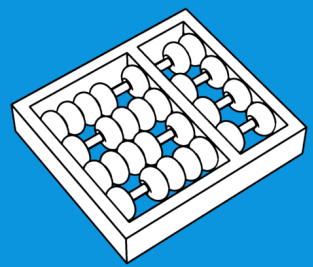
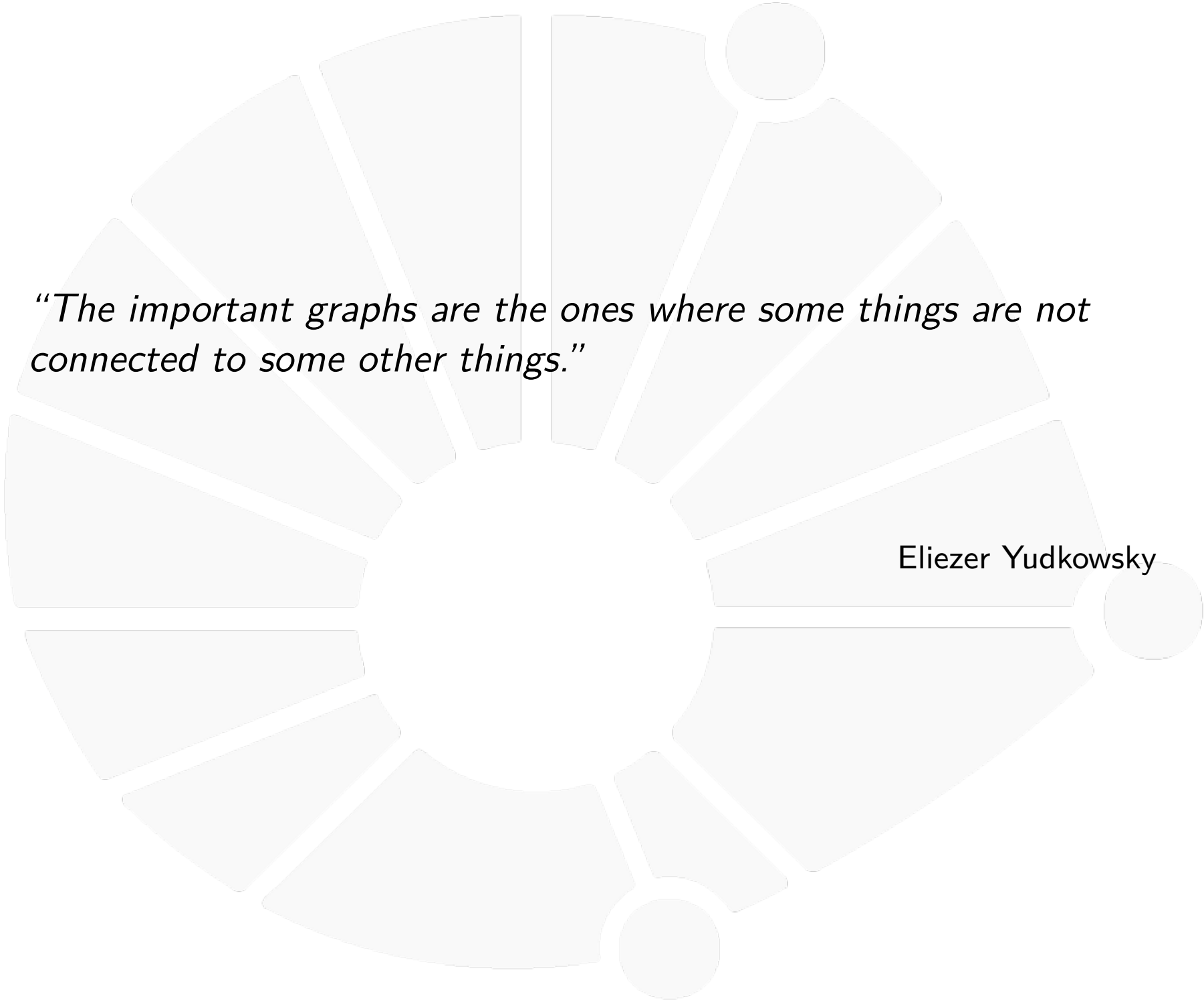
Componentes conexas e ordenação topológica

MC558 - Projeto e Análise de Algoritmos II

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

UNICAMP

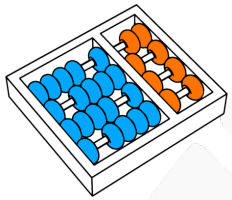




"The important graphs are the ones where some things are not connected to some other things."

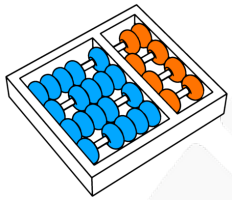
Eliezer Yudkowsky

Componentes conexas



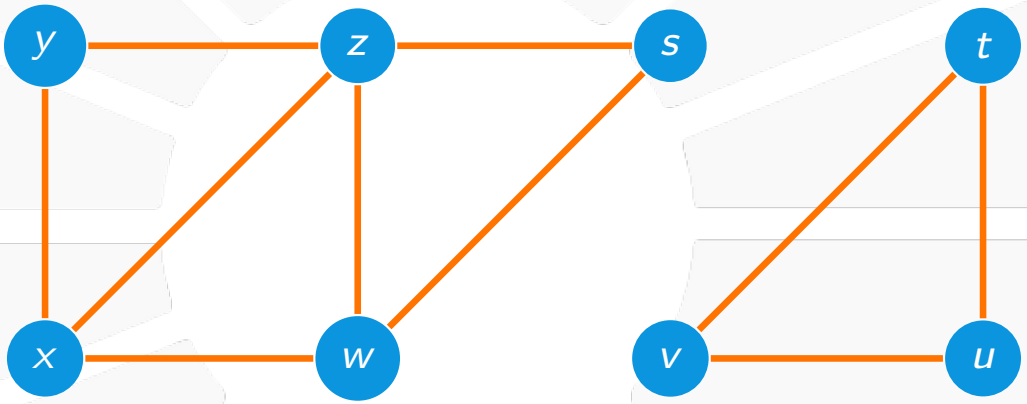
Componentes conexas

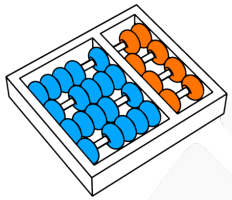
Problema: Determinar as componentes conexas de um grafo.



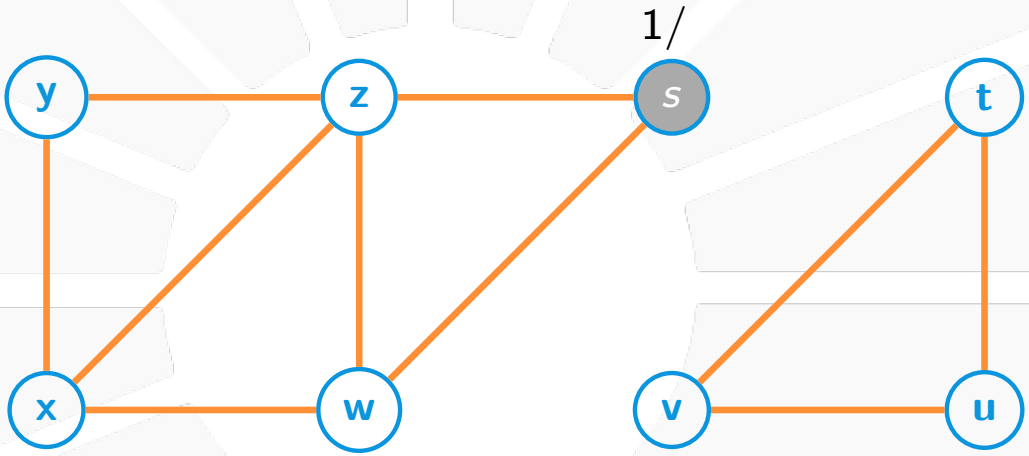
Componentes conexas

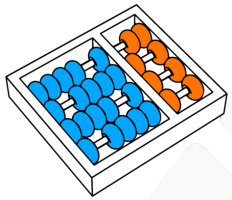
Problema: Determinar as componentes conexas de um grafo.



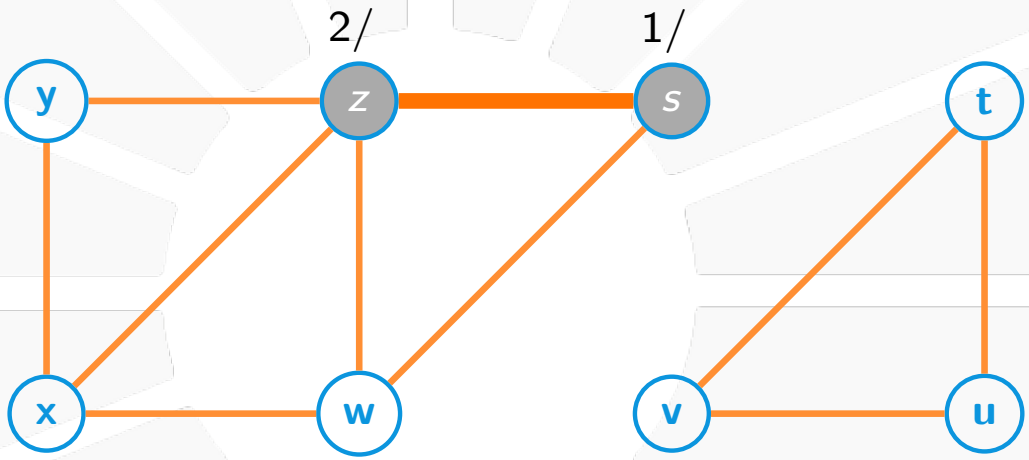


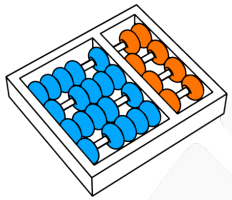
Executando DFS



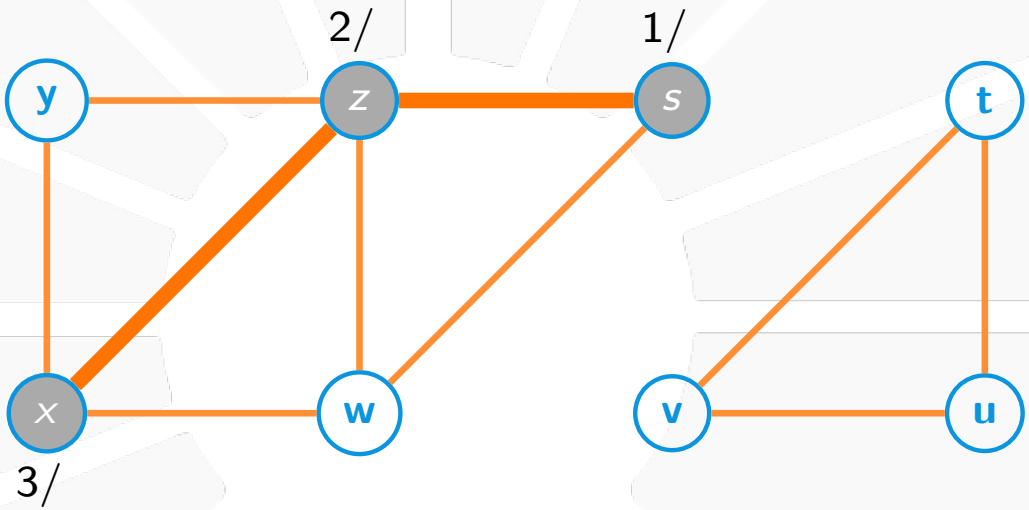


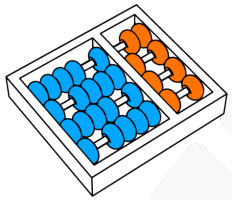
Executando DFS



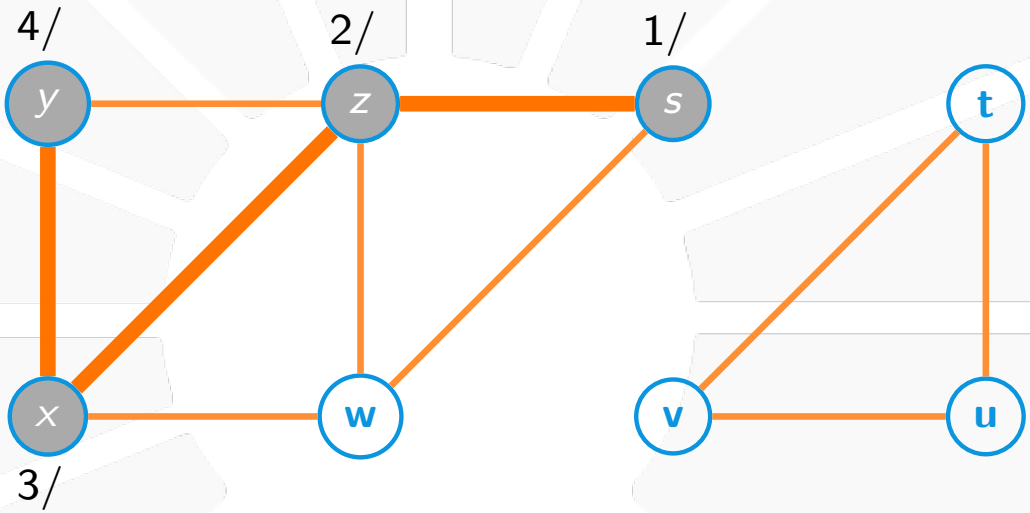


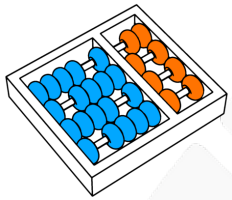
Executando DFS



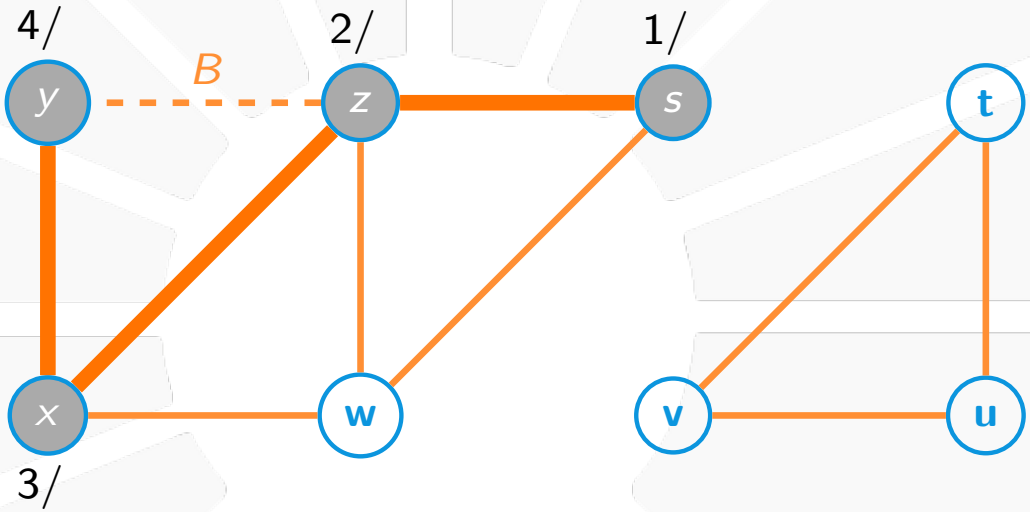


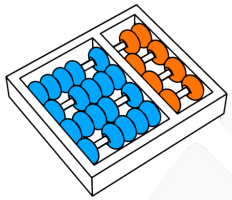
Executando DFS



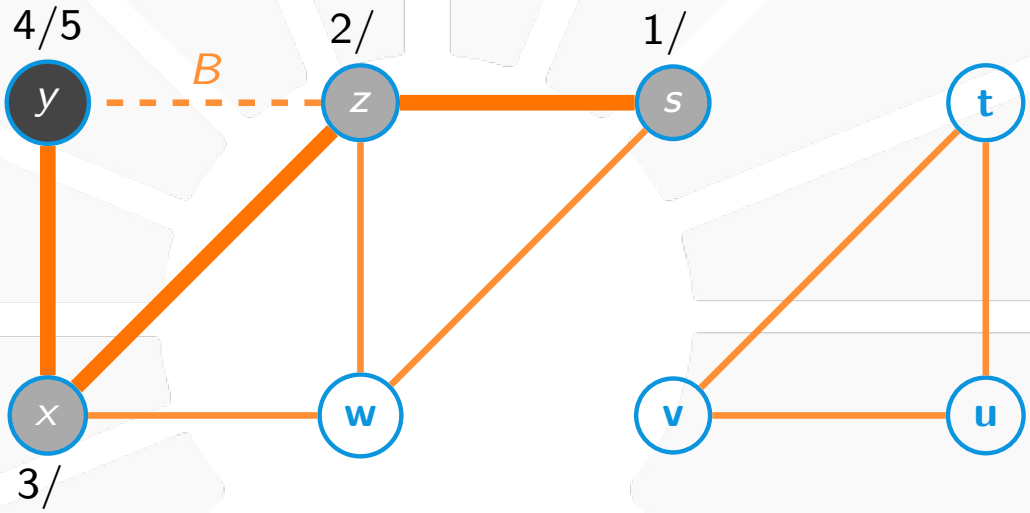


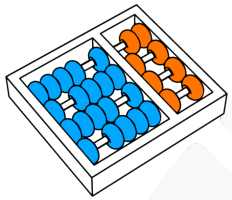
Executando DFS



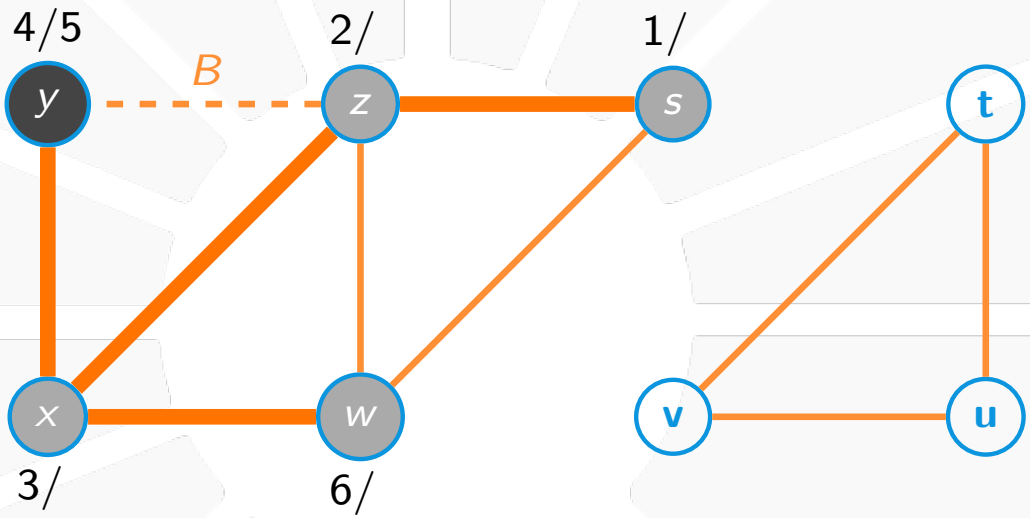


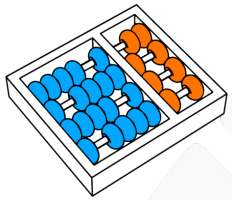
Executando DFS



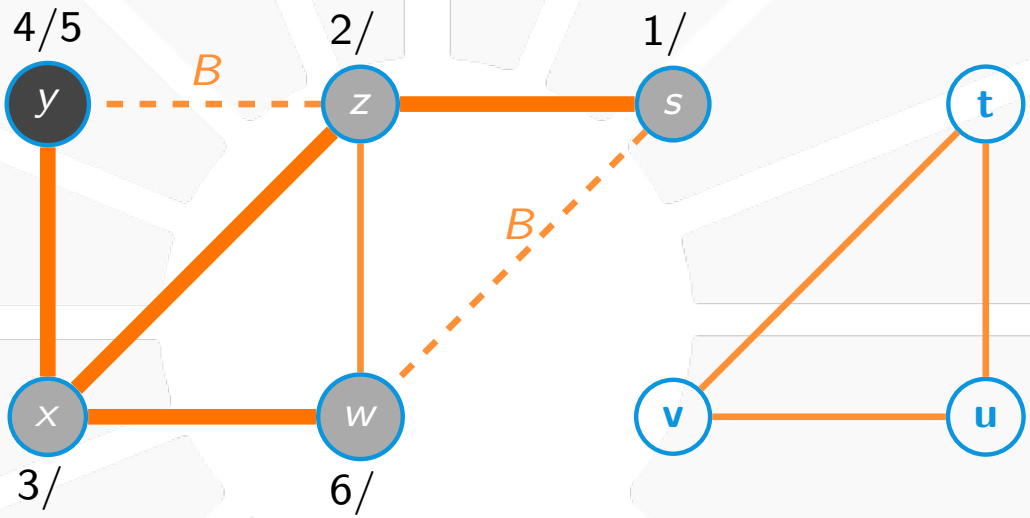


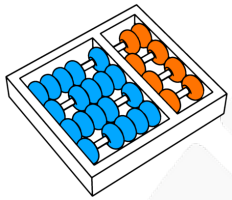
Executando DFS



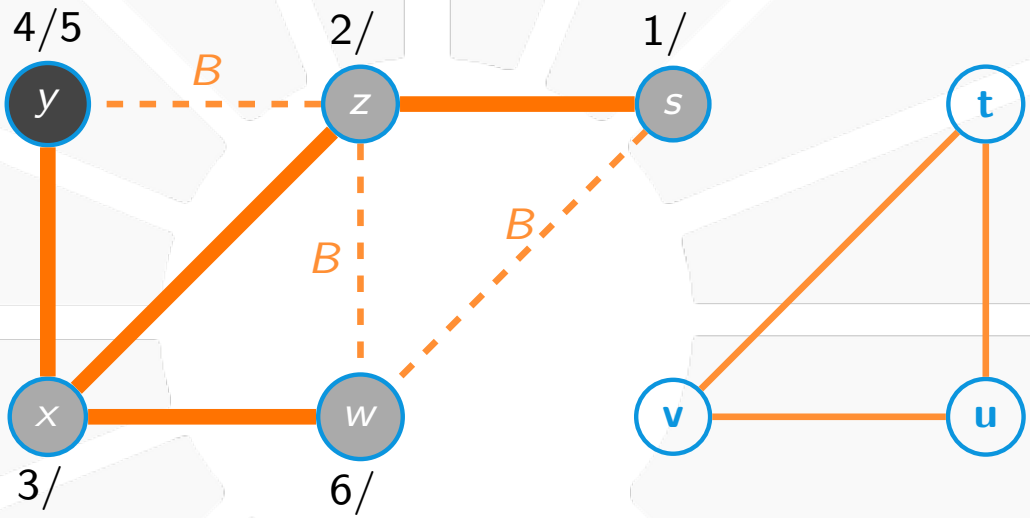


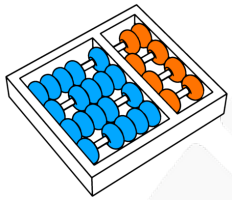
Executando DFS



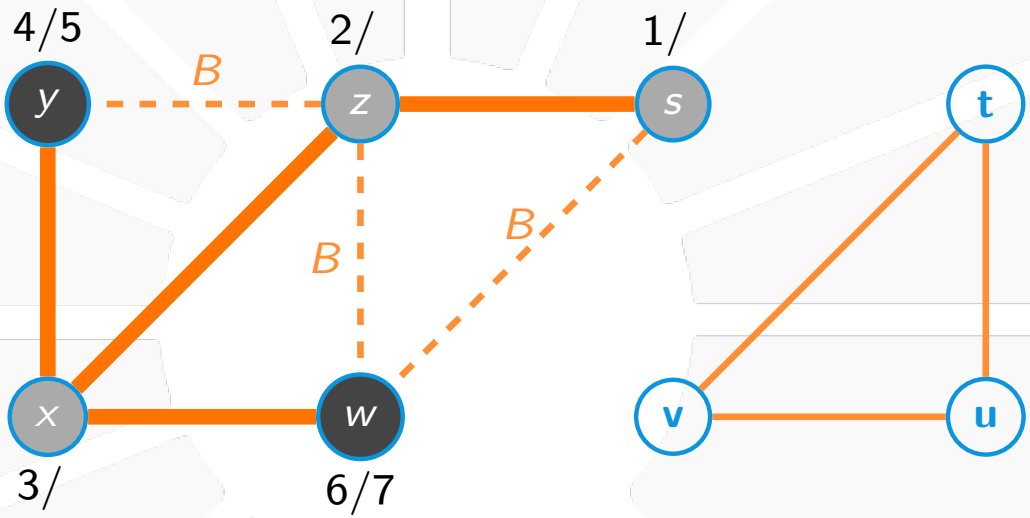


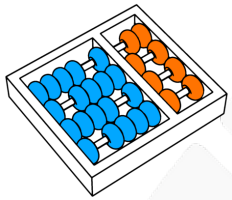
Executando DFS



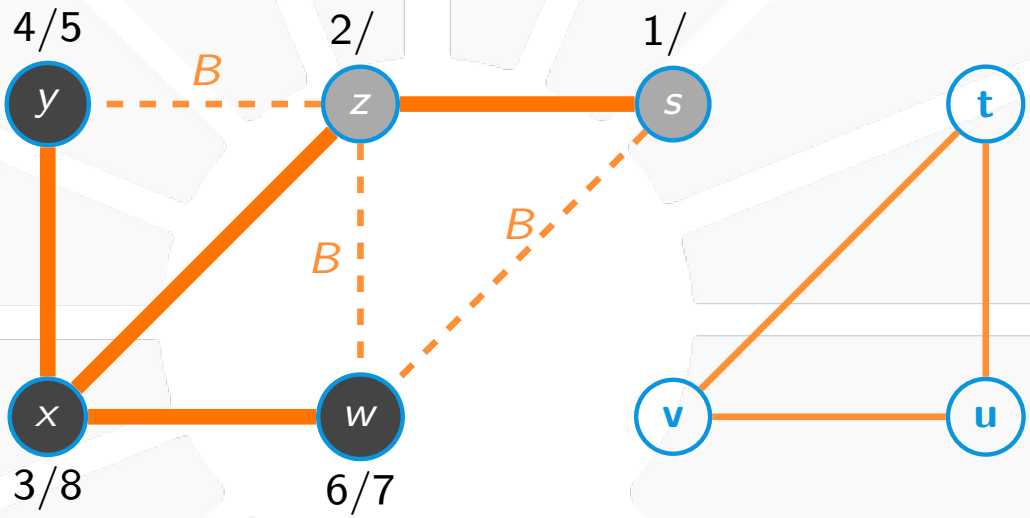


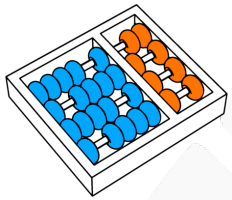
Executando DFS



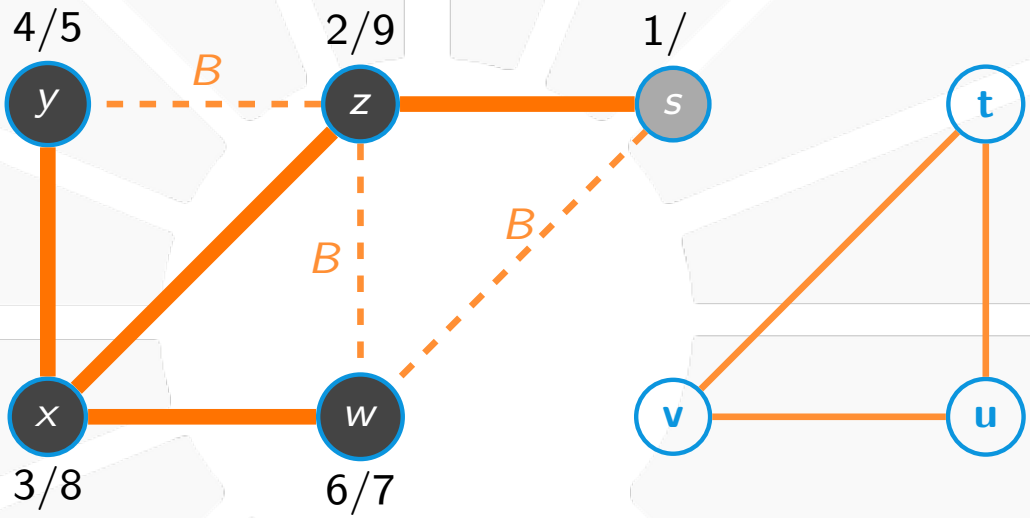


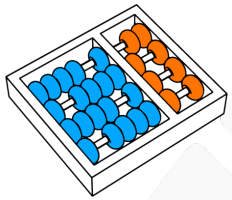
Executando DFS



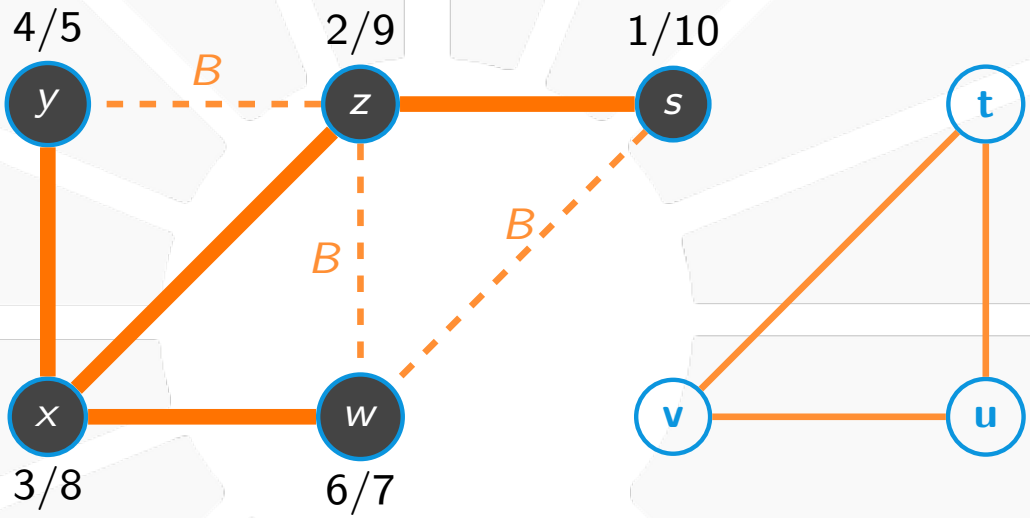


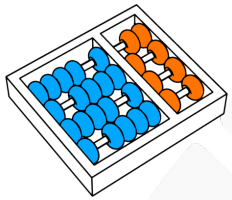
Executando DFS



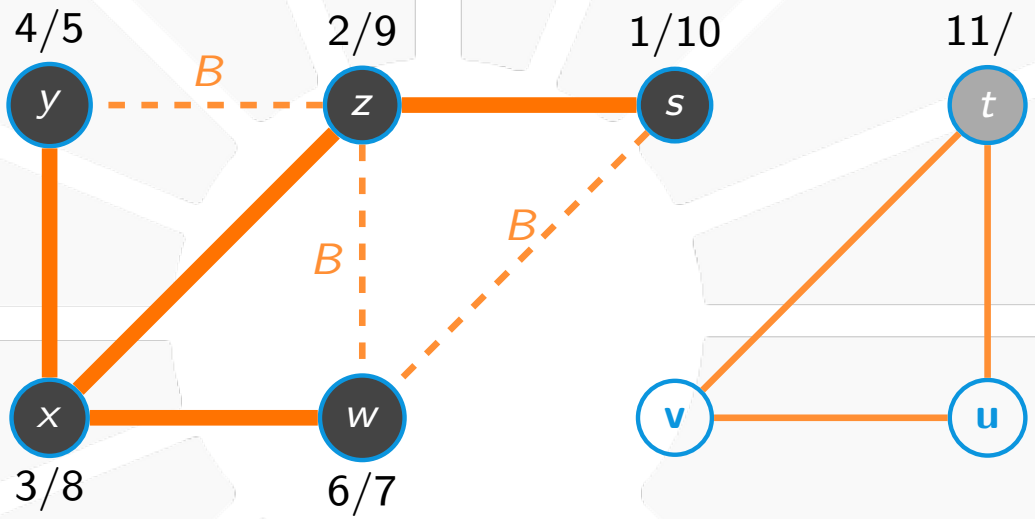


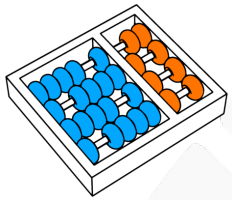
Executando DFS



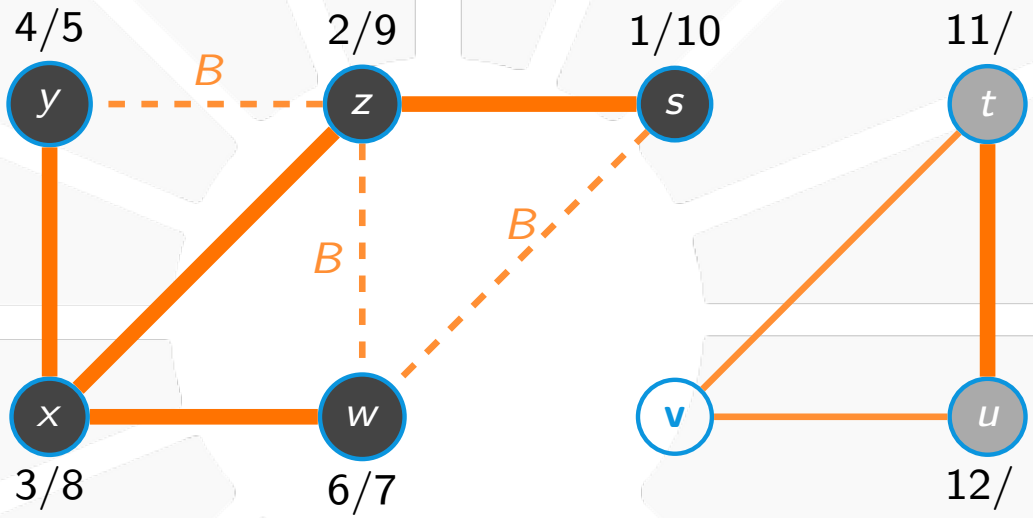


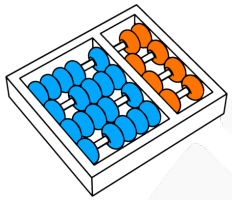
Executando DFS



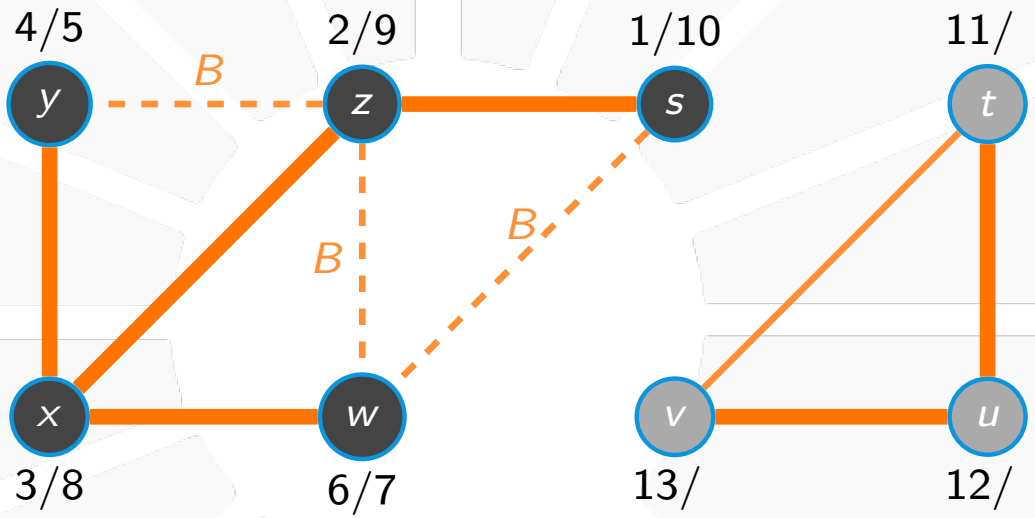


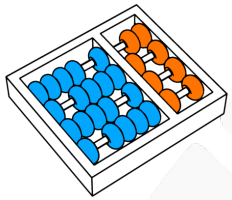
Executando DFS



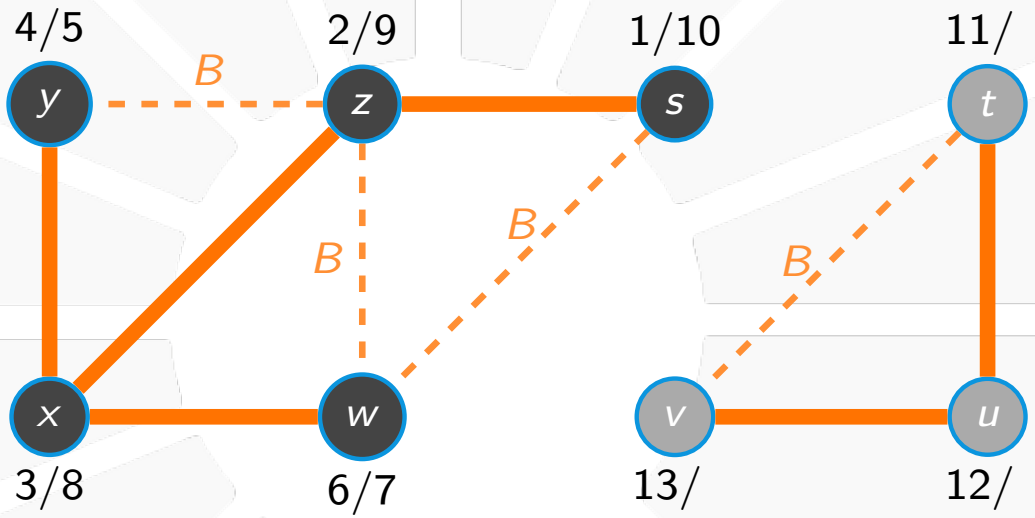


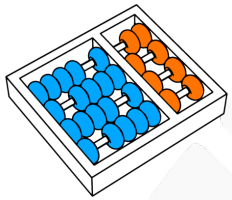
Executando DFS



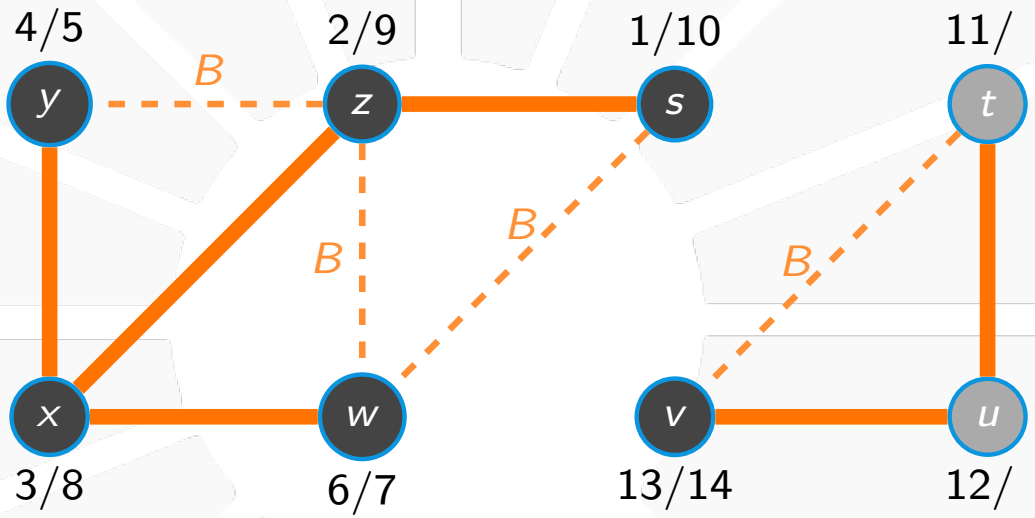


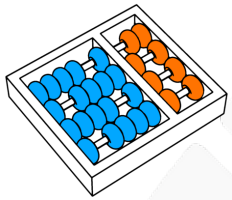
Executando DFS



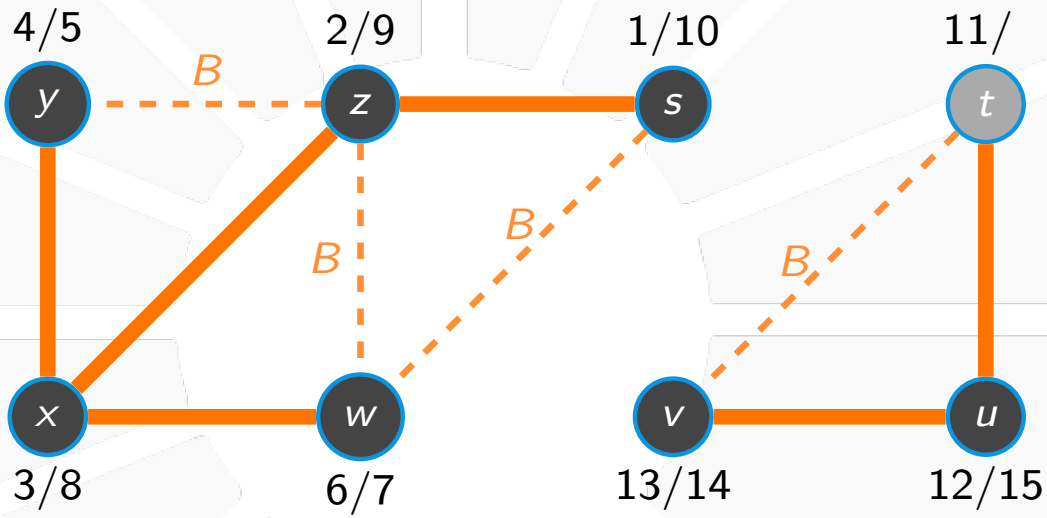


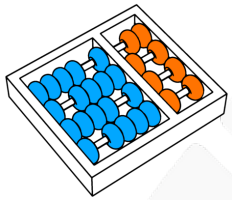
Executando DFS



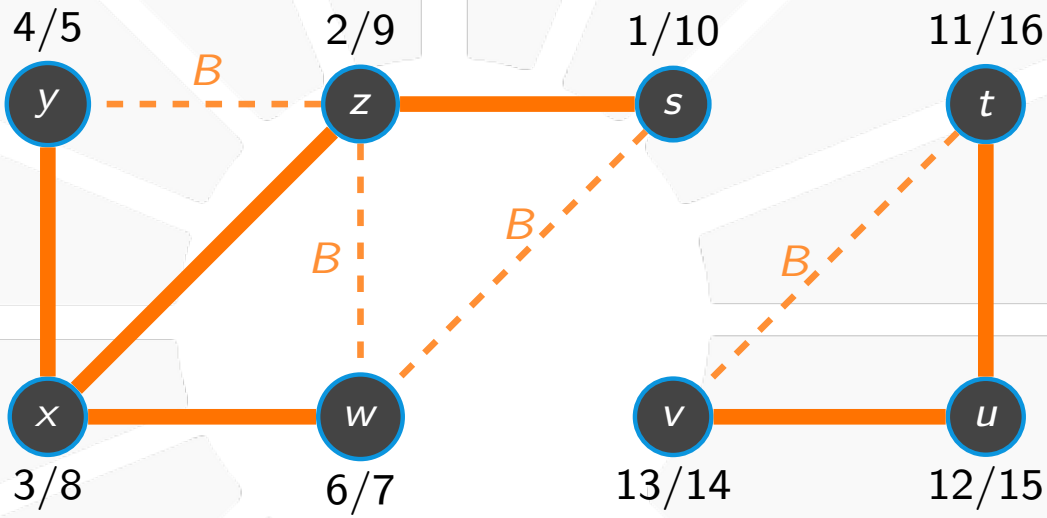


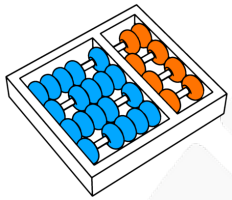
Executando DFS





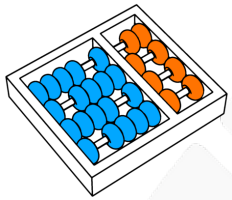
Executando DFS





Componentes conexas

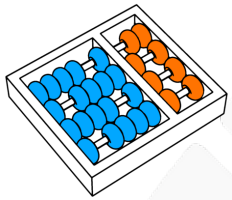
Contando o número de componentes:



Componentes conexas

Contando o número de componentes:

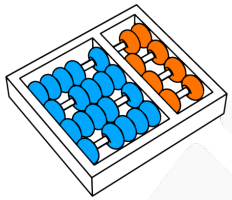
- Cada componente corresponde a uma árvore de busca.



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.

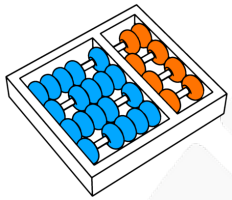


Componentes conexas

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:



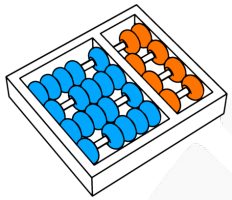
Componentes conexas

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.



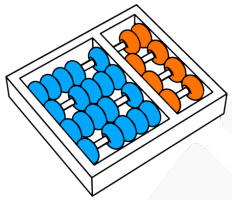
Componentes conexas

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:

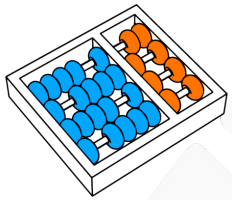
- Identificamos cada componente por um número.
- Denotaremos por *comp*[**v**] a componente de **v**.



Algoritmo DFS modificado

Algoritmo 1: $\mathrm{DFS}(G)$

```
1 para cada u ∈ V[G]
2     cor[u] ← branco
3 ℓ ← 0
4 para cada u ∈ V[G]
5     se cor[u] = branco
6         ℓ ← ℓ + 1
7         DFS-VISIT(u)
```

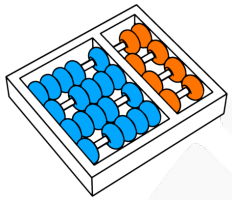


Algoritmo DFS modificado

Algoritmo 2: $\mathrm{DFS}(G)$

```
1 para cada u ∈ V[G]
2     cor[u] ← branco
3 ℓ ← 0
4 para cada u ∈ V[G]
5     se cor[u] = branco
6         ℓ ← ℓ + 1
7         DFS-VISIT(u)
```

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

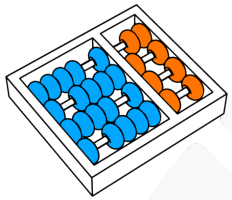


Algoritmo DFS-VISIT modificado

Algoritmo 3: DFS-VISIT(u)

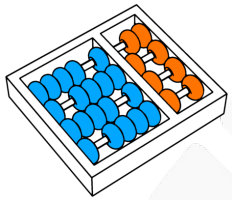
```
1 cor[u] ← cinza
2 para cada v ∈ Adj[u]
3     se cor[v] = branco
4         DFS-VISIT(v)
5 cor[u] ← preto
6 comp[u] ← ℓ
```

Ordenação Topológica



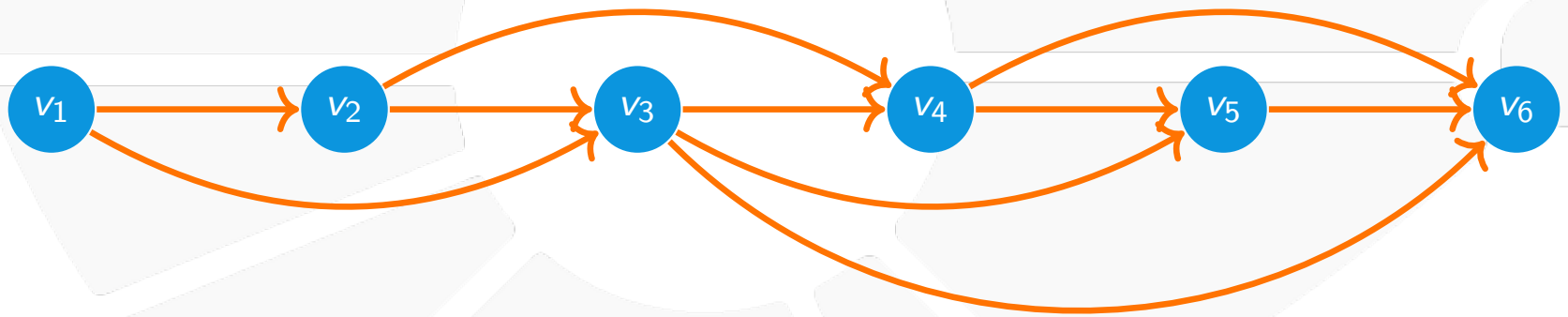
Definição

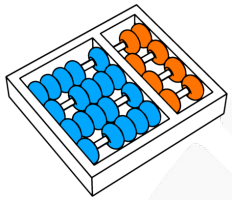
Uma **ORDENAÇÃO TOPOLÓGICA** de um grafo direcionado é um arranjo dos vértices $v_1, v_2, \ldots, v_n$ tal que se (v_i, v_j) é uma aresta do grafo, então $i < j$.



Definição

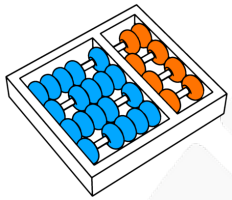
Uma **ORDENAÇÃO TOPOLÓGICA** de um grafo direcionado é um arranjo dos vértices $v_1, v_2, \ldots, v_n$ tal que se (v_i, v_j) é uma aresta do grafo, então $i < j$.





Exemplo de aplicação

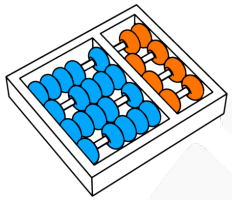
Representando dependências:



Exemplo de aplicação

Representando dependências:

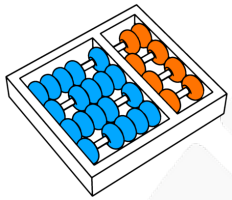
- Um grafo pode representar precedências entre tarefas.



Exemplo de aplicação

Representando dependências:

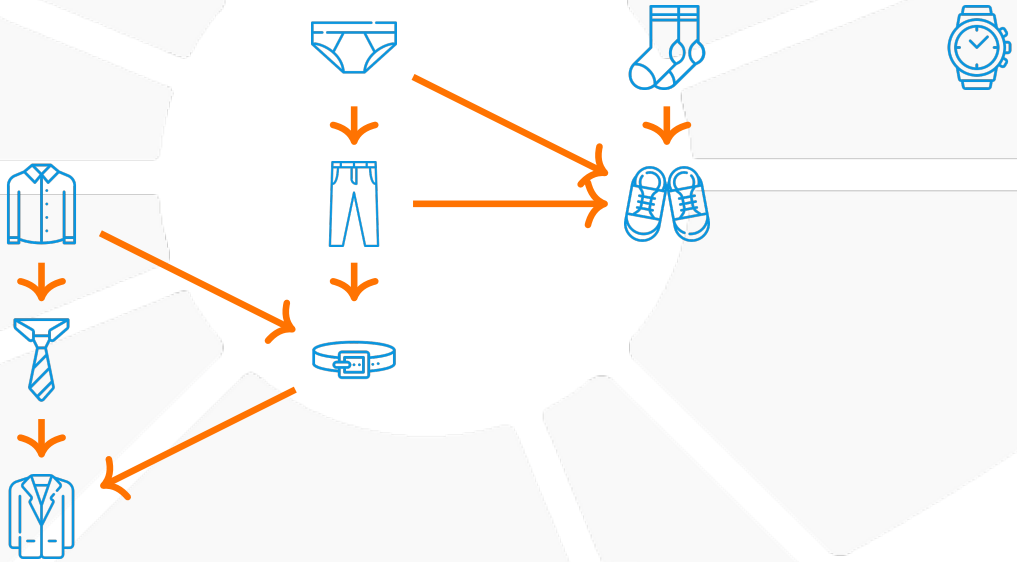
- Um grafo pode representar precedências entre tarefas.
- Queremos um ordem que respeita as precedências.

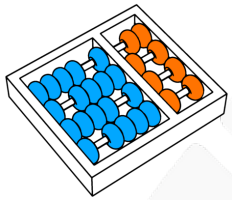


Exemplo de aplicação

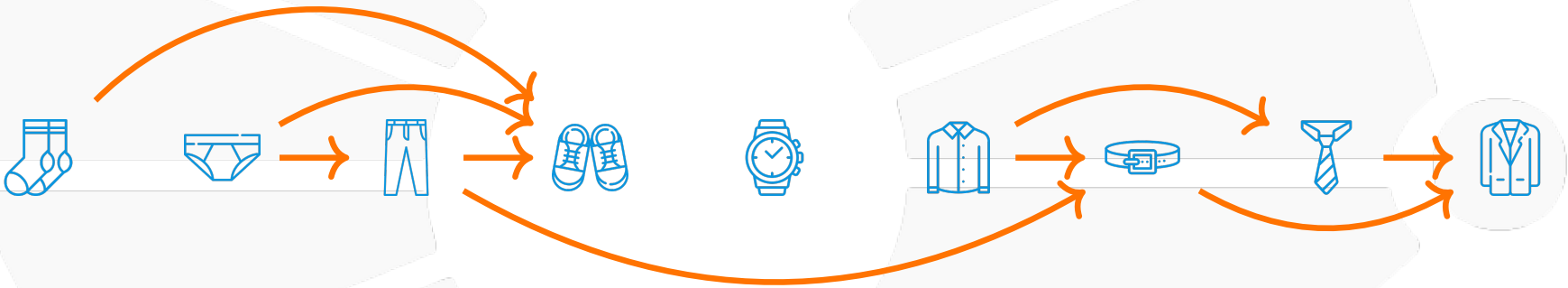
Representando dependências:

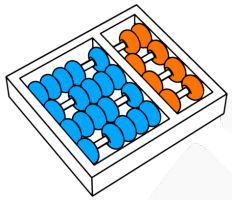
- Um grafo pode representar precedências entre tarefas.
- Queremos um ordem que respeita as precedências.





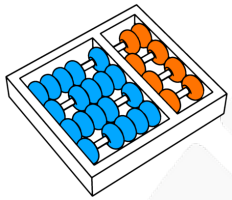
Exemplo de ordenação topológica





Condições de existência

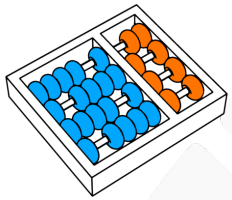
Todo grafo direcionado possui ordenação topológica?



Condições de existência

Todo grafo direcionado possui ordenação topológica?

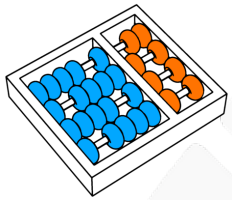
- **NÃO**, um ciclo direcionado não possui!



Condições de existência

Todo grafo direcionado possui ordenação topológica?

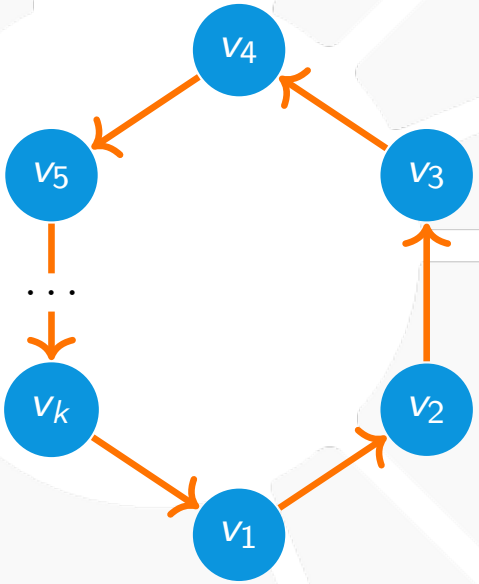
- **NÃO**, um ciclo direcionado não possui!
- Assim, nenhum grafo que contém um ciclo possui!

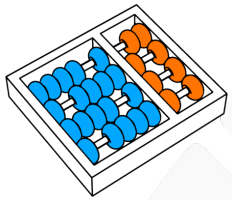


Condições de existência

Todo grafo direcionado possui ordenação topológica?

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

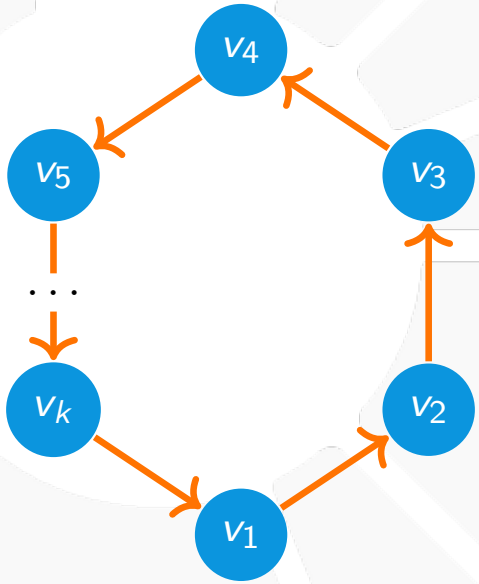




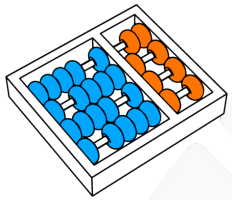
Condições de existência

Todo grafo direcionado possui ordenação topológica?

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



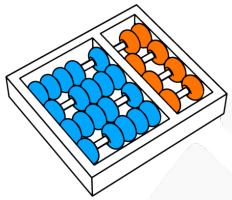
Um grafo direcionado é **ACÍCLICO** se não contiver um ciclo direcionado.



Condições de existência

Teorema

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

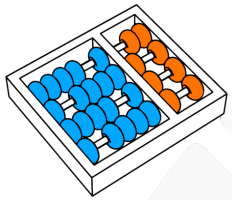


Condições de existência

Teorema

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

Demonstração:



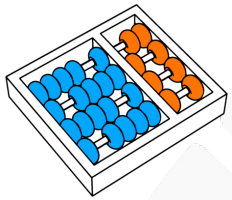
Condições de existência

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**.



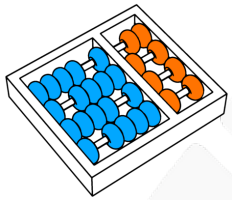
Condições de existência

Teorema

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

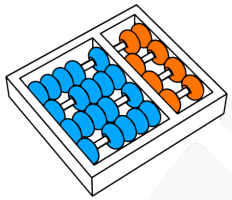
Demonstração:

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



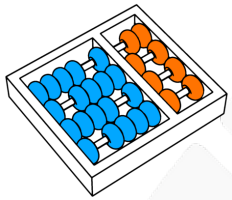
Um lema auxiliar

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



Um lema auxiliar

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

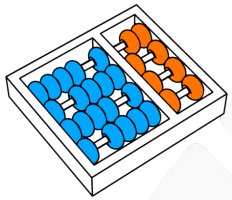


Um lema auxiliar

- 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**.*



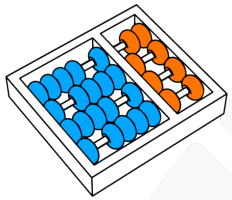
Um lema auxiliar

- 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:



Um lema auxiliar

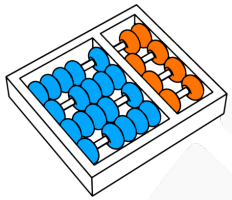
- Uma **FONTE** é um vértice com grau de entrada zero.
- Um **SORVEDOURO** é um vértice com grau de saída zero.

Lema

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

Demonstração:

- Tome um caminho maximal $P = (v_0 \dots v_k)$ em G.



Um lema auxiliar

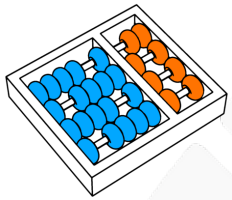
- Uma **FONTE** é um vértice com grau de entrada zero.
- Um **SORVEDOURO** é um vértice com grau de saída zero.

Lema

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

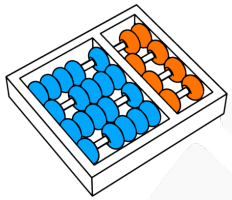
Demonstração:

- 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

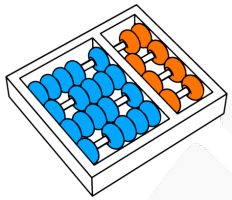
Considere um grafo acíclico $G = (\mathbf{V}, \mathbf{E})$.



Demonstração do teorema

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

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

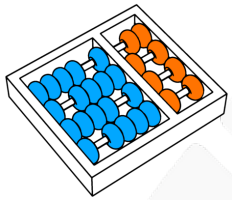


Demonstração do teorema

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.

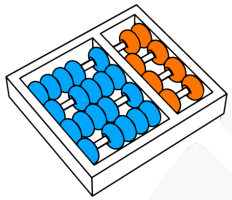


Demonstração do teorema

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:

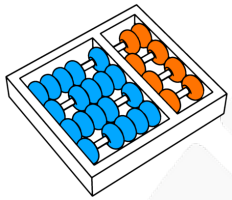


Demonstração do teorema

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 $\mathbf{u}$.

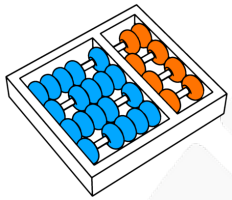


Demonstração do teorema

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 $\mathbf{v_1}, \ldots, \mathbf{v_{n-1}}$.

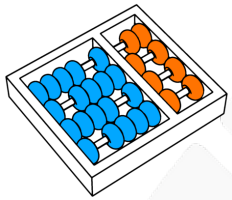


Demonstração do teorema

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

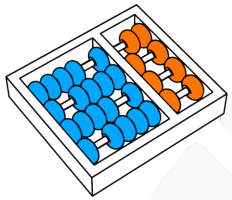
Mostraremos que G possui uma ordenação topológica por indução em $|\mathbf{V}|$:

- 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

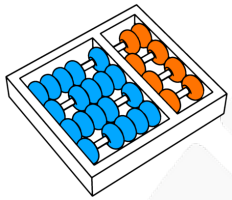
A demonstração anterior é construtiva:



Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

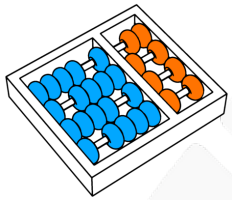
- É baseada em exibir uma ordenação topológica.



Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

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

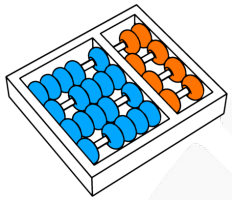


Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

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

Algoritmo para ordenação topológica:



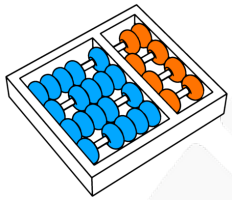
Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

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

Algoritmo para ordenação topológica:

1. Encontre uma fonte **u** de G.



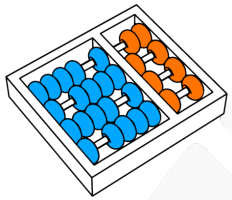
Encontrando uma ordenação topológica

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 $\mathbf{v_1, \ldots, v_{n-1}}$ de $G - \mathbf{u}$.



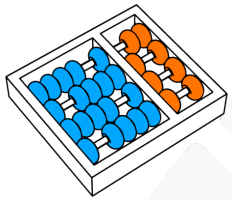
Encontrando uma ordenação topológica

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 $\mathbf{u}, \mathbf{v_1}, \ldots, \mathbf{v_{n-1}}$.



Encontrando uma ordenação topológica

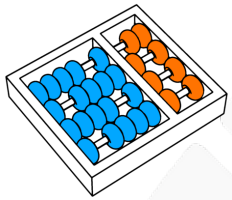
A demonstração anterior é construtiva:

- 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 $O(V^2)$:



Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

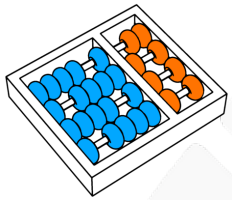
- É baseada em exibir uma ordenação topológica.
- Sugere um **ALGORITMO RECURSIVO**.

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 $O(V)$.



Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

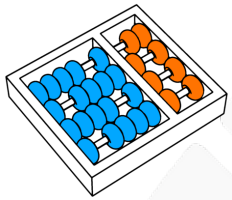
- É baseada em exibir uma ordenação topológica.
- Sugere um **ALGORITMO RECURSIVO**.

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.



Encontrando uma ordenação topológica

A demonstração anterior é construtiva:

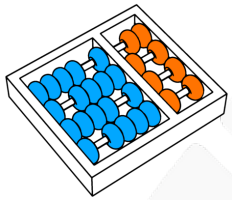
- É baseada em exibir uma ordenação topológica.
- Sugere um **ALGORITMO RECURSIVO**.

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 $\mathbf{u}, \mathbf{v_1}, \ldots, \mathbf{v_{n-1}}$.

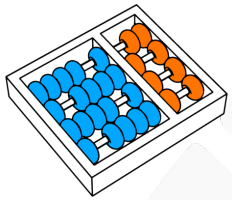
A complexidade desse algoritmo é $O(V^2)$:

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



Algoritmo baseado em DFS

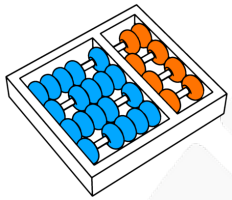
Considere um grafo direcionado acíclico:



Algoritmo baseado em DFS

Considere um grafo direcionado acíclico:

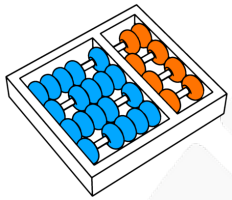
- Como não há ciclo, não existe aresta de retorno.



Algoritmo baseado em DFS

Considere um grafo direcionado acíclico:

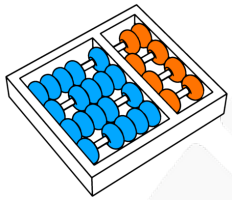
- Como não há ciclo, não existe aresta de retorno.
- Considere o instante em que **v** fica preto.



Algoritmo baseado em DFS

Considere um grafo direcionado acíclico:

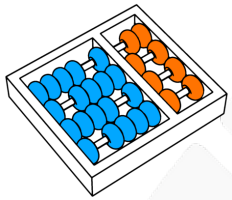
- Como não há ciclo, não existe aresta de retorno.
- Considere o instante em que **v** fica preto.
- Nesse instante **TODOS** seus vizinhos são pretos.



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.

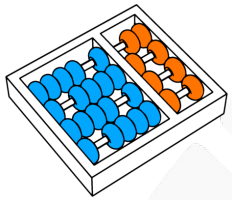


Algoritmo baseado em DFS

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:



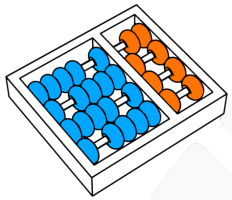
Algoritmo baseado em DFS

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.



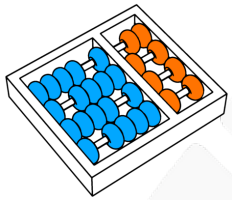
Algoritmo baseado em DFS

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.



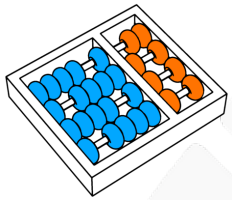
Algoritmo baseado em DFS

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.



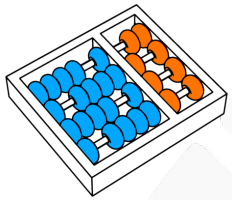
Algoritmo baseado em DFS

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:

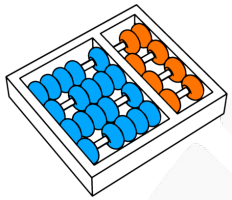
- O primeiro vértice a ficar preto não tem arestas saindo.
- O segundo só pode ter arestas para o primeiro.
- O terceiro só pode ter arestas para os dois primeiros.
- etc.



Algoritmo Topological-Sort

Algoritmo 4: Topological-Sort(u)

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

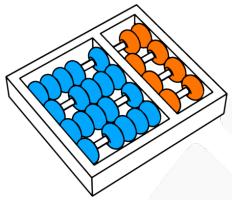


Algoritmo Topological-Sort

Algoritmo 5: Topological-Sort(u)

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

- Inserir cada um dos $|V|$ vértices leva tempo $O(1)$.

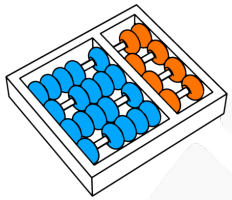


Algoritmo Topological-Sort

Algoritmo 6: Topological-Sort(u)

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

- Inserir cada um dos $|V|$ vértices leva tempo $O(1)$.
- Executamos DFS uma vez.

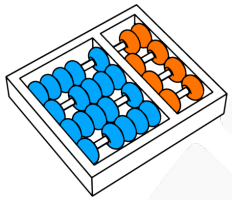


Algoritmo Topological-Sort

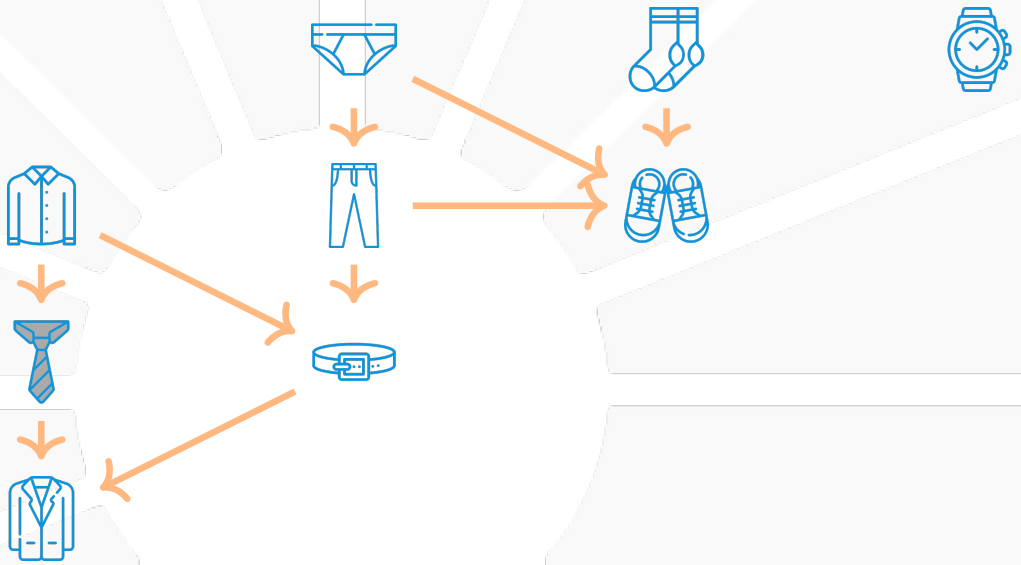
Algoritmo 7: Topological-Sort(u)

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

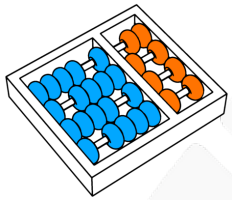
- Inserir cada um dos $|V|$ vértices leva tempo $O(1)$.
- Executamos DFS uma vez.
- Portanto, a complexidade de tempo é $O(V + E)$.



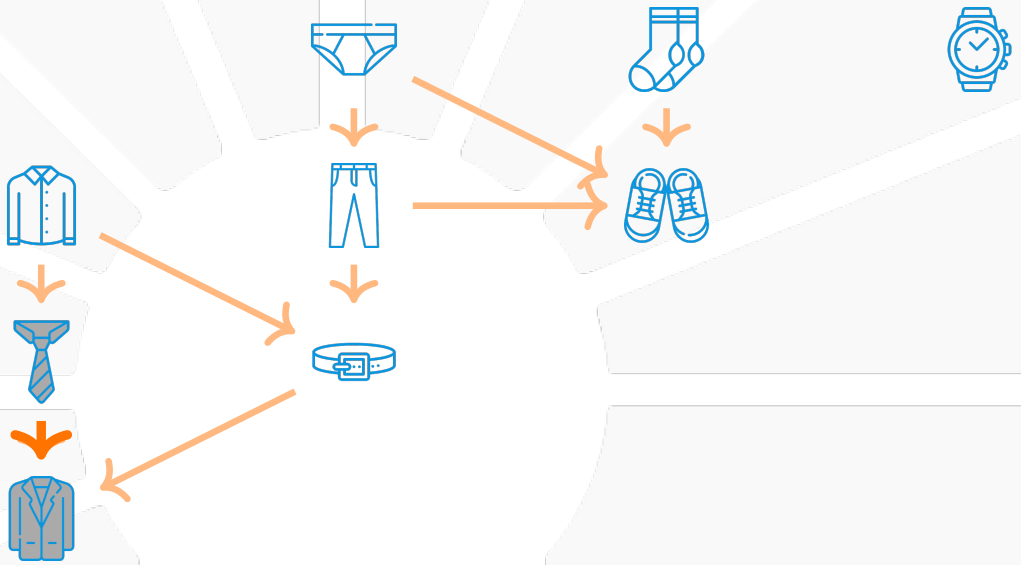
Exemplo



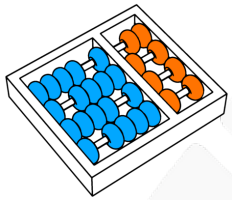
Lista:



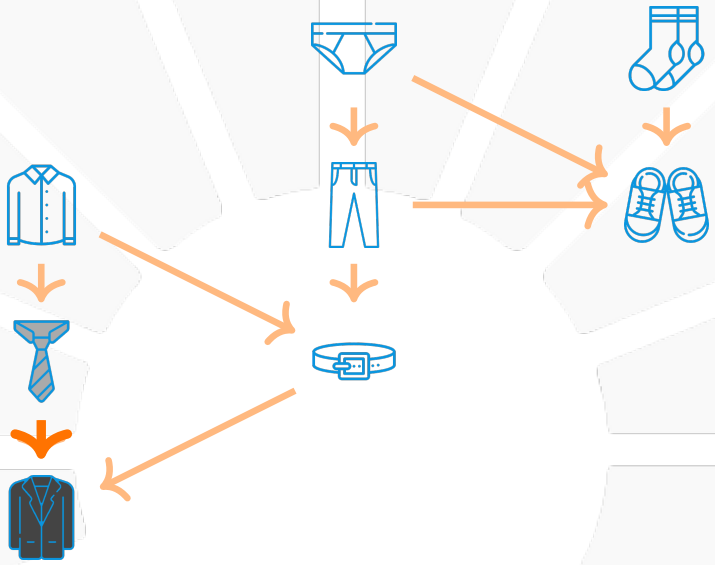
Exemplo



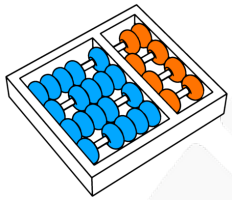
Lista:



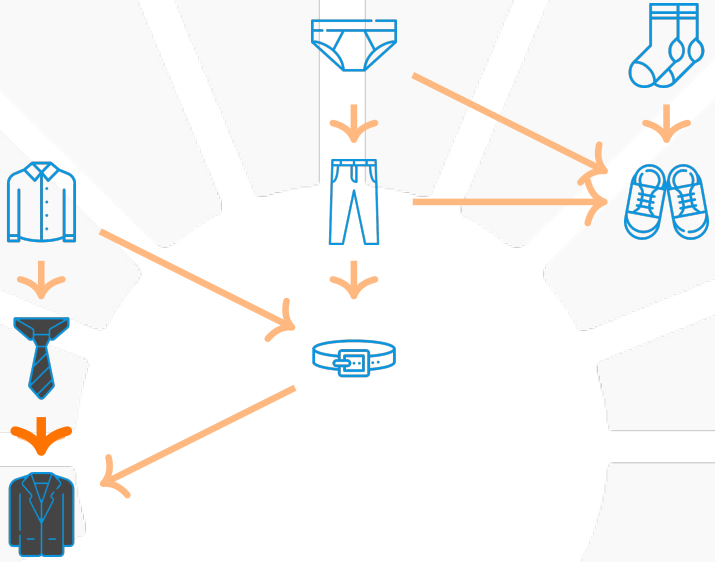
Exemplo



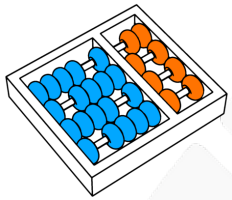
Lista:



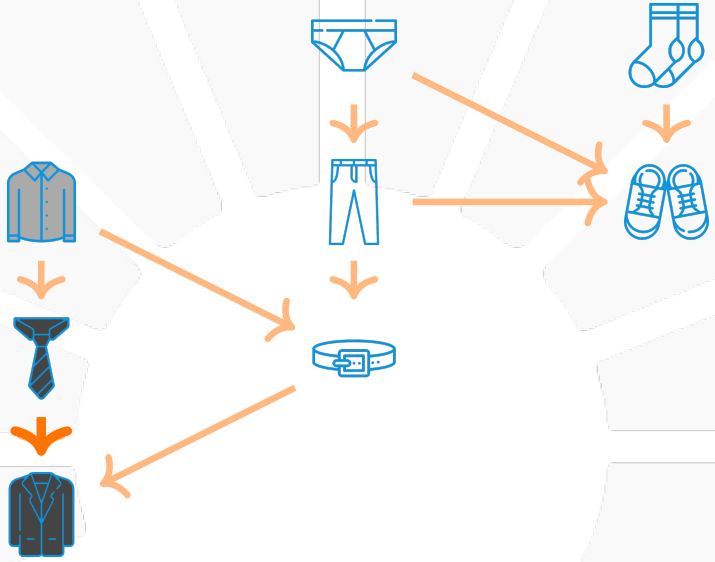
Exemplo



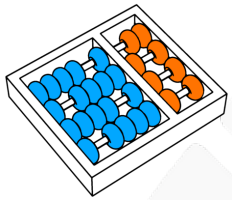
Lista:



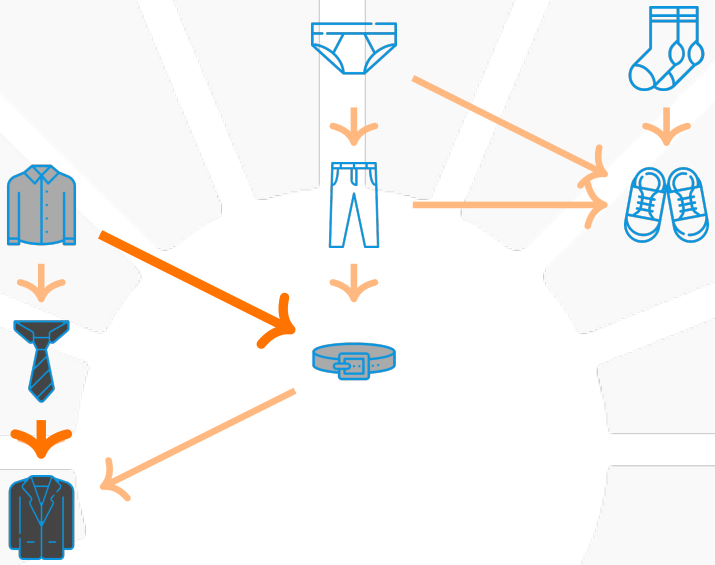
Exemplo



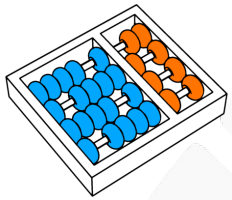
Lista:



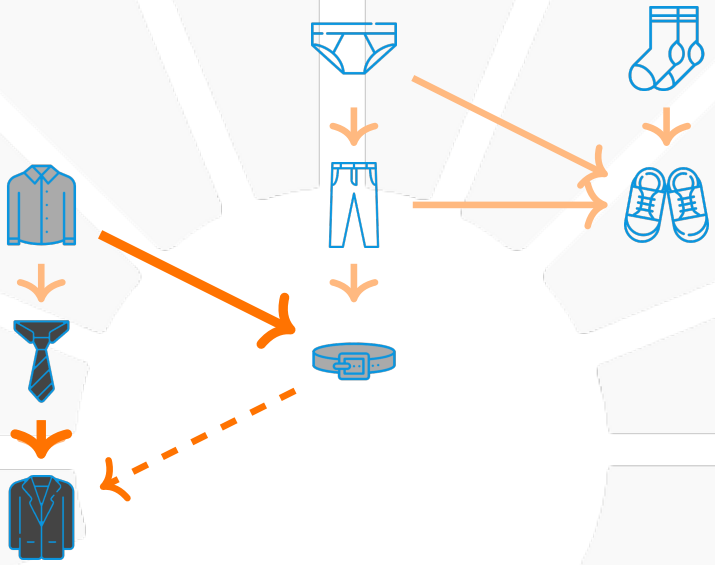
Exemplo



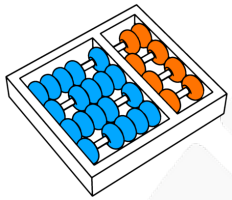
Lista:



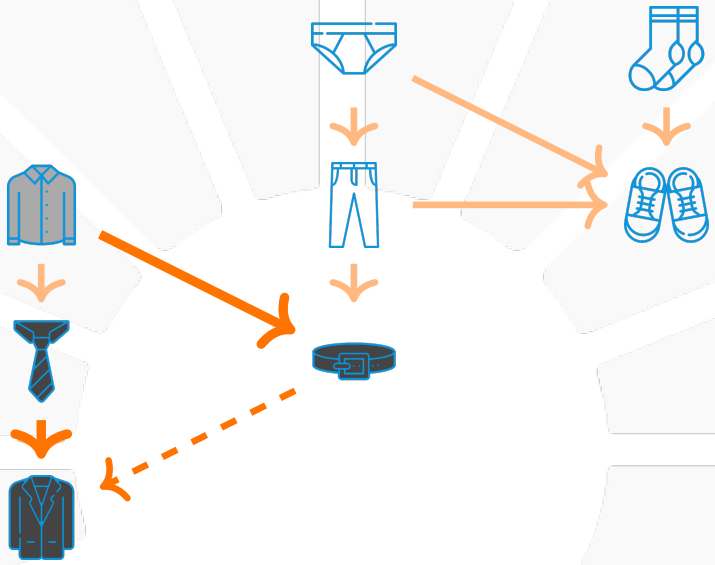
Exemplo



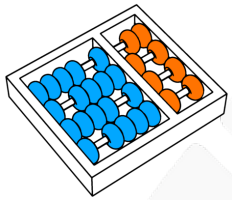
Lista:



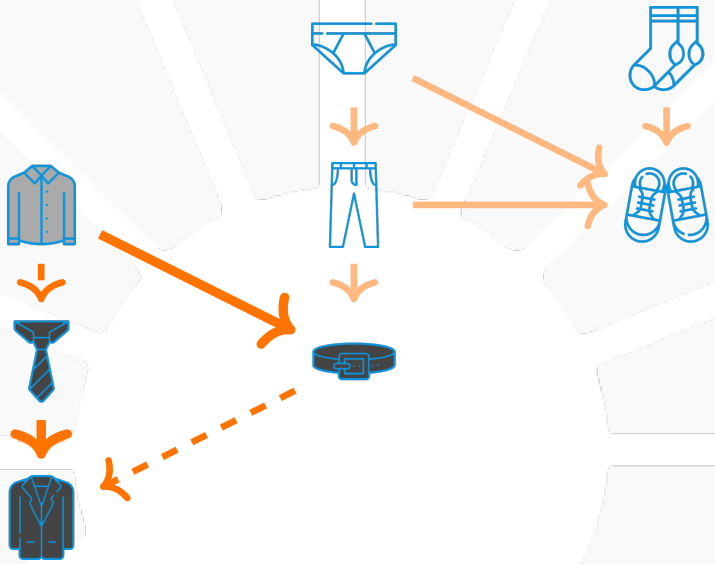
Exemplo



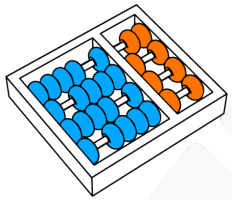
Lista:



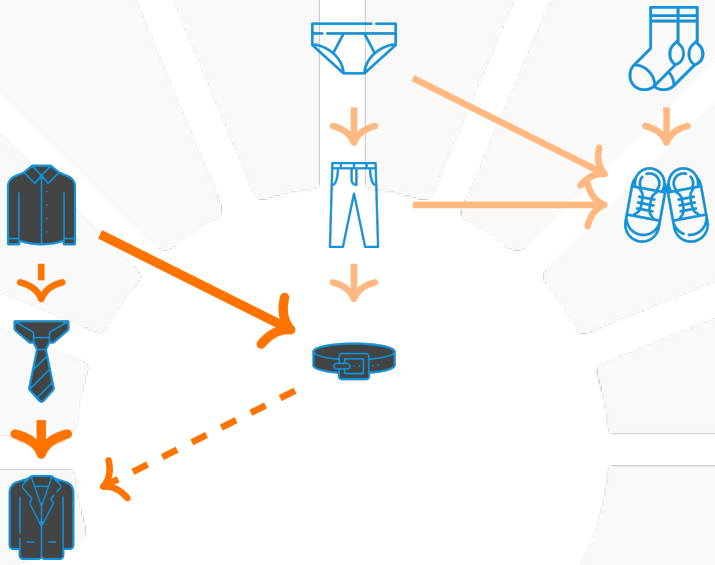
Exemplo



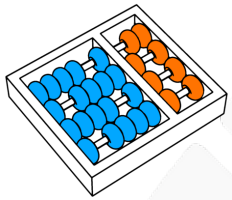
Lista:



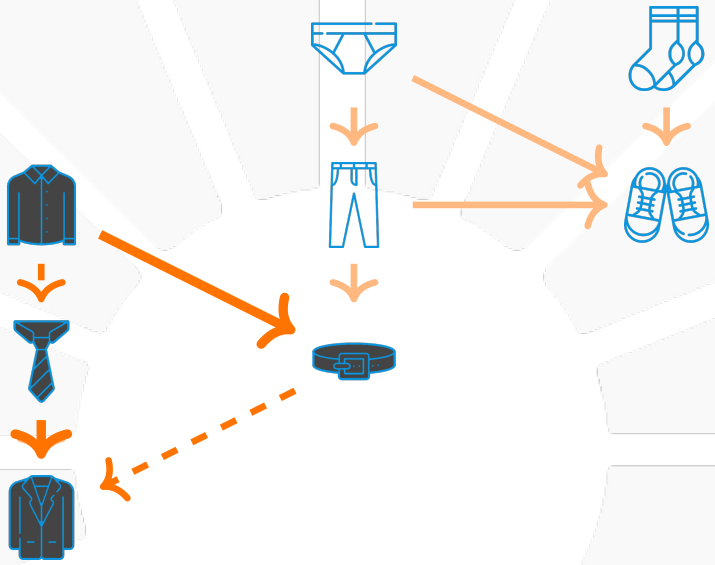
Exemplo



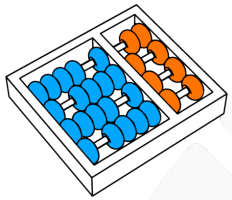
Lista:



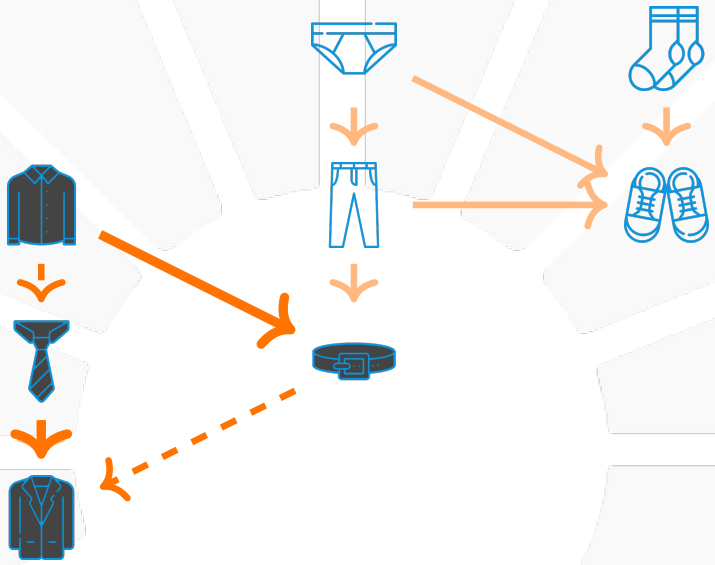
Exemplo



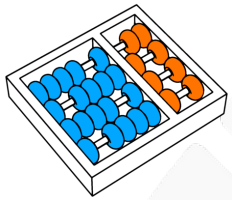
Lista:



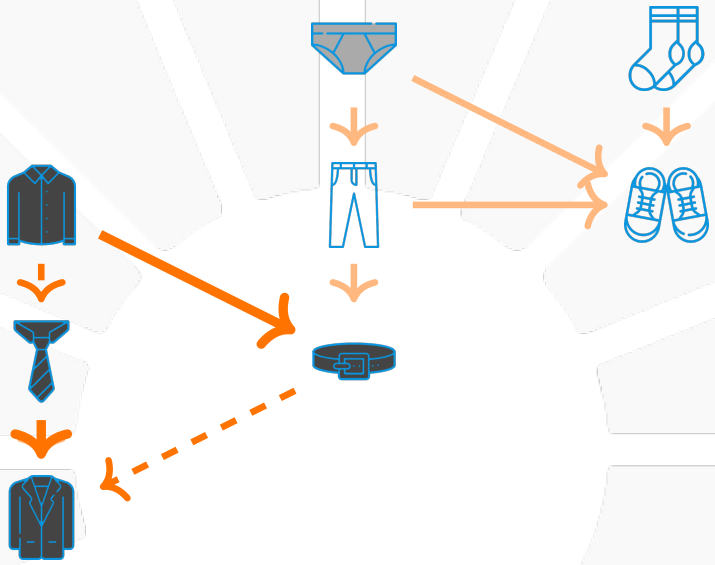
Exemplo



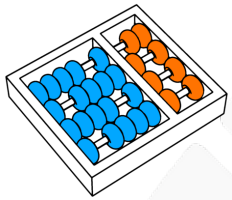
Lista:



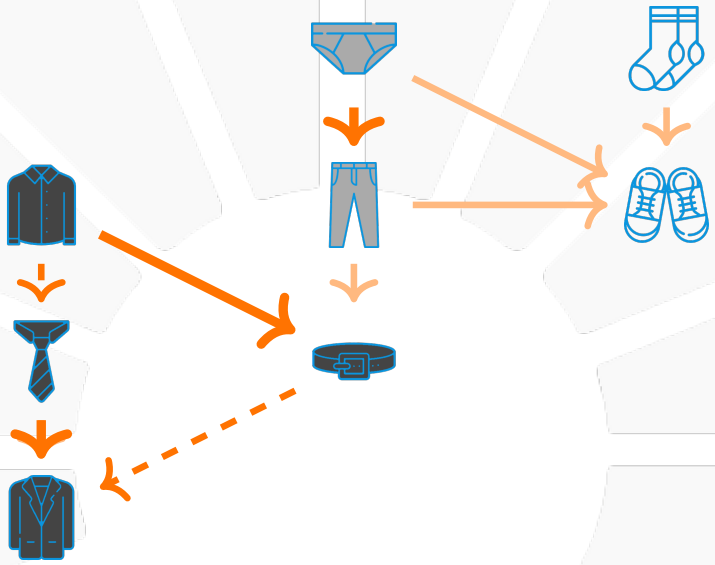
Exemplo



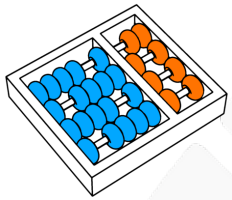
Lista:



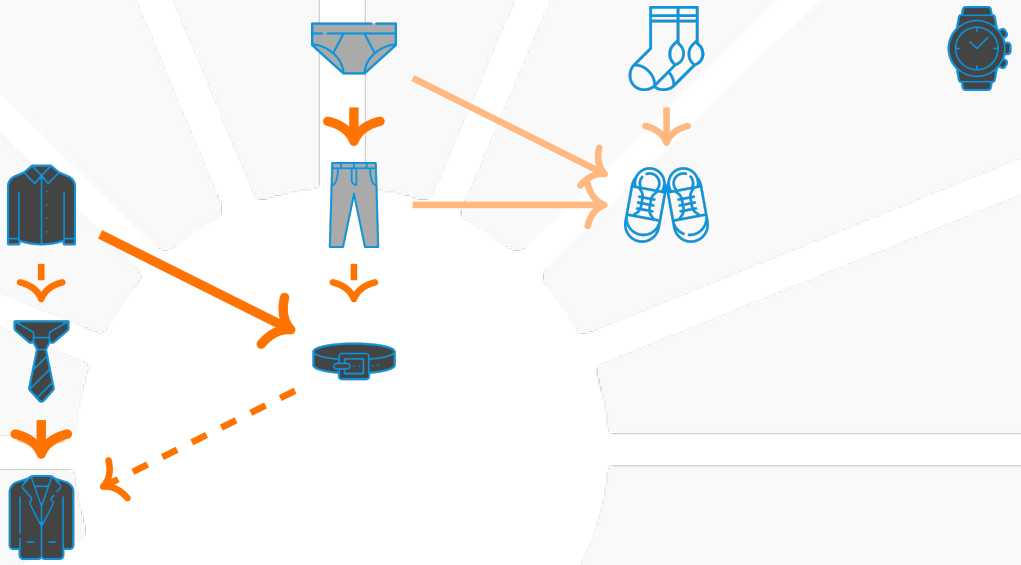
Exemplo



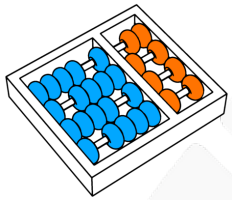
Lista:



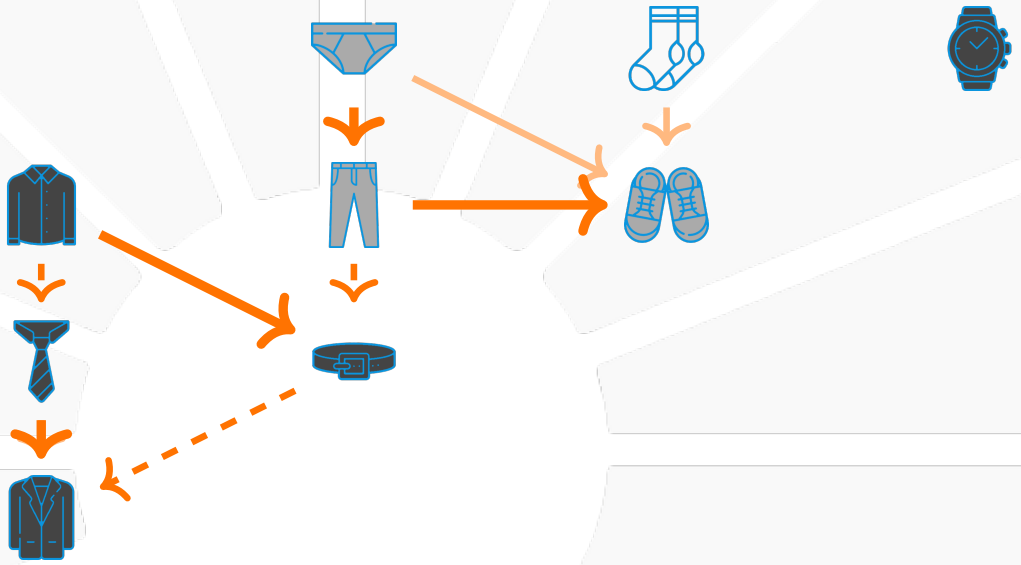
Exemplo



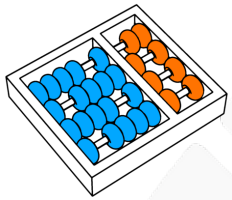
Lista:



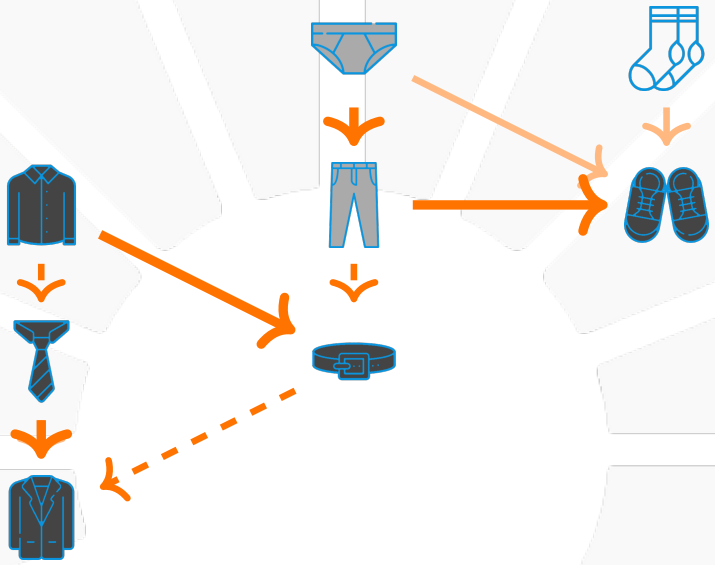
Exemplo



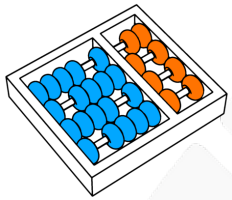
Lista:



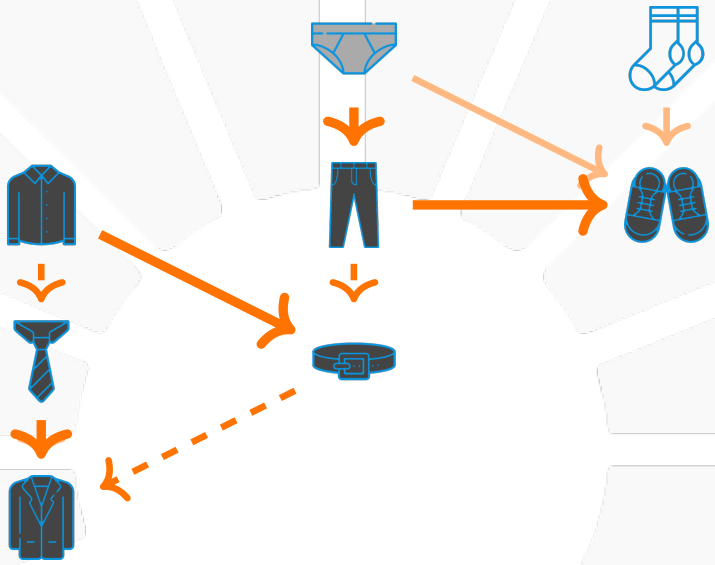
Exemplo



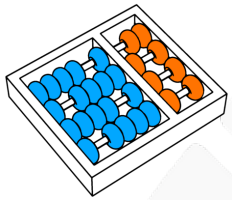
Lista:



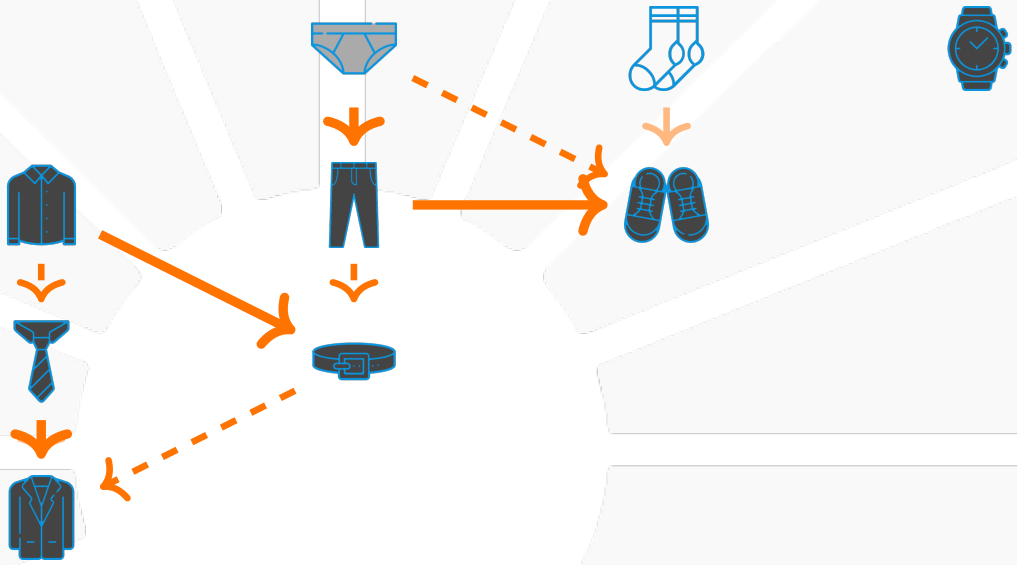
Exemplo



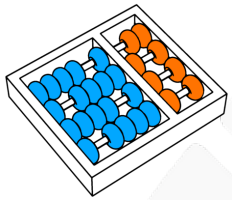
Lista:



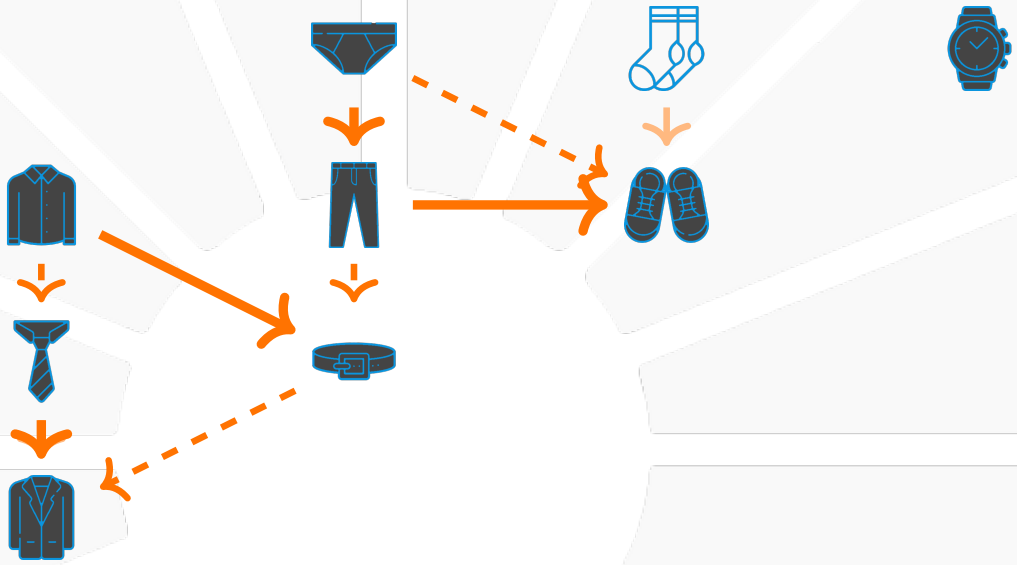
Exemplo



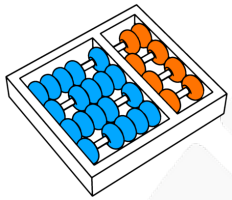
Lista:



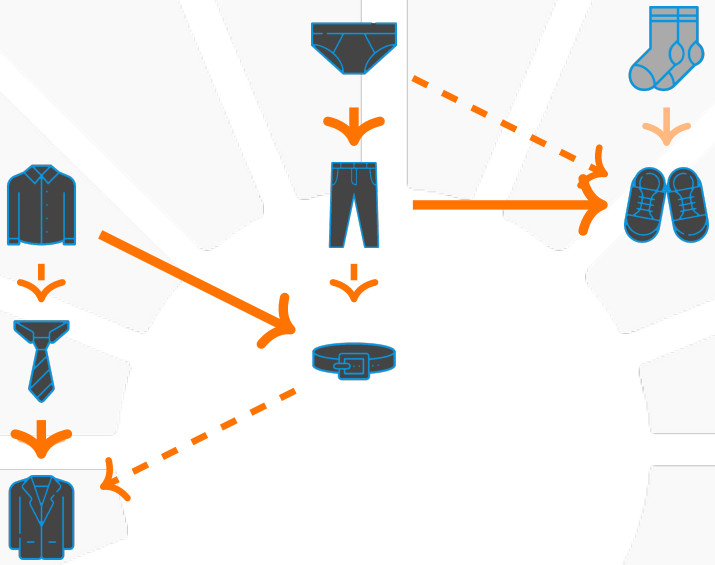
Exemplo



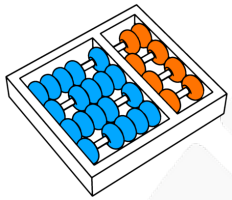
Lista:



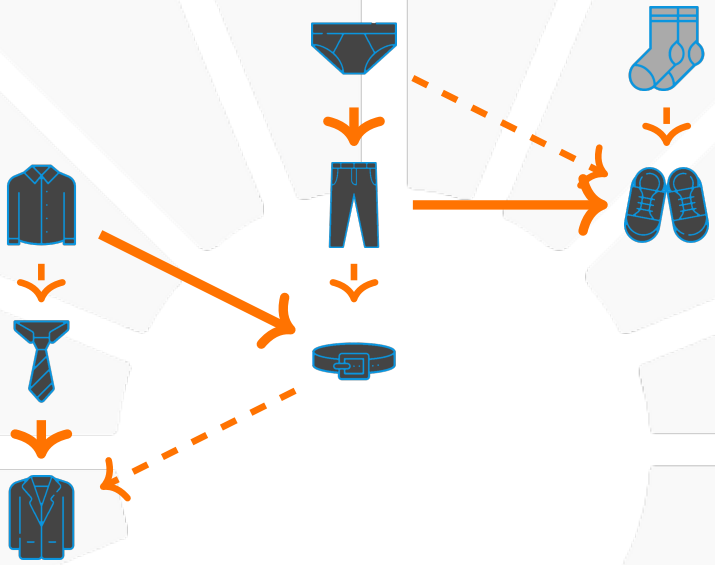
Exemplo



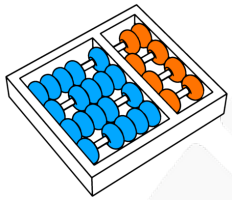
Lista:



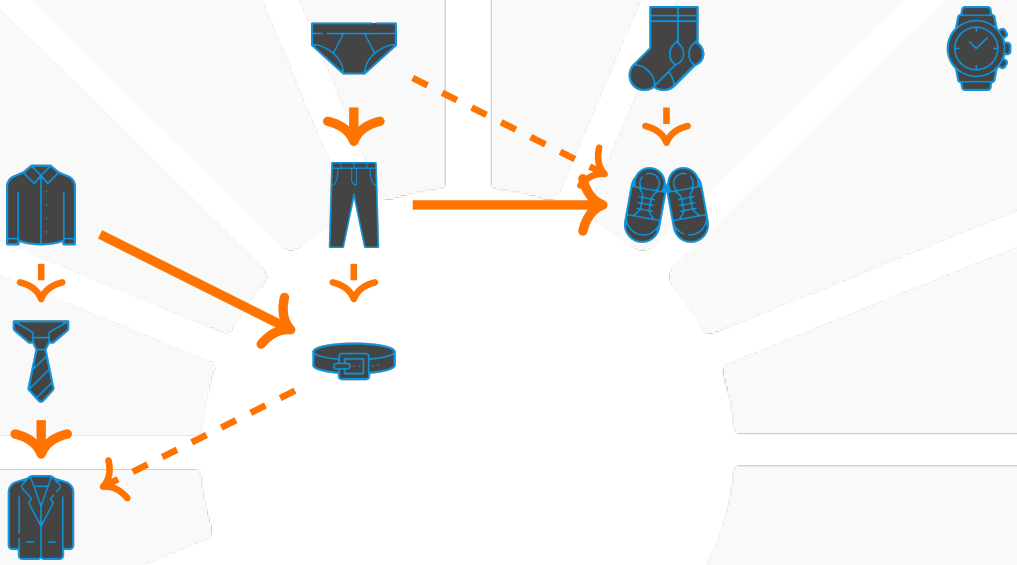
Exemplo



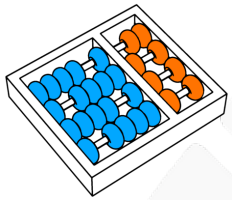
Lista:



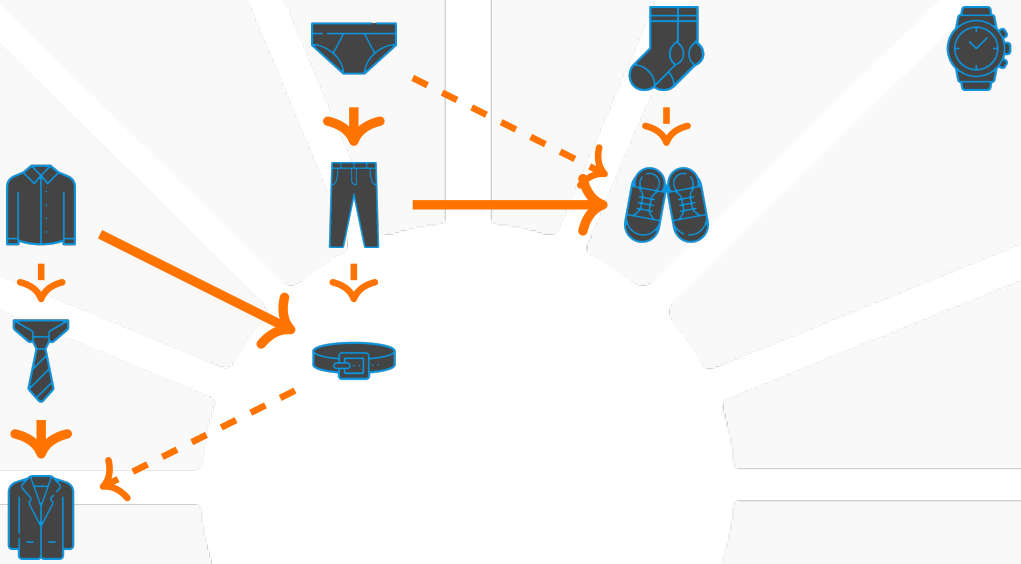
Exemplo



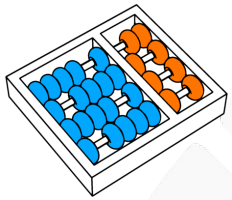
Lista:



Exemplo



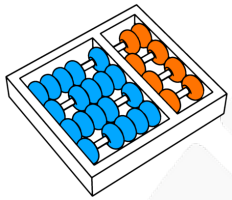
Lista:



Correção

Teorema

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

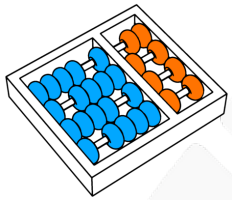


Correção

Teorema

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

Demonstração:



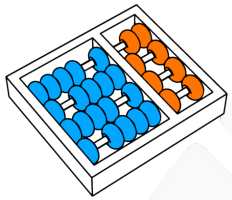
Correção

Teorema

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

Demonstração:

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



Correção

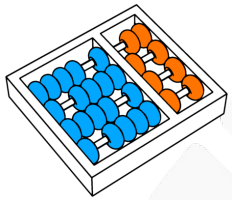
Teorema

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 $f[\mathbf{u}] > f[\mathbf{v}]$.



Correção

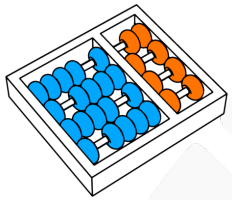
Teorema

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[**v**], basta mostrar que f[**u**] $>$ f[**v**].
- Considere o instante em que **(u,v)** foi examinada:



Correção

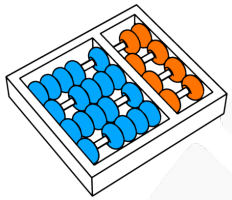
Teorema

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:



Correção

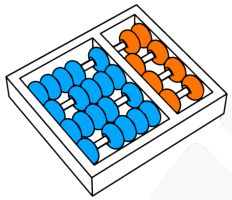
Teorema

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 $f[\mathbf{u}] > f[\mathbf{v}]$.



Correção

Teorema

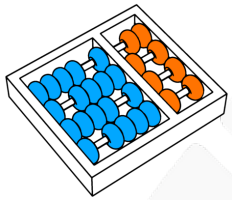
TOPOLOGICAL-SORT(G) *devolve uma ordenação topológica de* G.

Demonstração:

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

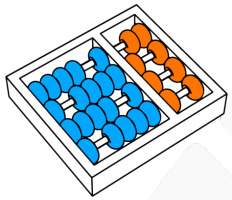
- Como a lista devolvida está em ordem **DECRESCENTE** de $f[\mathbf{v}]$, basta mostrar que $f[\mathbf{u}] > f[\mathbf{v}]$.
- Considere o instante em que **(u,v)** foi examinada:
- Como **(u,v)** não é aresta de retorno, **v** não pode ser cinza:
  1. Se **v** for branco, então ele será descendente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

Componentes fortemente conexas



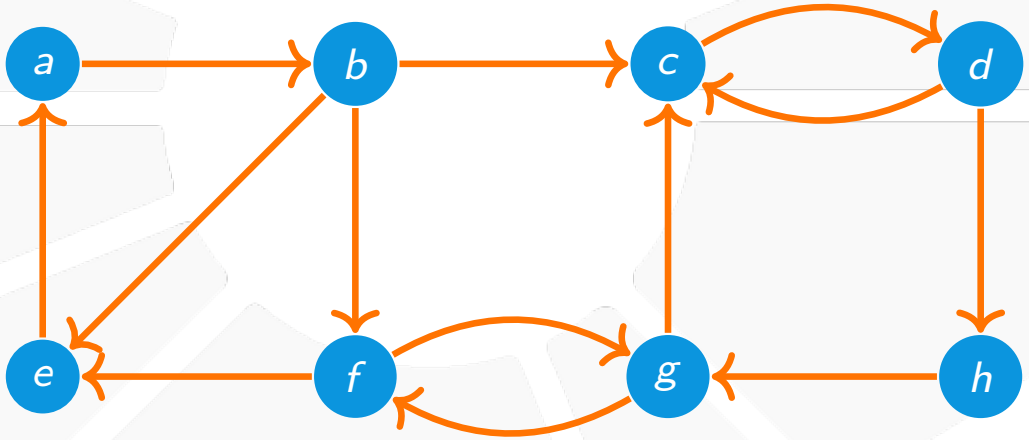
Grafo fortemente conexo

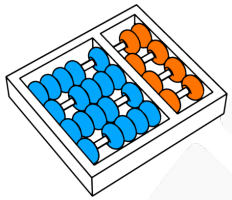
Um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é **FORTEMENTE CONEXO** se para todo par de vértices **u,v** de G, existe um caminho direcionado de **u** a **v**.



Grafo fortemente conexo

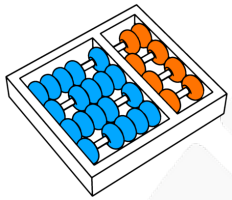
Um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é **FORTEMENTE CONEXO** se para todo par de vértices **u,v** de G, existe um caminho direcionado de **u** a **v**.





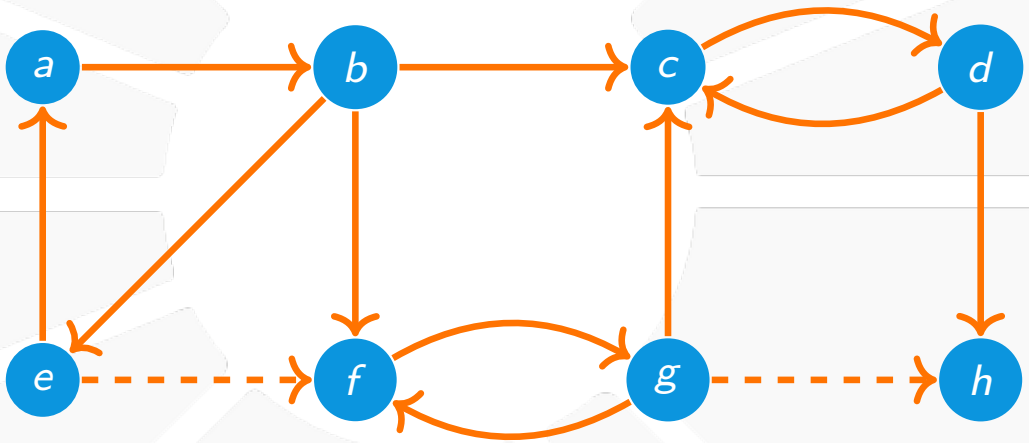
Grafo fortemente conexo

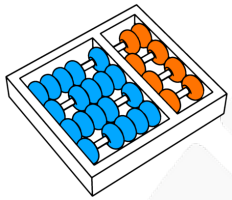
Nem todo grafo direcionado é fortemente conexo:



Grafo fortemente conexo

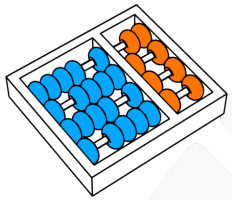
Nem todo grafo direcionado é fortemente conexo:





Componente fortemente conexa

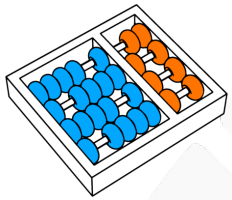
Uma **COMPONENTE FORTEMENTE CONEXA** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um subconjunto de vértices $\mathbf{C} \subseteq \mathbf{V}$ tal que:



Componente fortemente conexa

Uma **COMPONENTE FORTEMENTE CONEXA** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um subconjunto de vértices $\mathbf{C} \subseteq \mathbf{V}$ tal que:

(1) O subgrafo induzido por **C** é fortemente conexo e

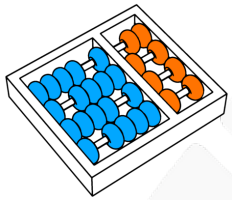


Componente fortemente conexa

Uma **COMPONENTE FORTEMENTE CONEXA** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um subconjunto de vértices $\mathbf{C} \subseteq \mathbf{V}$ tal que:

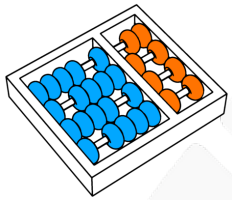
(1) O subgrafo induzido por $\mathbf{C}$ é fortemente conexo e

(2) $\mathbf{C}$ é maximal com respeito à propriedade (1).



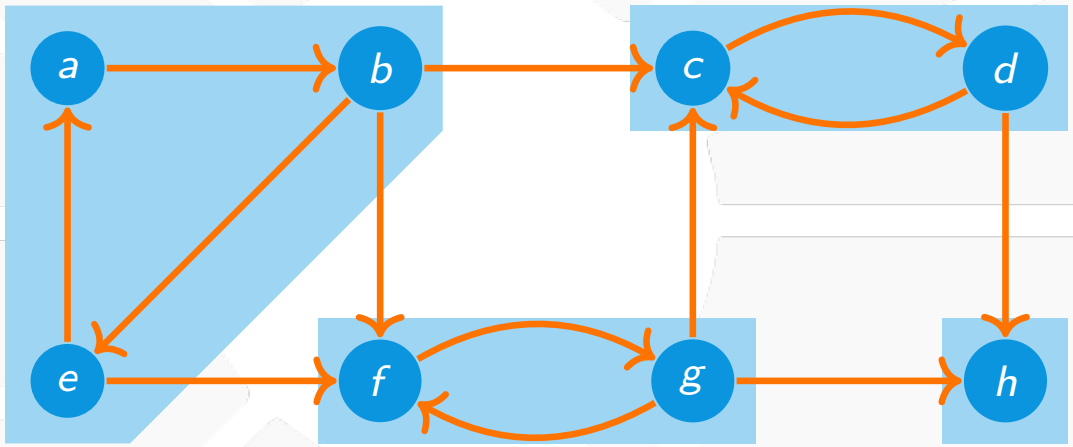
Componente fortemente conexa

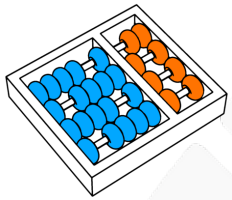
Podemos **PARTICIONAR** um grafo direcionado em componentes fortemente conexas.



Componente fortemente conexa

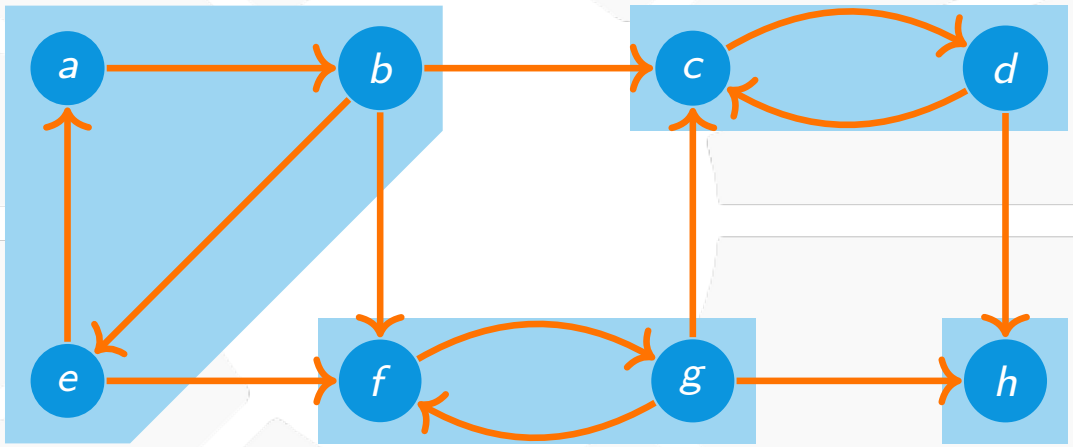
Podemos **PARTICIONAR** um grafo direcionado em componentes fortemente conexas.



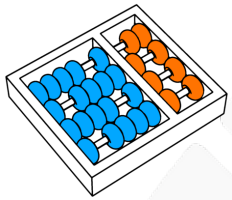


Componente fortemente conexa

Podemos **PARTICIONAR** um grafo direcionado em componentes fortemente conexas.

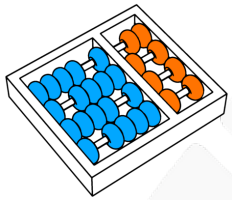


Como encontrar as componentes fortemente conexas?



Grafo componente

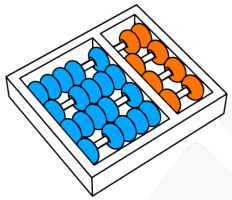
O **GRAFO COMPONENTE** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{CFC} em que:



Grafo componente

O **GRAFO COMPONENTE** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{CFC} em que:

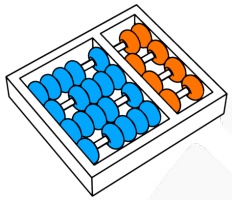
- Cada vértice é uma componente fortemente conexa.



Grafo componente

O **GRAFO COMPONENTE** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{CFC} em que:

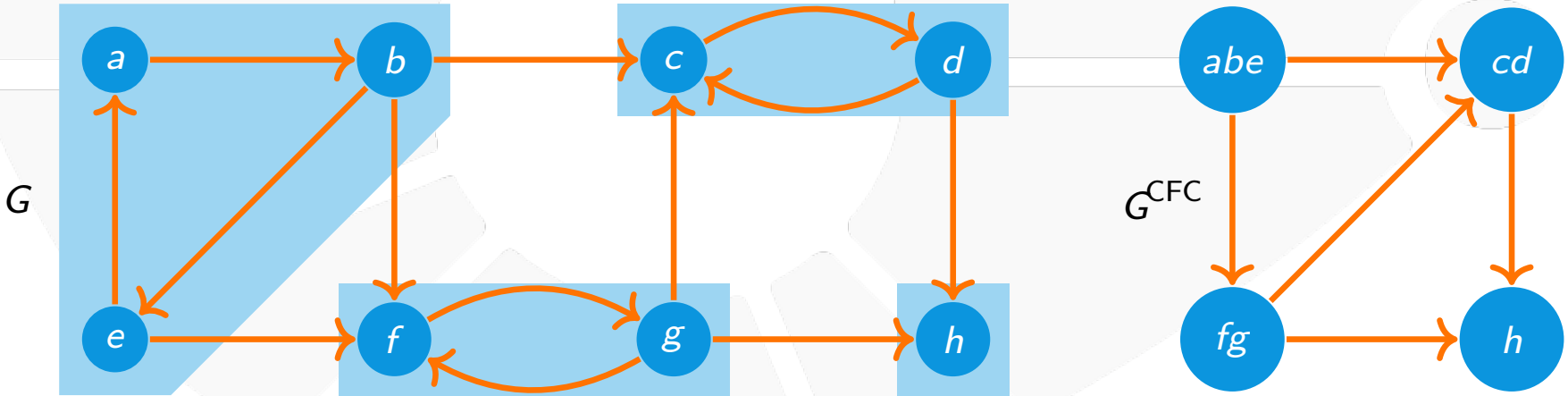
- Cada vértice é uma componente fortemente conexa.
- Existe aresta **(C,D)** se houver **(u,v)** $\in$ **E** com **u** $\in$ **C** e **v** $\in$ **D**.

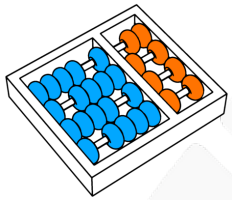


Grafo componente

O **GRAFO COMPONENTE** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{CFC} em que:

- Cada vértice é uma componente fortemente conexa.
- Existe aresta **(C,D)** se houver **(u,v)** $\in$ **E** com **u** $\in$ **C** e **v** $\in$ **D**.

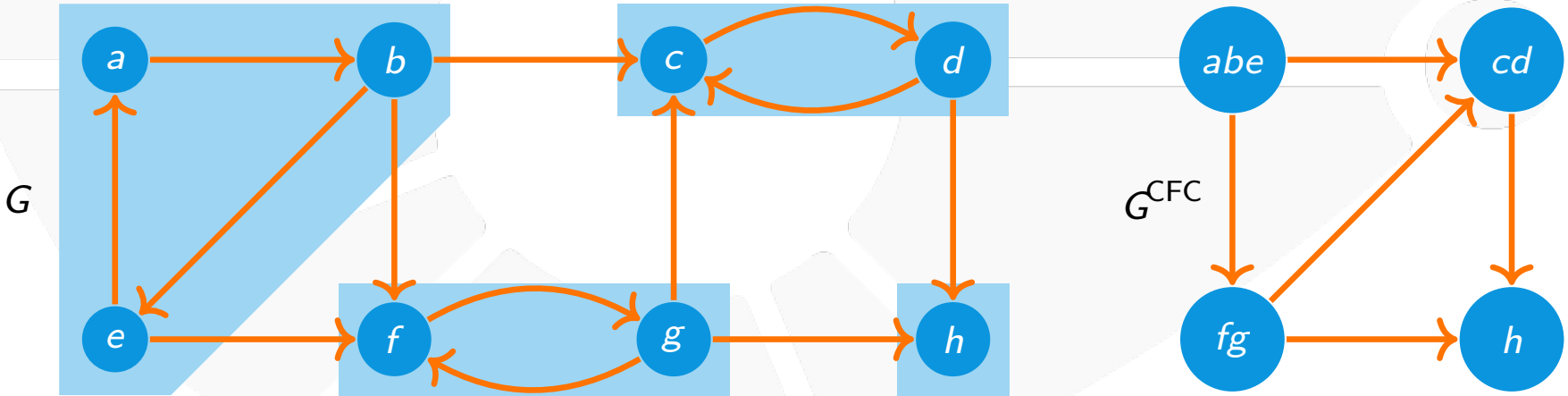




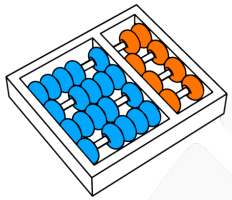
Grafo componente

O **GRAFO COMPONENTE** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{CFC} em que:

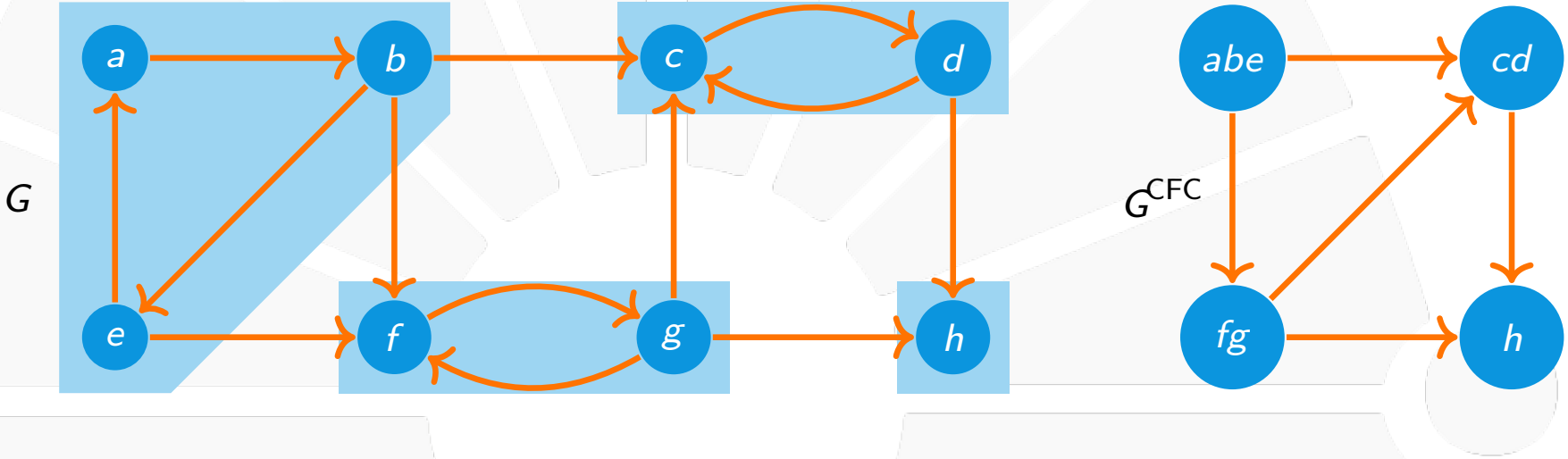
- Cada vértice é uma componente fortemente conexa.
- Existe aresta **(C,D)** se houver **(u,v)** $\in$ **E** com **u** $\in$ **C** e **v** $\in$ **D**.

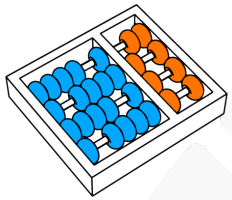


Note que G^{CFC} é acíclico. Por quê?

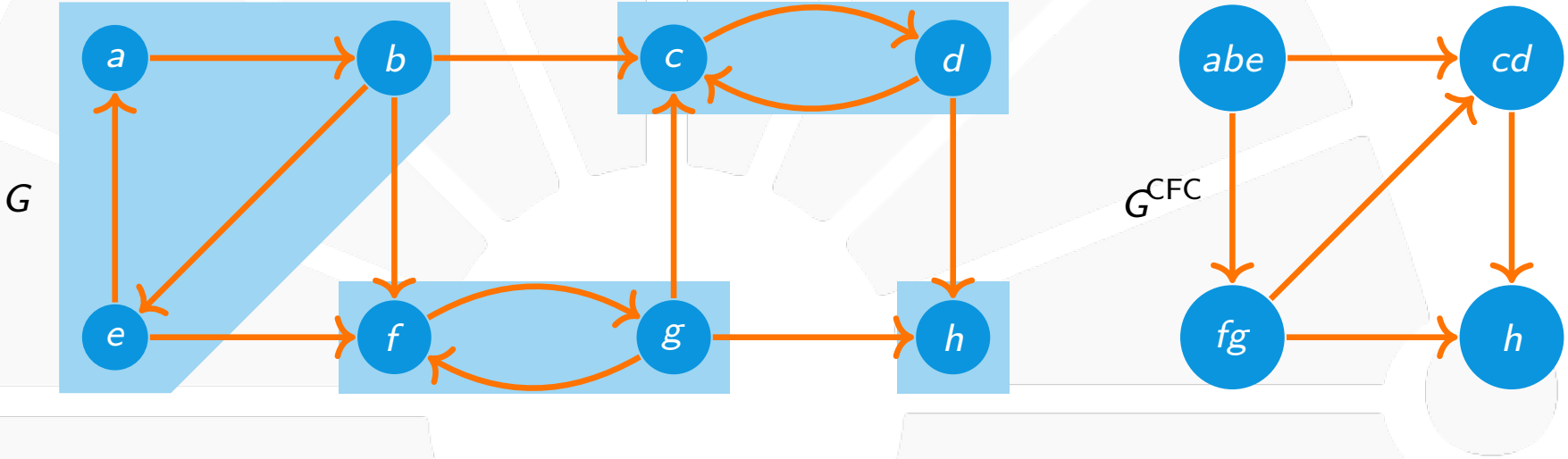


Grafo componente

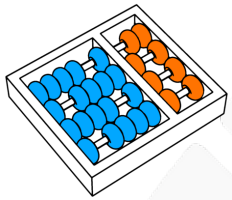




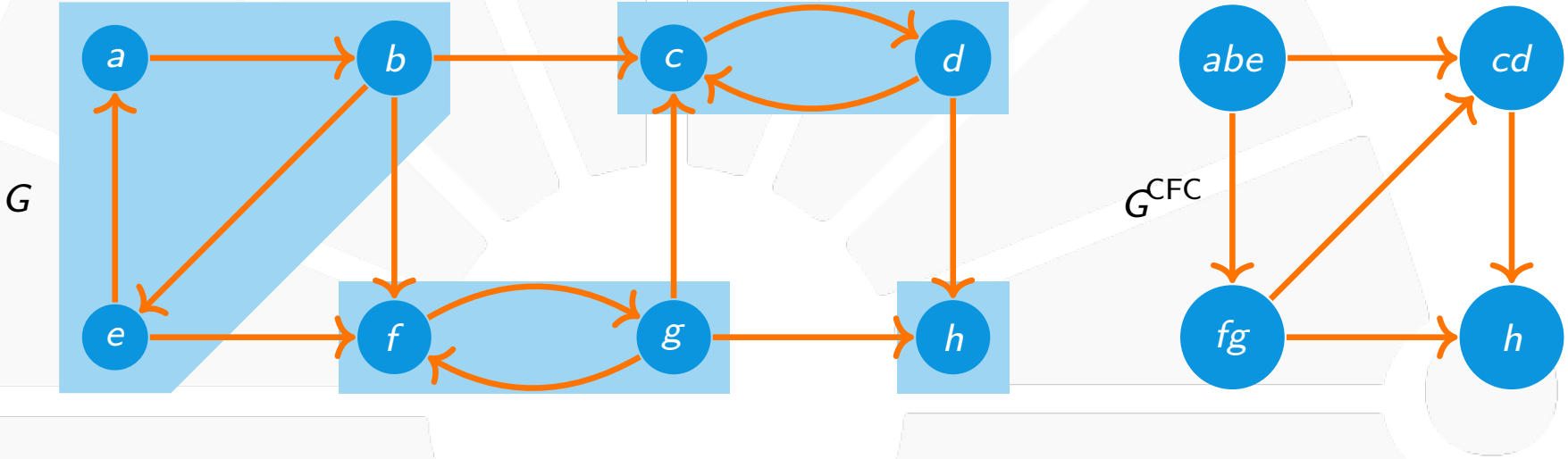
Grafo componente



Considere uma busca em profundidade sobre G:

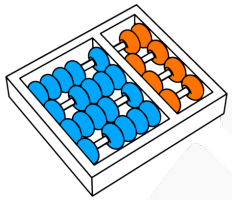


Grafo componente

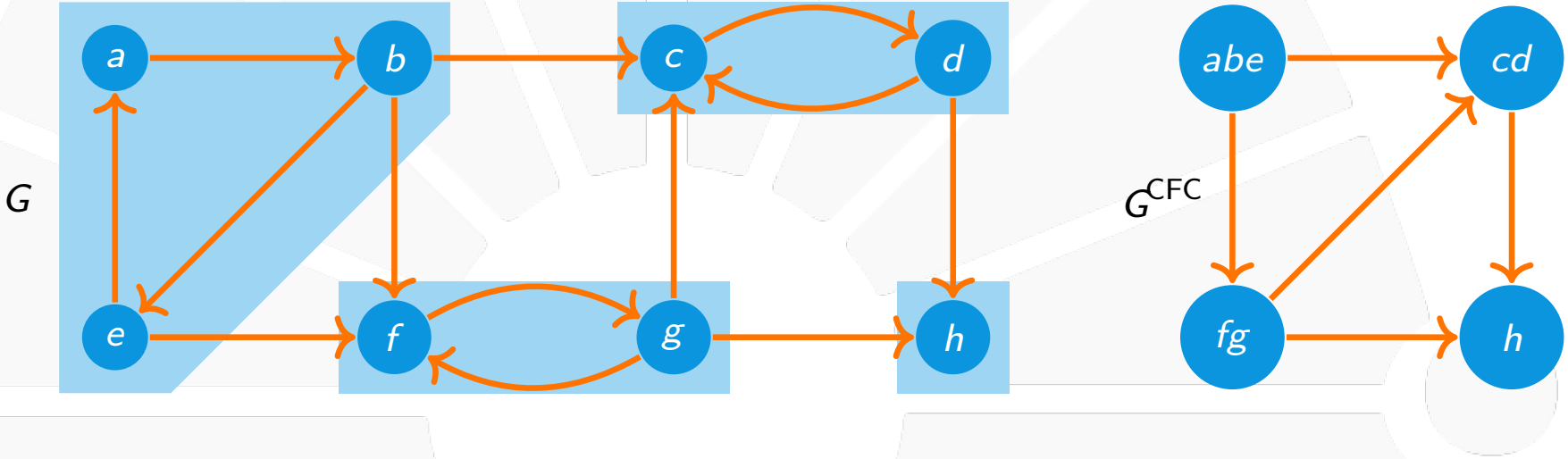


Considere uma busca em profundidade sobre G:

- Se **u** for o último vértice finalizado,

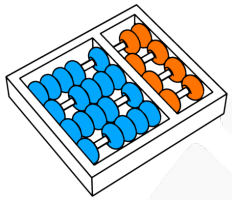


Grafo componente

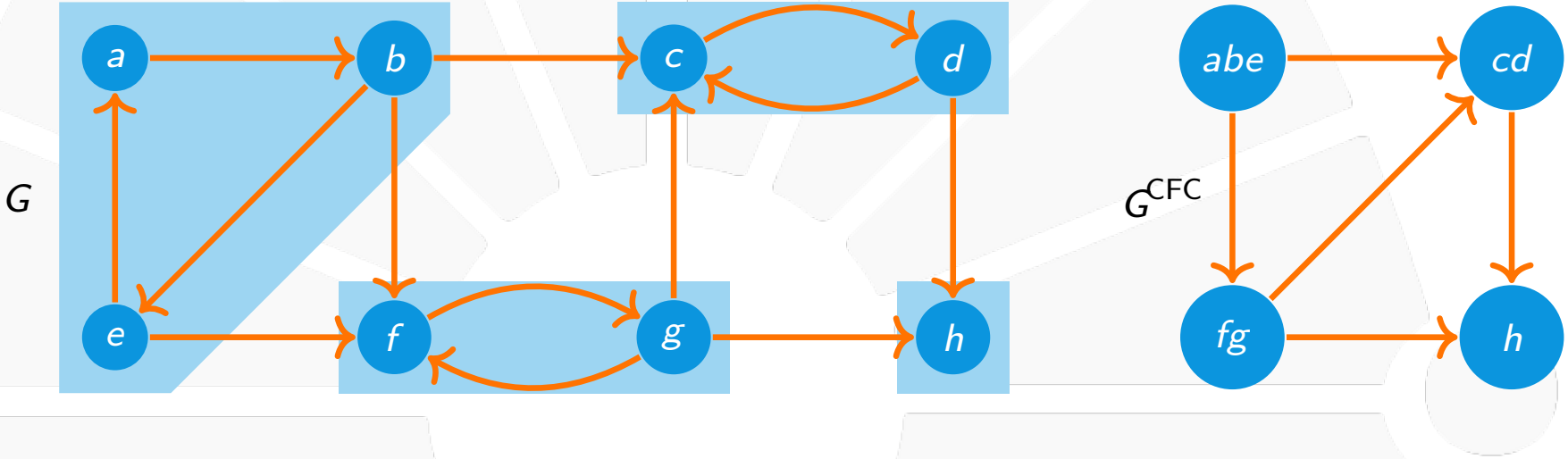


Considere uma busca em profundidade sobre G:

- Se **u** for o último vértice finalizado,
- então **u** deve pertencer a uma fonte de G^{CFC}. Por quê?

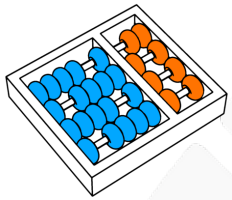


Grafo componente



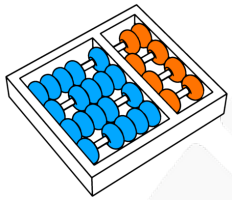
Considere uma busca em profundidade sobre G:

- Se **u** for o último vértice finalizado,
- então **u** deve pertencer a uma fonte de G^{CFC}. Por quê?
- As componentes de G^{CFC} são visitadas em **ORDEM TOPOLÓGICA**.



Grafo transposto

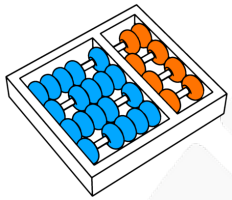
O **GRAFO TRANSPOSTO** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{T}, que:



Grafo transposto

O **GRAFO TRANSPOSTO** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{T}, que:

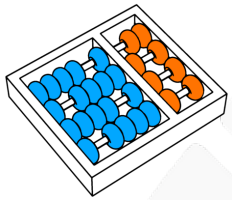
- Tem o mesmo conjunto de vértices $\mathbf{V}$.



Grafo transposto

O **GRAFO TRANSPOSTO** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{T}, que:

- Tem o mesmo conjunto de vértices **V**.
- Tem uma aresta **(u,v)** se houver uma aresta **(v,u)** em G.

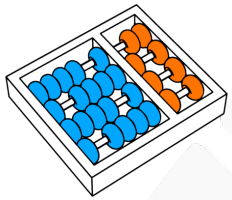


Grafo transposto

O **GRAFO TRANSPOSTO** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{T}, que:

- Tem o mesmo conjunto de vértices **V**.
- Tem uma aresta **(u,v)** se houver uma aresta **(v,u)** em G.

Observações:



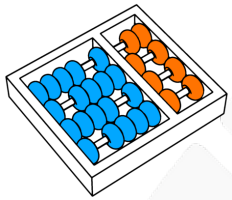
Grafo transposto

O **GRAFO TRANSPOSTO** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{T}, que:

- Tem o mesmo conjunto de vértices **V**.
- Tem uma aresta **(u,v)** se houver uma aresta **(v,u)** em G.

Observações:

- G^{T} é obtido invertendo-se as arestas de G.



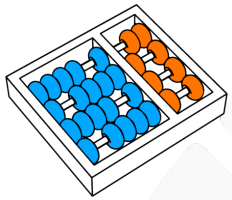
Grafo transposto

O **GRAFO TRANSPOSTO** de um grafo direcionado $G = (\mathbf{V}, \mathbf{E})$ é um grafo direcionado, denotado por G^{T}, que:

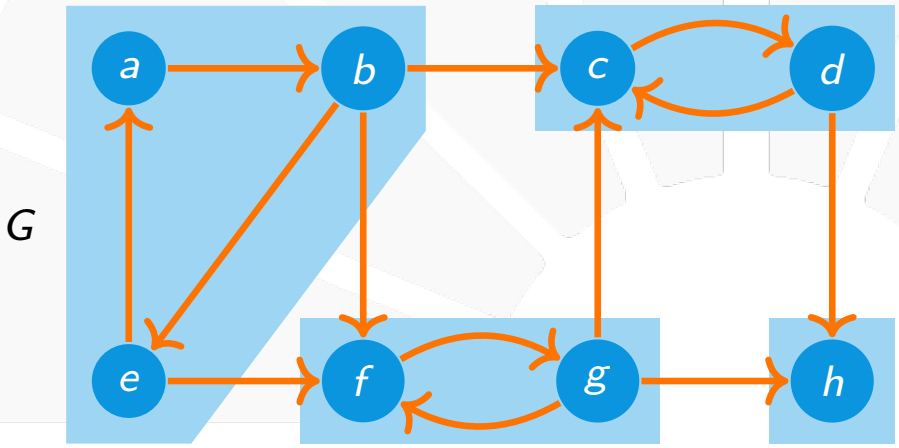
- Tem o mesmo conjunto de vértices **V**.
- Tem uma aresta **(u,v)** se houver uma aresta **(v,u)** em G.

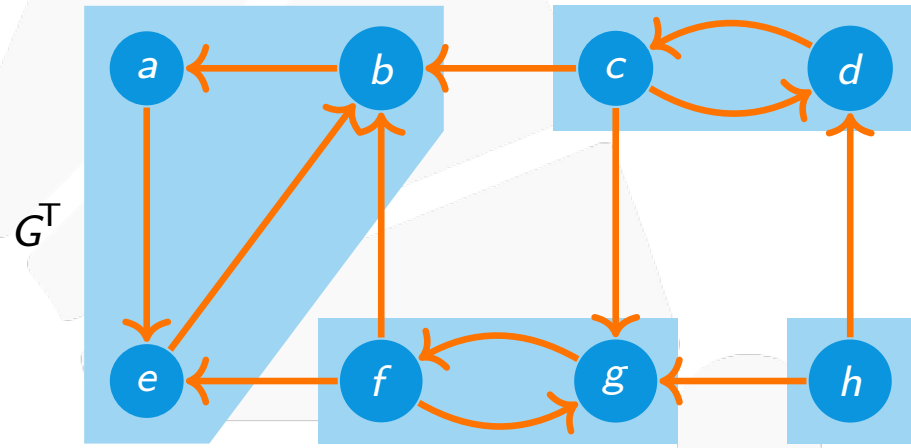
Observações:

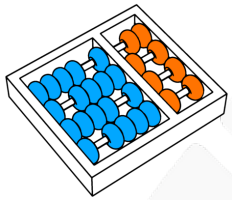
- G^{T} é obtido invertendo-se as arestas de G.
- Podemos calcular G^{T} em tempo $O(V + E)$.



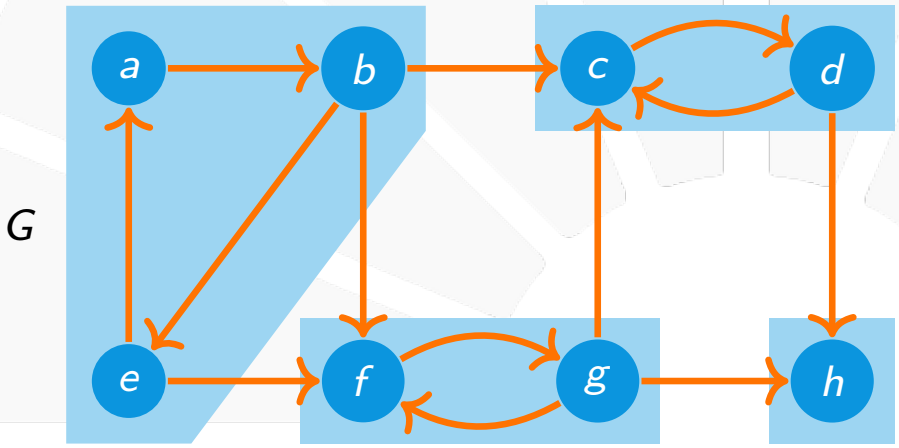
Grafo transposto

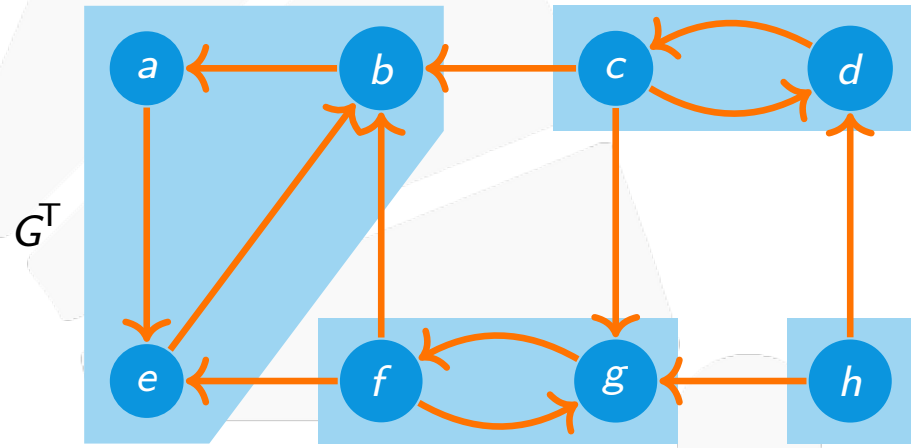




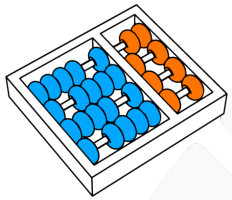


Grafo transposto

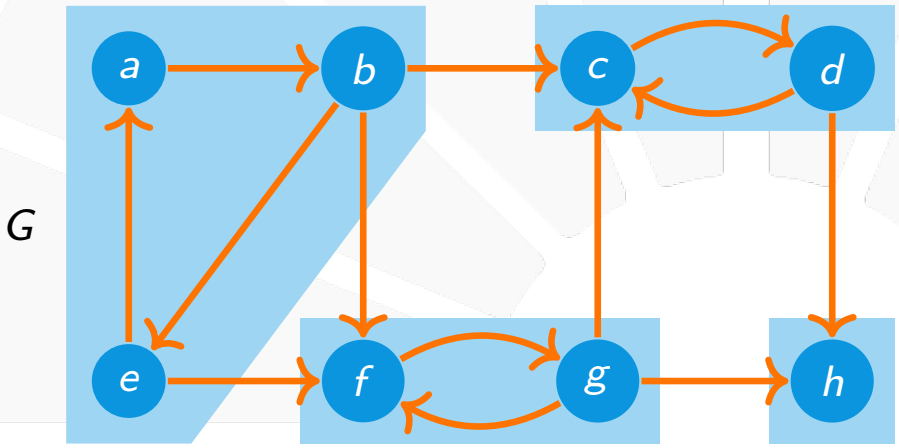


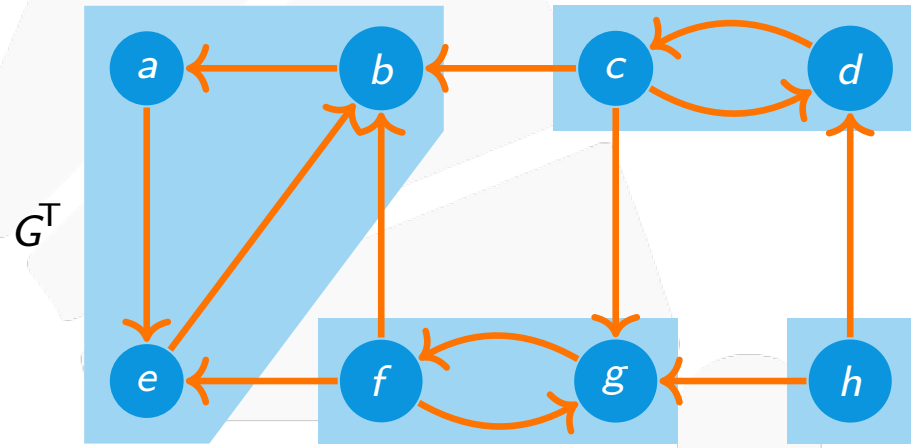


- Note que G e G^T têm as mesmas componentes. Por quê?

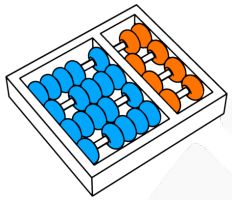


Grafo transposto

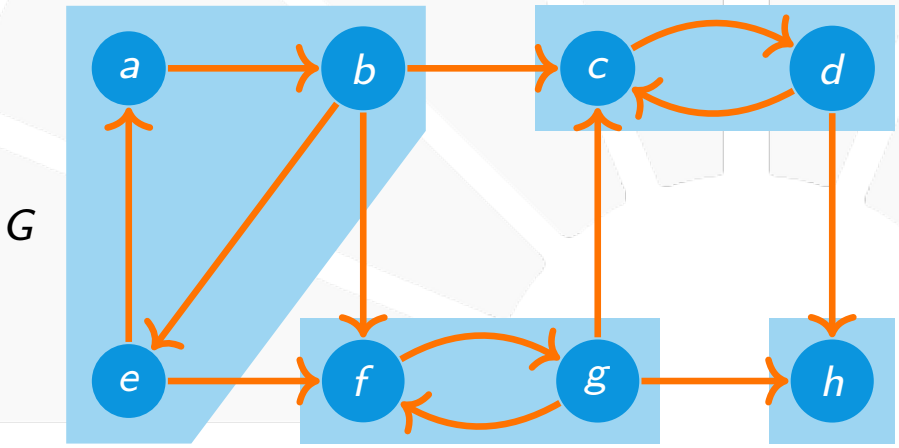


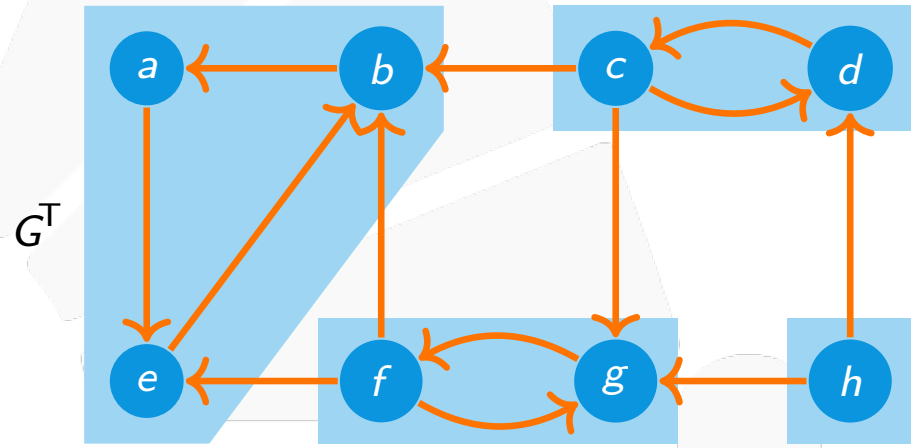


- Note que G e G^T têm as mesmas componentes. Por quê?
- Componentes fontes para G são sorvedouros para G^T.

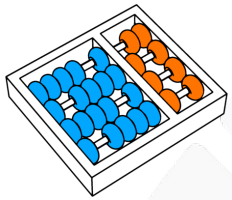


Grafo transposto





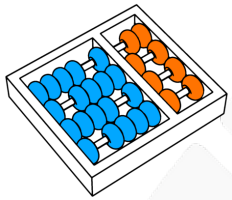
- Note que G e G^T têm as mesmas componentes. Por quê?
- Componentes fontes para G são sorvedouros para G^T.
- Se **u** for um vértice de uma fonte em G^{CFC}, então, em G^T, os **VÉRTICES ALCANÇÁVEIS** de **u** formam uma componente!



Algoritmo

Algoritmo 8: COMPONENTES-FORTEMENTE-CONEXAS(G)

1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$
2 execute $\mathrm{DFS}(G^{\mathsf{T}})$ considerando os vértices em **ORDEM DECRESCENTE** de $f[v]$
3 **devolva** os conjuntos de vértices de cada árvore da floresta de busca encontrada

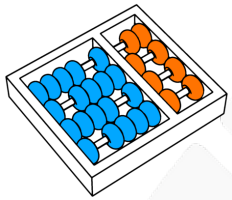


Algoritmo

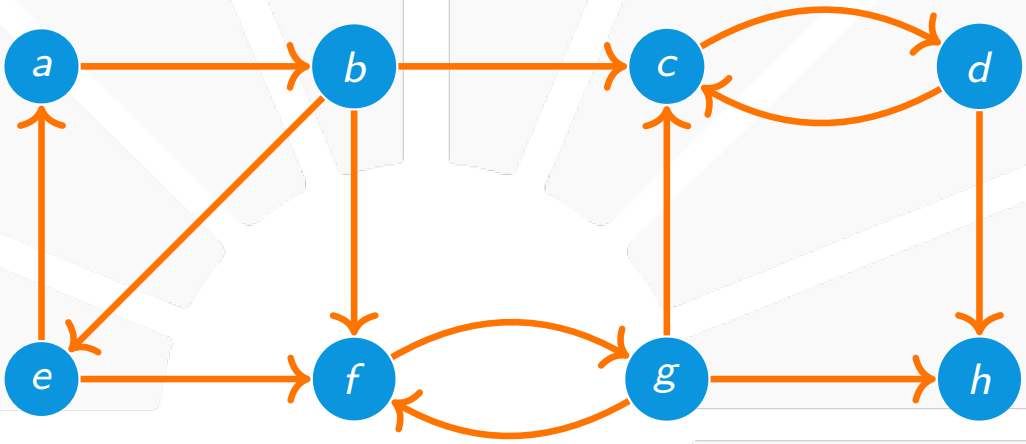
Algoritmo 9: COMPONENTES-FORTEMENTE-CONEXAS(G)

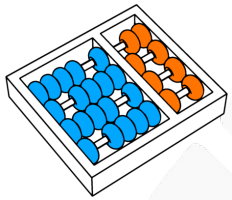
1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$
2 execute $\mathrm{DFS}(G^{\mathsf{T}})$ considerando os vértices em **ORDEM DECRESCENTE** de $f[v]$
3 **devolva** os conjuntos de vértices de cada árvore da floresta de busca encontrada

A complexidade de tempo é $O(V + E)$.

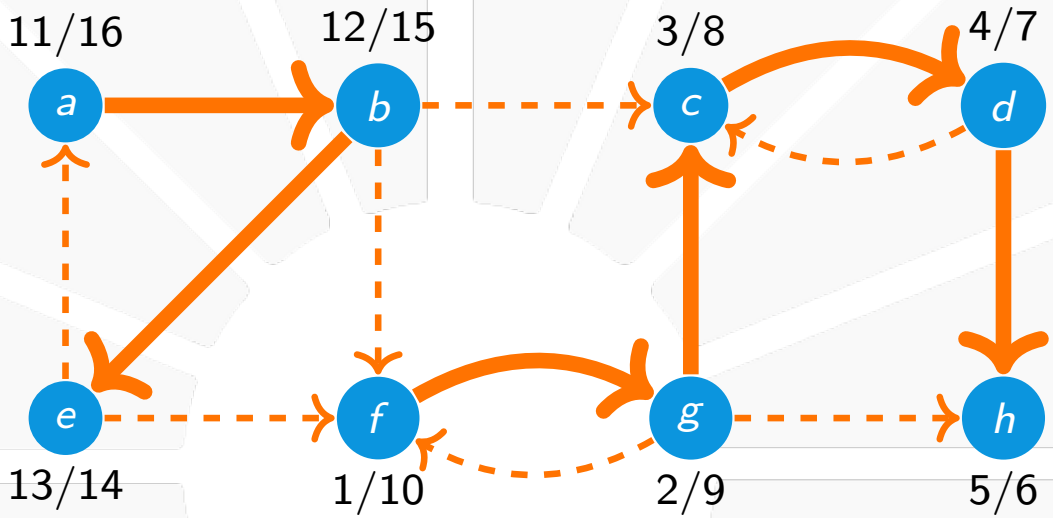


Exemplo

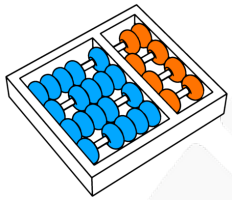




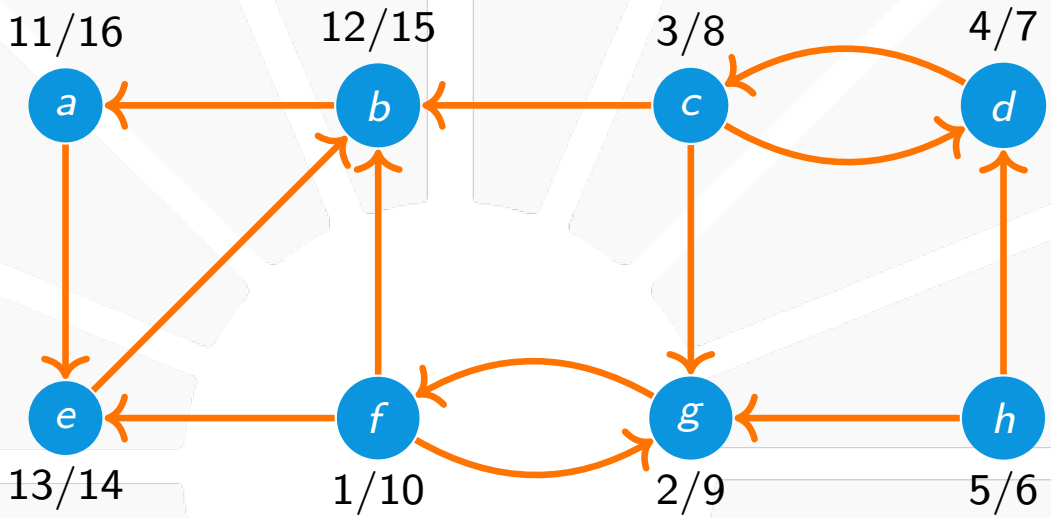
Exemplo



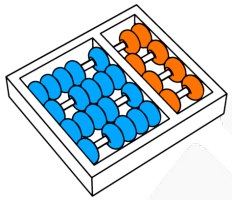
Execute $\mathrm{DFS}(G)$ e calcule $f[\mathbf{v}]$ para cada $\mathbf{v} \in \mathbf{V}$.



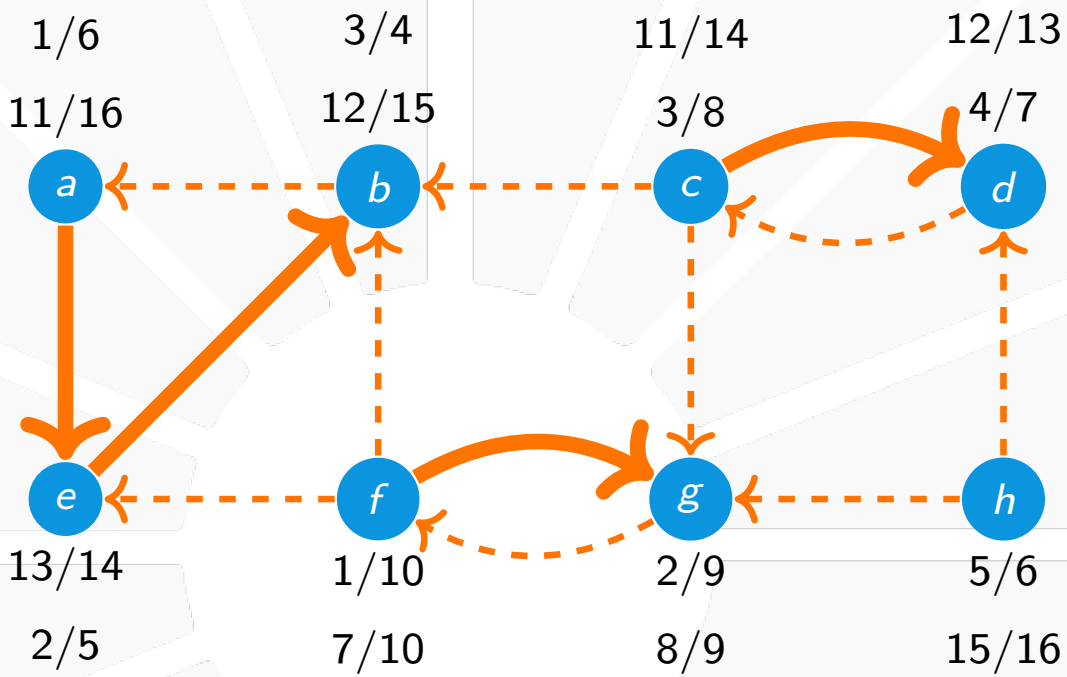
Exemplo



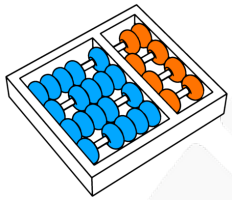
Execute $\mathrm{DFS}(G^{\mathrm{T}})$ considerando os vértices em **ORDEM DECRESCENTE** de $f[\mathbf{v}]$.



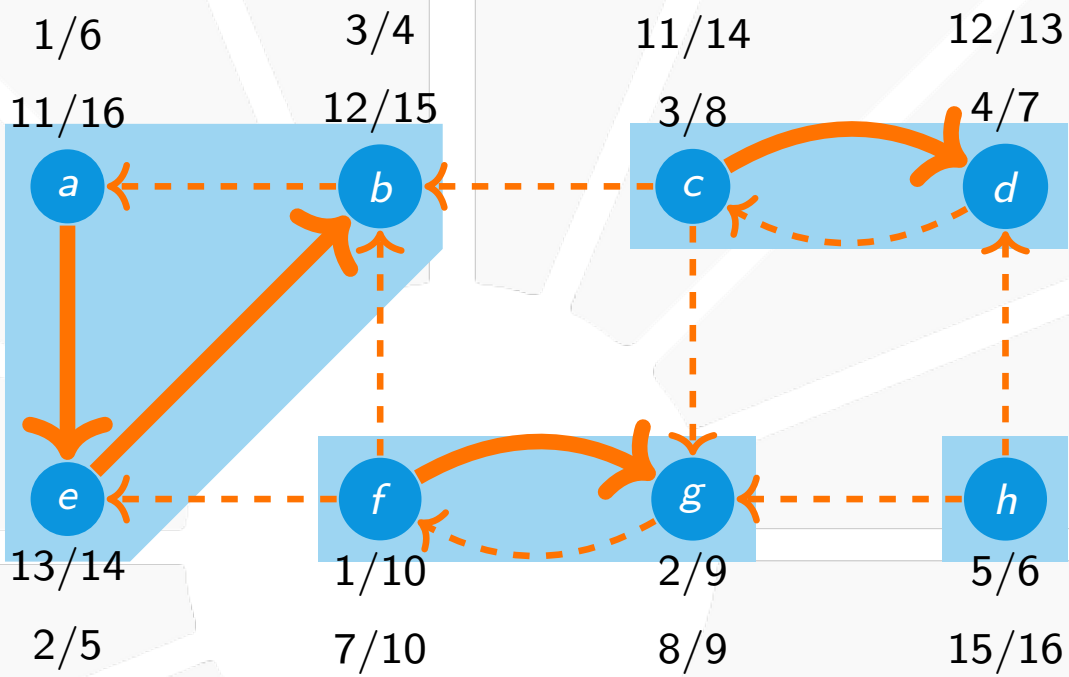
Exemplo



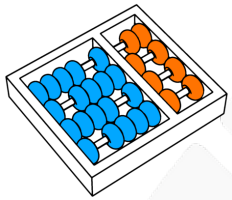
Execute $\mathrm{DFS}(G^{\mathrm{T}})$ considerando os vértices em **ORDEM DECRESCENTE** de $f[\mathbf{v}]$.



Exemplo



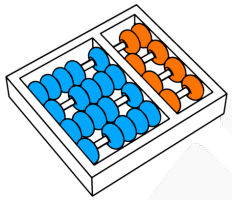
devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada.



Correção

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

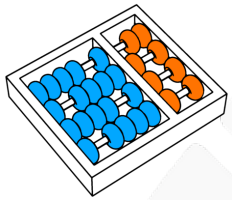


Correção

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

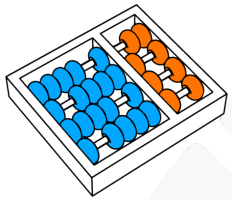
Antes da demonstração, precisamos de uma preparação.



Lema auxiliar

Lema (1)

Sejam C *e* D *duas componentes fortemente conexas e considere vértices* $u,v \in C$ *e* $u',v' \in D$.

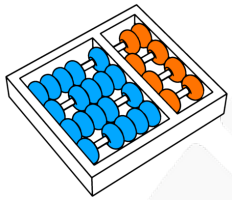


Lema auxiliar

Lema (1)

Sejam **C** *e* **D** *duas componentes fortemente conexas e considere vértices* **u,v** $\in$ **C** *e* **u',v'** $\in$ **D**.

- *Se existe algum caminho* **u** $\rightsquigarrow$ **u'**,

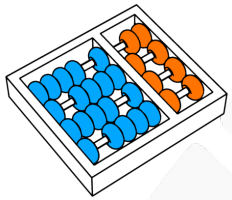


Lema auxiliar

Lema (1)

Sejam **C** *e* **D** *duas componentes fortemente conexas e considere vértices* **u,v** $\in$ **C** *e* **u',v'** $\in$ **D**.

- *Se existe algum caminho* **u** $\rightsquigarrow$ **u'**,
- *então* **NÃO** *existe um caminho* **v'** $\rightsquigarrow$ **v**.



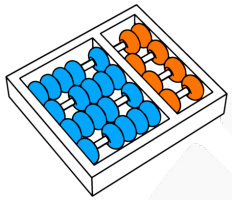
Lema auxiliar

Lema (1)

Sejam **C** *e* **D** *duas componentes fortemente conexas e considere vértices* **u,v** $\in$ **C** *e* **u',v'** $\in$ **D**.

- *Se existe algum caminho* **u** $\rightsquigarrow$ **u'**,
- *então* **NÃO** *existe um caminho* **v'** $\rightsquigarrow$ **v**.

- A prova segue da maximalidade de **C** e **D**.



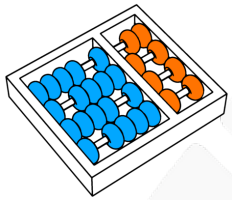
Lema auxiliar

Lema (1)

Sejam **C** *e* **D** *duas componentes fortemente conexas e considere vértices* **u,v** $\in$ **C** *e* **u',v'** $\in$ **D**.

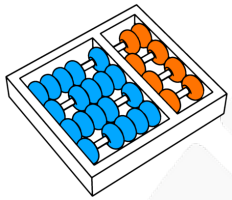
- *Se existe algum caminho* **u** $\rightsquigarrow$ **u'**,
- *então* ***NÃO*** *existe um caminho* **v'** $\rightsquigarrow$ **v**.

- A prova segue da maximalidade de **C** e **D**.
- O lema implica que G^{CFC} é **ACÍCLICO**.



Definições auxiliares

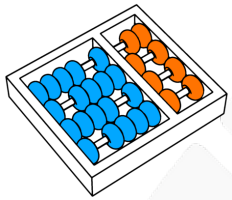
Adotaremos a seguinte convenção:



Definições auxiliares

Adotaremos a seguinte convenção:

- Consideramos que em uma execução do algoritmo d e f referem-se à busca em profundidade da linha 1 (em G).

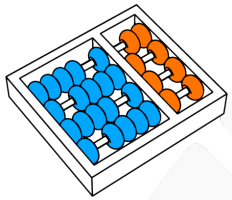


Definições auxiliares

Adotaremos a seguinte convenção:

- Consideramos que em uma execução do algoritmo d e f referem-se à busca em profundidade da linha 1 (em G).
- Para cada subconjunto $\mathbf{U}$ de vértices, definimos:

$$d(\mathbf{U}) = \min_{\mathbf{u} \in \mathbf{U}} \{d[\mathbf{u}]\} \quad \text{e} \quad f(\mathbf{U}) = \max_{\mathbf{u} \in \mathbf{U}} \{f[\mathbf{u}]\}$$



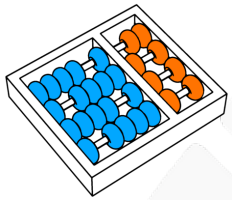
Definições auxiliares

Adotaremos a seguinte convenção:

- Consideramos que em uma execução do algoritmo d e f referem-se à busca em profundidade da linha 1 (em G).
- Para cada subconjunto $\mathbf{U}$ de vértices, definimos:

$$d(\mathbf{U}) = \min_{\mathbf{u} \in \mathbf{U}} \{d[\mathbf{u}]\} \quad \text{e} \quad f(\mathbf{U}) = \max_{\mathbf{u} \in \mathbf{U}} \{f[\mathbf{u}]\}$$

Em outras palavras:



Definições auxiliares

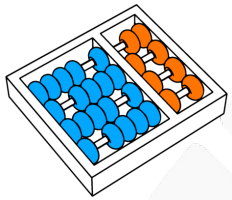
Adotaremos a seguinte convenção:

- Consideramos que em uma execução do algoritmo d e f referem-se à busca em profundidade da linha 1 (em G).
- Para cada subconjunto $\mathbf{U}$ de vértices, definimos:

$$d(\mathbf{U}) = \min_{\mathbf{u} \in \mathbf{U}} \{d[\mathbf{u}]\} \quad \text{e} \quad f(\mathbf{U}) = \max_{\mathbf{u} \in \mathbf{U}} \{f[\mathbf{u}]\}$$

Em outras palavras:

- $d(\mathbf{U})$ é o **PRIMEIRO** instante em que um vértice de $\mathbf{U}$ é descoberto.



Definições auxiliares

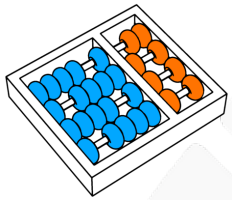
Adotaremos a seguinte convenção:

- Consideramos que em uma execução do algoritmo d e f referem-se à busca em profundidade da linha 1 (em G).
- Para cada subconjunto $\mathbf{U}$ de vértices, definimos:

$$d(\mathbf{U}) = \min_{\mathbf{u} \in \mathbf{U}} \{d[\mathbf{u}]\} \quad \text{e} \quad f(\mathbf{U}) = \max_{\mathbf{u} \in \mathbf{U}} \{f[\mathbf{u}]\}$$

Em outras palavras:

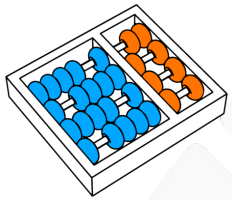
- $d(\mathbf{U})$ é o **PRIMEIRO** instante em que um vértice de $\mathbf{U}$ é descoberto.
- $f(\mathbf{U})$ é o **ÚLTIMO** instante em que um vértice de $\mathbf{U}$ é finalizado.



Outro lema auxiliar

Lema (2)

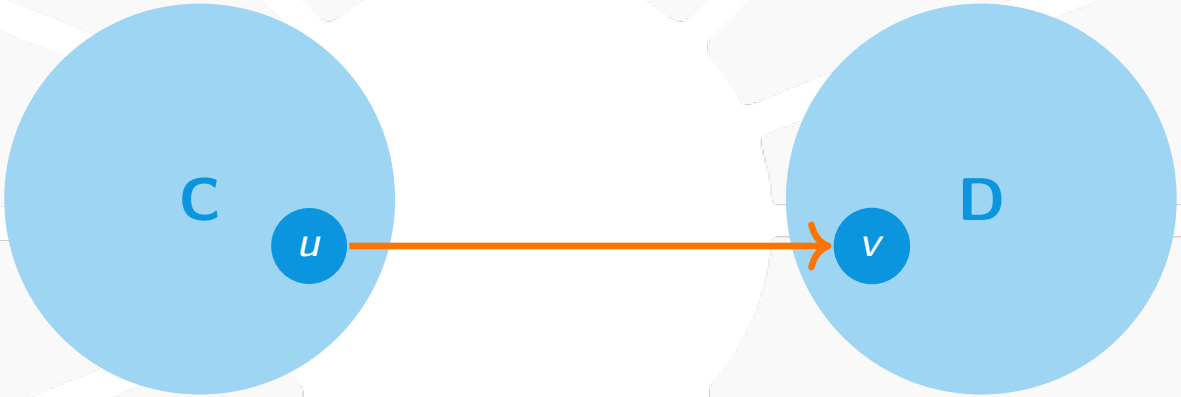
Sejam C *e* D *duas componentes fortemente conexas. Se existe aresta* (u,v) *tal que* $u \in C$ *e* $v \in D$, *então* $f(C) > f(D)$.

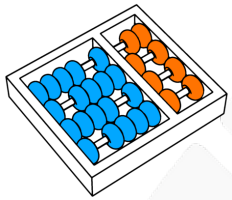


Outro lema auxiliar

Lema (2)

Sejam **C** *e* **D** *duas componentes fortemente conexas. Se existe aresta* **(u,v)** *tal que* $u \in C$ *e* $v \in D$, *então* $f(C) > f(D)$.

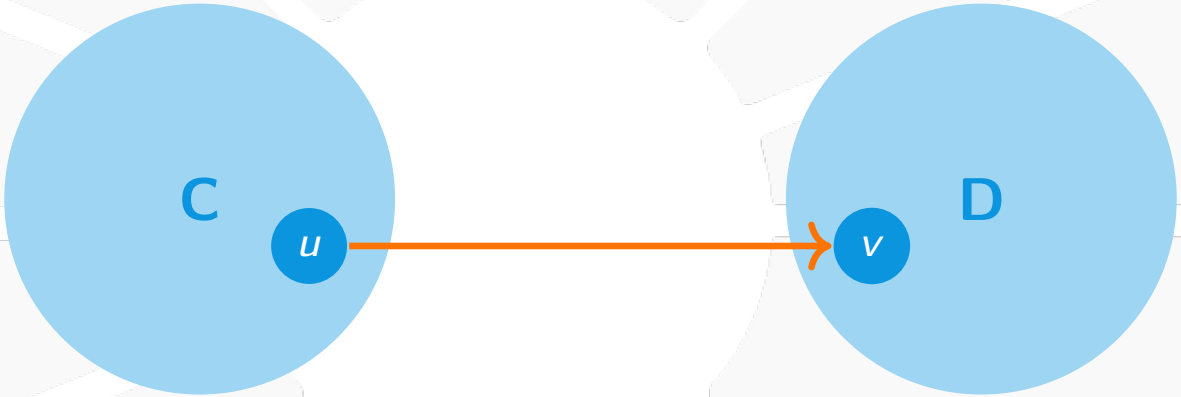




Outro lema auxiliar

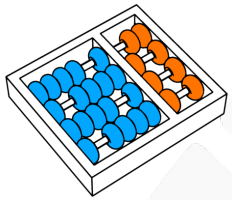
Lema (2)

Sejam **C** *e* **D** *duas componentes fortemente conexas. Se existe aresta* **(u,v)** *tal que* $u \in C$ *e* $v \in D$, *então* $f(C) > f(D)$.



Corolário

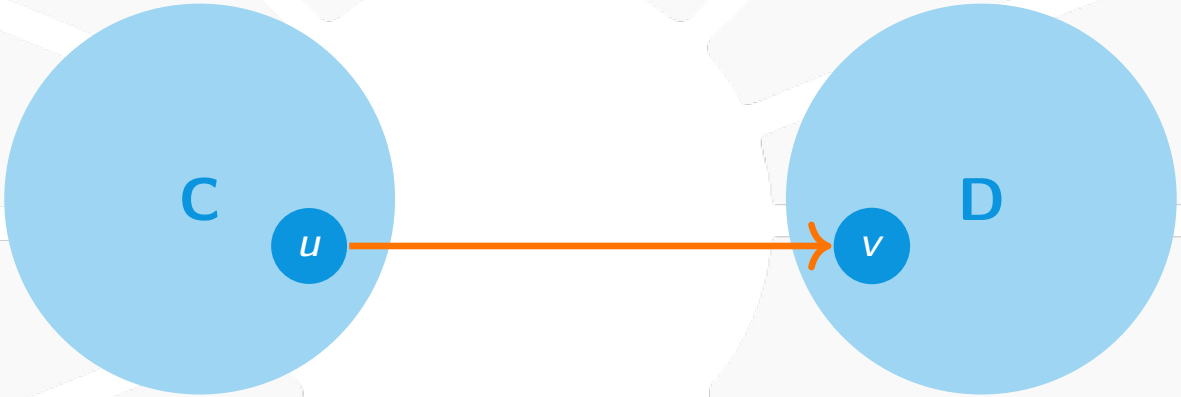
Se G^T *tem aresta* **(u,v)** *tal que* $u \in C$ *e* $v \in D$, *então* $f(C) < f(D)$.



Outro lema auxiliar

Lema (2)

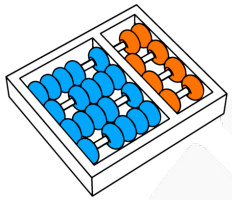
Sejam **C** *e* **D** *duas componentes fortemente conexas. Se existe aresta* **(u,v)** *tal que* $u \in C$ *e* $v \in D$, *então* $f(C) > f(D)$.



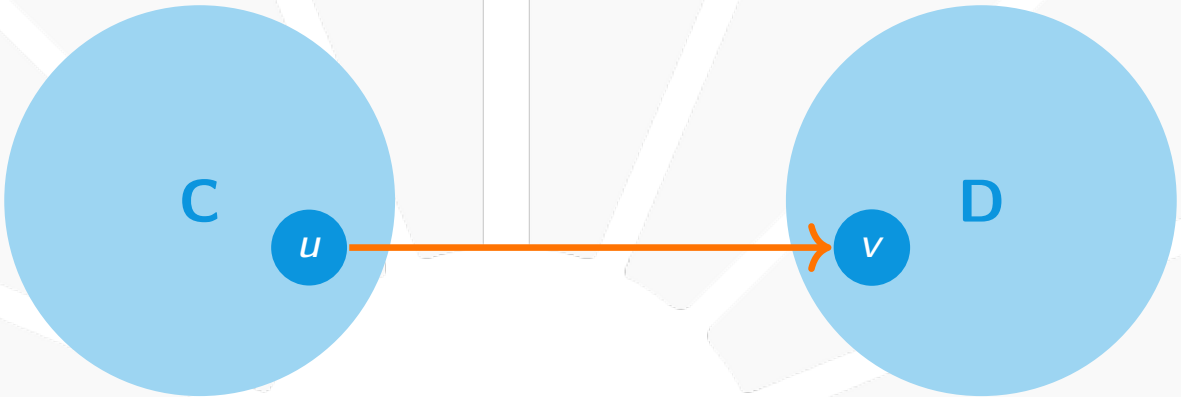
Corolário

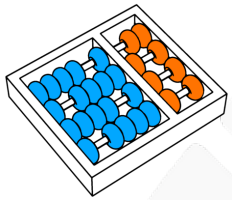
Se G^T *tem aresta* **(u,v)** *tal que* $u \in C$ *e* $v \in D$, *então* $f(C) < f(D)$.

Segue do fato de que G e G^T têm as mesmas componentes.

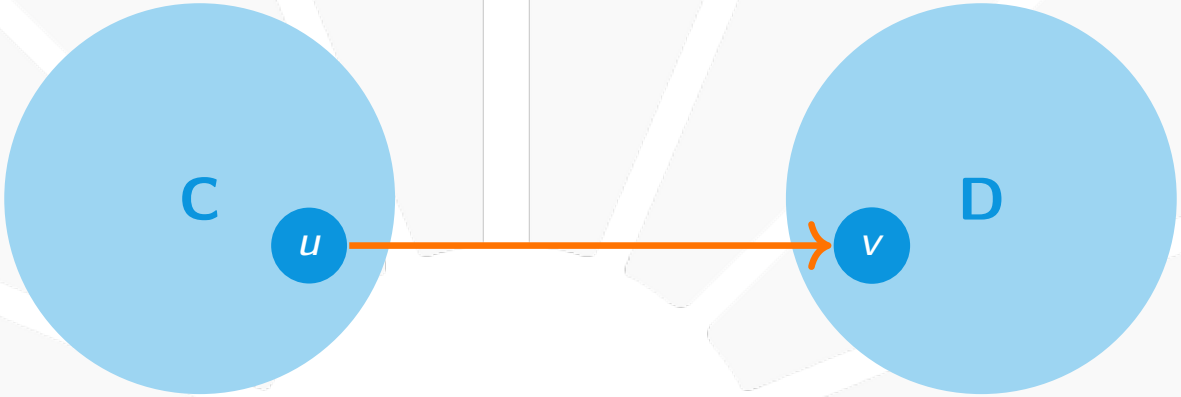


Demonstração do lema

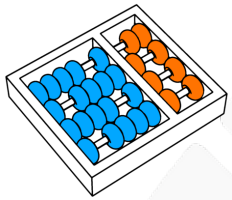




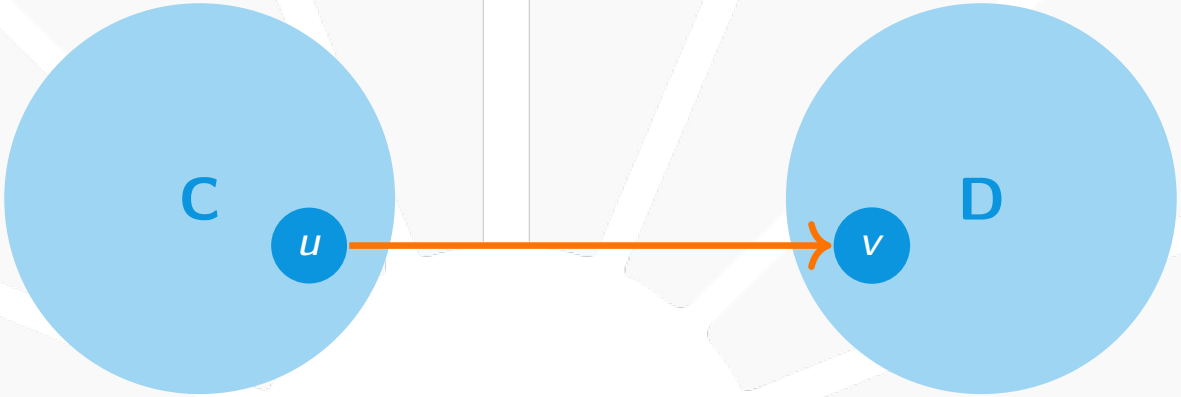
Demonstração do lema



Suponha que $d(\mathbf{C}) < d(\mathbf{D})$.

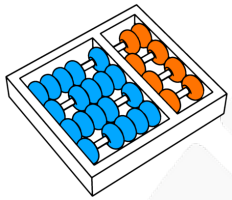


Demonstração do lema

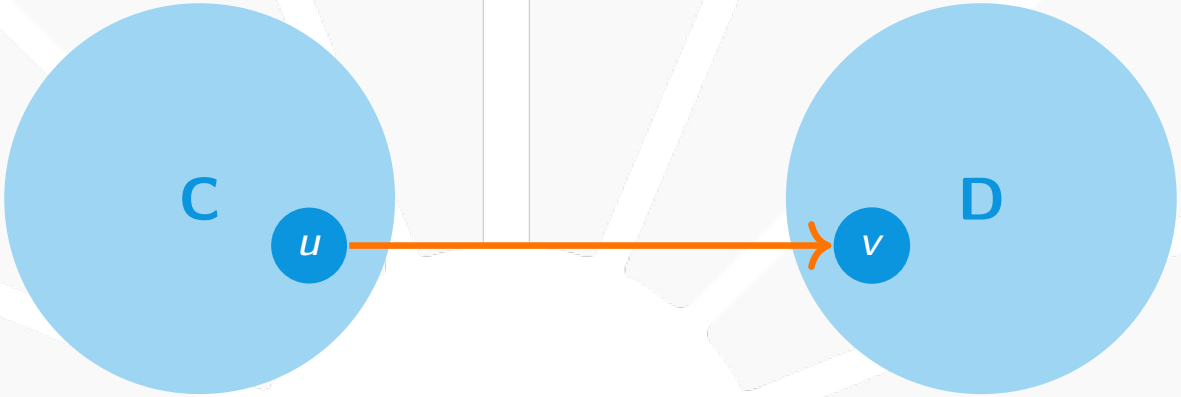


Suponha que $d(\mathbf{C}) < d(\mathbf{D})$.

- Isso é, **C** foi descoberto antes de **D**.

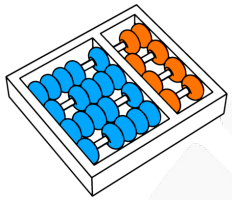


Demonstração do lema

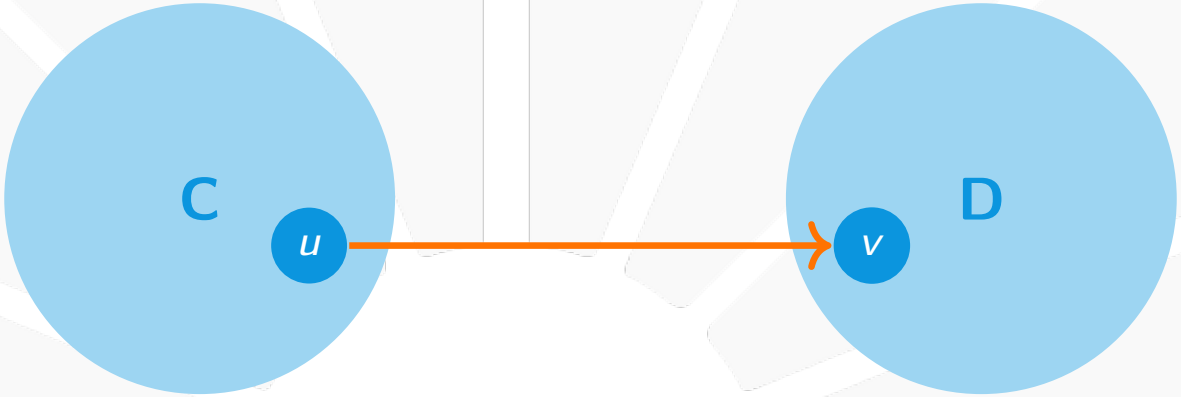


Suponha que $d(\mathbf{C}) < d(\mathbf{D})$.

- Isso é, **C** foi descoberto antes de **D**.
- Seja **x** o primeiro vértice de **C** a ser descoberto ($d[\mathbf{x}] = d(\mathbf{C})$).

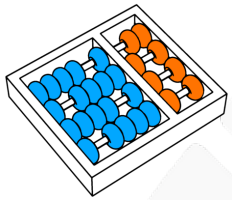


Demonstração do lema

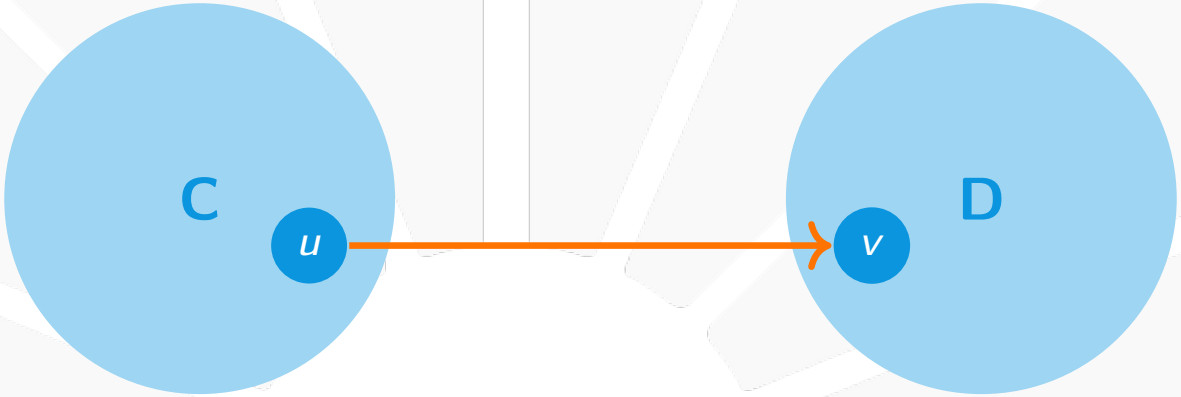


Suponha que $d(\mathbf{C}) < d(\mathbf{D})$.

- Isso é, **C** foi descoberto antes de **D**.
- Seja **x** o primeiro vértice de **C** a ser descoberto ($d[\mathbf{x}] = d(\mathbf{C})$).
- No instante $d[\mathbf{x}]$, existia um caminho branco de **x** a cada um dos vértices em $\mathbf{C} \cup \mathbf{D}$.

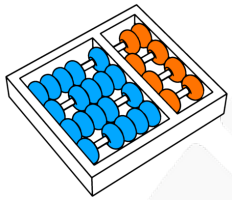


Demonstração do lema

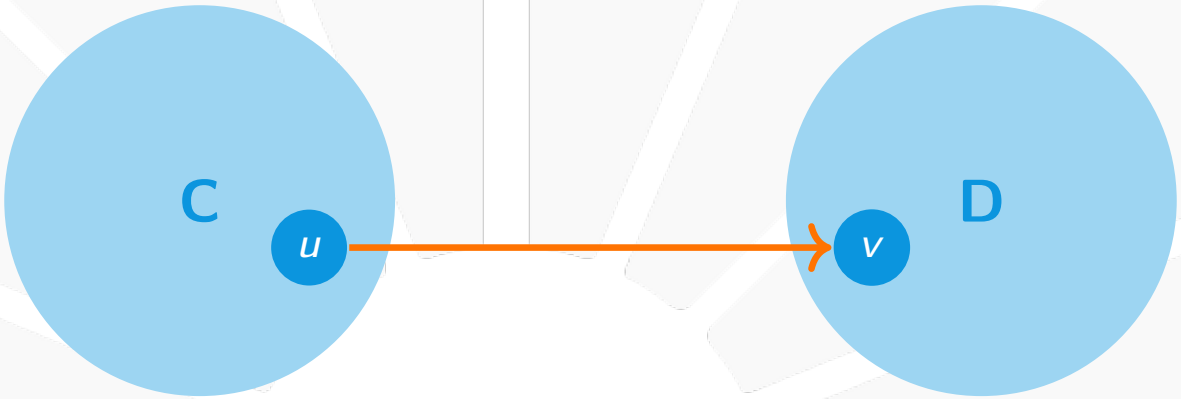


Suponha que $d(\mathbf{C}) < d(\mathbf{D})$.

- Isso é, **C** foi descoberto antes de **D**.
- Seja **x** o primeiro vértice de **C** a ser descoberto ($d[\mathbf{x}] = d(\mathbf{C})$).
- No instante $d[\mathbf{x}]$, existia um caminho branco de **x** a cada um dos vértices em $\mathbf{C} \cup \mathbf{D}$.
- Então, todos os vértices de $\mathbf{C} \cup \mathbf{D}$ são descendentes de **x**.

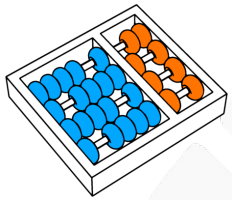


Demonstração do lema

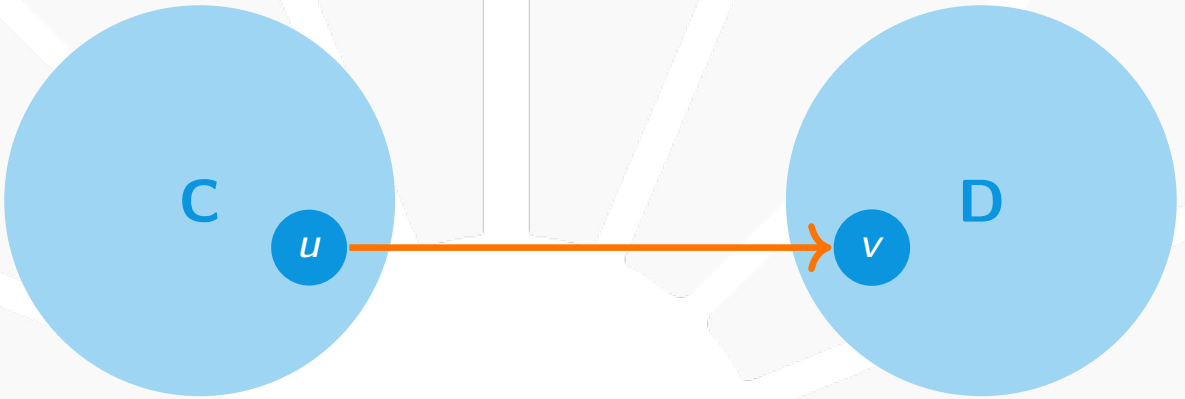


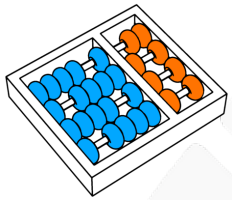
Suponha que $d(\mathbf{C}) < d(\mathbf{D})$.

- Isso é, **C** foi descoberto antes de **D**.
- Seja **x** o primeiro vértice de **C** a ser descoberto ($d[\mathbf{x}] = d(\mathbf{C})$).
- No instante $d[\mathbf{x}]$, existia um caminho branco de **x** a cada um dos vértices em $\mathbf{C} \cup \mathbf{D}$.
- Então, todos os vértices de $\mathbf{C} \cup \mathbf{D}$ são descendentes de **x**.
- Portanto, $f(\mathbf{D}) < f[\mathbf{x}] \leq f(\mathbf{C})$.

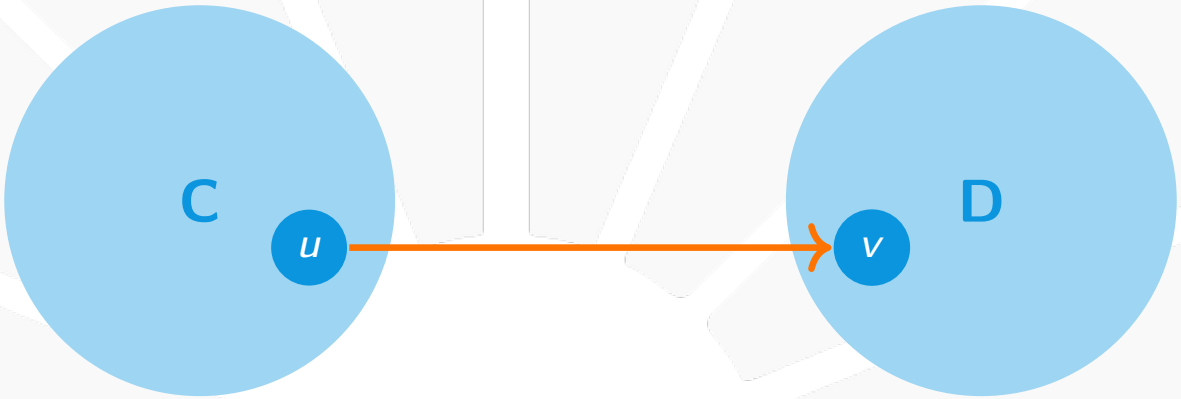


Demonstração do lema

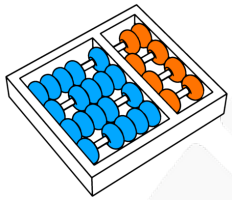




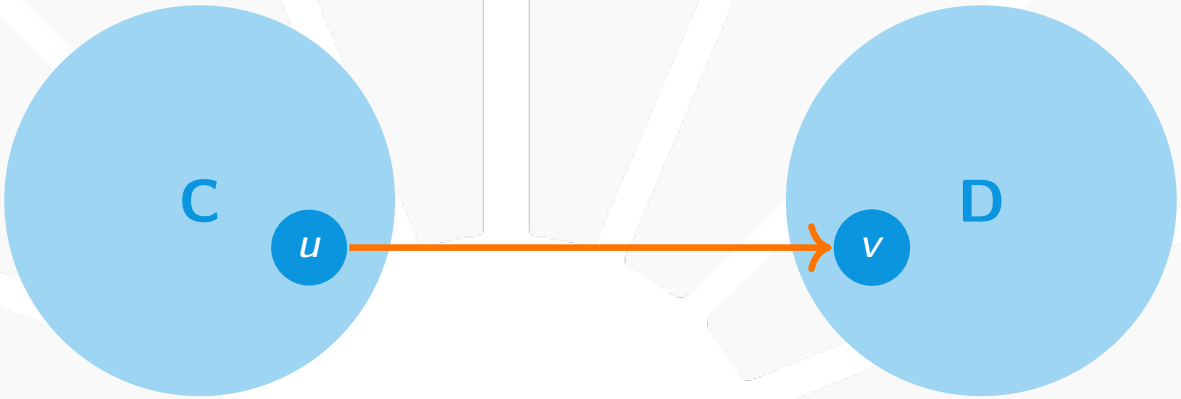
Demonstração do lema



Agora, suponha que $d(\mathbf{C}) > d(\mathbf{D})$:

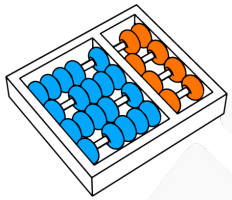


Demonstração do lema

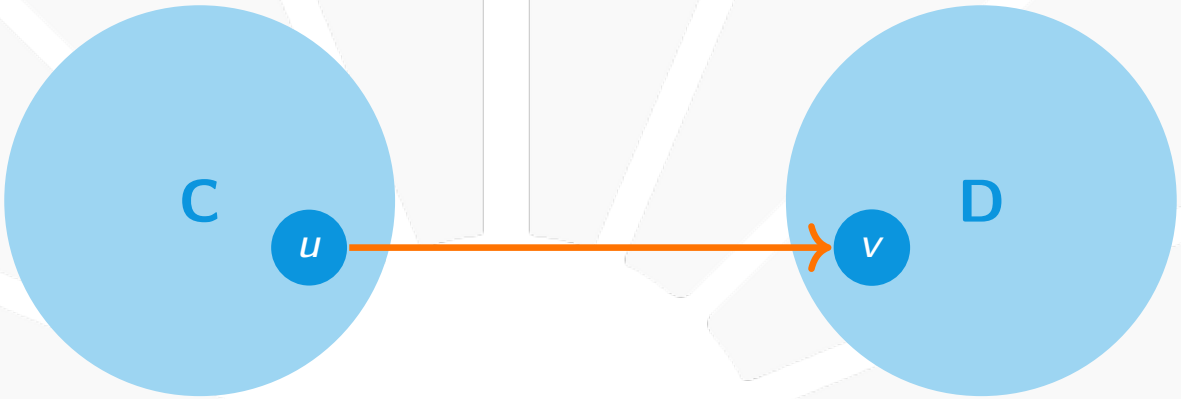


Agora, suponha que $d(\mathbf{C}) > d(\mathbf{D})$:

- Assim, o primeiro vértice a ser descoberto está em **D**.

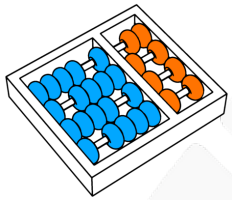


Demonstração do lema

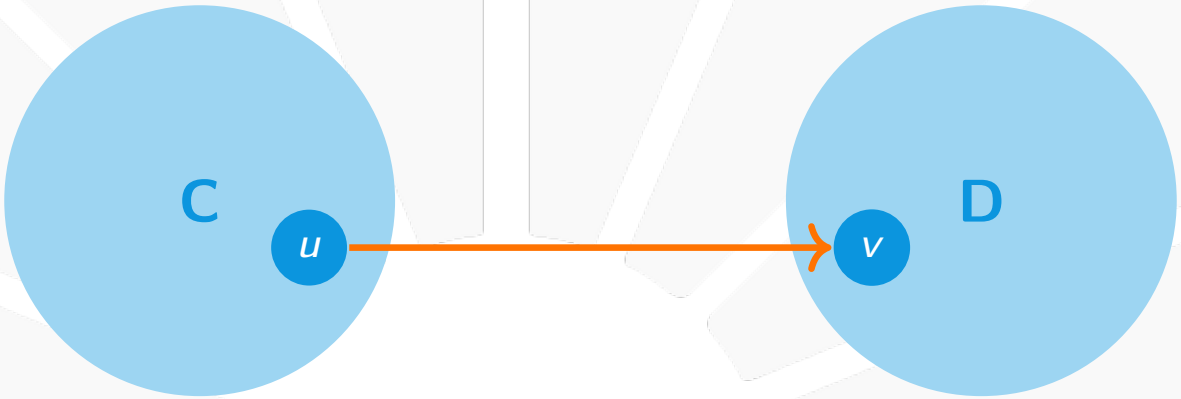


Agora, suponha que $d(\mathbf{C}) > d(\mathbf{D})$:

- Assim, o primeiro vértice a ser descoberto está em **D**.
- Logo, cada um dos vértices de **D** é finalizado antes de qualquer vértice de **C** ser descoberto.

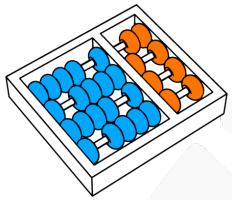


Demonstração do lema



Agora, suponha que $d(\mathbf{C}) > d(\mathbf{D})$:

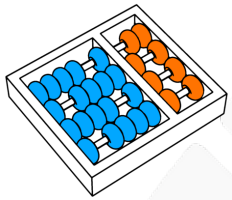
- Assim, o primeiro vértice a ser descoberto está em **D**.
- Logo, cada um dos vértices de **D** é finalizado antes de qualquer vértice de **C** ser descoberto.
- Portanto, $f(\mathbf{C}) > f(\mathbf{D})$.



Demonstração do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

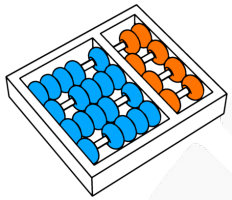


Demonstração do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

Demonstração:



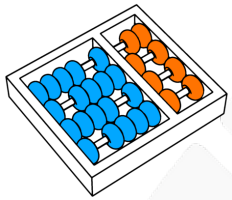
Demonstração do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

Demonstração:

- Provaremos que as **PRIMEIRAS** k árvores produzidas na linha 3 correspondem a componentes fortemente conexas.



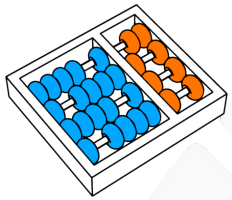
Demonstração do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

Demonstração:

- Provaremos que as **PRIMEIRAS** k árvores produzidas na linha 3 correspondem a componentes fortemente conexas.
- A prova é por indução em k.



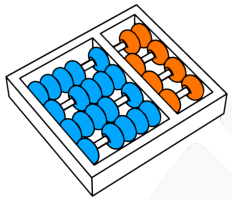
Demonstração do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

Demonstração:

- Provaremos que as **PRIMEIRAS** k árvores produzidas na linha 3 correspondem a componentes fortemente conexas.
- A prova é por indução em k.
- Quando $k = 0$, a afirmação é trivial, então tome $k \geq 1$.



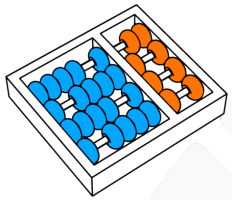
Demonstração do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS *determina as componentes fortemente conexas de G em tempo $O(V + E)$.*

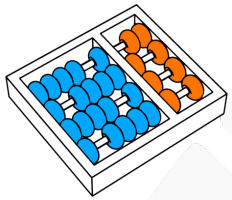
Demonstração:

- Provaremos que as **PRIMEIRAS** k árvores produzidas na linha 3 correspondem a componentes fortemente conexas.
- A prova é por indução em k.
- Quando $k = 0$, a afirmação é trivial, então tome $k \geq 1$.
- Suponha que as primeiras $k - 1$ árvores produzidas correspondem a componentes fortemente conexas.



Demonstração do teorema

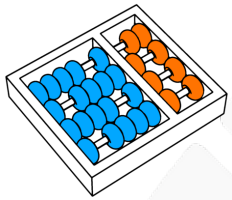
Considere a k-ésima árvore produzida pelo algoritmo:



Demonstração do teorema

Considere a k-ésima árvore produzida pelo algoritmo:

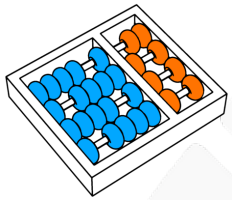
- Sejam **u** a raiz dessa árvore de busca e **C** a componente fortemente conexa que contém **u**.



Demonstração do teorema

Considere a k-ésima árvore produzida pelo algoritmo:

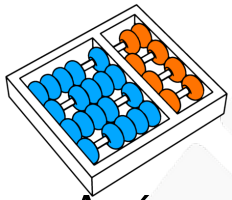
- Sejam **u** a raiz dessa árvore de busca e **C** a componente fortemente conexa que contém **u**.
- Mostraremos que a árvore produzida contém **TODOS** os vértices de **C** e **SOMENTE** os vértices de **C**.



Demonstração do teorema

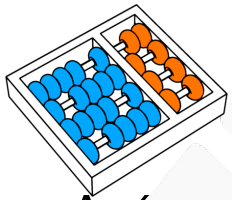
Considere a k-ésima árvore produzida pelo algoritmo:

- Sejam **u** a raiz dessa árvore de busca e **C** a componente fortemente conexa que contém **u**.
- Mostraremos que a árvore produzida contém **TODOS** os vértices de **C** e **SOMENTE** os vértices de **C**.
- Isso completará a indução e a prova do teorema.



Demonstração do teorema

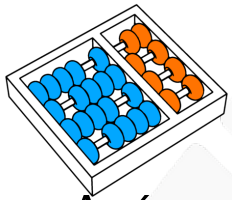
A árvore contém **TODOS** os vértices de **C**:



Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

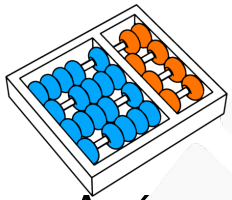
- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.



Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

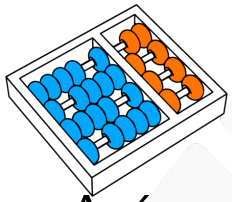
- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.



Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

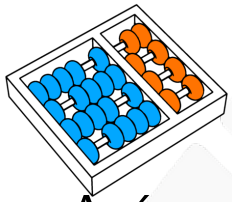
- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.



Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

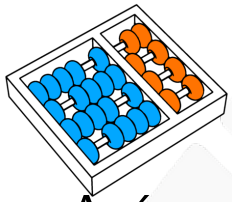


Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:



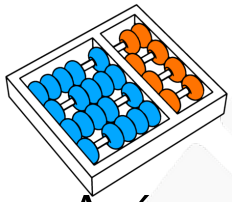
Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:

- Suponha que existe uma aresta **(x,v)** que sai de **C**.



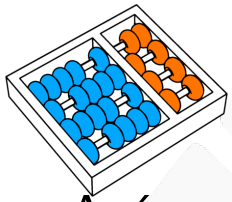
Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:

- Suponha que existe uma aresta **(x,v)** que sai de **C**.
- Seja **D** a componente fortemente conexa que contém **v**.



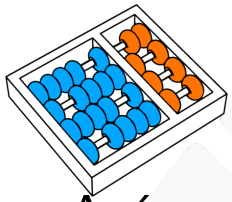
Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:

- Suponha que existe uma aresta **(x,v)** que sai de **C**.
- Seja **D** a componente fortemente conexa que contém **v**.
- Pelo corolário do Lema 2, temos que $f(\mathbf{C}) < f(\mathbf{D})$.



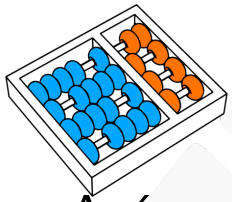
Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:

- Suponha que existe uma aresta **(x,v)** que sai de **C**.
- Seja **D** a componente fortemente conexa que contém **v**.
- Pelo corolário do Lema 2, temos que $f(\mathbf{C}) < f(\mathbf{D})$.
- Então, descobrimos vértices de **D** antes de **u**.



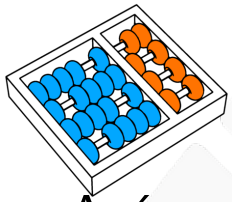
Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:

- Suponha que existe uma aresta **(x,v)** que sai de **C**.
- Seja **D** a componente fortemente conexa que contém **v**.
- Pelo corolário do Lema 2, temos que $f(\mathbf{C}) < f(\mathbf{D})$.
- Então, descobrimos vértices de **D** antes de **u**.
- Por indução, todos vértices de **D** já foram finalizados.



Demonstração do teorema

A árvore contém **TODOS** os vértices de **C**:

- Considere o instante $d[\mathbf{u}]$, em que **u** é descoberto.
- Por indução, nenhum vértice de **C** foi finalizado.
- Então, no instante $d[\mathbf{u}]$ os vértices de **C** são brancos.
- Portanto, todos os vértices de **C** tornam-se descendentes de **u** na árvore de busca de G^{T}.

A árvore contém **SOMENTE** vértices de **C**:

- Suponha que existe uma aresta **(x,v)** que sai de **C**.
- Seja **D** a componente fortemente conexa que contém **v**.
- Pelo corolário do Lema 2, temos que $f(\mathbf{C}) < f(\mathbf{D})$.
- Então, descobrimos vértices de **D** antes de **u**.
- Por indução, todos vértices de **D** já foram finalizados.
- Portanto, a árvore só contém vértices de **C**.

Componentes conexas e ordenação topológica

MC558 - Projeto e Análise de Algoritmos II

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

UNICAMP