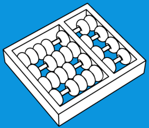
Buscas em grafos. Ordenação topológica

MO417 - Complexidade de Algoritmos I

19

05/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP

"... when making chocolate bundt cake, (1) the ingredients have to be mixed before going in the bundt pan; (2) the bundt pan has to be greased and floured before the batter can be poured in; (3) the oven has to be preheated before the cake can bake; (4) the cake has to be baked before it cools; (4) the cake has to cool before it can be iced.."

Interview Cake Website

Busca em profundidade

Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

- Começamos com o vértice de origem.
- Depois, todos os alcançáveis pelo primeiro vizinho.
- Depois, todos os alcançáveis pelo segundo vizinho.
- etc.

É a estratégia usada por vários algoritmos:

- Identificar as componentes conexas.
- Encontrar uma ordenação topológica.

Construindo uma árvore de busca

Ideia do algoritmo:

- Começamos com o vértice de origem **s**.
- Para cada vizinho não visitado **v** do vértice atual **u**:
  1. Adicionamos uma aresta **(u,v)** à árvore de busca.
  2. Visitamos **RECURSIVAMENTE** a partir de **v**.

Alternativa

Ideia alternativa:

- Percorremos os vértices usando uma **PILHA** S.
- Começamos adicionando o vértice de origem **s** em S.
- Enquanto houver vértices em S, repetimos o seguinte processo:
  - Removemos o vértice do topo de S, **u**.
  - Para cada vizinho **v** do vértice atual **u**:
    - Adicionamos uma aresta **(u,v)** à árvore de busca.
    - Inserimos **v** na pilha de processamento.

Observações:

- Pode levar a uma árvore de busca distinta à da primeira ideia.
- Compare com fila da a busca em largura.

Floresta de busca

Visitando todos os vértices:

- A árvore de busca contém só vértices alcançáveis de **s**.
- Algumas vezes queremos visitar todos os vértices.
- Repetimos o processo com os vértices não visitados.
- Obteremos uma **FLORESTA DE BUSCA**.

Representando uma floresta:

- Também utilizamos um vetor de pais π.
- Um vértice **v** com $\pi[\mathbf{v}] = \text{NIL}$ é raiz de uma árvore de busca.
- As arestas da floresta são:

$$\{(\pi[\mathbf{v}], \mathbf{v}) : \mathbf{v} \in \mathbf{V[G]} \text{ e } \pi[\mathbf{v}] \neq \text{NIL}\}$$

Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

1. cor[**v**] = branco se não descobrimos **v** ainda.
2. cor[**v**] = cinza se já descobrimos, mas não finalizamos **v**.
3. cor[**v**] = preto se já descobrimos e já finalizamos **v**.

Observações:

- Os vértices cinza têm suas chamadas recursivas ativas.
- A pilha de chamadas induz um caminho na floresta.

Tempo de descoberta e finalização

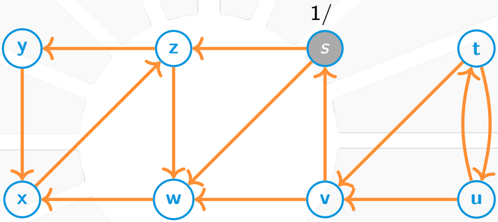
Ademais, vamos associar rótulos aos vértices:

- $d[\mathbf{v}]$ é o instante de **DESCOBERTA** de **v**.
- $f[\mathbf{v}]$ é o instante de **FINALIZAÇÃO** de **v**.

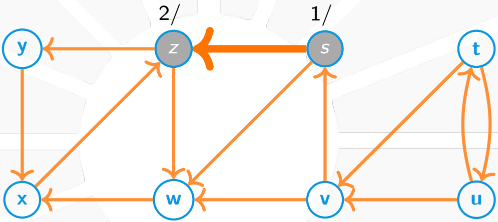
Observações:

- Os rótulos são inteiros distintos entre 1 e $2|\mathbf{V}|$.
- Os rótulos refletem os instantes em que **v** muda de cor.

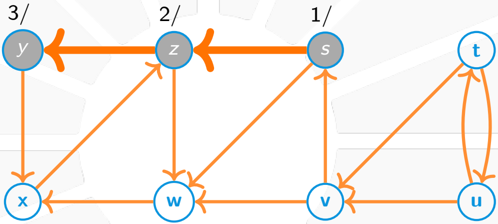
Exemplo de busca em profundidade



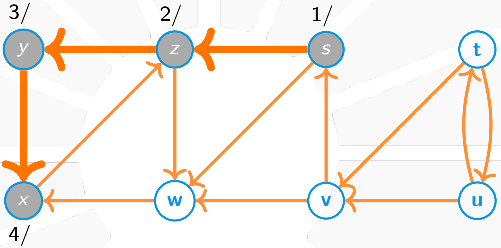
Exemplo de busca em profundidade



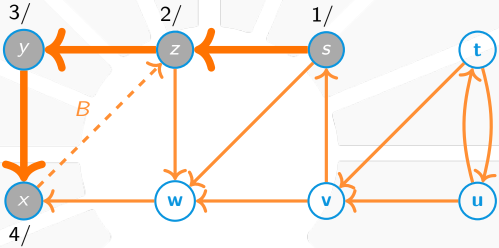
Exemplo de busca em profundidade



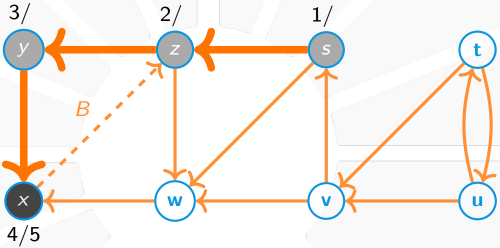
Exemplo de busca em profundidade



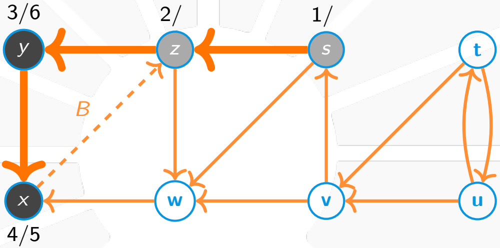
Exemplo de busca em profundidade



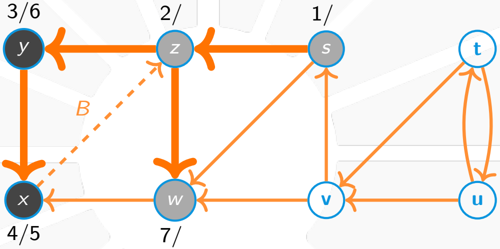
Exemplo de busca em profundidade



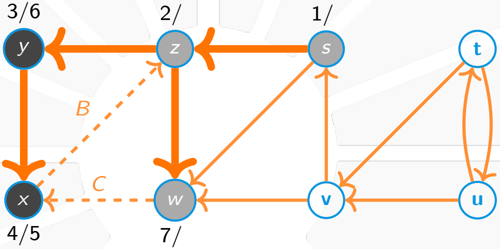
Exemplo de busca em profundidade



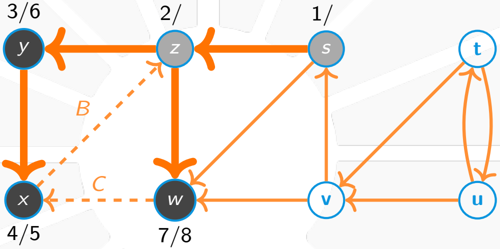
Exemplo de busca em profundidade



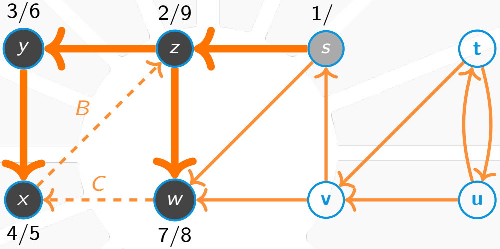
Exemplo de busca em profundidade



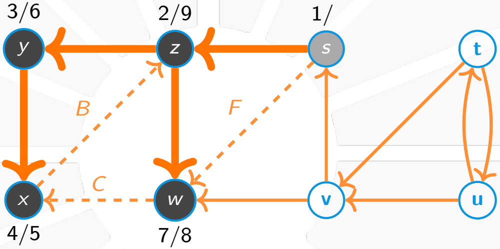
Exemplo de busca em profundidade



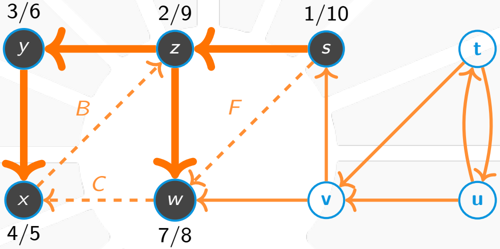
Exemplo de busca em profundidade



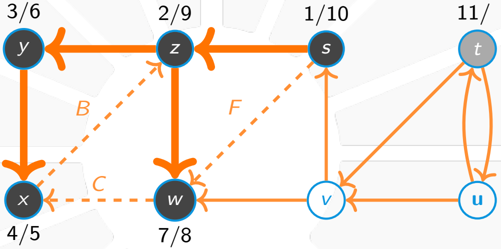
Exemplo de busca em profundidade



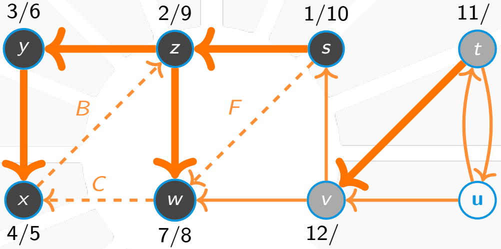
Exemplo de busca em profundidade



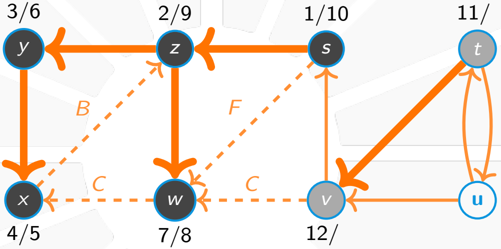
Exemplo de busca em profundidade



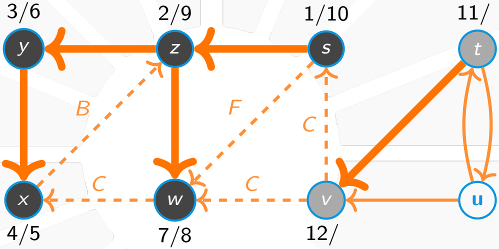
Exemplo de busca em profundidade



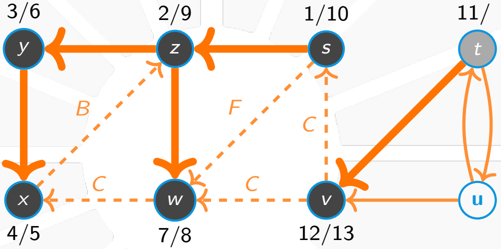
Exemplo de busca em profundidade



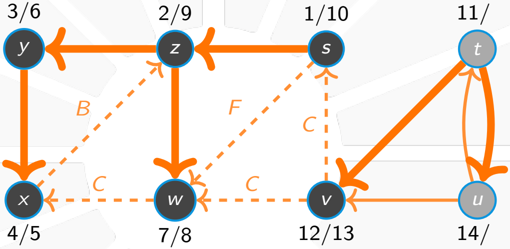
Exemplo de busca em profundidade



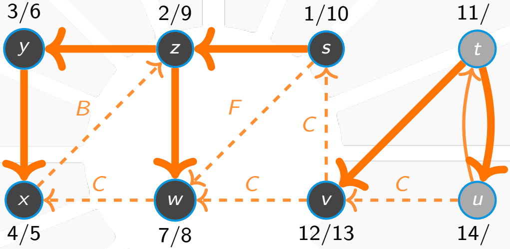
Exemplo de busca em profundidade



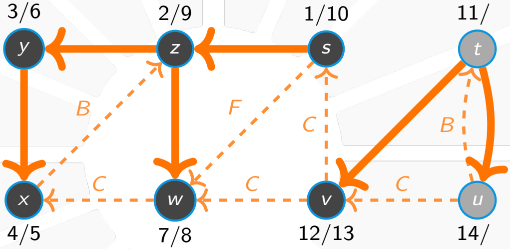
Exemplo de busca em profundidade



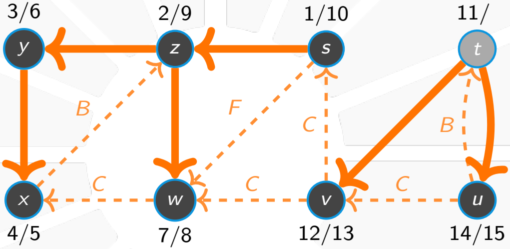
Exemplo de busca em profundidade



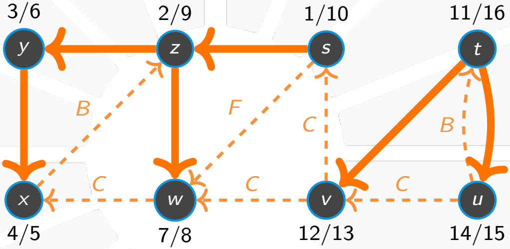
Exemplo de busca em profundidade



Exemplo de busca em profundidade



Exemplo de busca em profundidade



Rótulos versus cores

Observe que para todo vértice **v**:

- **v** é branco antes do instante $d[\mathbf{v}]$.
- **v** é cinza entre os instantes $d[\mathbf{v}]$ e $f[\mathbf{v}]$.
- **v** é preto após o instante $f[\mathbf{v}]$.

Algoritmo DFS

Algoritmo: $\mathrm{DFS}(G)$

para cada $u \in V[G]$
    $\mathrm{cor}[u] \leftarrow$ branco
    $\pi[u] \leftarrow$ NIL
tempo $\leftarrow 0$
para cada $u \in V[G]$
    se $\mathrm{cor}[u] =$ branco
        $\mathrm{DFS\text{-}VISIT}(u)$

- Representamos G com listas de adjacências.
- A floresta de busca em profundidade é representada por π.
- São calculados os instantes $d[\mathbf{v}]$ e $f[\mathbf{v}]$.

Algoritmo DFS-VISIT

Algoritmo: DFS-VISIT(u)

cor[u] $\leftarrow$ cinza
tempo $\leftarrow$ tempo $+ 1$
$d[u] \leftarrow$ tempo
para cada $v \in Adj[u]$
    se cor[v] $=$ branco
        $\pi[v] \leftarrow u$
        DFS-VISIT(v)
cor[u] $\leftarrow$ preto
tempo $\leftarrow$ tempo $+ 1$
$f[u] \leftarrow$ tempo

Constrói uma arvore de busca com origem **u**.

Análise de complexidade

Tempo do algoritmo principal DFS:

- A inicialização consome tempo $O(V)$.
- Realizamos $|V|$ chamadas a $\text{DFS-}\textsc{visit}$.

Tempo da sub-rotina $\text{DFS-}\textsc{visit}$:

- Processamos cada vértice exatamente uma vez.
- Cada chamada percorre sua lista de adjacências.
- O tempo gasto percorrendo adjacências é $O(E)$.

A complexidade da busca em profundidade é $O(V + E)$.

Teorema dos parênteses

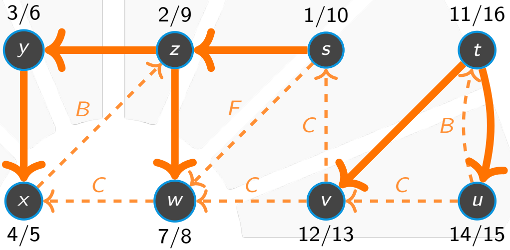
Teorema

*Se **u** e **v** são vértices de uma árvore de busca em profundidade, então ocorre exatamente um entre os três casos abaixo:*

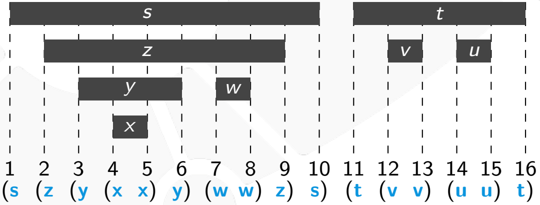
1. (a) *Os intervalos* $[d[u], f[u]]$ *e* $[d[v], f[v]]$ *são disjuntos.*
   (b) *Nesse caso* ***u*** *e* ***v*** *não são descendentes um do outro.*
2. (a) *O intervalo* $[d[u], f[u]]$ *está contido em* $[d[v], f[v]]$.
   (b) *Nesse caso* ***u*** *é descendente de* ***v***.
3. (a) *O intervalo* $[d[v], f[v]]$ *está contido em* $[d[u], f[u]]$.
   (b) *Nesse caso* ***v*** *é descendente de* ***u***.

Exemplo

Floresta de busca



Estrutura de parênteses



Demonstração do teorema

Demonstração do teorema dos parênteses:

- Podemos supor que $d[\mathbf{u}] < d[\mathbf{v}]$.
- Analisamos dois casos:

  Caso 1: Se $d[\mathbf{v}] < f[\mathbf{u}]$:
  - Então, **v** foi descoberto enquanto **u** era cinza.
  - Logo, a chamada recursiva para **v** termina antes da de **u**.
  - Portanto, **v** é descendente de **u** e $[d[\mathbf{v}], f[\mathbf{v}]]$ está contido em $[d[\mathbf{u}], f[\mathbf{u}]]$.

  Caso 2: Se $f[\mathbf{u}] < d[\mathbf{v}]$:
  - Então, **u** foi finalizado enquanto **v** era branco.
  - Logo, a chamada de **u** termina antes que a de **v** comece.
  - Portanto, **u** e **v** não são descendentes um do outro e $[d[\mathbf{v}], f[\mathbf{v}]]$ e $[d[\mathbf{u}], f[\mathbf{u}]]$ são disjuntos.

Teorema do caminho branco

Teorema

Considere dois vértices ***u*** *e* ***v***. *São equivalentes:*

(1) ***v*** *é descendente de* ***u*** *na floresta de busca.*

(2) *Quando* ***u*** *foi descoberto, existia um caminho de* ***u*** *a* ***v*** *formado apenas por vértices brancos.*

Demonstração do teorema do caminho branco

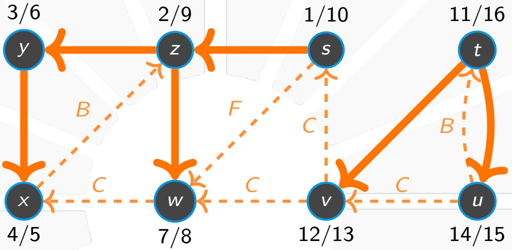
- (1) $\Rightarrow$ (2)
  - Suponha que **v** é um descendente de **u**.
  - Seja **z** um vértice qualquer no caminho de **u** até **v** na floresta.
  - Então, **z** é descendente de **u**, logo $d[\mathbf{u}] < d[\mathbf{z}]$.
  - Portanto, **z** era branco no instante $d[\mathbf{u}]$,
  - assim como todos os vértices no caminho.
- (2) $\Rightarrow$ (1)
  - Considere um caminho branco de **u** a **v** no instante $d[\mathbf{u}]$.
  - Suponha que há vértices no caminho não descendentes de **u**.
  - Sejam **z** o primeiro vértice não descendente de **u** no caminho e **w** seu antecessor.
  - Como **w** é descendente de **u**, temos $f[\mathbf{w}] \leq f[\mathbf{u}]$.
  - Como **z** não é descendente de **u**, temos $f[\mathbf{u}] < d[\mathbf{z}]$.
  - Logo, **z** era um vizinho branco de **w** no instante $f[\mathbf{w}]$.
  - Isso é uma contradição, então todo vértice do caminho branco é descendente de **u** na floresta de busca.

Classificação de arestas

Dada a floresta de busca, podemos classificar arestas do grafo:

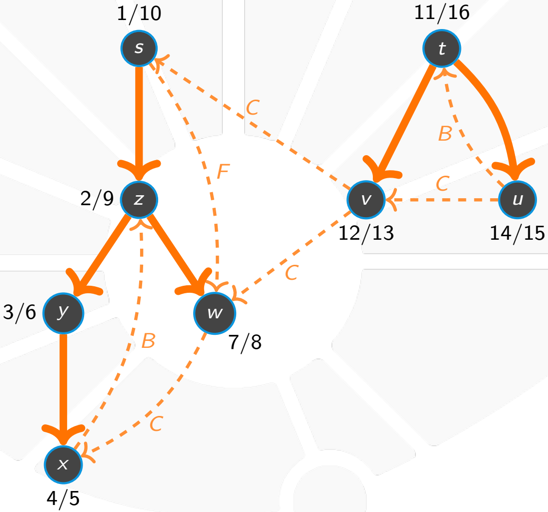
- **ARESTAS DE ÁRVORE** (*tree edges*) são arestas da floresta de busca em profundidade.
- **ARESTAS DE RETORNO** (*backward edges*) ligam um vértice a um ancestral.
- **ARESTAS DE AVANÇO** (*forward edges*) ligam um vértice a um descendente.
- **ARESTAS DE CRUZAMENTO** (*cross edges*) são todas as outras arestas do grafo.

Classificação de arestas



É fácil modificar o algoritmo $\mathrm{DFS}(G)$ para que ele também classifique as arestas de G. (exercício)

Classificação de arestas

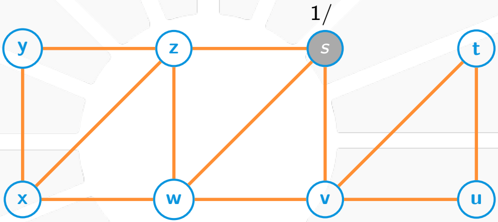


Grafos não direcionados

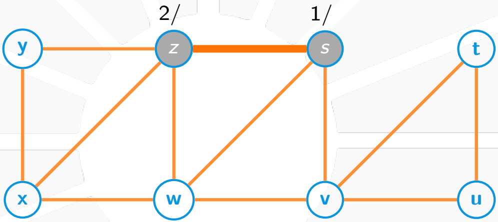
Classificando arestas não direcionadas:

- Não pode haver arestas de avanço. Por quê?
- Tampouco arestas de cruzamento. Por quê?
- Cada aresta é **ARESTA DE ÁRVORE** ou **ARESTA DE RETORNO**.

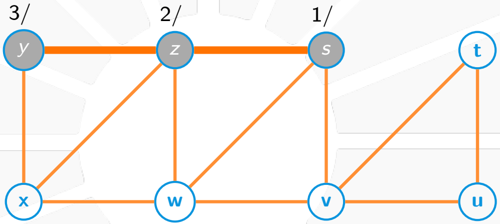
DFS em grafo não direcionado



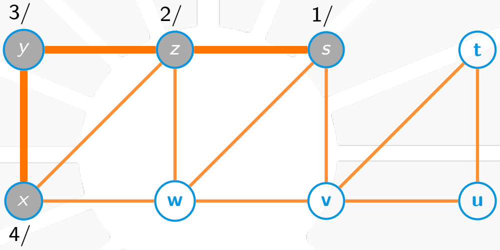
DFS em grafo não direcionado



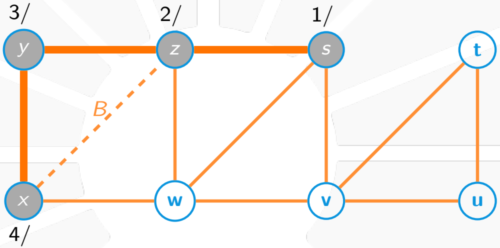
DFS em grafo não direcionado



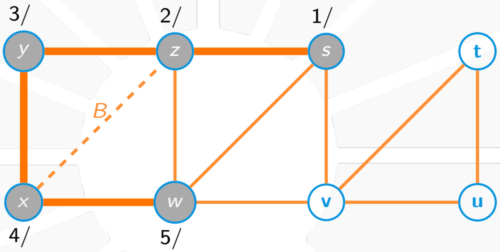
DFS em grafo não direcionado



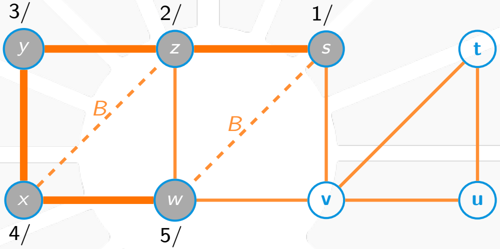
DFS em grafo não direcionado



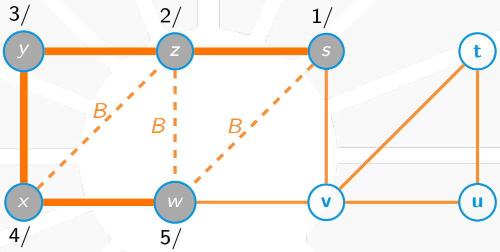
DFS em grafo não direcionado



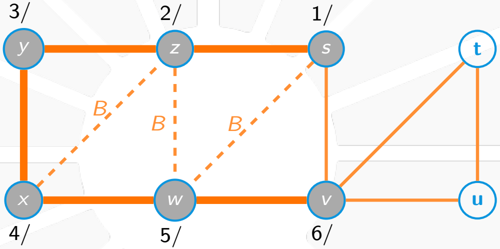
DFS em grafo não direcionado



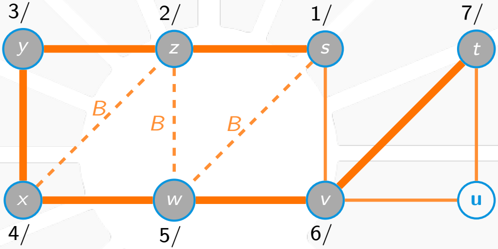
DFS em grafo não direcionado



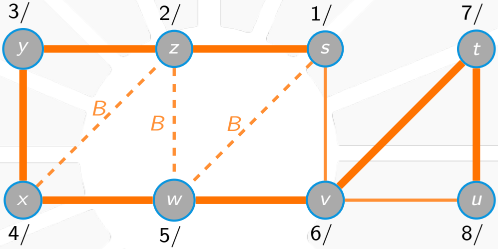
DFS em grafo não direcionado



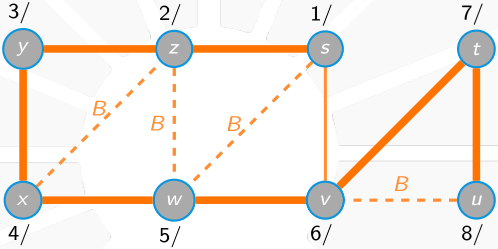
DFS em grafo não direcionado



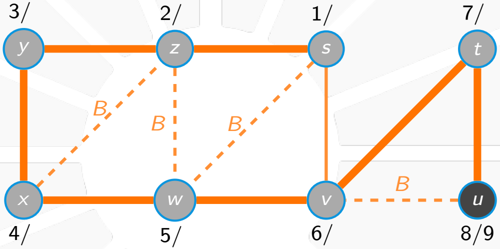
DFS em grafo não direcionado



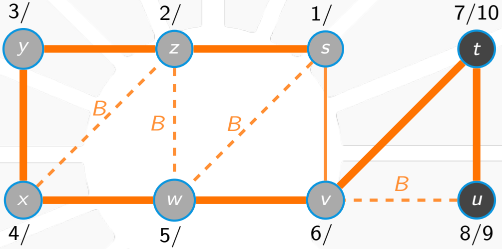
DFS em grafo não direcionado



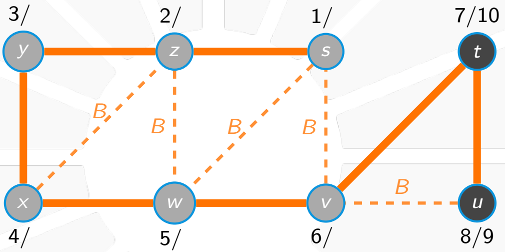
DFS em grafo não direcionado



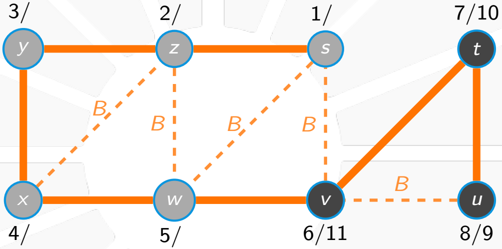
DFS em grafo não direcionado



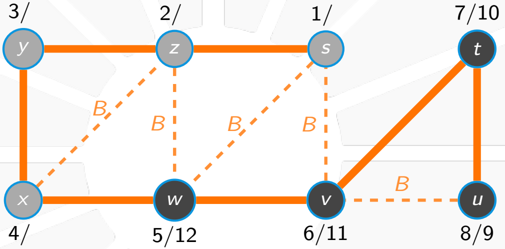
DFS em grafo não direcionado



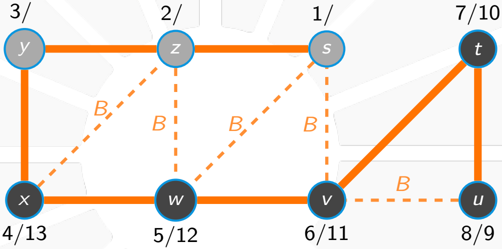
DFS em grafo não direcionado



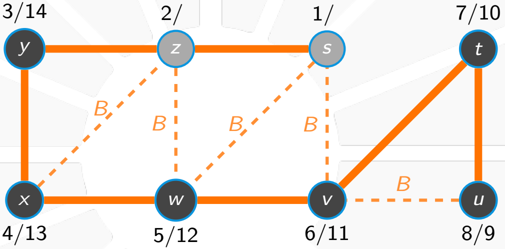
DFS em grafo não direcionado



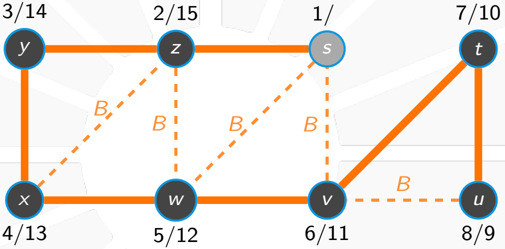
DFS em grafo não direcionado



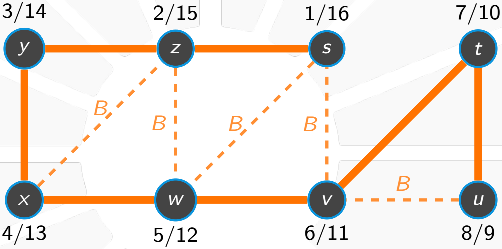
DFS em grafo não direcionado



DFS em grafo não direcionado



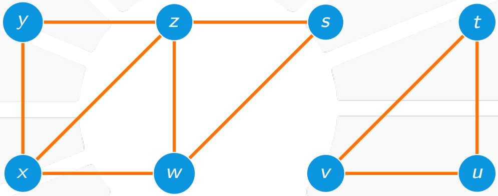
DFS em grafo não direcionado



Componentes conexas

Componentes conexas

Problema: Determinar as componentes conexas de um grafo.



Componentes conexas

Contando o número de componentes:

- Cada componente corresponde a uma árvore de busca.
- O número de componentes é o **NÚMERO DE CHAMADAS** a DFS-VISIT a partir de DFS.

Vamos modificar DFS:

- Identificamos cada componente por um número.
- Denotaremos por *comp*[**v**] a componente de **v**.

Algoritmo DFS modificado

Algoritmo: $\mathrm{DFS}(G)$

para cada $u \in V[G]$
    cor$[u] \leftarrow$ branco
$\ell \leftarrow 0$
para cada $u \in V[G]$
    se cor$[u] =$ branco
        $\ell \leftarrow \ell + 1$
        DFS-VISIT(u)

ℓ é o número de chamadas a DFS-VISIT a partir de DFS.

Algoritmo DFS-VISIT modificado

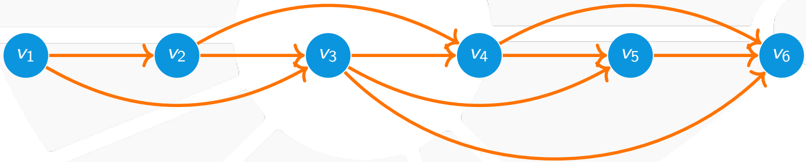
Algoritmo: DFS-VISIT(u)

cor$[u] \leftarrow$ cinza
para cada $v \in Adj[u]$
    se cor$[v] =$ branco
        DFS-VISIT(v)
cor$[u] \leftarrow$ preto
$comp[u] \leftarrow \ell$

Ordenação Topológica

Definição

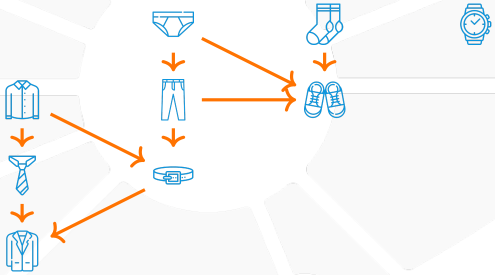
Uma **ORDENAÇÃO TOPOLÓGICA** de um grafo direcionado é um arranjo dos vértices $\mathbf{v_1, v_2, \ldots, v_n}$ tal que se $(\mathbf{v_i, v_j})$ é uma aresta do grafo, então $i < j$.



Exemplo de aplicação

Representando dependências:

- Um grafo pode representar precedências entre tarefas.
- Queremos um ordem que respeita as precedências.

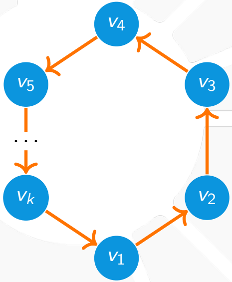


Exemplo de ordenação topológica

Condições de existência

Todo grafo direcionado possui ordenação topológica?

- **NÃO**, um ciclo direcionado não possui!
- Assim, nenhum grafo que contém um ciclo possui!



Um grafo direcionado é **ACÍCLICO** se não contiver um ciclo direcionado.

Condições de existência

Teorema

Um grafo direcionado é ***ACÍCLICO*** *se e somente se possui uma* ***ORDENAÇÃO TOPOLÓGICA****.*

Demonstração:

- Se G tem uma ordenação topológica, então ele é **ACÍCLICO**.
- Em seguida, mostraremos a recíproca.

Um lema auxiliar

- Uma **FONTE** é um vértice com grau de entrada zero.
- Um **SORVEDOURO** é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma ***FONTE*** *e um* ***SORVEDOURO****.*

Demonstração:

- Tome um caminho maximal $P = (\mathbf{v_0} \ldots \mathbf{v_k})$ em G.
- Então, $\mathbf{v_0}$ é uma fonte e $\mathbf{v_k}$ é um sorvedouro.

Demonstração do teorema

Considere um grafo acíclico $G = (\mathbf{V}, \mathbf{E})$.

Mostraremos que G possui uma ordenação topológica por indução em $|\mathbf{V}|$:

- Se $|\mathbf{V}| = 1$, então a afirmação é clara.
- Considere um grafo com pelo menos dois vértices:
  - Pelo lema anterior, G possui uma fonte $\mathbf{u}$.
  - Pela hipótese de indução, o grafo $G - \mathbf{u}$ possui uma ordenação topológica $\mathbf{v_1}, \ldots, \mathbf{v_{n-1}}$.
  - Logo, $\mathbf{u}, \mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_{n-1}}$ é uma ordenação topológica de G.

Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

- É baseada em exibir uma ordenação topológica.
- Sugere um **ALGORITMO RECURSIVO**.

Algoritmo para ordenação topológica:

1. Encontre uma fonte $\mathbf{u}$ de G.
2. Recursivamente, obtenha ordenação $\mathbf{v_1}, \ldots, \mathbf{v_{n-1}}$ de $G - \mathbf{u}$.
3. Devolva $\mathbf{u}, \mathbf{v_1}, \ldots, \mathbf{v_{n-1}}$.

A complexidade desse algoritmo é $O(V^2)$:

- Encontrar uma fonte leva tempo $O(V)$.
- Há $|V|$ chamadas recursivas.
- Pode-se fazer em tempo $O(V + E)$. (exercício)

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico:

- Como não há ciclo, não existe aresta de retorno.
- Considere o instante em que **v** fica preto.
- Nesse instante **TODOS** seus vizinhos são pretos.
- Isso sugere considerar os vértices na ordem de término.

Ideia para o algoritmo:

- O primeiro vértice a ficar preto não tem arestas saindo.
- O segundo só pode ter arestas para o primeiro.
- O terceiro só pode ter arestas para os dois primeiros.
- etc.

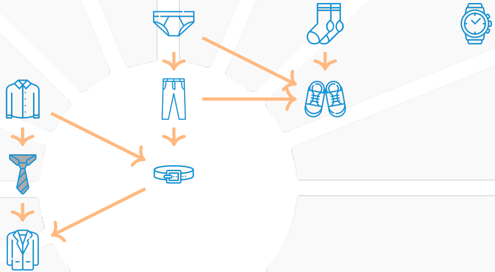
Algoritmo Topological-Sort

Algoritmo: Topological-Sort(u)

execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada vértice v
quando um vértice finalizar, insira-o no **INÍCIO** de uma lista
devolva a lista resultante

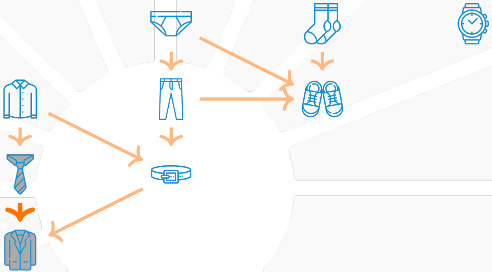
- Inserir cada um dos $|V|$ vértices leva tempo $O(1)$.
- Executamos DFS uma vez.
- Portanto, a complexidade de tempo é $O(V + E)$.

Exemplo



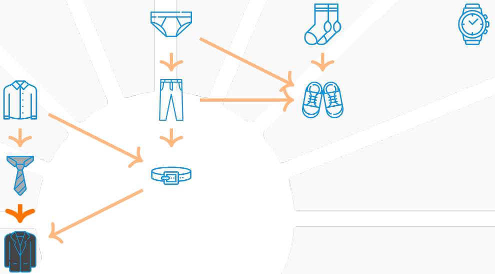
Lista:

Exemplo



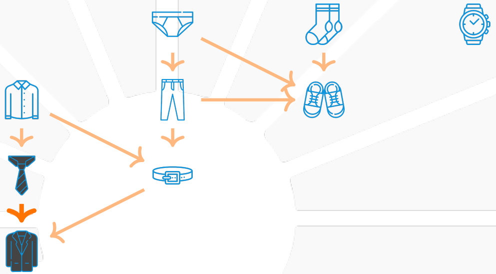
Lista:

Exemplo



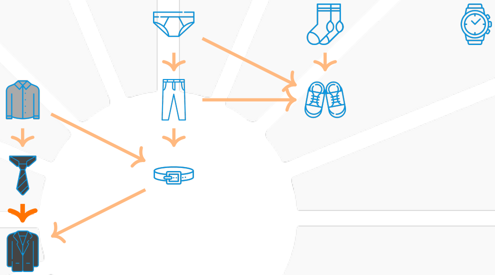
Lista:

Exemplo



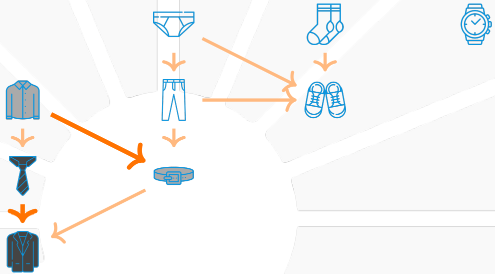
Lista:

Exemplo



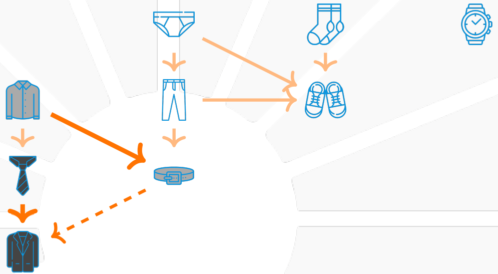
Lista:

Exemplo



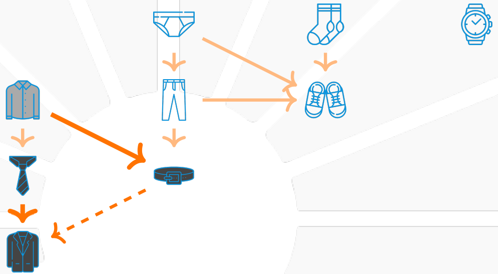
Lista:

Exemplo



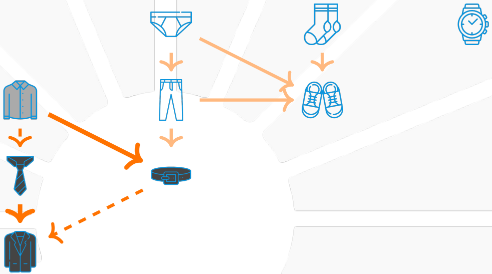
Lista:

Exemplo



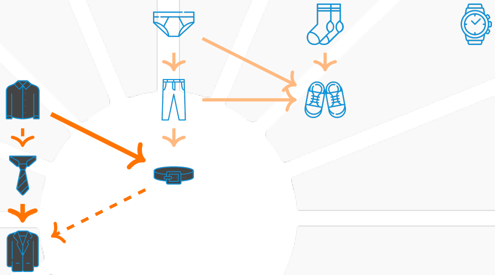
Lista:

Exemplo



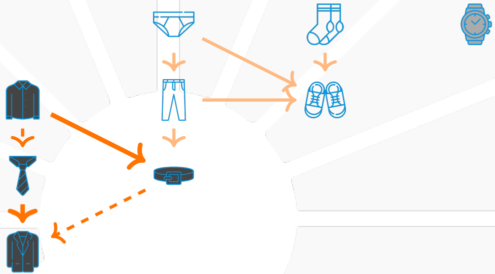
Lista:

Exemplo



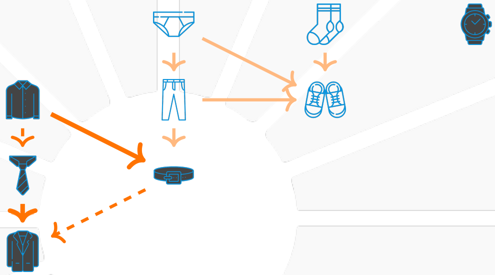
Lista:

Exemplo



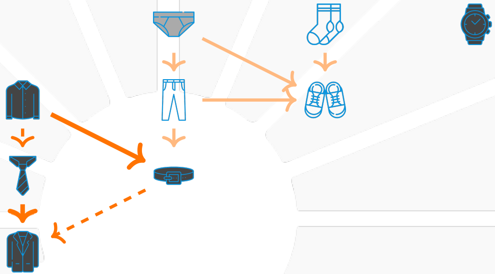
Lista:

Exemplo



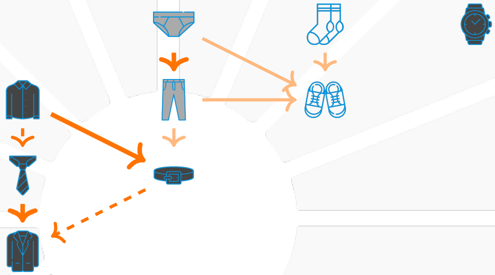
Lista:

Exemplo



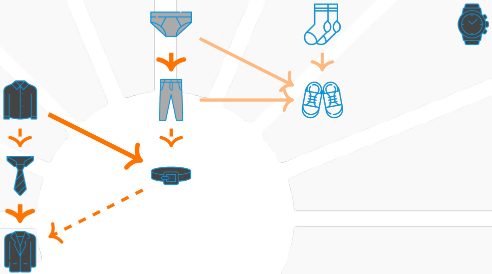
Lista:

Exemplo



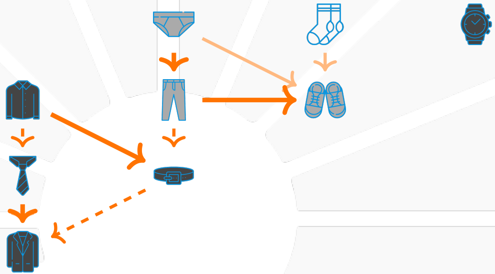
Lista:

Exemplo



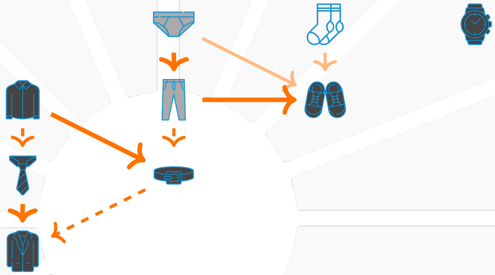
Lista:

Exemplo



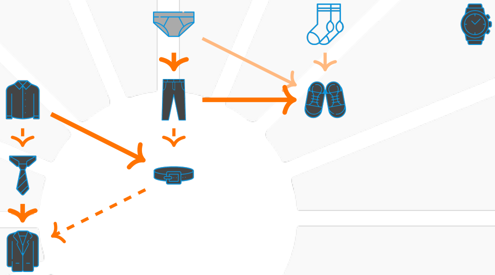
Lista:

Exemplo



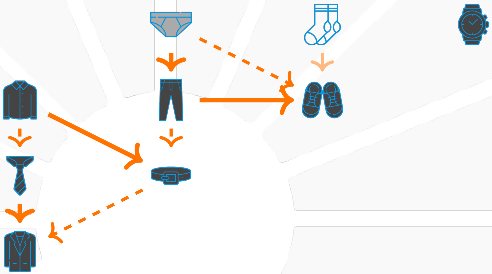
Lista:

Exemplo



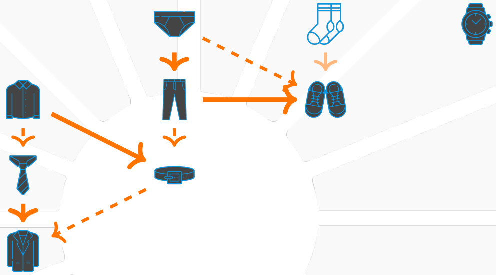
Lista:

Exemplo



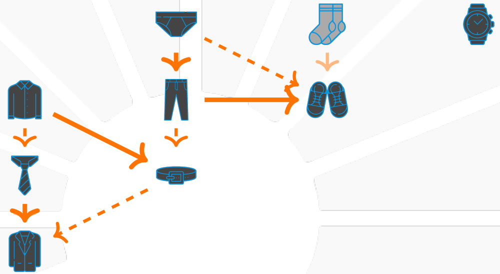
Lista:

Exemplo



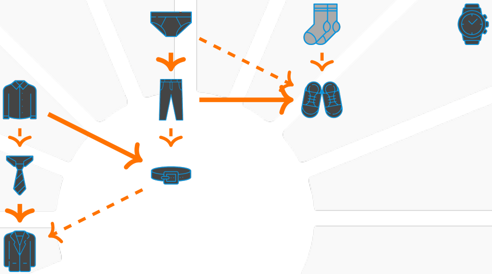
Lista:

Exemplo



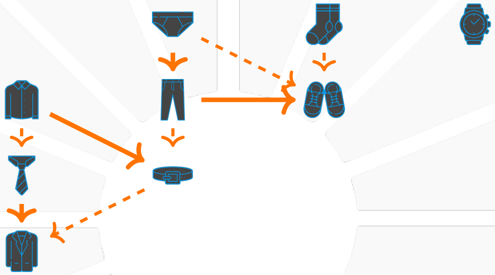
Lista:

Exemplo



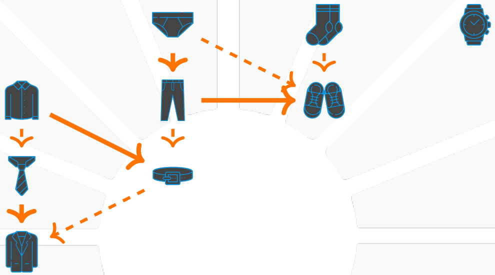
Lista:

Exemplo



Lista:

Exemplo



Lista:

Correção

Teorema

TOPOLOGICAL-SORT(G) *devolve uma ordenação topológica de* G.

Demonstração:

Considere uma aresta arbitrária **(u,v)**:

- Como a lista devolvida está em ordem **DECRESCENTE** de $f[\mathbf{v}]$, basta mostrar que $f[\mathbf{u}] > f[\mathbf{v}]$.
- Considere o instante em que **(u,v)** foi examinada:
- Como **(u,v)** não é aresta de retorno, **v** não pode ser cinza:
    1. Se **v** for branco, então ele será descendente de **u** e $f[\mathbf{u}] > f[\mathbf{v}]$.
    2. Se **v** for preto, então ele já foi finalizado e $f[\mathbf{u}] > f[\mathbf{v}]$.

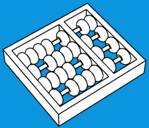
Buscas em grafos. Ordenação topológica

MO417 - Complexidade de Algoritmos I

05/24

19

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br



UNICAMP