

BUSCAS EM GRAFOS

MC558 - Projeto e Análise de Algoritmos II

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

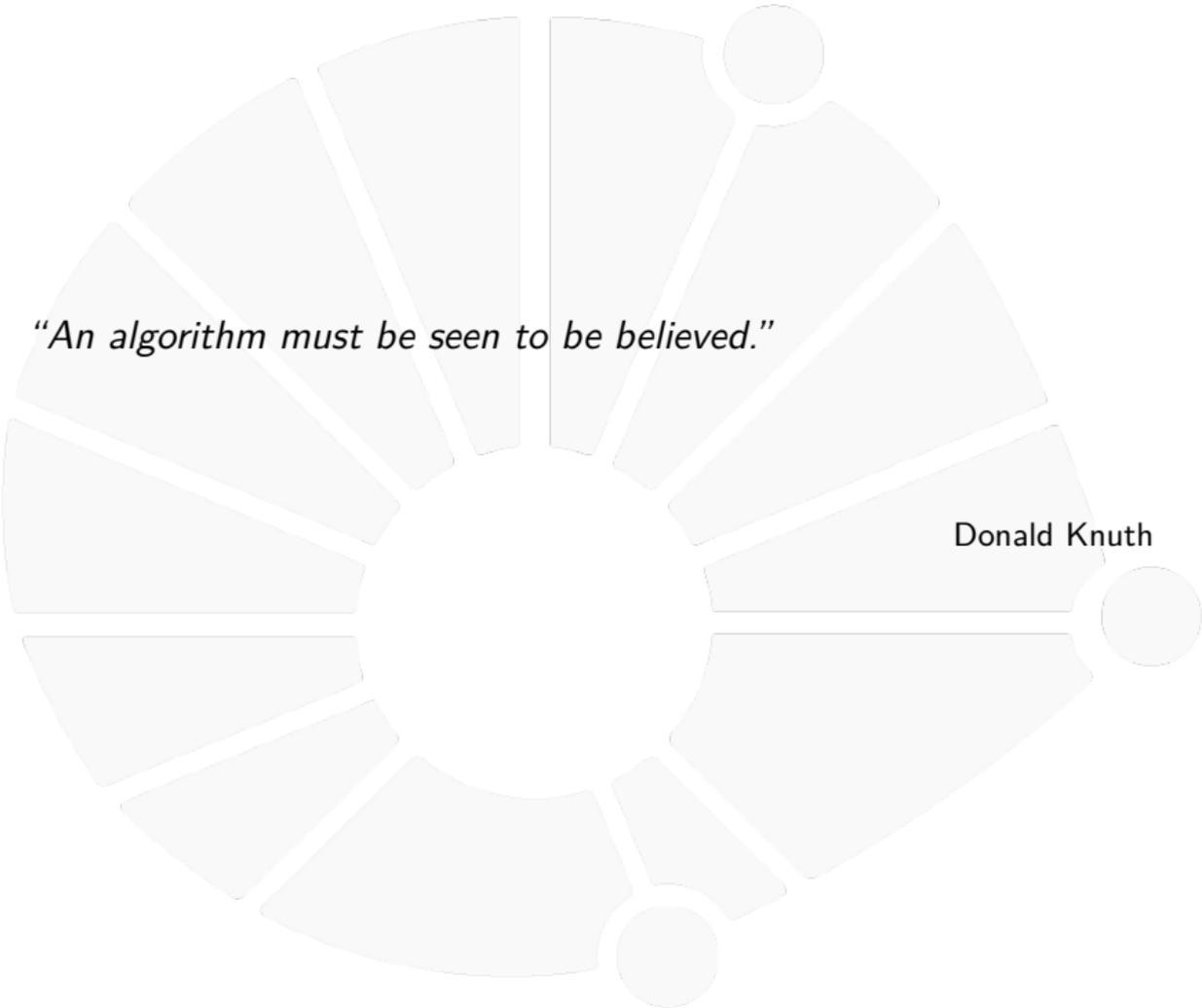
01/24

3



UNICAMP



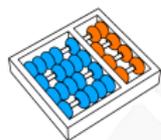


"An algorithm must be seen to be believed."

Donald Knuth

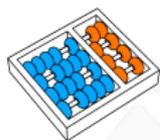


NOÇÕES BÁSICAS



Buscas em grafos

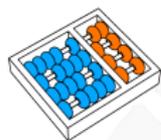
Como percorrer os vértices de um grafo?



Buscas em grafos

Como percorrer os vértices de um grafo?

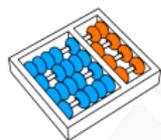
- ▶ Mais complicado que lista, vetor, árvore binária.



Buscas em grafos

Como percorrer os vértices de um grafo?

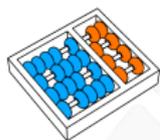
- ▶ Mais complicado que lista, vetor, árvore binária.
- ▶ Podem ser direcionados ou não direcionados.



Buscas em grafos

Como percorrer os vértices de um grafo?

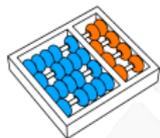
- ▶ Mais complicado que lista, vetor, árvore binária.
- ▶ Podem ser direcionados ou não direcionados.
- ▶ Queremos descobrir informações sobre sua estrutura.



Buscas em grafos

Como percorrer os vértices de um grafo?

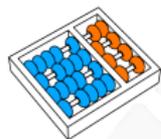
- ▶ Mais complicado que lista, vetor, árvore binária.
- ▶ Podem ser direcionados ou não direcionados.
- ▶ Queremos descobrir informações sobre sua estrutura.
- ▶ Podemos pensar em cada componente separadamente.



Buscas em grafos

Como percorrer os vértices de um grafo?

- ▶ Mais complicado que lista, vetor, árvore binária.
- ▶ Podem ser direcionados ou não direcionados.
- ▶ Queremos descobrir informações sobre sua estrutura.
- ▶ Podemos pensar em cada componente separadamente.
- ▶ **Objetivo:** encontrar uma **ÁRVORE GERADORA**.



Buscas em grafos

Dois algoritmos:



Buscas em grafos

Dois algoritmos:

1. Busca em largura (BFS, do inglês **BREADTH-FIRST SEARCH**).



Buscas em grafos

Dois algoritmos:

1. Busca em largura (BFS, do inglês **BREADTH-FIRST SEARCH**).
2. Busca em profundidade (DFS, do inglês **DEPTH-FIRST SEARCH**).



Representação de árvores

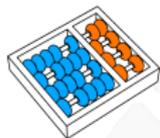
Como representar uma árvore de busca?



Representação de árvores

Como representar uma árvore de busca?

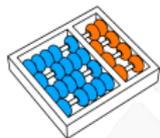
- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** *s*.



Representação de árvores

Como representar uma árvore de busca?

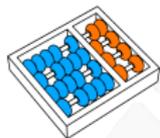
- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** s .
- ▶ A representamos com um vetor π de pais.



Representação de árvores

Como representar uma árvore de busca?

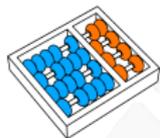
- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** s .
- ▶ A representamos com um vetor π de pais.
- ▶ O pai de um vértice v é $\pi[v]$.



Representação de árvores

Como representar uma árvore de busca?

- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** s .
- ▶ A representamos com um vetor π de pais.
- ▶ O pai de um vértice v é $\pi[v]$.
- ▶ convencionamos que $\pi[s] = \text{NIL}$

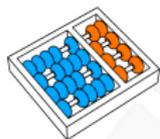


Representação de árvores

Como representar uma árvore de busca?

- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** s .
- ▶ A representamos com um vetor π de pais.
- ▶ O pai de um vértice v é $\pi[v]$.
- ▶ convencionamos que $\pi[s] = \text{NIL}$

Algumas propriedades:



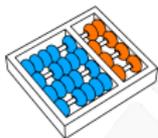
Representação de árvores

Como representar uma árvore de busca?

- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** s .
- ▶ A representamos com um vetor π de pais.
- ▶ O pai de um vértice v é $\pi[v]$.
- ▶ convencionamos que $\pi[s] = \text{NIL}$

Algumas propriedades:

- ▶ Existe aresta de $\pi[v]$ até v .



Representação de árvores

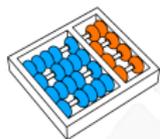
Como representar uma árvore de busca?

- ▶ A enraizamos em um **VÉRTICE DE ORIGEM** s .
- ▶ A representamos com um vetor π de pais.
- ▶ O pai de um vértice v é $\pi[v]$.
- ▶ convençionamos que $\pi[s] = \text{NIL}$

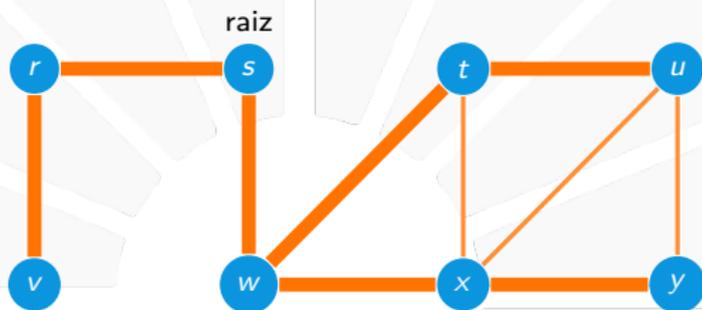
Algumas propriedades:

- ▶ Existe aresta de $\pi[v]$ até v .
- ▶ O caminho de s a v na árvore é:

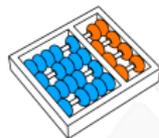
$$s \rightarrow \dots \rightarrow \pi[\pi[\pi[v]]] \rightarrow \pi[\pi[v]] \rightarrow \pi[v] \rightarrow v$$



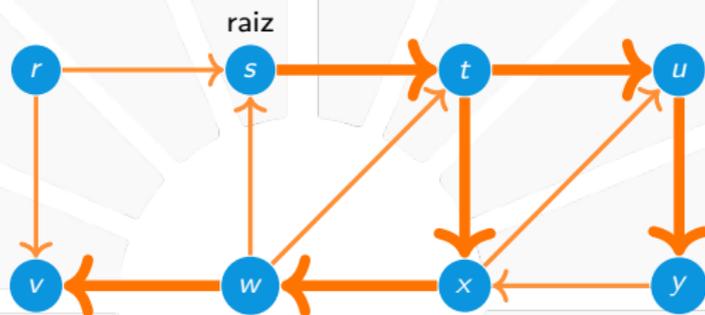
Exemplo com grafo não direcionado



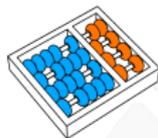
vértice	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	x



Exemplo com grafo direcionado



vértice	r	s	t	u	v	w	x	y
π	N	N	s	t	w	x	t	u

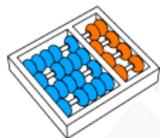


Caminho da árvore

Algoritmo 1: PRINT-PATH(G, s, v)

```
1 se  $v = s$ 
2   ┌ imprima  $s$ 
3 senão se  $\pi[v] = \text{NIL}$ 
4   ┌ imprima não existe caminho de  $s$  a  $v$ 
5 senão
6   ┌ PRINT-PATH( $G, s, \pi[v]$ )
7   ┌ imprima  $v$ 
```

- ▶ Imprime o caminho de s a v na árvore de raiz s .



Caminho da árvore

Algoritmo 2: PRINT-PATH(G, s, v)

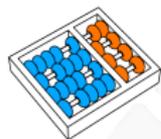
```
1 se  $v = s$ 
2   imprima  $s$ 
3 senão se  $\pi[v] = \text{NIL}$ 
4   imprima não existe caminho de  $s$  a  $v$ 
5 senão
6   PRINT-PATH( $G, s, \pi[v]$ )
7   imprima  $v$ 
```

- ▶ Imprime o caminho de s a v na árvore de raiz s .
- ▶ Gasta tempo linear no tamanho desse caminho.



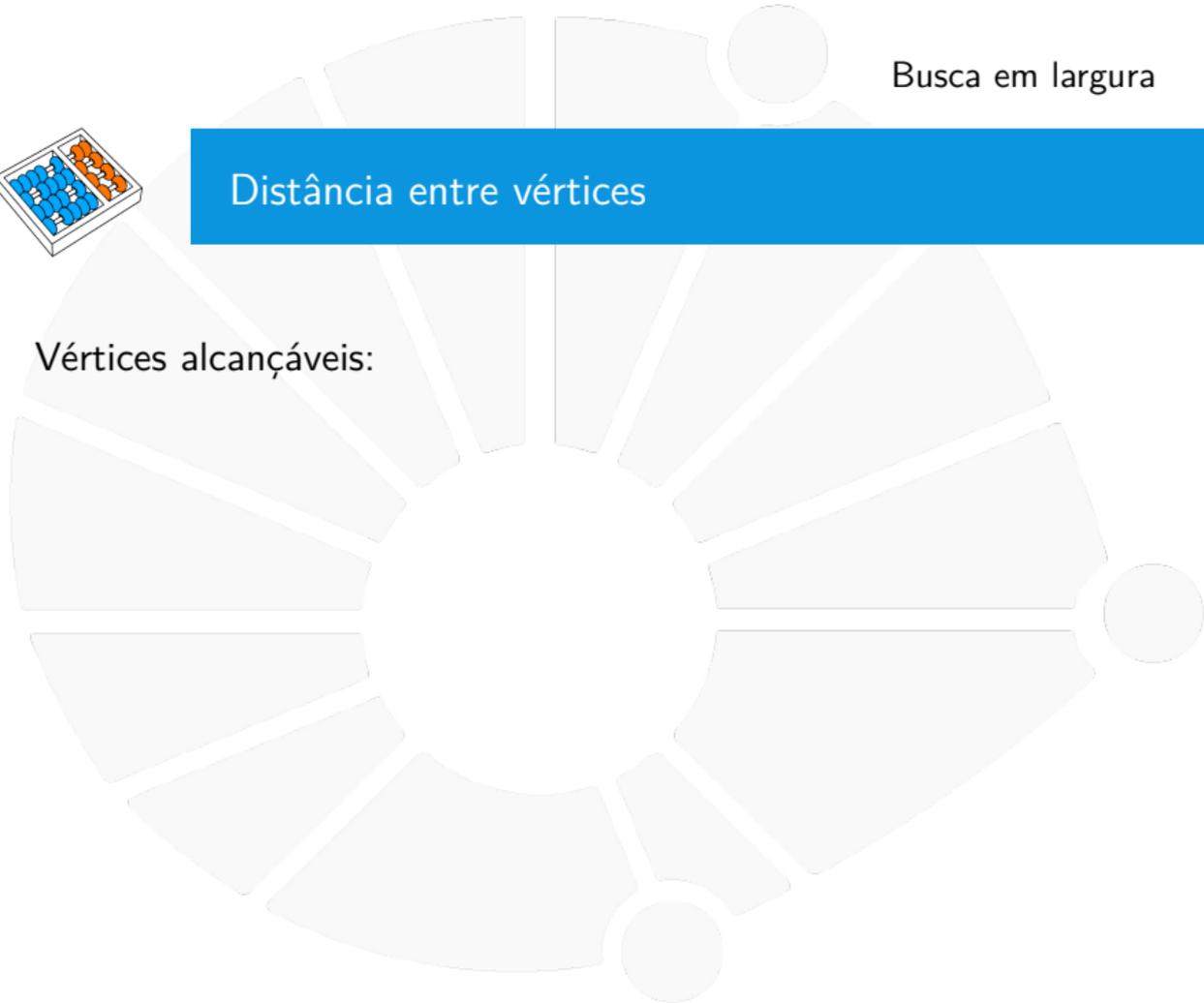
BUSCA EM LARGURA

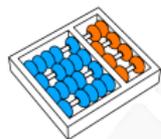
Busca em largura



Distância entre vértices

Vértices alcançáveis:

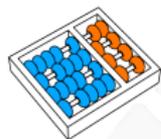




Distância entre vértices

Vértices alcançáveis:

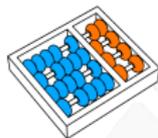
- ▶ Alcançamos v a partir de s se há caminho de s a v .



Distância entre vértices

Vértices alcançáveis:

- ▶ Alcançamos v a partir de s se há caminho de s a v .
- ▶ Pode haver diversos caminhos entre s a v .



Distância entre vértices

Vértices alcançáveis:

- ▶ Alcançamos **v** a partir de **s** se há caminho de **s** a **v**.
- ▶ Pode haver diversos caminhos entre **s** a **v**.
- ▶ Queremos algum com o menor **COMPRIMENTO**.

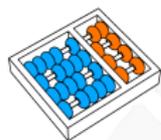


Distância entre vértices

Vértices alcançáveis:

- ▶ Alcançamos **v** a partir de **s** se há caminho de **s** a **v**.
- ▶ Pode haver diversos caminhos entre **s** a **v**.
- ▶ Queremos algum com o menor **COMPRIMENTO**.

A **DISTÂNCIA** de **s** a **v** é o comprimento de um caminho mais curto de **s** a **v**:



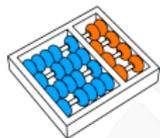
Distância entre vértices

Vértices alcançáveis:

- ▶ Alcançamos **v** a partir de **s** se há caminho de **s** a **v**.
- ▶ Pode haver diversos caminhos entre **s** a **v**.
- ▶ Queremos algum com o menor **COMPRIMENTO**.

A **DISTÂNCIA** de **s** a **v** é o comprimento de um caminho mais curto de **s** a **v**:

- ▶ Denotamos este valor por $\text{dist}(\mathbf{s}, \mathbf{v})$.



Distância entre vértices

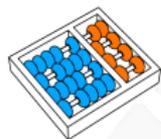
Vértices alcançáveis:

- ▶ Alcançamos **v** a partir de **s** se há caminho de **s** a **v**.
- ▶ Pode haver diversos caminhos entre **s** a **v**.
- ▶ Queremos algum com o menor **COMPRIMENTO**.

A **DISTÂNCIA** de **s** a **v** é o comprimento de um caminho mais curto de **s** a **v**:

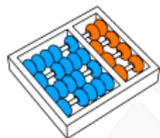
- ▶ Denotamos este valor por $\text{dist}(\mathbf{s}, \mathbf{v})$.
- ▶ Se **v** não for alcançável, definimos $\text{dist}(\mathbf{s}, \mathbf{v}) = \infty$.

Busca em largura



Busca em largura

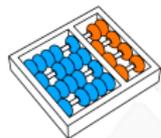
Buscando os vértices alcançáveis em **LARGURA**:



Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

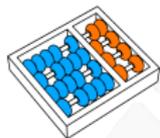
- ▶ Primeiro o vértice de origem.



Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

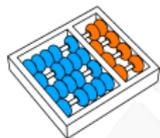
- ▶ Primeiro o vértice de origem.
- ▶ Depois os vizinhos do vértice de origem.



Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

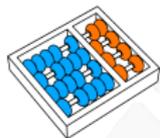
- ▶ Primeiro o vértice de origem.
- ▶ Depois os vizinhos do vértice de origem.
- ▶ Depois os vizinhos dos vizinhos do vértice de origem.



Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

- ▶ Primeiro o vértice de origem.
- ▶ Depois os vizinhos do vértice de origem.
- ▶ Depois os vizinhos dos vizinhos do vértice de origem.
- ▶ etc.

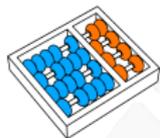


Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

- ▶ Primeiro o vértice de origem.
- ▶ Depois os vizinhos do vértice de origem.
- ▶ Depois os vizinhos dos vizinhos do vértice de origem.
- ▶ etc.

Descobrimo a distância



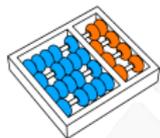
Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

- ▶ Primeiro o vértice de origem.
- ▶ Depois os vizinhos do vértice de origem.
- ▶ Depois os vizinhos dos vizinhos do vértice de origem.
- ▶ etc.

Descobrimo a distância

- ▶ Um produto da busca são as distâncias à origem.



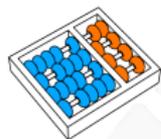
Busca em largura

Buscando os vértices alcançáveis em **LARGURA**:

- ▶ Primeiro o vértice de origem.
- ▶ Depois os vizinhos do vértice de origem.
- ▶ Depois os vizinhos dos vizinhos do vértice de origem.
- ▶ etc.

Descobrimo a distância

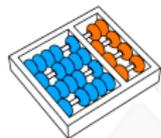
- ▶ Um produto da busca são as distâncias à origem.
- ▶ A árvore de busca fornece um caminho mais curto.



Construindo uma árvore de busca

Ideia do algoritmo:

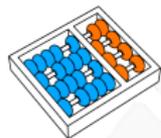




Construindo uma árvore de busca

Ideia do algoritmo:

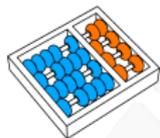
- ▶ Percorremos os vértices usando uma **FILA Q**.



Construindo uma árvore de busca

Ideia do algoritmo:

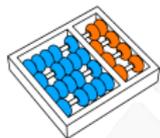
- ▶ Percorremos os vértices usando uma **FILA** Q .
- ▶ Começamos adicionando o vértice de origem s em Q .



Construindo uma árvore de busca

Ideia do algoritmo:

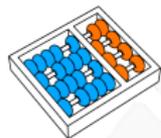
- ▶ Percorremos os vértices usando uma **FILA** Q .
- ▶ Começamos adicionando o vértice de origem s em Q .
- ▶ Enquanto houver vértices em Q , repetimos o seguinte processo:



Construindo uma árvore de busca

Ideia do algoritmo:

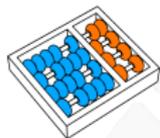
- ▶ Percorremos os vértices usando uma **FILA** Q .
- ▶ Começamos adicionando o vértice de origem s em Q .
- ▶ Enquanto houver vértices em Q , repetimos o seguinte processo:
 - ▶ Removemos o primeiro vértice de Q , u .



Construindo uma árvore de busca

Ideia do algoritmo:

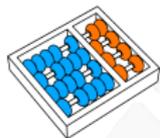
- ▶ Percorremos os vértices usando uma **FILA** Q .
- ▶ Começamos adicionando o vértice de origem s em Q .
- ▶ Enquanto houver vértices em Q , repetimos o seguinte processo:
 - ▶ Removemos o primeiro vértice de Q , u .
 - ▶ Para cada vizinho v do vértice atual u :



Construindo uma árvore de busca

Ideia do algoritmo:

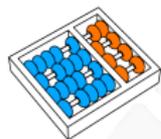
- ▶ Percorremos os vértices usando uma **FILA** Q .
- ▶ Começamos adicionando o vértice de origem s em Q .
- ▶ Enquanto houver vértices em Q , repetimos o seguinte processo:
 - ▶ Removemos o primeiro vértice de Q , u .
 - ▶ Para cada vizinho v do vértice atual u :
 - ▶ Adicionamos uma aresta (u,v) à árvore de busca.



Construindo uma árvore de busca

Ideia do algoritmo:

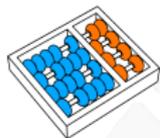
- ▶ Percorremos os vértices usando uma **FILA** Q .
- ▶ Começamos adicionando o vértice de origem s em Q .
- ▶ Enquanto houver vértices em Q , repetimos o seguinte processo:
 - ▶ Removemos o primeiro vértice de Q , u .
 - ▶ Para cada vizinho v do vértice atual u :
 - ▶ Adicionamos uma aresta (u,v) à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



Cores dos vértices

Vamos pintar o grafo durante a busca:

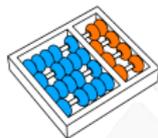




Cores dos vértices

Vamos pintar o grafo durante a busca:

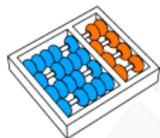
1. $\text{cor}[v] = \text{branco}$ se não descobrimos v ainda.



Cores dos vértices

Vamos pintar o grafo durante a busca:

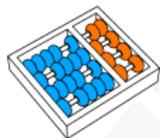
1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .



Cores dos vértices

Vamos pintar o grafo durante a busca:

1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

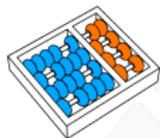


Cores dos vértices

Vamos pintar o grafo durante a busca:

1. $\text{cor}[\mathbf{v}] = \text{branco}$ se não descobrimos \mathbf{v} ainda.
2. $\text{cor}[\mathbf{v}] = \text{cinza}$ se já descobrimos, mas não finalizamos \mathbf{v} .
3. $\text{cor}[\mathbf{v}] = \text{preto}$ se já descobrimos e já finalizamos \mathbf{v} .

Observações:



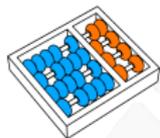
Cores dos vértices

Vamos pintar o grafo durante a busca:

1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

Observações:

- ▶ Não é necessário em uma implementação.



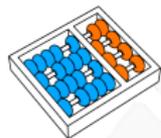
Cores dos vértices

Vamos pintar o grafo durante a busca:

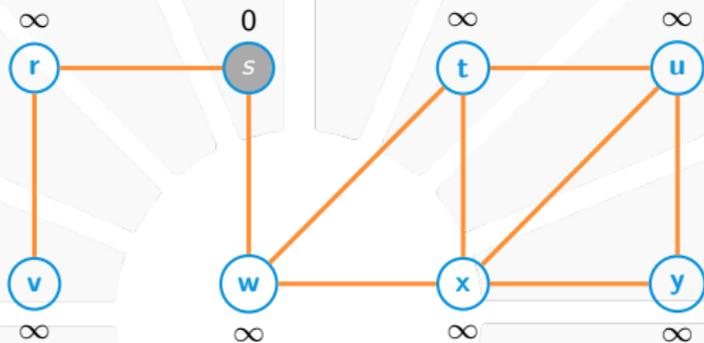
1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

Observações:

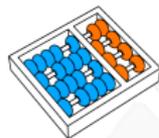
- ▶ Não é necessário em uma implementação.
- ▶ Facilita o entendimento do algoritmo.



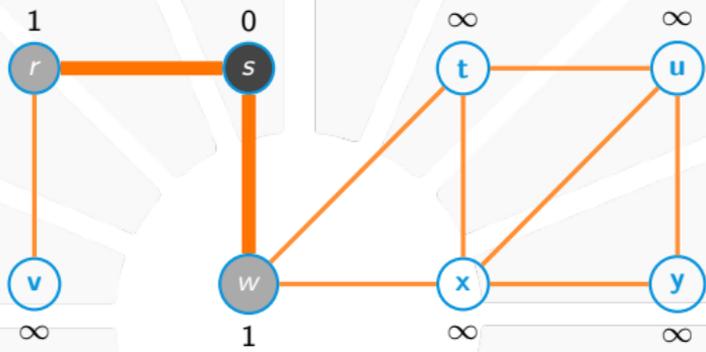
Exemplo de busca em largura



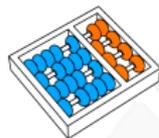
Q	s
distância	0



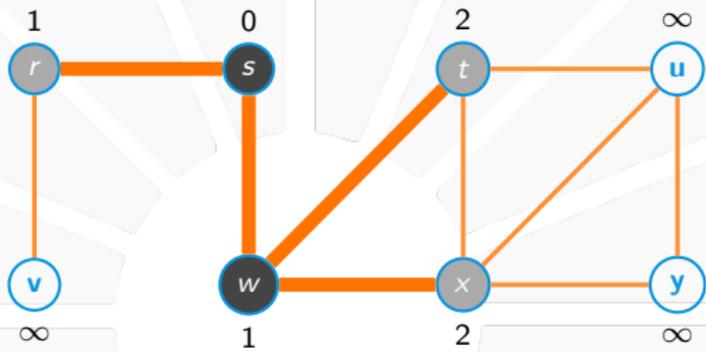
Exemplo de busca em largura



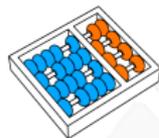
Q	w	r
distância	1	1



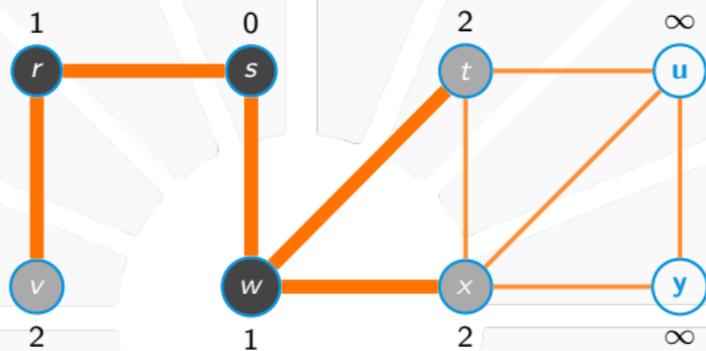
Exemplo de busca em largura



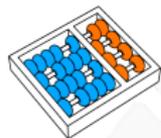
Q	r	t	x
distância	1	2	2



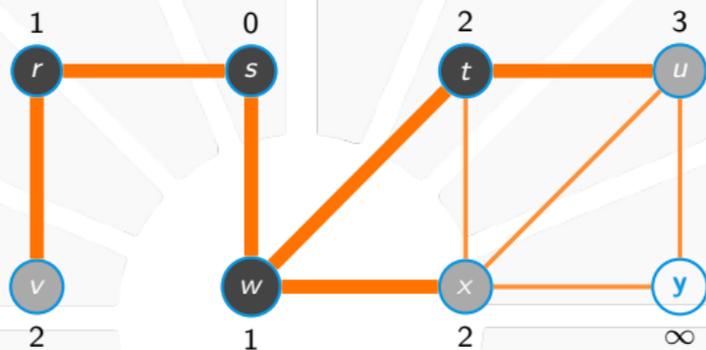
Exemplo de busca em largura



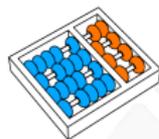
Q	t	x	v
distância	2	2	2



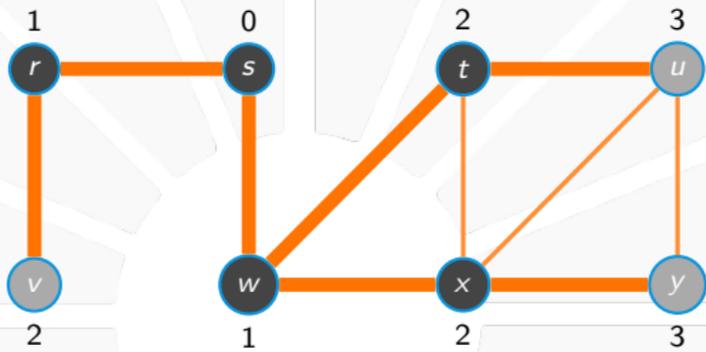
Exemplo de busca em largura



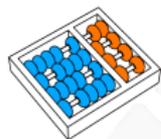
Q	x	v	u
distância	2	2	3



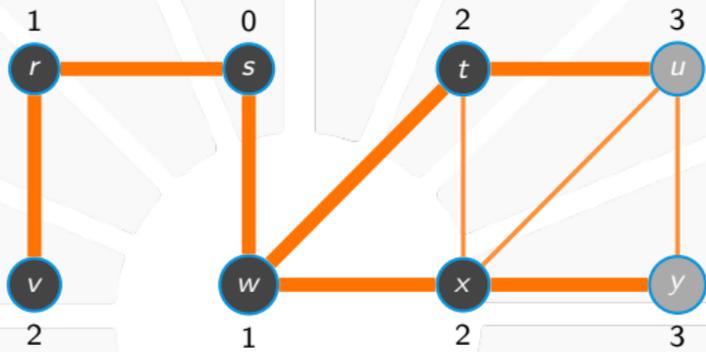
Exemplo de busca em largura



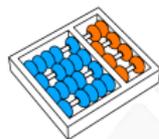
Q	v	u	y
distância	2	3	3



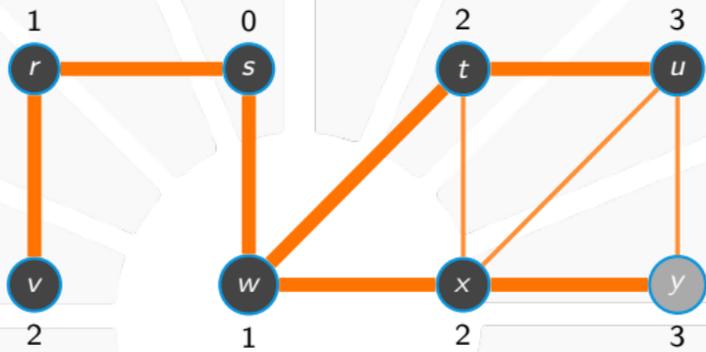
Exemplo de busca em largura



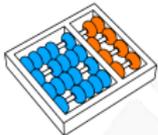
Q	u	y
distância	3	3



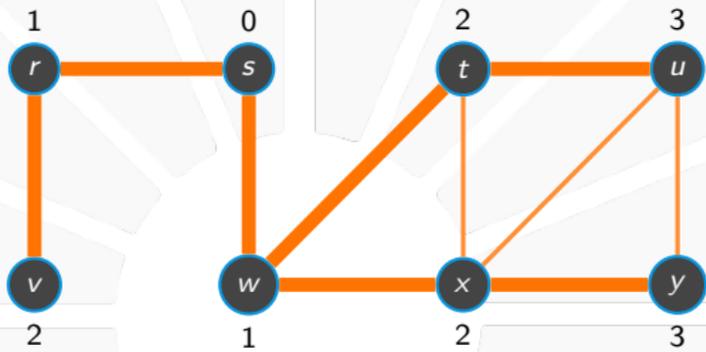
Exemplo de busca em largura



Q	y
distância	3

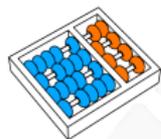


Exemplo de busca em largura



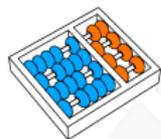
Q	\emptyset
distância	

Busca em largura



Algoritmo BFS

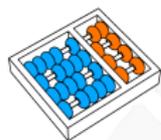
Observações:



Algoritmo BFS

Observações:

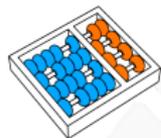
- ▶ Representamos G com listas de adjacências.



Algoritmo BFS

Observações:

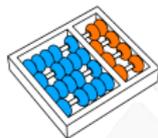
- ▶ Representamos G com listas de adjacências.
- ▶ A árvore de busca em largura é representada por π .



Algoritmo BFS

Observações:

- ▶ Representamos G com listas de adjacências.
- ▶ A árvore de busca em largura é representada por π .
- ▶ Calculamos a distância $d[v]$ de s a v .

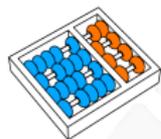


Algoritmo BFS

Algoritmo 3: BFS(G, s)

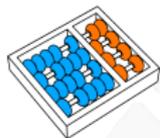
```

1  para cada  $u \in V[G]$ 
2     $\lfloor$  cor[ $u$ ]  $\leftarrow$  branco,  $d[u] \leftarrow \infty$ ,  $\pi[u] \leftarrow \text{NIL}$ 
3  cor[ $s$ ]  $\leftarrow$  cinza
4   $d[s] \leftarrow 0$ 
5   $Q \leftarrow \emptyset$ 
6  ENQUEUE( $Q, s$ )
7  enquanto  $Q \neq \emptyset$ 
8     $u \leftarrow$  DEQUEUE( $Q$ )
9    para cada  $v \in \text{Adj}[u]$ 
10     se cor[ $v$ ] = branco
11       cor[ $v$ ]  $\leftarrow$  cinza
12        $d[v] \leftarrow d[u] + 1$ 
13        $\pi[v] \leftarrow u$ 
14       ENQUEUE( $Q, v$ )
15  $\lfloor$  cor[ $u$ ]  $\leftarrow$  preto
  
```



Análise de complexidade

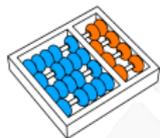
Analizamos de forma **AGREGADA**:



Análise de complexidade

Analisamos de forma **AGREGADA**:

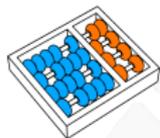
1. O tempo de inicialização é $O(V)$.



Análise de complexidade

Analisamos de forma **AGREGADA**:

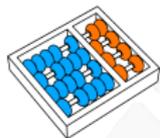
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:



Análise de complexidade

Analisamos de forma **AGREGADA**:

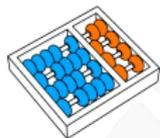
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.



Análise de complexidade

Analisamos de forma **AGREGADA**:

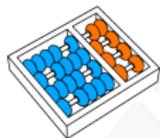
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.



Análise de complexidade

Analizamos de forma **AGREGADA**:

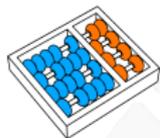
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.



Análise de complexidade

Analizamos de forma **AGREGADA**:

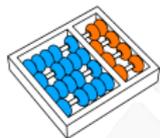
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.
 - ▶ O tempo gasto com a fila é $O(V)$.



Análise de complexidade

Analizamos de forma **AGREGADA**:

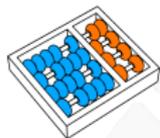
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.
 - ▶ O tempo gasto com a fila é $O(V)$.
3. Processamos cada vértice uma vez:



Análise de complexidade

Analizamos de forma **AGREGADA**:

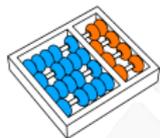
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.
 - ▶ O tempo gasto com a fila é $O(V)$.
3. Processamos cada vértice uma vez:
 - ▶ Cada lista de adjacências é percorrida uma vez.



Análise de complexidade

Analizamos de forma **AGREGADA**:

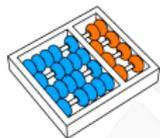
1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.
 - ▶ O tempo gasto com a fila é $O(V)$.
3. Processamos cada vértice uma vez:
 - ▶ Cada lista de adjacências é percorrida uma vez.
 - ▶ No pior caso, percorremos todas as listas.



Análise de complexidade

Analisamos de forma **AGREGADA**:

1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.
 - ▶ O tempo gasto com a fila é $O(V)$.
3. Processamos cada vértice uma vez:
 - ▶ Cada lista de adjacências é percorrida uma vez.
 - ▶ No pior caso, percorremos todas as listas.
 - ▶ O tempo gasto percorrendo adjacências é $O(E)$.

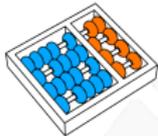


Análise de complexidade

Analizamos de forma **AGREGADA**:

1. O tempo de inicialização é $O(V)$.
2. Um vértice não volta a ser branco:
 - ▶ Enfileiramos cada vértice no máximo uma vez.
 - ▶ Desenfileiramos cada vértice no máximo uma vez.
 - ▶ Cada operação na fila leva tempo $O(1)$.
 - ▶ O tempo gasto com a fila é $O(V)$.
3. Processamos cada vértice uma vez:
 - ▶ Cada lista de adjacências é percorrida uma vez.
 - ▶ No pior caso, percorremos todas as listas.
 - ▶ O tempo gasto percorrendo adjacências é $O(E)$.

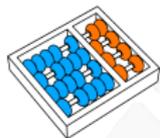
A complexidade da busca em largura é $O(V + E)$.



Correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

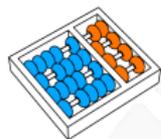


Correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,

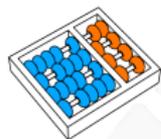


Correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.



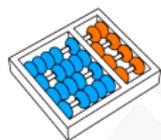
Correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Precisamos de dois lemas:



Correção do algoritmo

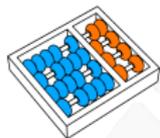
Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Precisamos de dois lemas:

- ▶ **Lema 1:** O caminho de s a v na árvore tem tamanho $d[v]$.



Correção do algoritmo

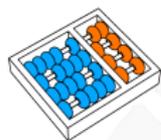
Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Precisamos de dois lemas:

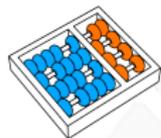
- ▶ **Lema 1:** O caminho de s a v na árvore tem tamanho $d[v]$.
- ▶ **Lema 2:** A fila Q respeita a ordem de $d[v]$.



Lema 1

Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

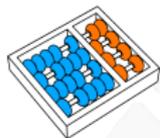


Lema 1

Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,

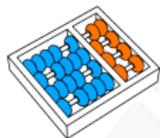


Lema 1

Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,
2. o caminho de s a v em T tem comprimento $d[v]$.



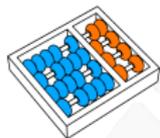
Lema 1

Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,
2. o caminho de s a v em T tem comprimento $d[v]$.

Demonstração:



Lema 1

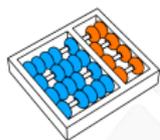
Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,
2. o caminho de s a v em T tem comprimento $d[v]$.

Demonstração:

- ▶ Por indução no número de vezes que executamos ENQUEUE.



Lema 1

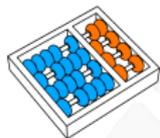
Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,
2. o caminho de s a v em T tem comprimento $d[v]$.

Demonstração:

- ▶ Por indução no número de vezes que executamos ENQUEUE.
- ▶ Após executar ENQUEUE pela primeira vez:



Lema 1

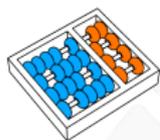
Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,
2. o caminho de s a v em T tem comprimento $d[v]$.

Demonstração:

- ▶ Por indução no número de vezes que executamos ENQUEUEE.
- ▶ Após executar ENQUEUEE pela primeira vez:
 - ▶ T contém apenas s e vale $d[s] = 0$



Lema 1

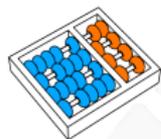
Lema (1)

Seja T a árvore induzida por π . Se $d[v] < \infty$, então:

1. v é um vértice de T ,
2. o caminho de s a v em T tem comprimento $d[v]$.

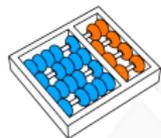
Demonstração:

- ▶ Por indução no número de vezes que executamos ENQUEUE.
- ▶ Após executar ENQUEUE pela primeira vez:
 - ▶ T contém apenas s e vale $d[s] = 0$
 - ▶ Como $d[s]$ nunca mais muda, isso completa a base.



Demonstração do lema

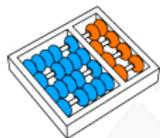
Considere o instante em que enfileiramos v :



Demonstração do lema

Considere o instante em que enfileiramos v :

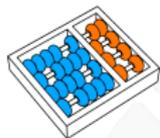
- ▶ Então, v foi descoberto percorrendo os vizinhos de u .



Demonstração do lema

Considere o instante em que enfileiramos **v**:

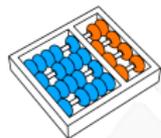
- ▶ Então, **v** foi descoberto percorrendo os vizinhos de **u**.
- ▶ Logo, **u** havia sido enfileirado antes desse instante.



Demonstração do lema

Considere o instante em que enfileiramos **v**:

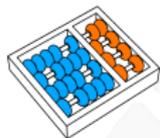
- ▶ Então, **v** foi descoberto percorrendo os vizinhos de **u**.
- ▶ Logo, **u** havia sido enfileirado antes desse instante.
- ▶ Pela hipótese de indução:



Demonstração do lema

Considere o instante em que enfileiramos v :

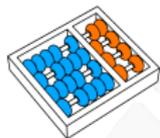
- ▶ Então, v foi descoberto percorrendo os vizinhos de u .
- ▶ Logo, u havia sido enfileirado antes desse instante.
- ▶ Pela hipótese de indução:
 1. Existe um caminho de s a u em T com comprimento $d[u]$.



Demonstração do lema

Considere o instante em que enfileiramos v :

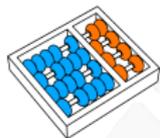
- ▶ Então, v foi descoberto percorrendo os vizinhos de u .
- ▶ Logo, u havia sido enfileirado antes desse instante.
- ▶ Pela hipótese de indução:
 1. Existe um caminho de s a u em T com comprimento $d[u]$.
- ▶ Portanto:



Demonstração do lema

Considere o instante em que enfileiramos v :

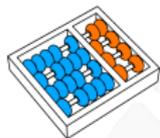
- ▶ Então, v foi descoberto percorrendo os vizinhos de u .
- ▶ Logo, u havia sido enfileirado antes desse instante.
- ▶ Pela hipótese de indução:
 1. Existe um caminho de s a u em T com comprimento $d[u]$.
- ▶ Portanto:
 1. Há um caminho de s a v em T , passando por u , com comprimento $d[v] = d[u] + 1$, pois $\pi[v] = u$.



Demonstração do lema

Considere o instante em que enfileiramos v :

- ▶ Então, v foi descoberto percorrendo os vizinhos de u .
- ▶ Logo, u havia sido enfileirado antes desse instante.
- ▶ Pela hipótese de indução:
 1. Existe um caminho de s a u em T com comprimento $d[u]$.
- ▶ Portanto:
 1. Há um caminho de s a v em T , passando por u , com comprimento $d[v] = d[u] + 1$, pois $\pi[v] = u$.
- ▶ Com isso, completamos a indução.



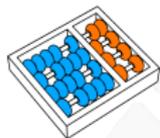
Demonstração do lema

Considere o instante em que enfileiramos v :

- ▶ Então, v foi descoberto percorrendo os vizinhos de u .
- ▶ Logo, u havia sido enfileirado antes desse instante.
- ▶ Pela hipótese de indução:
 1. Existe um caminho de s a u em T com comprimento $d[u]$.
- ▶ Portanto:
 1. Há um caminho de s a v em T , passando por u , com comprimento $d[v] = d[u] + 1$, pois $\pi[v] = u$.
- ▶ Com isso, completamos a indução.

Corolário (1)

Durante a execução, $d[v] \geq \text{dist}(s, v)$ para todo $v \in V$.

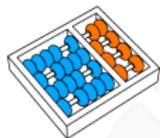


Lema 2

Lema (2)

Suponha que $\langle \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então

$$d[\mathbf{v}_1] \leq d[\mathbf{v}_2] \leq \dots \leq d[\mathbf{v}_r] \leq d[\mathbf{v}_1] + 1.$$



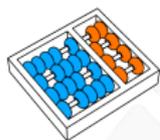
Lema 2

Lema (2)

Suponha que $\langle \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então

$$d[\mathbf{v}_1] \leq d[\mathbf{v}_2] \leq \dots \leq d[\mathbf{v}_r] \leq d[\mathbf{v}_1] + 1.$$

Demonstração:



Lema 2

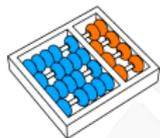
Lema (2)

Suponha que $\langle \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então

$$d[\mathbf{v}_1] \leq d[\mathbf{v}_2] \leq \dots \leq d[\mathbf{v}_r] \leq d[\mathbf{v}_1] + 1.$$

Demonstração:

- ▶ Por indução no número de iterações.



Lema 2

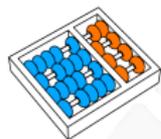
Lema (2)

Suponha que $\langle \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então

$$d[\mathbf{v}_1] \leq d[\mathbf{v}_2] \leq \dots \leq d[\mathbf{v}_r] \leq d[\mathbf{v}_1] + 1.$$

Demonstração:

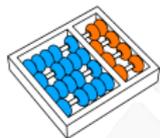
- ▶ Por indução no número de iterações.
- ▶ Antes da primeira iteração, $Q = \langle \mathbf{s} \rangle$ e o lema vale.



Demonstração do lema

Considere uma execução do laço:

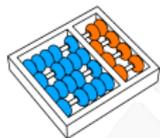




Demonstração do lema

Considere uma execução do laço:

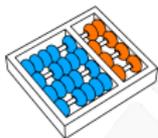
- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.



Demonstração do lema

Considere uma execução do laço:

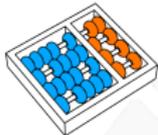
- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.
- ▶ Na iteração, removemos v_1 e inserimos v_{r+1}, \dots, v_{r+t} .



Demonstração do lema

Considere uma execução do laço:

- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.
- ▶ Na iteração, removemos v_1 e inserimos v_{r+1}, \dots, v_{r+t} .
- ▶ No final da iteração, a fila será $\langle v_2, \dots, v_r, v_{r+1}, \dots, v_{r+t} \rangle$.

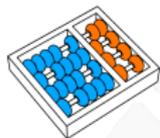


Demonstração do lema

Considere uma execução do laço:

- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.
- ▶ Na iteração, removemos v_1 e inserimos v_{r+1}, \dots, v_{r+t} .
- ▶ No final da iteração, a fila será $\langle v_2, \dots, v_r, v_{r+1}, \dots, v_{r+t} \rangle$.

Inserimos vizinhos de v_1 :



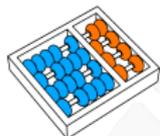
Demonstração do lema

Considere uma execução do laço:

- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.
- ▶ Na iteração, removemos v_1 e inserimos v_{r+1}, \dots, v_{r+t} .
- ▶ No final da iteração, a fila será $\langle v_2, \dots, v_r, v_{r+1}, \dots, v_{r+t} \rangle$.

Inserimos vizinhos de v_1 :

- ▶ Se v_j é um vértice inserido, então $d[v_j] = d[v_1] + 1$.



Demonstração do lema

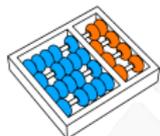
Considere uma execução do laço:

- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.
- ▶ Na iteração, removemos v_1 e inserimos v_{r+1}, \dots, v_{r+t} .
- ▶ No final da iteração, a fila será $\langle v_2, \dots, v_r, v_{r+1}, \dots, v_{r+t} \rangle$.

Inserimos vizinhos de v_1 :

- ▶ Se v_j é um vértice inserido, então $d[v_j] = d[v_1] + 1$.
- ▶ Pela hipótese de indução:

$$d[v_1] \leq d[v_2] \leq \dots \leq d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1.$$



Demonstração do lema

Considere uma execução do laço:

- ▶ No início da iteração, a fila é $\langle v_1, v_2, \dots, v_r \rangle$.
- ▶ Na iteração, removemos v_1 e inserimos v_{r+1}, \dots, v_{r+t} .
- ▶ No final da iteração, a fila será $\langle v_2, \dots, v_r, v_{r+1}, \dots, v_{r+t} \rangle$.

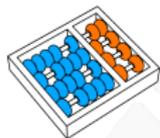
Inserimos vizinhos de v_1 :

- ▶ Se v_j é um vértice inserido, então $d[v_j] = d[v_1] + 1$.
- ▶ Pela hipótese de indução:

$$d[v_1] \leq d[v_2] \leq \dots \leq d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1.$$

- ▶ Portanto:

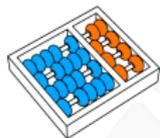
$$d[v_2] \leq \dots \leq d[v_r] \leq d[v_{r+1}] \leq \dots \leq d[v_{r+t}] \leq d[v_2] + 1.$$



Demonstração do teorema

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

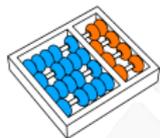


Demonstração do teorema

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,

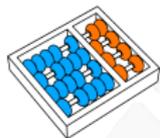


Demonstração do teorema

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.



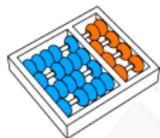
Demonstração do teorema

Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Demonstração:



Demonstração do teorema

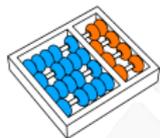
Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Demonstração:

- ▶ Note que π define uma árvore enraizada em s . Por quê?



Demonstração do teorema

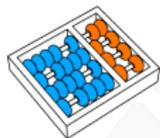
Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Demonstração:

- ▶ Note que π define uma árvore enraizada em s . Por quê?
- ▶ Pelo Corolário 1, se $\text{dist}(s, v) = \infty$, então $d[v] = \infty$.



Demonstração do teorema

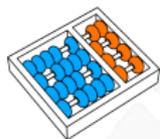
Teorema

Seja $G = (V, E)$ um grafo e s um vértice de G . Então, depois de executar $\text{BFS}(G, s)$, temos:

1. π define uma árvore enraizada em s ,
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

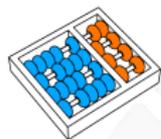
Demonstração:

- ▶ Note que π define uma árvore enraizada em s . Por quê?
- ▶ Pelo Corolário 1, se $\text{dist}(s, v) = \infty$, então $d[v] = \infty$.
- ▶ Resta provar que, se $\text{dist}(s, v) < \infty$, então $d[v] = \text{dist}(s, v)$.



Demonstração do teorema

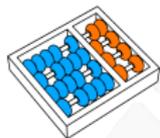
Considere um vértice v com $\text{dist}(s, v) = k$:



Demonstração do teorema

Considere um vértice v com $\text{dist}(s, v) = k$:

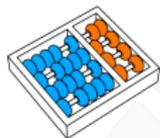
- ▶ Iremos provar que $d[v] = k$ por indução em k .



Demonstração do teorema

Considere um vértice v com $\text{dist}(s, v) = k$:

- ▶ Iremos provar que $d[v] = k$ por indução em k .
- ▶ Se $k = 0$, devemos ter $v = s$ e a afirmação vale.

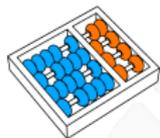


Demonstração do teorema

Considere um vértice v com $\text{dist}(s, v) = k$:

- ▶ Iremos provar que $d[v] = k$ por indução em k .
- ▶ Se $k = 0$, devemos ter $v = s$ e a afirmação vale.

Considere o caso em que $k \geq 1$. Por hipótese de indução, $d[u] = \text{dist}(s, u)$ para todo u com $\text{dist}(s, u) < k$:



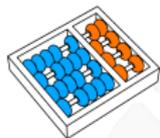
Demonstração do teorema

Considere um vértice v com $\text{dist}(s, v) = k$:

- ▶ Iremos provar que $d[v] = k$ por indução em k .
- ▶ Se $k = 0$, devemos ter $v = s$ e a afirmação vale.

Considere o caso em que $k \geq 1$. Por hipótese de indução, $d[u] = \text{dist}(s, u)$ para todo u com $\text{dist}(s, u) < k$:

- ▶ Seja v um vértice com $\text{dist}(s, v) = k$ e considere um caminho de comprimento k de s a v .



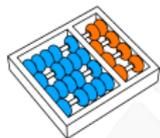
Demonstração do teorema

Considere um vértice v com $\text{dist}(s, v) = k$:

- ▶ Iremos provar que $d[v] = k$ por indução em k .
- ▶ Se $k = 0$, devemos ter $v = s$ e a afirmação vale.

Considere o caso em que $k \geq 1$. Por hipótese de indução, $d[u] = \text{dist}(s, u)$ para todo u com $\text{dist}(s, u) < k$:

- ▶ Seja v um vértice com $\text{dist}(s, v) = k$ e considere um caminho de comprimento k de s a v .
- ▶ Chame de u o vértice que antecede v nesse caminho.



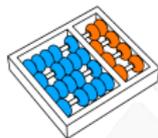
Demonstração do teorema

Considere um vértice v com $\text{dist}(s, v) = k$:

- ▶ Iremos provar que $d[v] = k$ por indução em k .
- ▶ Se $k = 0$, devemos ter $v = s$ e a afirmação vale.

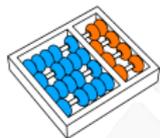
Considere o caso em que $k \geq 1$. Por hipótese de indução, $d[u] = \text{dist}(s, u)$ para todo u com $\text{dist}(s, u) < k$:

- ▶ Seja v um vértice com $\text{dist}(s, v) = k$ e considere um caminho de comprimento k de s a v .
- ▶ Chame de u o vértice que antecede v nesse caminho.
- ▶ Temos que, $\text{dist}(s, u) = k - 1$ e portanto $d[u] = k - 1$.



Demonstração do teorema

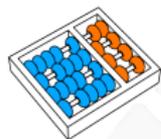
Considere o instante em que u foi removido de Q :



Demonstração do teorema

Considere o instante em que u foi removido de Q :

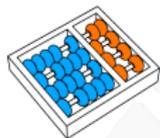
- ▶ Suponha (por contradição) que v seja preto:



Demonstração do teorema

Considere o instante em que u foi removido de Q :

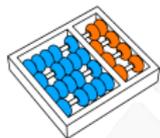
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .



Demonstração do teorema

Considere o instante em que u foi removido de Q :

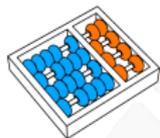
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.



Demonstração do teorema

Considere o instante em que u foi removido de Q :

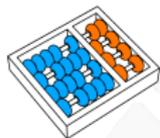
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.



Demonstração do teorema

Considere o instante em que u foi removido de Q :

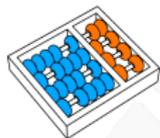
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.



Demonstração do teorema

Considere o instante em que u foi removido de Q :

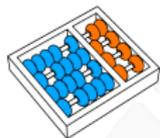
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:



Demonstração do teorema

Considere o instante em que u foi removido de Q :

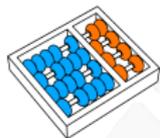
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:



Demonstração do teorema

Considere o instante em que u foi removido de Q :

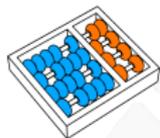
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.



Demonstração do teorema

Considere o instante em que u foi removido de Q :

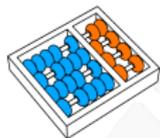
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.
 - ▶ Logo, $d[v] = d[u] + 1 = k$.



Demonstração do teorema

Considere o instante em que u foi removido de Q :

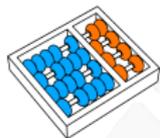
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.
 - ▶ Logo, $d[v] = d[u] + 1 = k$.
 - ▶ Se v for cinza:



Demonstração do teorema

Considere o instante em que u foi removido de Q :

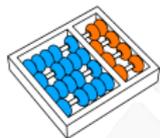
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.
 - ▶ Logo, $d[v] = d[u] + 1 = k$.
 - ▶ Se v for cinza:
 - ▶ v já estava na fila.



Demonstração do teorema

Considere o instante em que u foi removido de Q :

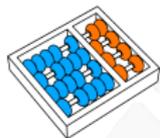
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.
 - ▶ Logo, $d[v] = d[u] + 1 = k$.
 - ▶ Se v for cinza:
 - ▶ v já estava na fila.
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] + 1 = k$



Demonstração do teorema

Considere o instante em que u foi removido de Q :

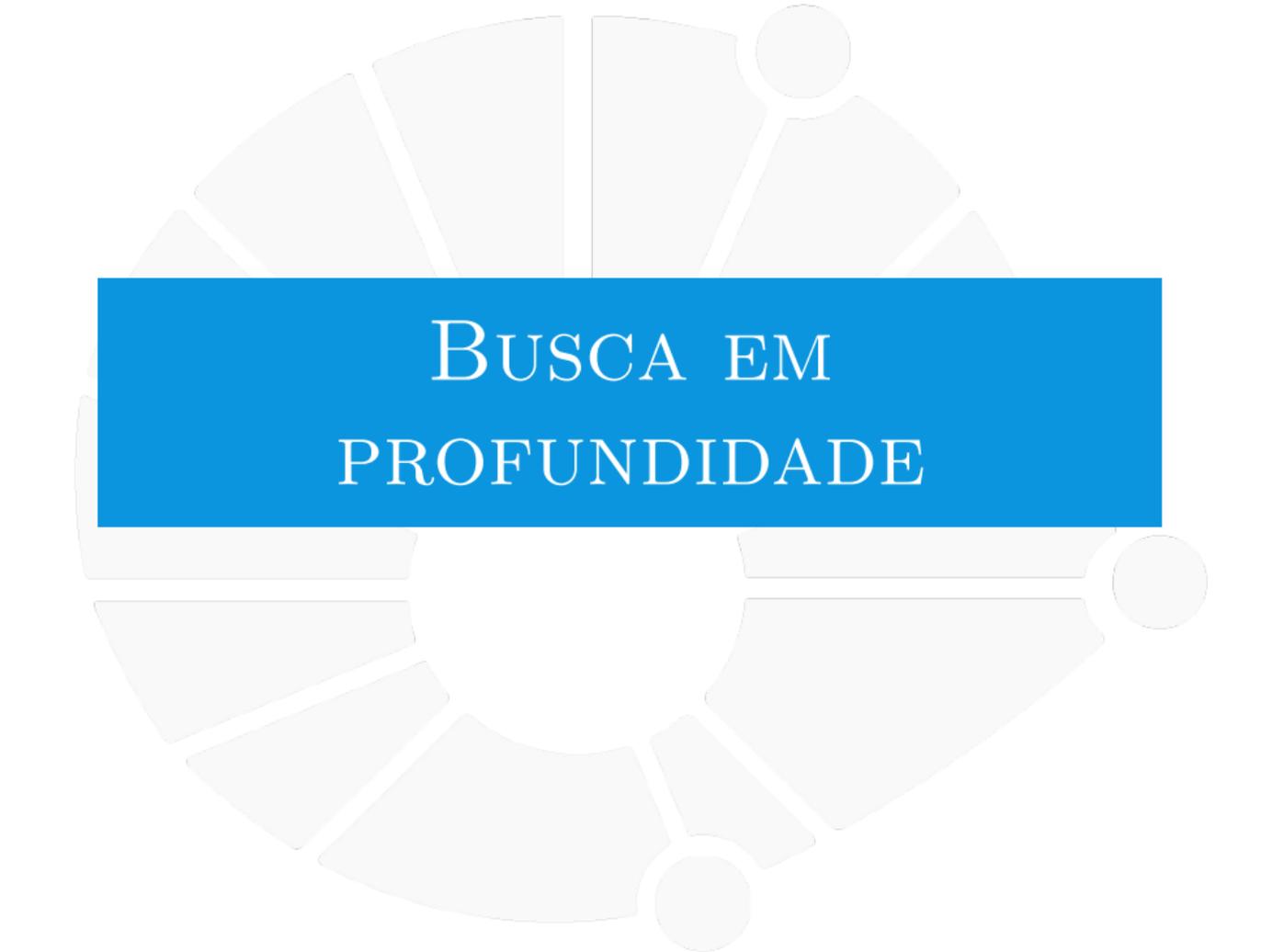
- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.
 - ▶ Logo, $d[v] = d[u] + 1 = k$.
 - ▶ Se v for cinza:
 - ▶ v já estava na fila.
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] + 1 = k$
 - ▶ Pelo Corolário 1 temos que $k \leq d[v]$, portanto $d[v] = k$.



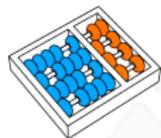
Demonstração do teorema

Considere o instante em que u foi removido de Q :

- ▶ Suponha (por contradição) que v seja preto:
 - ▶ Então v foi removido de Q antes de u .
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] < k$.
 - ▶ Mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$.
 - ▶ Logo, temos uma contradição e v **NÃO** pode ser preto.
- ▶ Assim, nesse instante, v só pode ser branco ou cinza:
 - ▶ Se v for branco:
 - ▶ v será inserido na fila nessa iteração.
 - ▶ Logo, $d[v] = d[u] + 1 = k$.
 - ▶ Se v for cinza:
 - ▶ v já estava na fila.
 - ▶ Pelo Lema 2 temos que $d[v] \leq d[u] + 1 = k$
 - ▶ Pelo Corolário 1 temos que $k \leq d[v]$, portanto $d[v] = k$.
- ▶ Em qualquer caso, concluímos a indução.



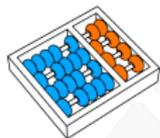
BUSCA EM
PROFUNDIDADE



Busca em profundidade

Busca em profundidade

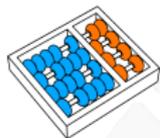
Buscando os vértices alcançáveis em **PROFUNDIDADE**:



Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

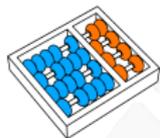
- ▶ Começamos com o vértice de origem.



Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

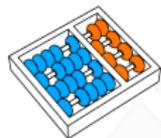
- ▶ Começamos com o vértice de origem.
- ▶ Depois, todos os alcançáveis pelo primeiro vizinho.



Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

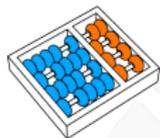
- ▶ Começamos com o vértice de origem.
- ▶ Depois, todos os alcançáveis pelo primeiro vizinho.
- ▶ Depois, todos os alcançáveis pelo segundo vizinho.



Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

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

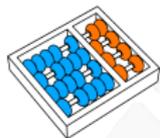


Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

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

É a estratégia usada por vários algoritmos:



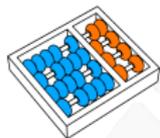
Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

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

É a estratégia usada por vários algoritmos:

- ▶ Identificar as componentes conexas.



Busca em profundidade

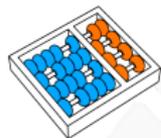
Buscando os vértices alcançáveis em **PROFUNDIDADE**:

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

É a estratégia usada por vários algoritmos:

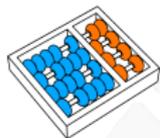
- ▶ Identificar as componentes conexas.
- ▶ Encontrar uma ordenação topológica.

Busca em profundidade



Construindo uma árvore de busca

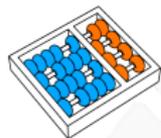
Ideia do algoritmo:



Construindo uma árvore de busca

Ideia do algoritmo:

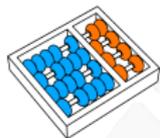
- ▶ Começamos com o vértice de origem **s**.



Construindo uma árvore de busca

Ideia do algoritmo:

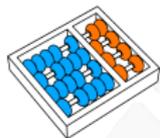
- ▶ Começamos com o vértice de origem **s**.
- ▶ Para cada vizinho não visitado **v** do vértice atual **u**:



Construindo uma árvore de busca

Ideia do algoritmo:

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

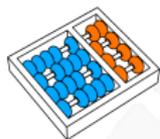


Construindo uma árvore de busca

Ideia do algoritmo:

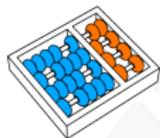
- ▶ Começamos com o vértice de origem **s**.
- ▶ Para cada vizinho não visitado **v** do vértice atual **u**:
 1. Adicionamos uma aresta **(u,v)** à árvore de busca.
 2. Visitamos **RECURSIVAMENTE** a partir de **v**.

Busca em profundidade



Alternativa

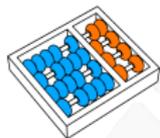
Ideia alternativa:



Alternativa

Ideia alternativa:

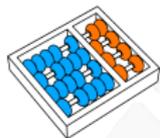
- ▶ Percorreremos os vértices usando uma **PILHA S**.



Alternativa

Ideia alternativa:

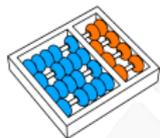
- ▶ Percorreremos os vértices usando uma **PILHA** S .
- ▶ Começamos adicionando o vértice de origem s em S .



Alternativa

Ideia alternativa:

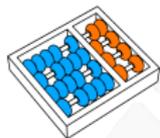
- ▶ Percorremos os vértices usando uma **PILHA** S .
- ▶ Começamos adicionando o vértice de origem s em S .
- ▶ Enquanto houver vértices em S , repetimos o seguinte processo:



Alternativa

Ideia alternativa:

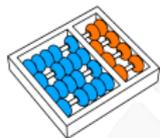
- ▶ Percorremos os vértices usando uma **PILHA** S .
- ▶ Começamos adicionando o vértice de origem s em S .
- ▶ Enquanto houver vértices em S , repetimos o seguinte processo:
 - ▶ Removemos o vértice do topo de S , u .



Alternativa

Ideia alternativa:

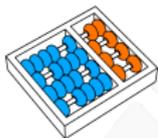
- ▶ Percorremos os vértices usando uma **PILHA** S .
- ▶ Começamos adicionando o vértice de origem s em S .
- ▶ Enquanto houver vértices em S , repetimos o seguinte processo:
 - ▶ Removemos o vértice do topo de S , u .
 - ▶ Para cada vizinho v do vértice atual u :



Alternativa

Ideia alternativa:

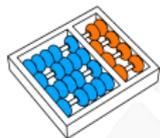
- ▶ Percorremos os vértices usando uma **PILHA** S .
- ▶ Começamos adicionando o vértice de origem s em S .
- ▶ Enquanto houver vértices em S , repetimos o seguinte processo:
 - ▶ Removemos o vértice do topo de S , u .
 - ▶ Para cada vizinho v do vértice atual u :
 - ▶ Adicionamos uma aresta (u,v) à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



Alternativa

Ideia alternativa:

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

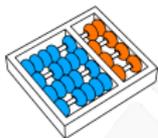


Alternativa

Ideia alternativa:

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

Observações:



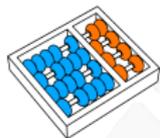
Alternativa

Ideia alternativa:

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

Observações:

- ▶ Pode levar a uma árvore de busca distinta à da primeira ideia.



Alternativa

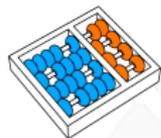
Ideia alternativa:

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

Observações:

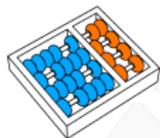
- ▶ Pode levar a uma árvore de busca distinta à da primeira ideia.
- ▶ Compare com fila da a busca em largura.

Busca em profundidade



Floresta de busca

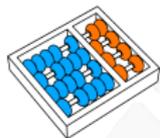
Visitando todos os vértices:



Floresta de busca

Visitando todos os vértices:

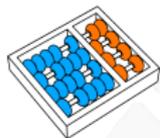
- ▶ A árvore de busca contém só vértices alcançáveis de **s**.



Floresta de busca

Visitando todos os vértices:

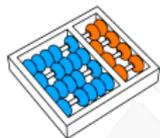
- ▶ A árvore de busca contém só vértices alcançáveis de s .
- ▶ Algumas vezes queremos visitar todos os vértices.



Floresta de busca

Visitando todos os vértices:

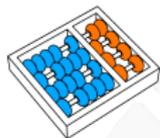
- ▶ A árvore de busca contém só vértices alcançáveis de s .
- ▶ Algumas vezes queremos visitar todos os vértices.
- ▶ Repetimos o processo com os vértices não visitados.



Floresta de busca

Visitando todos os vértices:

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

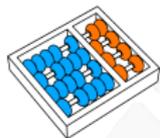


Floresta de busca

Visitando todos os vértices:

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

Representando uma floresta:



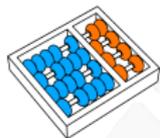
Floresta de busca

Visitando todos os vértices:

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

Representando uma floresta:

- ▶ Também utilizamos um vetor de pais π .



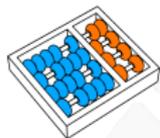
Floresta de busca

Visitando todos os vértices:

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

Representando uma floresta:

- ▶ Também utilizamos um vetor de pais π .
- ▶ Um vértice v com $\pi[v] = \text{NIL}$ é raiz de uma árvore de busca.



Floresta de busca

Visitando todos os vértices:

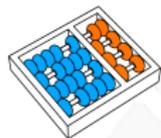
- ▶ A árvore de busca contém só vértices alcançáveis de s .
- ▶ Algumas vezes queremos visitar todos os vértices.
- ▶ Repetimos o processo com os vértices não visitados.
- ▶ Obteremos uma **FLORESTA DE BUSCA**.

Representando uma floresta:

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

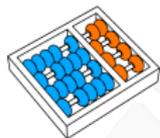
$$\{(\pi[v], v) : v \in V[G] \text{ e } \pi[v] \neq \text{NIL}\}$$

Busca em profundidade



Cores dos vértices

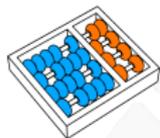
De novo, vamos pintar o grafo durante a busca:



Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

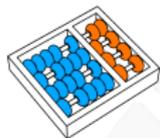
1. $\text{cor}[v] = \text{branco}$ se não descobrimos v ainda.



Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

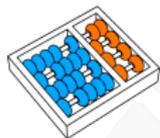
1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .



Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

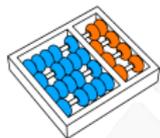


Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

Observações:



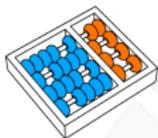
Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

Observações:

- ▶ Os vértices cinza têm suas chamadas recursivas ativas.



Cores dos vértices

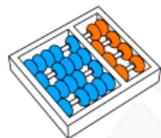
De novo, vamos pintar o grafo durante a busca:

1. $\text{cor}[v]$ = branco se não descobrimos v ainda.
2. $\text{cor}[v]$ = cinza se já descobrimos, mas não finalizamos v .
3. $\text{cor}[v]$ = preto se já descobrimos e já finalizamos v .

Observações:

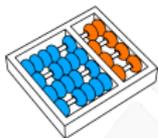
- ▶ Os vértices cinza têm suas chamadas recursivas ativas.
- ▶ A pilha de chamadas induz um caminho na floresta.

Busca em profundidade



Tempo de descoberta e finalização

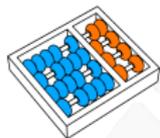
Ademais, vamos associar rótulos aos vértices:



Tempo de descoberta e finalização

Ademais, vamos associar rótulos aos vértices:

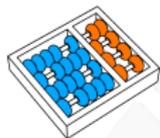
- ▶ $d[v]$ é o instante de **DESCOBERTA** de v .



Tempo de descoberta e finalização

Ademais, vamos associar rótulos aos vértices:

- ▶ $d[v]$ é o instante de **DESCOBERTA** de v .
- ▶ $f[v]$ é o instante de **FINALIZAÇÃO** de v .

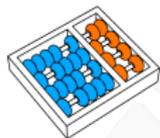


Tempo de descoberta e finalização

Ademais, vamos associar rótulos aos vértices:

- ▶ $d[v]$ é o instante de **DESCOBERTA** de v .
- ▶ $f[v]$ é o instante de **FINALIZAÇÃO** de v .

Observações:



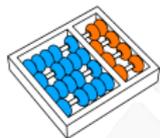
Tempo de descoberta e finalização

Ademais, vamos associar rótulos aos vértices:

- ▶ $d[v]$ é o instante de **DESCOBERTA** de v .
- ▶ $f[v]$ é o instante de **FINALIZAÇÃO** de v .

Observações:

- ▶ Os rótulos são inteiros distintos entre 1 e $2|V|$.



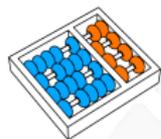
Tempo de descoberta e finalização

Ademais, vamos associar rótulos aos vértices:

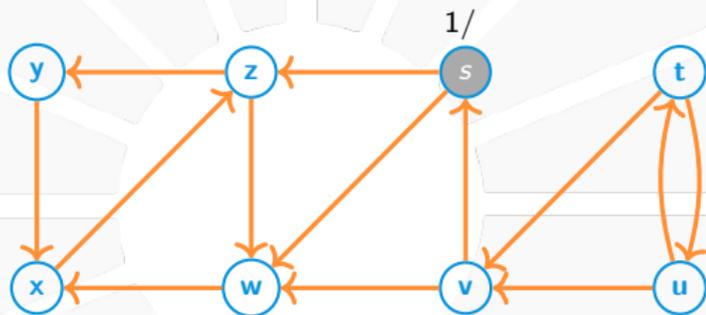
- ▶ $d[v]$ é o instante de **DESCOBERTA** de v .
- ▶ $f[v]$ é o instante de **FINALIZAÇÃO** de v .

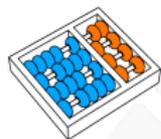
Observações:

- ▶ Os rótulos são inteiros distintos entre 1 e $2|V|$.
- ▶ Os rótulos refletem os instantes em que v muda de cor.

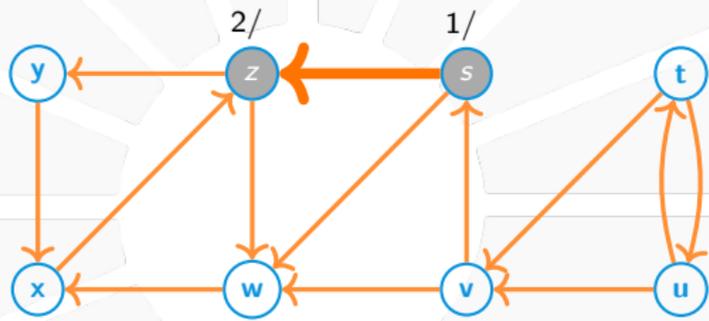


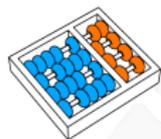
Exemplo de busca em profundidade



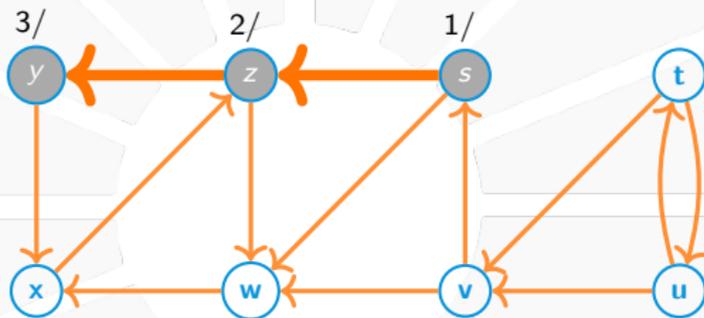


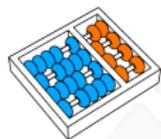
Exemplo de busca em profundidade



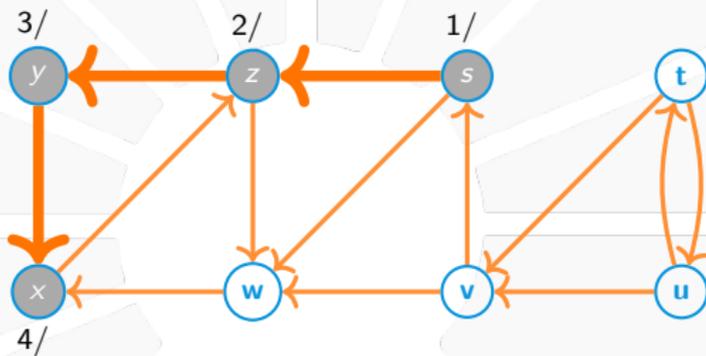


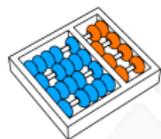
Exemplo de busca em profundidade



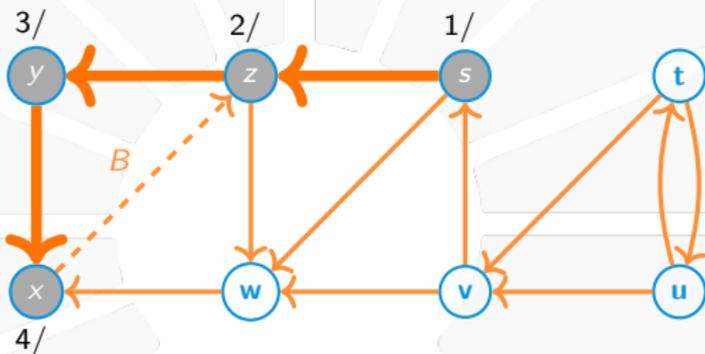


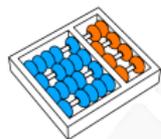
Exemplo de busca em profundidade



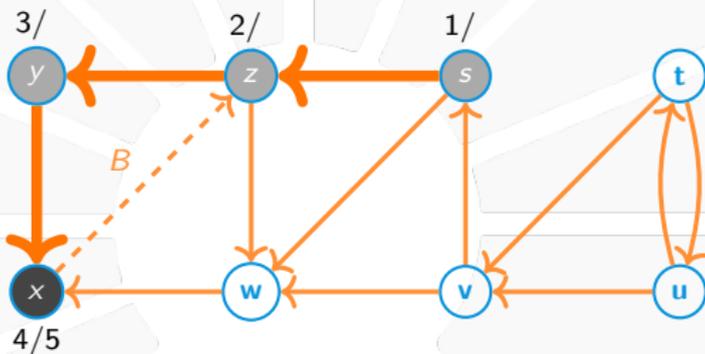


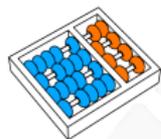
Exemplo de busca em profundidade



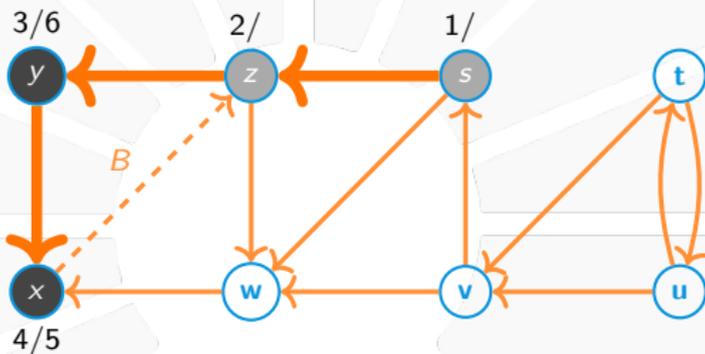


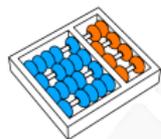
Exemplo de busca em profundidade



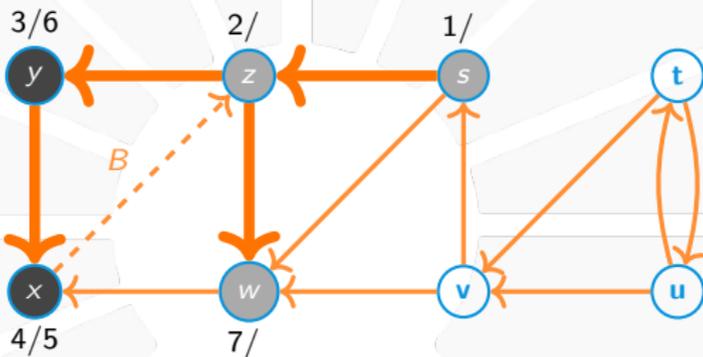


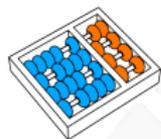
Exemplo de busca em profundidade



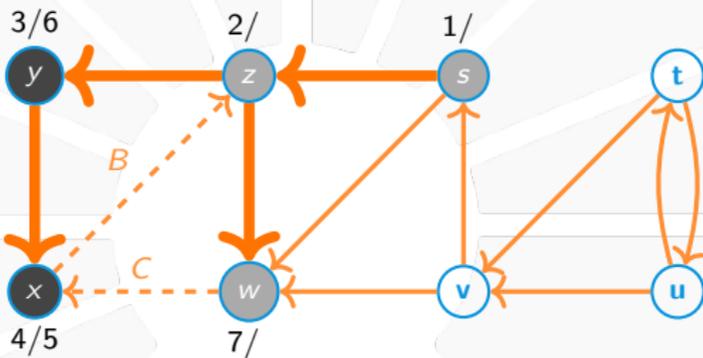


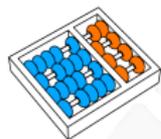
Exemplo de busca em profundidade



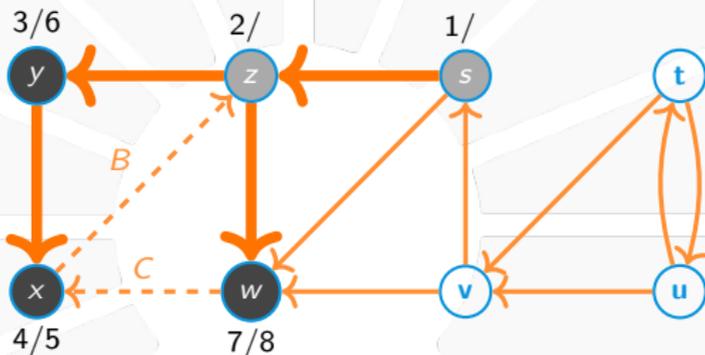


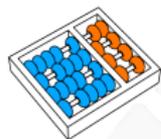
Exemplo de busca em profundidade



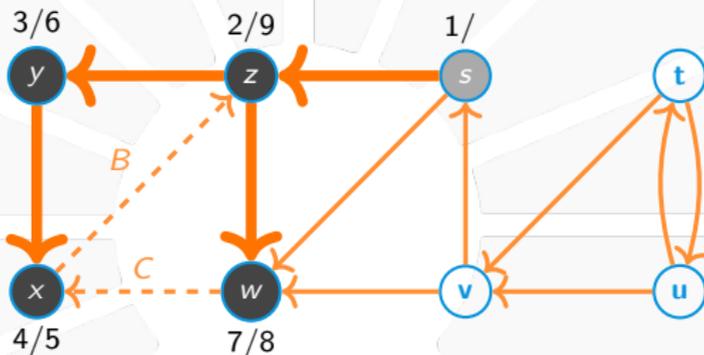


Exemplo de busca em profundidade



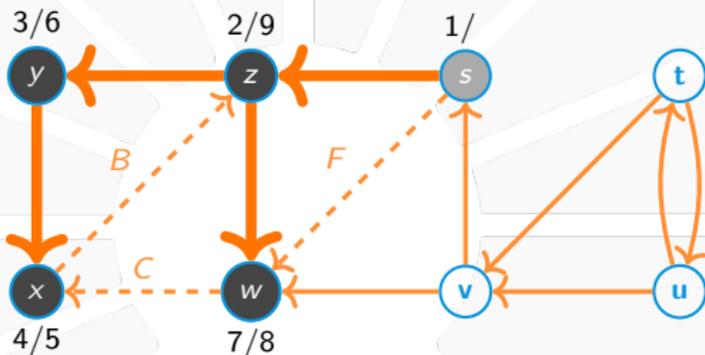


Exemplo de busca em profundidade



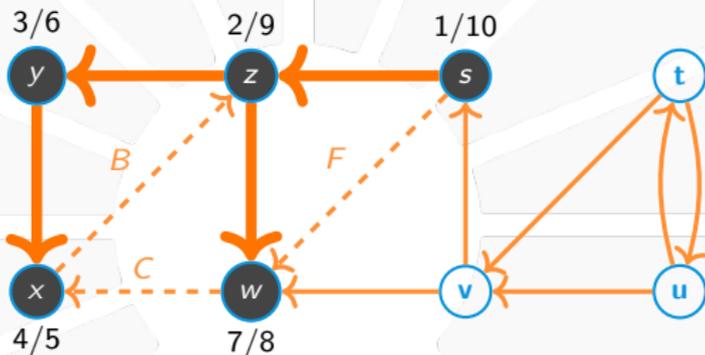


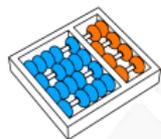
Exemplo de busca em profundidade



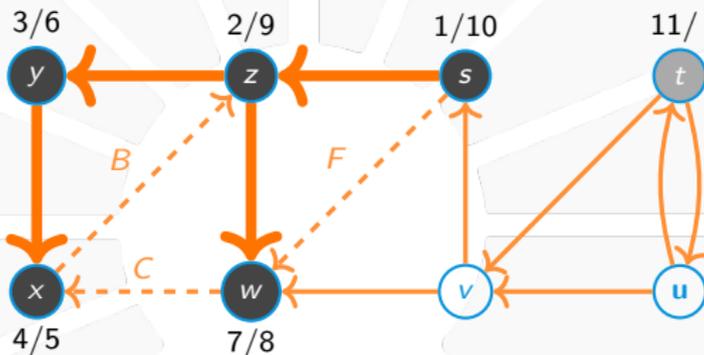


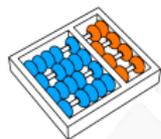
Exemplo de busca em profundidade



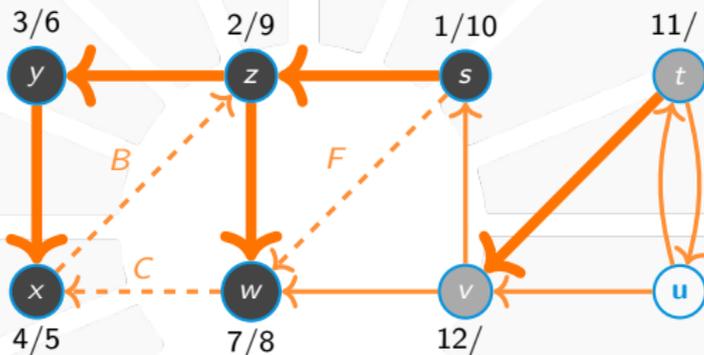


Exemplo de busca em profundidade



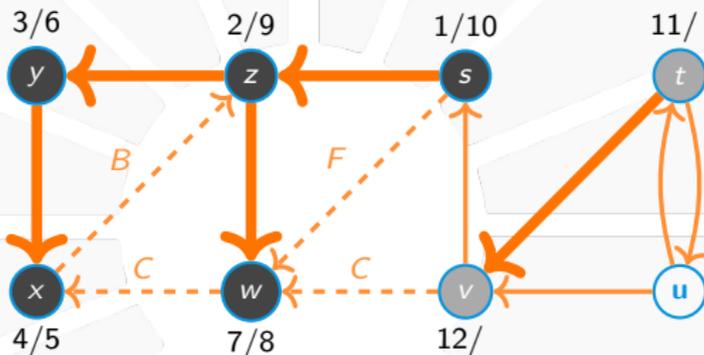


Exemplo de busca em profundidade



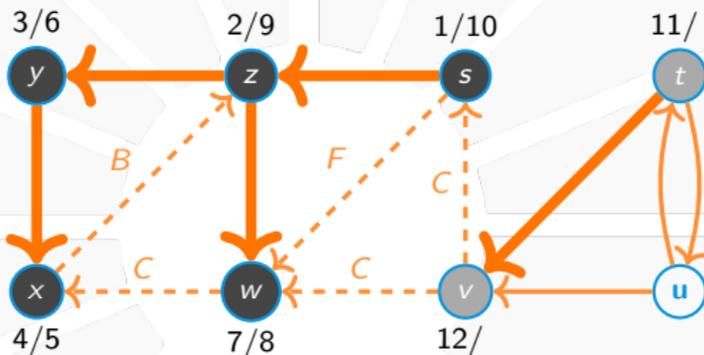


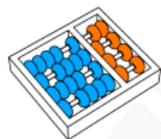
Exemplo de busca em profundidade



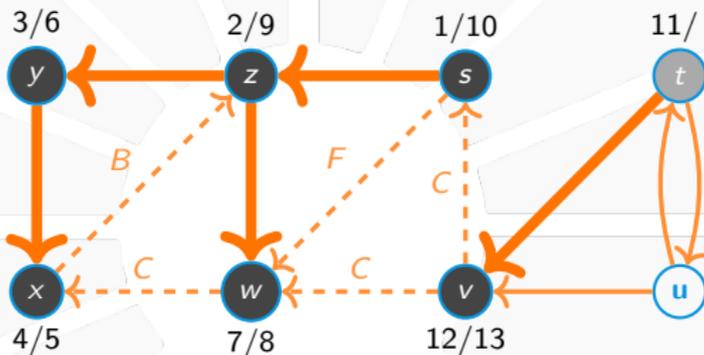


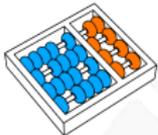
Exemplo de busca em profundidade



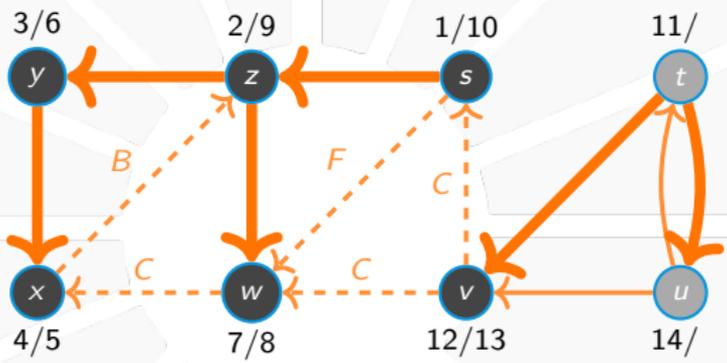


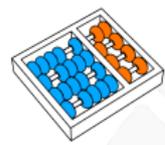
Exemplo de busca em profundidade



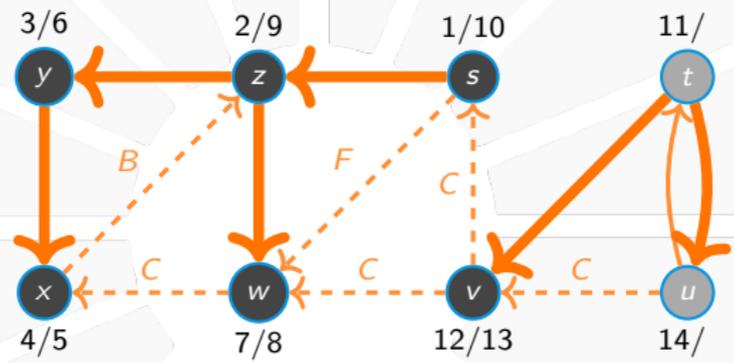


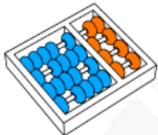
Exemplo de busca em profundidade



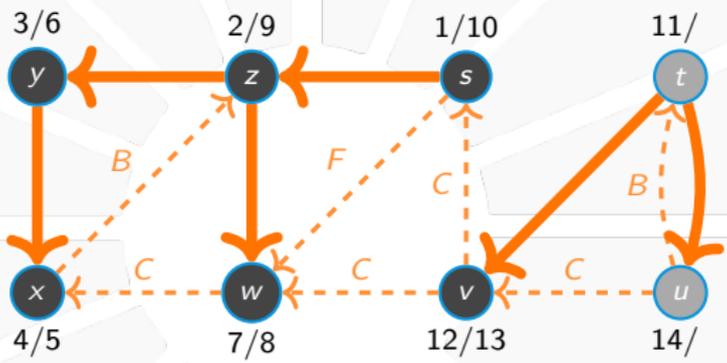


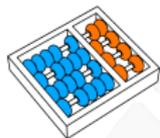
Exemplo de busca em profundidade



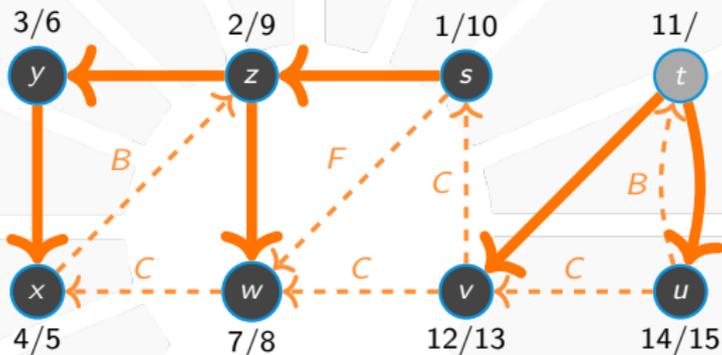


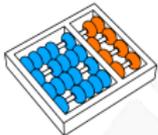
Exemplo de busca em profundidade



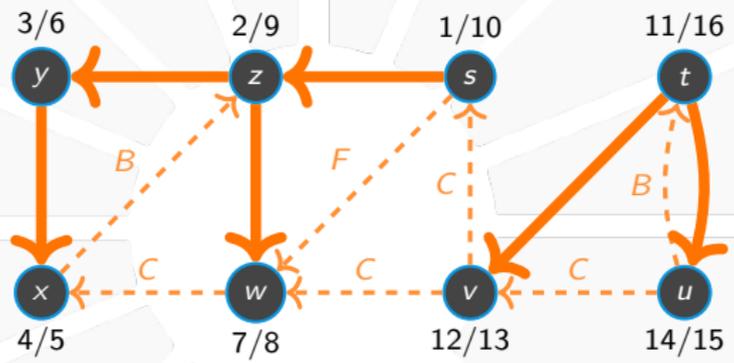


Exemplo de busca em profundidade

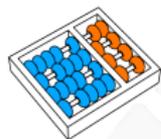




Exemplo de busca em profundidade

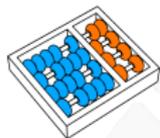


Busca em profundidade



Rótulos versus cores

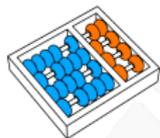
Observe que para todo vértice v :



Rótulos versus cores

Observe que para todo vértice v :

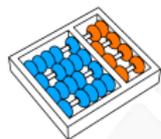
- ▶ v é branco antes do instante $d[v]$.



Rótulos versus cores

Observe que para todo vértice v :

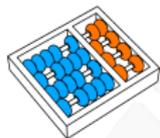
- ▶ v é branco antes do instante $d[v]$.
- ▶ v é cinza entre os instantes $d[v]$ e $f[v]$.



Rótulos versus cores

Observe que para todo vértice v :

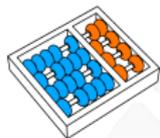
- ▶ v é branco antes do instante $d[v]$.
- ▶ v é cinza entre os instantes $d[v]$ e $f[v]$.
- ▶ v é preto após o instante $f[v]$.



Algoritmo DFS

Algoritmo 4: DFS(G)

```
1 para cada  $u \in V[G]$ 
2   | cor[ $u$ ]  $\leftarrow$  branco
3   |  $\pi[u] \leftarrow$  NIL
4 tempo  $\leftarrow$  0
5 para cada  $u \in V[G]$ 
6   | se cor[ $u$ ] = branco
7   |   | DFS-VISIT( $u$ )
```

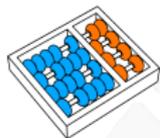


Algoritmo DFS

Algoritmo 5: DFS(G)

```
1 para cada  $u \in V[G]$ 
2   | cor[ $u$ ]  $\leftarrow$  branco
3   |  $\pi[u] \leftarrow$  NIL
4 tempo  $\leftarrow$  0
5 para cada  $u \in V[G]$ 
6   | se cor[ $u$ ] = branco
7   |   | DFS-VISIT( $u$ )
```

-
- Representamos G com listas de adjacências.

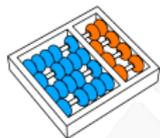


Algoritmo DFS

Algoritmo 6: DFS(G)

```
1 para cada  $u \in V[G]$ 
2    $\text{cor}[u] \leftarrow \text{branco}$ 
3    $\pi[u] \leftarrow \text{NIL}$ 
4 tempo  $\leftarrow 0$ 
5 para cada  $u \in V[G]$ 
6   se  $\text{cor}[u] = \text{branco}$ 
7      $\text{DFS-VISIT}(u)$ 
```

- ▶ Representamos G com listas de adjacências.
- ▶ A floresta de busca em profundidade é representada por π .

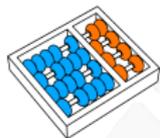


Algoritmo DFS

Algoritmo 7: DFS(G)

```
1 para cada  $u \in V[G]$ 
2   cor[ $u$ ]  $\leftarrow$  branco
3    $\pi[u] \leftarrow$  NIL
4 tempo  $\leftarrow$  0
5 para cada  $u \in V[G]$ 
6   se cor[ $u$ ] = branco
7     DFS-VISIT( $u$ )
```

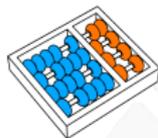
- ▶ Representamos G com listas de adjacências.
- ▶ A floresta de busca em profundidade é representada por π .
- ▶ São calculados os instantes $d[v]$ e $f[v]$.



Algoritmo DFS-VISIT

Algoritmo 8: DFS-VISIT(u)

- 1 $cor[u] \leftarrow$ cinza
 - 2 $tempo \leftarrow$ tempo + 1
 - 3 $d[u] \leftarrow$ tempo
 - 4 **para cada** $v \in Adj[u]$
 - 5 **se** $cor[v] =$ branco
 - 6 $\pi[v] \leftarrow u$
 - 7 DFS-VISIT(v)
 - 8 $cor[u] \leftarrow$ preto
 - 9 $tempo \leftarrow$ tempo + 1
 - 10 $f[u] \leftarrow$ tempo
-



Algoritmo DFS-VISIT

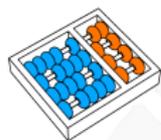
Algoritmo 9: DFS-VISIT(u)

```

1 cor[ $u$ ]  $\leftarrow$  cinza
2 tempo  $\leftarrow$  tempo + 1
3  $d[u] \leftarrow$  tempo
4 para cada  $v \in Adj[u]$ 
5   se cor[ $v$ ] = branco
6      $\pi[v] \leftarrow u$ 
7     DFS-VISIT( $v$ )
8 cor[ $u$ ]  $\leftarrow$  preto
9 tempo  $\leftarrow$  tempo + 1
10  $f[u] \leftarrow$  tempo

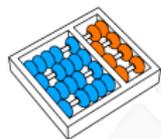
```

Constrói uma árvore de busca com origem u .



Análise de complexidade

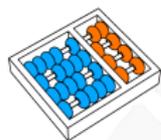
Tempo do algoritmo principal DFS:



Análise de complexidade

Tempo do algoritmo principal DFS:

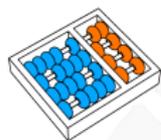
- ▶ A inicialização consome tempo $O(V)$.



Análise de complexidade

Tempo do algoritmo principal DFS:

- ▶ A inicialização consome tempo $O(V)$.
- ▶ Realizamos $|V|$ chamadas a DFS-VISIT.

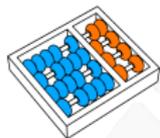


Análise de complexidade

Tempo do algoritmo principal DFS:

- ▶ A inicialização consome tempo $O(V)$.
- ▶ Realizamos $|V|$ chamadas a DFS-VISIT.

Tempo da sub-rotina DFS-VISIT:



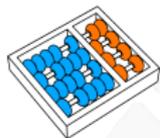
Análise de complexidade

Tempo do algoritmo principal DFS:

- ▶ A inicialização consome tempo $O(V)$.
- ▶ Realizamos $|V|$ chamadas a DFS-VISIT.

Tempo da sub-rotina DFS-VISIT:

- ▶ Processamos cada vértice exatamente uma vez.



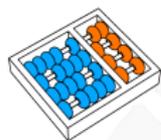
Análise de complexidade

Tempo do algoritmo principal DFS:

- ▶ A inicialização consome tempo $O(V)$.
- ▶ Realizamos $|V|$ chamadas a DFS-VISIT.

Tempo da sub-rotina DFS-VISIT:

- ▶ Processamos cada vértice exatamente uma vez.
- ▶ Cada chamada percorre sua lista de adjacências.



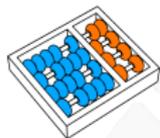
Análise de complexidade

Tempo do algoritmo principal DFS:

- ▶ A inicialização consome tempo $O(V)$.
- ▶ Realizamos $|V|$ chamadas a DFS-VISIT.

Tempo da sub-rotina DFS-VISIT:

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



Análise de complexidade

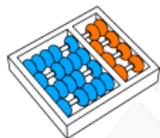
Tempo do algoritmo principal DFS:

- ▶ A inicialização consome tempo $O(V)$.
- ▶ Realizamos $|V|$ chamadas a DFS-VISIT.

Tempo da sub-rotina DFS-VISIT:

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

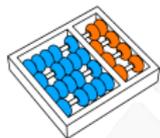
A complexidade da busca em profundidade é $O(V + E)$.



Teorema dos parênteses

Teorema

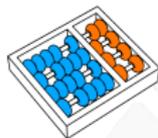
Se u e v são vértices de uma árvore de busca em profundidade, então ocorre exatamente um entre os três casos abaixo:



Teorema dos parênteses

Teorema

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

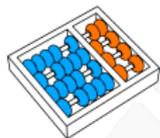


Teorema dos parênteses

Teorema

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

1. (a) Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.

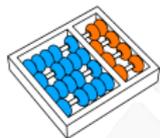


Teorema dos parênteses

Teorema

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

1. (a) Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.
(b) Nesse caso u e v não são descendentes um do outro.

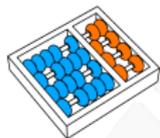


Teorema dos parênteses

Teorema

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

1. (a) Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.
(b) Nesse caso u e v não são descendentes um do outro.

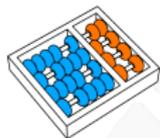


Teorema dos parênteses

Teorema

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

- Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.
 - Nesse caso u e v não são descendentes um do outro.
- O intervalo $[d[u], f[u]]$ está contido em $[d[v], f[v]]$.

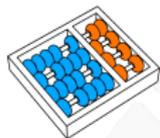


Teorema dos parênteses

Teorema

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

- Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.
 - Nesse caso u e v não são descendentes um do outro.
- O intervalo $[d[u], f[u]]$ está contido em $[d[v], f[v]]$.
 - Nesse caso u é descendente de v .

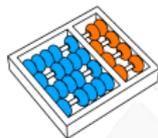


Teorema dos parênteses

Teorema

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

- Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.
 - Nesse caso u e v não são descendentes um do outro.
- O intervalo $[d[u], f[u]]$ está contido em $[d[v], f[v]]$.
 - Nesse caso u é descendente de v .



Teorema dos parênteses

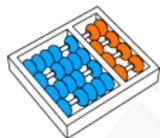
Teorema

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

- (a) Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.

(b) Nesse caso u e v não são descendentes um do outro.
- (a) O intervalo $[d[u], f[u]]$ está contido em $[d[v], f[v]]$.

(b) Nesse caso u é descendente de v .
- (a) O intervalo $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.



Teorema dos parênteses

Teorema

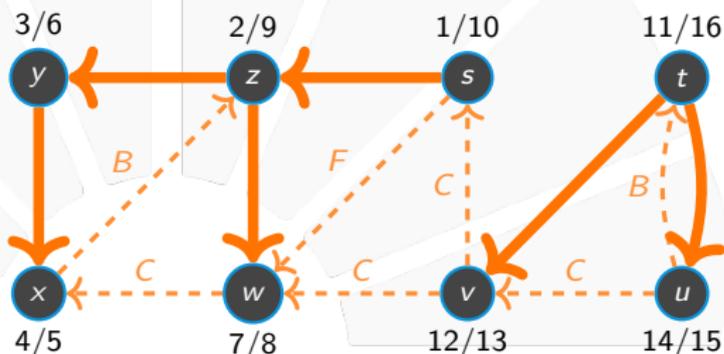
Se u e v são vértices de uma árvore de busca em profundidade, então ocorre exatamente um entre os três casos abaixo:

- Os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos.
 - Nesse caso u e v não são descendentes um do outro.
- O intervalo $[d[u], f[u]]$ está contido em $[d[v], f[v]]$.
 - Nesse caso u é descendente de v .
- O intervalo $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.
 - Nesse caso v é descendente de u .

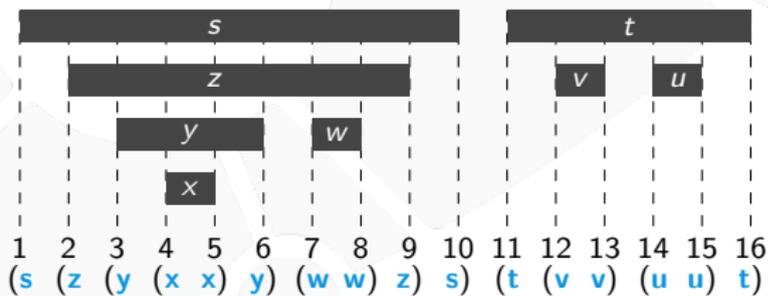


Exemplo

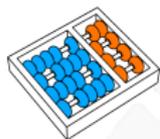
Floresta de busca



Estrutura de parênteses

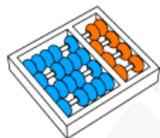


Busca em profundidade



Demonstração do teorema

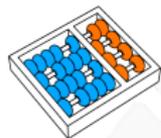
Demonstração do teorema dos parênteses:



Demonstração do teorema

Demonstração do teorema dos parênteses:

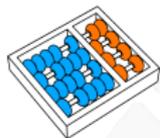
- ▶ Podemos supor que $d[u] < d[v]$.



Demonstração do teorema

Demonstração do teorema dos parênteses:

- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

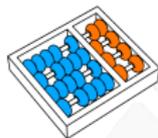


Demonstração do teorema

Demonstração do teorema dos parênteses:

- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:



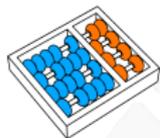
Demonstração do teorema

Demonstração do teorema dos parênteses:

- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.



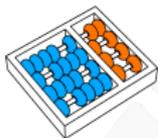
Demonstração do teorema

Demonstração do teorema dos parênteses:

- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.
- ▶ Logo, a chamada recursiva para v termina antes da de u .



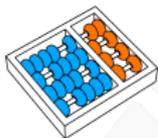
Demonstração do teorema

Demonstração do teorema dos parênteses:

- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.
- ▶ Logo, a chamada recursiva para v termina antes da de u .
- ▶ Portanto, v é descendente de u e $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.



Demonstração do teorema

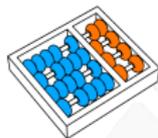
Demonstração do teorema dos parênteses:

- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.
- ▶ Logo, a chamada recursiva para v termina antes da de u .
- ▶ Portanto, v é descendente de u e $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.

Caso 2: Se $f[u] < d[v]$:



Demonstração do teorema

Demonstração do teorema dos parênteses:

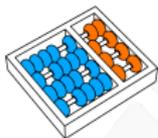
- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.
- ▶ Logo, a chamada recursiva para v termina antes da de u .
- ▶ Portanto, v é descendente de u e $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.

Caso 2: Se $f[u] < d[v]$:

- ▶ Então, u foi finalizado enquanto v era branco.



Demonstração do teorema

Demonstração do teorema dos parênteses:

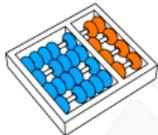
- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.
- ▶ Logo, a chamada recursiva para v termina antes da de u .
- ▶ Portanto, v é descendente de u e $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.

Caso 2: Se $f[u] < d[v]$:

- ▶ Então, u foi finalizado enquanto v era branco.
- ▶ Logo, a chamada de u termina antes que a de v comece.



Demonstração do teorema

Demonstração do teorema dos parênteses:

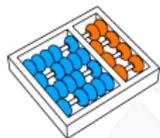
- ▶ Podemos supor que $d[u] < d[v]$.
- ▶ Analisamos dois casos:

Caso 1: Se $d[v] < f[u]$:

- ▶ Então, v foi descoberto enquanto u era cinza.
- ▶ Logo, a chamada recursiva para v termina antes da de u .
- ▶ Portanto, v é descendente de u e $[d[v], f[v]]$ está contido em $[d[u], f[u]]$.

Caso 2: Se $f[u] < d[v]$:

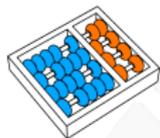
- ▶ Então, u foi finalizado enquanto v era branco.
- ▶ Logo, a chamada de u termina antes que a de v comece.
- ▶ Portanto, u e v não são descendentes um do outro e $[d[v], f[v]]$ e $[d[u], f[u]]$ são disjuntos.



Teorema do caminho branco

Teorema

Considere dois vértices u e v . São equivalentes:

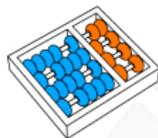


Teorema do caminho branco

Teorema

Considere dois vértices u e v . São equivalentes:

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



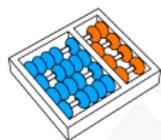
Teorema do caminho branco

Teorema

Considere dois vértices u e v . São equivalentes:

- (1) v é descendente de u na floresta de busca.
- (2) Quando u foi descoberto, existia um caminho de u a v formado apenas por vértices brancos.

Busca em profundidade



Demonstração do teorema do caminho branco

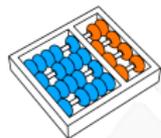
▶ (1) \Rightarrow (2)



Demonstração do teorema do caminho branco

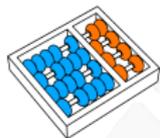
▶ (1) \Rightarrow (2)

▶ Suponha que v é um descendente de u .



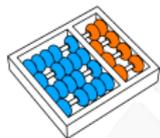
Demonstração do teorema do caminho branco

- ▶ (1) \Rightarrow (2)
 - ▶ Suponha que v é um descendente de u .
 - ▶ Seja z um vértice qualquer no caminho de u até v na floresta.



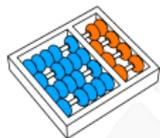
Demonstração do teorema do caminho branco

- ▶ (1) \Rightarrow (2)
 - ▶ Suponha que v é um descendente de u .
 - ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
 - ▶ Então, z é descendente de u , logo $d[u] < d[z]$.



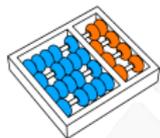
Demonstração do teorema do caminho branco

- ▶ (1) \Rightarrow (2)
 - ▶ Suponha que v é um descendente de u .
 - ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
 - ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
 - ▶ Portanto, z era branco no instante $d[u]$,



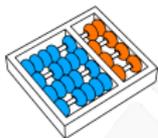
Demonstração do teorema do caminho branco

- ▶ (1) \Rightarrow (2)
 - ▶ Suponha que v é um descendente de u .
 - ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
 - ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
 - ▶ Portanto, z era branco no instante $d[u]$,
 - ▶ assim como todos os vértices no caminho.



Demonstração do teorema do caminho branco

- ▶ (1) \Rightarrow (2)
 - ▶ Suponha que v é um descendente de u .
 - ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
 - ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
 - ▶ Portanto, z era branco no instante $d[u]$,
 - ▶ assim como todos os vértices no caminho.
- ▶ (2) \Rightarrow (1)



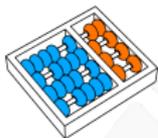
Demonstração do teorema do caminho branco

▶ (1) \Rightarrow (2)

- ▶ Suponha que v é um descendente de u .
- ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
- ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
- ▶ Portanto, z era branco no instante $d[u]$,
- ▶ assim como todos os vértices no caminho.

▶ (2) \Rightarrow (1)

- ▶ Considere um caminho branco de u a v no instante $d[u]$.



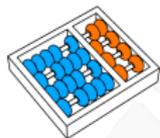
Demonstração do teorema do caminho branco

▶ (1) \Rightarrow (2)

- ▶ Suponha que v é um descendente de u .
- ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
- ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
- ▶ Portanto, z era branco no instante $d[u]$,
- ▶ assim como todos os vértices no caminho.

▶ (2) \Rightarrow (1)

- ▶ Considere um caminho branco de u a v no instante $d[u]$.
- ▶ Suponha que há vértices no caminho não descendentes de u .



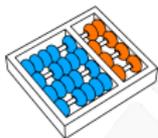
Demonstração do teorema do caminho branco

▶ (1) \Rightarrow (2)

- ▶ Suponha que v é um descendente de u .
- ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
- ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
- ▶ Portanto, z era branco no instante $d[u]$,
- ▶ assim como todos os vértices no caminho.

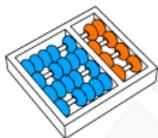
▶ (2) \Rightarrow (1)

- ▶ Considere um caminho branco de u a v no instante $d[u]$.
- ▶ Suponha que há vértices no caminho não descendentes de u .
- ▶ Sejam z o primeiro vértice não descendente de u no caminho e w seu antecessor.



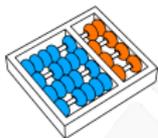
Demonstração do teorema do caminho branco

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



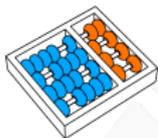
Demonstração do teorema do caminho branco

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



Demonstração do teorema do caminho branco

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



Demonstração do teorema do caminho branco

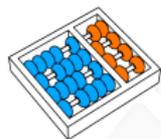
▶ (1) \Rightarrow (2)

- ▶ Suponha que v é um descendente de u .
- ▶ Seja z um vértice qualquer no caminho de u até v na floresta.
- ▶ Então, z é descendente de u , logo $d[u] < d[z]$.
- ▶ Portanto, z era branco no instante $d[u]$,
- ▶ assim como todos os vértices no caminho.

▶ (2) \Rightarrow (1)

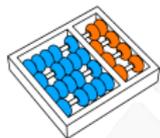
- ▶ Considere um caminho branco de u a v no instante $d[u]$.
- ▶ Suponha que há vértices no caminho não descendentes de u .
- ▶ Sejam z o primeiro vértice não descendente de u no caminho e w seu antecessor.
- ▶ Como w é descendente de u , temos $f[w] \leq f[u]$.
- ▶ Como z não é descendente de u , temos $f[u] < d[z]$.
- ▶ Logo, z era um vizinho branco de w no instante $f[w]$.
- ▶ Isso é uma contradição, então todo vértice do caminho branco é descendente de u na floresta de busca.

Busca em profundidade



Classificação de arestas

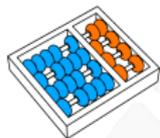
Dada a floresta de busca, podemos classificar arestas do grafo



Classificação de arestas

Dada a floresta de busca, podemos classificar arestas do grafo

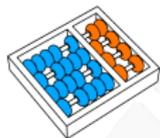
- ▶ **ARESTAS DE ÁRVORE** (*tree edges*) são arestas da floresta de busca em profundidade



Classificação de arestas

Dada a floresta de busca, podemos classificar arestas do grafo

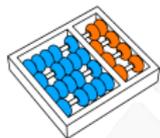
- ▶ **ARESTAS DE ÁRVORE** (*tree edges*) são arestas da floresta de busca em profundidade
- ▶ **ARESTAS DE RETORNO** (*backward edges*) ligam um vértice a um ancestral



Classificação de arestas

Dada a floresta de busca, podemos classificar arestas do grafo

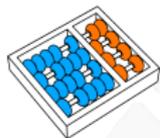
- ▶ **ARESTAS DE ÁRVORE** (*tree edges*) são arestas da floresta de busca em profundidade
- ▶ **ARESTAS DE RETORNO** (*backward edges*) ligam um vértice a um ancestral
- ▶ **ARESTAS DE AVANÇO** (*forward edges*) ligam um vértice a um descendente



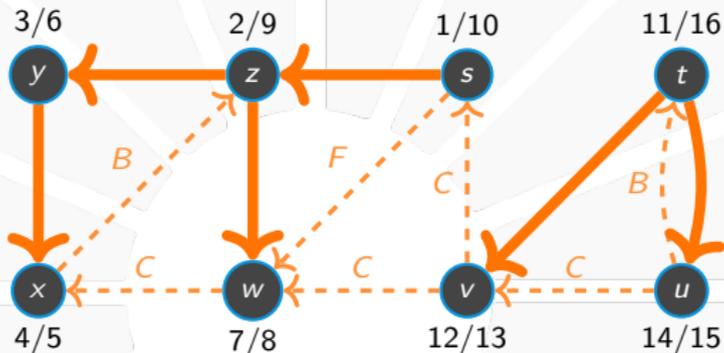
Classificação de arestas

Dada a floresta de busca, podemos classificar arestas do grafo

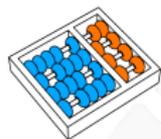
- ▶ **ARESTAS DE ÁRVORE** (*tree edges*) são arestas da floresta de busca em profundidade
- ▶ **ARESTAS DE RETORNO** (*backward edges*) ligam um vértice a um ancestral
- ▶ **ARESTAS DE AVANÇO** (*forward edges*) ligam um vértice a um descendente
- ▶ **ARESTAS DE CRUZAMENTO** (*cross edges*) são todas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



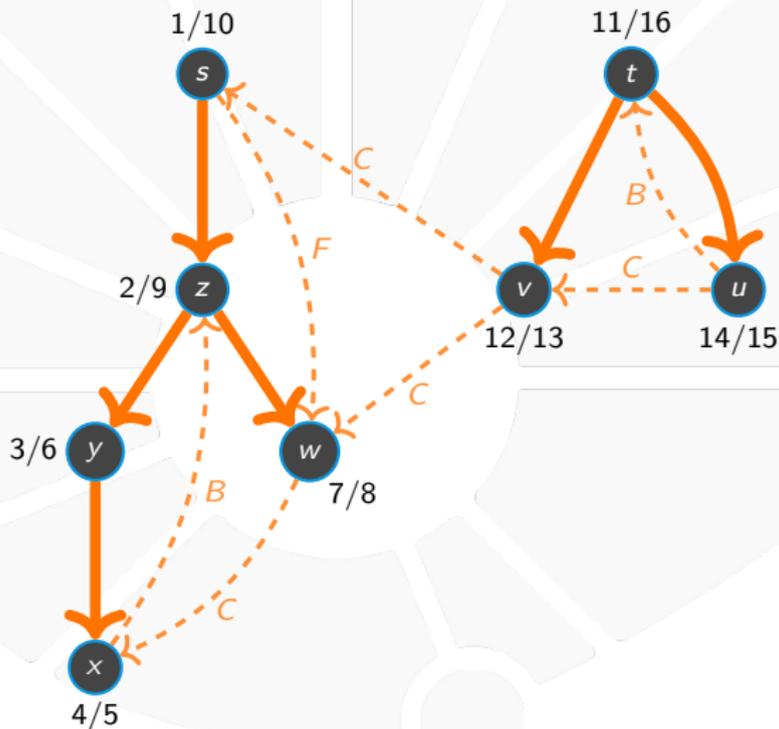
Classificação de arestas



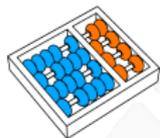
É fácil modificar o algoritmo $\text{DFS}(G)$ para que ele também classifique as arestas de G . (exercício)



Classificação de arestas

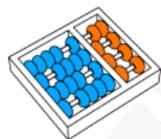


Busca em profundidade



Grafos não direcionados

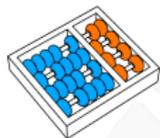
Classificando arestas não direcionadas:



Grafos não direcionados

Classificando arestas não direcionadas:

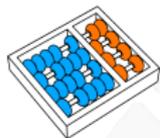
- ▶ Não pode haver arestas de avanço. Por quê?



Grafos não direcionados

Classificando arestas não direcionadas:

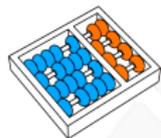
- ▶ Não pode haver arestas de avanço. Por quê?
- ▶ Tampouco arestas de cruzamento. Por quê?



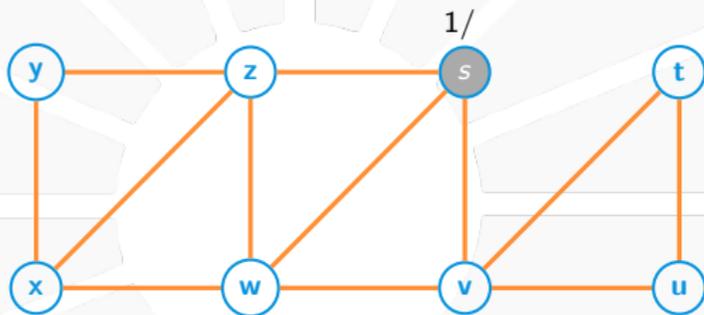
Grafos não direcionados

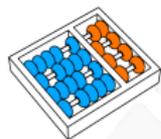
Classificando arestas não direcionadas:

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

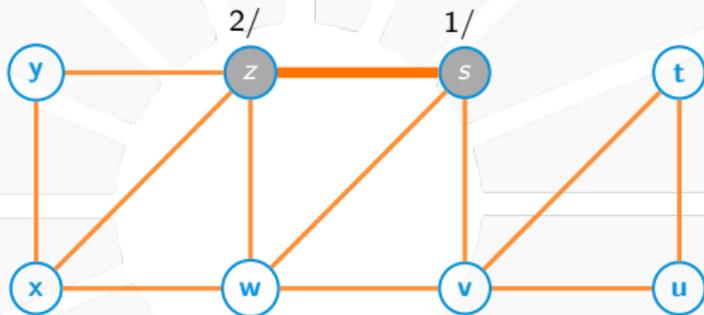


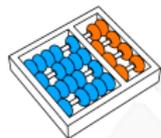
DFS em grafo não direcionado



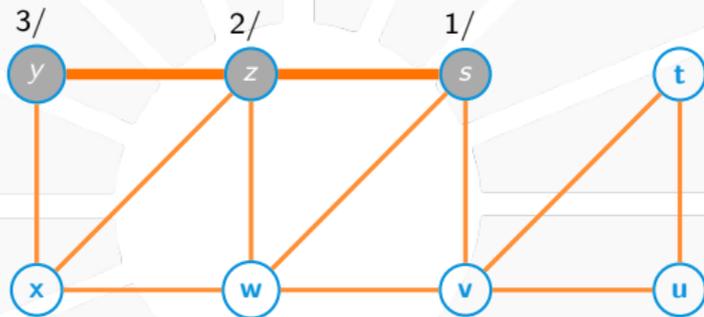


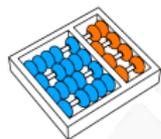
DFS em grafo não direcionado



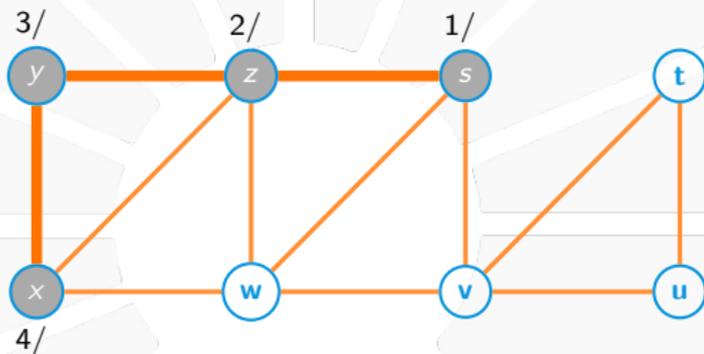


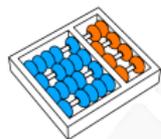
DFS em grafo não direcionado



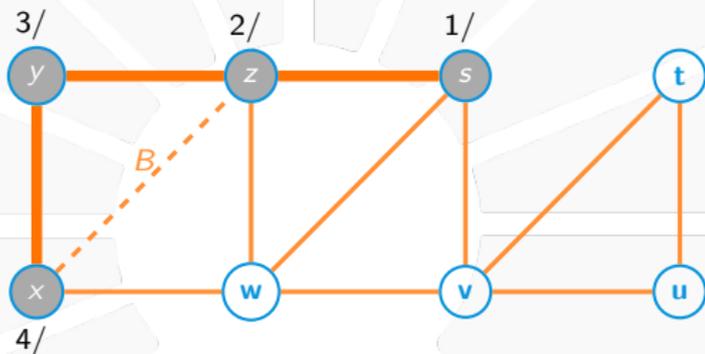


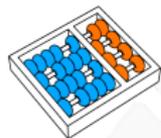
DFS em grafo não direcionado



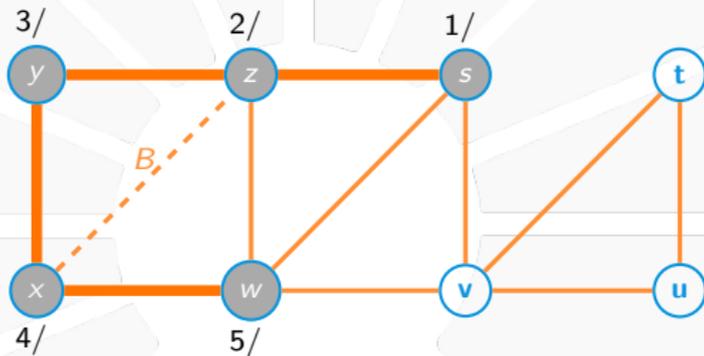


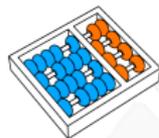
DFS em grafo não direcionado



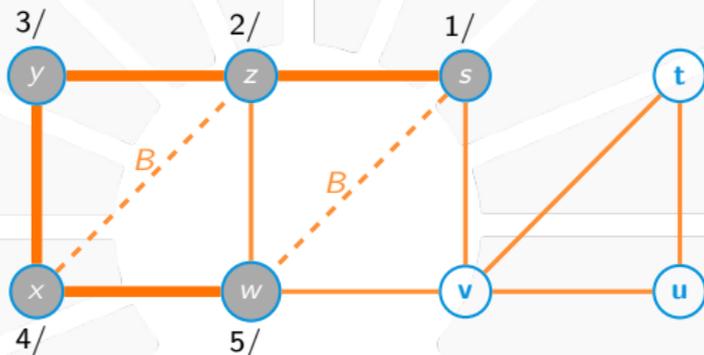


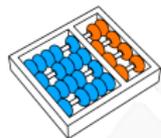
DFS em grafo não direcionado



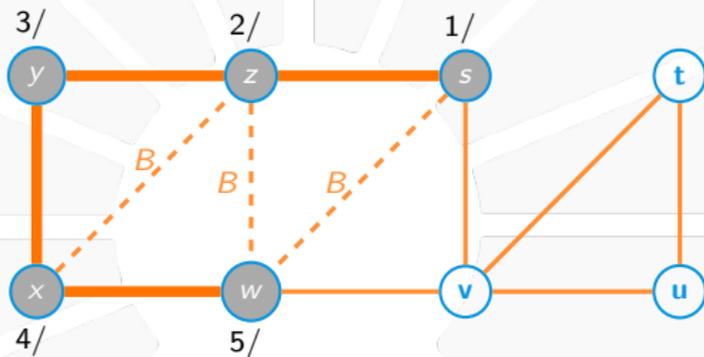


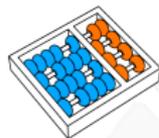
DFS em grafo não direcionado



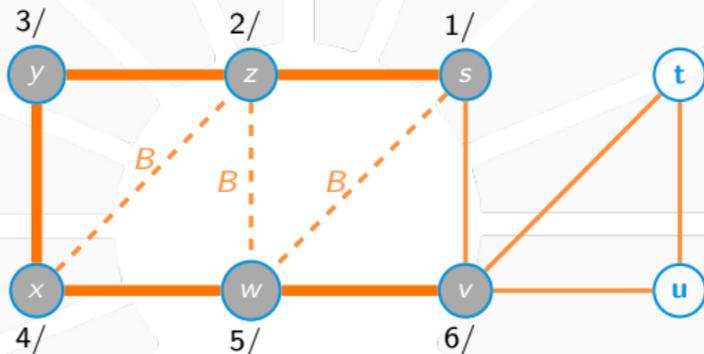


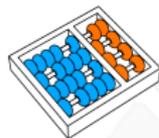
DFS em grafo não direcionado



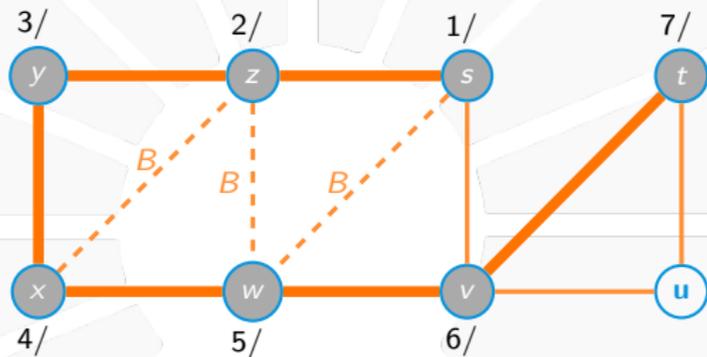


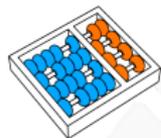
DFS em grafo não direcionado



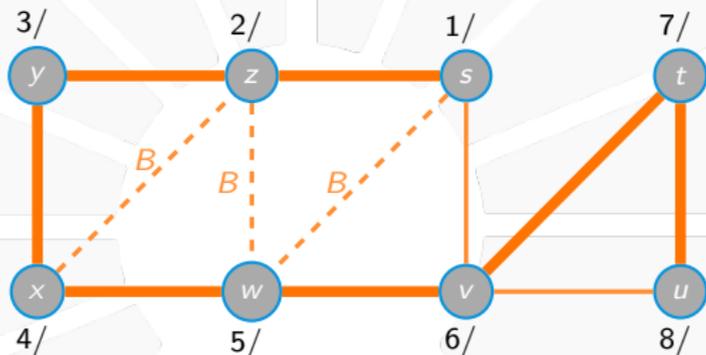


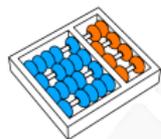
DFS em grafo não direcionado



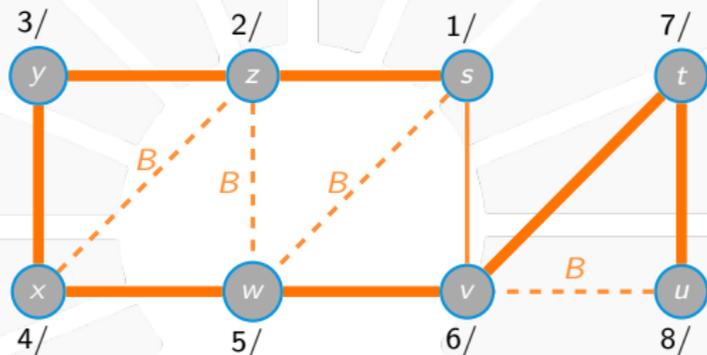


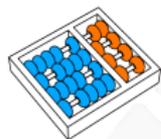
DFS em grafo não direcionado



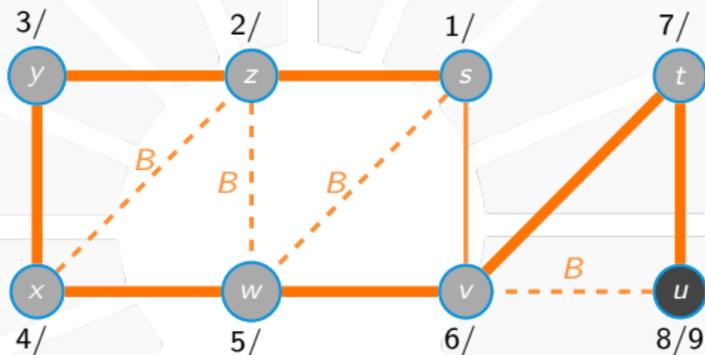


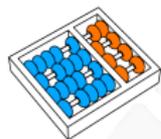
DFS em grafo não direcionado



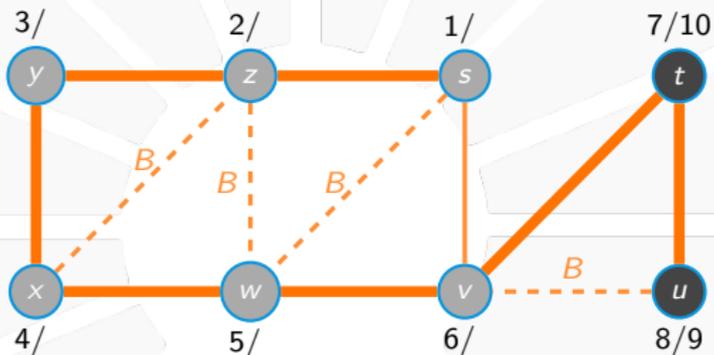


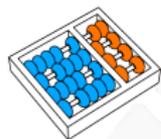
DFS em grafo não direcionado



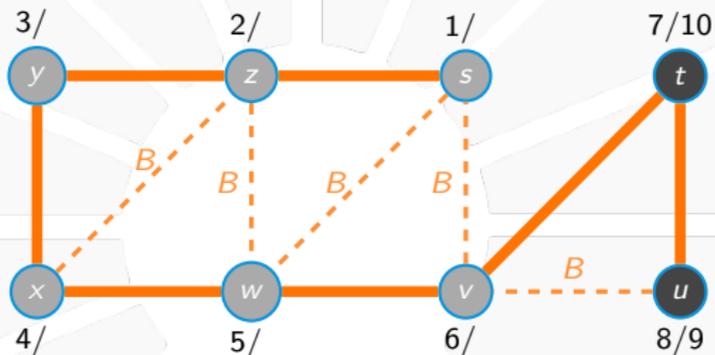


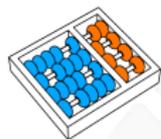
DFS em grafo não direcionado



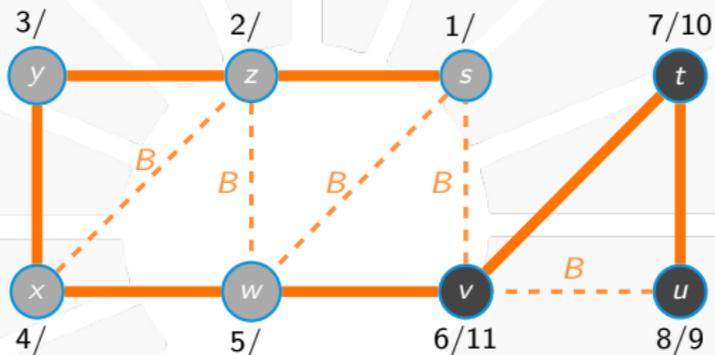


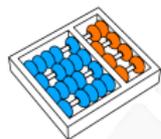
DFS em grafo não direcionado



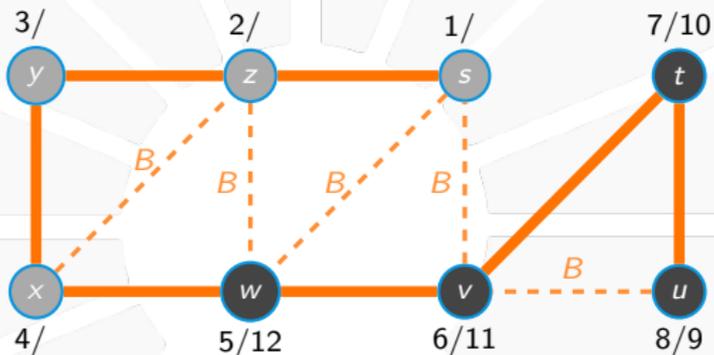


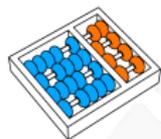
DFS em grafo não direcionado



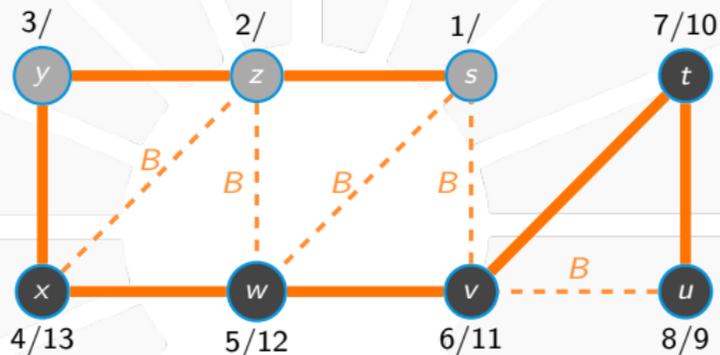


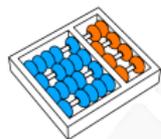
DFS em grafo não direcionado



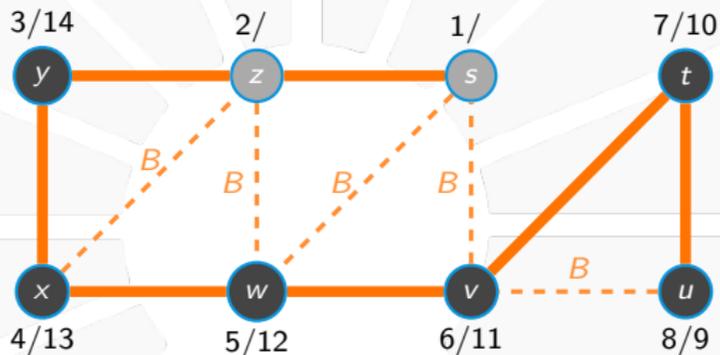


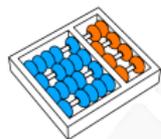
DFS em grafo não direcionado



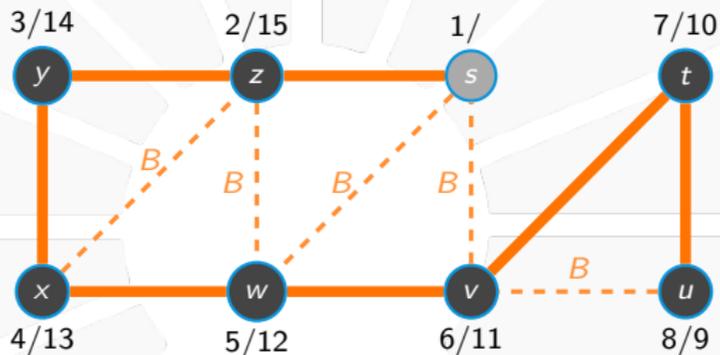


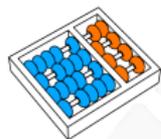
DFS em grafo não direcionado



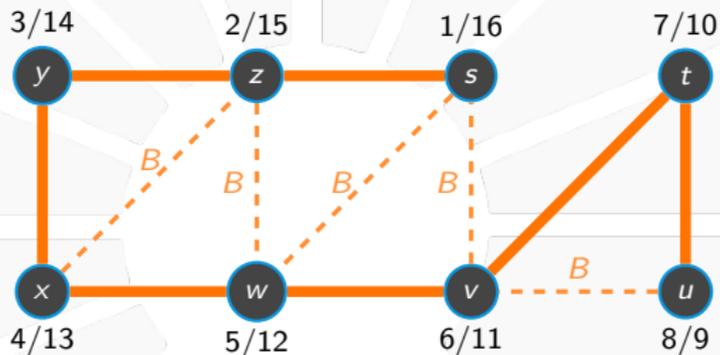


DFS em grafo não direcionado





DFS em grafo não direcionado



BUSCAS EM GRAFOS

MC558 - Projeto e Análise de Algoritmos II

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

01/24

3



UNICAMP

