Busca em Profundidade

Estrutura de Dados II

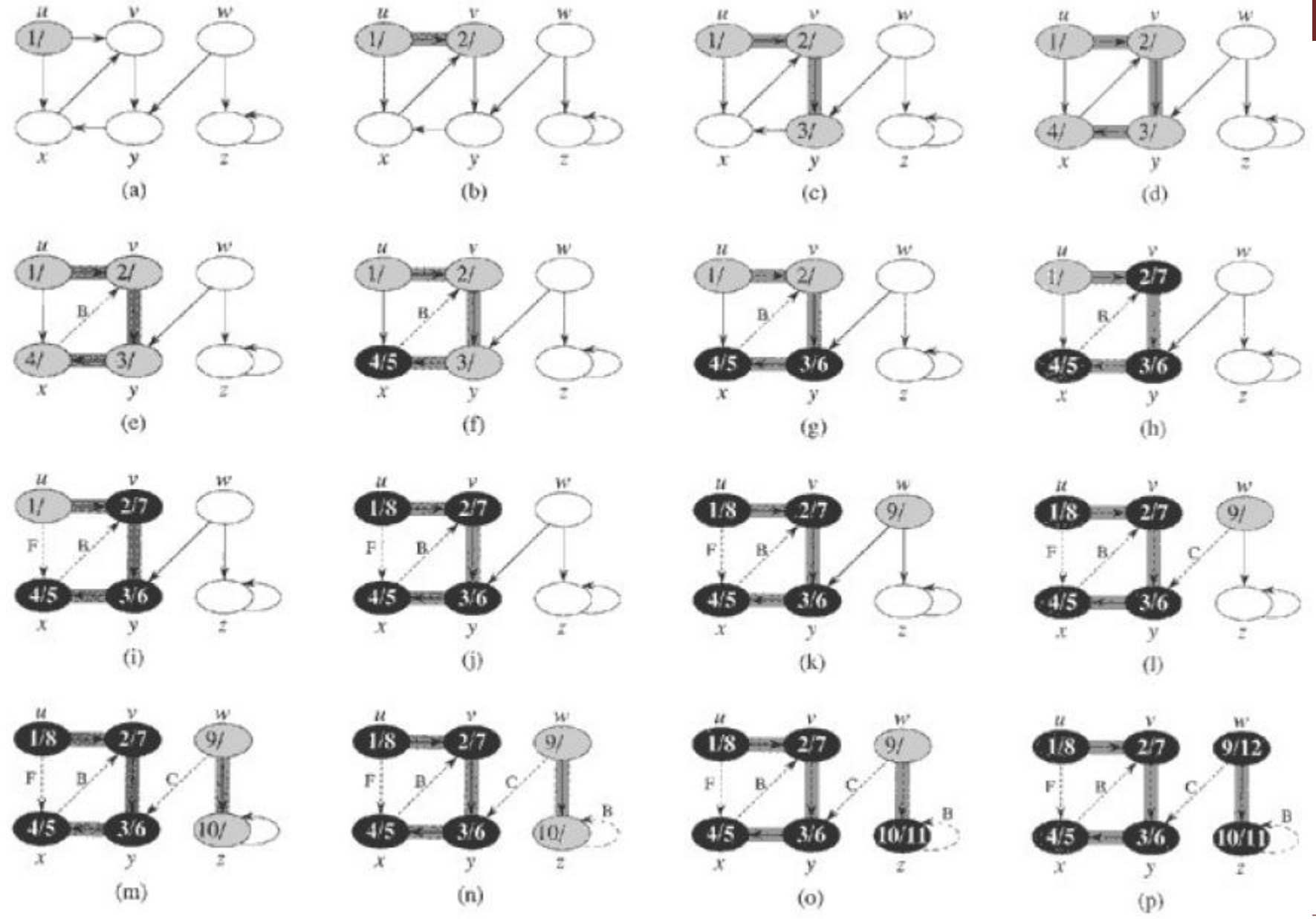
Busca em profundidade (DFS)

- A busca em profundidade, do inglês *depth-first search (DFS*), é um algoritmo para caminhar no grafo.
- A estratégia é buscar o mais profundo no grafo sempre que possível.
- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda possui arestas não exploradas saindo dele.
- Quando todas as arestas adjacentes a v tiverem sido exploradas a busca anda para trás para explorar vértices que saem do vértice do qual v foi descoberto.
- O algoritmo é a base para muitos outros algoritmos importantes, tais como verificação de grafos acíclicos, ordenação topológica e componentes fortemente conectados.

Busca em profundidade (DFS)

- Para acompanhar o progresso do algoritmo cada vértice é colorido de branco, cinza ou preto.
- Todos os vértices são inicializados branco.
- Quando um vértice é *descoberto* pela primeira vez ele torna-se cinza, e é tornado preto quando sua lista de adjacentes tenha sido completamente examinada.
- d[v]: tempo de descoberta
- f[v]: tempo de término do exame da lista de adjacentes de v.
- Estes registros são inteiros entre 1 e 2V pois existe um evento de descoberta e um evento de término para cada um dos V vértices.

Funcionamento



Busca em profundidade (DFS)

- **Algoritmo: versão não recursivo**

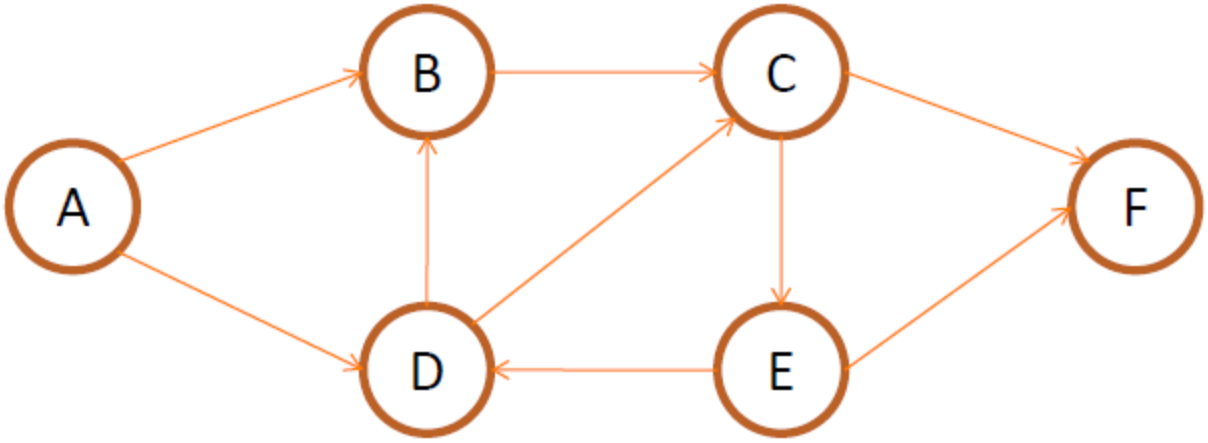
```
BUSCA-EM-PROFUNDIDADE (n, Adj, r)
 1  para u ← 1 até n faça
 2      cor[u] ← branco
 3  cor[r] ← cinza
 4  P ← CRIA-PILHA (r)
 5  enquanto P não estiver vazia faça
 6      u ← COPIA-TOPO-DA-PILHA (P)
 7      v ← PRÓXIMO (Adj[u])
 8      se v ≠ NIL
 9          então se cor[v] = branco
10                  então cor[v] ← cinza
11                        COLOCA-NA-PILHA (v, P)
12          senão cor[u] ← preto
13                TIRA-DA-PILHA (P)
14  devolva cor[1..n]
```

Busca em profundidade (DFS)

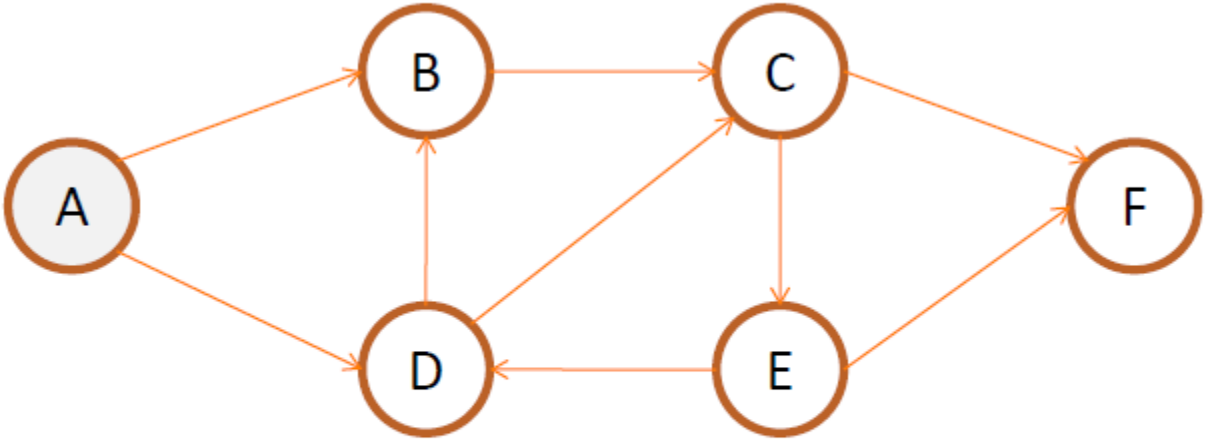
```
DFS(V, E)
for each u ∈ V
    do color[u] ← WHITE
time ← 0
for each u ∈ V
    do if color[u] = WHITE
        then DFS-VISIT(u)

DFS-VISIT(u)
color[u] ← GRAY          ▷ discover u
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]      ▷ explore (u, v)
    do if color[v] = WHITE
        then DFS-VISIT(v)
color[u] ← BLACK
time ← time + 1
f[u] ← time              ▷ finish u
```

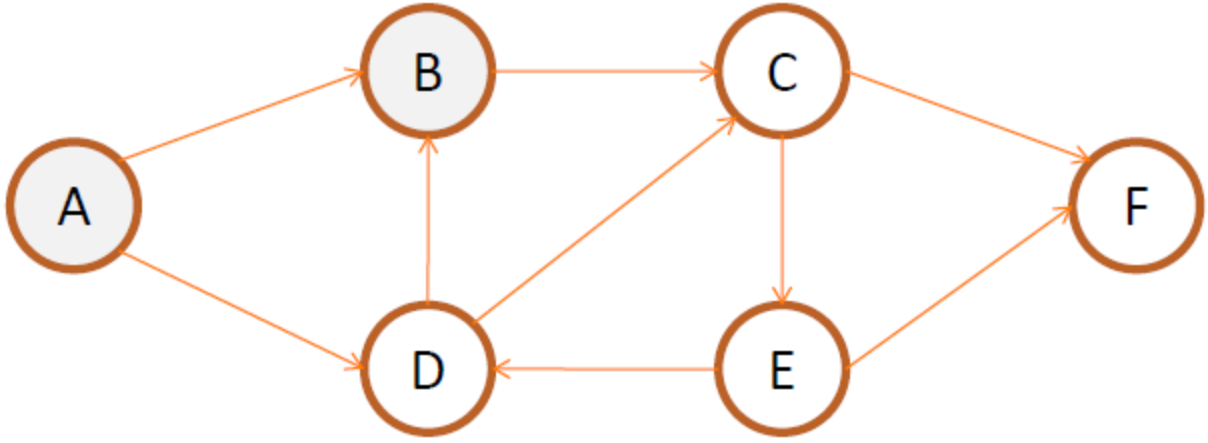
Exemplo



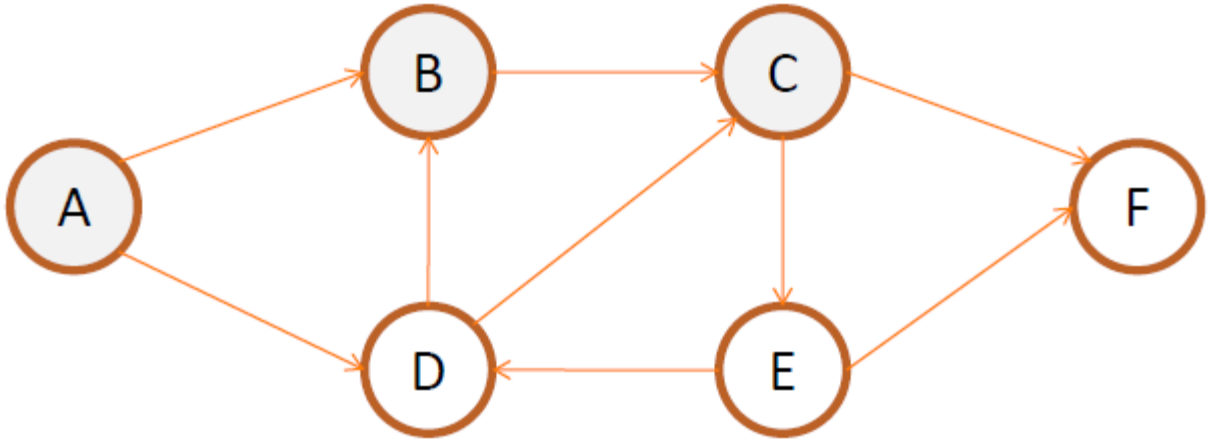
Exemplo



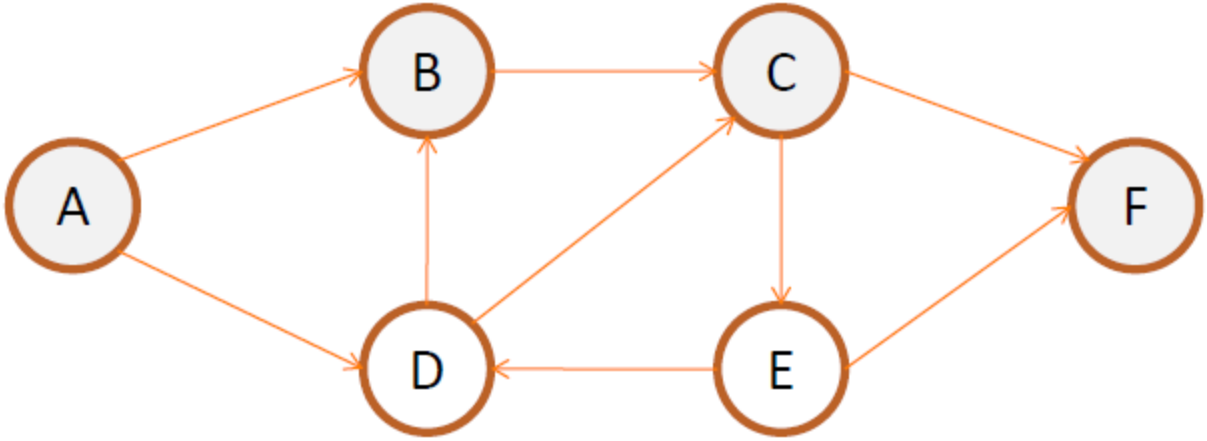
Exemplo



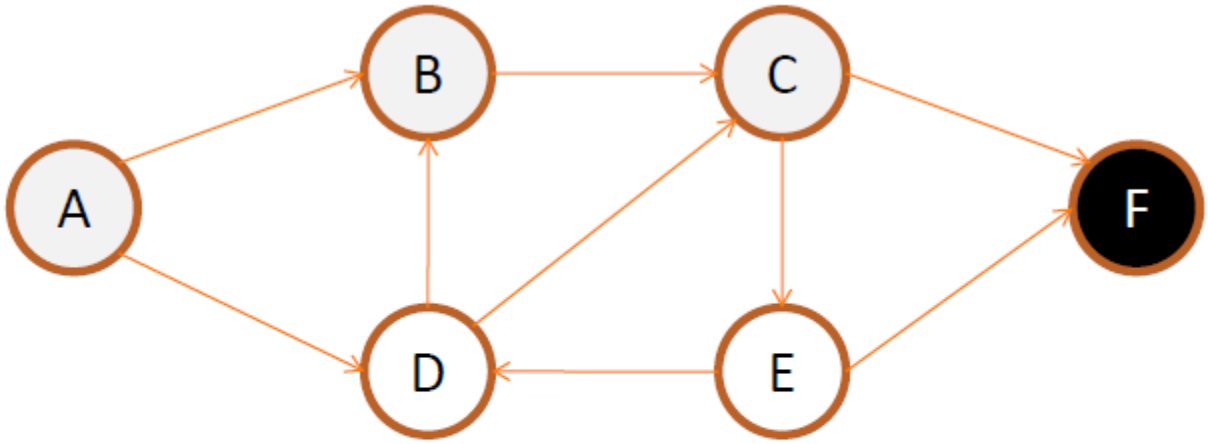
Exemplo



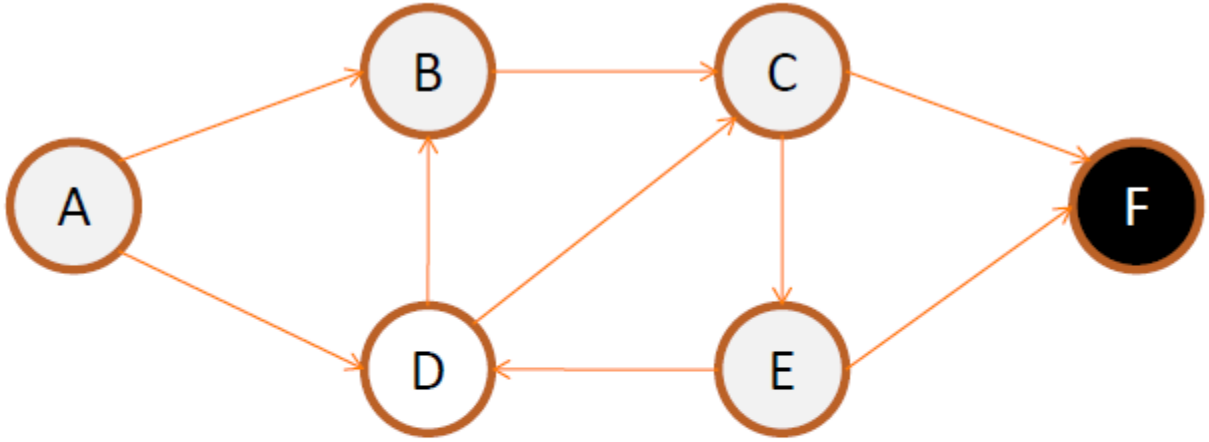
Exemplo



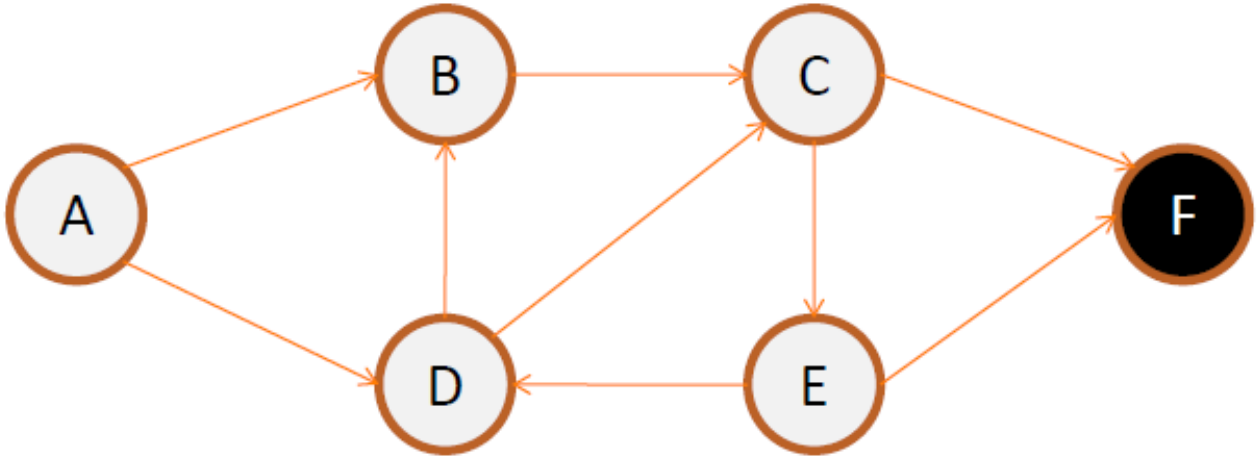
Exemplo



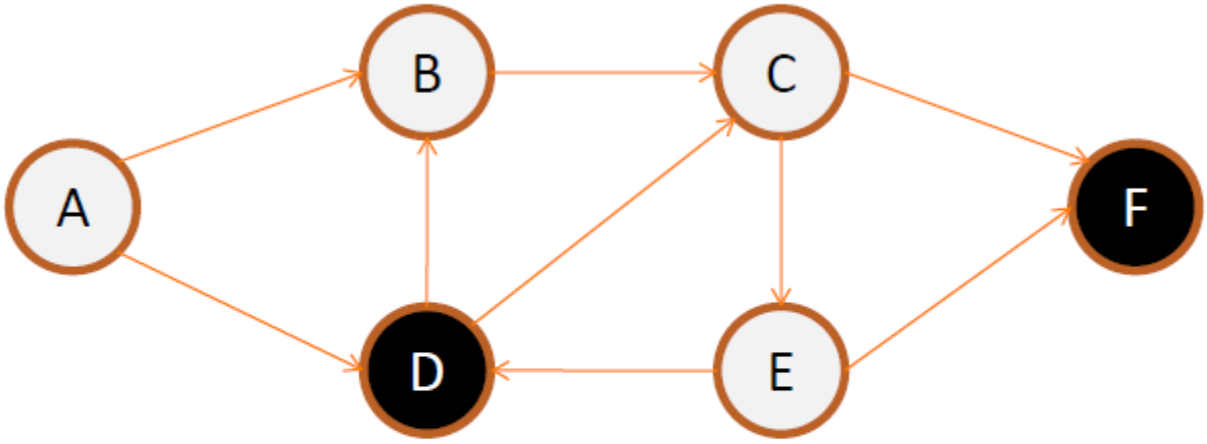
Exemplo



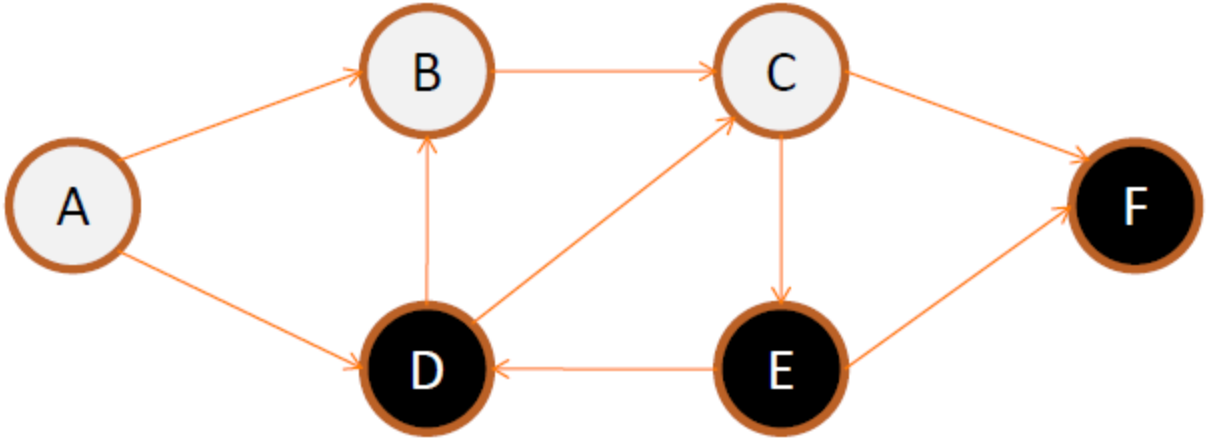
Exemplo



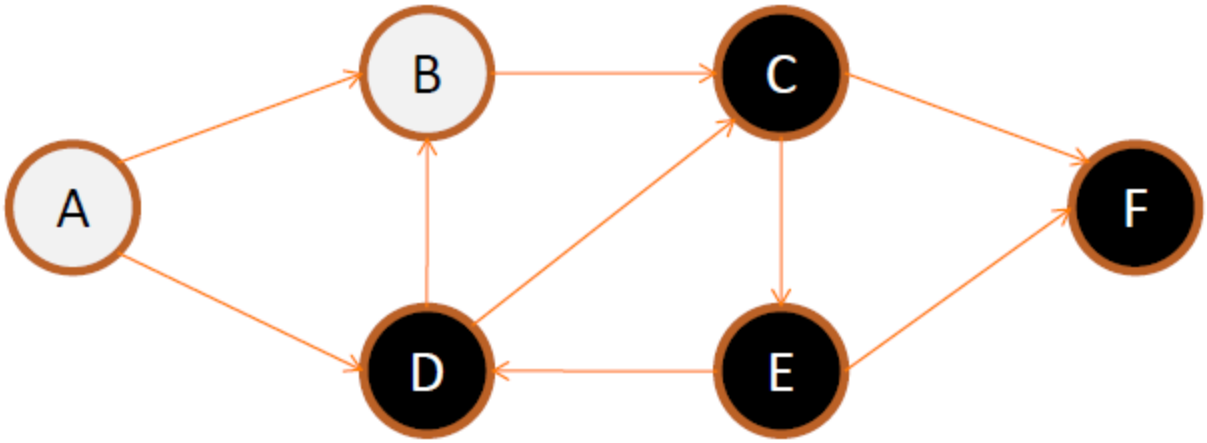
Exemplo



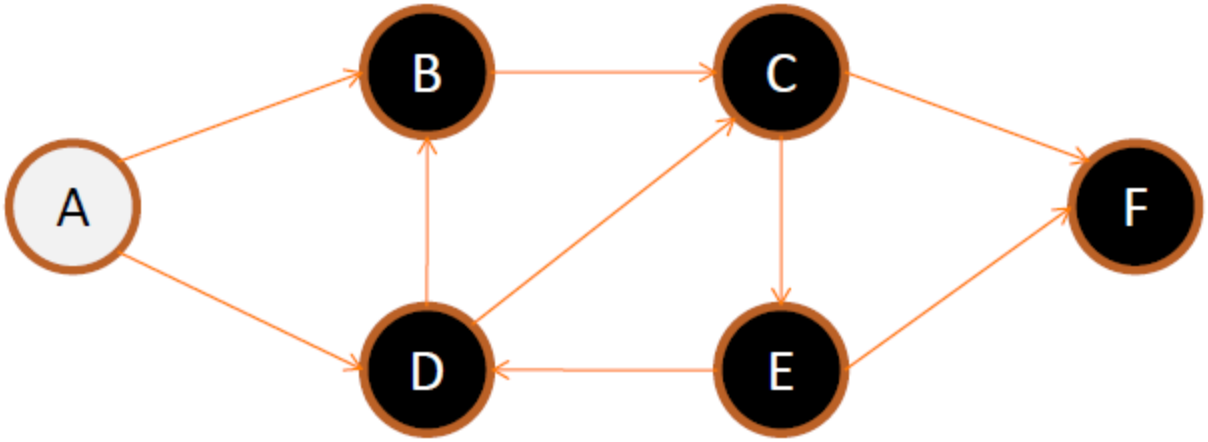
Exemplo



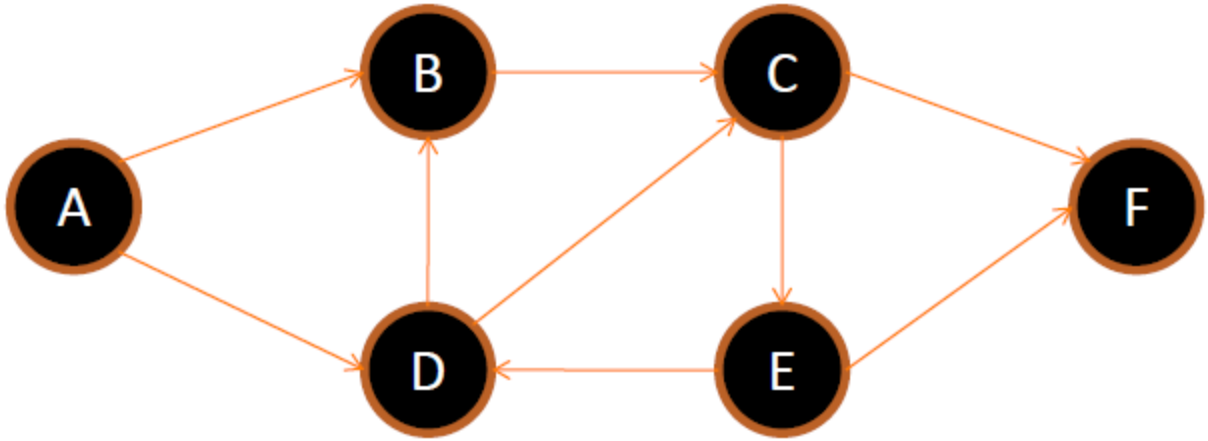
Exemplo



Exemplo



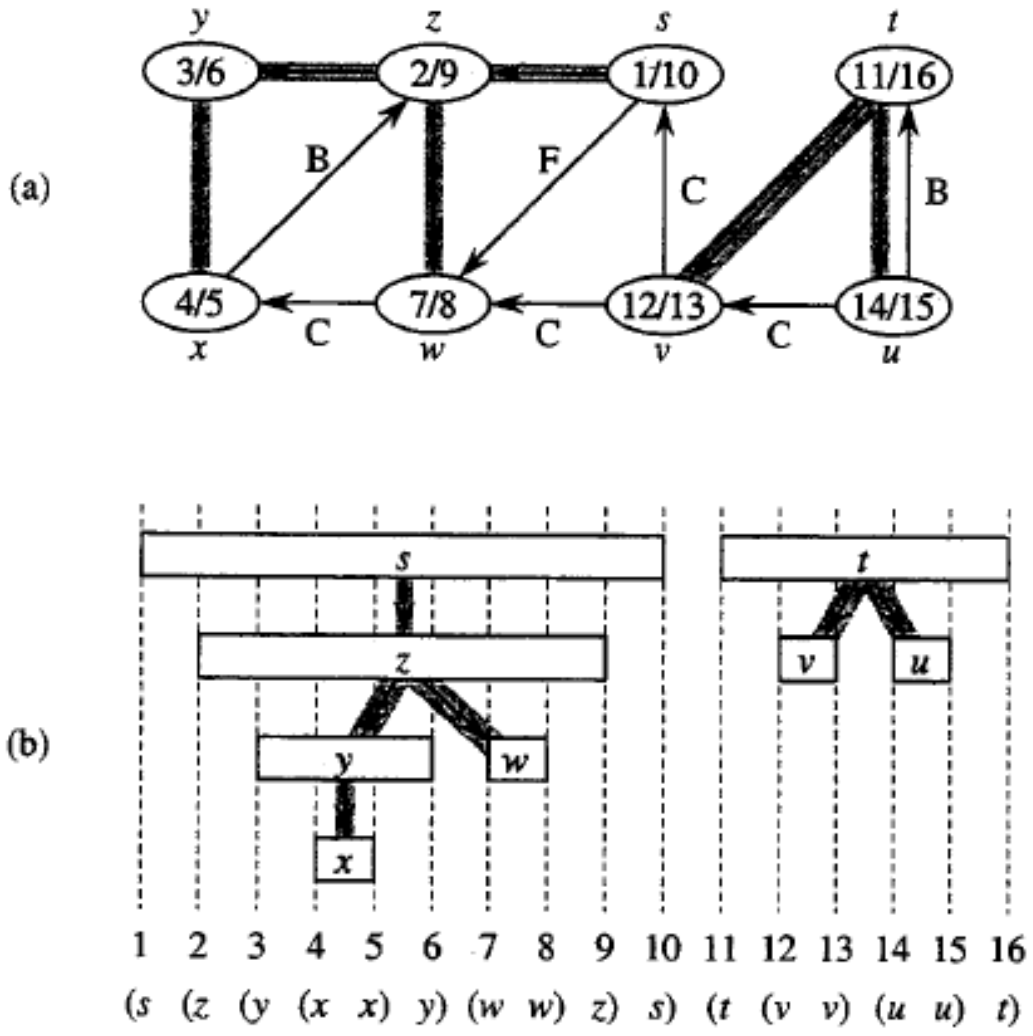
Exemplo



Propriedades da DFS

- **Subgrafo predecessor forma uma floresta**
  - Cada árvore dessa floresta mostra as componentes conectadas do grafo
- **Os tempos de descoberta e de termino tem estrutura de parêntesis**
  - Nós estão aninhados por quantidade de arestas de chegada a cada um
  - A história de descobertas e termino irá gerar uma expressão bem formada, no sentido de que os parênteses serão aninhados de modo apropriado

Propriedades – teorema do parênteses

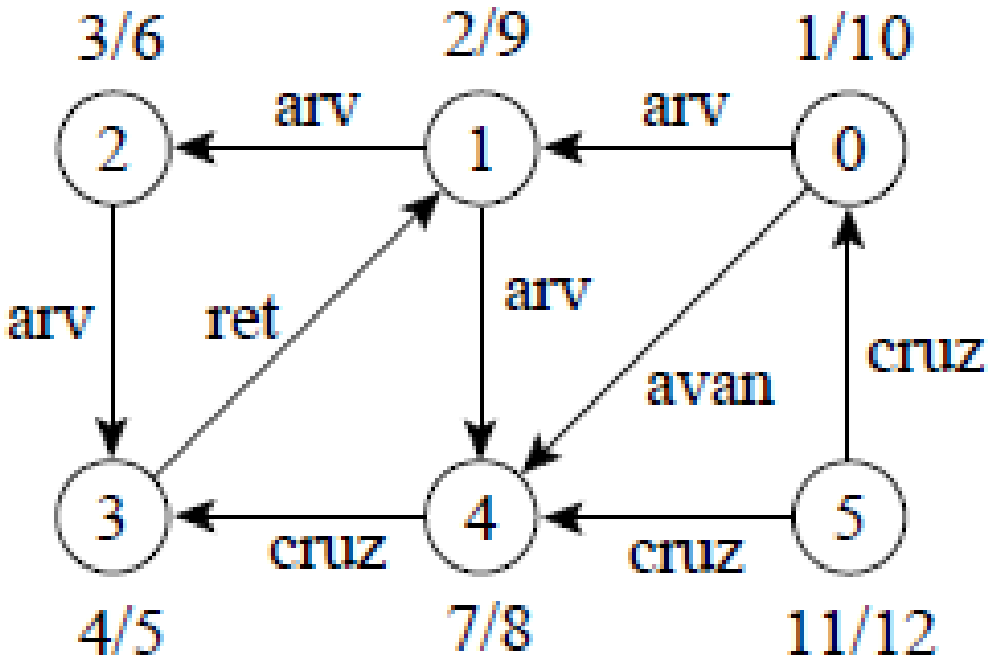


Propriedades

- **Classificação de Arestas:**
  - Arestas de **árvore**:
    - A aresta (u,v) é uma aresta de árvore se v foi descoberto primeiro pela exploração da aresta (u,v)
  - Arestas de **retorno**
    - São as arestas (u,v) que conectam um vertice u a um ancestral v em uma árvore primeiro na profundidade. Autoloops são considerados arestas de retorno
  - Arestas de **Avanço**
    - não pertencem à árvore de busca em profundidade mas conectam um vértice a um descendente que pertence à árvore de busca em profundidade.
  - Arestas **Cruzadas**
    - podem conectar vértices na mesma árvore de busca em profundidade, ou em duas árvores diferentes.

Classificação de Arestas

- **Pode ser classificada pela cor do vértice que é alcançado pela primeira vez:**
  - Branco indica uma aresta de árvore.
  - Cinza indica uma aresta de retorno.
  - Preto indica uma aresta de avanço quando
    - u é descoberto antes de v ou uma aresta de cruzamento caso contrário.



Teste para Verificar se Grafo é Acíclico

- A busca em profundidade pode ser usada para verificar se um grafo é acíclico ou contém um ou mais ciclos.
- Se uma aresta de retorno é encontrada durante a busca em profundidade em G, então o grafo tem ciclo.
- Um grafo direcionado G é acíclico se e somente se a busca em profundidade em G não apresentar arestas de retorno.

Comparação DFS x BFS

- DFS requer menos memória que BFS
  - Não precisa carregar todos os ponteiros para os filhos em cada nível
- DFS e melhor que BFS:
  - Dependo do problema a resolver
  - BFS visita cada nível um por vez. Se conhecemos o que estamos buscando e sabemos que está em baixa profundidade, a BFS é melhor. Se a solução é a máxima profundidade DFS é melhor.

Ordenação Topológica

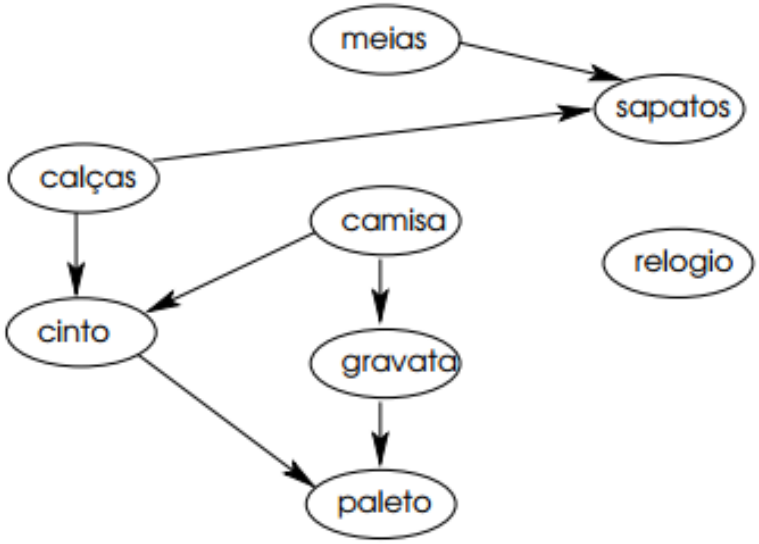
- A Ordenação Topológica de um grafo orientado acíclico G = (V, E) é uma ordenação linear de todos os seus vértices tal que se G contém uma aresta (u, v) então u aparece antes de v.

- Uma ordenação topológica de um grafo fornece a ordem em que as atividades devem ser processadas

- A ordem topológica de um grafo pode ser vista como uma ordenação de seus vértices ao longo de uma linha horizontal dado que todas as arestas direcionadas vão da esquerda para a direita

Ordenação Topológica

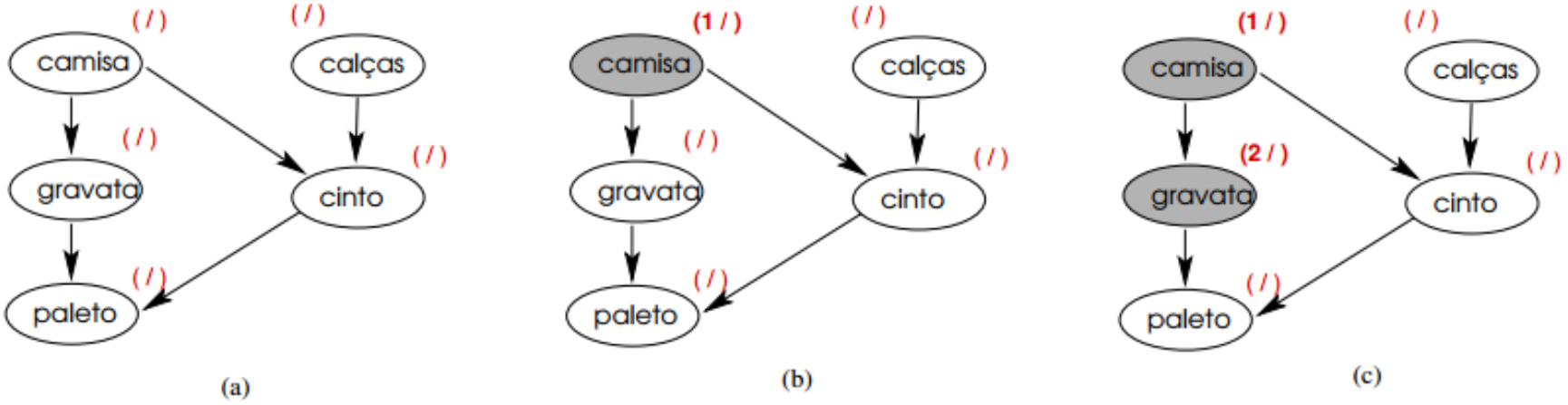
Os grafos orientados acíclicos são usados para indicar **precedências** entre eventos. Uma aresta direcionada num grafo orientado acíclico indica que a atividade *u* deve ser realizada antes da *v*.



Ordenação Topológica

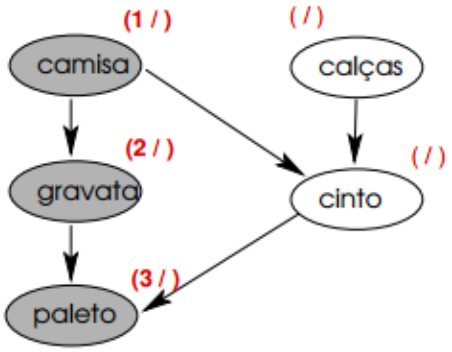
- **Aplicação do algoritmo de Busca em Profundidade**

Aplicação do algoritmo de Busca em Profundidade para a marcação dos tempos de descoberta e término em cada vértice de um trecho do exemplo anterior:

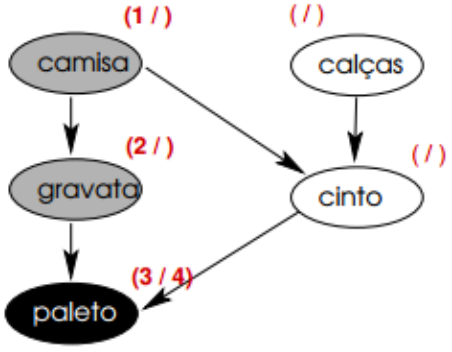


Ordenação Topológica

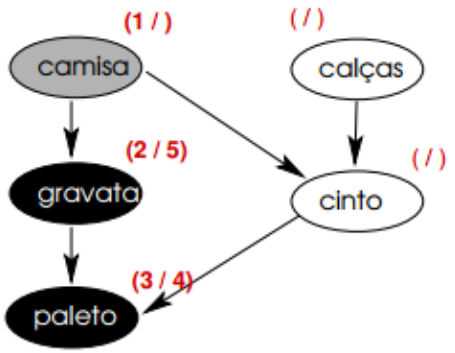
- **Aplicação do algoritmo de Busca em Profundidade**



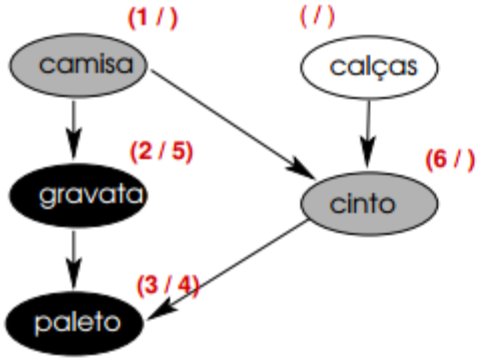
(d)



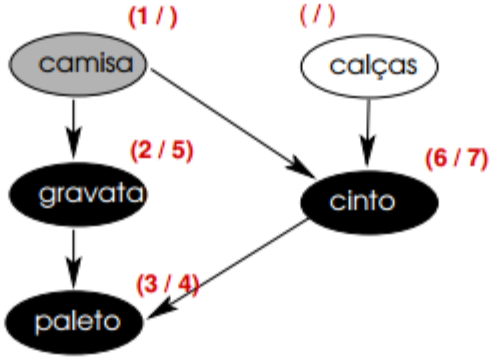
(e)



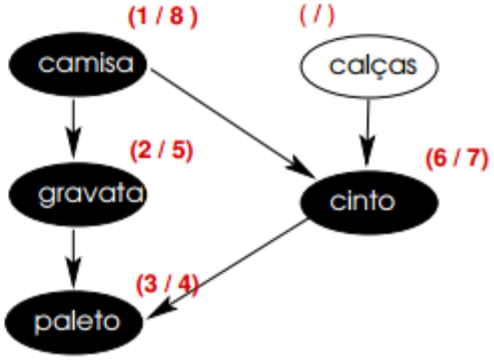
(f)



(g)

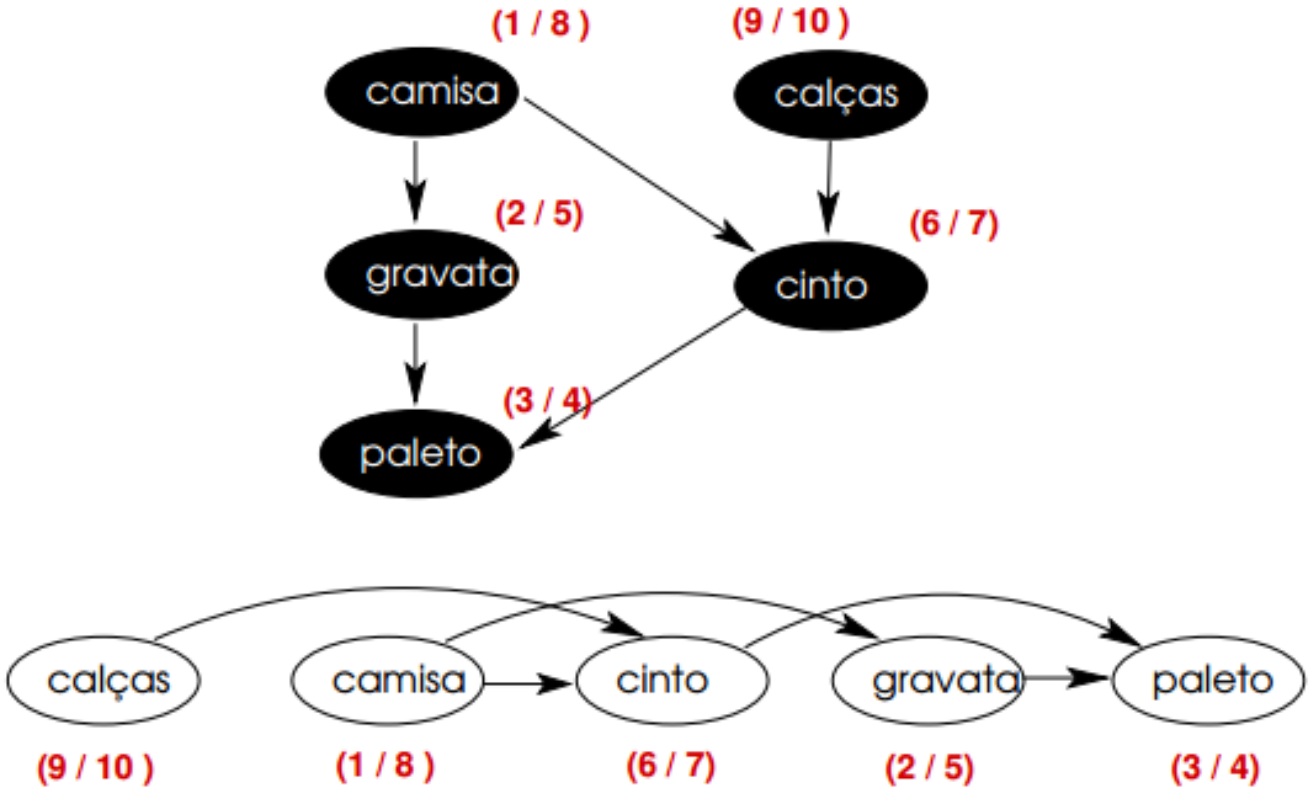


(h)



(i)

Ordenação Topológica



O grafo com ordenação topológica apresenta seus vértices organizados da esquerda para a direita, em ordem de tempo de término decrescente.

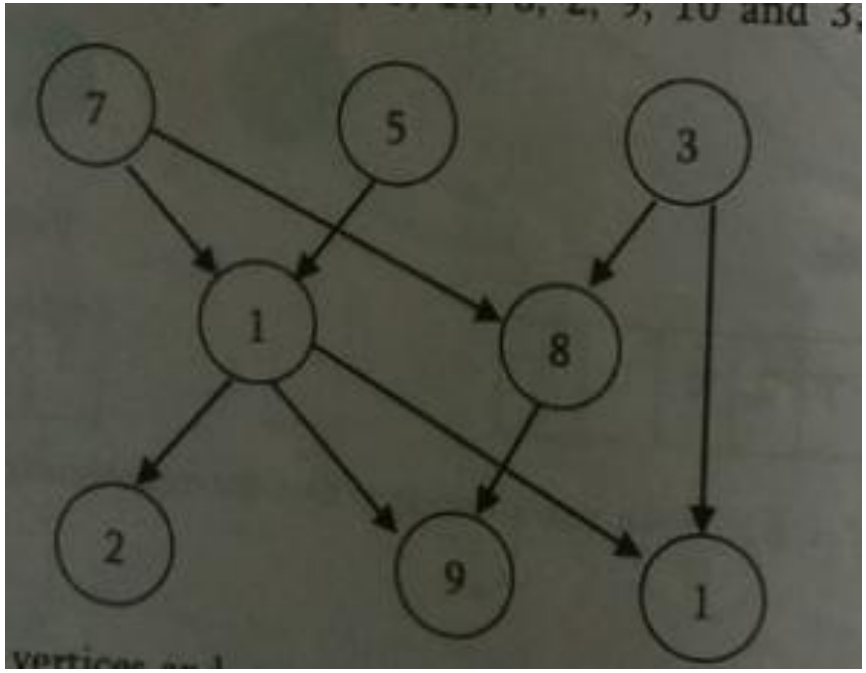
Algoritmo de Ordenação Topológica

O algoritmo simples a seguir ordena topologicamente um grafo acíclico orientado utilizando o algoritmo de Busca em Profundidade:

```
OrdenacaoTopologica(G)
1  Chamar BuscaEmProfundidade(G) para calcular o tempo
   de término t[v] para cada vértice v
2  A medida que cada vértice é terminado, inserir o
   vértice à frente de uma lista ligada
3  Retornar a lista ligada de vértices
Fim
```

Exemplo

- **Pode ter duas ou mais ordenações topológica:**
  - 7, 5,3, 1, 8, 2, 9, 1
  - 3, 5, 7, 8, 1, 1, 9, 2



Custo

- **O Custo O(V + E), uma vez que a busca em profundidade tem complexidade de tempo O(V + E) e o custo para inserir cada um dos V vértices na frente da lista linear encadeada custa O(1).**

Aplicações de Ordenação Topológica

- Representar pré-requisitos de cursos
- Detectar deadlocks
- Pipeline das tarefas de computação
- Avaliar fórmula em planilha

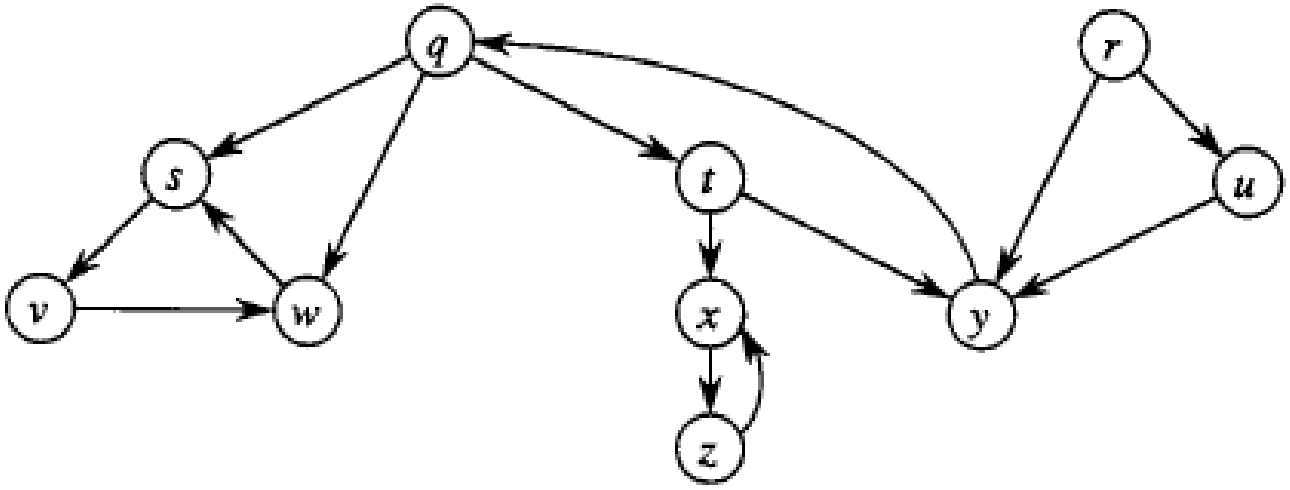
Exercício

22.3-2

Mostre como a busca em profundidade funciona sobre o grafo da Figura 22.6. Suponha que o loop **for** das linhas 5 a 7 do procedimento DFS considere os vértices em ordem alfabética, e suponha que cada lista de adjacências esteja em ordem alfabética. Mostre os tempos de descoberta e término para cada vértice, e mostre também a classificação de cada aresta.

22.3-3

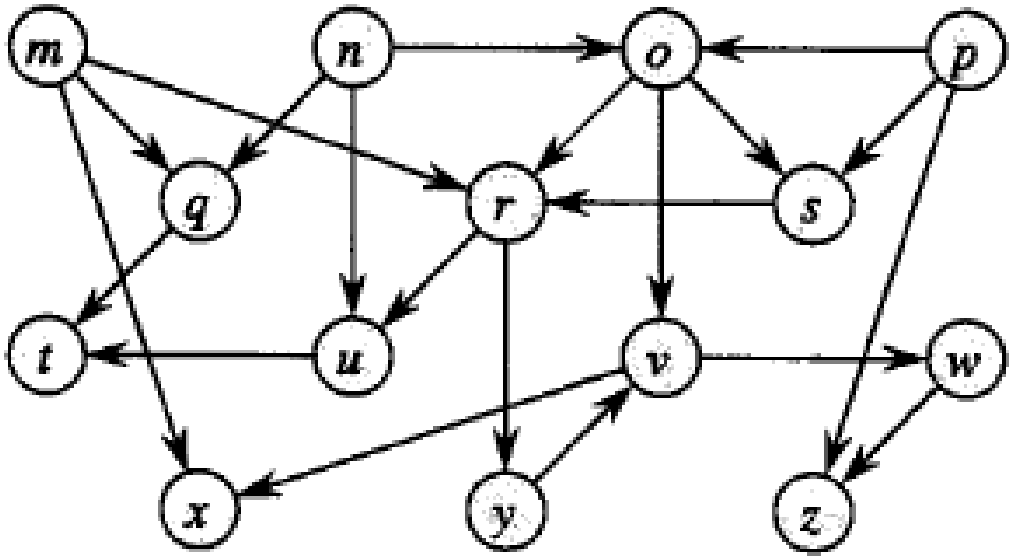
Mostre a estrutura de parênteses da busca em profundidade apresentada na Figura 22.4.



Exercício

22.4-1

Mostre a ordenação de vértices produzida por TOPOLOGICAL-SORT quando ele é executado sobre o gao na Figura 22.8, sob a hipótese do Exercício 22.3-2.



Referencias

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus, 2002
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Cengage Learning, 2004.
- Notas de aula. Prof. Rafael Fernandes DAI/IFMA
- **http://www.facom.ufu.br/~madriana/EBD/Didatica.pdf**