

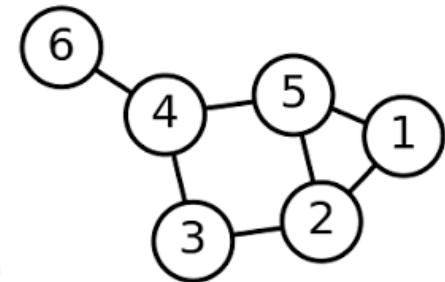
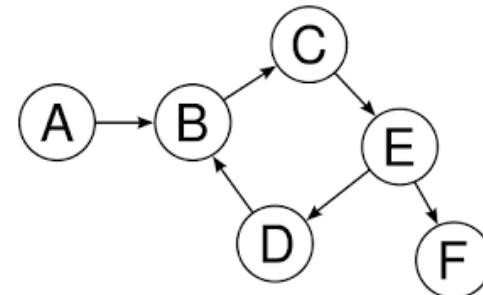
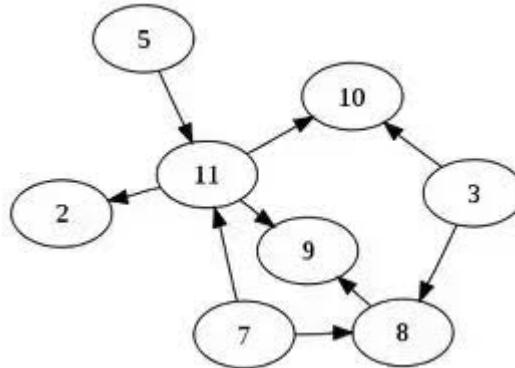
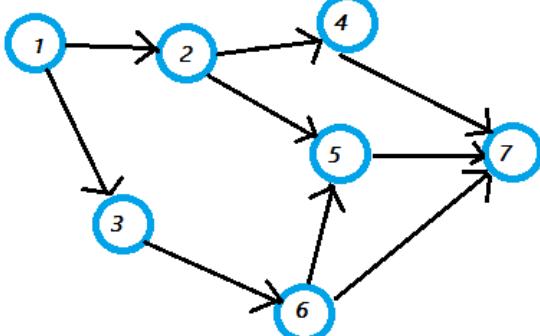
# Grafos – Aula 7

## Roteiro

- DAG
- Executando tarefas
- Ordenação topológica
- Algoritmo e melhorias

# DAG

- Grafo direcionado acíclico
  - Directed Acyclic Graph = DAG
  - importante classe de grafos
- Exemplos



Não são DAGs

São DAGs

- Toda árvore direcionada é um DAG
- Todo DAG possui ao menos um vértice com grau de entrada zero

# Executando Tarefas

- N tarefas precisam ser executadas
- Tarefas são dependentes
  - Tarefas tem “pré-requisitos” (como sua grade curricular)
- **Problema:** Qual ordem de execução não viola as dependências?



**Modelar problema utilizando  
grafos direcionados**

# Executando Tarefas

## ■ Exemplo com 5 tarefas

B depende de A

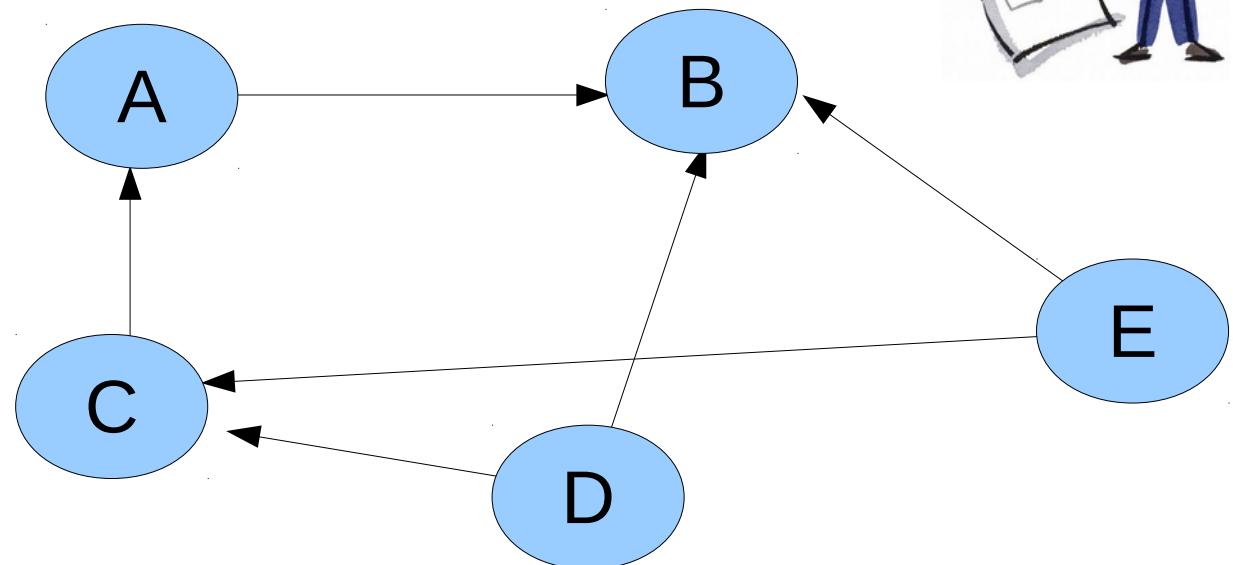
A depende de C

C depende de D

B depende de E

B depende de D

C depende de E



## ■ Qual é a ordem de execução?



# Tarefas e DAG

- Grafo de dependência é direcionado
- Tarefas podem ser executadas somente se grafo de dependência não possui ciclos
  - grafo de dependência precisar ser um DAG!
- Dado um DAG, como determinar a ordem de execução das tarefas?



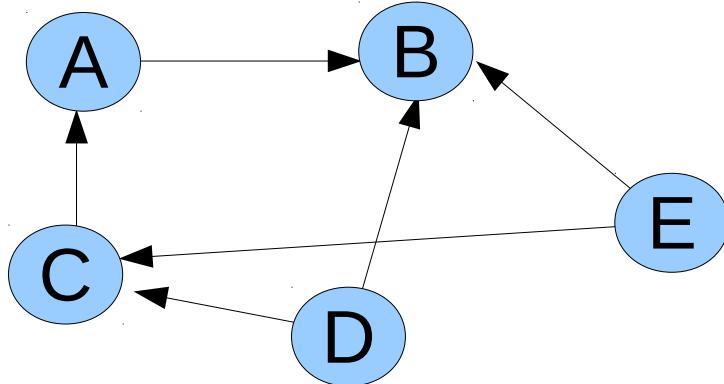
**Algoritmo (eficiente)!**

# Ordenação Topológica

- Ordenação dos vértices de forma que arestas “apontam” sempre para frente
- Dado grafo direcionado  $G$
- Ordenação dos vértices de  $G$ :  $v_1, v_2, \dots, v_n$ , tal que para toda aresta  $(v_i, v_j)$  temos que  $i < j$ 
  - $v_i$  corresponde a um vértice do grafo, para  $i = 1, 2, \dots, n$
  - toda aresta do grafo é mapeada para algum  $(v_i, v_j)$

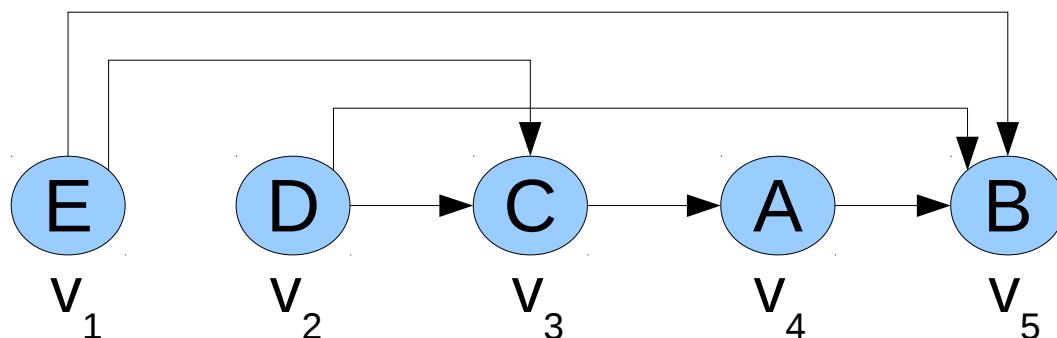
# Exemplo

- Ordenação dos vértices de  $G: v_1, v_2, \dots, v_n$ , tal que para toda aresta  $(v_i, v_j)$  temos que  $i < j$



Ordenação topológica:

$$\begin{aligned}v_1 &= E \\v_2 &= D \\v_3 &= C \\v_4 &= A \\v_5 &= B\end{aligned}$$



- Para toda aresta  $(v_i, v_j)$  temos que  $i < j$
- Ex:  $(E, C) = (v_1, v_3) \rightarrow 1 < 3$
- Ex.  $(D, B) = (v_2, v_5) \rightarrow 2 < 5$

# Ordenação Topológica

- Define uma ordem de execução das tarefas
  - podemos executar tarefas na ordem dada pela ordenação topológica
  - $v_i$  não depende de nenhuma tarefa  $v_j$  com  $j < i$
- **Problema:** Dado um DAG, como determinar uma ordenação topológica?

Idéias?

- **Ideia:** começar com vértices que tem grau de entrada zero
  - não dependem de ninguém

# Ordenação Topológica

## ■ Algoritmo

1.  $ordem = 0$

2. Enquanto  $|V| > 0$  faça

3.   Encontrar  $u$  com grau de entrada zero

4.    $ordem = ordem + u$

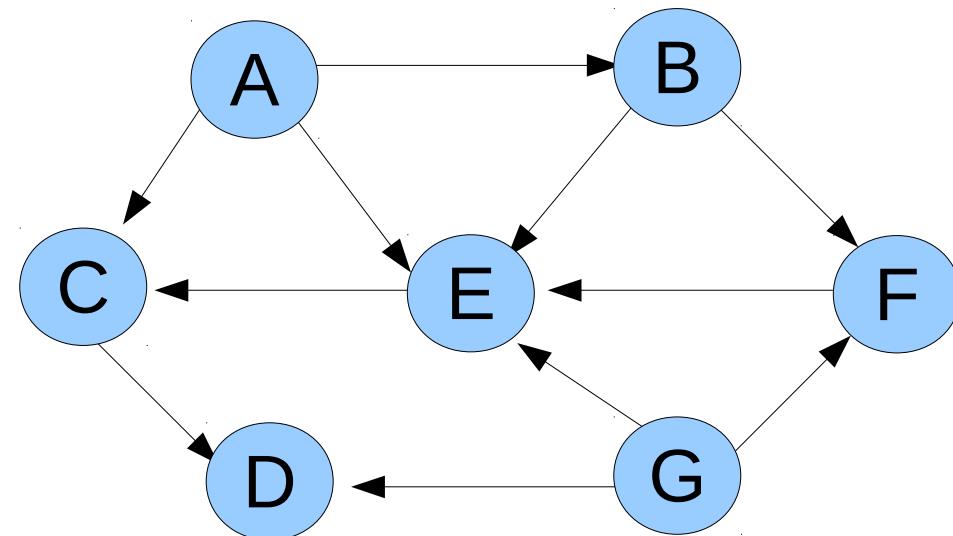
5.   Remover  $u$  do grafo  $G$

■ Remover  $u$  (grau de entrada zero) mantém  $G$  como um DAG

■ teremos outro grau de entrada zero no grafo

■ Algoritmo sempre termina com uma ordenação

# Exemplo



Passo	Grau entrada zero	Ordem
0	A,G	-
1	G,B	A
2	B	A,G
3	F	A,G,B
4	E	A,G,B,F
5	C	A,G,B,F,E
6	D	A,G,B,F,E,C
7	-	A,G,B,F,E,C,D

- Passo zero descobre vértices com grau de entrada zero
- Escolhe vértice de grau zero mais antigo primeiro
- Ordenação topológica não é única (algoritmo produz uma ordenação topológica)

# Ordenação Topológica

## ■ Complexidade?

1.  $ordem = 0$

2. Enquanto  $|V| > 0$  faça

3.     Encontrar  $u$  com grau de entrada zero

4.      $ordem = ordem + u$

5.     Remover  $u$  do grafo  $G$

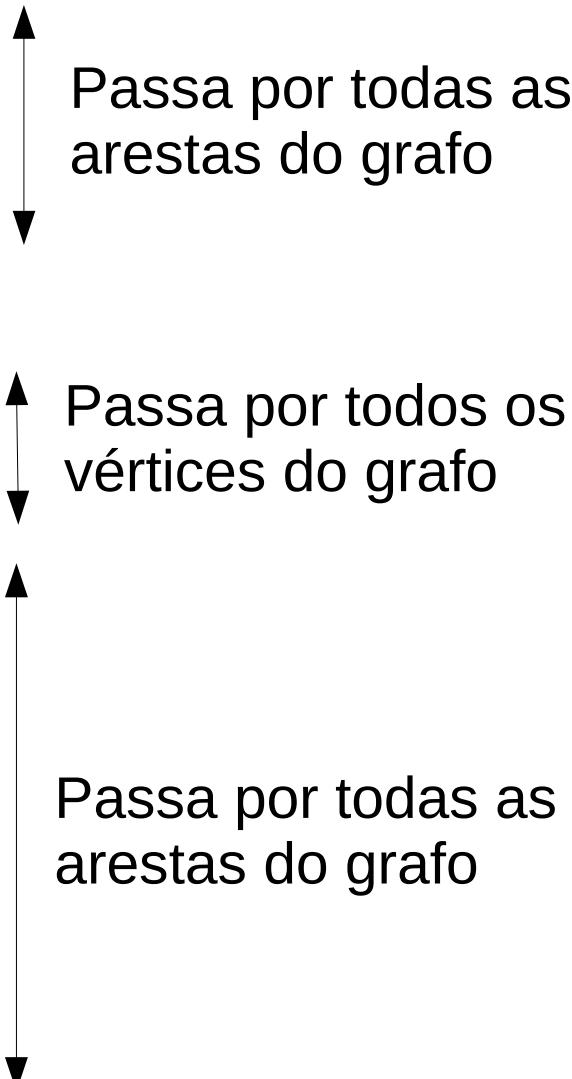
- Depende do tempo necessário para encontrar  $u$  (vértice de grau zero), linha 3
- Procurar sequencialmente tem custo  $O(n)$
- Complexidade total:  $O(n^2)$



# Ordenação Topológica

- Como melhorar complexidade?
- Manter vetor com grau de entrada dos vértices
- Manter lista de vértices com grau de entrada zero
- Atualizar vetor e lista a cada “remoção” de vértice
  - inserir na lista quando grau for zero
- Não precisa remover o vértice de grau zero do grafo (ou fazer qualquer alteração no grafo)
- Complexidade:  $O(m + n)$

# Algoritmo Melhorado

1. Inicializar  $g_e[u] = 0$  para todo  $u$
  2. Para cada vértice  $u$
  3.   Para cada aresta direcionada  $(u, v)$
  4.      $g_e[v] = g_e[v] + 1$
  5. lista = 0, ordem = 0
  6. Para cada vértice  $u$
  7.   Adicionar  $u$  na lista se  $g_e[u] = 0$
  8. Repetir
  9.   Remover  $u$  da lista
  10.   ordem = ordem + 1
  11.   Para cada aresta direcionada  $(u, v)$
  12.      $g_e[v] = g_e[v] - 1$
  13.     Adicionar  $v$  na lista se  $g_e[v] = 0$
  14. Até que lista = 0
  15. Retornar ordem
- 

■ Complexidade:  $O(m + n)$