Aula 23: Grafos: algoritmos elementares (I)

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

Plano

Introdução

Definições

Representação

Busca em largura

Busca em profundidade

Referência: Cormen, cap 23.

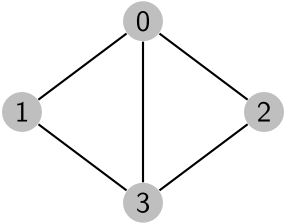
Motivação

- Grafos: modelagem de dados e relacionamentos
- Aplicações em todos os domínios da computação
- Problemas clássicos:
  - ordenação topológica
  - componentes conexos
  - árvore geradora
  - caminhos de custo mínimo
  - fluxo máximo

Grafo não dirigido

Definição (Grafo não dirigido, vértice, aresta)

Um *grafo não dirigido* G é um par (V, E), onde V é o conjunto dos *vértices*, e $E \subseteq V \times V$, é uma relação binária sobre V, é o conjunto das *arestas*[1].



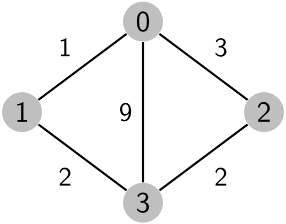
$$\begin{array}{rcl} V & = & 0, 1, 2, 3, \\ E & = & \{(0,1), (0,2), (0,3), (1,3), (2,3)\} \end{array}$$

[1] Existe uma aresta e entre v e v' sse $(v, v') \in E$ ou $(v', v) \in E$.

Grafo não dirigido, com pesos

Definição (Grafo não dirigido com pesos)

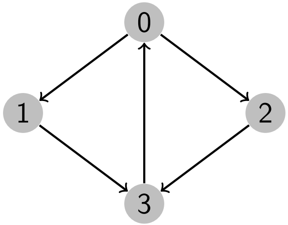
Um *grafo não dirigido com pesos* G é um par (V, E, w), onde (V, E) é um grafo não dirigido e $w : E \to \mathbb{R}$ é o *peso* das arestas.



$$w = \{(0,1) \mapsto 1, (0,2) \mapsto 3, (0,3) \mapsto 9, (1,3) \mapsto 2, (2,3) \mapsto 2\}$$

Definição (Grafo, vértice, seta)

Um *grafo dirigido* G é um par (V, E), onde V é o conjunto dos *vértices*, e $E \subseteq V \times V$, é uma relação binária sobre V, é o conjunto das *setas*[2].



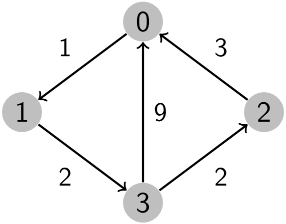
$$
\begin{aligned}
V &= 0, 1, 2, 3, \\
E &= \{(0, 1), (0, 2), (3, 0), (1, 3), (2, 3)\}
\end{aligned}
$$

[2] Existe uma seta e de v para v' sse $(v, v') \in E$.

Grafo dirigido, com pesos

Definição (Grafo dirigido com pesos, vértice, seta)

Um *grafo dirigido com pesos* G é uma tripla (V, E, w), onde (V, E) é um grafo dirigdo e $w : V \to \mathbb{R}$ é o peso de cada seta.

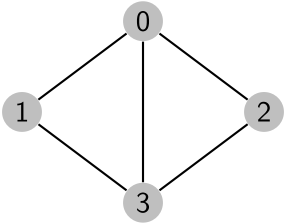


$$w = \{(0,1) \mapsto 1, (0,2) \mapsto 3, (3,0) \mapsto 9, (1,3) \mapsto 2, (2,3) \mapsto 2\}$$

Representação computacional

1. lista de adjacência (matriz esparsa) $\Theta(V + E)$
   (listas encadeadas)
2. matriz de adjacência (matriz densa, ou pequena suficiente)
   $\Theta(V^2)$

Grafo não dirigido sem pesos

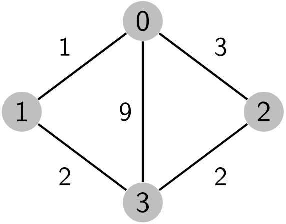


listas de adjacência

$$0 \mapsto \langle 1, 2, 3 \rangle; 1 \mapsto \langle 0, 3 \rangle; 2 \mapsto \langle 0, 3 \rangle, 3 \mapsto \langle 0, 1, 2 \rangle$$

matriz de adjacência $\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$

Grafo não dirigido com pesos

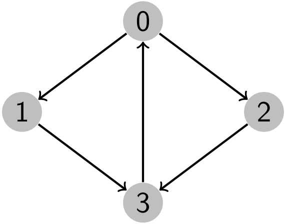


listas de adjacência

$$0 \mapsto \langle 1, 2, 3\rangle; 1 \mapsto \langle 0, 3\rangle; 2 \mapsto \langle 0, 3\rangle, 3 \mapsto \langle 0, 1, 2\rangle$$

matriz de adjacência $\begin{pmatrix} 0 & 1 & 3 & 9 \\ 1 & 0 & \infty & 2 \\ 3 & \infty & 0 & 2 \\ 9 & 2 & 2 & 0 \end{pmatrix}$

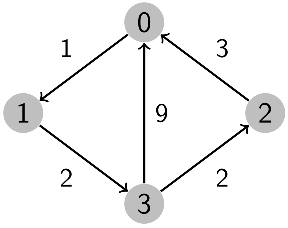
Grafo dirigido



listas de adjacência $0 \mapsto \langle 1, 2 \rangle; 1 \mapsto \langle 3 \rangle; 2 \mapsto \langle 3 \rangle, 3 \mapsto \langle 0 \rangle$

matriz de adjacência $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$

Grafo dirigido com pesos



listas de adjacência $\begin{array}{ll} 0 \mapsto \langle (1,1) \rangle; & 1 \mapsto \langle (2,3) \rangle; \\ 2 \mapsto \langle (1,3) \rangle, & 3 \mapsto \langle (0,9),(2,2) \rangle \end{array}$

matriz de adjacência $\begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & \infty & 2 \\ 3 & \infty & 0 & \infty \\ 9 & \infty & 2 & 0 \end{pmatrix}$

PPgSC

Exercícios

- A matriz de adjacência de um grafo não dirigido é igual à sua transposta. A diagonal não carrega informação. Assim, mais da metade da representação é redundante ou inútil. Para reduzir o tamanho da representação, pode se utilizar um arranjo simples para armazenar apenas as entradas abaixo da diagonal (ou apenas as entradas acima da diagonal).
- Ilustrando esta ideia com o nosso exemplo de grafo simple, a matriz de adjacência pode ser compactada no arranjo seguinte:

1	1	0	1	1	1

  A mesma coisa pode ser feita para grafos com pesos:

1	3	∞	9	2	2

- Dado dos vértices u e v, tal que $u < v$, qual a posição correspondente no arranjo de adjacência?

Exercícios

1. O grau de saída (de entrada) um vértice é o número de arestas que tem este vértice como fonte (destinho).
   - Na representação com listas de adjacência, como calcular o grau de saída e de entrada de um vértice? Qual a complexidade destas operações?
   - E com matrizes de adjacência?
2. Considere o problema de inverter as arestas de um grafo dirigido.
   - Como fazer isto com a representação por listas de adjacência? Qual a complexidade?
   - E com matrizes de adjacência?
   - Relacione este problema com o de calcular a matriz transposta de uma matriz.
3. Dado um grafo $G = (V, E)$, o grafo G^2 é tal que $G^2 = (V, E')$, onde $E = \{(u, w) \mid \exists v \cdot (u, v) \in E \wedge (v, w) \in E\}$.
   - Como calcular G^2 com a representação por listas de adjacência? Qual a complexidade?
   - E com matrizes de adjacência?

Busca em largura

- algoritmo elementar para visitar os vértices
- base para outros algoritmos importantes
  - algoritmo de Dijkstra
  - algoritmo de Prim
- entrada: $G = (V, E)$ e $s \in V$
  - dirigido ou não
- resultado:
  - distância (menor número de arestas) de s para cada $v \in V$
  - árvore "em largura primeiro" de raiz s dos vértices alcançáveis
- largura: processa vértices de distância k de s antes dos vértices de distância $k + 1$ de s.

PPgSC

Princípios

1. Coloração dos vértices:
   - branco: a visitar
   - cinza: sendo visitado
   - preto: visitado
   - branco $\rightarrow$ cinza $\rightarrow$ preto
   - chegada a um vértice: branco $\rightarrow$ cinza
   - cinza: pode possuir vizinhos brancos
   - preto: todos os vértices adjacentes são da cor cinza ou preto.
2. Árvore de processamento:
   - se v foi descoberto a partir de u: u é predecessor de v

Implementação: dados

- $b.color$: cor do vértice v
  - inicialmente: branco
- $v.d$: distância de s a v
  - inicialmente: ∞
- $v.up$: predecessor de v
  - inicialmente: NIL

Implementação: algoritmo

Parte 1: inicialização

$\text{BFS}(G, s)$

    // Inicialização

    for $v \in V - \{s\}$

        $v.color = \text{WHITE}$

        $v.up = \text{NIL}$

        $v.d = \infty$

    $s.color = \text{GRAY}$

    $s.up = \text{NIL}$

    $s.d = 0$

    $\text{ENQUEUE}(Q, s)$

PPgSC

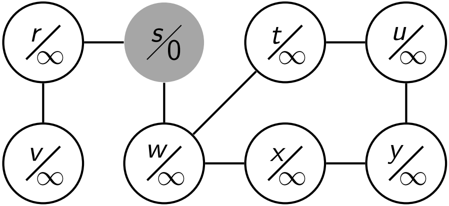
Implementação: algoritmo

Parte 2: processamento

```
// Processamento
while ¬EMPTY(Q)
    u = HEAD(Q)
    for v ∈ u.adj
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.up = u
            ENQUEUE(Q, v)
    DEQUEUE(Q)
    u.color = BLACK
```

Algoritmo

Ilustração

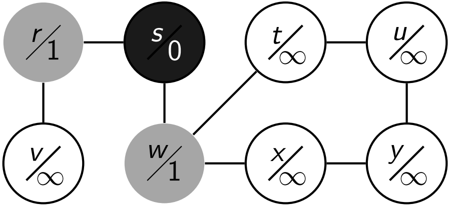


s

Nil	Nil	Nil	Nil	Nil	Nil	Nil	Nil

Algoritmo

Ilustração

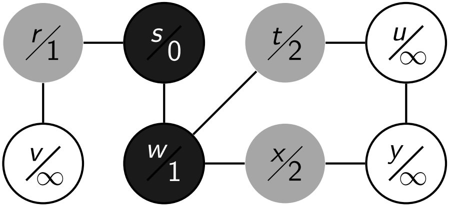


w	r

s	Nil	Nil	Nil	Nil	s	Nil	Nil

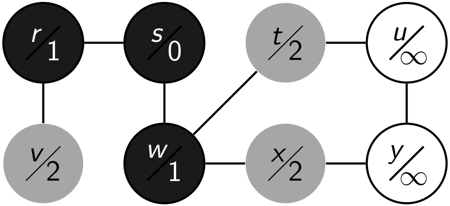
Algoritmo

Ilustração



r	t	x

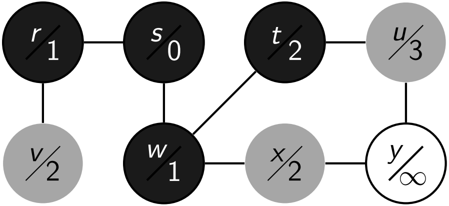
s	Nil	w	Nil	Nil	s	w	Nil



t	x	v

s	Nil	w	Nil	r	s	w	Nil

Algoritmo
Ilustração

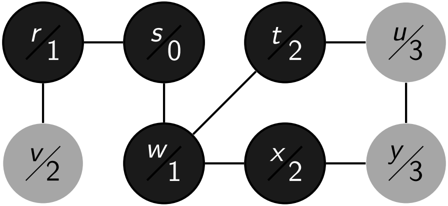


x	v	u

s	Nil	w	u	r	s	w	Nil

Algoritmo

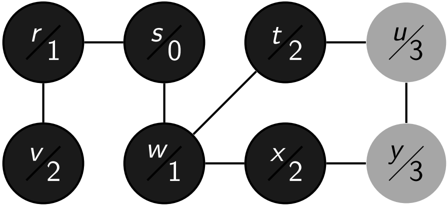
Ilustração



v	u	y

s	Nil	w	u	r	s	w	x

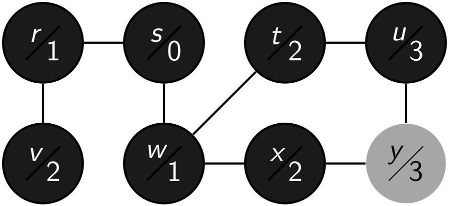
Algoritmo
Ilustração



u	y

s	Nil	w	u	r	s	w	x

Algoritmo
Ilustração

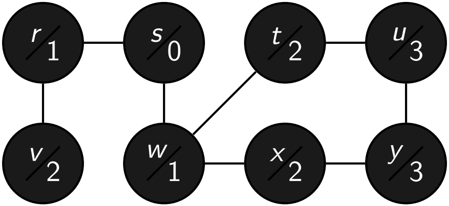


$\boxed{y}$

s	Nil	w	u	r	s	w	x

Algoritmo

Ilustração



s	Nil	w	u	r	s	w	x

Complexidade

- dois laços aninhados
  - um vértice entra em Q quando é branco
  - é imediatamente alterado para cinza
  - logo entra no máximo uma vez em Q
- a cada iteração do laço mais externo, um vértice é eliminado
- logo, o laço mais externo é executado no máximo $|V|$ vezes,
- o laço mais interno enumera as arestas de um nó
- cada aresta é enumerada no máximo duas vezes
- a complexidade é $O(|V| + |E|)$

```
// Processamento
while ¬EMPTY(Q)
    u = HEAD(Q)
    for v ∈ u.adj
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.up = u
            ENQUEUE(Q, v)
    DEQUEUE(Q)
    u.color = BLACK
```

PPgSC

Correção

É correto o algoritmo BFS*?*

Correção

É correto o algoritmo BFS*?*

Roteiro da demonstração:

1. Definições: distância de menor caminho, menor caminho.
2. Propriedade da distância de menor caminho.
3. Propriedade sobre a fila Q no algoritmo BFS
4. Propriedade sobre o atributo d do algoritmo BFS
    - maior ou igual à distância de menor caminho
    - igual à distância de menor caminho
5. Teorema: correção de BFS

Distância de menor caminho, menor caminho

Correção

Definição (Distância de menor caminho)

A distância de menor caminho $\delta(s, v)$ entre dois vértices s e v é o menor número de arestas entre todos os caminhos de s até v, se existir algum caminho. Caso contrário é ∞.

Definição (Menor caminho)

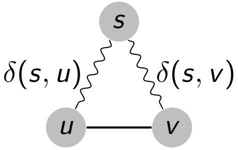
Um caminho entre s e v é um menor caminho se tem $\delta(s, v)$ arestas.

Lema (Lema da distância de menor caminho)

Seja $G = (V, E)$ um grafo (dirigido ou não) e $s \in V$. Para qualquer aresta $(u, v) \in E$, temos: $\delta(s, v) \leq \delta(s, u) + 1$.

Lema (Lema da distância de menor caminho)

Seja $G = (V, E)$ um grafo (dirigido ou não) e $s \in V$. Para qualquer aresta $(u, v) \in E$, temos: $\delta(s, v) \leq \delta(s, u) + 1$.



Demonstração.

Se há um caminho de s até u, também há um até v. O menor caminho até v não pode ser maior que o menor caminho até u seguido de (u, v).
Se não existe caminho de s até u, então $\delta(s, v) = \delta(s, u) = \infty$.

Mostramos então uma propriedade sobre a fila Q

Lema (fila no algoritmo BFS)

Durante a execução de $\mathrm{BFS}(G, s)$, *a fila* $Q = \langle v_1, v_2, \ldots v_r \rangle$ *(*v_1 *cabeça). Então* $v_r.d \leq v_1.d + 1$ *e* $v_i.d \leq v_{i+1}.d$*, para* $i \in 1 \mathinner{\ldotp\ldotp} r - 1$.

Mostramos então uma propriedade sobre a fila Q

Lema (fila no algoritmo BFS)

Durante a execução de $\mathrm{BFS}(G, s)$, a fila $Q = \langle v_1, v_2, \ldots v_r \rangle$ (v_1 cabeça). Então $v_r.d \leq v_1.d + 1$ e $v_i.d \leq v_{i+1}.d$, para $i \in 1 \ldots r-1$.

Demonstração.

Indução sobre operações da fila.

base $Q = \langle s \rangle$ e a propriedade é satisfeita.

$\square$

Mostramos então uma propriedade sobre a fila Q

Lema (fila no algoritmo BFS)

Durante a execução de $\mathrm{BFS}(G, s)$, *a fila* $Q = \langle v_1, v_2, \ldots v_r \rangle$ *(*v_1 *cabeça). Então* $v_r.d \leq v_1.d + 1$ *e* $v_i.d \leq v_{i+1}.d$, *para* $i \in 1\,..\,r-1$.

Demonstração.

	Indução sobre operações da fila.
em geral	prova por caso (Enqueue e Dequeue)

□

Mostramos então uma propriedade sobre a fila Q

Lema (fila no algoritmo BFS)

Durante a execução de $\mathrm{BFS}(G, s)$, *a fila* $Q = \langle v_1, v_2, \ldots v_r \rangle$ *(*v_1 *cabeça). Então* $v_r.d \leq v_1.d + 1$ *e* $v_i.d \leq v_{i+1}.d$*, para* $i \in 1\,..\,r-1$*.*

Demonstração.

Indução sobre operações da fila.

DEQUEUE A nova cabeça é v_2. Por hipótese, temos $v_r.d \leq v_1.+1 \leq v_2.+1$, pela hipótese de indução. As outras desigualdades permanecem.

Mostramos então uma propriedade sobre a fila Q

Lema (fila no algoritmo BFS)

Durante a execução de $\mathrm{BFS}(G, s)$, *a fila* $Q = \langle v_1, v_2, \ldots v_r \rangle$ *(v_1 cabeça). Então* $v_r.d \leq v_1.d + 1$ *e* $v_i.d \leq v_{i+1}.d$*, para* $i \in 1 \,..\, r-1$.

Demonstração.

Indução sobre operações da fila.

ENQUEUE Um vértice v é inserido, tornando-se v_{r+1}. Neste caso, a variável u tem como valor v_1. Logo (v_1, v_{r+1}) é uma aresta, e $v_{r+1}.d = v_1.d + 1$. Temos que $v_r.d \leq v_1.d + 1 = v_{r+1}.d$. As outras desigualdades permanecem.

Prova que $\delta(s, v) \leq v.d$ ...

Lema (atributo d no algoritmo BFS)

Seja $G = (V, E)$ um grafo (dirigido ou não), $s \in V$ um vértice qualquer. Aplicamos o algoritmo $\mathrm{BFS}(G, s)$.
Quando o algoritmo termina, para qualquer v: $v.d \geq \delta(s, v)$.

Propriedade do atributo d no algoritmo BFS

Correção

Prova que $\delta(s, v) \leq v.d$ ...

Lema (atributo d no algoritmo BFS)

Seja $G = (V, E)$ um grafo (dirigido ou não), $s \in V$ um vértice qualquer. Aplicamos o algoritmo $\mathrm{BFS}(G, s)$.
Quando o algoritmo termina, para qualquer v: $v.d \geq \delta(s, v)$.

Demonstração.

Por indução no número de aplicações de ENQUEUE.

base (após ENQUEUE(s)) $v.d = 0 = \delta(s, s)$, e , se $v \neq s$, $v.d = \infty \geq \delta(s, v)$.

caso geral seja v um vértice branco alcançado na visita de u.

- por hipótese de indução, $u.d \geq \delta(s, u)$
- o algoritmo diz que $v.d = u.d + 1$
- logo $v.d \geq \delta(s, u) + 1$
- e, pelo lema da distância de menor caminho, $v.d \geq \delta(s, v)$.

o valor de $v.d$ não é mais alterado

Mostramos enfim que BFS calcula as distâncias de menor caminho.

Teorema (Correção do algoritmo BFS)

Durante a execução de $\mathrm{BFS}(G, s)$, *para cada vértice* $v \in V$ *alcançável a partir de* s*:*

- v *é processado,*
- *no término,* $v.d = \delta(s, v)$.
- *se* $v \neq s$, *um dos menores caminhos de* s *até* v *é composto por um dos menores caminhos de* s *até* $v.up$ *e pela aresta* $(v.up, v)$.

Por casos.

- v não é alcançável a partir de s
- v é alcançável a partir de s

$\square$

Por casos.

v não é alcançável a partir de s:

- $\delta(s, v) = \infty$.
- Pelo lema do atributo d, $v.d \geq \delta(s, v)$,
- como não é possível atribuir ∞ a $v.d$ no laço principal de BFS,
- logo o laço só processa nós alcançáveis a partir de s.
- Então, no término do algoritmo $v.d = \infty = \delta(s, v)$.

$\square$

Por casos.

v é alcançável a partir de s:

- Seja $V_k = \{v \in V \mid \delta(s, v) = k\}$
- prova por indução sobre k.
- propriedade sobre k: se $v \in V_k$, existe um ponto durante a execução tal que
  1. $v.color = \text{GRAY}$
  2. $v.d$ é atribuído k
  3. se $v \neq s$, então $v.up$ é atribuído u, onde $u \in V_{k-1}$
  4. v é inserido em Q
- obs.: se esse ponto existir, necessariamente é unico (*color*, *d* e *up* são atribuídos no máximo uma vez).

Caso de base: $k = 0$

- Por definição, $V_0 = \{s\}$.
- Na fase de inicialização do algoritmo:
  1. $s.color$ é atribuído $\textsc{Gray}$
  2. $s.d$ é atribuído 0
  3. s é inserido em Q

Em geral: $k > 0$

- $Q \neq \langle\rangle$ antes do término.
- pelo lema da fila, se os vértices são inseridos em ordem $v_1, v_2, \dots v_r$ então $v_1.d \leq v_2.d \leq \dots v_r.d$.
- seja $v \in V_k$, $k \geq 1$,
- v só pode ser encontrado quando todos os vértices de V_{k-1} tiverem entrados na fila.
- $v \in V_k \Rightarrow \delta(s, v) = k$, logo há um menor caminho de k arestas entre s e v, passando por $u \in V_{k-1}$, e $(u, v) \in E$.
- seja u o primeiro desses vértices que foi encontrado: u deve ter entrado em Q
- u necessariamente torna-se a cabeça de Q em um momento
- v deve ser descoberto quando processa as arestas de u, e
    1. $v.color$ é atribuído GRAY
    2. $v.d$ é atribuído $u.d + 1 = \delta(s, u) + 1 = (k - 1) + 1 = k$
    3. $v.up$ é atribuído u
    4. v é inserido em Q

- Isso conclui a prova por indução
- Além disto, quando $v \in V_k$, então $v.up \in V_{k-1}$.
- podemos formar um menor caminho entre s e v com o menor caminho entre s e $v.up$, seguido da aresta $(v.up, v)$.

BFS e árvore de processamento em largura

- BFS calcula também uma árvore, formada pelas arestas $(v.up, v)$.
- Esta árvore é uma árvore de processamento em largura.

Definição (Sub-grafo dos predecessores)

Seja $G = (V, E)$ um grafo, o sub-grafo dos predecessores é o grafo $G_\pi = (V_\pi, E_\pi)$, onde

- $V_\pi = \{s\} \cup \{v \in V \mid v.up \neq \textsc{Nil}\}$, e
- $E_\pi = \{(v.up, v) \mid v \in V_\pi - \{s\}\}$

Definição (Árvore de processamento em largura)

Um sub-grafo dos predecessores é uma árvore de processamento em largura quando V_π é composto por s e por todos os vértices alcançáveis $v \in V_\pi$, existe um único caminho entre s e v em G_π. Este caminho é um menor caminho entre s e v em G.

Exercícios

1. Escreva um algoritmo que, dado um grafo $G = (V, E)$, dois vértices u e v, imprime um menor caminho entre u e v, se esse existir.
2. Qual seria a complexidade de BFS, caso seja adotada uma matriz de adjacência, uma vez feitas as adaptações necessárias?
3. Mostre que o valor atribuído a $u.d$ é independente da ordem dos vértices nas listas de adjacência.
4. Um grafo é bipartido se o conjunto de vértices pode ser separado em dois conjuntos, digamos V_1 e V_2, tal que todas as arestas relacionam um vértice de V_1 com um vértice de V_2. Forneça um algoritmo eficiente que testa se um grafo é bipartido.

PPgSC

Busca em profundidade

- algoritmo elementar para visitar os vértices
- base para outros algoritmos importantes
- entrada: $G = (V, E)$
  - dirigido ou não
- resultado:
  - floresta de árvores “em profundidade primeiro” dos vértices de G
- profundidade: prioriza vértices mais distantes do ponto de partida.

PPgSC

Princípios

- inicia em um vértice
- visita os vértices alcançáveis, indo o mais "profundo" possível
- volte e explora outro caminho repetidamente
- recomeça a busca a partir de outro vértice não visitado
- gera uma floresta
- cada vértice v tem atributo $v.up$.
- Sub-grafo dos predecessores:
  $G_\pi = \{(v.up, v) \mid v \in V \wedge v.up \neq \text{NIL}\}$.
- A mesma convenção de coloração é utilizada
- Cada vértice tem dois marcadores:
  - $v.d$: quando v é descoberto
  - $v.f$: quando f é finalizado
  - $v.color = \text{BLACK} \Rightarrow v.d < v.f$
  - variável global *time* marca as etapas

PPgSC

Algoritmo

TICK

```
time = time + 1
return time
```

DFS(G)

```
for v ∈ G.V
    v.color = WHITE
    v.up = NIL
time = 0
for v ∈ G.V
    if v.color == WHITE
        DFS-VISIT(v)
```

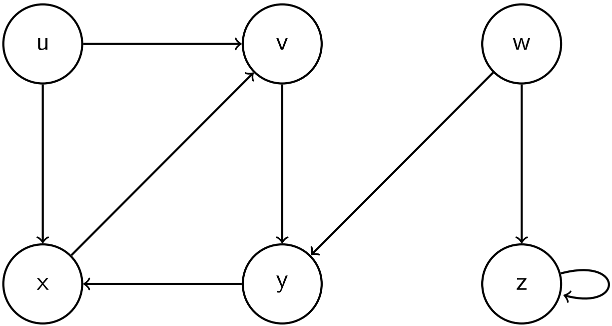
Algoritmo

$\text{DFS-VISIT}(v)$

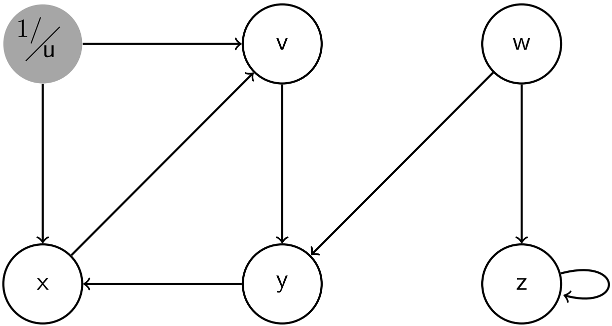
```
v.color = GRAY
v.d = TICK
for w ∈ v.adj
    if w.color == WHITE
        w.up = v
        DFS-VISIT(w)
v.color = BLACK
v.f = TICK
```

PPgSC

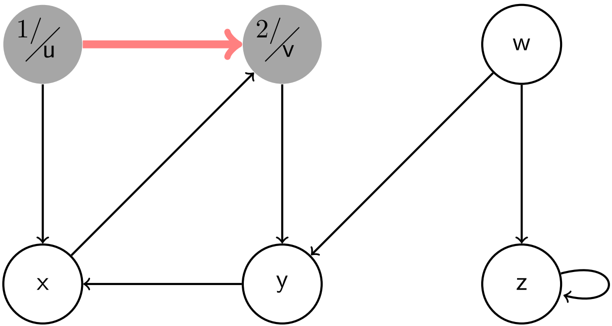
Ilustração



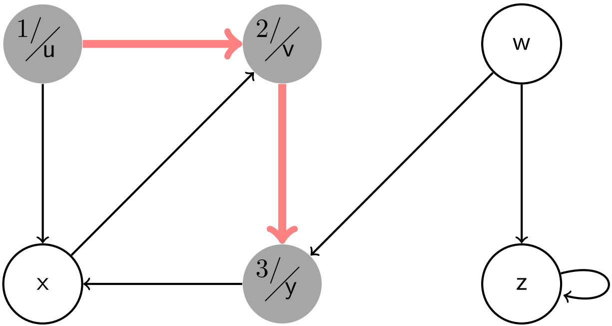
Ilustração



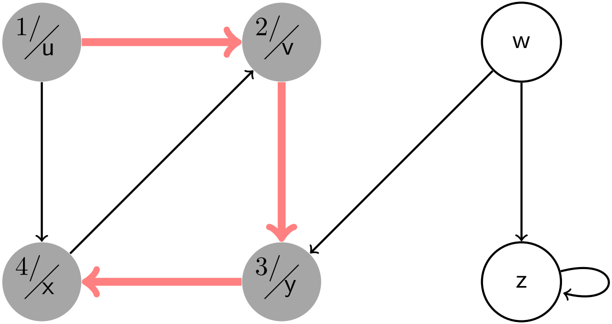
Ilustração



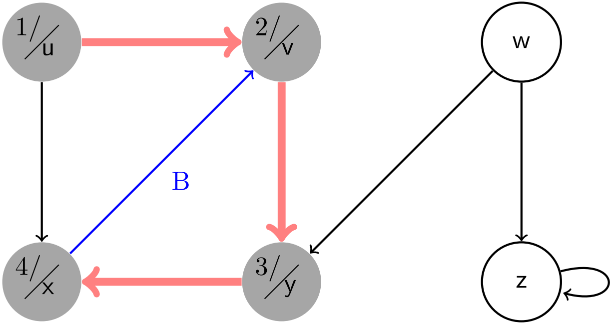
Ilustração



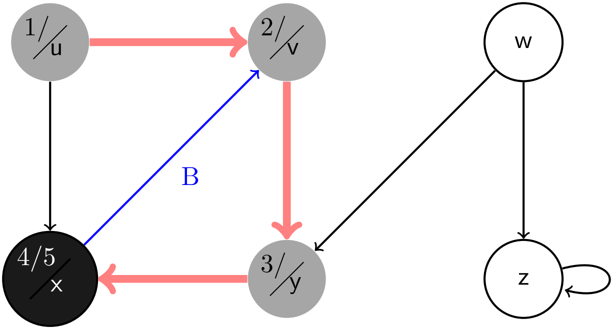
Ilustração



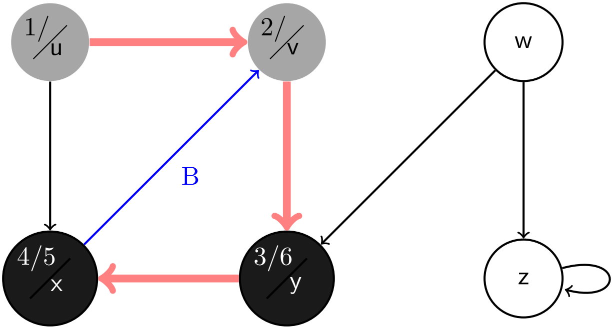
Ilustração



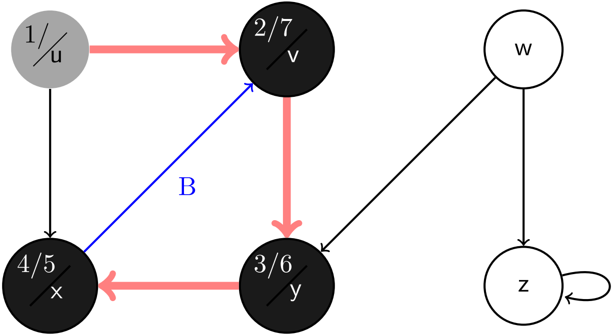
Ilustração



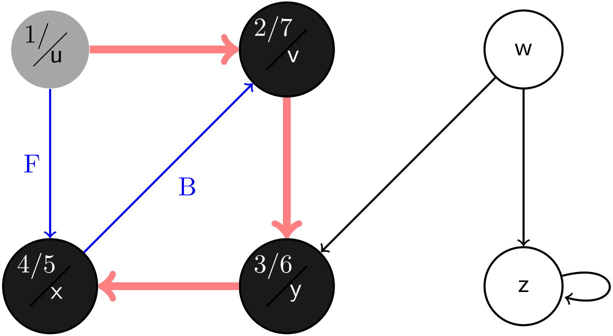
Ilustração



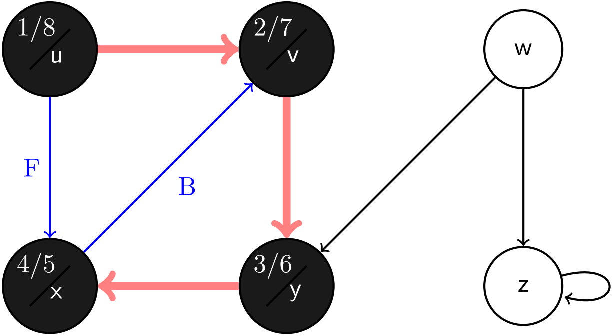
Ilustração



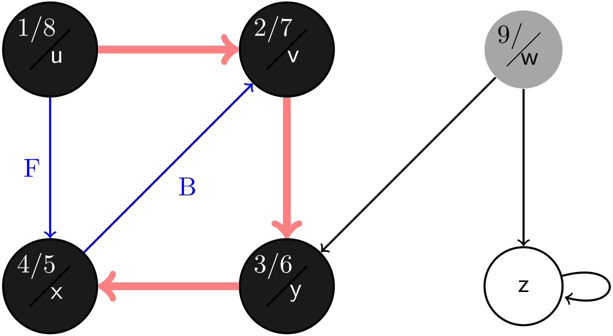
Ilustração



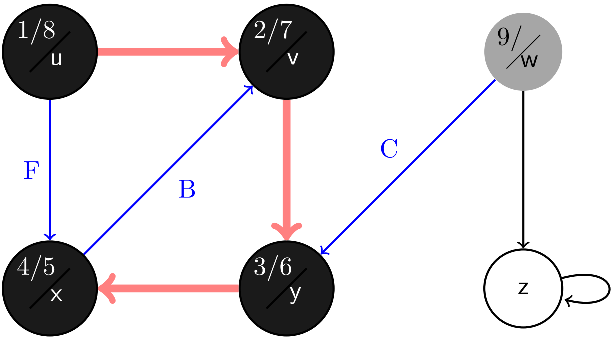
Ilustração



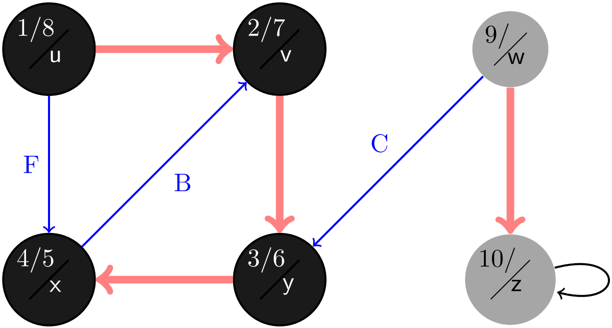
Ilustração



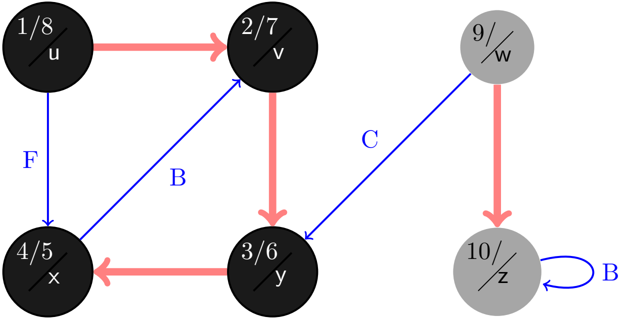
Ilustração

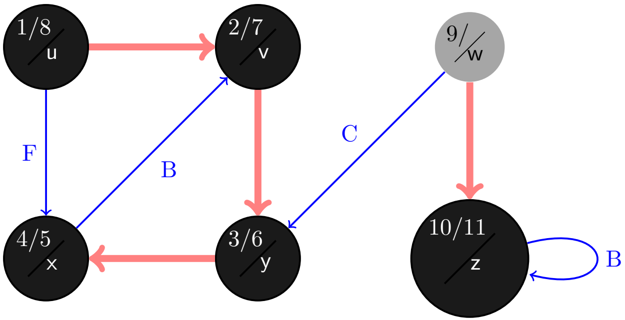


Ilustração

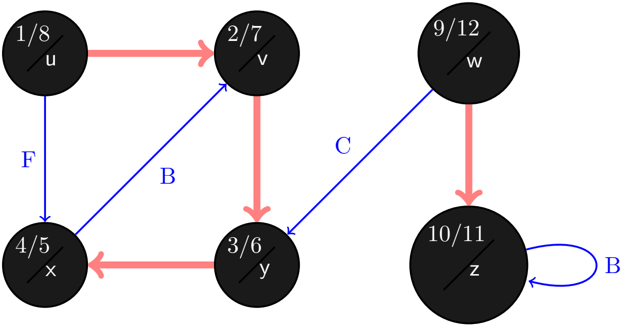


Ilustração




1/8
u
2/7
v
9/
w
F
B
C
4/5
x
3/6
y
10/11
z
B

Ilustração



Complexidade do algoritmo

DFS(G)

for $v \in G.V$ // $\Theta(|V|)$

$v.color = $ WHITE

$v.up = $ NIL

$time = 0$

for $v \in G.V$ // $\Theta(|V|)$

if $v.color == $ WHITE

DFS-VISIT(v)

Complexidade do algoritmo

```
DFS-VISIT(v)
    // Θ(|V|) chamadas
    v.color = GRAY
    v.d = TICK
    for w ∈ v.adj // total: Θ(|E|)
        if w.color == WHITE
            w.up = v
            DFS-VISIT(w)
    v.color = BLACK
    v.f = TICK
```

Complexidade do algoritmo

$$\Theta(|V| + |E|)$$

Propriedades

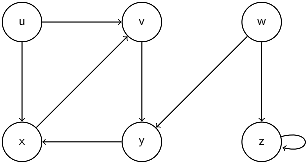
1. a relação predecessor forma uma floresta de árvores
2. estrutura de aninhamento da busca: teorema dos parênteses
3. condição necessária e suficiente de alcançabilidade: teorema do caminho branco
4. classificação das arestas na busca em profundidade
5. busca em profundidade em grafos não dirigidos

A relação predecessor

- O grafo dos predecessores G_π é uma floresta de árvores.
- as arestas correspondem às chamadas de DFS-VISIT

Estrutura aninhada

- Etapas de descoberta e finalização formam uma estrutura de parênteses.
  - $v.d = ... \Longrightarrow (v$
  - $v.f = ... \Longrightarrow v)$
  - exemplo $(u(v(y(xx)y)v)u)(w(zz)w)$



Teorema (Teorema dos parênteses)

No percurso em profundidade de um grafo G, para qualquer par de vértices u e v, uma das três condições seguintes é satisfeita:

1. *os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ não têm superposição;*
2. *o intervalo $[u.d, u.f]$ é inteiramente contido em $[v.d, v.f]$;*
3. *o intervalo $[u.d, u.f]$ contem inteiramente $[v.d, v.f]$.*

Corolário (Aninhamento dos intervalos dos descendentes)

Se v é um descendente próprio de u na floresta de busca em profundidade, então

$$u.d < v.d < v.f < u.f$$

Demonstração.

Por casos:

1. se $u.d < v.d$
   1.1 se $v.d < u.f$:
      - v é descoberto quando u é cinza, logo v é descendente de u.
      - v é mais recente que u e as arestas saindo de v são processadas antes de finalizar as de u, logo $v.f < u.f$.
      - O intervalo para v é contido no intervalo para u.

   1.2 se $u.f < v.d$:
      - $v.d < v.f$
      - o intervalo para u vem antes do intervalo para v.
      - não há sobreposição.
2. caso simétrico: raciocínio simétrico

Teorema do caminho branco

Teorema (Teorema do caminho branco)

Na floresta de profundidade de um grafo, dirigido ou não, o vértice v é descendente de u s e somente se na etapa $u.d$, v pode ser alcançado a partir de u por um caminho composto apenas por vértices brancos.

Teorema do caminho branco

Teorema (Teorema do caminho branco)

Na floresta de profundidade de um grafo, dirigido ou não, o vértice v é descendente de u s e somente se na etapa $u.d$, v pode ser alcançado a partir de u por um caminho composto apenas por vértices brancos.

Demonstração.

$\Rightarrow$

- v é um descendente de u,
- w é um vértice no caminho de u até v,
- pelo corolário do aninhamento, $u.d < w.d$.
- logo, na etapa $w.d$, w é branco.

$\square$

Teorema do caminho branco

Teorema (Teorema do caminho branco)

Na floresta de profundidade de um grafo, dirigido ou não, o vértice v é descendente de u s e somente se na etapa $u.d$, v pode ser alcançado a partir de u por um caminho composto apenas por vértices brancos.

Demonstração.

$\Leftarrow$

- na etapa $u.d$, v é alcançável por um caminho branco a partir de u
- supondo que v não é descendente de u na árvore em profundidade
- supondo que os demais vértices no caminho tornam-se descendentes na árvore em profundidade
- seja w o predecessor de v
- temos que $w.f < u.f$

Classificação das arestas

- arestas de árvore: são as arestas (u, v) processadas em DFS-VISIT tais que v é encontrado pela primeira vez.
- arestas para trás: são as arestas processadas em DFS-VISIT que conectam v a um ancestro de v
- arestas para frente: são as arestas processadas em DFS-VISIT que conectam v a um descendente de v já visitado
- arestas cruzadas: as demais arestas.
  - entre vértices da mesma árvore, tais que nenhuma é ancestro de outra;
  - entre vértices de árvores diferentes.

PPgSC

Classificação das arestas

- arestas de árvore: são as arestas (u, v) processadas em DFS-VISIT tais que v é encontrado pela primeira vez.
- arestas para trás: são as arestas processadas em DFS-VISIT que conectam v a um ancestro de v
- arestas para frente: são as arestas processadas em DFS-VISIT que conectam v a um descendente de v já visitado
- arestas cruzadas: as demais arestas.
  - entre vértices da mesma árvore, tais que nenhuma é ancestro de outra;
  - entre vértices de árvores diferentes.

Como o algoritmo DFS-VISIT pode identificar o tipo de uma aresta?

Classificação das arestas

- arestas de árvore: são as arestas (u, v) processadas em DFS-VISIT tais que v é encontrado pela primeira vez. $v.color = \text{WHITE}$
- arestas para trás: são as arestas processadas em DFS-VISIT que conectam v a um ancestro de v
- arestas para frente: são as arestas processadas em DFS-VISIT que conectam v a um descendente de v já visitado
- arestas cruzadas: as demais arestas.
  - entre vértices da mesma árvore, tais que nenhuma é ancestro de outra;
  - entre vértices de árvores diferentes.

Como o algoritmo DFS-VISIT pode identificar o tipo de uma aresta?

Classificação das arestas

- arestas de árvore: são as arestas (u, v) processadas em DFS-VISIT tais que v é encontrado pela primeira vez.
- arestas para trás: são as arestas processadas em DFS-VISIT que conectam v a um ancestro de v $v.color = \text{GRAY}$
- arestas para frente: são as arestas processadas em DFS-VISIT que conectam v a um descendente de v já visitado
- arestas cruzadas: as demais arestas.
  - entre vértices da mesma árvore, tais que nenhuma é ancestro de outra;
  - entre vértices de árvores diferentes.

Como o algoritmo DFS-VISIT pode identificar o tipo de uma aresta?

Classificação das arestas

- arestas de árvore: são as arestas (u, v) processadas em DFS-VISIT tais que v é encontrado pela primeira vez.
- arestas para trás: são as arestas processadas em DFS-VISIT que conectam v a um ancestro de v
- arestas para frente: são as arestas processadas em DFS-VISIT que conectam v a um descendente de v já visitado
  $v.color = \text{BLACK}$ $u.d < v.d$
- arestas cruzadas: as demais arestas. $v.color = \text{BLACK}$
  - entre vértices da mesma árvore, tais que nenhuma é ancestro de outra;
  - entre vértices de árvores diferentes.

Como o algoritmo DFS-VISIT pode identificar o tipo de uma aresta?

Classificação das arestas

- arestas de árvore: são as arestas (u, v) processadas em $\text{DFS-}\textsc{visit}$ tais que v é encontrado pela primeira vez.
- arestas para trás: são as arestas processadas em $\text{DFS-}\textsc{visit}$ que conectam v a um ancestro de v
- arestas para frente: são as arestas processadas em $\text{DFS-}\textsc{visit}$ que conectam v a um descendente de v já visitado $u.d < v.d$
- arestas cruzadas: as demais arestas. $v.color = \textsc{Black}$ $u.d > v.d$
  - entre vértices da mesma árvore, tais que nenhuma é ancestro de outra;
  - entre vértices de árvores diferentes.

Como o algoritmo $\text{DFS-}\textsc{visit}$ pode identificar o tipo de uma aresta?

Arestas em grafos não dirigidos

Teorema

Na busca em profundidade de um grafo não dirigido, todas as arestas são de árvore ou para trás.

Demonstração.

- Seja (u, v) uma aresta em um grafo não dirigido.
- Se $u.d < v.d$:
  - Então v deve ser finalizado antes de u, pois (u, v) pertence à $u.adj$.
  - Se (u, v) é processada primeiro em $\text{BFS-VISIT}(u)$, então é de árvore.
  - Se (u, v) é processada primeiro em $\text{BFS-VISIT}(v)$, então é para trás.
- Se $v.d < u.d$: argumento simétrico.

Exercícios

1. Desenha uma tabela 3 por 3 com *White*, *Gray Black* como cabeçalho das linhas e colunas.
   Indique em i, j se DFS-VISIT pode encontrar uma aresta entre vértices da cor i e j em um grafo dirigido, e o tipo de aresta correspondente.
   Repita o exercício para grafos não dirigidos.
2. Considere a afirmação: Em um grafo dirigido, se há um caminho de u para v, e $u.d < v.d$ em uma busca em profundidade, então v será um descendente de u na floresta correspondente.
   Dê um contra-exemplo que invalida esta proposição.
3. Altere DFS-VISIT para imprimir as arestas visitadas e seus tipos.
   É necessário modificações para tratar grafos não dirigidos?
4. u é um vértice com arestas entrando e saindo. Em um percurso em profundidade, u encontra-se o único vértice de uma das árvores da floresta produzida. Como isto é possível?