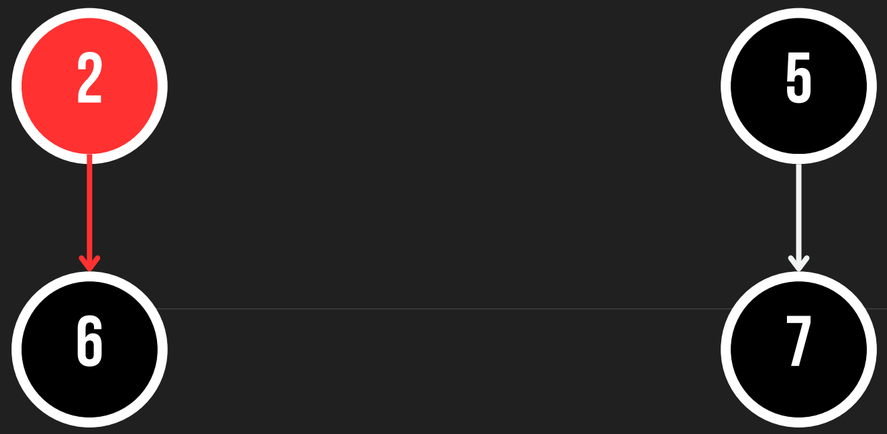
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	0
3	-
4	-
5	0
6	1
7	1
8	-

Ordem: 1 3 4 8

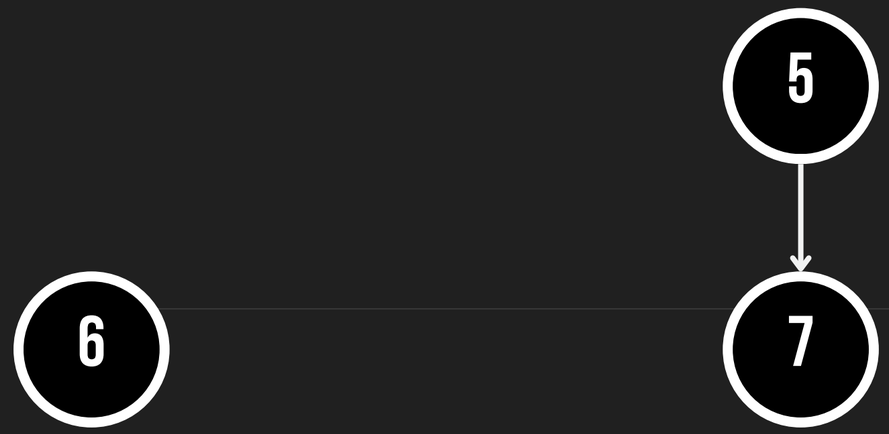
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	0
3	-
4	-
5	0
6	0
7	1
8	-

Ordem: 1 3 4 8 2

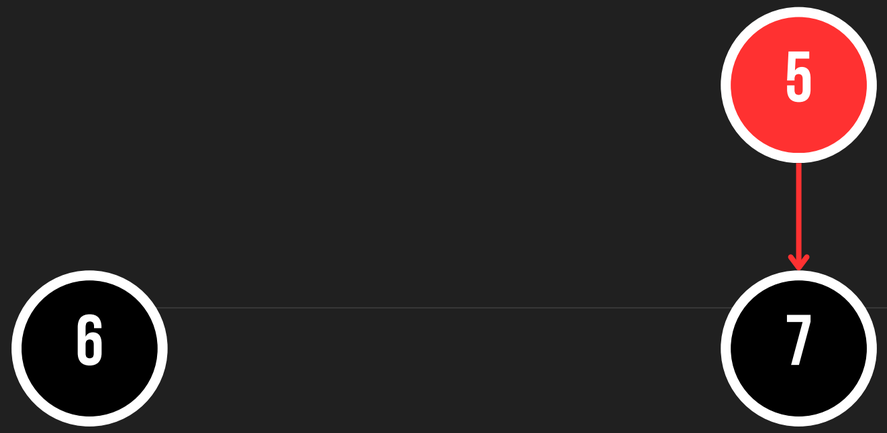
03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Grau
1	-
2	-
3	-
4	-
5	0
6	0
7	1
8	-

Ordem: 1 3 4 8 2

03 - FUNCIONAMENTO DO ALGORITMO

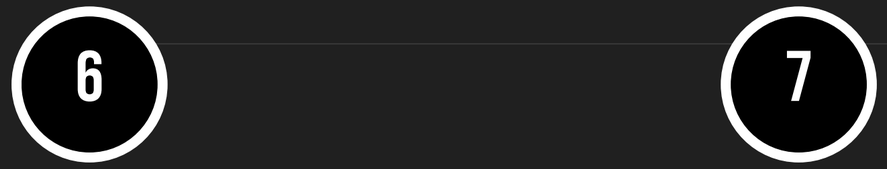


Vértice	Grau
1	-
2	-
3	-
4	-
5	0
6	0
7	0
8	-

Ordem: 1 3 4 8 2 5

03 - FUNCIONAMENTO DO ALGORITMO

Vértice	Grau
1	-
2	-
3	-
4	-
5	-
6	0
7	0
8	-



Ordem: 1 3 4 8 2 5

03 - FUNCIONAMENTO DO ALGORITMO

Vértice	Grau
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-

Ordem: 1 3 4 8 2 5 6 7

04 - ALGORITMO

Algoritmo de Kahn:

```cpp
#define MAXN 50010
vector<int> grauEntrada(MAXN, 0); // Vetor que armazena o grau de entrada de cada vértice
vector<vector<int>> adj(MAXN); // Listas de adjacência

vector<int> ordenacaoTopologica(int V) {
    vector<int> L; // Lista de ordenação topológica
    queue<int> S; // Conjunto de vértices sem arcos de entrada
    // Insere os vértices sem arcos de entrada no conjunto S
    for (int i = 0; i < V; ++i) {
        if (grauEntrada[i] == 0) {
            S.push(i);
        }
    }
    // Enquanto S não estiver vazio
    while (!S.empty()) {
        int v = S.front(); S.pop(); // Remove um vértice de S
        L.push_back(v); // Insere o vértice em L

        for (int w : adj[v]) { // Para cada arco (v, w)
            grauEntrada[w]--; // Remove o arco v->w
            if (grauEntrada[w] == 0) {
                S.push(w); // Insere w em S se não possuir mais arcos de entrada
            }
        }
    }
    if (L.size() != V) throw runtime_error("O grafo possui pelo menos um ciclo!");
    return L;
}
```

```cpp
int main() {
    int V; // Número de vértices
    cin >> V;

    // Inicializa o grau de entrada de cada vértice
    int ini, fim;
    while(V--){
        cin >> ini >> fim;
        grauEntrada[fim]++;
        adj[ini].push_back(fim);
    }

    try {
        vector<int> ordenacao = ordenacaoTopologica(V);
        cout << "Ordenação topológica: ";
        for (int v : ordenacao) {
            cout << v << " ";
        }
        cout << endl;
    } catch (const runtime_error &e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

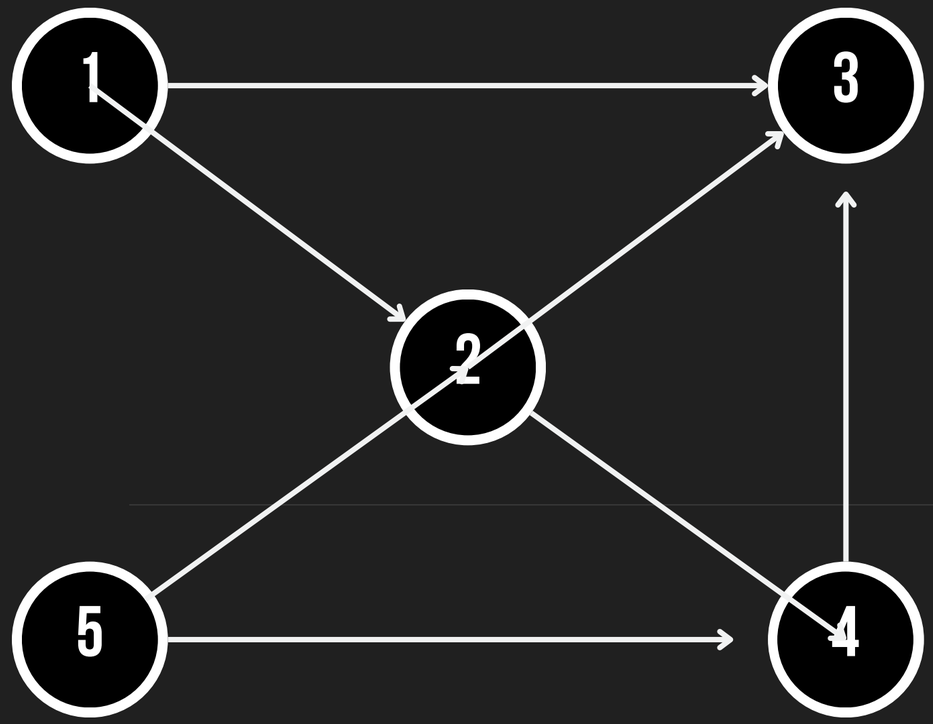
03 - FUNCIONAMENTO DO ALGORITMO

DFS para ordenação topológica:

- Algoritmo proposto por Robert Tarjan, que utiliza uma busca em profundidade. O código consiste em:

- Iniciando a visita em v, visite todos os seus adjacentes (v, w) chamando a função DFS para w;
- Após finalizar a lista de adjacências de cada vértice v, sendo processado, adiciona-se v ao começo da lista;

03 - FUNCIONAMENTO DO ALGORITMO

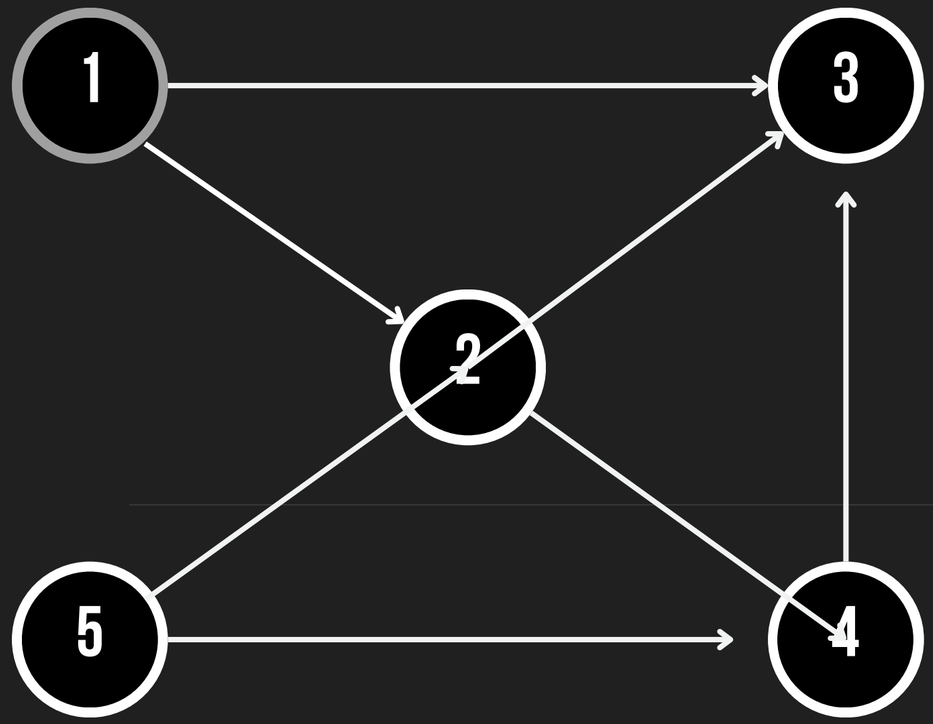


Vértice	Pai	Início	Fim
1	-1		
2	-1		
3	-1		
4	-1		
5	-1		

Ordem:

Tempo: 0

03 - FUNCIONAMENTO DO ALGORITMO

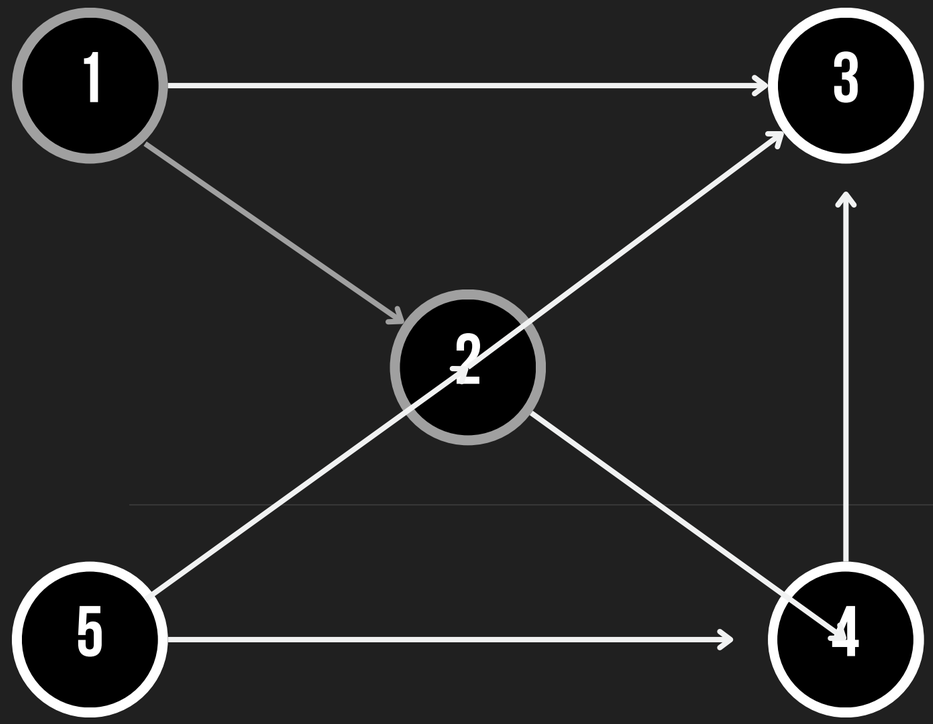


Vértice	Pai	Início	Fim
1	-1	1	
2	-1		
3	-1		
4	-1		
5	-1		

Ordem:

Tempo: 1

03 - FUNCIONAMENTO DO ALGORITMO

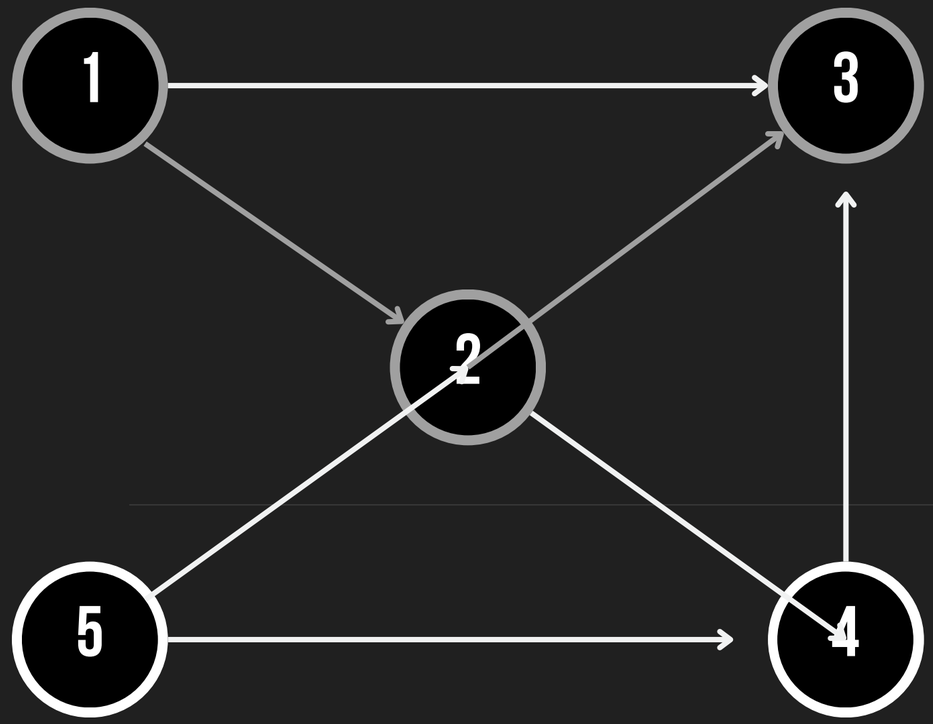


Vértice	Pai	Início	Fim
1	-1	1	
2	1	2	
3	-1		
4	-1		
5	-1		

Ordem:

Tempo: 2

03 - FUNCIONAMENTO DO ALGORITMO

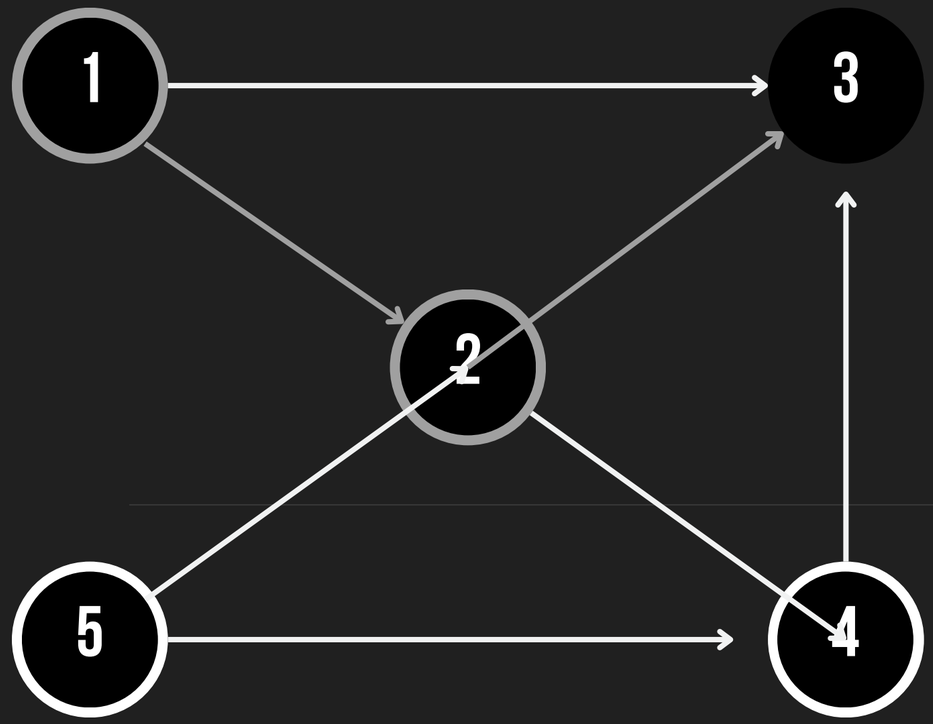


Vértice	Pai	Início	Fim
1	-1	1	
2	1	2	
3	2	3	
4	-1		
5	-1		

Ordem:

Tempo: 3

03 - FUNCIONAMENTO DO ALGORITMO

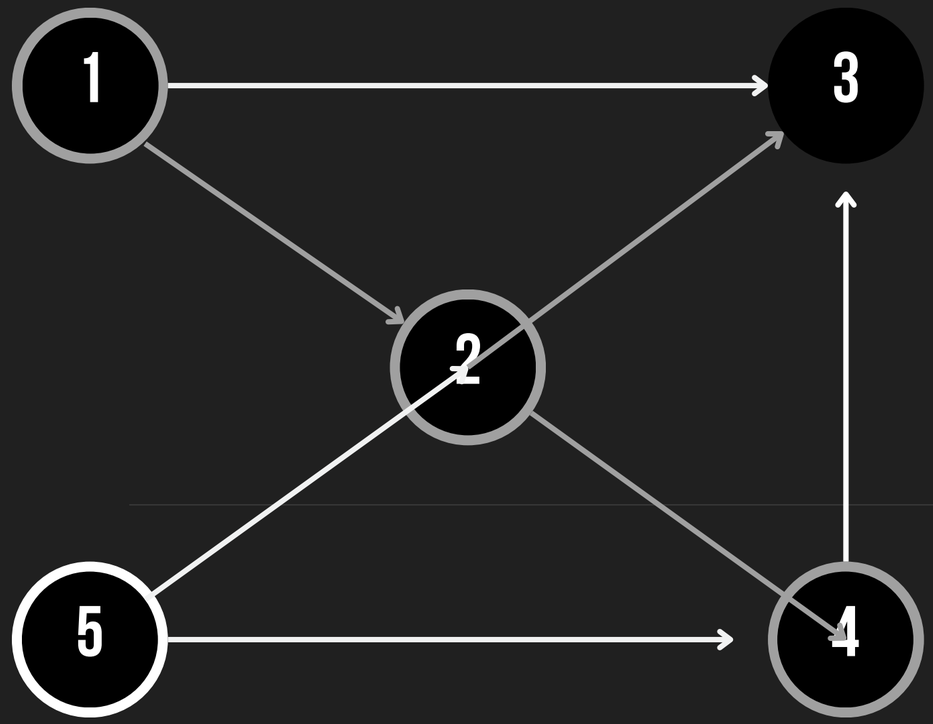


Vértice	Pai	Início	Fim
1	-1	1	
2	1	2	
3	2	3	4
4	-1		
5	-1		

Ordem: 3

Tempo: 4

03 - FUNCIONAMENTO DO ALGORITMO

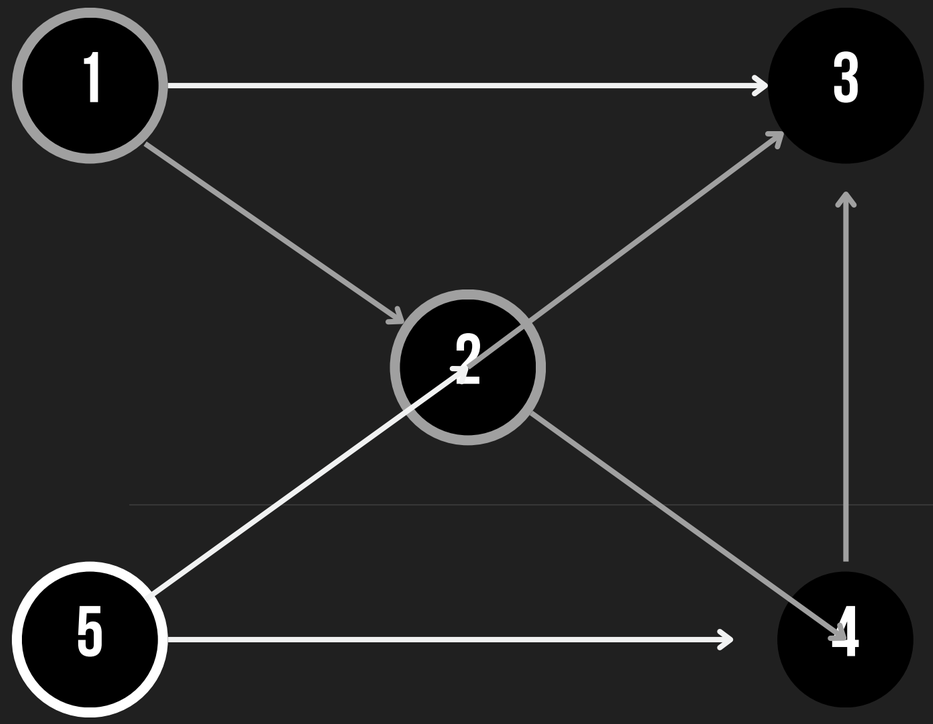


Vértice	Pai	Início	Fim
1	-1	1	
2	1	2	
3	2	3	4
4	2	5	
5	-1		

Ordem: 3

Tempo: 5

03 - FUNCIONAMENTO DO ALGORITMO

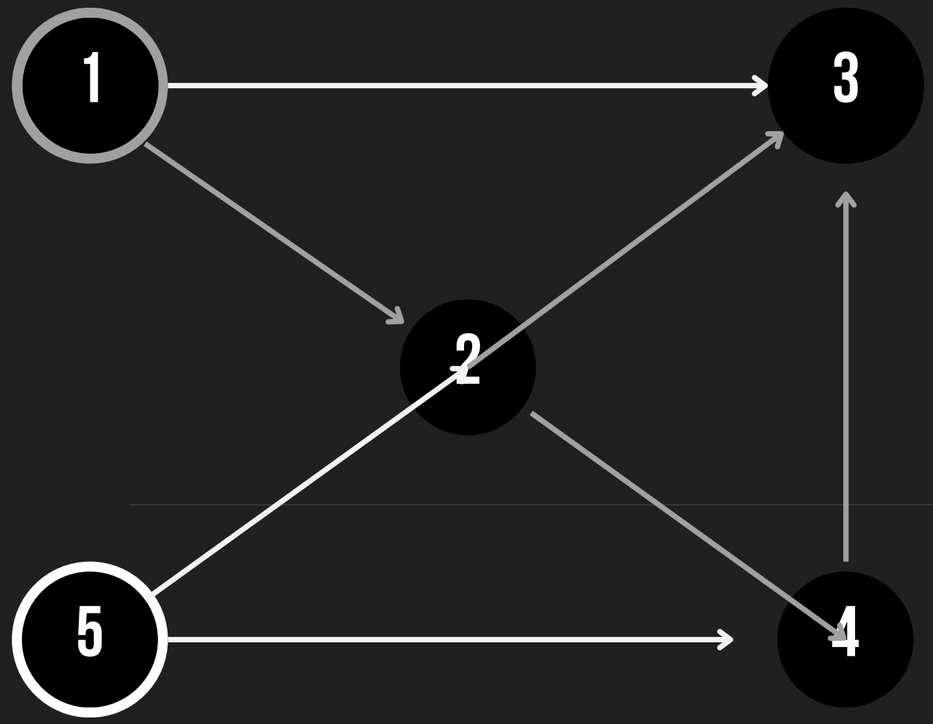


Vértice	Pai	Início	Fim
1	-1	1	
2	1	2	
3	2	3	4
4	2	5	6
5	-1		

Ordem: 4 -> 3

Tempo: 6

03 - FUNCIONAMENTO DO ALGORITMO

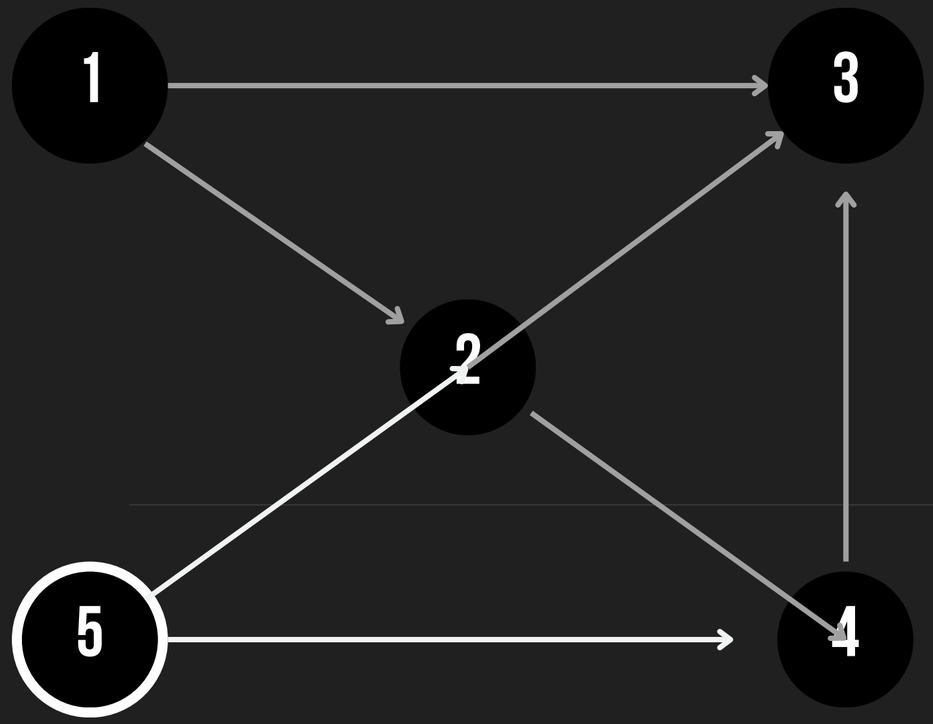


Vértice	Pai	Início	Fim
1	-1	1	
2	1	2	7
3	2	3	4
4	2	5	6
5	-1		

Ordem: 2 -> 4 -> 3

Tempo: 7

03 - FUNCIONAMENTO DO ALGORITMO

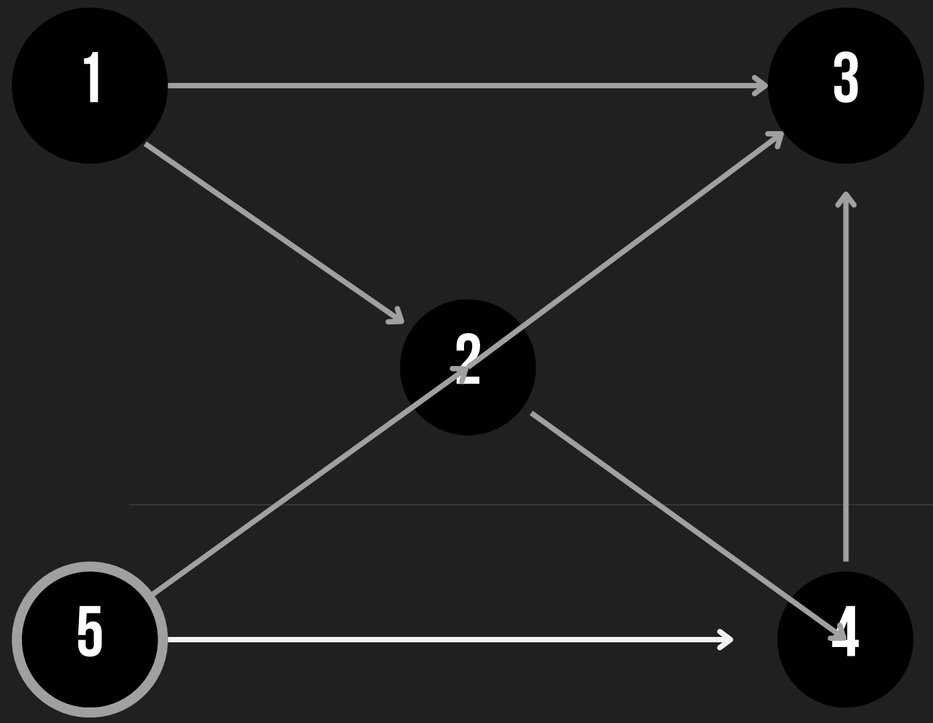


Vértice	Pai	Início	Fim
1	-1	1	8
2	1	2	7
3	2	3	4
4	2	5	6
5	-1		

Ordem: 1 -> 2 -> 4 -> 3

Tempo: 8

03 - FUNCIONAMENTO DO ALGORITMO

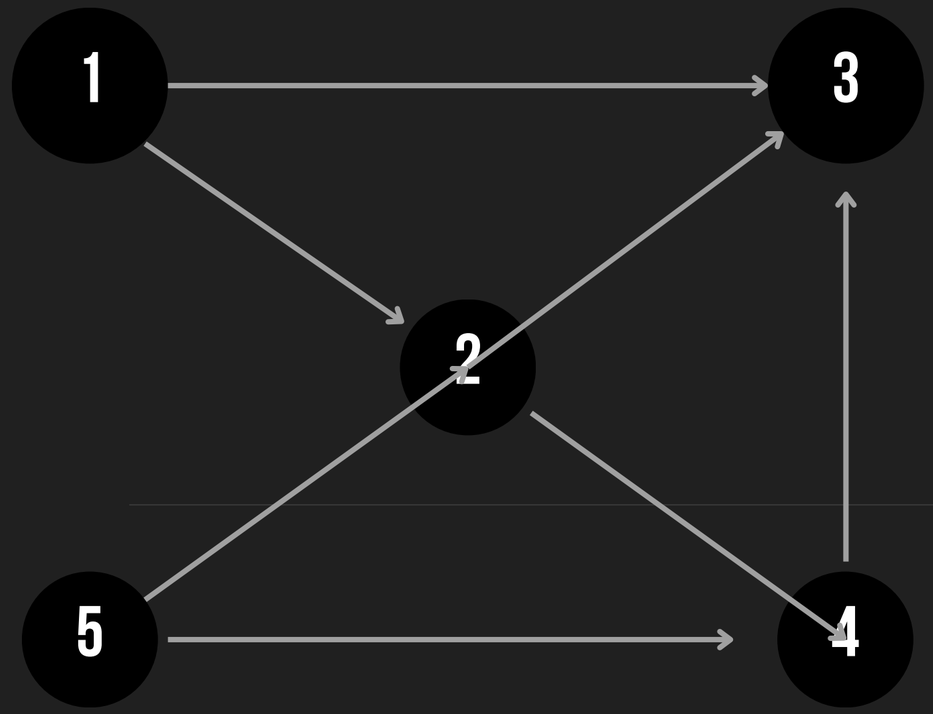


Vértice	Pai	Início	Fim
1	-1	1	8
2	1	2	7
3	2	3	4
4	2	5	6
5	-1	9	

Ordem: 1 -> 2 -> 4 -> 3

Tempo: 9

03 - FUNCIONAMENTO DO ALGORITMO



Vértice	Pai	Início	Fim
1	-1	1	8
2	1	2	7
3	2	3	4
4	2	5	6
5	-1	9	10

Ordem: 5 -> 1 -> 2 -> 4 -> 3

Tempo: 10

04 - ALGORITMO

DFS para ordenação topológica:

```cpp
void visite(int v) {
    if (temporario[v]) {
        throw runtime_error("Detecção de ciclo: o grafo possui um ciclo!");
    }

    if (!definitivo[v]) {
        temporario[v] = true; // Marque temporariamente
        for (int w : adj[v]) {
            visite(w); // Chamada recursiva
        }
        definitivo[v] = true;    // Marque definitivamente
        L.insert(L.begin(), v);  // Adicione no inicio de L
    }
}

vector<int> ordenacaoTopologica(int V) {
    for (int v = 0; v < V; ++v) {
        if (!definitivo[v]) {
            visite(v); // Visite o vértice
        }
    }

    return L; // Retorna a ordenação topológica
}
```

```cpp
int main() {
    int V; // Número de vértices
    cin >> V;

    int ini, fim;
    while(V--){
        cin >> ini >> fim;
        adj[ini].push_back(fim);
    }

    try {
        vector<int> ordenacao = ordenacaoTopologica(V);
        cout << "Ordenação topológica: ";
        for (int v : ordenacao) {
            cout << v << " ";
        }
        cout << endl;
    } catch (const runtime_error &e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

05 - RESOLUÇÃO DO PROBLEMA MOTIVADOR

Problema Beecrowd 2403 - Escalonamento Ótimo

Descrição do problema: https://judge.beecrowd.com/pt/problems/view/2403

Solução do problema: https://drive.google.com

06 - OUTRAS APLICAÇÕES

- **Dependência entre tarefas:** uma ordenação topológica é uma sequência válida de tarefas;
- **Pré-requisitos de disciplinas:** uma ordenação topológica é uma sequência válida para se cursar as disciplinas;
- **Instalação de pacotes:** uma ordenação topológica é uma sequência válida para instalação de pacotes com dependências;
- **Compilação de sistemas:** uma ordenação topológica é uma sequência válida para compilar bibliotecas com dependências;

07 - PROBLEMAS

Cadeia Alimentar - https://judge.beecrowd.com/pt/problems/view/1903

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

GCC.CEFETMG

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg