TÉCNICO LISBOA

Análise e Síntese de Algoritmos

DFS. Topological sort.

CLRS Cap. 22

Instituto Superior Técnico

2022/2023

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica [CLRS, Cap.15]
  - Algoritmos greedy [CLRS, Cap.16]
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Fluxos máximos [CLRS, Cap.26]
  - Árvores abrangentes [CLRS, Cap.23]
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais
  - Emparelhamento de Cadeias de Caracteres [CLRS, Cap.32]
  - Complexidade Computacional [CLRS, Cap.34]

Resumo

DFS - Procura em profundidade primeiro

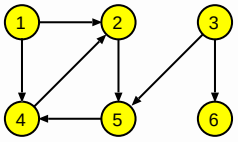
Ordenação Topológica

Grafos

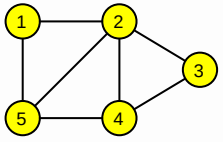
Grafo $G = (V, E)$ é definido por um conjunto V de vértices e um conjunto E de arcos

- Arcos representam ligações entre pares de vértices: $E \subseteq V \times V$
  - Se $|E| << |V \times V|$, o grafo diz-se esparso
  - Caso contrário, diz-se denso
- Grafos podem ser ou não dirigidos
  - Existência (ou não) da noção de direção nos arcos

Grafo Dirigido



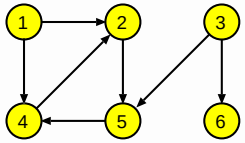
Grafo Não Dirigido



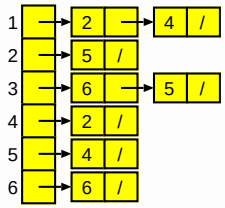
Representação dos arcos

- Matriz de adjacências: arcos representados por matriz
  → para grafos densos
- Listas de adjacências: arcos representados por listas
  → para grafos esparsos

Grafo Dirigido



Listas de Adjacências



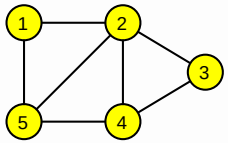
Matriz de Adjacências

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

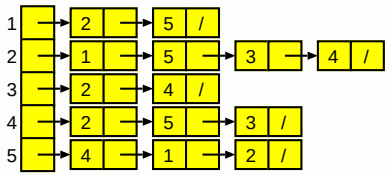
Representação dos arcos

- Matriz de adjacências: arcos representados por matriz
  → para grafos densos
- Listas de adjacências: arcos representados por listas
  → para grafos esparsos

Grafo Não Dirigido



Lista de Adjacências



Matriz de Adjacências

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Matriz de Adjacências

- $\Theta(V^2)$ para qualquer grafo

Listas de adjacências

- Tamanho das listas é $|E|$ para grafos dirigidos
- Tamanho das listas é $2|E|$ para grafos não dirigidos
- Tamanho total das listas de adjacências é $\Theta(V + E)$

Grafos pesados

- Existência de uma função de pesos $\omega : E \rightarrow IR$
- Função de pesos ω associa um peso a cada arco $(u, v) \in E$
- Pesos guardados nas listas de adjacências ou matriz de adjacências

Questões

- Qual a representação mais adequada para um grafo denso?
- E se a operação mais frequente for ler o peso dos arcos?
- E se quiser representar o grafo da World Wide Web?

Intuição

Grafo pesquisado dando prioridade aos arcos dos vértices visitados mais recentemente

Aplicações

- Resolução de labirintos
- Detecção de ciclos
- Ordenação topológica
- Testar se um grafo é bipartido
- Descobrir componentes fortemente ligados/conexos

Implementação

- $d[v]$: tempo de início (de visita do vértice)
- $f[v]$: tempo de fim (de visita do vértice)
- $\pi[v]$: predecessor de v na árvore DF
- $color[v]$: cor do vértice v: branco/cinzento/preto

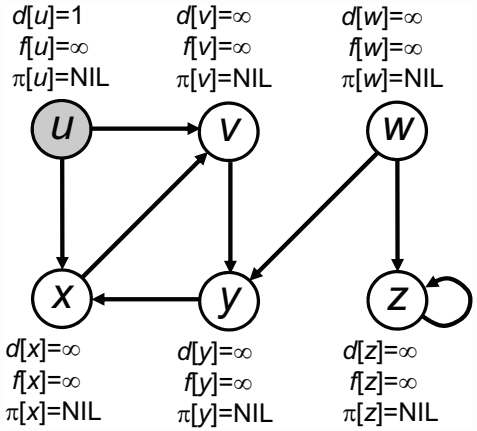
DFS(G)

```
for u ∈ G.V do
    color[u] ← white
    d[u] ← ∞
    f[u] ← ∞
    π[u] ← NIL
end for
time ← 1
for u ∈ G.V do
    if color[u] == white then
        DFS-Visit(G, u)
    end if
end for
```

DFS-Visit(G,u)

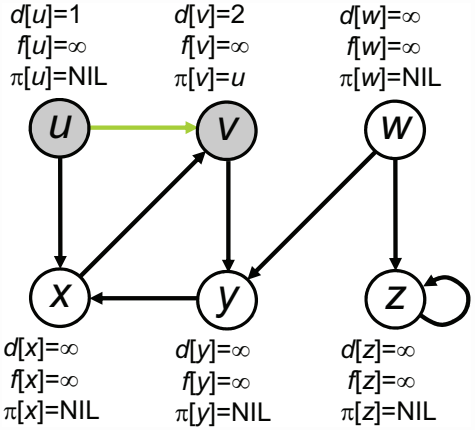
```
color[u] ← gray
d[u] ← time
time ← time + 1
for v ∈ G.Adj[u] do
    if color[v] == white then
        π[v] ← u
        DFS-Visit(G, v)
    end if
end for
color[u] ← black
f[u] ← time
time ← time + 1
```


Exemplo



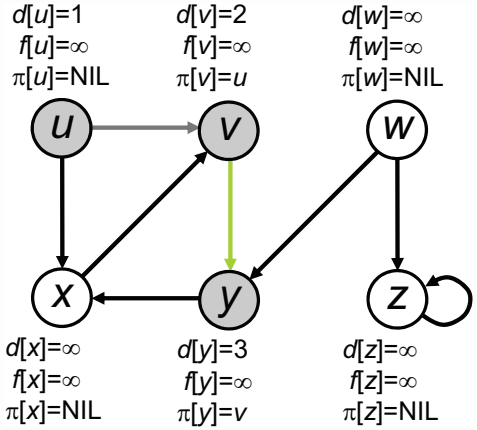
Procura em profundidade primeiro (DFS)

Exemplo



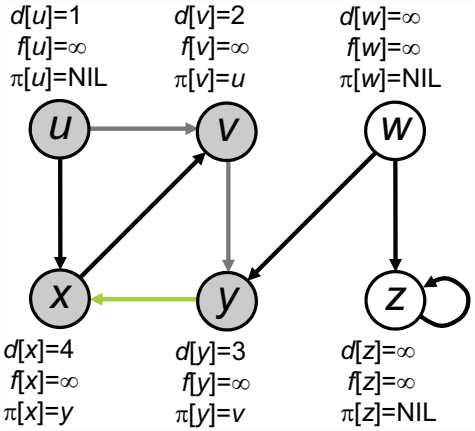
Procura em profundidade primeiro (DFS)

Exemplo



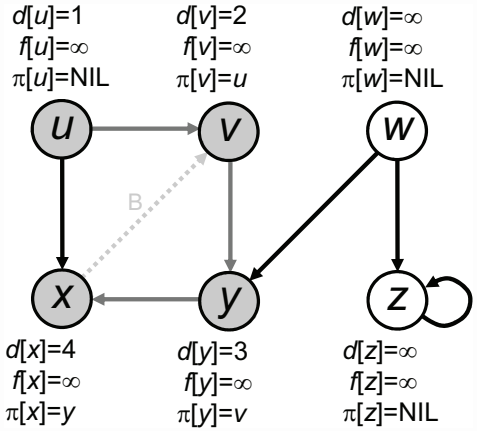
Procura em profundidade primeiro (DFS)

Exemplo



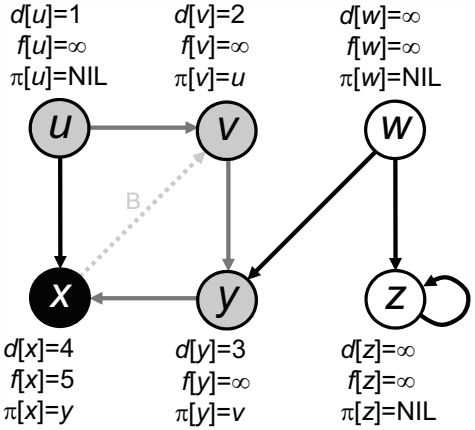
Procura em profundidade primeiro (DFS)

Exemplo



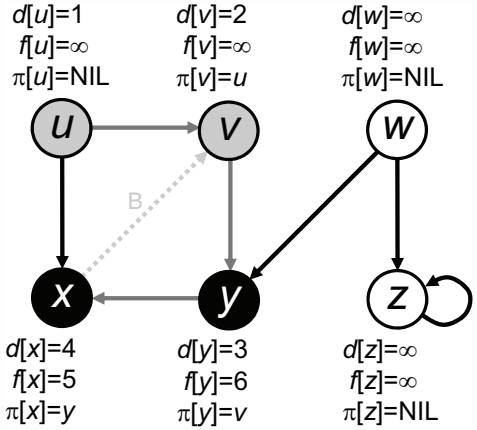
Procura em profundidade primeiro (DFS)

Exemplo



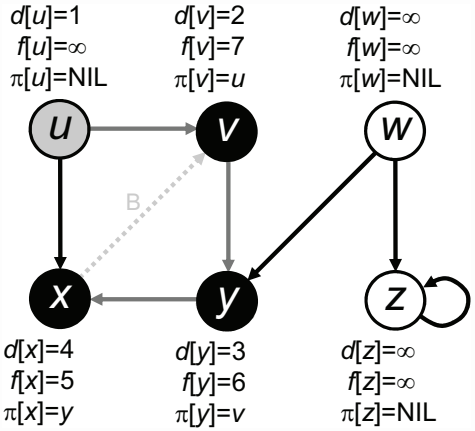
Procura em profundidade primeiro (DFS)

Exemplo



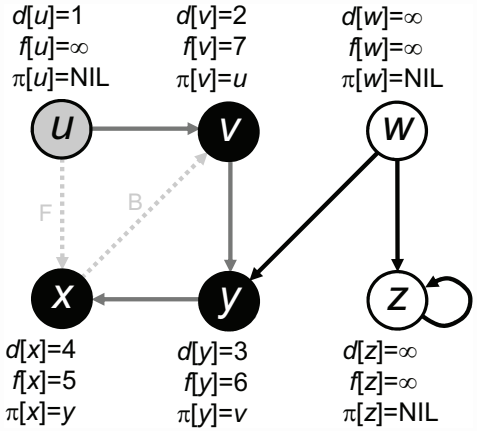
Procura em profundidade primeiro (DFS)

Exemplo



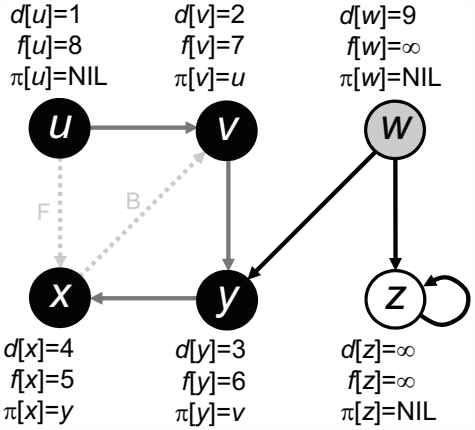
Procura em profundidade primeiro (DFS)

Exemplo



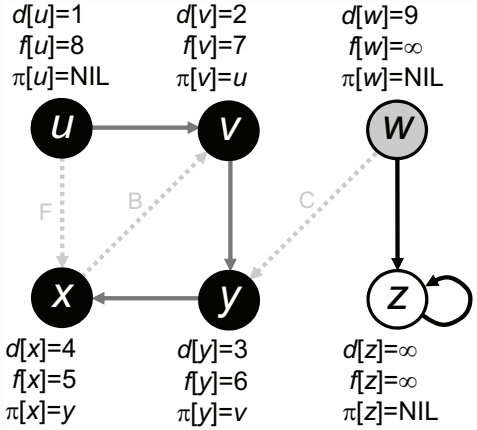
Procura em profundidade primeiro (DFS)

Exemplo



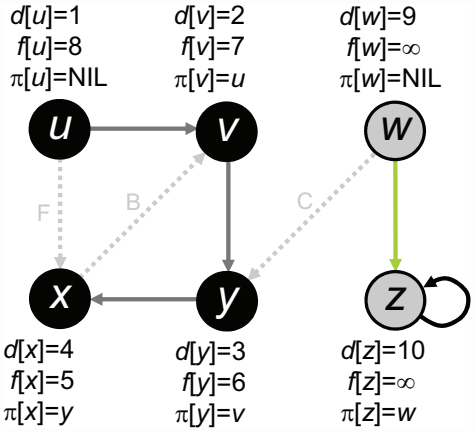
Procura em profundidade primeiro (DFS)

Exemplo



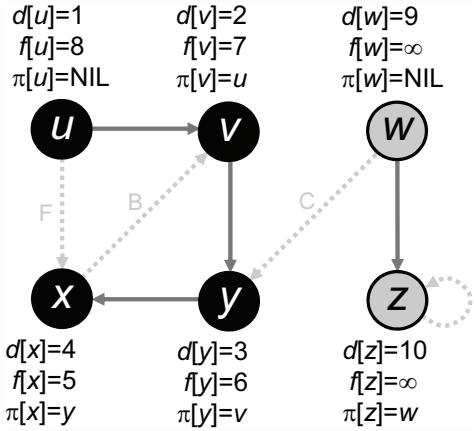
Procura em profundidade primeiro (DFS)

Exemplo

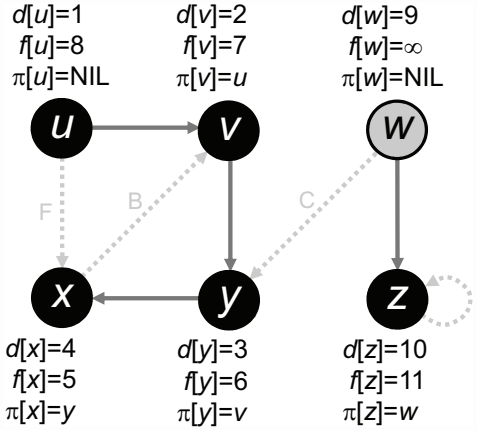


Procura em profundidade primeiro (DFS)

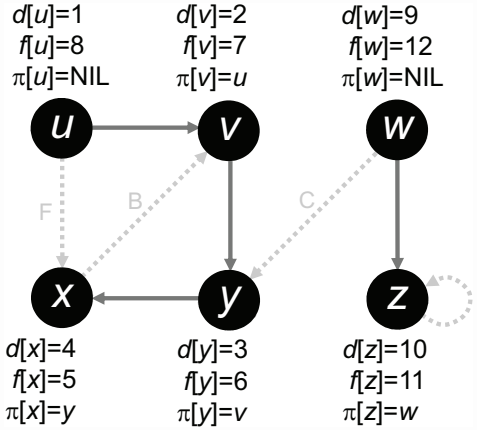
Exemplo



Exemplo



Exemplo



DFS(G)

```
for u ∈ G.V do
    color[u] ← white
    d[u] ← ∞
    f[u] ← ∞
    π[u] ← NIL
end for
time ← 1
for u ∈ G.V do
    if color[u] == white then
        DFS-Visit(G, u)
    end if
end for
```

DFS-Visit(G,u)

```
color[u] ← gray
d[u] ← time
time ← time + 1
for v ∈ G.Adj[u] do
    if color[v] == white then
        π[v] ← u
        DFS-Visit(G, v)
    end if
end for
color[u] ← black
f[u] ← time
time ← time + 1
```

Complexidade?

Complexidade

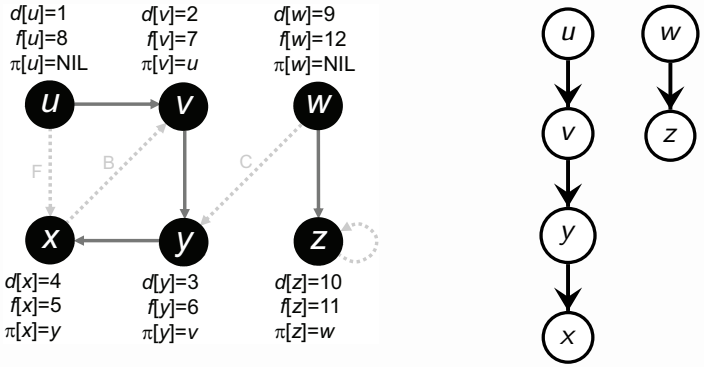
- Inicialização: $\Theta(V)$
- Chamadas a DFS-Visit dentro de DFS: $\Theta(V)$
- Arcos analisados em DFS-Visit: $\Theta(E)$
  - Chamadas a DFS-Visit dentro de DFS-Visit: $O(V)$
  - Mas $\sum_{v \in V} |Adj[v]| = \Theta(E)$

Tempo de execução: $\Theta(V + E)$

Resultado da DFS

Floresta Depth-First (DF)

- $G_\pi = (V, E_\pi)$
- $E_\pi = \{(\pi[v], v) : v \in V \wedge \pi[v] \neq NIL\}$
- Floresta DF composta por várias árvores DF



Propriedade: Estrutura de parêntesis

Se considerarmos que:

- (u - representa a descoberta de u
- u) - representa o fim de u

a história de descobertas e fim formam uma expressão bem formada com parêntesis aninhados

Exemplo

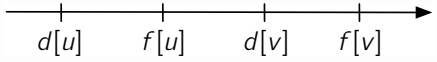
s z y x x y w w z s

(((( )) ( )))

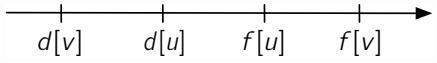
Teorema dos parêntesis

Para qualquer DFS de $G = (V, E)$, para cada par de vértices u e v apenas um dos 3 casos seguintes é verdade:

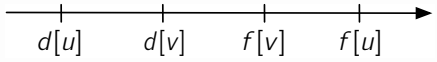
- $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos



- $[d[u], f[u]] \subset [d[v], f[v]]$ e u é descendente de v na árvore DF



- $[d[v], f[v]] \subset [d[u], f[u]]$ e v é descendente de u na árvore DF



- $[d[u], f[u]]$ e $[d[v], f[v]]$ não se podem intersetar parcialmente

Propriedade: Classificação de arcos (u, v)

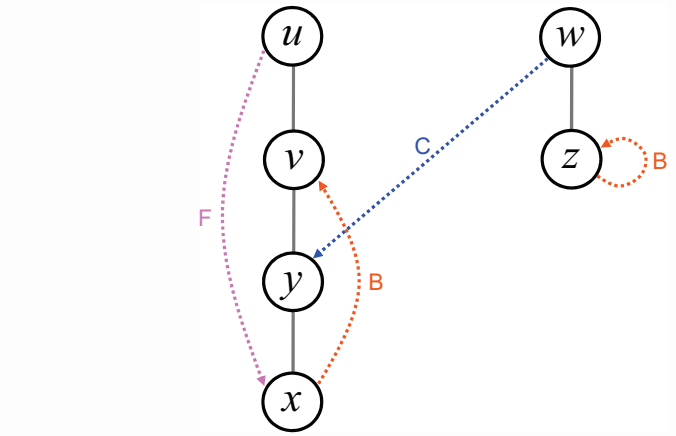
- Arcos de árvore: (**tree edges**)
  - arcos na floresta DF, G_π
  - (u, v) é arco de árvore se v foi visitado devido ao arco (u, v) ser visitado
- Arcos para trás: (**back edges**)
  - ligam vértice u a vértice v antecessor na mesma árvore DF
- Arcos para a frente: (**forward edges**)
  - ligam vértice v a vértice descendente na mesma árvore DF
- Arcos de cruzamento: (**cross edges**)
  - na mesma árvore DF, se u (ou v) não antecessor de v (ou u)
  - ou entre árvores DF diferentes

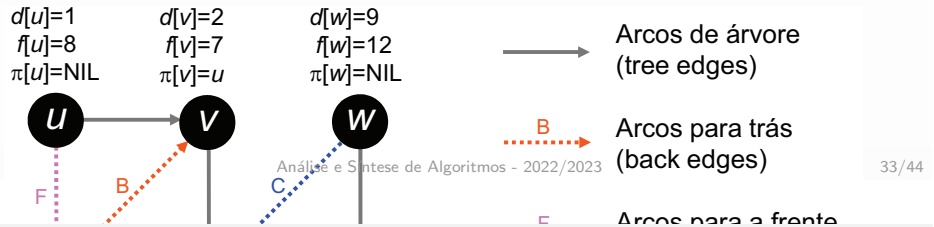
Procura em profundidade primeiro (DFS)

Propriedade: **Classificação de arcos** (u, v)

- Arcos de árvore: (**tree edges**)
  - $d[u] < d[v] < f[v] < f[u]$
  - $color[v] = white$ quando (u, v) é analisado
- Arcos para trás: (**back edges**)
  - $d[u] < d[v] < f[v] < f[u]$
  - $color[u] = gray$ quando (v, u) é analisado
- Arcos para a frente: (**forward edges**)
  - $d[u] < d[v] < f[v] < f[u]$
  - $color[v] = black$ quando (u, v) é analisado
- Arcos de cruzamento: (**cross edges**)
  - $d[v] < f[v] < d[u] < f[u]$
  - $color[v] = black$ quando (u, v) é analisado

Procura em profundidade primeiro (DFS)





Procura em profundidade primeiro (DFS)

Propriedade

Dado $G = (V, E)$ **não dirigido**, cada arco é arco de árvore ou para trás

- i.e., não existem arcos para a frente e de cruzamento

Teorema caminho branco

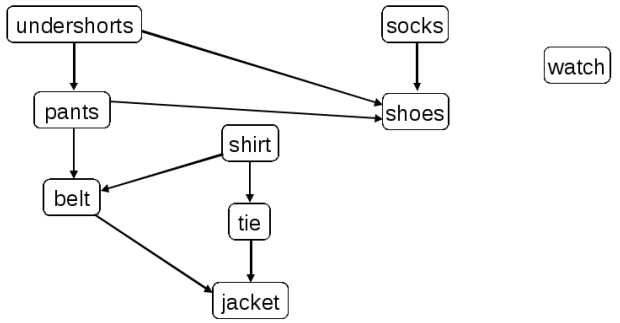
Numa floresta DF (grafo dirigido ou não dirigido):

- v descendente de $u \Leftrightarrow$ existe caminho de vértices brancos de u para v, quando u é descoberto
  - Qualquer vértice w descendente de u verifica $[d[w], f[w]] \subset [d[u], f[u]]$, pelo que w é branco quando u é descoberto

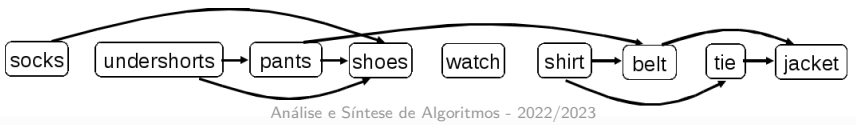
Ordenação Topológica

Motivação

Grafo que representa um conjunto de dependências ou precedências:



Ordenação Topológica:



Caminhos em Grafos

Dado um grafo $G = (V, E)$, um caminho p é uma sequência $< v_0, v_1, ..., v_k >$ tal que para todo o i, $0 \leq i \leq k-1$, $(v_i, v_{i+1}) \in E$

- Se existe um caminho p de u para v, então v diz-se atingível a partir de u usando p
- Um ciclo num grafo $G = (V, E)$ é um caminho $< v_0, v_1, ..., v_k >$, tal que $v_0 = v_k$
- Um grafo dirigido $G = (V, E)$ se não tem ciclos diz-se acíclico (Directed Acyclic Graph – DAG)

Ordenação Topológica

Dado um **DAG** $G = (V, E)$ é uma ordenação de todos os vértices tal que se $(u, v) \in E$ então u aparece antes de v na ordenação

Aplicações

- Gestão dependências pacotes
- Avaliação de células em folhas de cálculo
- Resolução dependências símbolos em linkers
- ...

Soluções Algorítmicas

- Eliminação de vértices
- Utilizando informação de DFS

Algoritmo eliminação de vértices (Kahn's)

Propriedades DAG

Dado que não contém ciclos:

- Existe pelo menos um vértice com *in-degree* $= 0$
  - No qual não existem arcos a incidir
- Existe pelo menos um nó com *out-degree* $= 0$
  - Do qual não existem arcos a sair

Algoritmo eliminação de vértices (Kahn's)

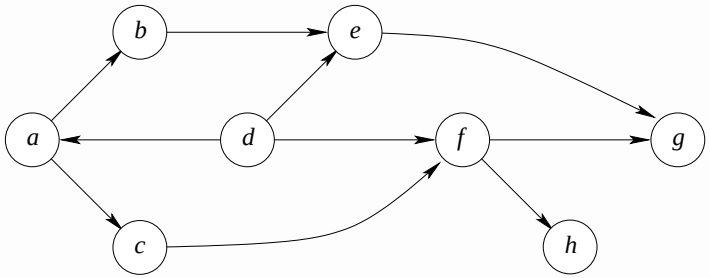
Topological-Sort-1(G)

```
L ← ∅
Q ← ∅
for each v ∈ G.V do
    if (w, v) ∉ G.E then
        Enqueue(Q, v)
    end if
end for
while Q ≠ ∅ do
    u = Dequeue(Q)
    Eliminar todos os arcos (u, v)
    if (w, v) ∉ G.E then
        Enqueue(Q, v)
    end if
    L ← L + u
end while
return L
```

Complexidade

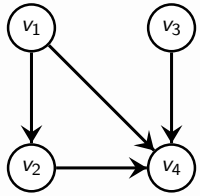
- $\Theta(V + E)$

Exemplo



Ordenação? d, a, b, c, e, f, g, h

Algoritmo baseado na DFS: Intuição



Depois de executar a DFS:

- $f[v_3]$ é sempre $> f[v_4]$
- $f[v_2]$ é sempre $> f[v_4]$
- $f[v_1]$ é sempre $> f[v_2]$,$f[v_4]$

Num DAG, se existe caminho de u para v, então $f[u] > f[v]$!
Logo, basta ordenar os vértices de forma decrescente dos tempos de fim

Algoritmo baseado na DFS

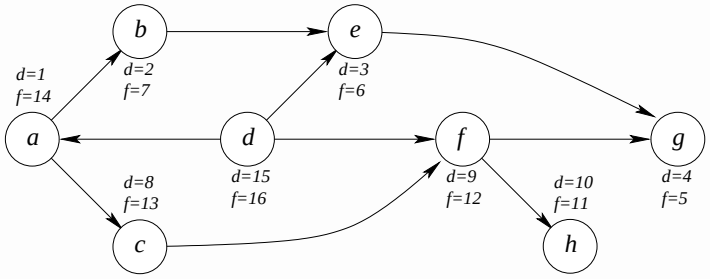
Topological-Sort-2(G)

DFS(G) para calculo do tempo de fim $f[v]$, para cada $v \in G.V$
Quando um vértice é terminado, inserir numa pilha
return pilha

Complexidade

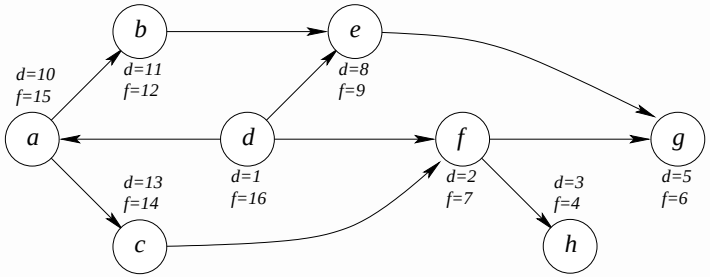
- $\Theta(V + E)$

Exemplo



Ordenação? $\langle d, a, c, f, h, b, e, g \rangle$

Exemplo



Ordenação? $\langle d, a, c, b, e, f, g, h \rangle$