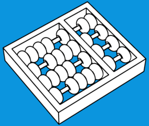
Buscas em grafos

MO417 - Complexidade de Algoritmos I

05/24

4



UNICAMP

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

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

Eliezer Yudkowsky

Noções básicas

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:

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?

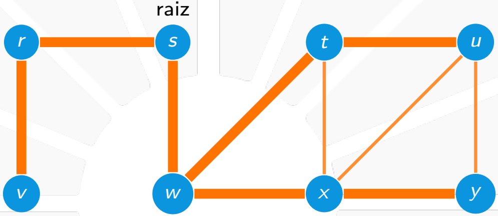
- A enraizamos em um **VÉRTICE DE ORIGEM** $\mathbf{s}$.
- A representamos com um vetor π de pais.
- O pai de um vértice $\mathbf{v}$ é $\pi[\mathbf{v}]$.
- convencionamos que $\pi[\mathbf{s}] = \text{NIL}$

Algumas propriedades:

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

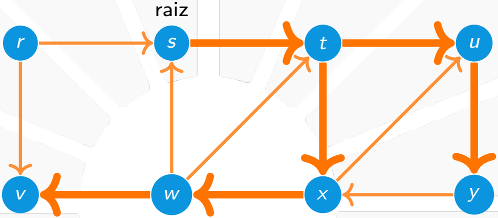
$$\mathbf{s} \to \cdots \to \pi[\pi[\pi[\mathbf{v}]]] \to \pi[\pi[\mathbf{v}]] \to \pi[\mathbf{v}] \to \mathbf{v}$$

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: $\textsc{Print-Path}(G, s, v)$

se $v = s$
  imprima s
senão se $\pi[v] = \text{NIL}$
  imprima não existe caminho de s a v
senão
  $\textsc{Print-Path}(G, s, \pi[v])$
  imprima v

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

Busca em largura

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 $\mathrm{dist}(\mathbf{s}, \mathbf{v})$.
- Se **v** não for alcançável, definimos $\mathrm{dist}(\mathbf{s}, \mathbf{v}) = \infty$.

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.

Descobrindo 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:

- 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

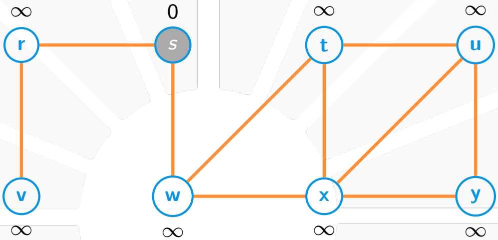
Vamos pintar o grafo durante a busca:

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

Observações:

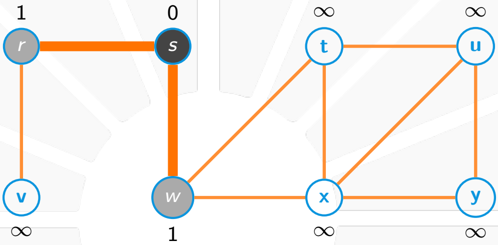
- Não é necessário em uma implementação.
- Facilita o entendimento do algoritmo.

Exemplo de busca em largura



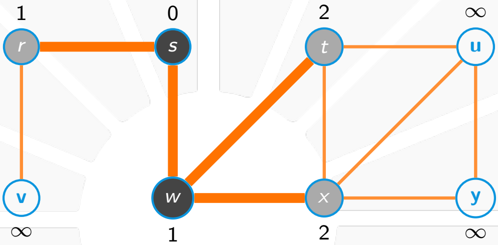
Q	s
distância	0

Exemplo de busca em largura



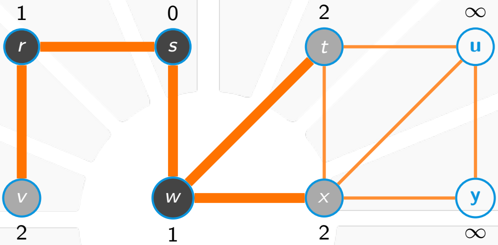
Q	**w**	**r**
distância	1	1

Exemplo de busca em largura



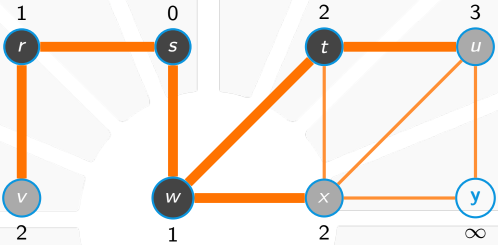
Q	**r**	**t**	**x**
distância	1	2	2

Exemplo de busca em largura



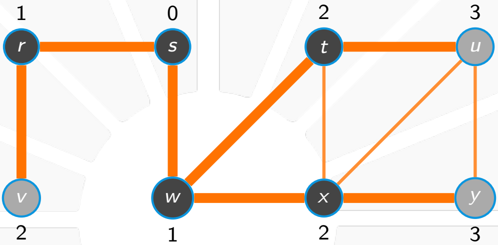
Q	t	x	v
distância	2	2	2

Exemplo de busca em largura



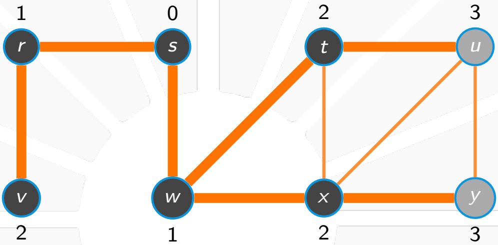
Q	**x**	**v**	**u**
distância	2	2	3

Exemplo de busca em largura



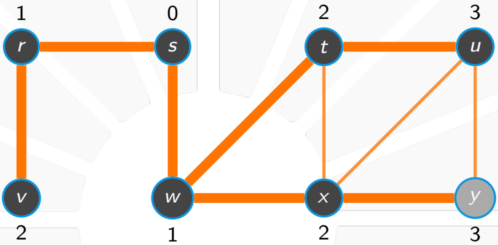
Q	v	u	y
distância	2	3	3

Exemplo de busca em largura



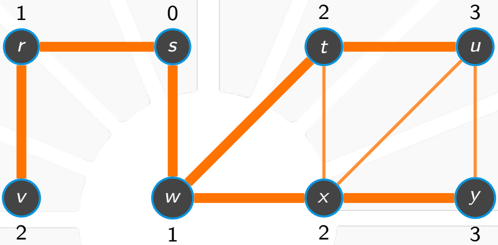
Q	**u**	**y**
distância	3	3

Exemplo de busca em largura



Q	y
distância	3

Exemplo de busca em largura



Q	$\emptyset$
distância	

Algoritmo BFS

Observações:

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

Algoritmo BFS

Algoritmo 2: BFS(G, s)

para cada $u \in V[G]$
    cor$[u] \leftarrow$ branco, $d[u] \leftarrow \infty$, $\pi[u] \leftarrow$ NIL
cor$[s] \leftarrow$ cinza
$d[s] \leftarrow 0$
$Q \leftarrow \emptyset$
Enqueue(Q, s)
enquanto $Q \neq \emptyset$
    $u \leftarrow$ Dequeue(Q)
    para cada $v \in$ Adj$[u]$
        se cor$[v] =$ branco
            cor$[v] \leftarrow$ cinza
            $d[v] \leftarrow d[u] + 1$
            $\pi[v] \leftarrow u$
            Enqueue(Q, v)
    cor$[u] \leftarrow$ preto

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 $O(E)$.

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* $\mathrm{BFS}(G, s)$*, temos:*

1. π *define uma árvore enraizada em* s*,*
2. $d[v] = \mathrm{dist}(s, v)$ *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:

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 contem 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**:

- 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 $d[\mathbf{u}]$.
- Portanto:
    1. Há um caminho de **s** a **v** em T, passando por **u**, com comprimento $d[\mathbf{v}] = d[\mathbf{u}] + 1$, pois $\pi[\mathbf{v}] = \mathbf{u}$.
- Com isso, completamos a indução.

Corolário (1)

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

Lema 2

Lema (2)

Suponha que $\langle \mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_r} \rangle$ *seja a disposição da fila* Q *em alguma iteração do algoritmo. Então*

$$d[\mathbf{v_1}] \leq d[\mathbf{v_2}] \leq \cdots \leq d[\mathbf{v_r}] \leq d[\mathbf{v_1}] + 1.$$

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:

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

Inserimos vizinhos de $\mathbf{v_1}$:

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

$$d[\mathbf{v_1}] \le d[\mathbf{v_2}] \le \cdots \le d[\mathbf{v_r}] \le d[\mathbf{v_1}] + 1 \le d[\mathbf{v_2}] + 1.$$

- Portanto:

$$d[\mathbf{v_2}] \le \cdots \le d[\mathbf{v_r}] \le d[\mathbf{v_{r+1}}] \le \cdots \le d[\mathbf{v_{r+t}}] \le d[\mathbf{v_2}] + 1.$$

Demonstração do teorema

Teorema

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

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

Demonstração:

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

Demonstração do teorema

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

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

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

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

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 $d[\mathbf{v}] \leq d[\mathbf{u}] < k$.
  - Mas o Corolário 1 implica que $k = \text{dist}(\mathbf{s}, \mathbf{v}) \leq d[\mathbf{v}]$.
  - 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, $d[\mathbf{v}] = d[\mathbf{u}] + 1 = k$.
  - Se **v** for cinza:
    - **v** já estava na fila.
    - Pelo Lema 2 temos que $d[\mathbf{v}] \leq d[\mathbf{u}] + 1 = k$
    - Pelo Corolário 1 temos que $k \leq d[\mathbf{v}]$, portanto $d[\mathbf{v}] = k$.
  - Em qualquer caso, concluímos a indução.

Busca em profundidade

Busca em profundidade

Buscando os vértices alcançáveis em **PROFUNDIDADE**:

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

É a estratégia usada por vários algoritmos:

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

Construindo uma árvore de busca

Ideia do algoritmo:

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

Alternativa

Ideia alternativa:

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

Observações:

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

Floresta de busca

Visitando todos os vértices:

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

Representando uma floresta:

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

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

Cores dos vértices

De novo, vamos pintar o grafo durante a busca:

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

Observações:

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

Tempo de descoberta e finalização

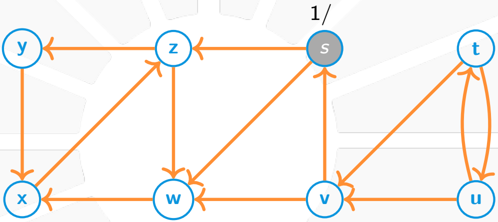
Ademais, vamos associar rótulos aos vértices:

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

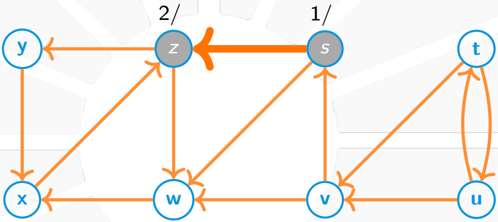
Observações:

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

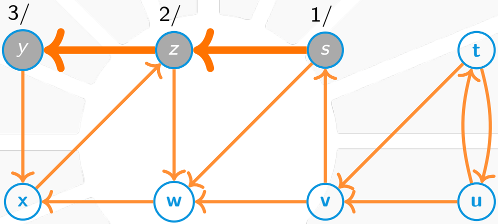
Exemplo de busca em profundidade



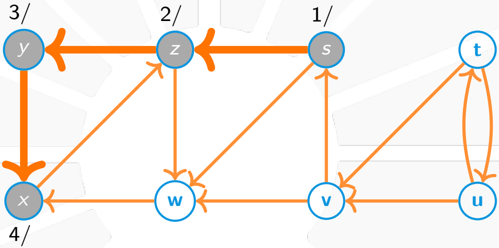
Exemplo de busca em profundidade



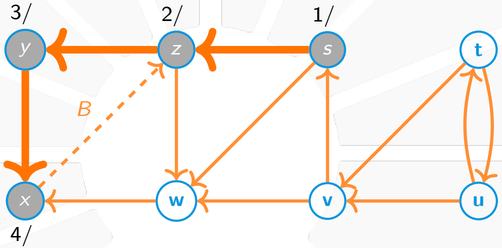
Exemplo de busca em profundidade



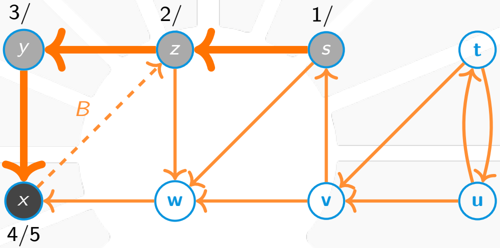
Exemplo de busca em profundidade



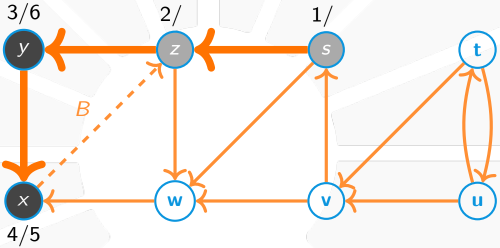
Exemplo de busca em profundidade



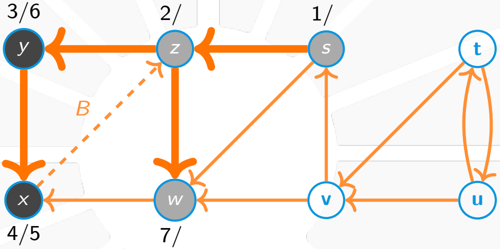
Exemplo de busca em profundidade



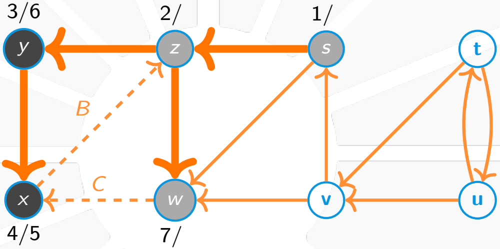
Exemplo de busca em profundidade



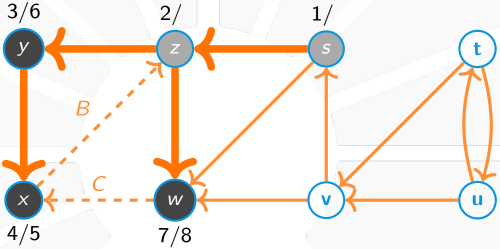
Exemplo de busca em profundidade



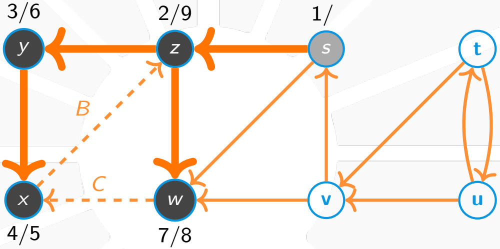
Exemplo de busca em profundidade



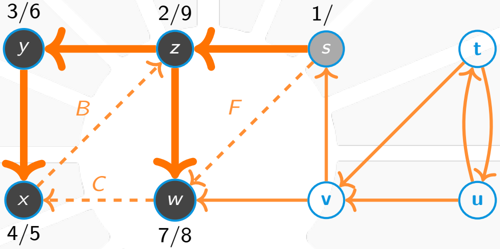
Exemplo de busca em profundidade



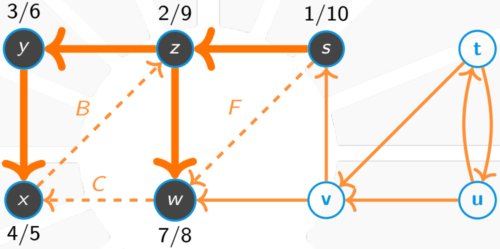
Exemplo de busca em profundidade



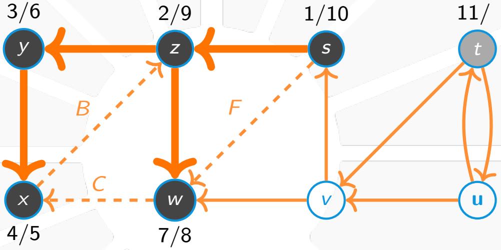
Exemplo de busca em profundidade



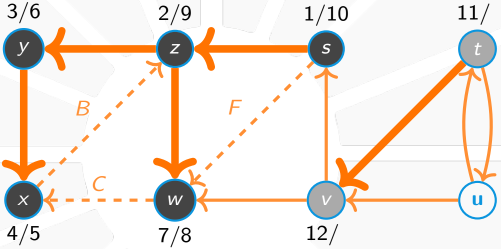
Exemplo de busca em profundidade



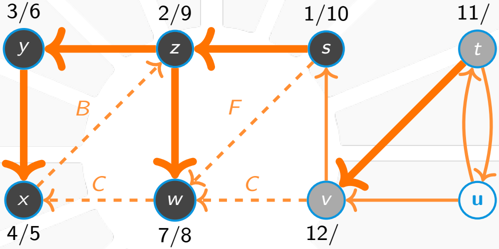
Exemplo de busca em profundidade



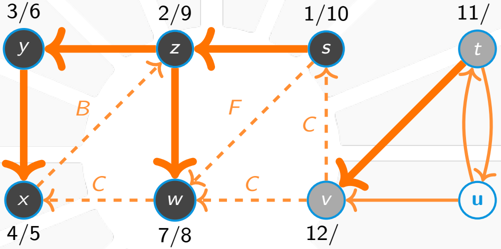
Exemplo de busca em profundidade



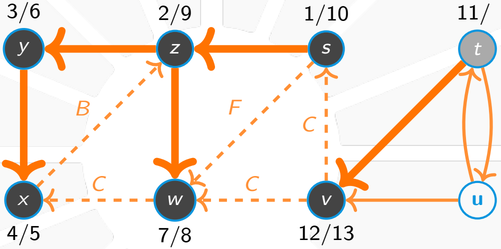
Exemplo de busca em profundidade



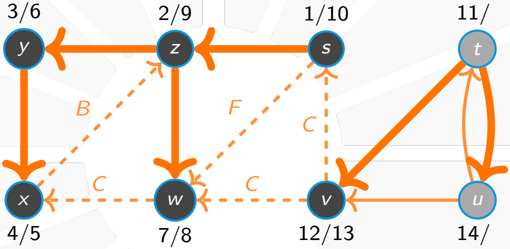
Exemplo de busca em profundidade



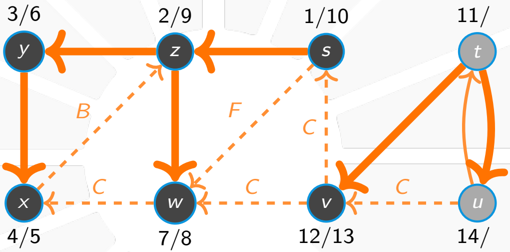
Exemplo de busca em profundidade



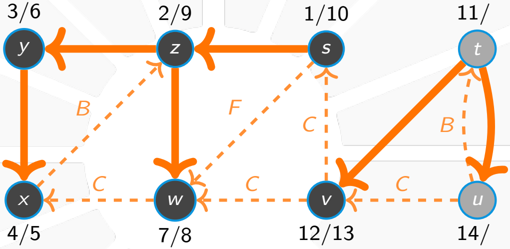
Exemplo de busca em profundidade



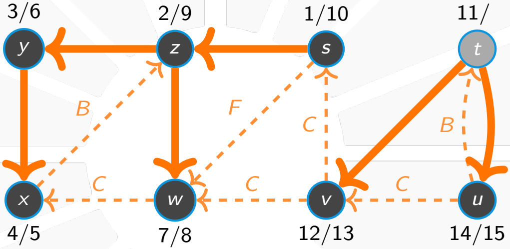
Exemplo de busca em profundidade



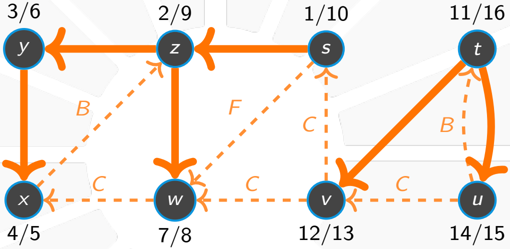
Exemplo de busca em profundidade



Exemplo de busca em profundidade



Exemplo de busca em profundidade



Rótulos versus cores

Observe que para todo vértice **v**:

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

Algoritmo DFS

Algoritmo 3: $\mathrm{DFS}(G)$

```
para cada u ∈ V[G]
    cor[u] ← branco
    π[u] ← NIL
tempo ← 0
para cada u ∈ V[G]
    se cor[u] = branco
        DFS-VISIT(u)
```

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

Algoritmo DFS-VISIT

Algoritmo 4: DFS-VISIT(u)

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

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

Análise de complexidade

Tempo do algoritmo principal DFS:

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

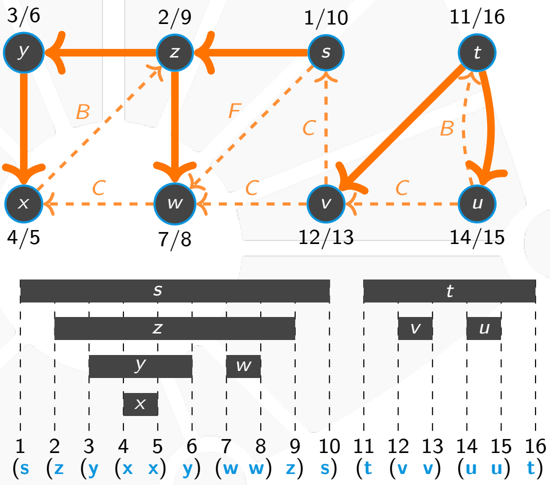
*Se **u** e **v** são vértices de uma árvore de busca em profundidade, então ocorre exatamente um entre os três casos abaixo:*

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

Exemplo

Floresta de busca

Estrutura de parênteses



Demonstração do teorema

Demonstração do teorema dos parênteses:

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

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

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

Teorema do caminho branco

Teorema

Considere dois vértices $\boldsymbol{u}$ *e* $\boldsymbol{v}$*. São equivalentes:*

(1) $\boldsymbol{v}$ *é descendente de* $\boldsymbol{u}$ *na floresta de busca.*

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

Demonstração do teorema do caminho branco

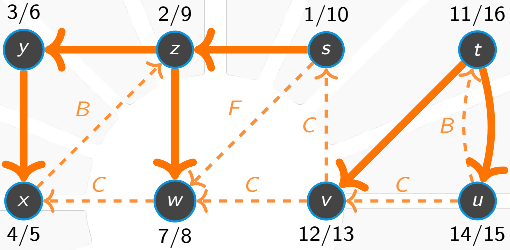
- (1) $\Rightarrow$ (2)
  - Suponha que **v** é um descendente de **u**.
  - Seja **z** um vértice qualquer no caminho de **u** até **v** na floresta.
  - Então, **z** é descendente de **u**, logo $d[\mathbf{u}] < d[\mathbf{z}]$.
  - Portanto, **z** era branco no instante $d[\mathbf{u}]$,
  - assim como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

Classificação de arestas

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

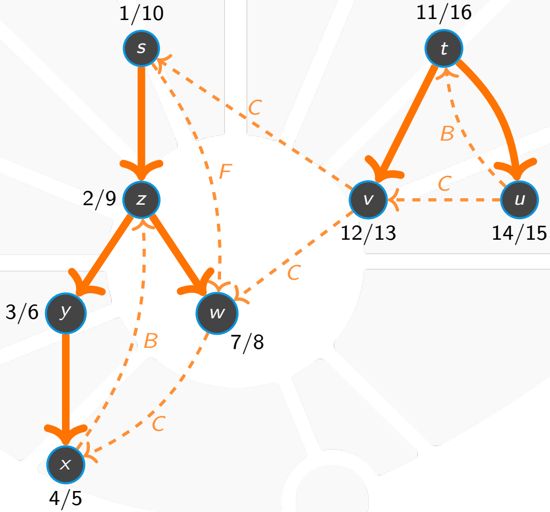
- **ARESTAS DE ÁRVORE** (*tree edges*) são arestas da floresta de busca em profundidade.
- **ARESTAS DE RETORNO** (*backward edges*) ligam um vértice a um ancestral.
- **ARESTAS DE AVANÇO** (*forward edges*) ligam um vértice a um descendente.
- **ARESTAS DE CRUZAMENTO** (*cross edges*) são todas as outras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

Classificação de arestas



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

Classificação de arestas

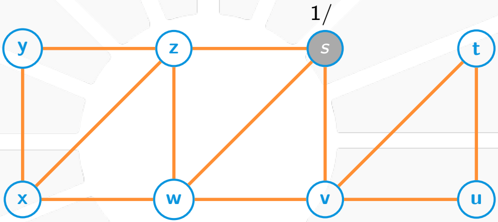


Grafos não direcionados

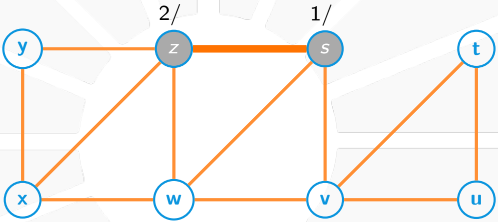
Classificando arestas não direcionadas:

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

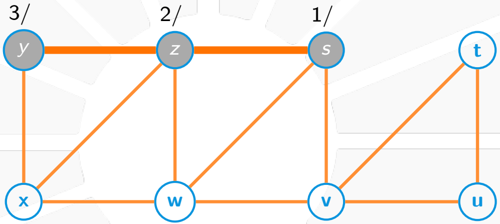
DFS em grafo não direcionado



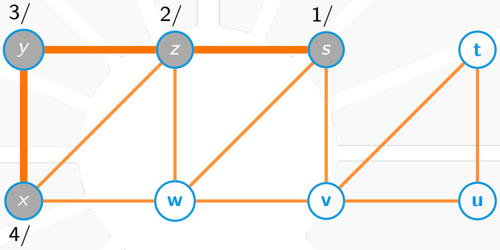
DFS em grafo não direcionado



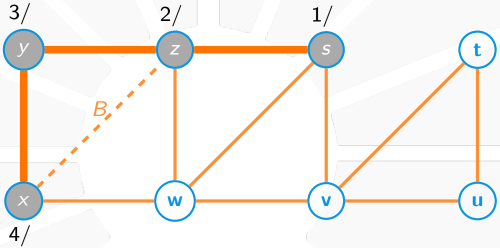
DFS em grafo não direcionado



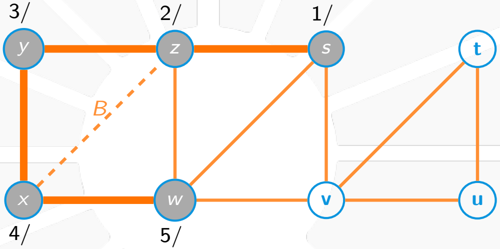
DFS em grafo não direcionado



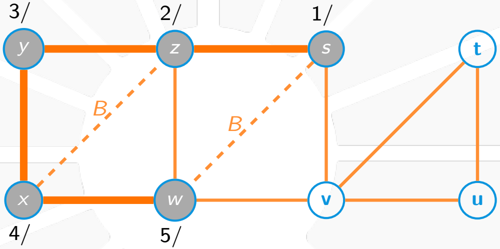
DFS em grafo não direcionado



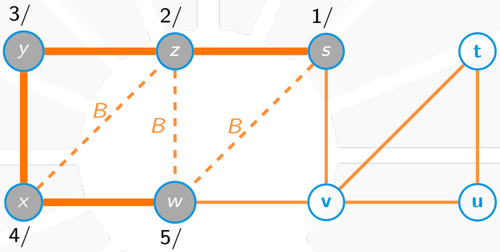
DFS em grafo não direcionado



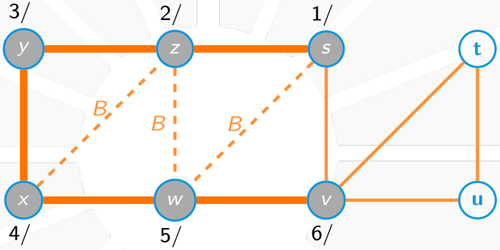
DFS em grafo não direcionado



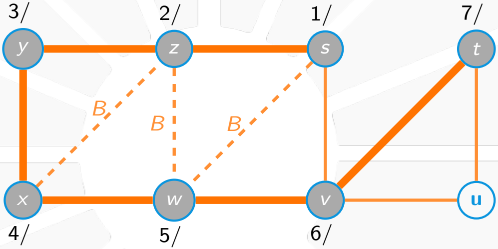
DFS em grafo não direcionado



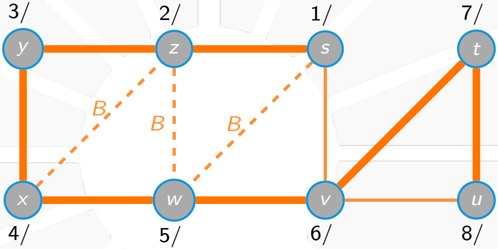
DFS em grafo não direcionado



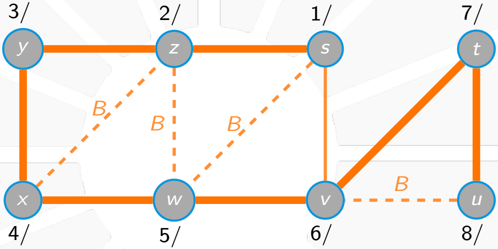
DFS em grafo não direcionado



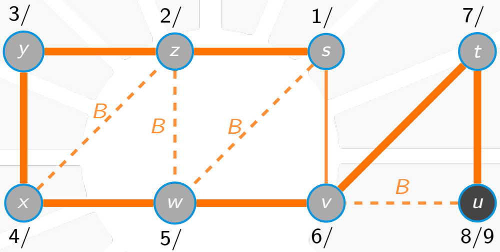
DFS em grafo não direcionado



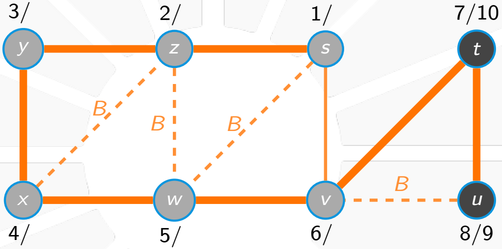
DFS em grafo não direcionado



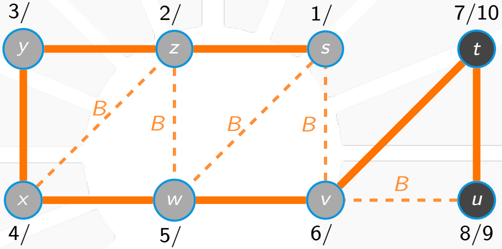
DFS em grafo não direcionado



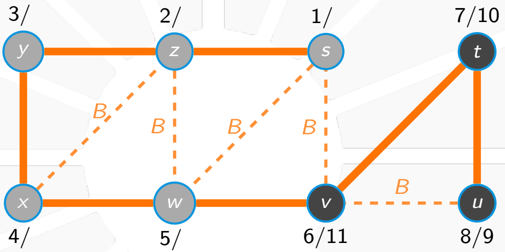
DFS em grafo não direcionado



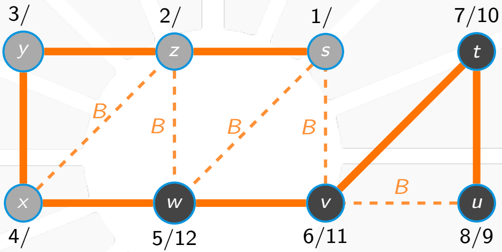
DFS em grafo não direcionado



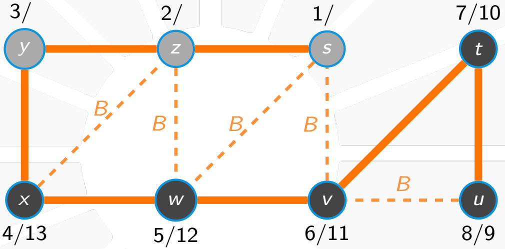
DFS em grafo não direcionado



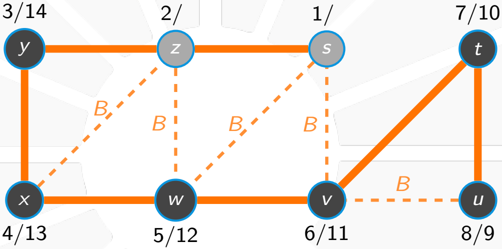
DFS em grafo não direcionado



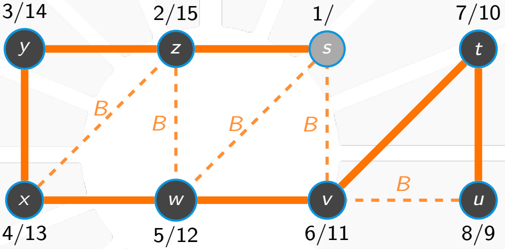
DFS em grafo não direcionado



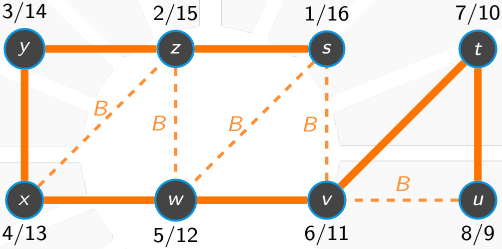
DFS em grafo não direcionado



DFS em grafo não direcionado



DFS em grafo não direcionado

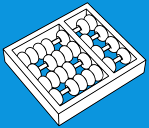


Buscas em grafos

MO417 - Complexidade de Algoritmos I

05/24

4



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

UNICAMP