

# Trabalho Prático - Grafos

24/05/2025

## Especificações

### ✓ Escolha do Tema:

Tema **9. Gestão de Conflitos em Sistemas de Dependência**, esse tema envolve **detecção de ciclos, ordenação topológica e execução segura de tarefas em sistemas com dependências**, como em gerenciadores de pacotes, fluxos de trabalho ou pipelines de dados.

---

## Visão geral

### 🎯 Problema a Ser Resolvido

Sistemas que possuem tarefas interdependentes (como instalação de pacotes, workflows de CI/CD ou módulos de software) frequentemente enfrentam **conflitos de dependência** ou **ciclos** que impedem a execução segura. A proposta é criar uma ferramenta que:

- Modele o sistema como um grafo dirigido
- Detecte e exiba ciclos (conflitos)
- Forneça uma execução segura (ordenada topologicamente)

## 📌 Plano Completo de Execução


### 17 Etapa 1 – Modelagem e Planejamento da Solução (Entregável 1)

#### 1.1. Descrição do Problema

Criar uma ferramenta que represente tarefas e suas dependências como um grafo dirigido. O sistema deverá ser capaz de detectar **ciclos** (conflitos) e produzir uma **ordem de execução segura** das tarefas (ordenamento topológico).

---

## 1.2. Justificativa da Modelagem

- Os grafos são ideais para representar **relações de dependência**.
  - A presença de ciclos impede a execução segura.
  - Algoritmos bem estabelecidos como **Tarjan** e **Kahn** permitem detectar e resolver esses conflitos.
  -  Ferramentas como *NetworkX* (em Python) ou *JGraphT* (em Java) facilitam a implementação.
- 

## 1.3. Planejamento

- **Linguagem:** Python (ou considerar Java se desejado)
- **Biblioteca de grafos:** NetworkX (Python) ou JGraphT (Java)
- **Interface gráfica:** Tkinter, PyQt, ou Streamlit (mais simples)
- **Fontes de dados:** arquivos **.json**, **.csv** ou **.txt**
- **Diagramas UML:** Casos de Uso, Classes e Sequência

---

## 1.4. Divisão de Responsabilidades

Etapa	Tarefa
1	Modelagem e definição de entrada/saída
2	Algoritmos de ciclo e ordenação
3	Interface gráfica e visualização
4	Integração, testes e relatório final

---

## 1.5. Referencial Teórico

- [PMC – Artigo sobre ciclos em dependências](#)
- [Kenah & Robins \(2007\) – SIR e redes estocásticas](#)
- [Cocomello & Ramanan \(2023\) – SIR em grafos tipo árvore](#)
- [ScienceDirect – Dependência e execução segura](#)

## Conteúdo

### Exemplos de Aplicações

- Gerenciadores de pacotes (como `apt`, `pip`, `npm`)
- Ferramentas de build (`Makefile`, `Gradle`, `CMake`)

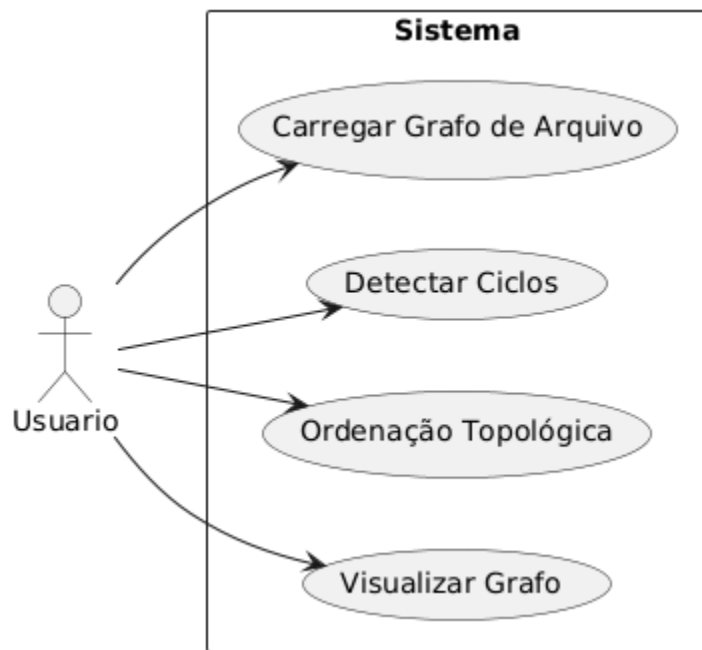
- **Tarefas em nuvem/serverless** com dependências

## Modelagem do Grafo

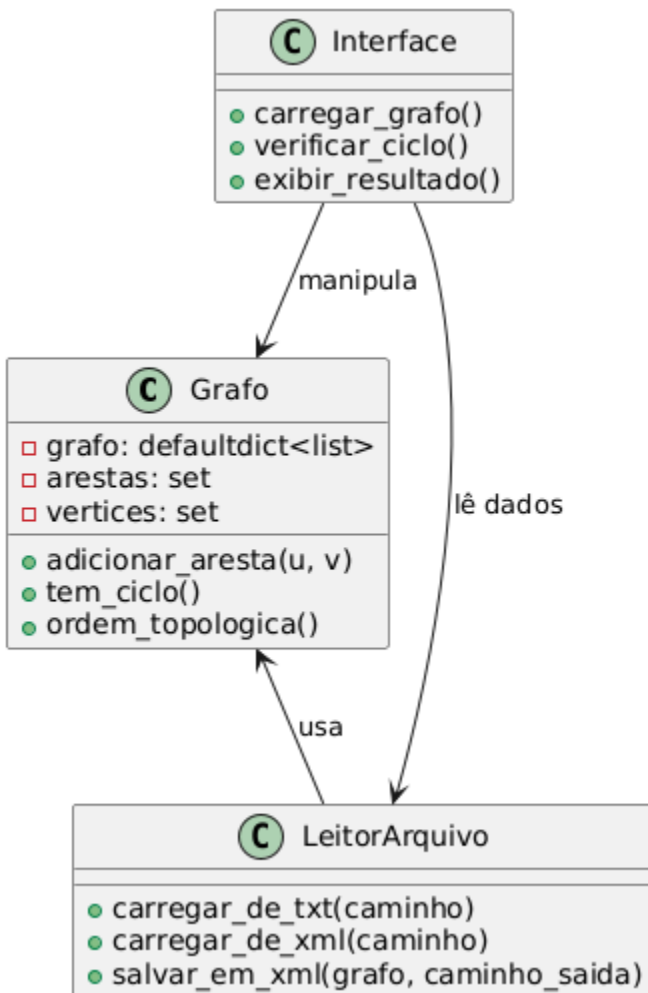
- **Nós (V):** tarefas ou pacotes
- **Arestas (E):** dependências direcionadas ( $A \rightarrow B$  indica que A depende de B)
- **Grafo dirigido acíclico (DAG)** desejado para execução segura
- **Problemas comuns:** ciclos (deadlocks), redundâncias, múltiplos caminhos

## Diagramas Uml

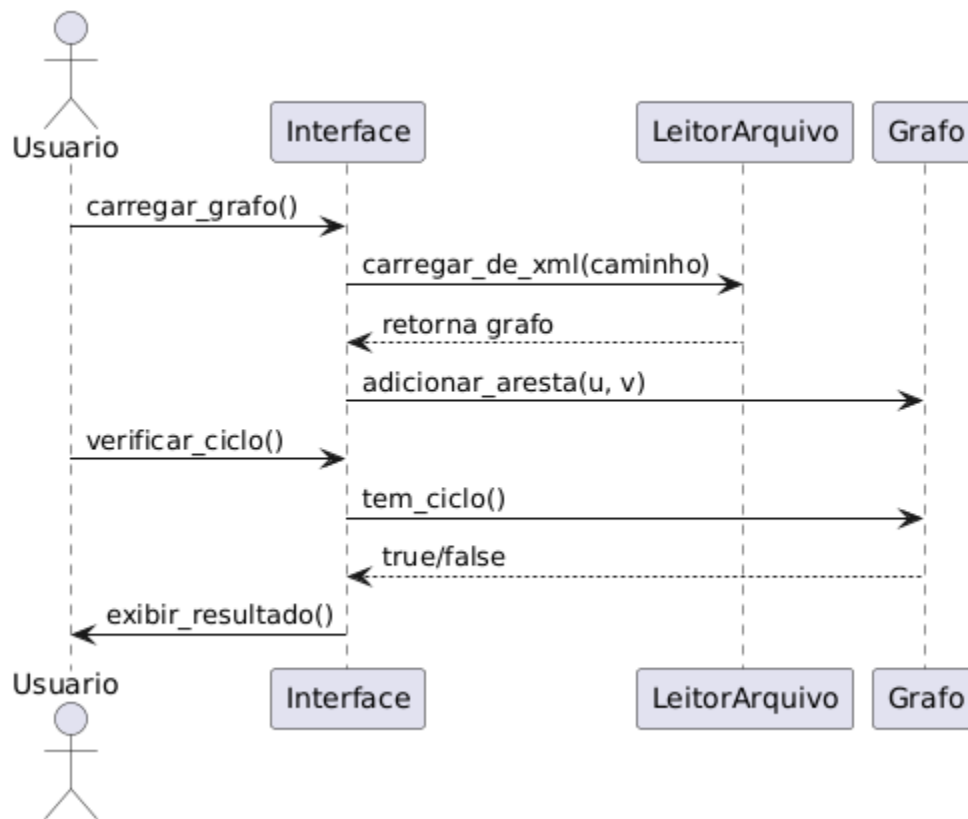
### - Casos de uso



- Diagrama de classes:



## - Diagrama de Sequência



### Funcionalidades a Serem Desenvolvidas

1. Leitura de um arquivo JSON ou CSV com dependências
2. Construção do grafo de dependência
3. Detecção automática de ciclos (algoritmo de Tarjan ou DFS)
4. Ordenação topológica segura (Kahn ou DFS pós-ordem)
5. Visualização do grafo (com destaque em ciclos, se houver)
6. Interface gráfica ou de terminal para interação

## Desenvolvimento

### Tecnologias Recomendadas

- **Python com NetworkX** (ou Java com JGraphT, ou C++ Boost Graph)
  - **Tkinter / PyQt / Streamlit** (opcional GUI)
  - **Graphviz / Matplotlib** para visualização
  - **Git** para versionamento
- 

### Algoritmos Essenciais

- **Detecção de Ciclos:**
    - Busca em profundidade (DFS) com marcação de estados
    - Algoritmo de Tarjan (para componentes fortemente conectados)
  - **Ordenação Topológica:**
    - Algoritmo de Kahn (com fila de nós com grau 0)
    - DFS inverso (pós-ordem | busca por profundidade começando dos nós folha)
- 

### Interface (Sugestão Mínima)

- Leitura de arquivos **.json**, **.csv** ou **.txt**
- Botões: “Detectar ciclos”, “Executar ordenação”, “Exibir grafo”
- Saída: lista ordenada, erros de dependência, grafo visualizado



## Previsão de como será o Relatório

### Estrutura do Relatório SBC:

#### 1. Introdução

- Problema real de conflitos em sistemas de dependência
- Justificativa do uso de grafos

#### 2. Modelagem

- Estrutura do grafo
- Algoritmos utilizados
- Diagrama UML

#### 3. Implementação

- Tecnologias utilizadas
- Interface
- Entrada e saída do sistema

#### 4. Estudo de Caso

- Exemplo com ciclo (erro)
- Exemplo com DAG válido (execução)
- Gráficos e análise de tempo

#### 5. Resultados

- Tempos de execução



- Casos de conflito
- Eficiência da ordenação

## 6. Conclusão