Prof. Hélder Pereira Borges

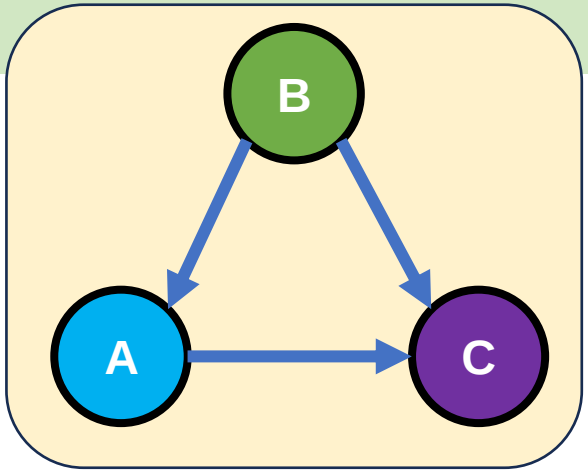
helder@ifma.edu.br

Grafos
Ordenaçao Topologica

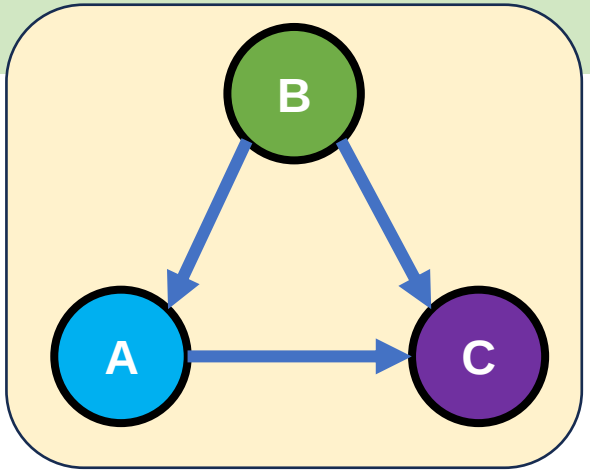
IFMA
Departamento de Computação

Introdução

- Um grafo é **acíclico** e **dirigido** (direct acyclic graph - **DAG**) quando não é possível partir de um vértice e retornar a ele seguindo uma sequência de arestas



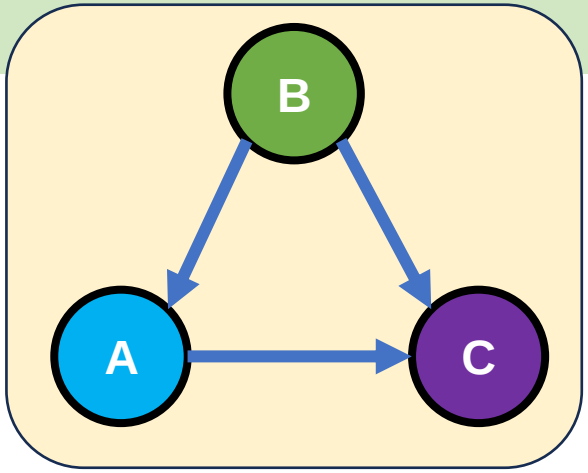
Introdução



- Um grafo é **acíclico** e **dirigido** (direct acyclic graph - **DAG**) quando não é possível partir de um vértice e retornar a ele seguindo uma sequência de arestas

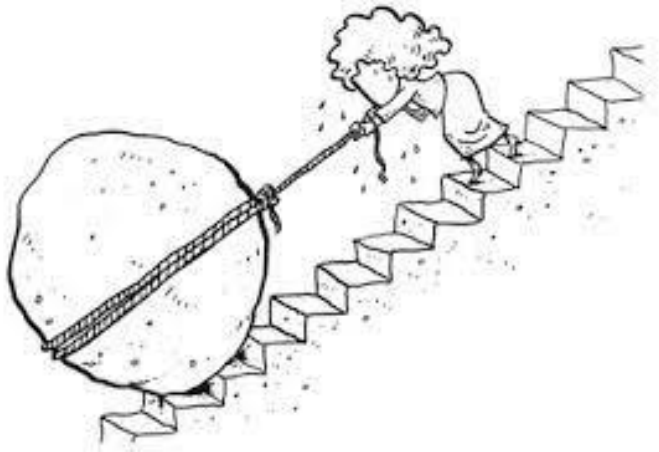
- Esse tipo de grafo é muito útil na modelagem de dependências entre **T-.A-.R-.E-.F-.A-.S**

Introdução



- Um grafo é **acíclico** e **dirigido** (direct acyclic graph - DAG) quando não é possível partir de um vértice e retornar a ele seguindo uma sequência de arestas

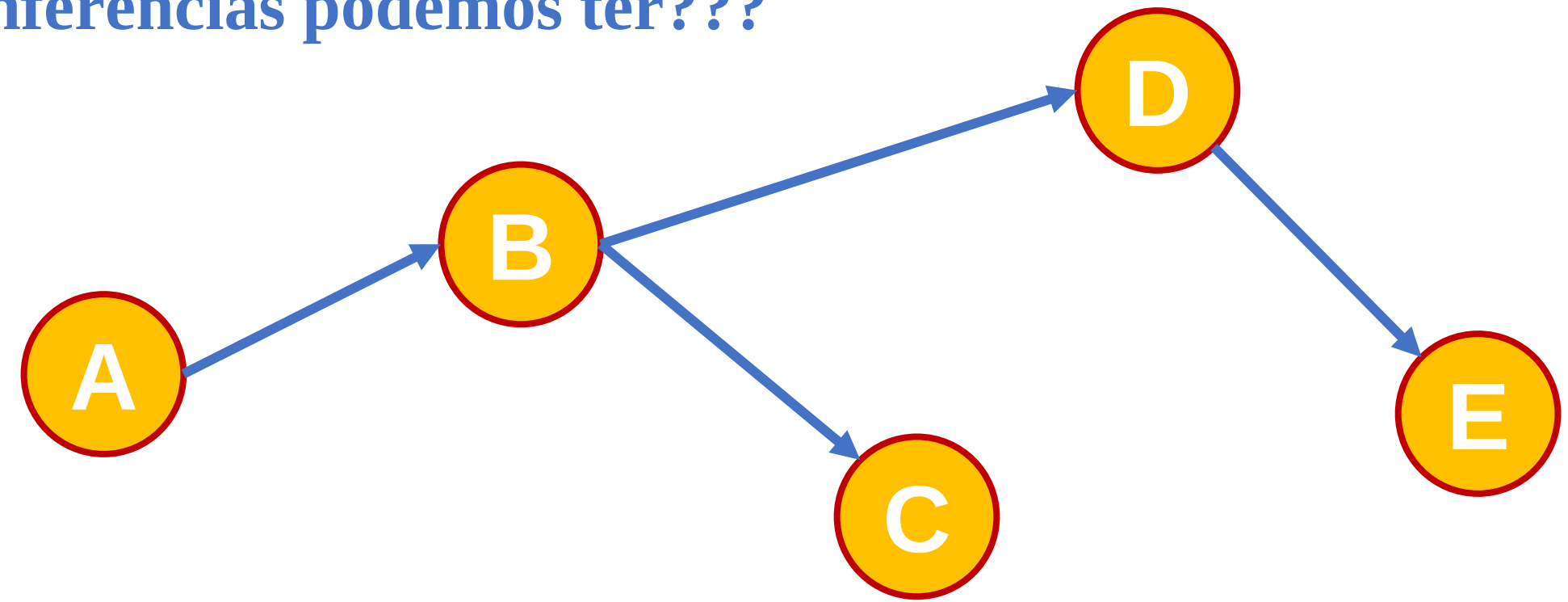
- Esse tipo de grafo é muito útil na modelagem de dependências entre **T-A-R-E-F-A-S**



- Dado um conjunto de tarefas, algumas dependentes entre si, em que ordem devem ser executadas?
  - Fundamentais no planejamento de projetos

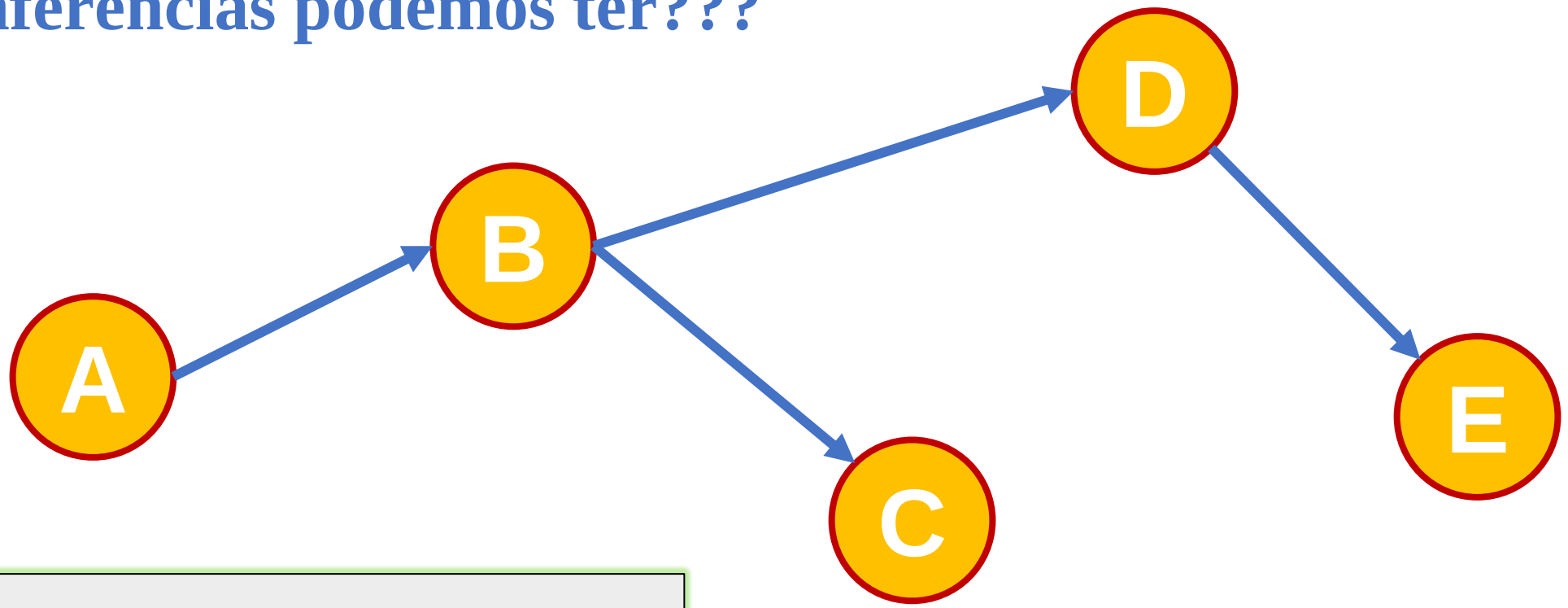
Introdução

- **Os vértices representam tarefas, quais as inferências podemos ter???**



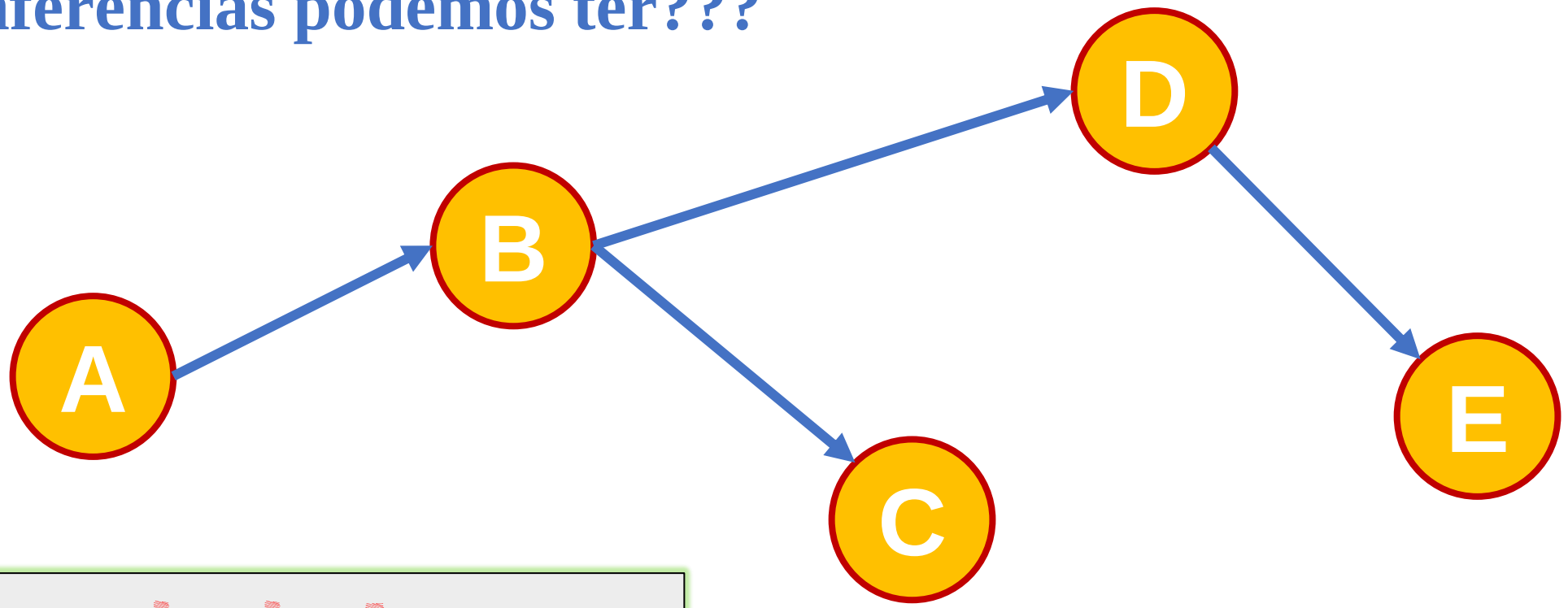
Introdução

- **Os vértices representam tarefas, quais as inferências podemos ter???**



Introdução

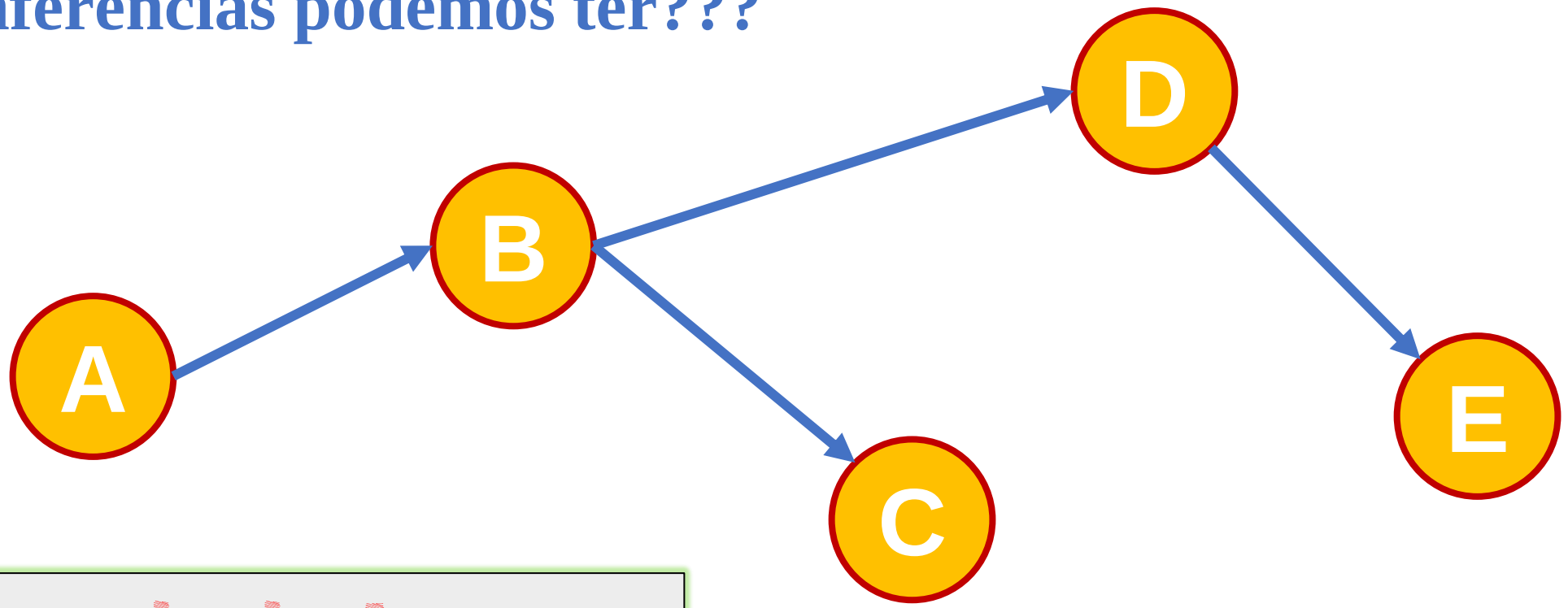
- **Os vértices representam tarefas, quais as inferências podemos ter???**



✓ B depende de A

Introdução

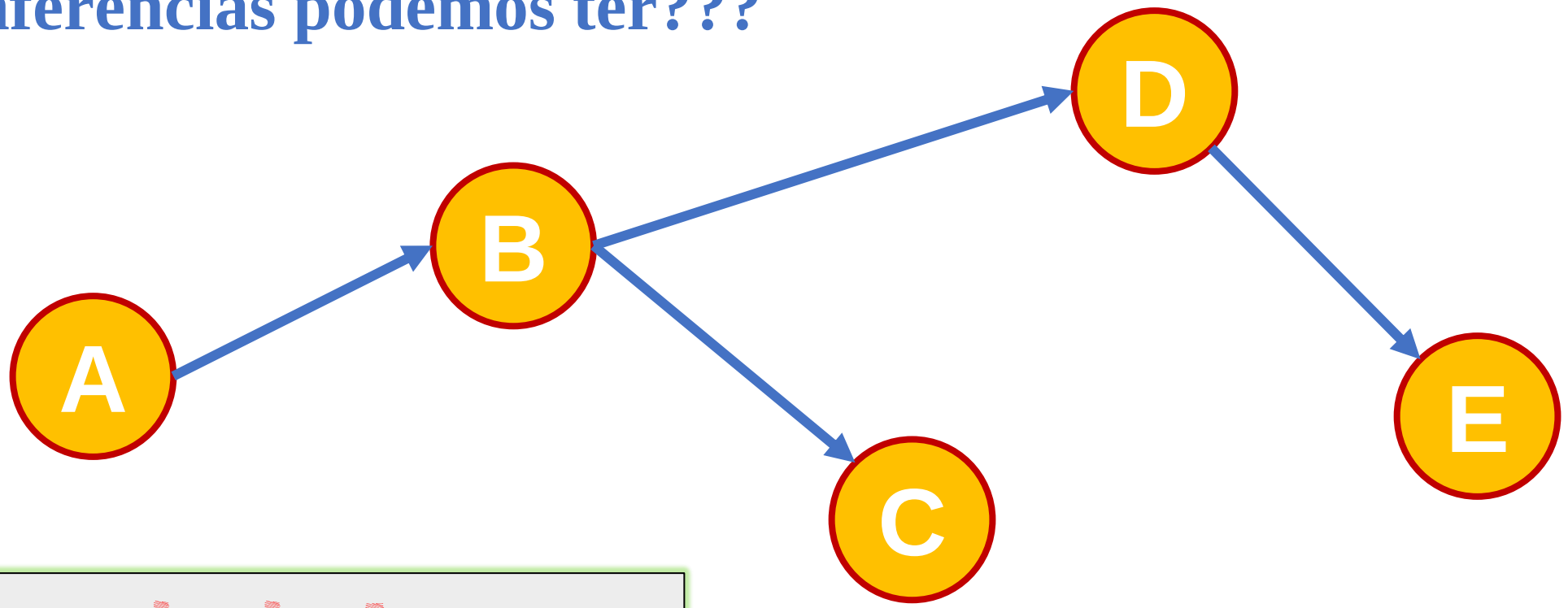
- **Os vértices representam tarefas, quais as inferências podemos ter???**



- ✓ B depende de A
- ✓ C e D dependem de B

Introdução

- **Os vértices representam tarefas, quais as inferências podemos ter???**



- ✓ B depende de A
- ✓ C e D dependem de B
- ✓ E depende de D

Ordenação Topológica (OT)

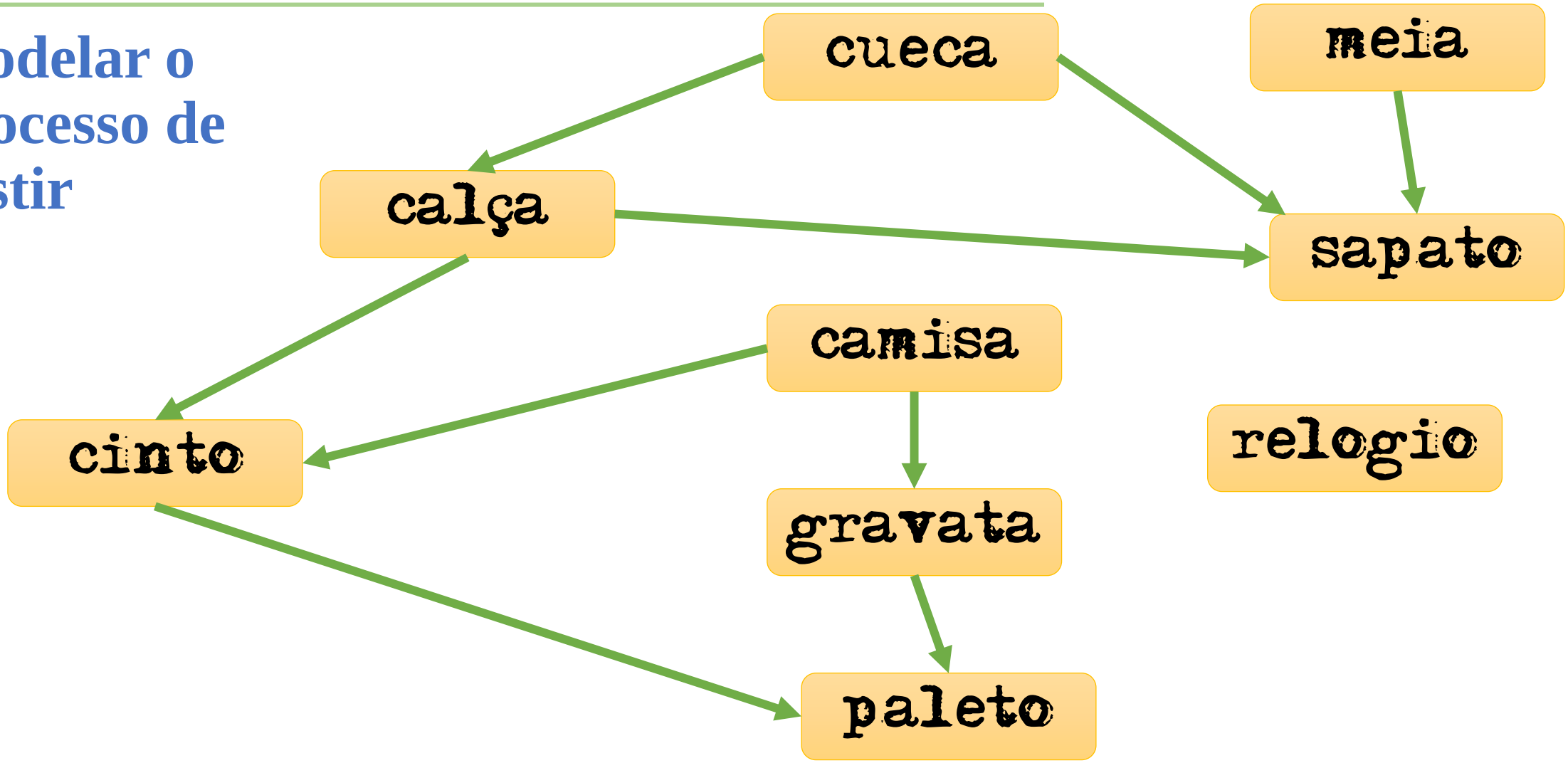
- Dado um grafo, **G = (V, A)**, **dirigido** e **acíclico**, uma OT apresenta uma classificação **linear** dos vértices do grafo, de tal forma que, para cada aresta **(*u*, *v*)**, obrigatoriamente, ***u*** aparece antes de ***v*** [CORMEN et al., 2012]
  - Representa uma ordem para execução do processo modelado
- Uma OT também pode ser considerada uma **permutação dos vértices** do grafo, respeitando as relações de dependências impostas pelas regras de negócio do problema
- A ordem linear produzida por uma OT não é necessariamente **única**, podem existir várias atendendo às restrições do conceito

Ordenação Topológica

- Modelar o processo de vestir

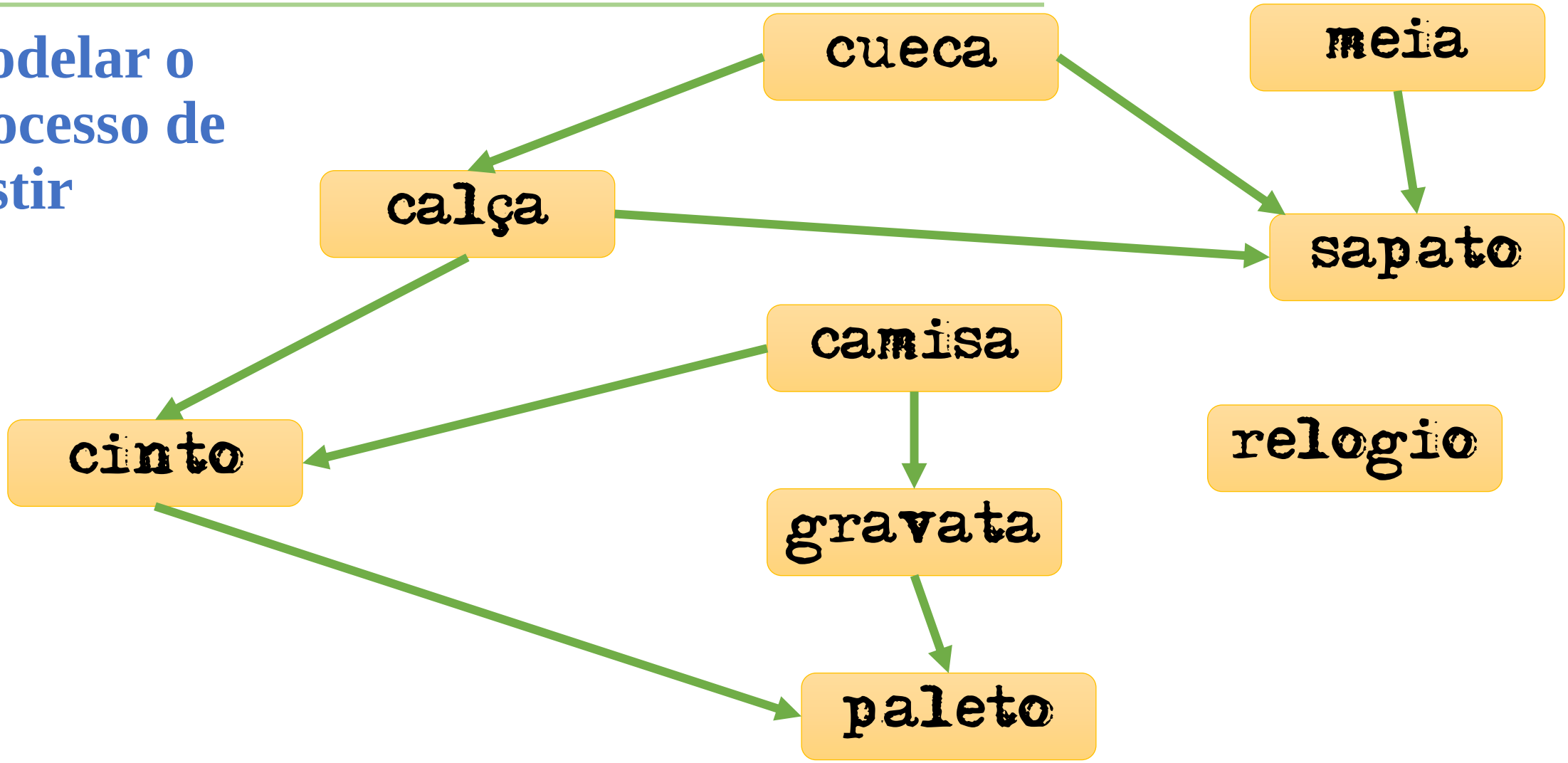
Ordenação Topológica

- Modelar o processo de vestir



Ordenação Topológica

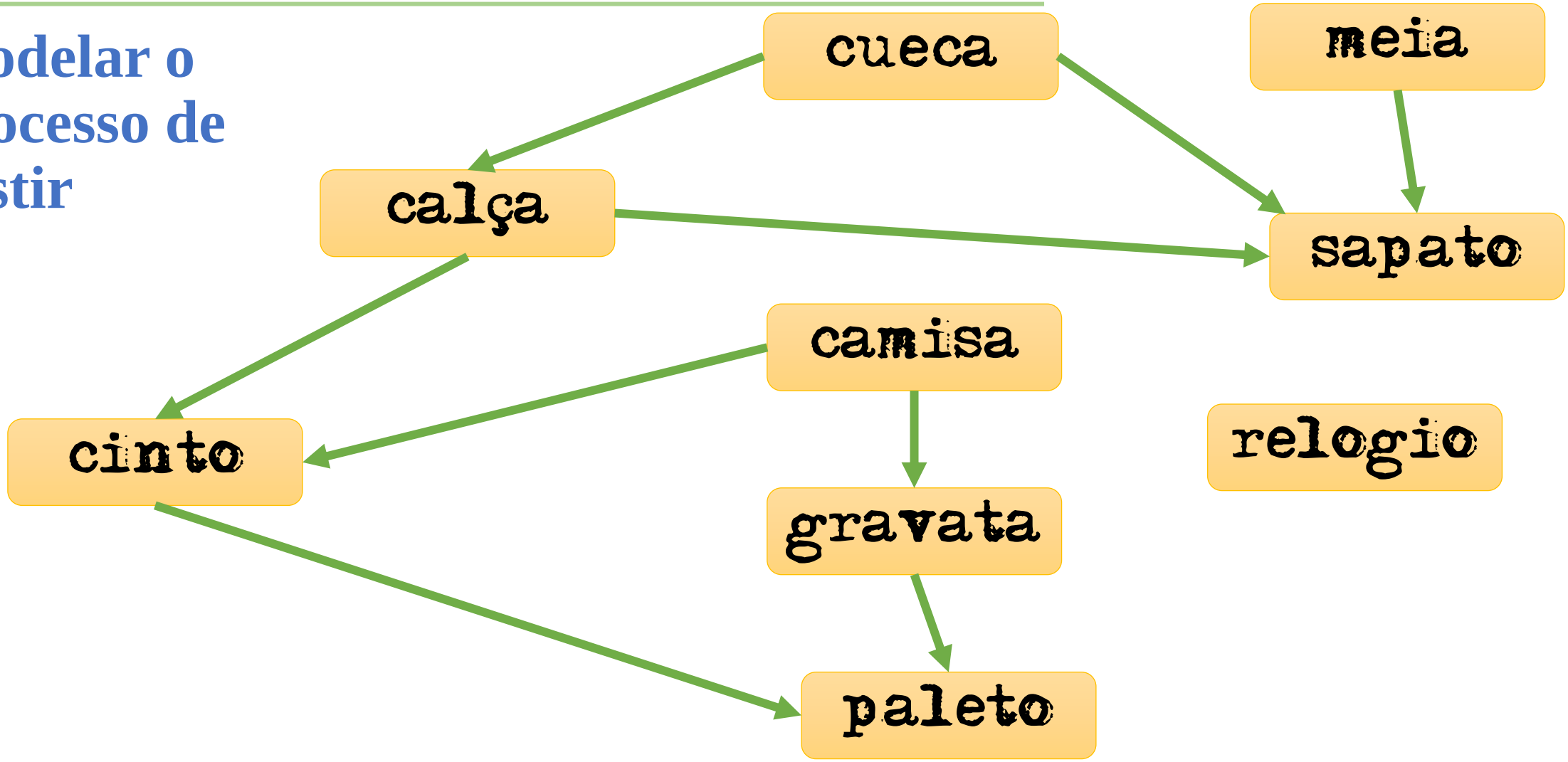
- Modelar o processo de vestir



meia

Ordenação Topológica

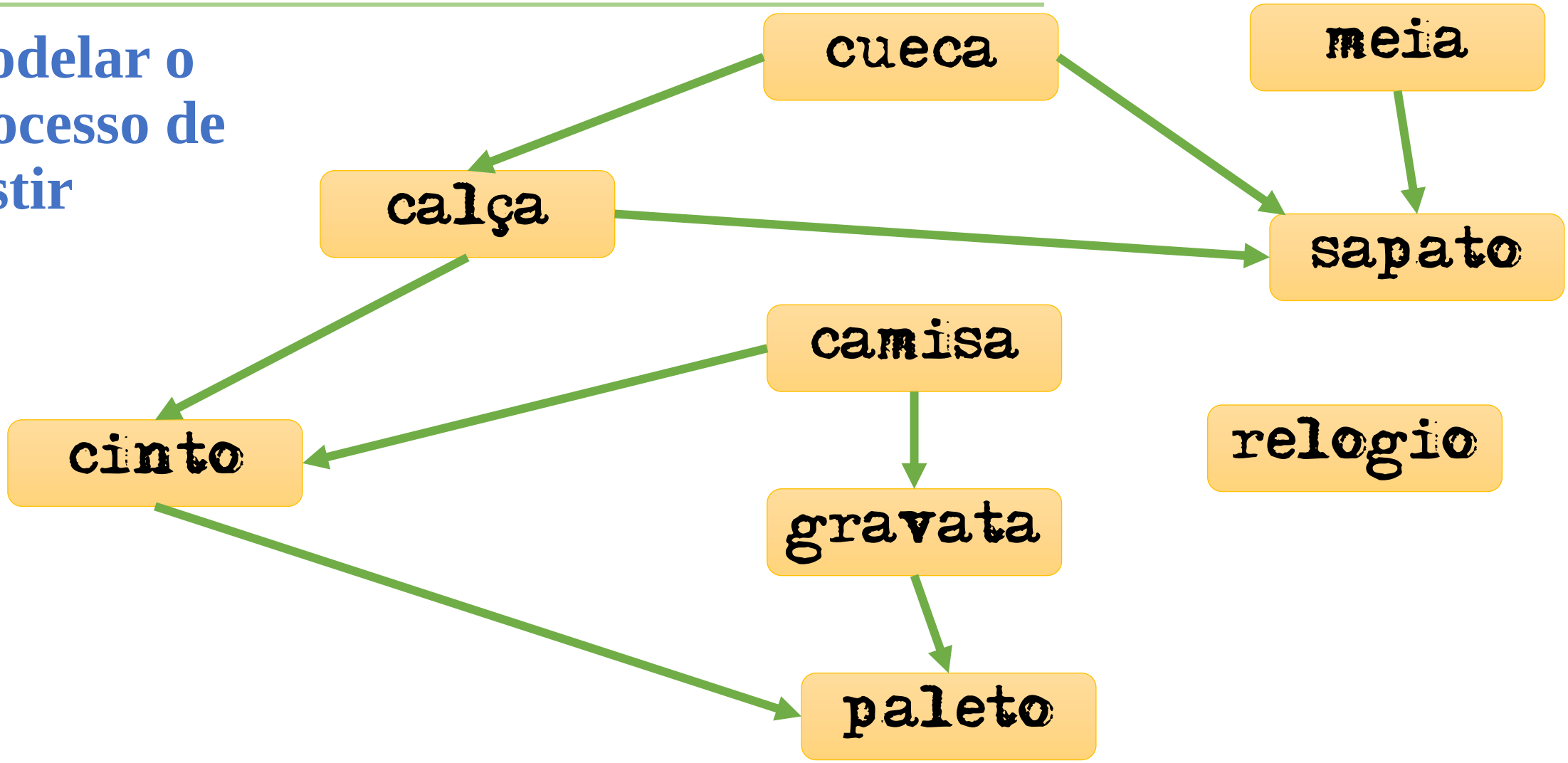
- Modelar o processo de vestir



meia cueca

Ordenação Topológica

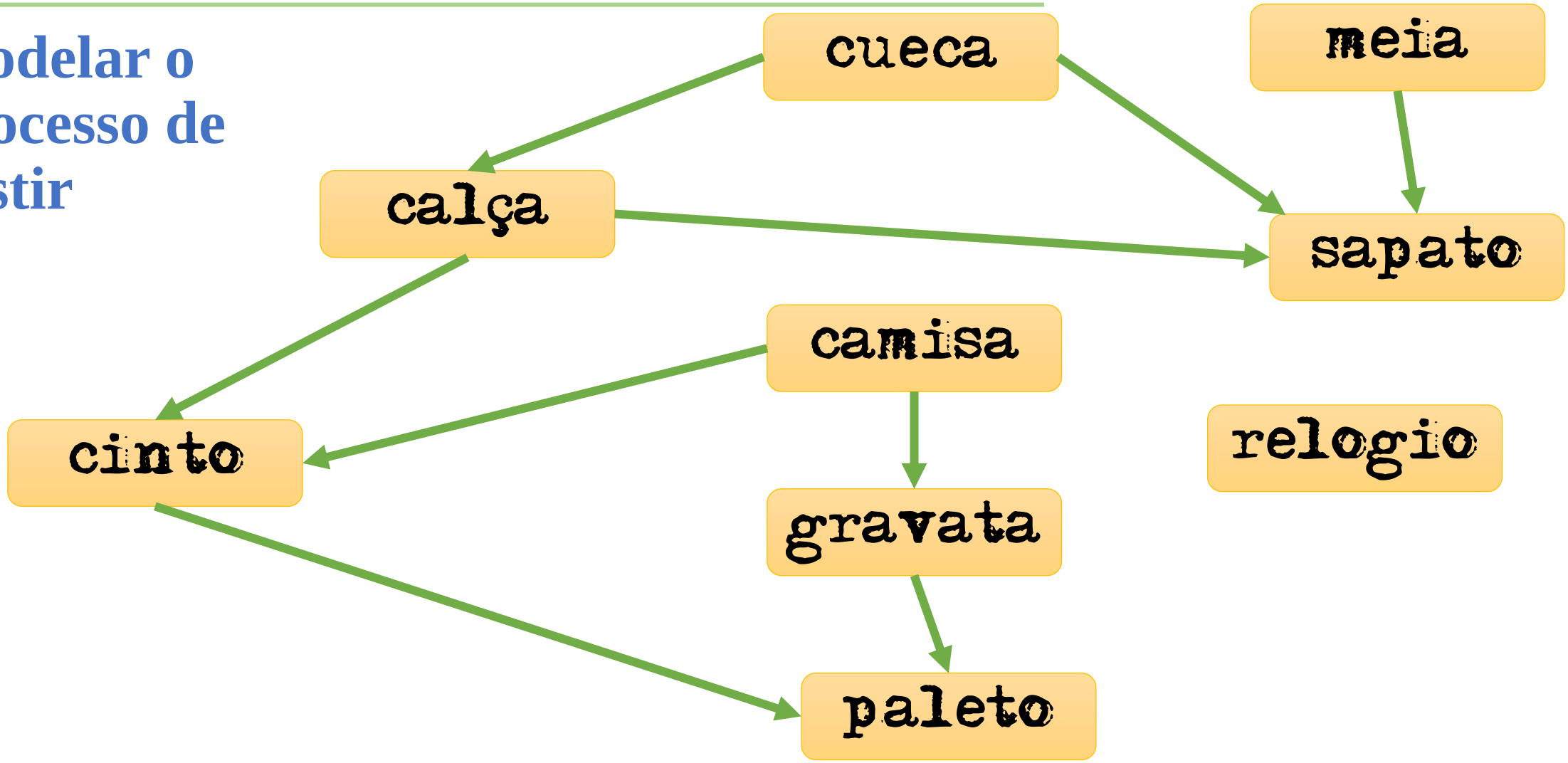
- Modelar o processo de vestir



meia | cueca | calça

Ordenação Topológica

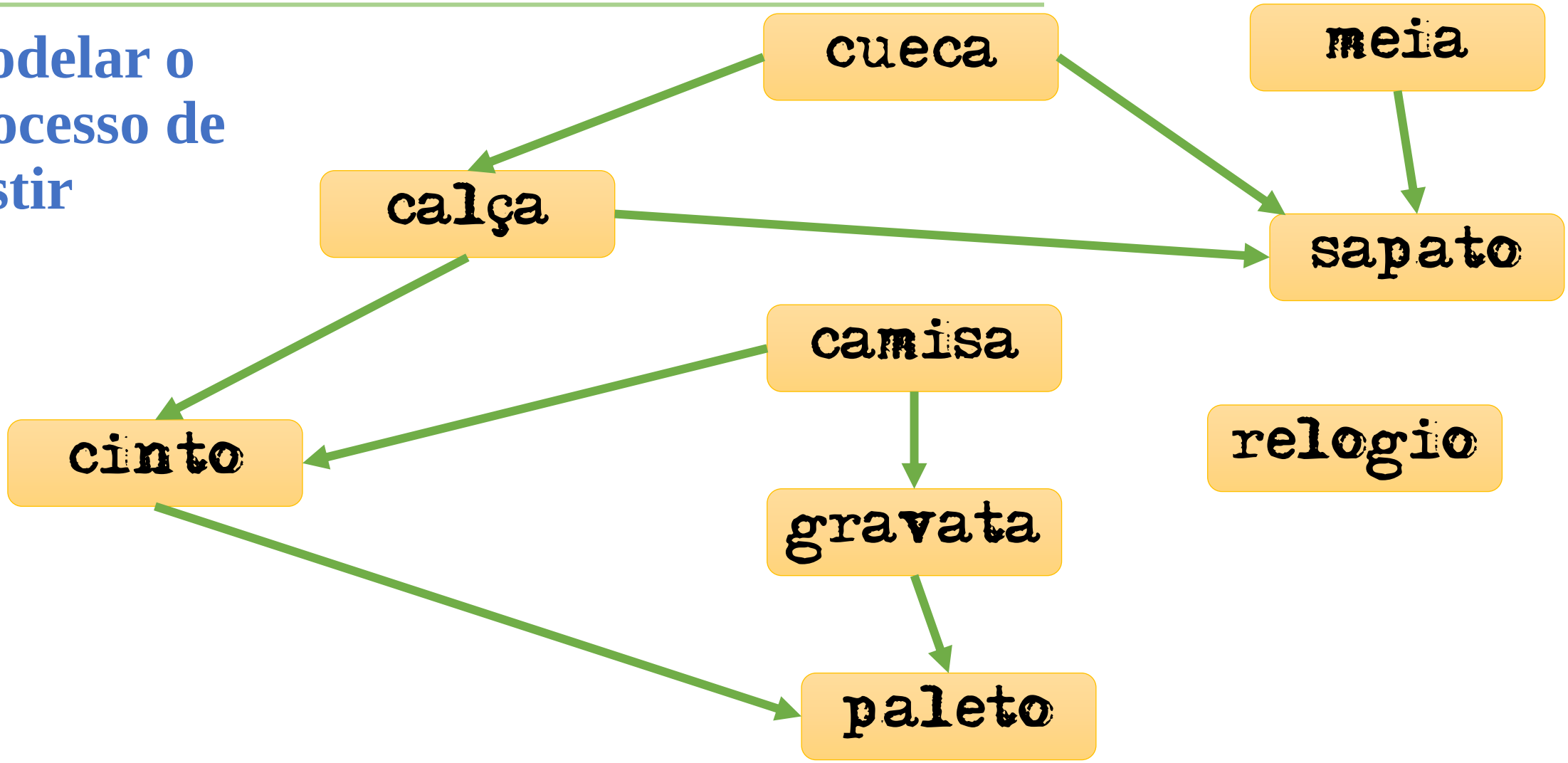
- Modelar o processo de vestir



meia | cueca | calça | sapato

Ordenação Topológica

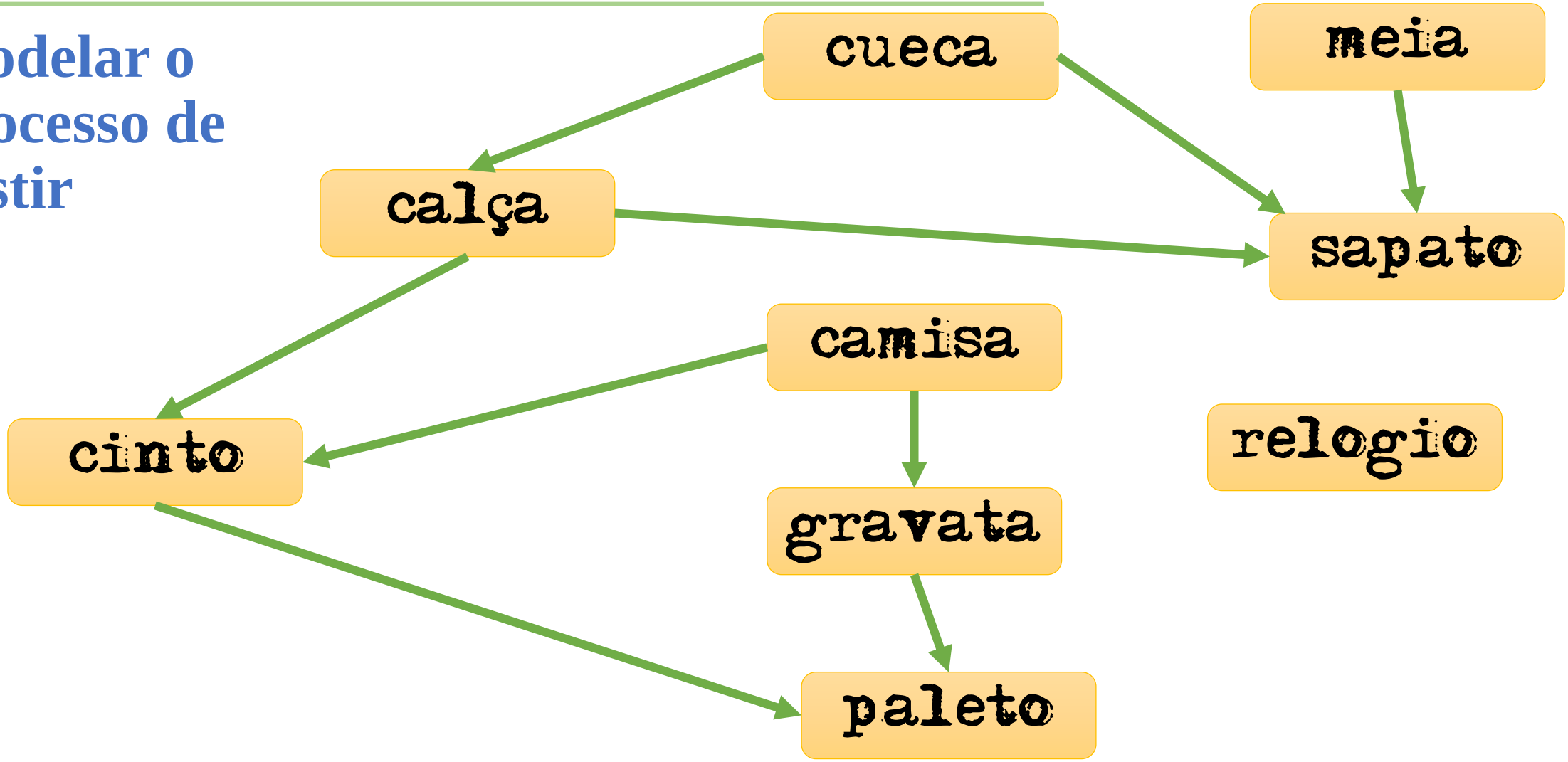
- Modelar o processo de vestir



meia | cueca | calça | sapato | relogio

Ordenação Topológica

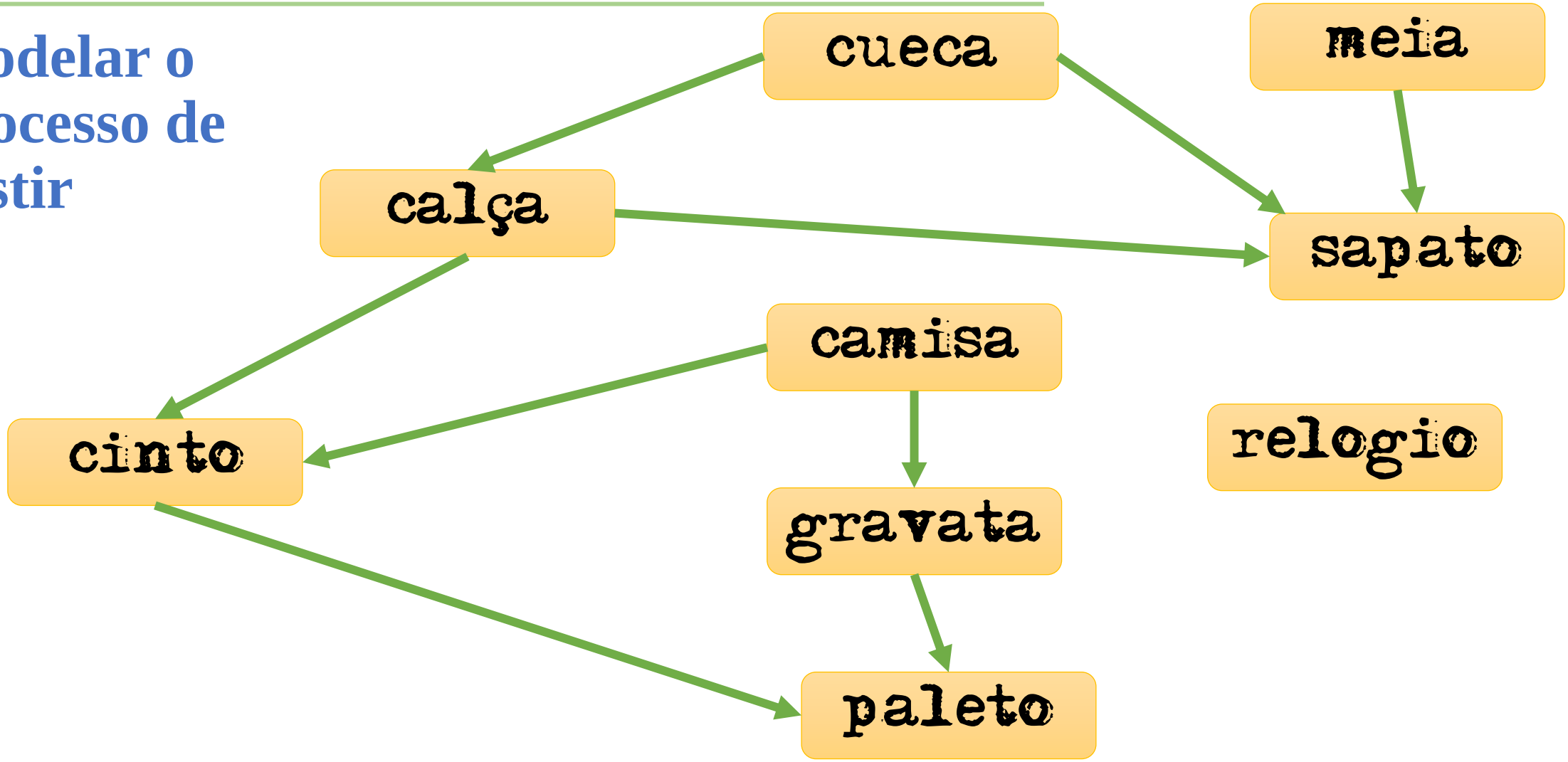
- Modelar o processo de vestir



meia | cueca | calça | sapato | relogio | camisa

Ordenação Topológica

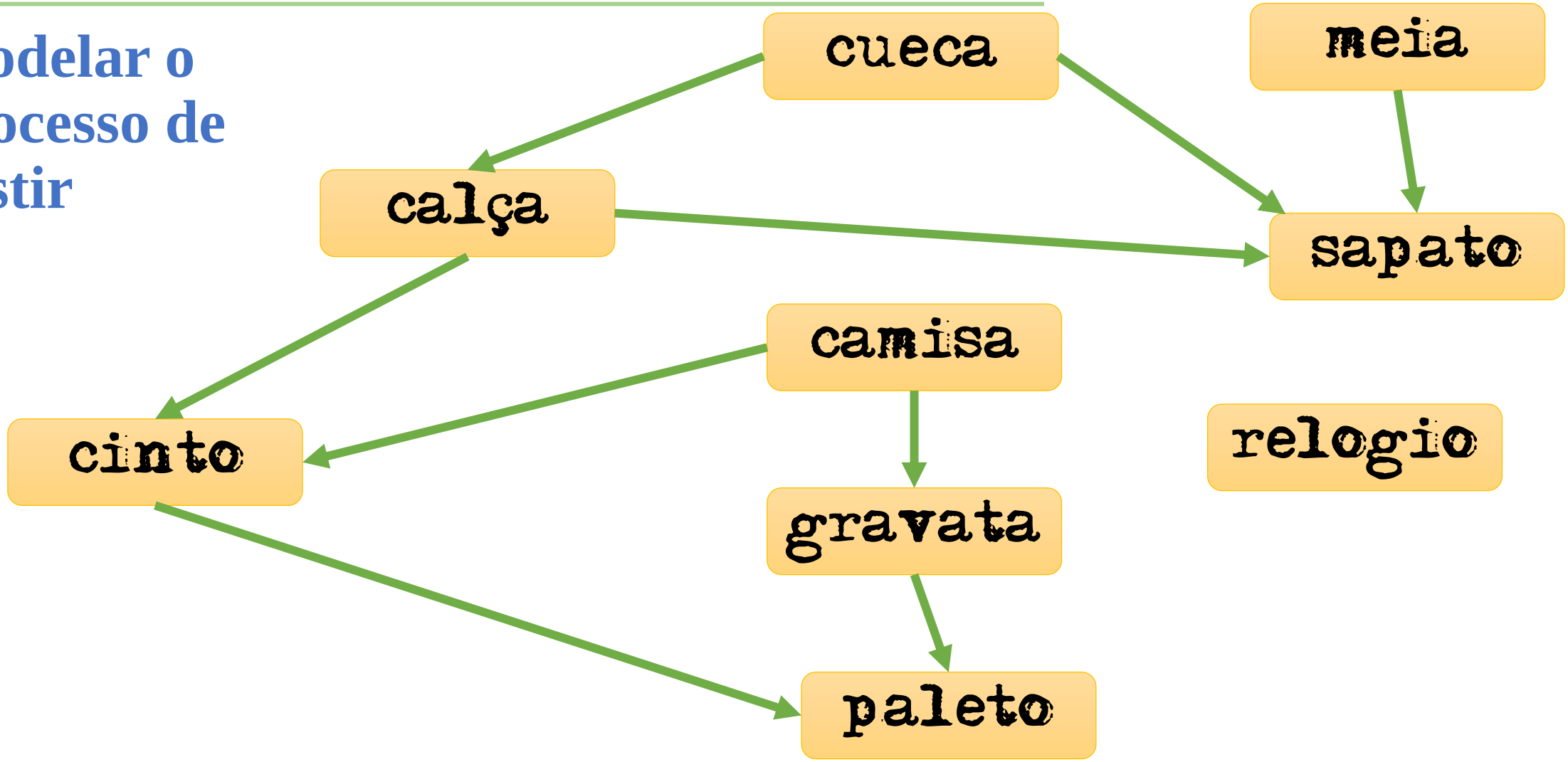
- Modelar o processo de vestir



meia | cueca | calça | sapato | relogio | camisa | cinto

Ordenação Topológica

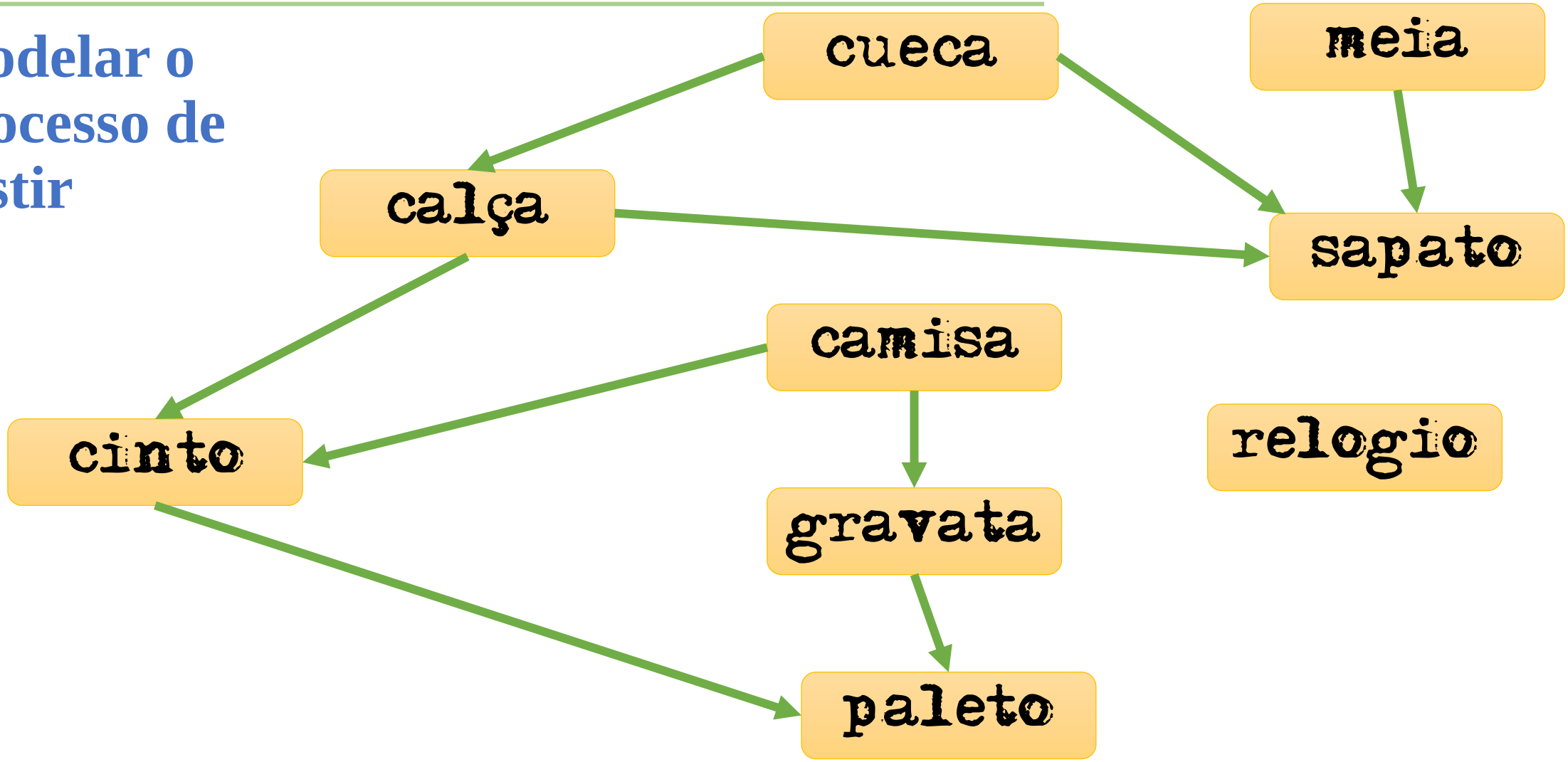
- Modelar o processo de vestir



meia | cueca | calça | sapato | relogio | camisa | cinto | gravata

Ordenação Topológica

- Modelar o processo de vestir



meia | cueca | calça | sapato | relogio | camisa | cinto | gravata | paleto

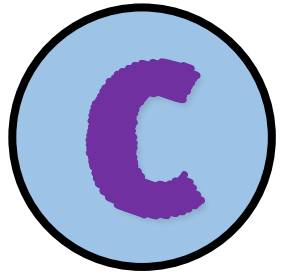
Ordenação Topológica

Ordenação Topológica

GRAFOS
COM CICLOS
NAO
ADMITEM
UMA OT

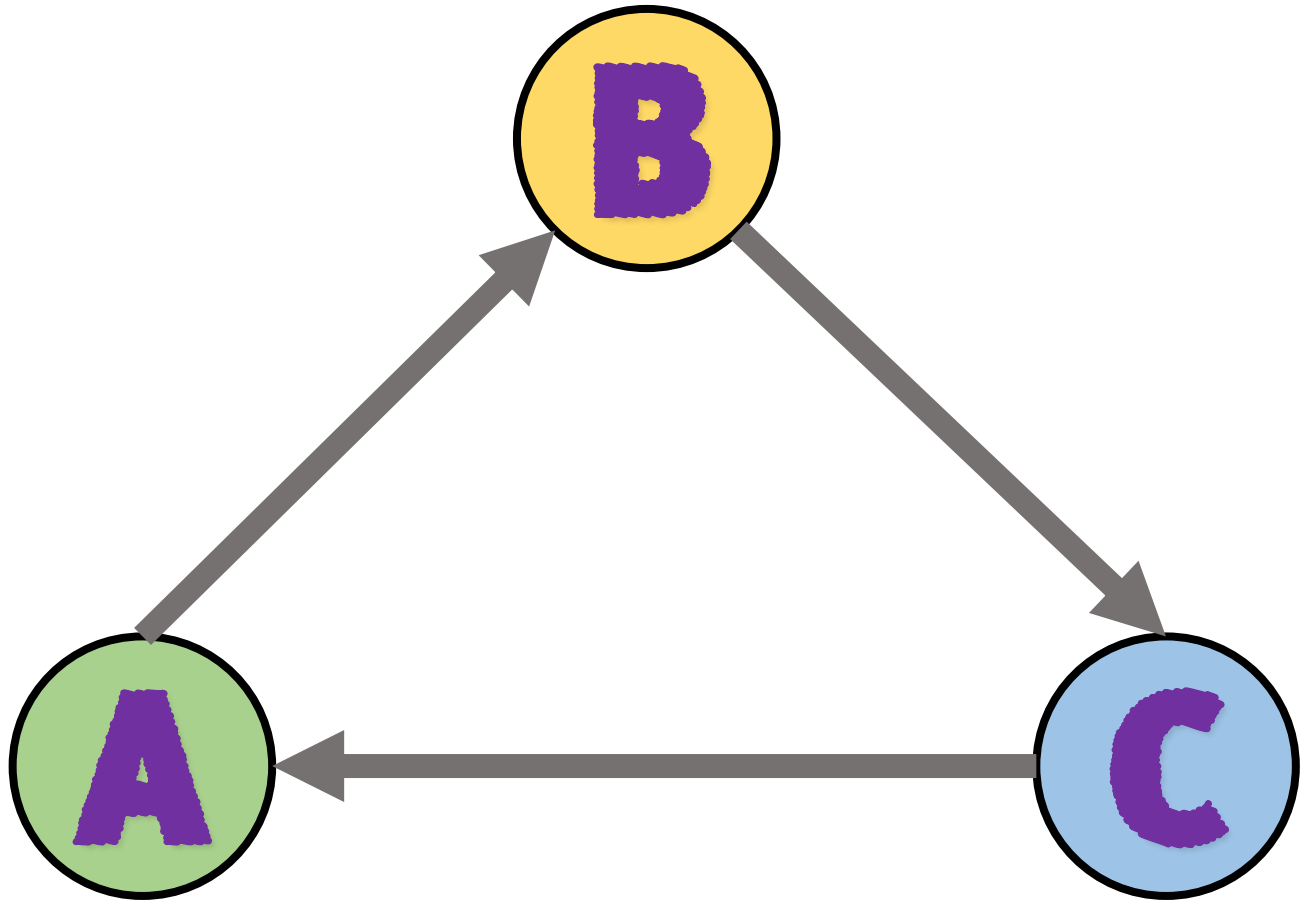
Ordenação Topológica

GRAFOS COM CICLOS NAO ADMITEM UMA OT

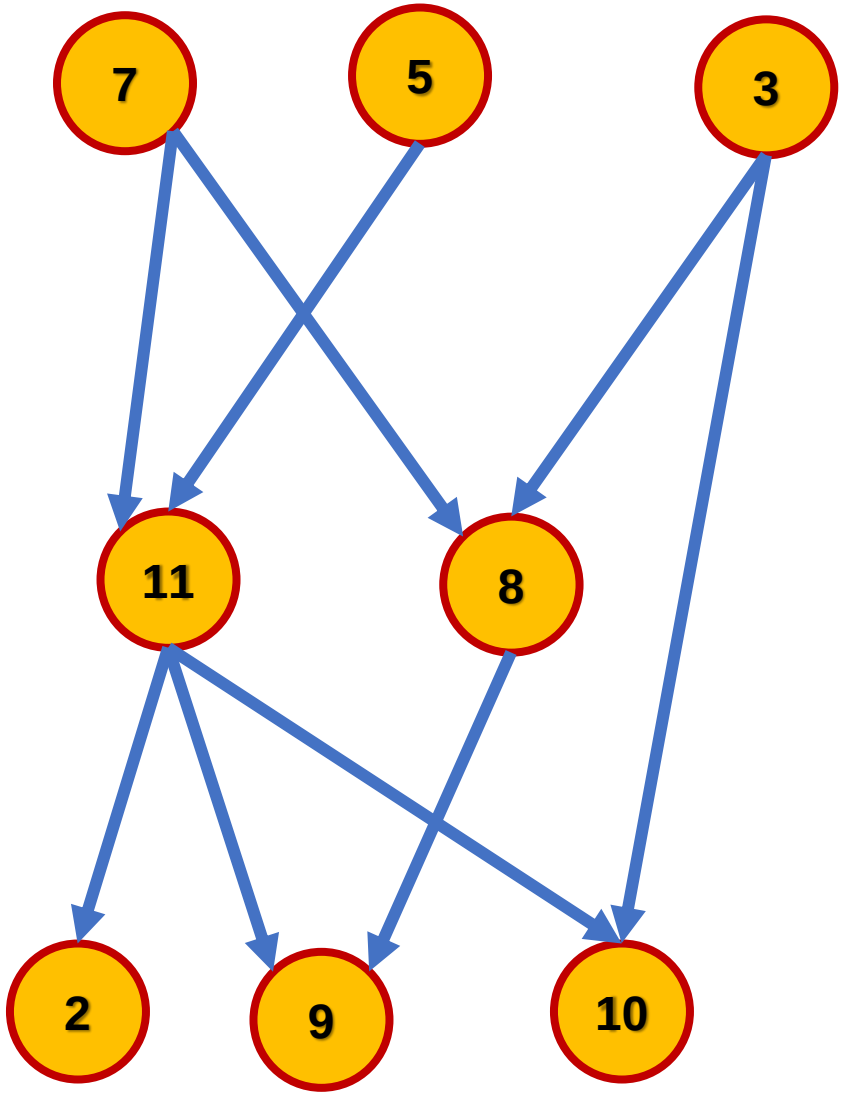


Ordenação Topológica

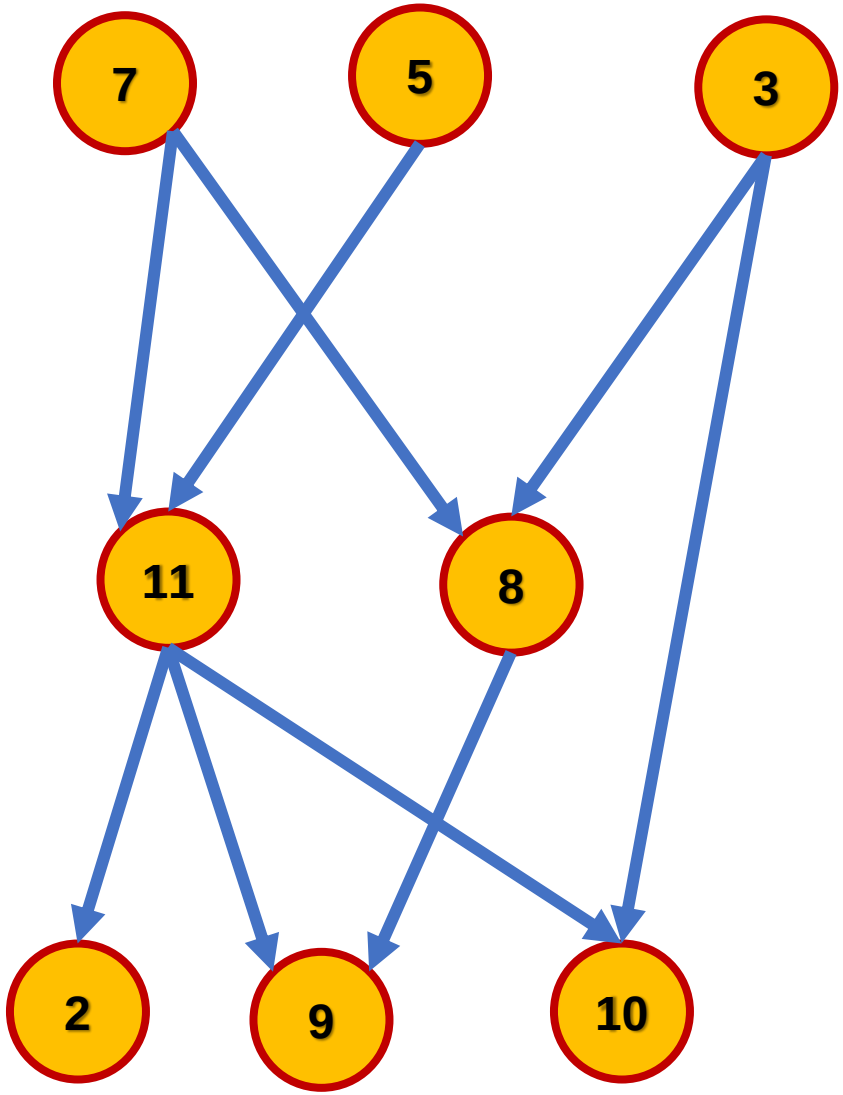
GRAFOS COM CICLOS NAO ADMITEM UMA OT



Ordenações topológicas possíveis

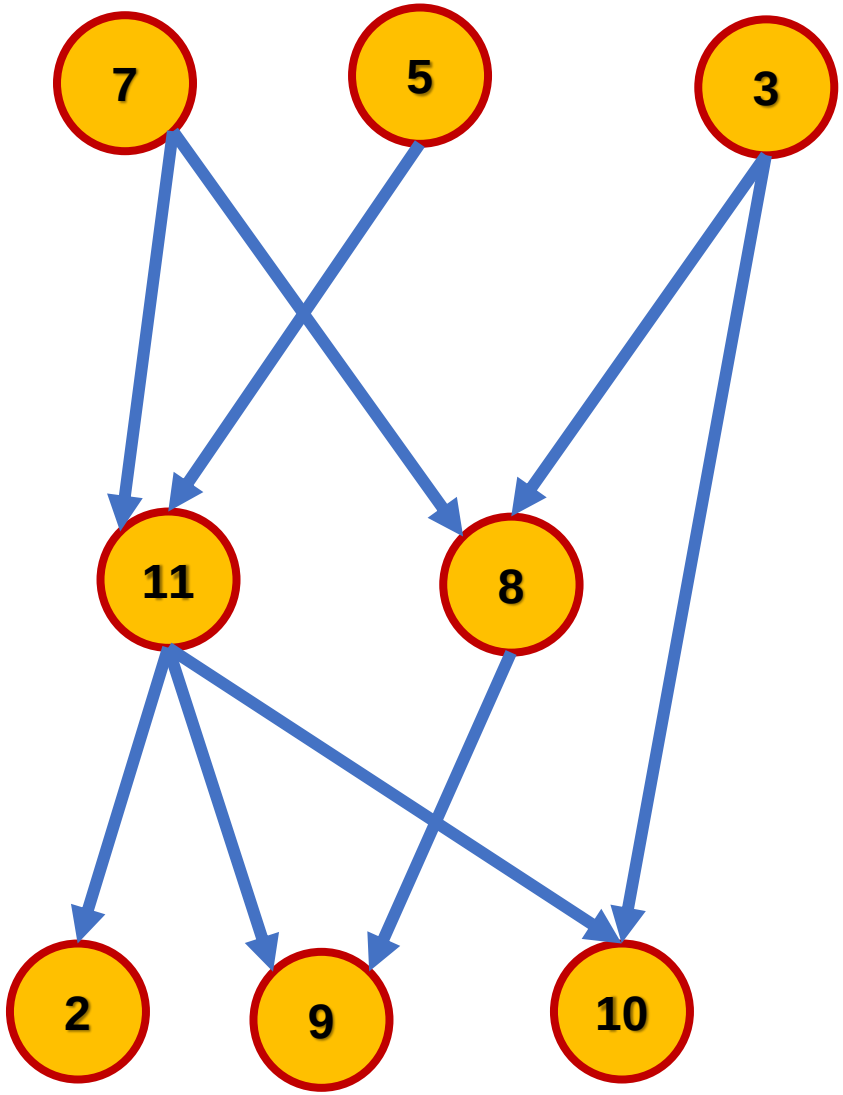


Ordenações topológicas possíveis



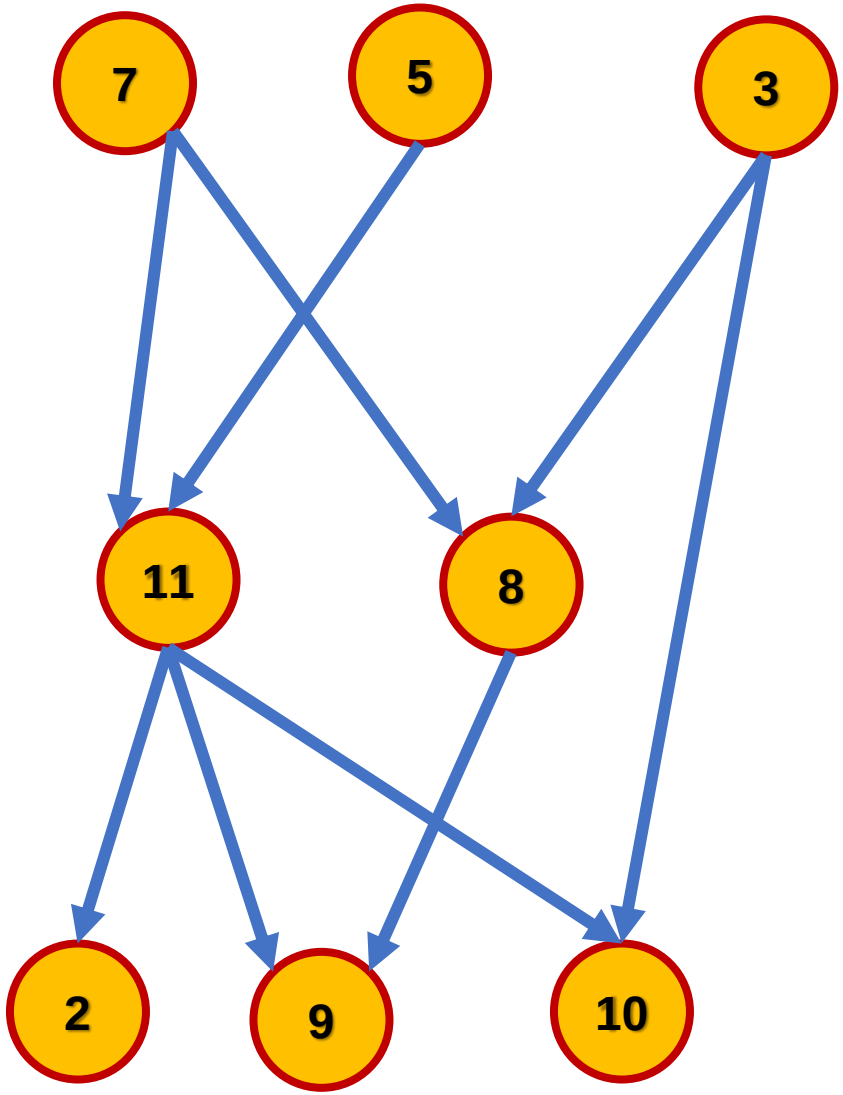
- 7, 5, 3, 11, 8, 2, 9, 10

Ordenações topológicas possíveis



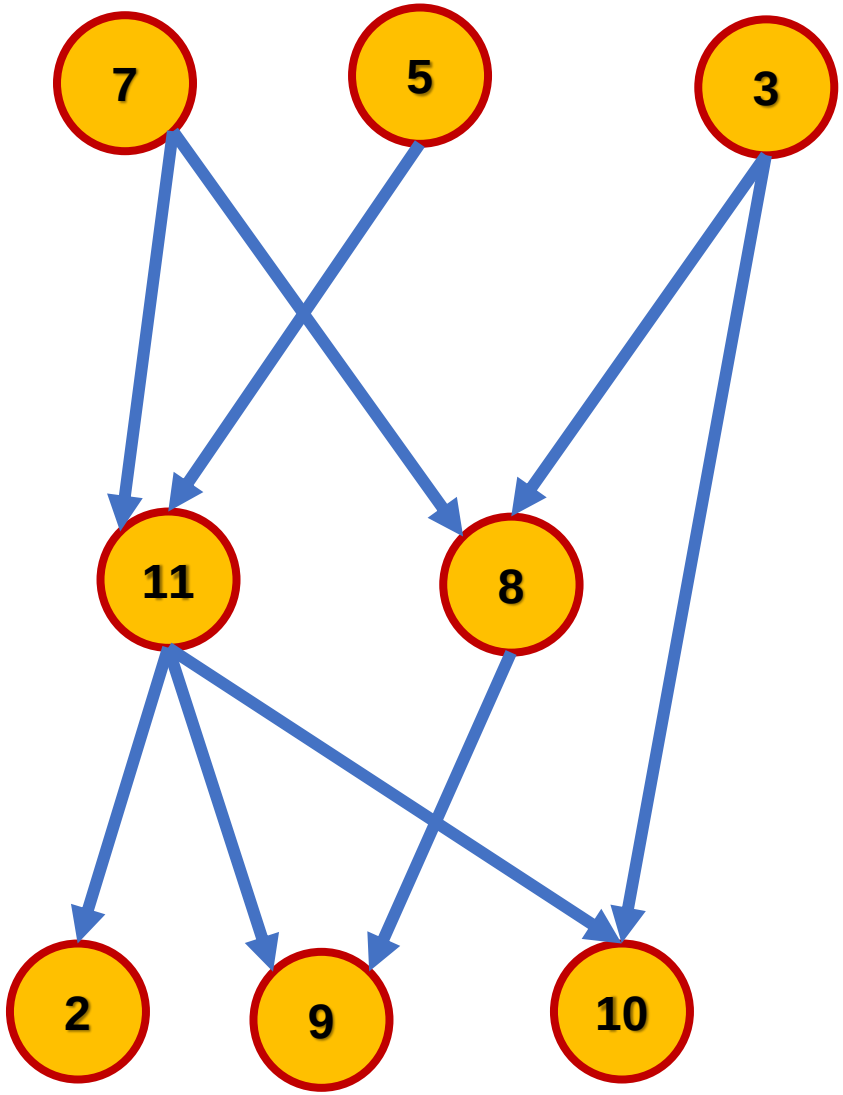
- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo

Ordenações topológicas possíveis



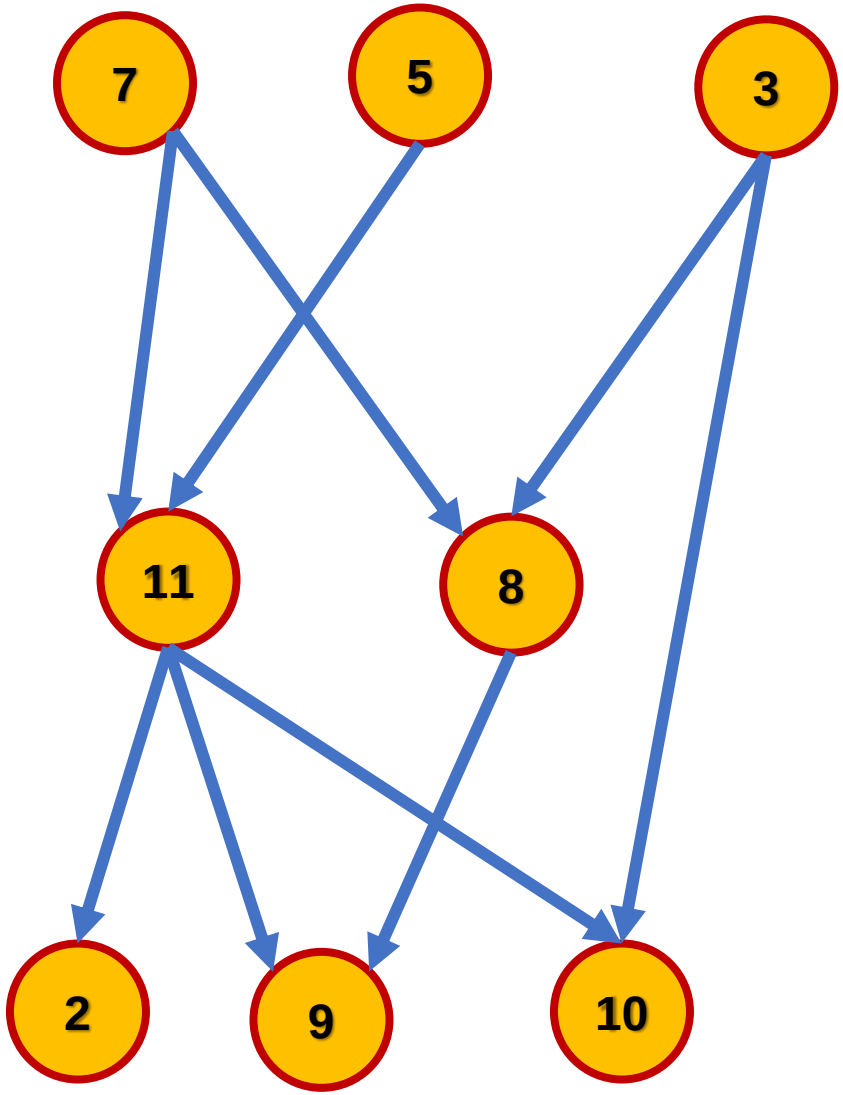
- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10

Ordenações topológicas possíveis



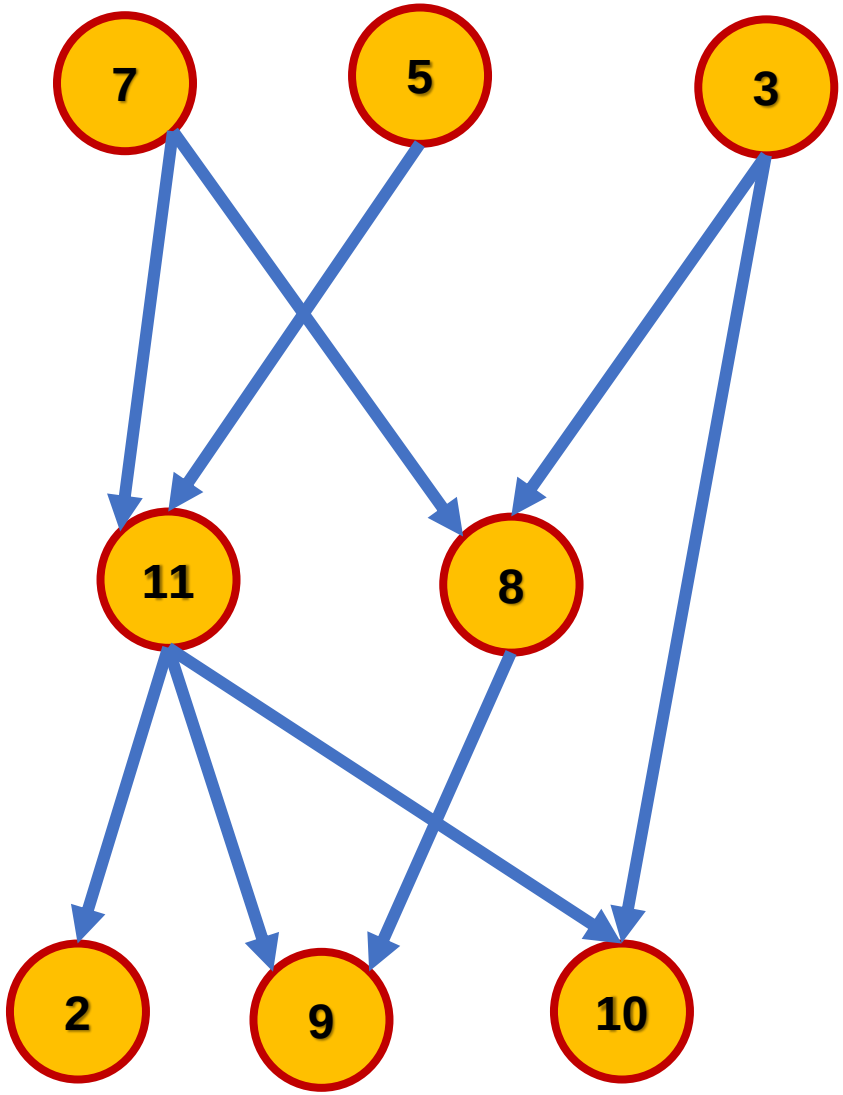
- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10
  - vértice menor primeiro

Ordenações topológicas possíveis



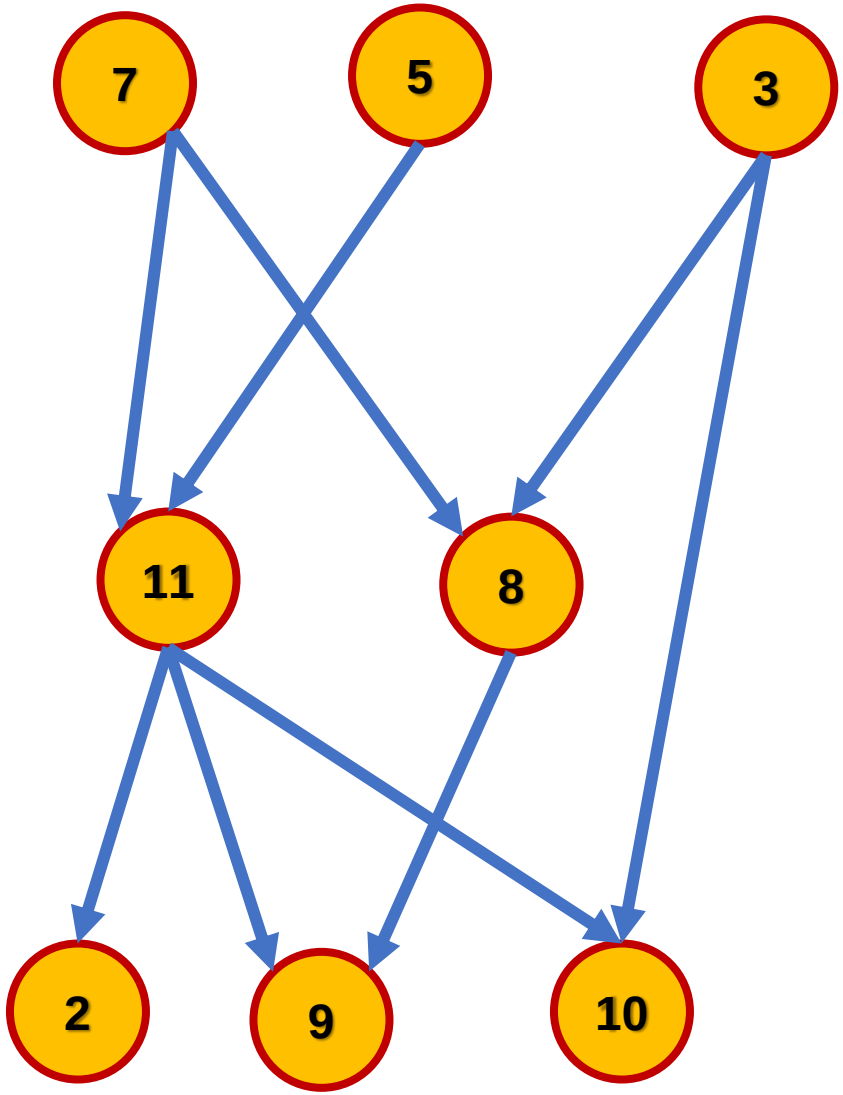
- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10
  - vértice menor primeiro
- 5, 7, 3, 8, 11, 10, 9, 2

Ordenações topológicas possíveis



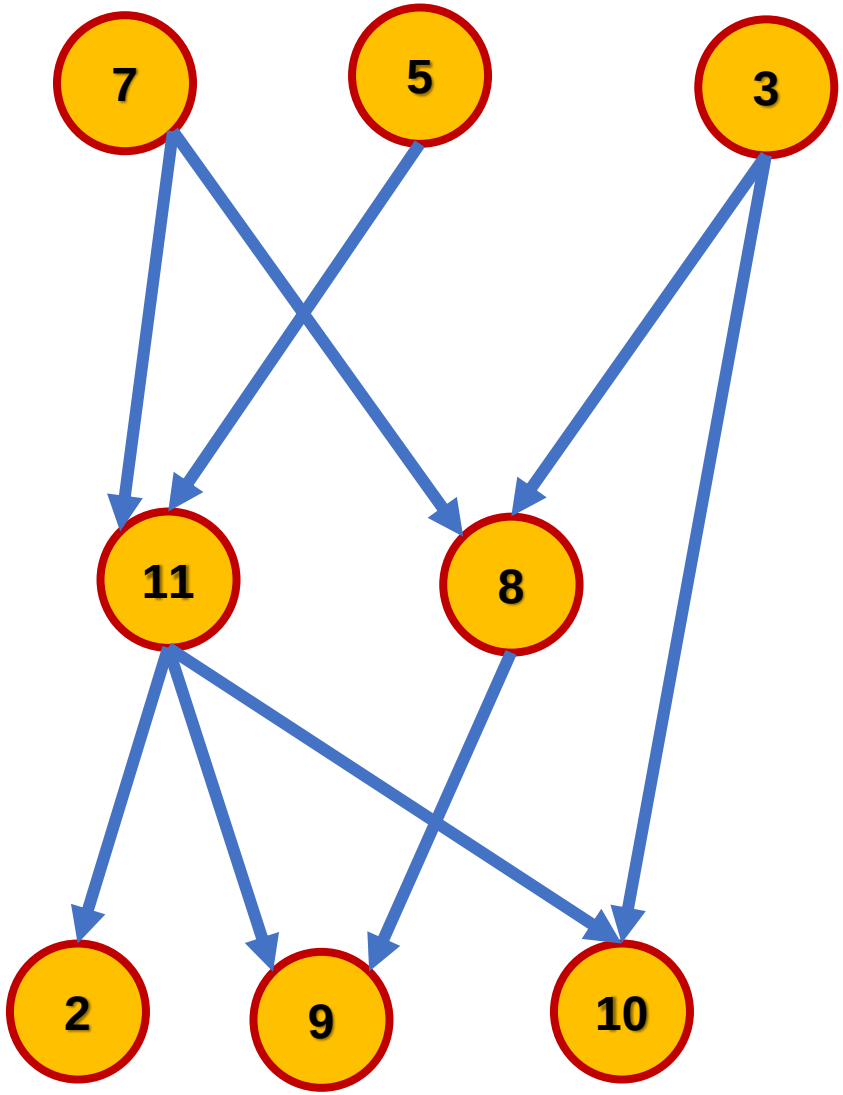
- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10
  - vértice menor primeiro
- 5, 7, 3, 8, 11, 10, 9, 2
  - menor número de arestas primeiro

Ordenações topológicas possíveis



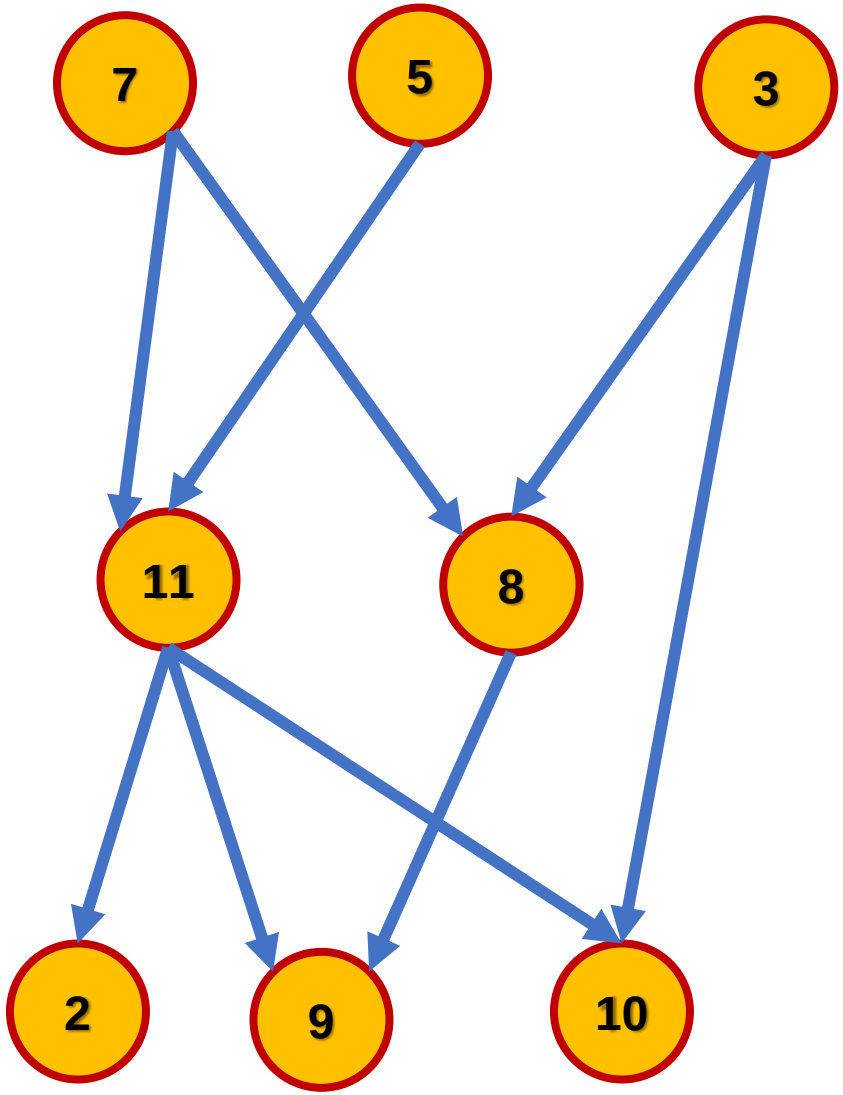
- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10
  - vértice menor primeiro
- 5, 7, 3, 8, 11, 10, 9, 2
  - menor número de arestas primeiro
- 7, 5, 11, 3, 10, 8, 9, 2

Ordenações topológicas possíveis



- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10
  - vértice menor primeiro
- 5, 7, 3, 8, 11, 10, 9, 2
  - menor número de arestas primeiro
- 7, 5, 11, 3, 10, 8, 9, 2
  - vértice maior primeiro

Ordenações topológicas possíveis



- 7, 5, 3, 11, 8, 2, 9, 10
  - esquerda-direita, de cima para baixo
- 3, 5, 7, 8, 11, 2, 9, 10
  - vértice menor primeiro
- 5, 7, 3, 8, 11, 10, 9, 2
  - menor número de arestas primeiro
- 7, 5, 11, 3, 10, 8, 9, 2
  - vértice maior primeiro
- 7, 5, 11, 2, 3, 8, 9, 10

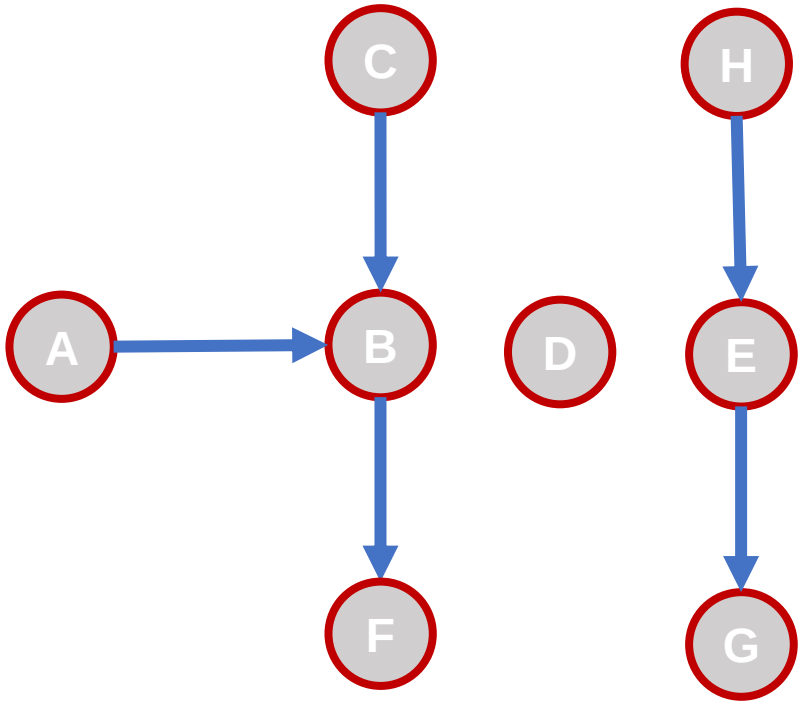
Algoritmo de Kahn

- Apresentado em 1962, é baseado na estratégia de eliminação de vértices com **grau de entrada** igual a **ZERO**, também denominados de **vértices fontes**
  - Considera que, se os **vértices fontes** [e suas arestas] forem **removidos**, o grafo remanescente é também um DAG

```
01  Calcular os graus de entrada dos vértices
02  Enquanto houverem vértices fontes
03      Escolher um
04          Remover as arestas de saída
05          Atualizar tabela de graus
```

ao fim da execução, se o tamanho da ordem topológica for diferente da quantidade de vértices, existem ciclos

Algoritmo de Kahn

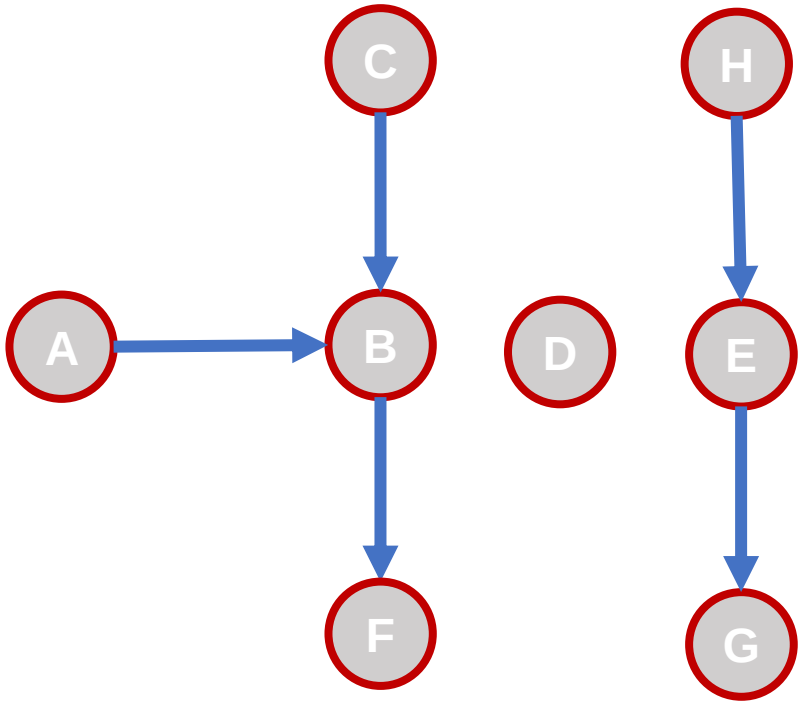


01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus

Ordem Topológica

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus

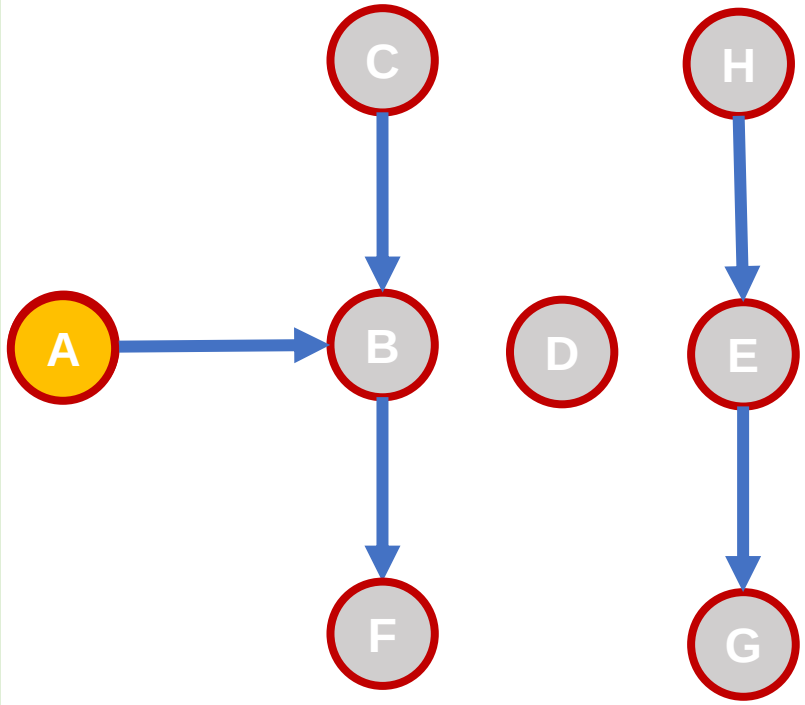


Vértices	Grau Entrada
A	0
B	2
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



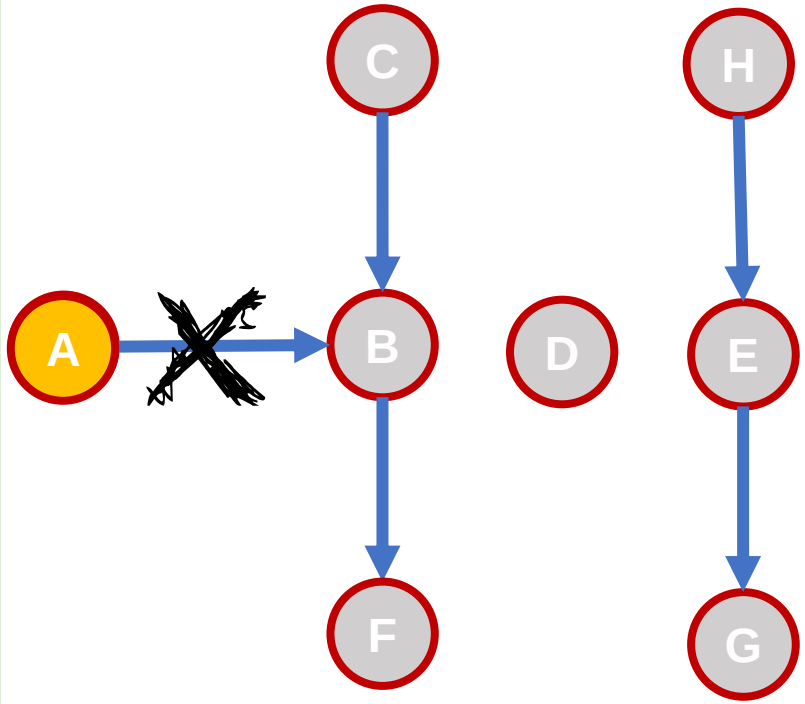
Vértices	Grau Entrada
A	0
B	2
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



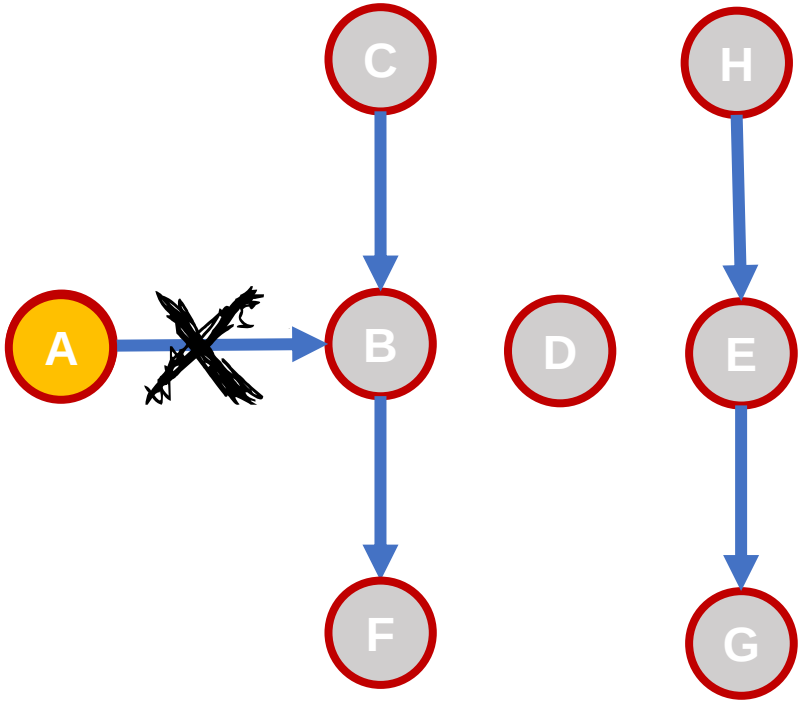
Vértices	Grau Entrada
A	0
B	2
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



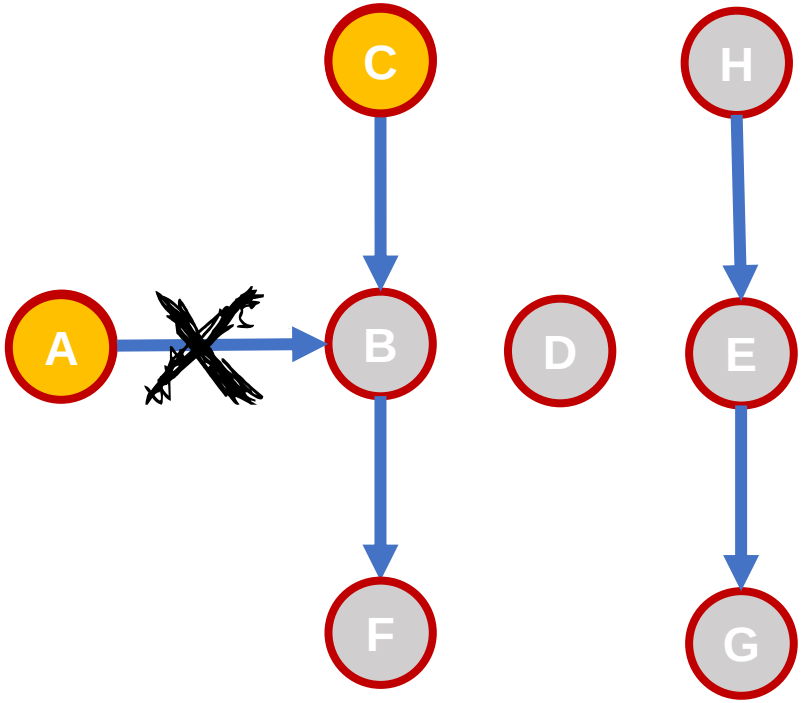
Vértices	Grau Entrada
A	0
B	~~2~~ 1
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



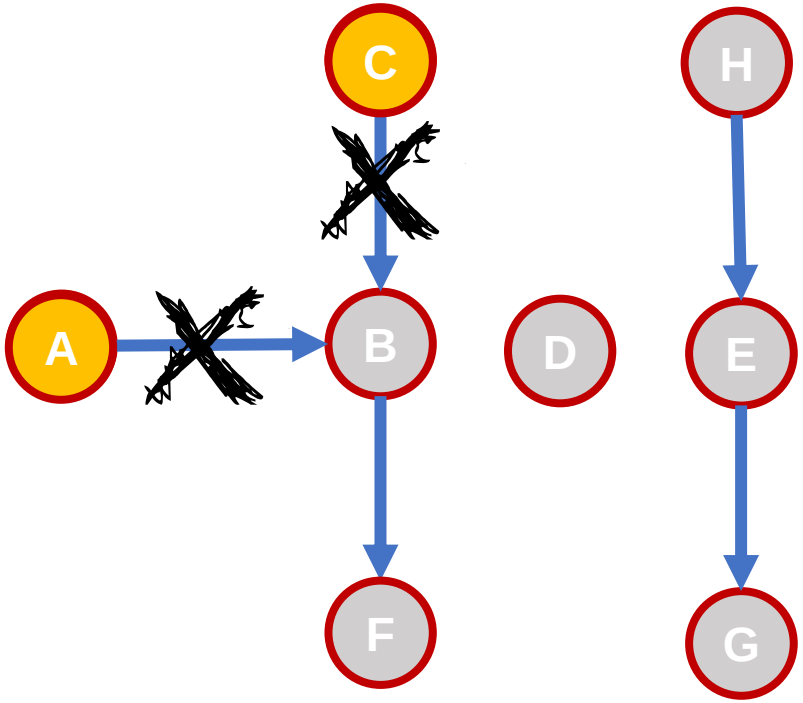
Vértices	Grau Entrada
A	0
B	~~2~~ 1
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A–C

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



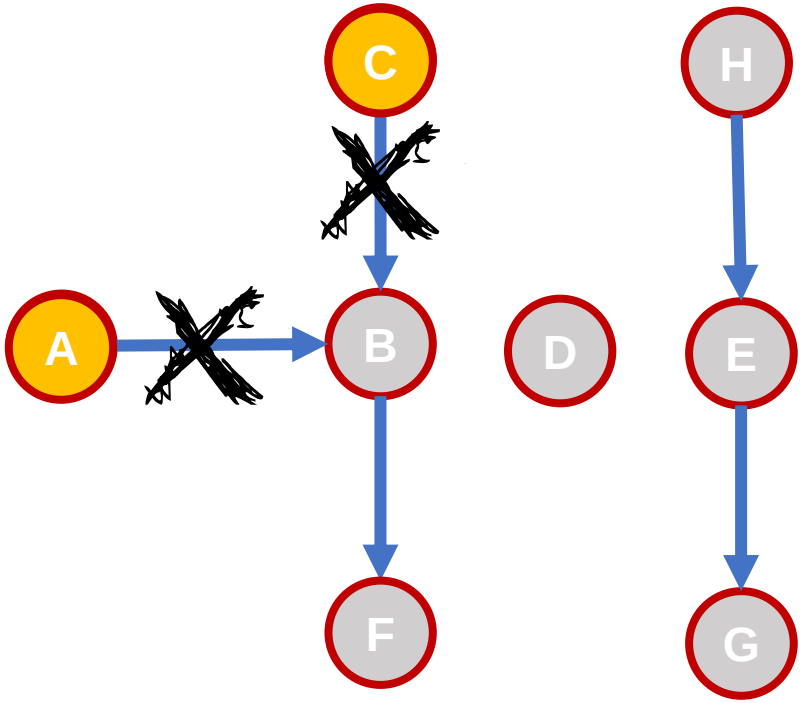
Vértices	Grau Entrada
A	0
B	~~2~~ 1
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A–C

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



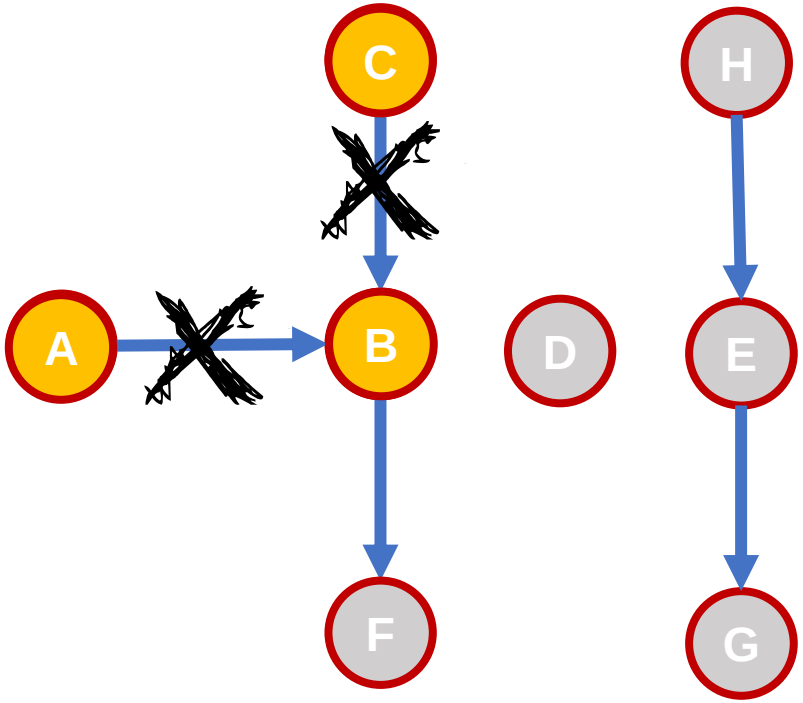
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A – C

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



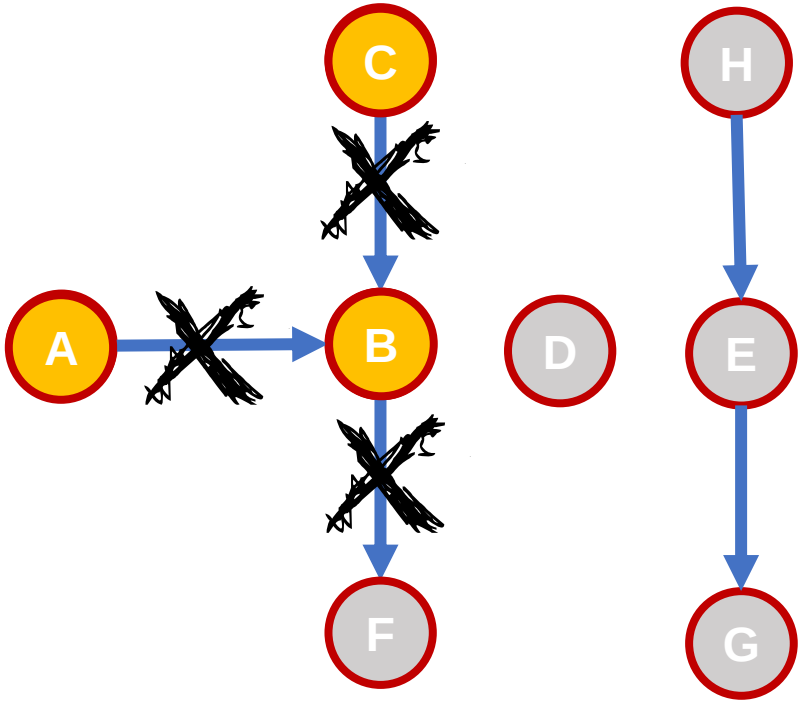
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A – C – B

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



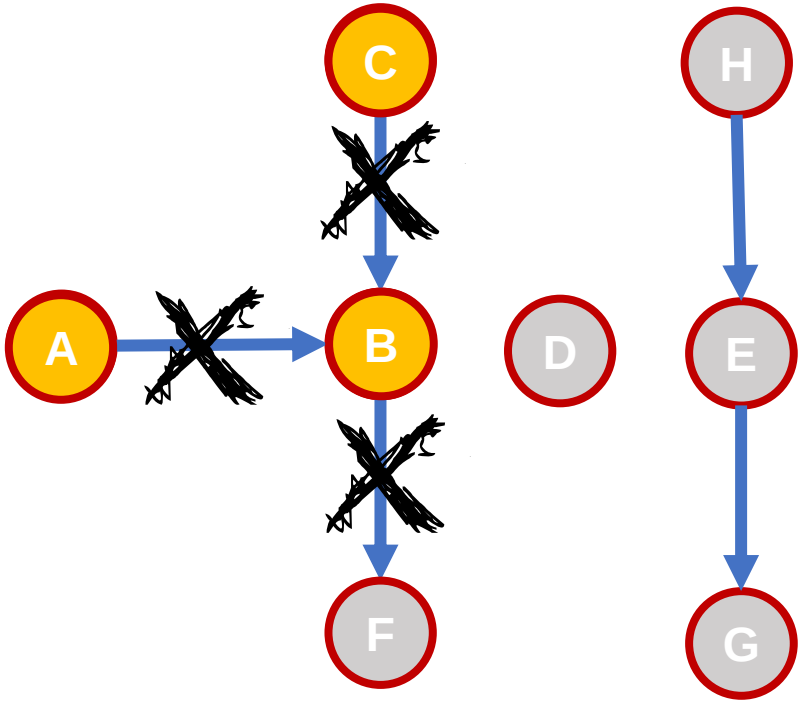
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	1
G	1
H	0

Ordem Topológica

A–C–B

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



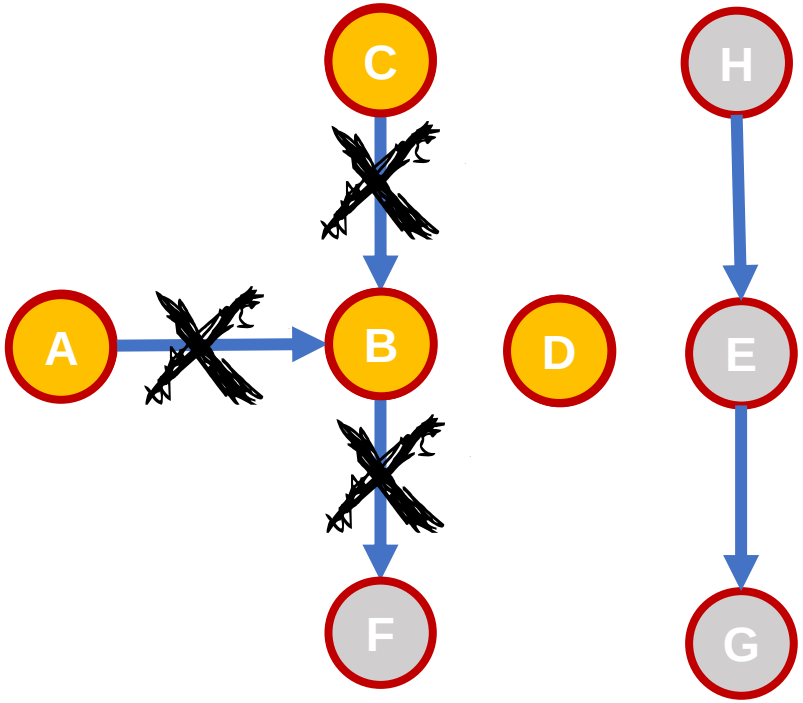
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A – C – B

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



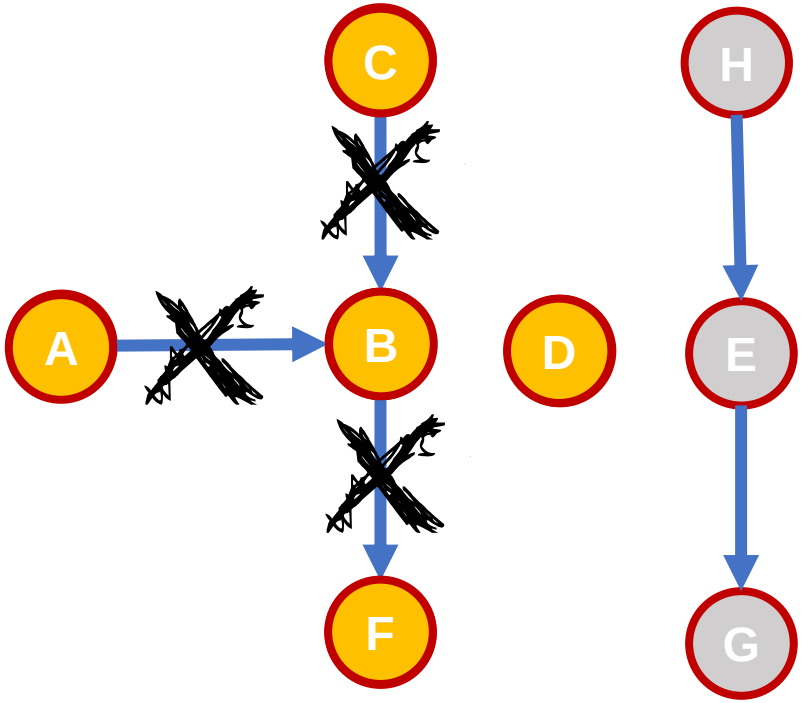
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



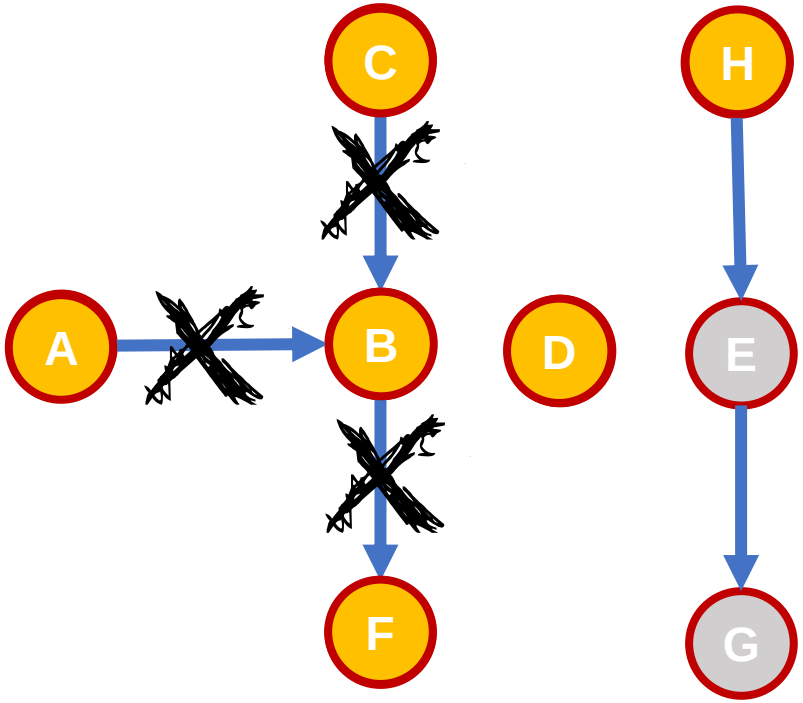
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D–F

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



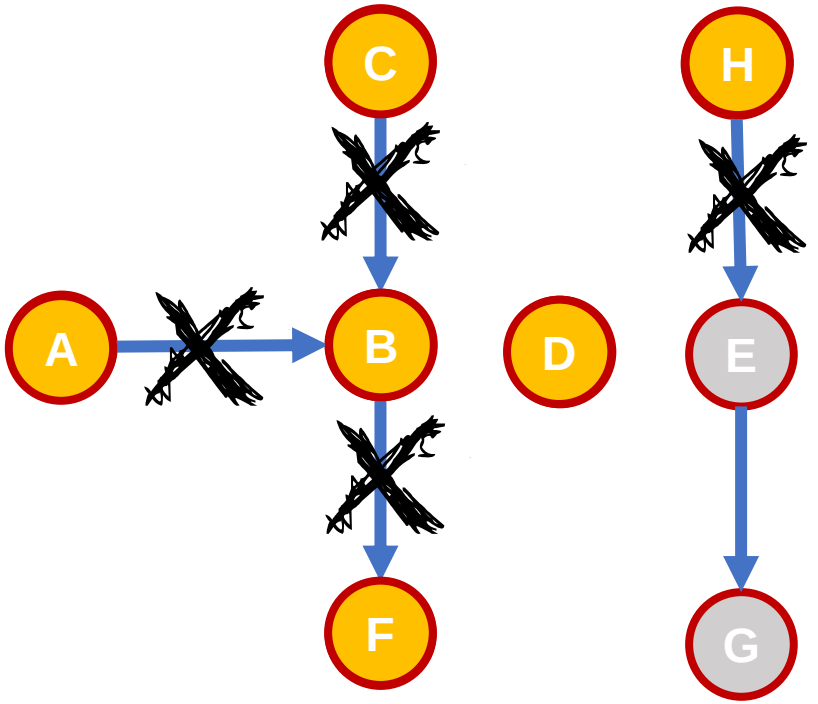
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D–F–H

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



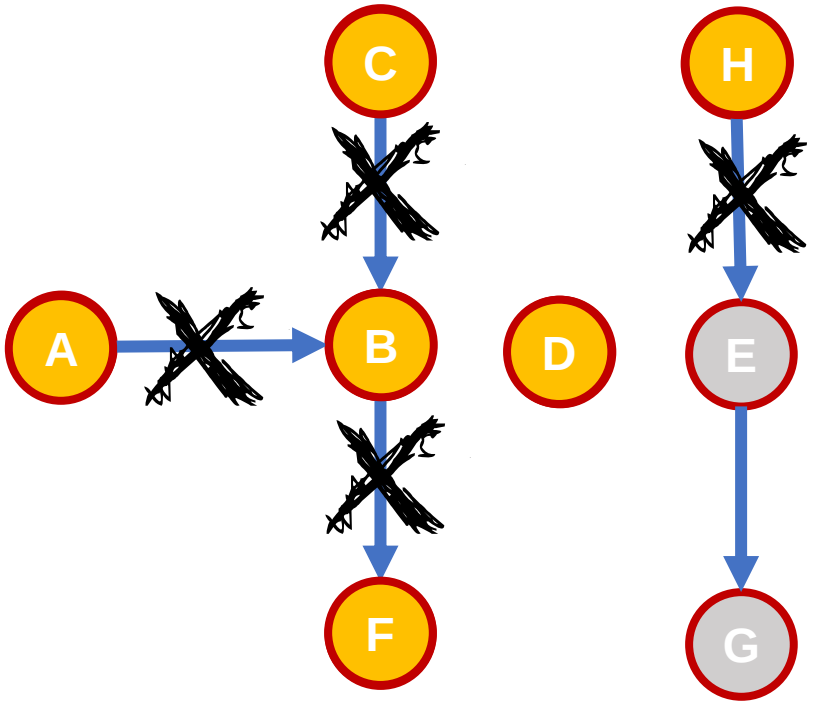
Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	1
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D–F–H

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



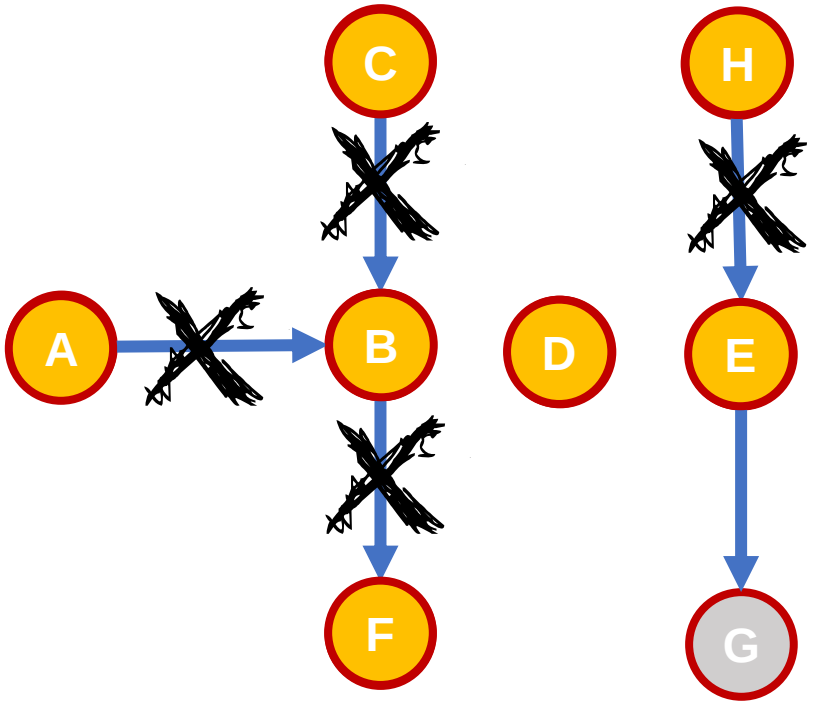
Vértices	Grau Entrada
A	0
B	~~1~~ ~~1~~ 0
C	0
D	0
E	~~1~~ 0
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D–F–H

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



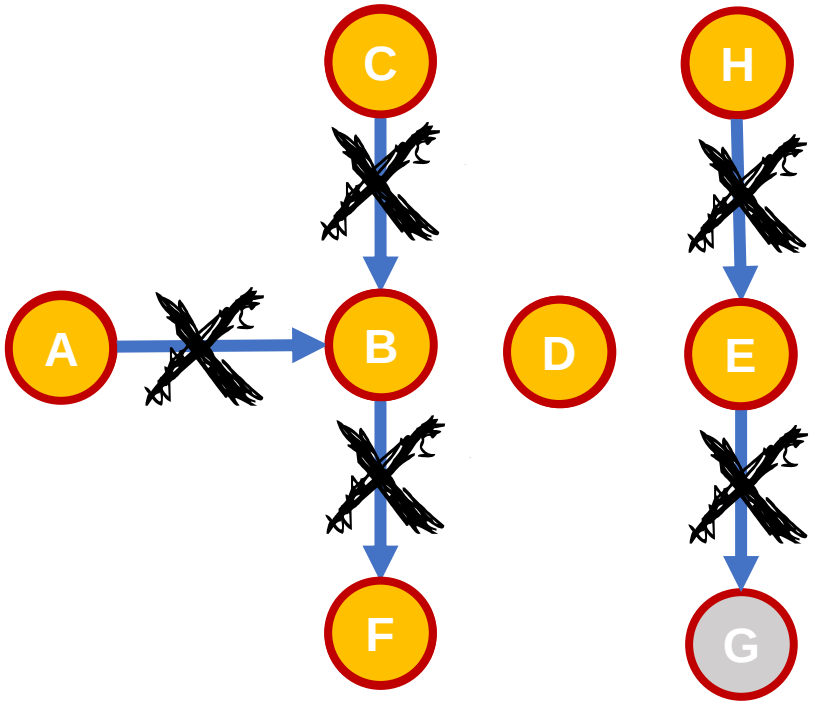
Vértices	Grau Entrada
A	0
B	~~1~~ ~~1~~ 0
C	0
D	0
E	~~1~~ 0
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D–F–H–E

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



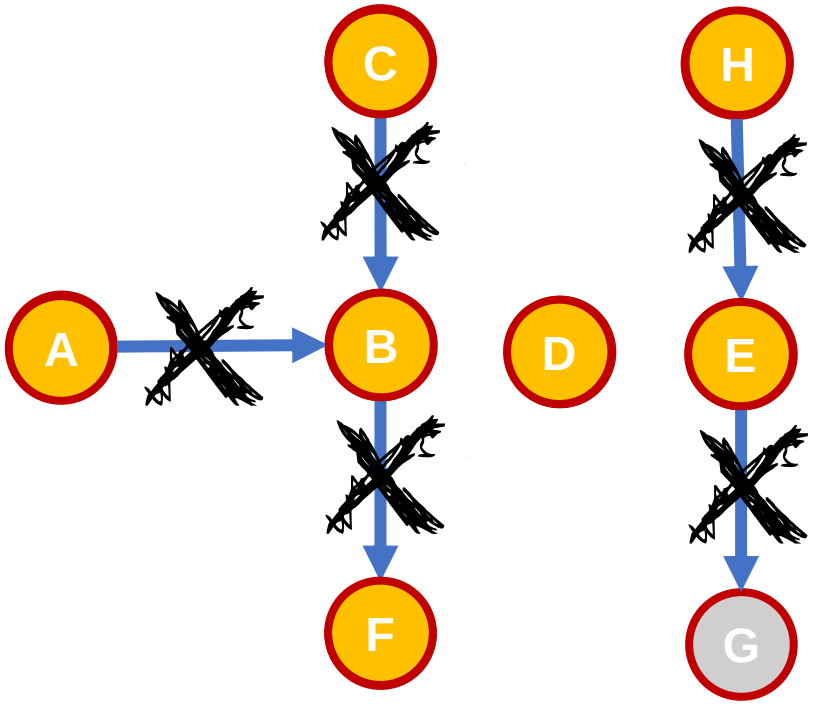
Vértices	Grau Entrada
A	0
B	~~1~~ ~~1~~ 0
C	0
D	0
E	~~1~~ 0
F	~~1~~ 0
G	1
H	0

Ordem Topológica

A–C–B–D–F–H–E

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



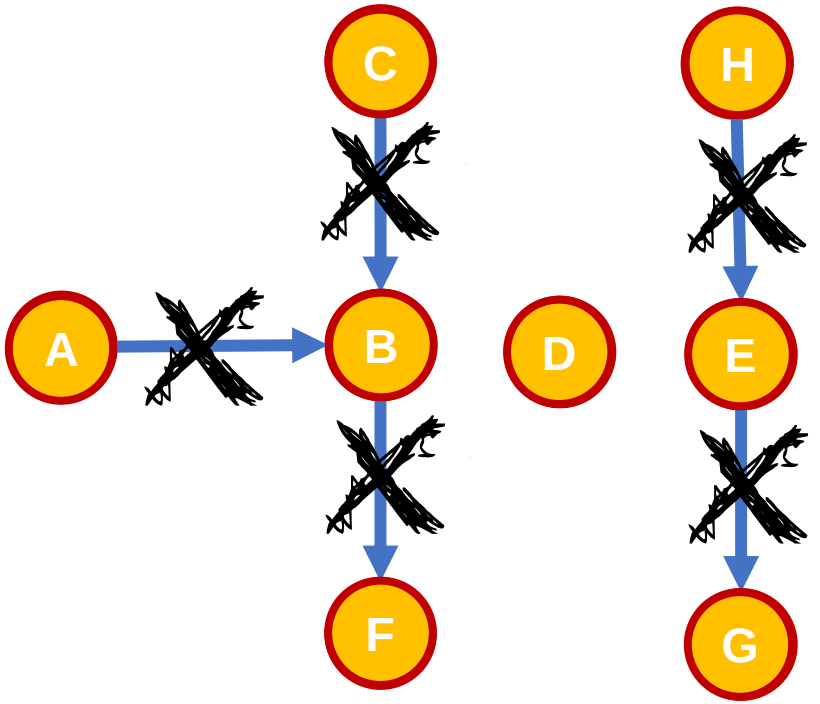
Vértices	Grau Entrada
A	0
B	~~1~~ ~~1~~ 0
C	0
D	0
E	~~1~~ 0
F	~~1~~ 0
G	~~1~~ 0
H	0

Ordem Topológica

A–C–B–D–F–H–E

Algoritmo de Kahn

01	Calcular os graus de entrada dos vértices
02	Enquanto houverem vértices fontes
03	Escolher um
04	Remover as arestas de saída
05	Atualizar tabela de graus



Vértices	Grau Entrada
A	0
B	~~2~~ ~~1~~ 0
C	0
D	0
E	~~2~~ 0
F	~~1~~ 0
G	~~1~~ 0
H	0

Ordem Topológica

A–C–B–D–F–H–E–G

Ordenação Topológica com DFS

- Robert Tarjan propôs uma forma para identificar uma ordem topológica baseada na Busca em Profundidade
  - Define um vértice inicial v
    - Aplica DFS(v)
      - Quando encontrar um vértice sem adjacentes ele deve ser incluído na ordem topológica

Ordenação Topológica com DFS

- Robert Tarjan propôs uma forma para identificar uma ordem topológica baseada na Busca em Profundidade
  - Define um vértice inicial v
    - Aplica DFS(v)
      - Quando encontrar um vértice sem adjacentes ele deve ser incluído na ordem topológica

- Nesta proposta, as tarefas são organizadas de trás para frente, ou seja, as últimas atividades serão selecionadas primeiro

Algoritmos

... estudar e praticar ... the best way for everything

Algoritmos


```
01  Dado um G (V, A)
02  Se tem certeza que é um DAG
03          dfs_01 (vértice_inicial)
04  senão
05      para cada v em V
06          cor[v] = branco \\ não visitado
07          pai[v] = -1
08      tempo = 0
09      dfs_02(vértice_inicial)
```

Algoritmos


```
01  Dado um G (V, A)
02  Se tem certeza que é um DAG
03          dfs_01 (vértice_inicial)
04  senão
05      para cada v em V
06          cor[v] = branco \\ não visitado
07          pai[v] = -1
08      tempo = 0
09      dfs_02(vértice_inicial)
```

```
01  dfs_01 (v)
02      para cada u adjacente de v
03          se !visitado (u)
04              dfs_01 (u)
05      inserir v na frente da lista
```

Algoritmos

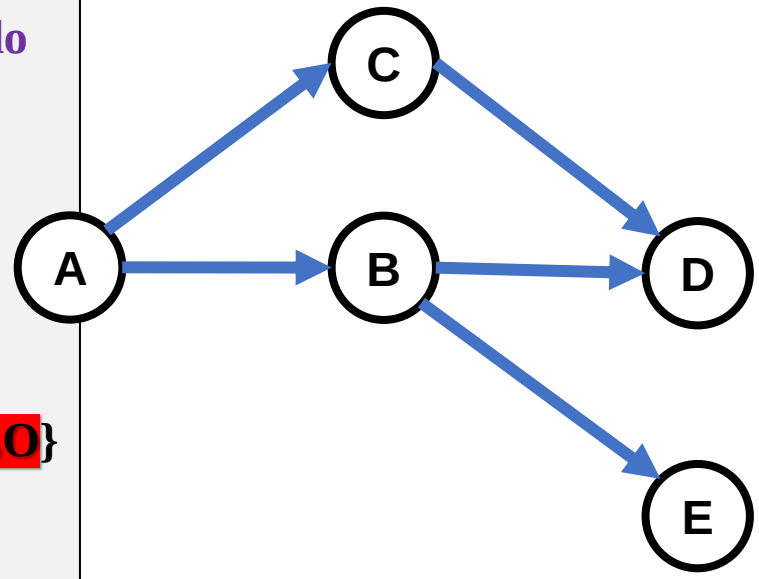

```
01  Dado um G (V, A)
02  Se tem certeza que é um DAG
03          dfs_01 (vértice_inicial)
04  senão
05      para cada v em V
06          cor[v] = branco \\ não visitado
07          pai[v] = -1
08      tempo = 0
09      dfs_02(vértice_inicial)
```

```
01  dfs_01 (v)
02      para cada u adjacente de v
03          se !visitado (u)
04              dfs_01 (u)
05      inserir v na frente da lista
```

```
01  dfs_02(v){
02      cor[v] = laranja \\ processando
03      d[v] = ++tempo
04      para cada u adjacente de v {
05          se cor[u] == branco{
06                  pai[u] = v
07                  dfs_02(u)
08          }
09          se cor[u] == laranja {CICLO}
10      }
11       cor[v] = verde \\ finalizado
12      f [v] = tempo++
13      inserir v no fim da lista
14  }
```

Executando DFS 2

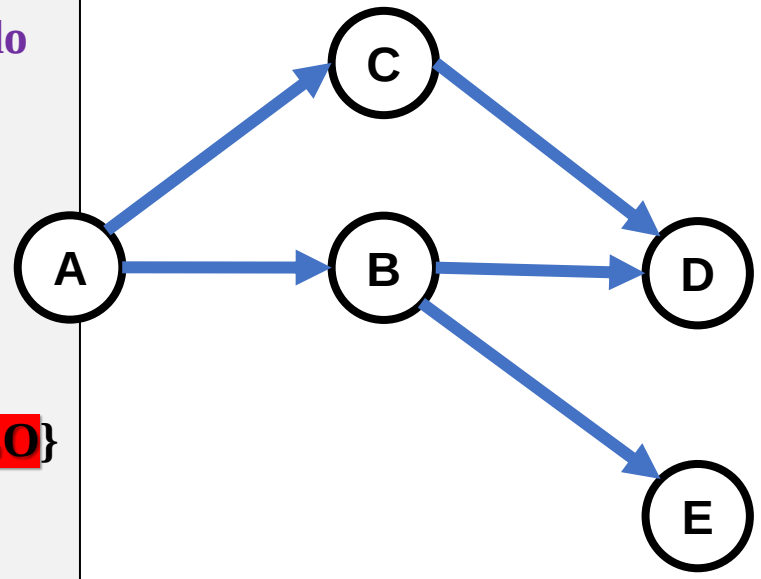
```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

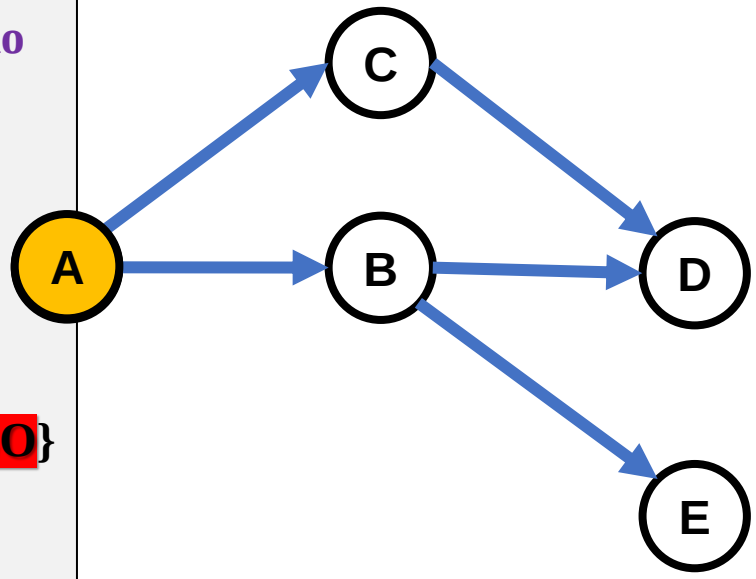


Vértice	pai	d	f
A	-I		
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

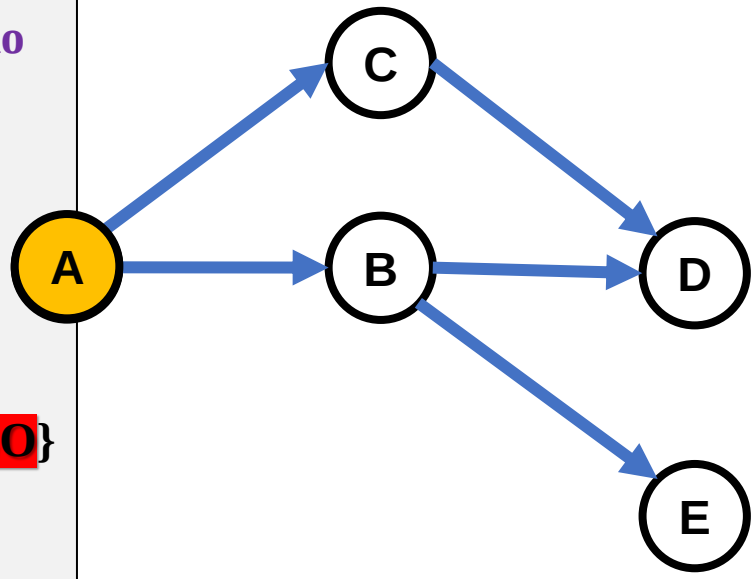


Vértice	pai	d	f
A	-I		
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

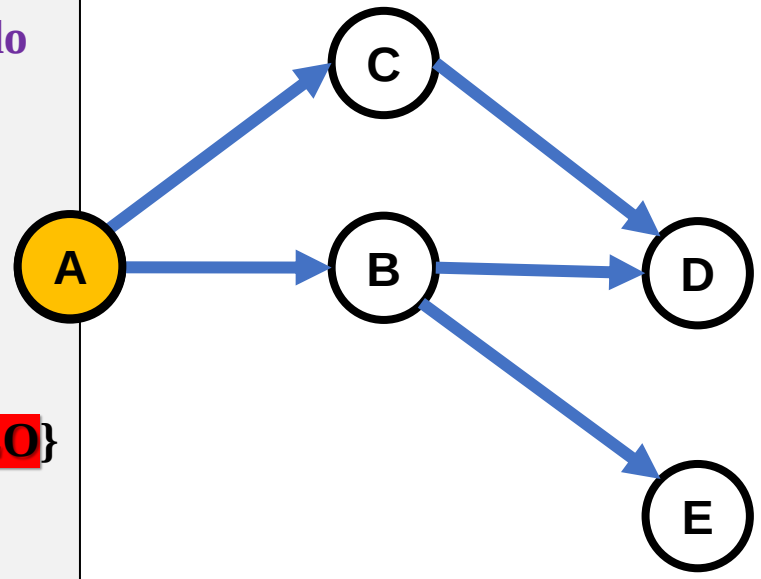


Vértice	pai	d	f
A	-I	I	
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



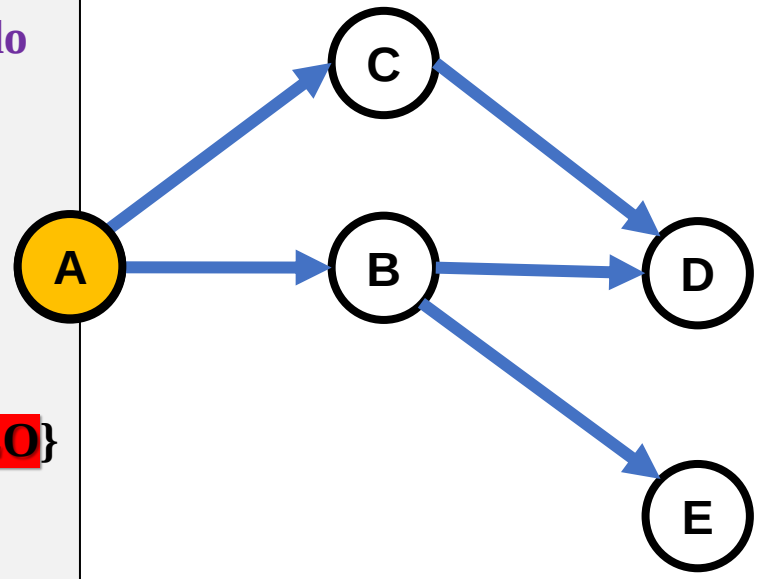
Vértice	pai	d	f
A	-I	I	
B	-I		
C	-I		
D	-I		
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



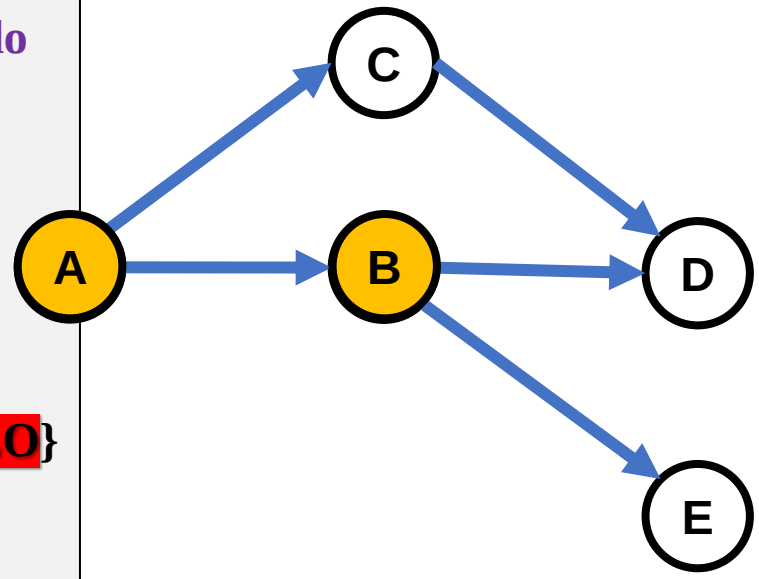
Vértice	pai	d	f
A	-I	I	
B	-X A		
C	-I		
D	-I		
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



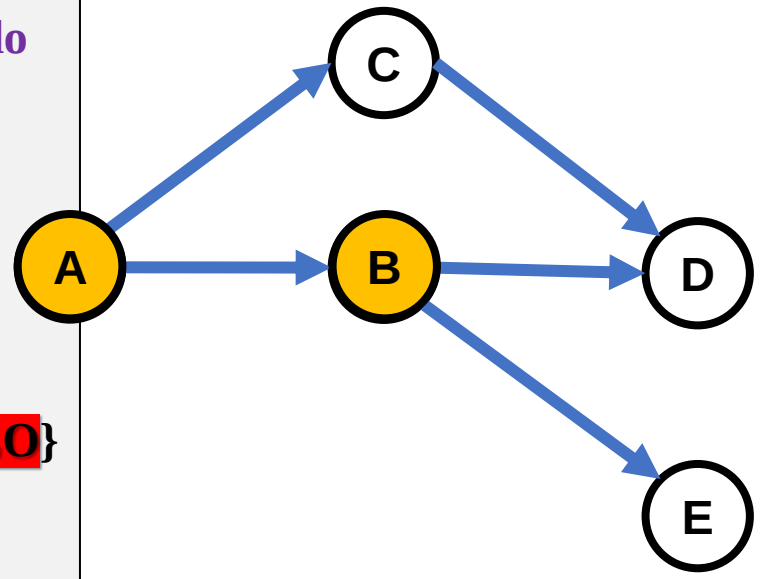
Vértice	pai	d	f
A	-I	I	
B	-X A		
C	-I		
D	-I		
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



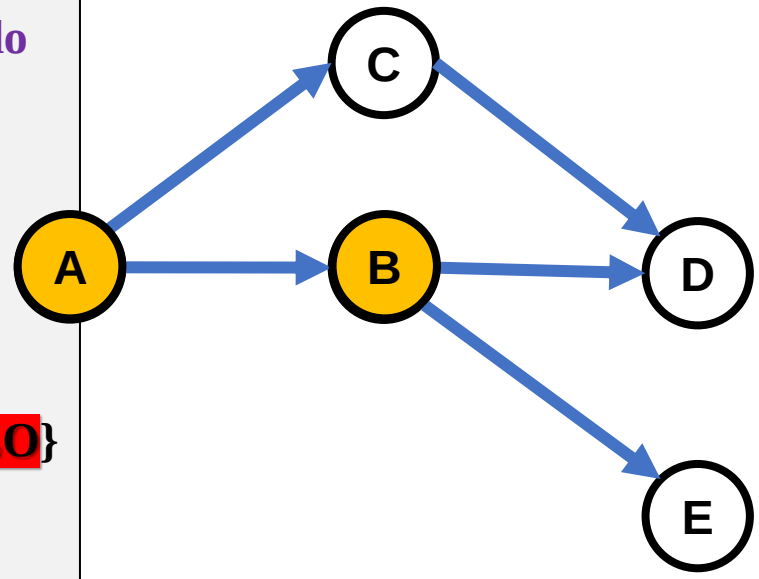
Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I		
D	-I		
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



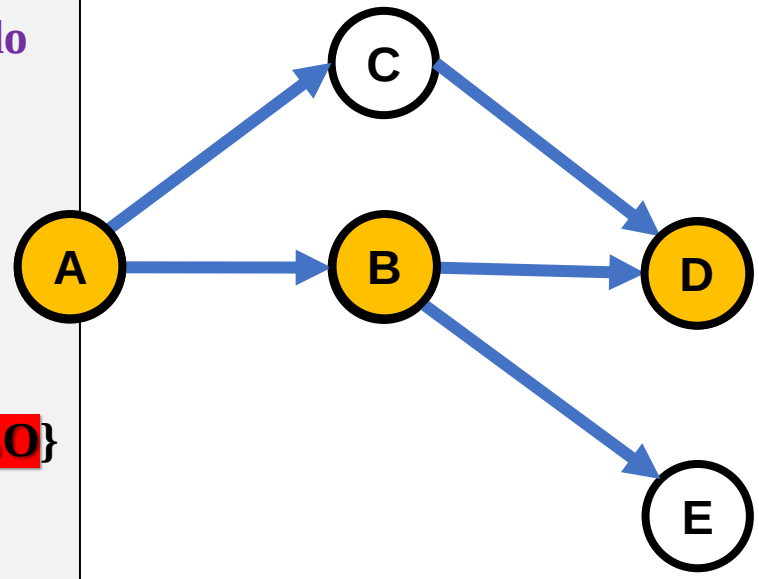
Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B		
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



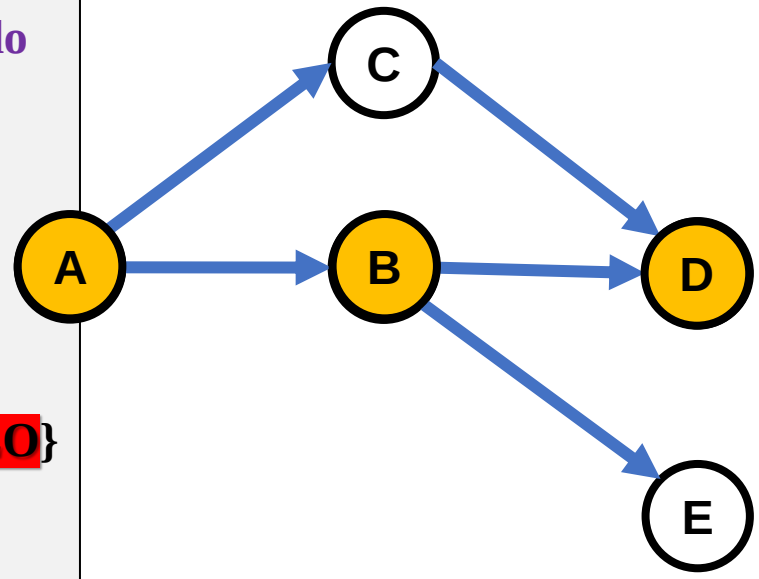
Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B		
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



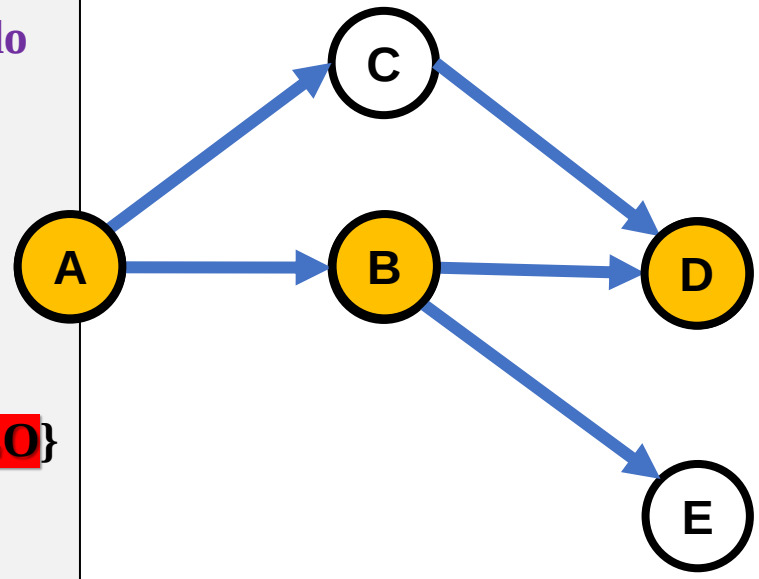
Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	
E	-I		

Verificar os adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	
E	-I		

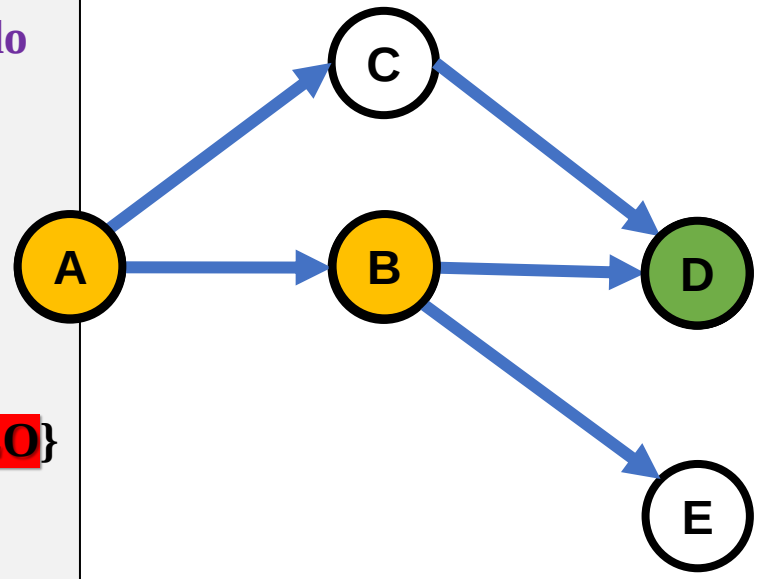
Verificar os adjacentes

Sem adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	
E	-I		

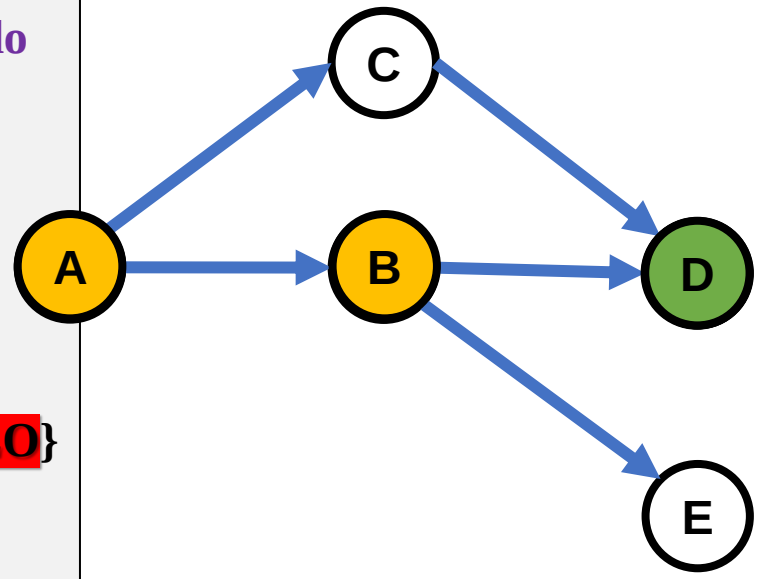
Verificar os adjacentes

Sem adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-I		

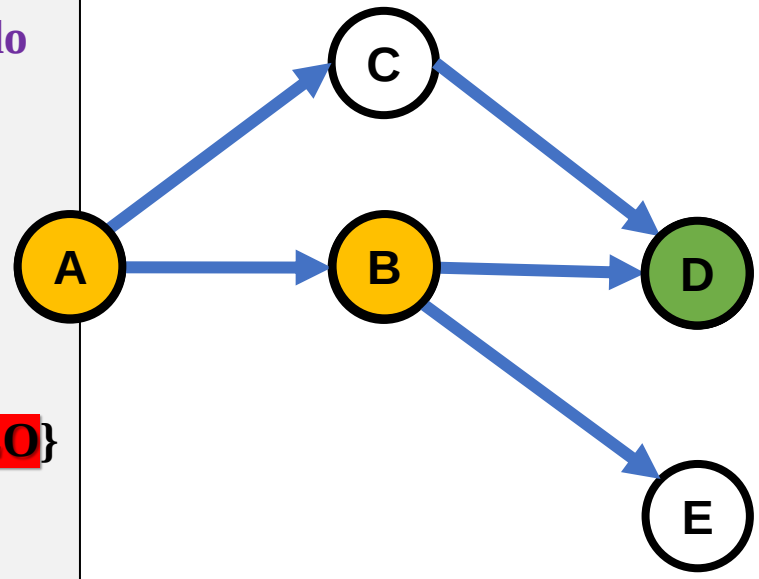
Verificar os adjacentes

Sem adjacentes

Ordem Topológica

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-I		

Verificar os adjacentes

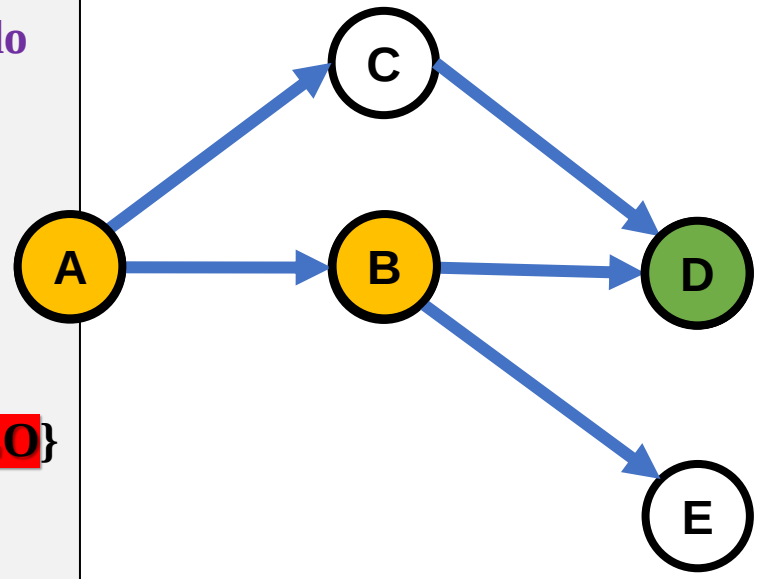
Sem adjacentes

Ordem Topológica

D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B		

Verificar os adjacentes

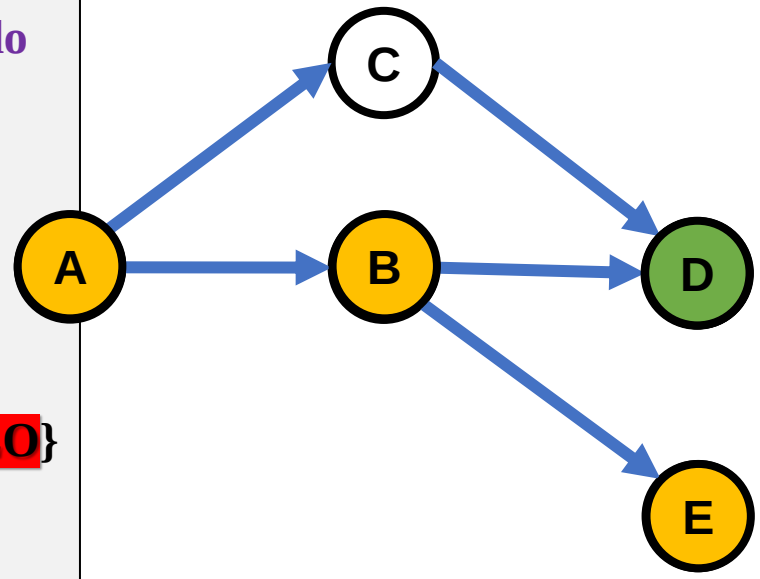
Sem adjacentes

Ordem Topológica

D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B		

Verificar os adjacentes

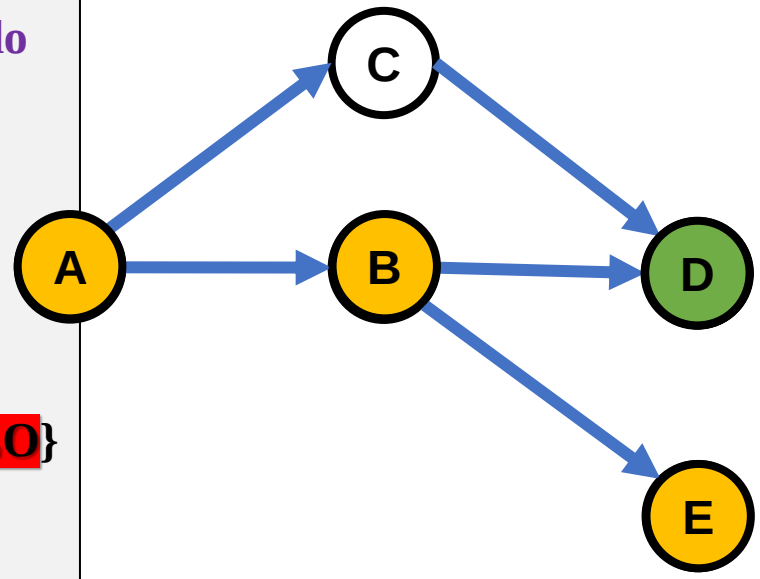
Sem adjacentes

Ordem Topológica

D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B	5	

Verificar os adjacentes

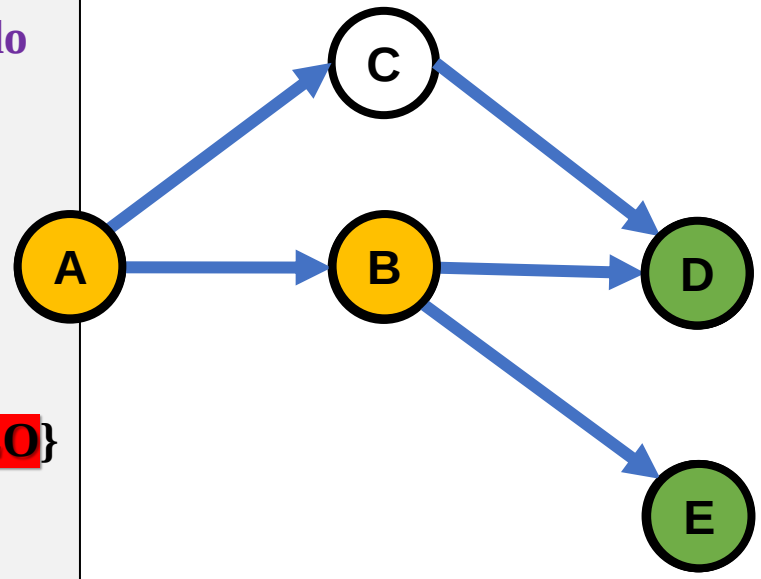
Sem adjacentes

Ordem Topológica

D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B	5	

Verificar os adjacentes

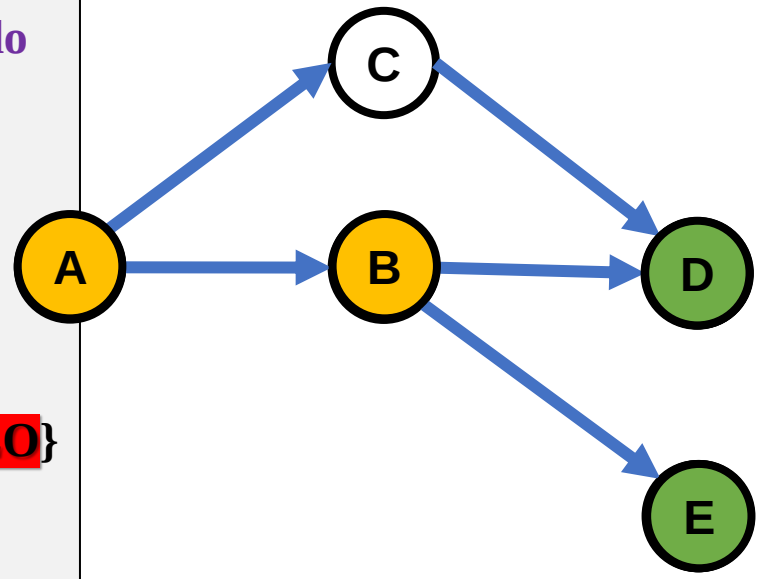
Sem adjacentes

Ordem Topológica

D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

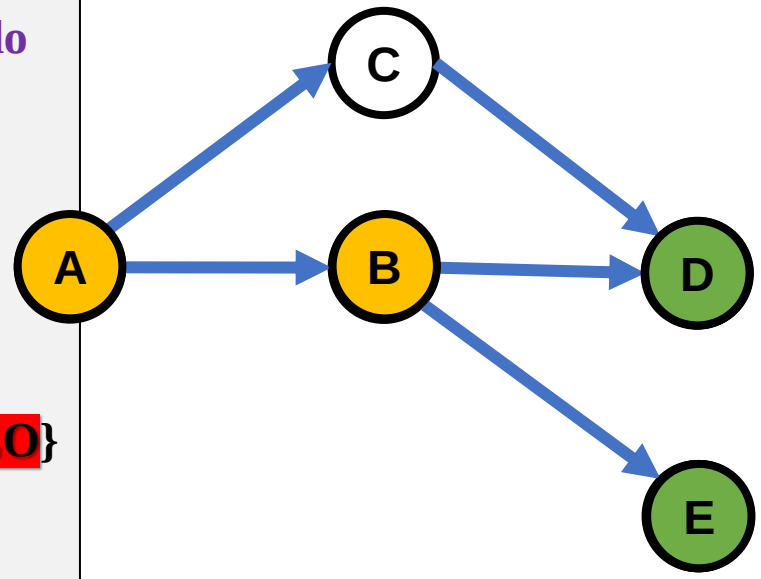
Sem adjacentes

Ordem Topológica

D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

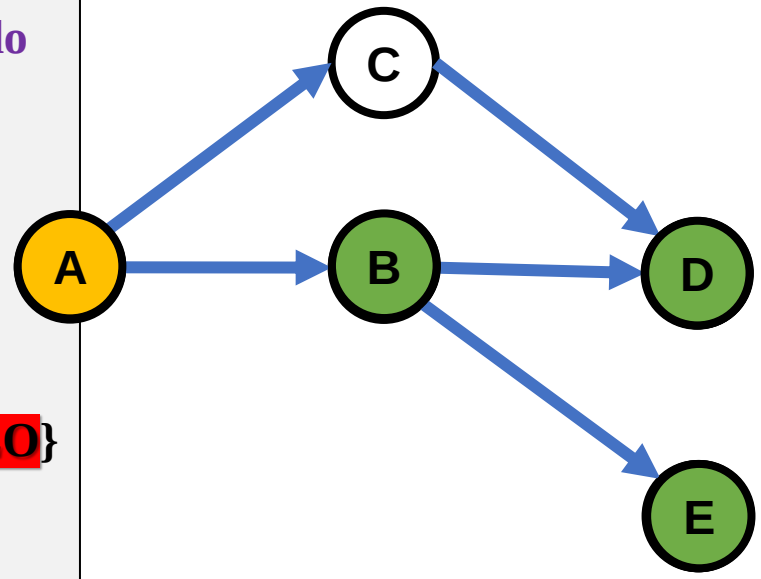
Sem adjacentes

Ordem Topológica

E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	
C	-I		
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

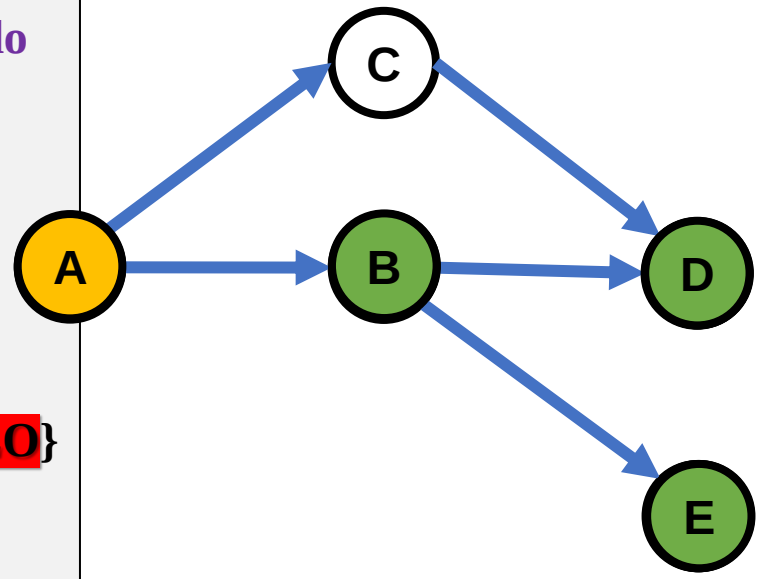
Sem adjacentes

Ordem Topológica

E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-I		
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

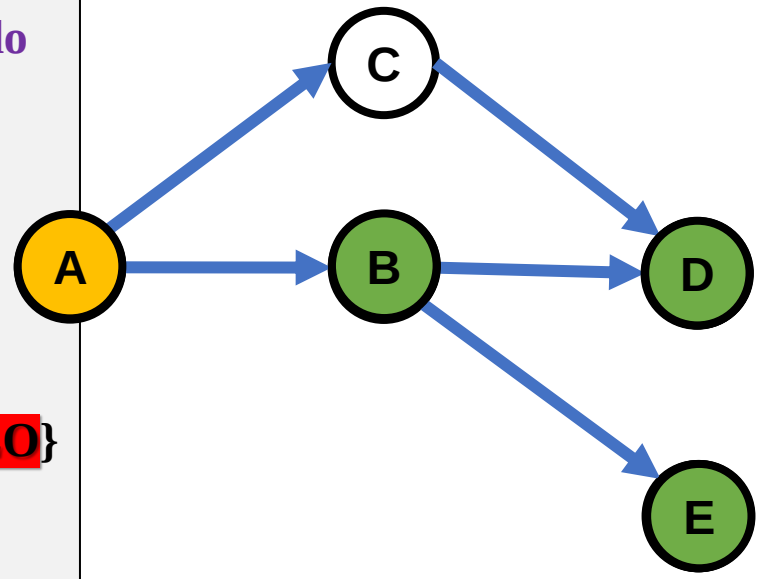
Sem adjacentes

Ordem Topológica

E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-I		
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

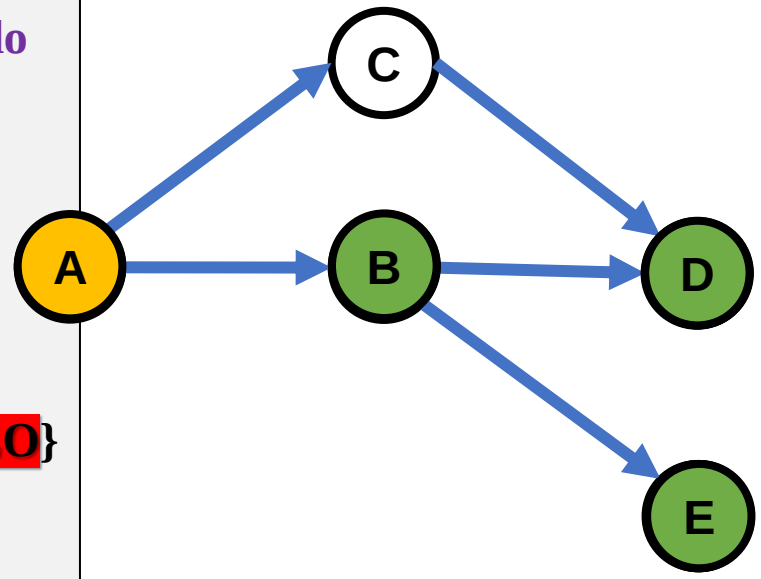
Sem adjacentes

Ordem Topológica

B-E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-X A		
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

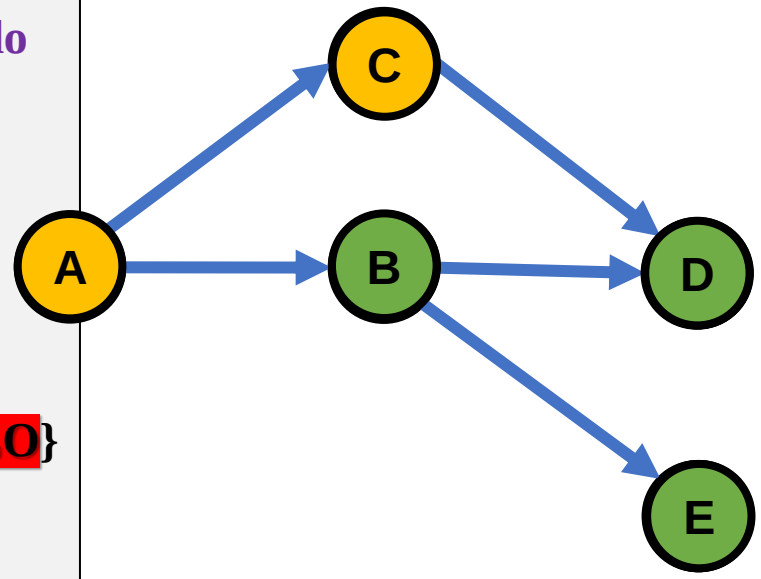
Sem adjacentes

Ordem Topológica

B-E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	~~-1~~	1	
B	~~-1~~ A	2	7
C	~~-1~~ A		
D	~~-1~~ B	3	4
E	~~-1~~ B	5	6

Verificar os adjacentes

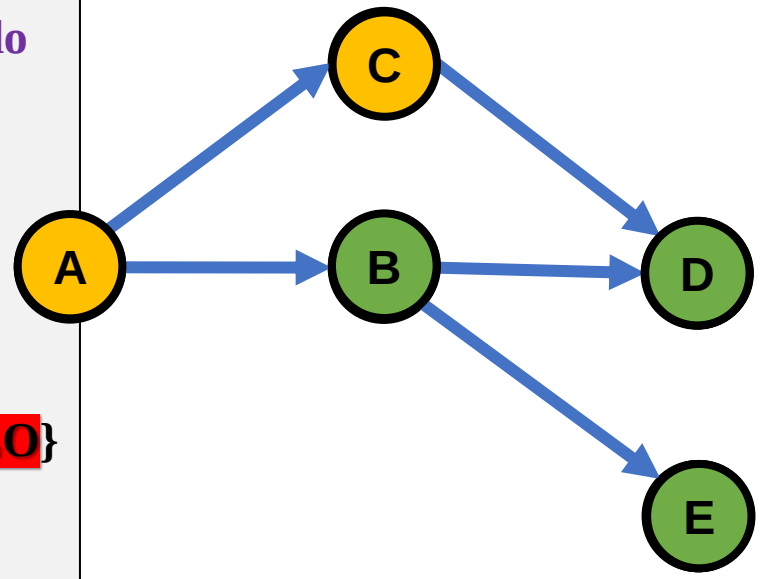
Sem adjacentes

Ordem Topológica

B – E – D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-X A	8	
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

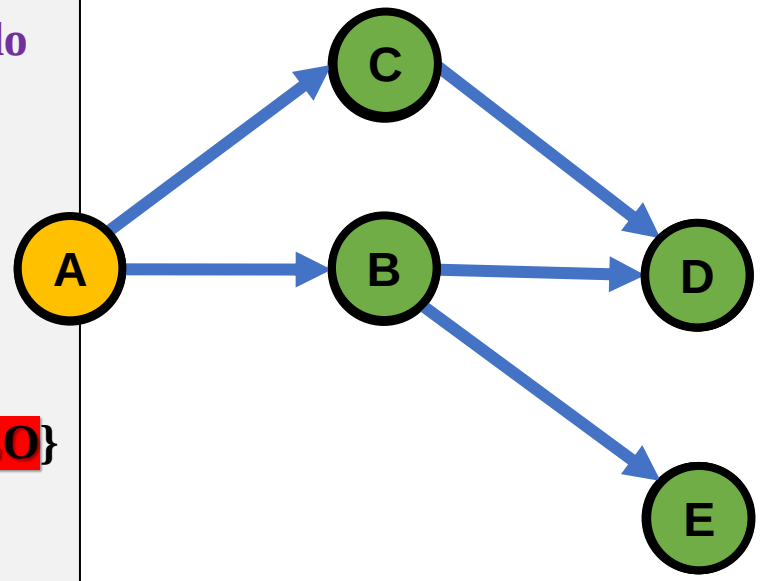
Sem adjacentes

Ordem Topológica

B-E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-X A	8	
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

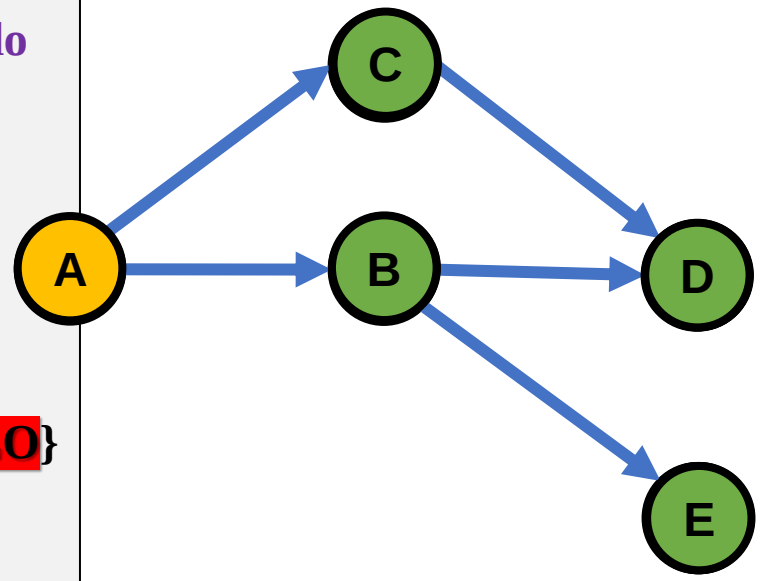
Sem adjacentes

Ordem Topológica

B-E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-X A	8	9
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

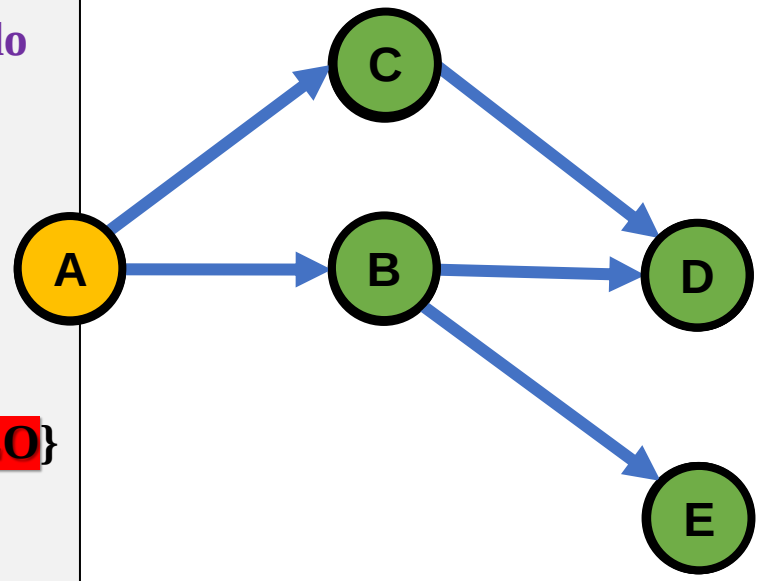
Sem adjacentes

Ordem Topológica

B-E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-X A	8	9
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

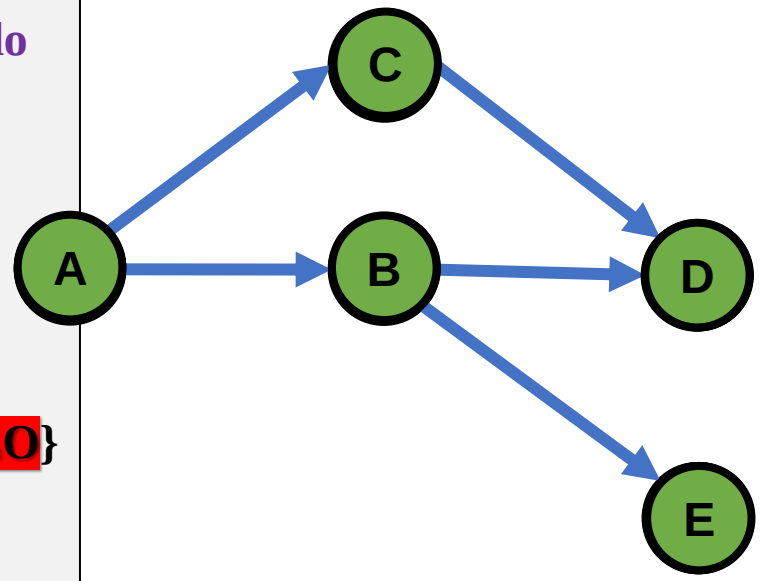
Sem adjacentes

Ordem Topológica

C – B – E – D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	
B	-X A	2	7
C	-X A	8	9
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

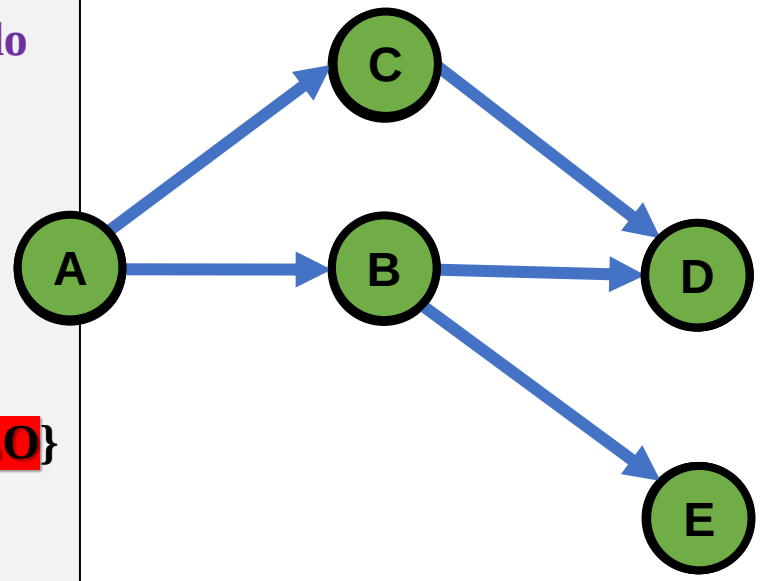
Sem adjacentes

Ordem Topológica

C – B – E – D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	IO
B	-X A	2	7
C	-X A	8	9
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

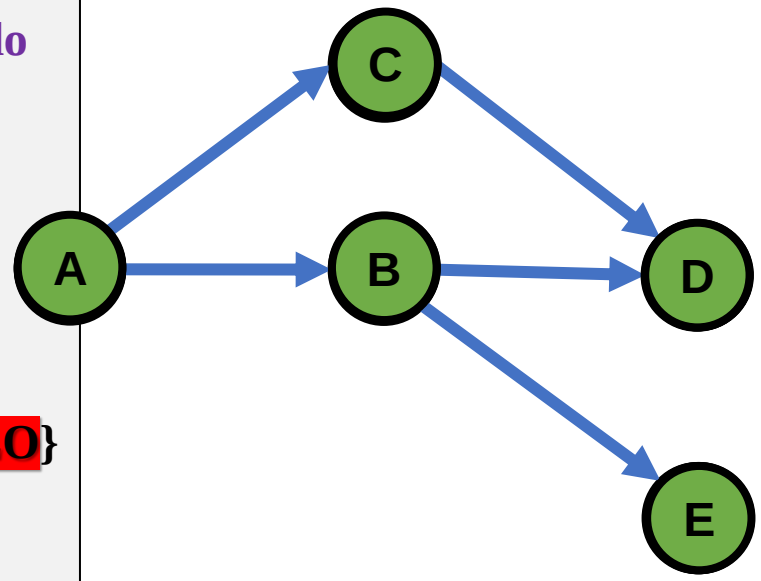
Sem adjacentes

Ordem Topológica

C-B-E-D

Executando DFS 2

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Vértice	pai	d	f
A	-I	I	IO
B	-X A	2	7
C	-X A	8	9
D	-X B	3	4
E	-X B	5	6

Verificar os adjacentes

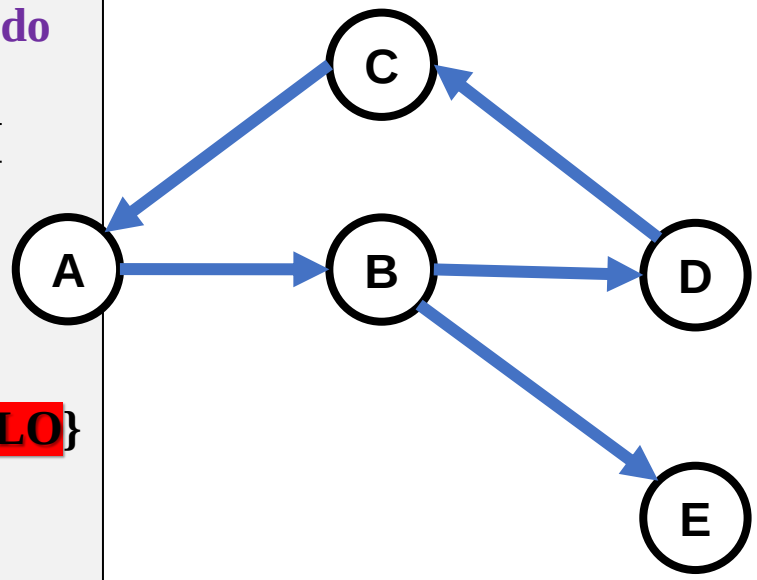
Sem adjacentes

Ordem Topológica

A-C-B-E-D

Executando DFS 2 novamente

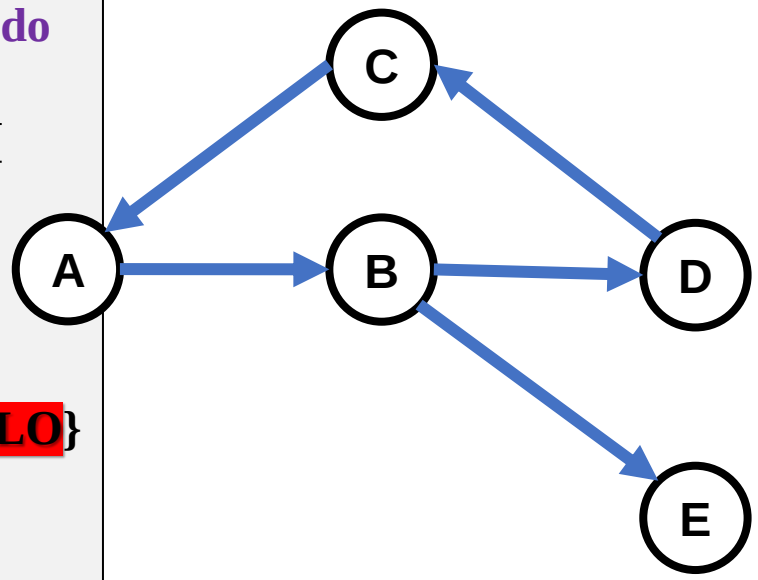
```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

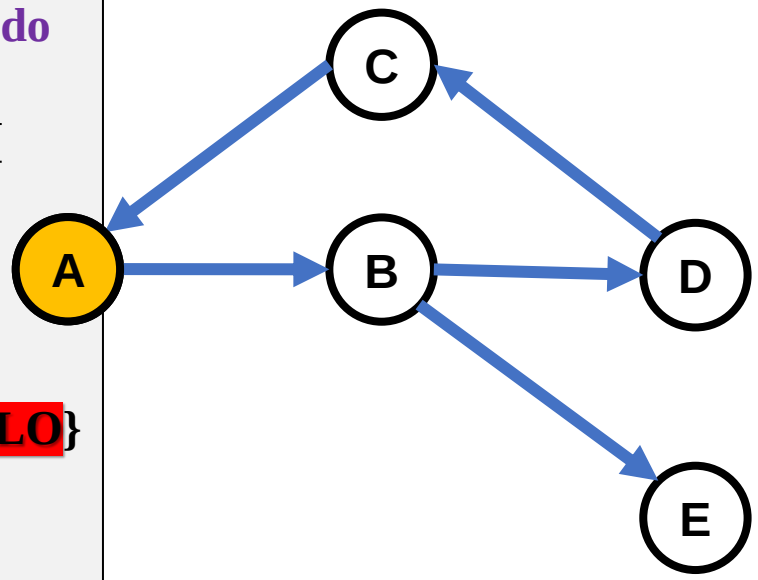


Vértice	pai	d	f
A	-I		
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

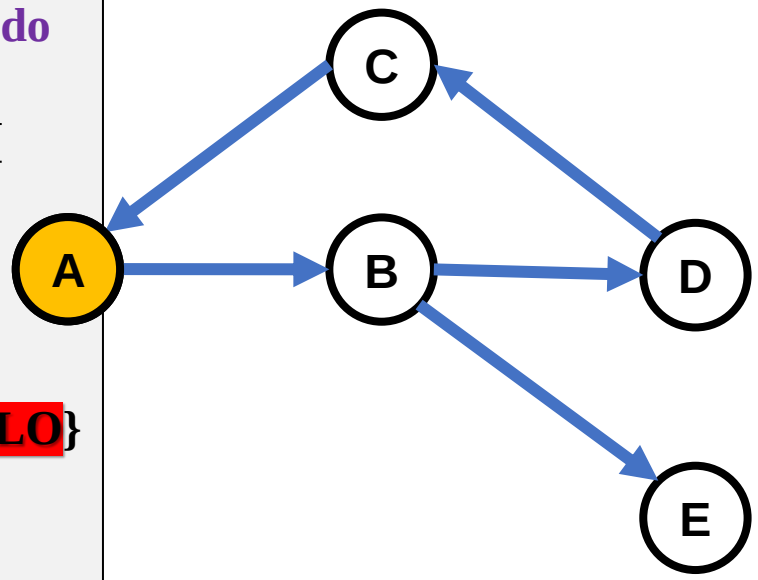


Vértice	pai	d	f
A	-I		
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

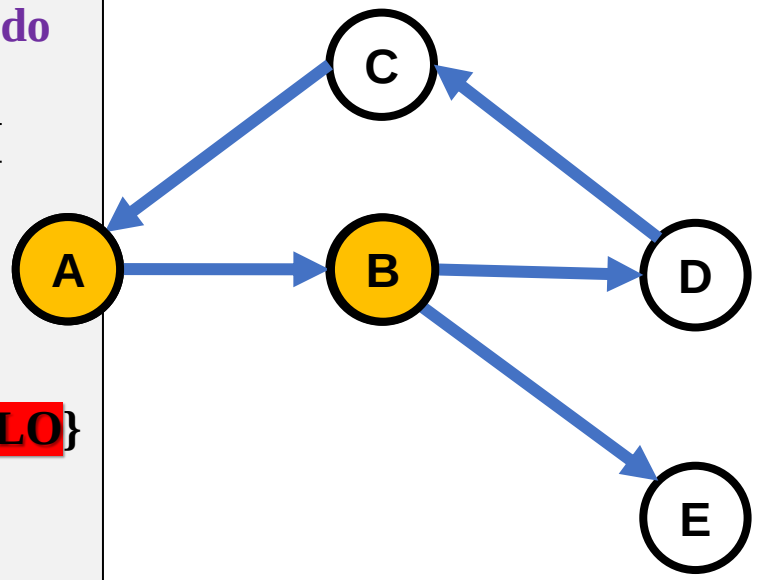


Vértice	pai	d	f
A	-I	I	
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

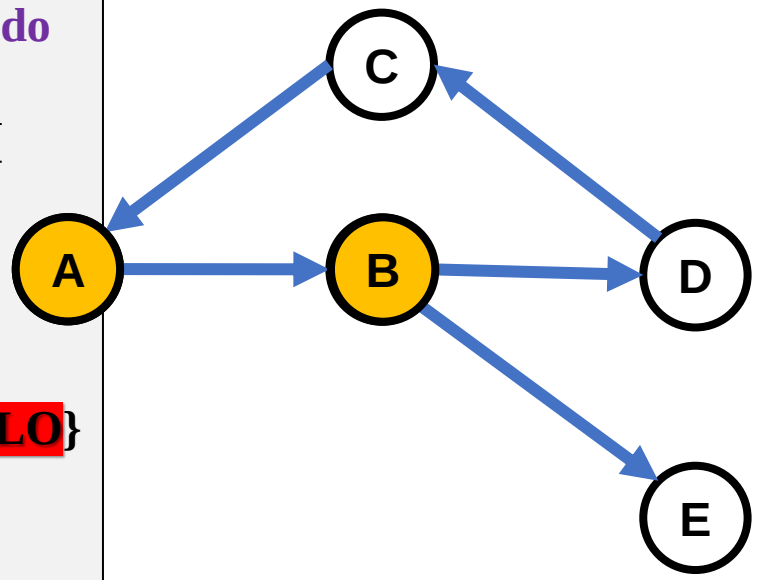


Vértice	pai	d	f
A	-I	I	
B	-I		
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

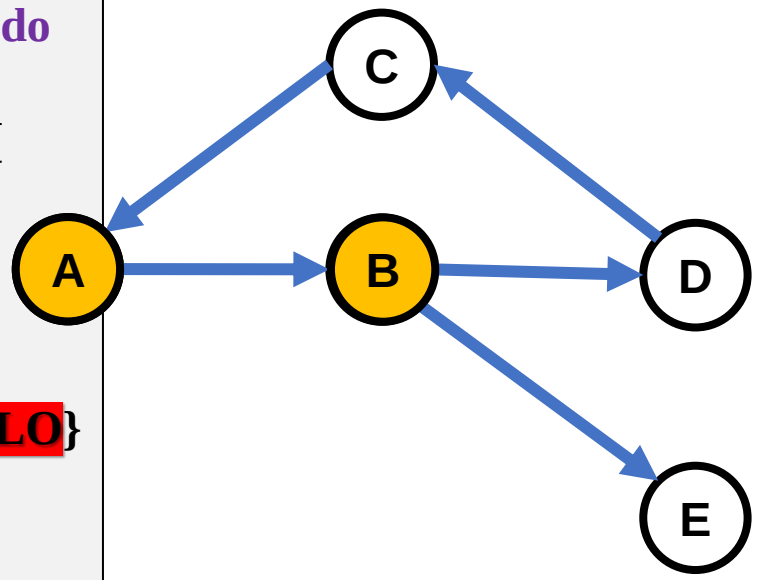


Vértice	pai	d	f
A	-I	I	
B	-I	2	
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

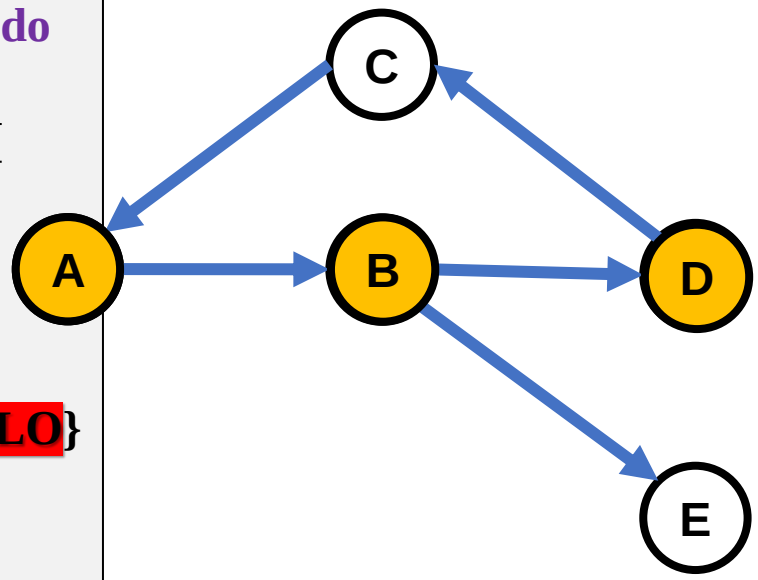


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

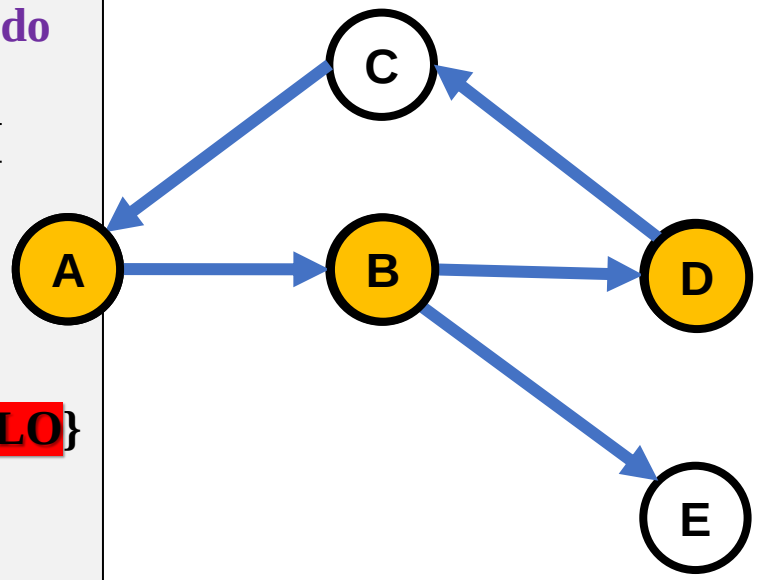


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I		
D	-I		
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

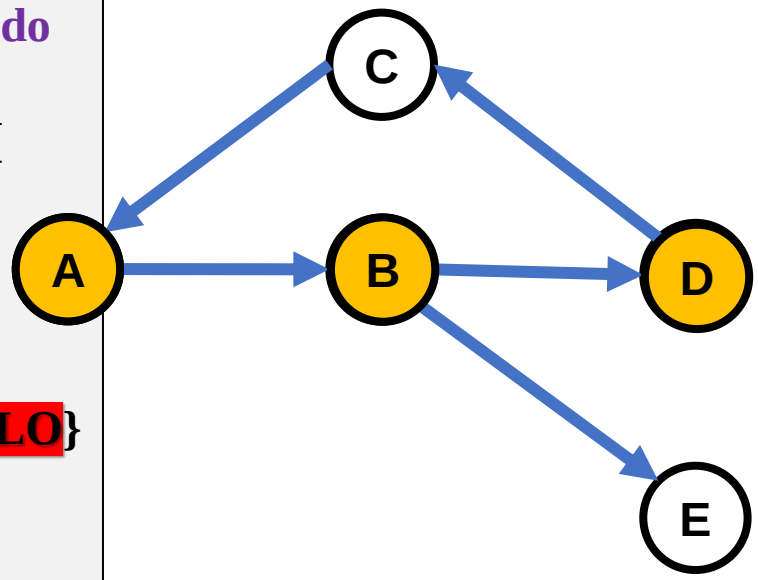


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I		
D	-I	3	
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

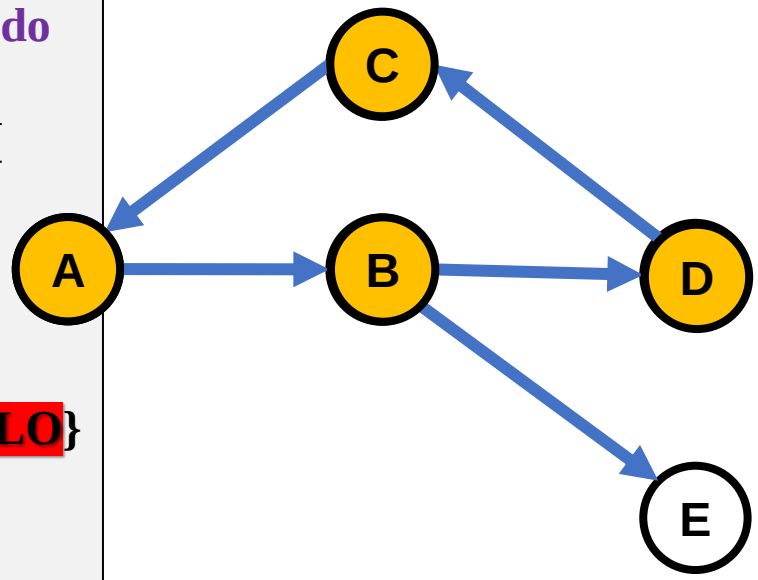


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I		
D	~~-I~~ B	3	
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

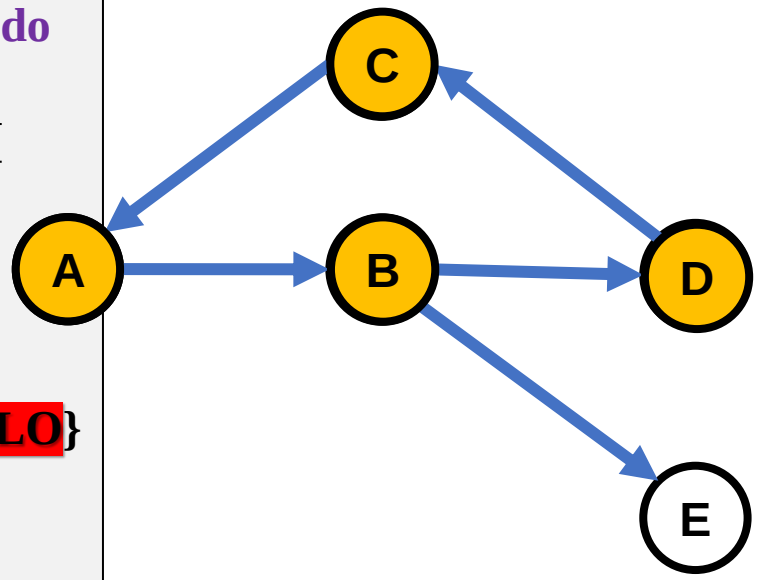


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I		
D	~~-I~~ B	3	
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

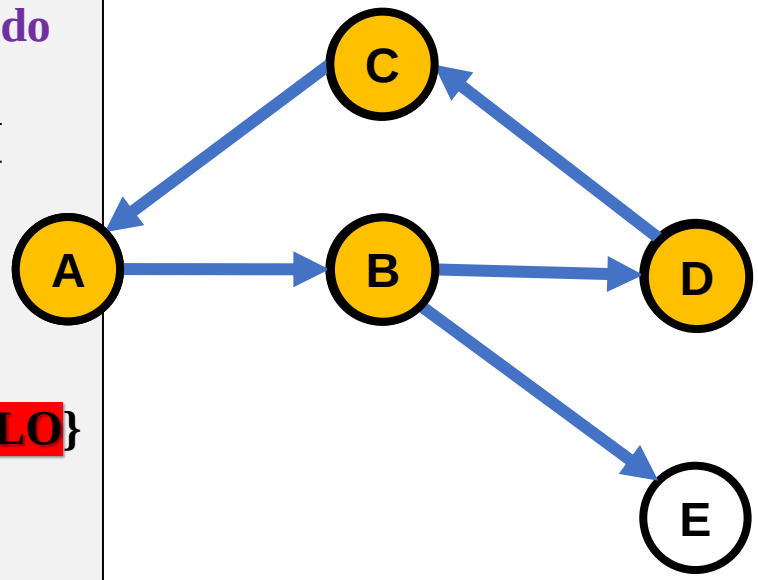


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	-I	4	
D	~~-I~~ B	3	
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

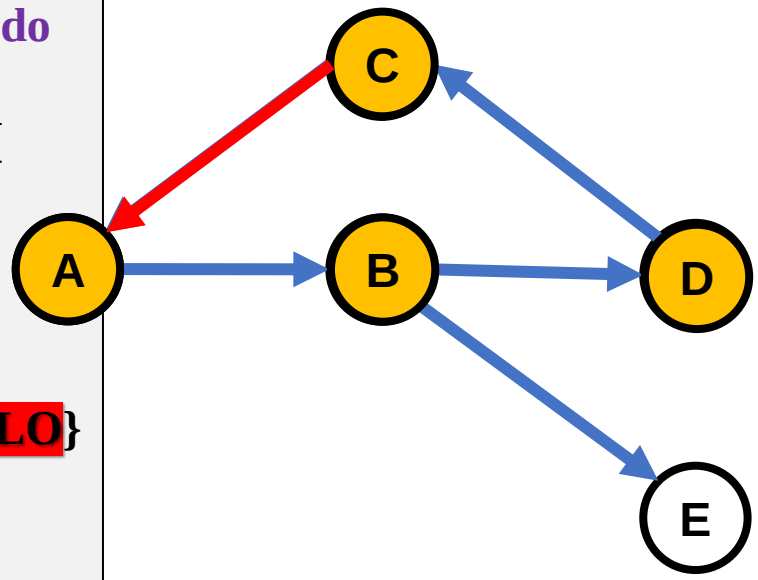


Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	~~-I~~ D	4	
D	~~-I~~ B	3	
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```

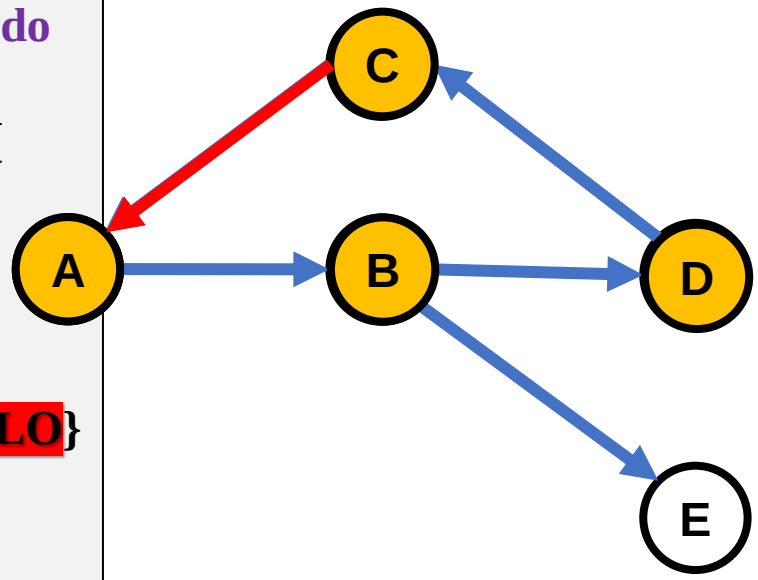


Vértice	pai	d	f
A	-I	I	
B	-I A	2	
C	-I D	4	
D	-I B	3	
E	-I		

Ordem Topológica

Executando DFS 2 novamente

```
01  dfs_02(v){
02        cor[v] = laranja \\ processando
03        d[v] = ++tempo
04        para cada u adjacente de v {
05           se cor[u] == branco{
06                 pai[u] = v
07                 dfs_02(u)
08            }
09           se cor[u] == laranja {CICLO}
10        }
11        cor[v] = verde \\ finalizado
12        f [v] = tempo++
13        inserir v na fim da lista
14  }
```



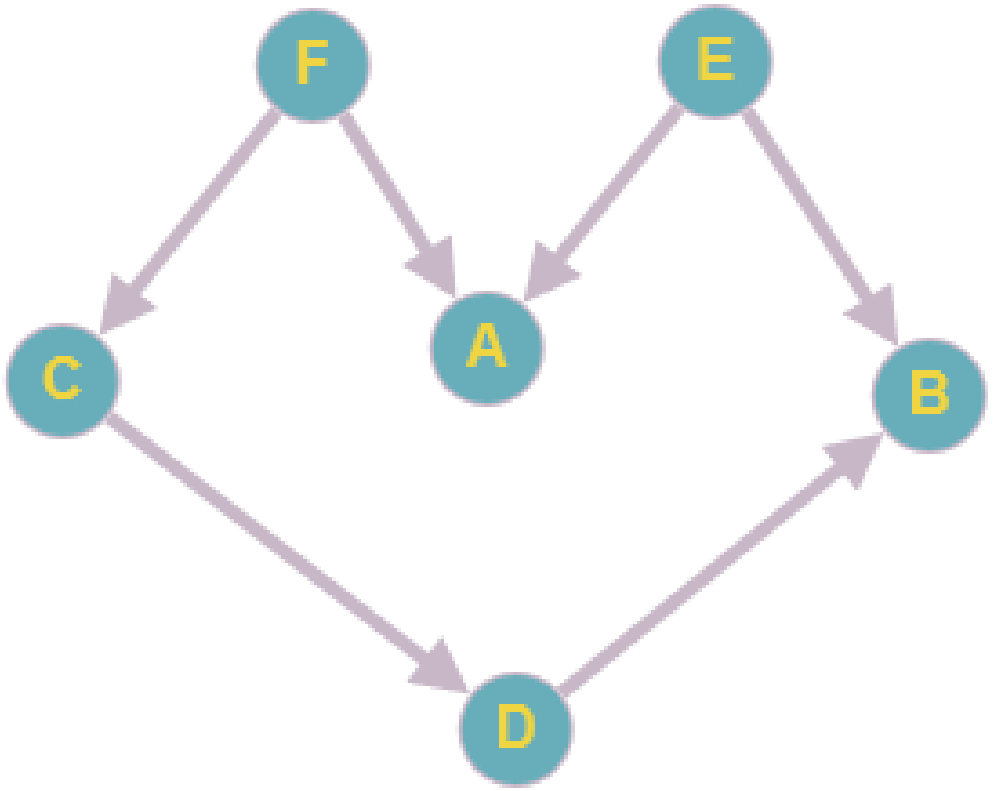
Vértice	pai	d	f
A	-I	I	
B	~~-I~~ A	2	
C	~~-I~~ D	4	
D	~~-I~~ B	3	
E	-I		

adjacente laranja, ciclo detectado

Ordem Topológica

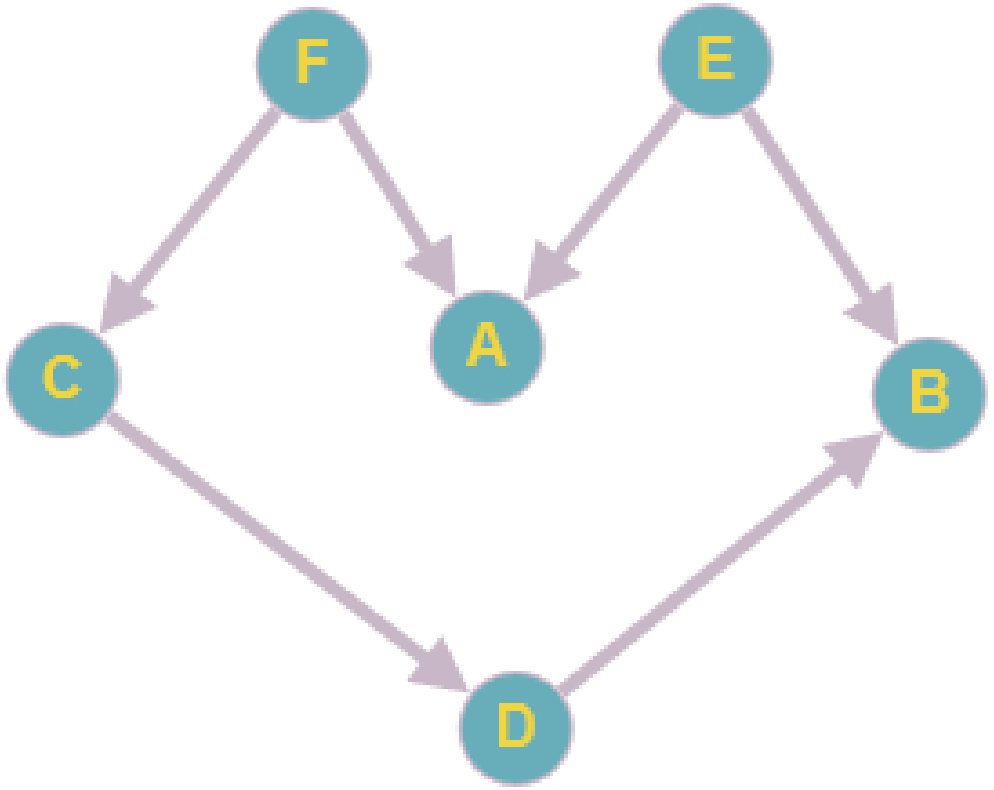
Encontrar todas as possibilidades de OT

```
01  dfs_02(v){
02      cor[v] = laranja \\ processando
03      d[v] = ++tempo
04      para cada u adjacente de v {
05          se cor[u] == branco{
06                  pai[u] = v
07                  dfs_02(u)
08           }
09          se cor[u] == laranja {CICLO}
10      }
11      cor[v] = verde \\ finalizado
12      f [v] = tempo++
13      inserir v na fim da lista
14  }
```



Encontrar todas as possibilidades de OT


```
01  dfs_02(v){
02      cor[v] = laranja \\ processando
03      d[v] = ++tempo
04      para cada u adjacente de v {
05          se cor[u] == branco{
06                  pai[u] = v
07                  dfs_02(u)
08           }
09          se cor[u] == laranja {CICLO}
10      }
11      cor[v] = verde \\ finalizado
12      f [v] = tempo++
13      inserir v na fim da lista
14  }
```



E F A C D B
E F C A D B
E F C D A B
E F C D B A
F C D E A B
F C D E B A
F C E A D B
F C E D A B
F C E D B A
F E A C D B
F E C A D B
F E C D A B
F E C D B A

Prof. Hélder Pereira Borges

helder@ifma.edu.br

Grafos
Ordenaçao Topologica

IFMA
Departamento de Computação