

Projeto e Análise de Algoritmos II (MC558)

Busca em grafos

Prof. Dr. Ruben Interian

Resumo

- 1 Introdução e objetivo
- 2 Algoritmos de busca em grafos
- 3 Busca em largura, BFS
- 4 Síntese

Resumo

- 1 Introdução e objetivo
- 2 Algoritmos de busca em grafos
- 3 Busca em largura, BFS
- 4 Síntese

Introdução

- Até agora, vimos diversos **conceitos e resultados** da [Teoria de grafos](#). Vimos também como os grafos são **representados computacionalmente**.

Introdução

- Até agora, vimos diversos **conceitos e resultados** da **Teoria de grafos**. Vimos também como os grafos são **representados computacionalmente**.
- Vamos começar a estudar como estes conceitos, resultados e estruturas de dados podem ser usados para **resolver problemas**.

Introdução

Como explorar/percorrer um grafo?

- O grafo pode ser direcionado ou não direcionado.
- Um dos objetivos pode ser identificar vértices com características estruturais específicas (e.g., identificar as componentes do grafo).
- Outro objetivo: conseguir enumerar os vértices em uma ordem específica.
- A enumeração de vértices seguindo uma determinada ordem é usada em diversos outros algoritmos que veremos no curso, por exemplo: encontrar uma ordenação topológica, encontrar caminhos mínimos.
- **Algoritmos de busca em grafos são a base de alguns dos mais importantes algoritmos que veremos no curso.**

Resumo

- 1 Introdução e objetivo
- 2 Algoritmos de busca em grafos
- 3 Busca em largura, BFS
- 4 Síntese

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis?

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis? – **Sim**.

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis? – **Sim**.
- Para explorar/percorrer um grafo, faz sentido andar em círculos (**ciclos**)?

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis? – Sim.
- Para explorar/percorrer um grafo, faz sentido andar em círculos (**ciclos**)? – Não.

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis? – Sim.
- Para explorar/percorrer um grafo, faz sentido andar em círculos (**ciclos**)? – Não.
- Seria desejável saber qual vértice foi visitado a partir de qual.

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis? – **Sim**.
- Para explorar/percorrer um grafo, faz sentido andar em círculos (**ciclos**)? – **Não**.
- Seria desejável saber qual vértice foi visitado a partir de qual.

Estrutural acíclica, contém todos os vértices \Rightarrow

Algoritmos de busca em grafos

Algoritmo de busca em grafos:

- Entrada: Grafo G .
- Saída: ?

Quais características estamos procurando:

- Precisamos percorrer **todos** os vértices possíveis? – Sim.
- Para explorar/percorrer um grafo, faz sentido andar em círculos (**ciclos**)? – Não.
- Seria desejável saber qual vértice foi visitado a partir de qual.

Estrutural acíclica, contém todos os vértices \Rightarrow

Árvore (floresta?) **geradora**.

Algoritmos de busca em grafos

Vamos estudar **dois algoritmos** de busca em grafos:

- Busca em largura
 - É chamada de **BFS**, do inglês **breadth-first search**.
- Busca em profundidade
 - É chamada de **DFS**, do inglês **depth-first search**.

Algoritmos de busca em grafos

Vamos estudar **dois algoritmos** de busca em grafos:

- Busca em largura
 - É chamada de **BFS**, do inglês **breadth-first search**.
- Busca em profundidade
 - É chamada de **DFS**, do inglês **depth-first search**.

Ambos os algoritmos:

- recebem como **entrada** um **grafo** G , e
- geram como **saída** uma **arvore** (floresta) **geradora**.

Representação de árvores

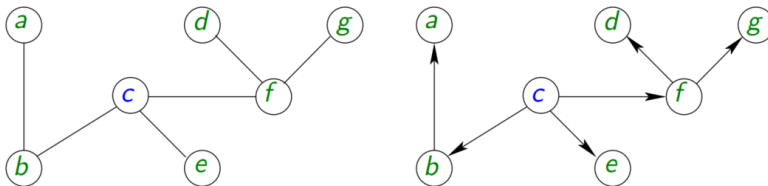
Como representar essa **arvore**? (chamada de **árvore de busca**)

- Vamos usar uma árvore **enraizada**, cuja raiz é um vértice inicial **s**.
- Veja que podemos representar essa árvore por meio de um **vetor** π .
- O pai de um vértice **v** é $\pi[v]$. O vértice raiz não tem pai: $\pi[s] = \text{NIL}$.

Vetor π ($\pi[v]$ é pai de v):

v	a	b	c	d	e	f	g
$\pi[v]$	b	c	NIL	f	c	c	f

NIL: símbolo usado para indicar não existência.



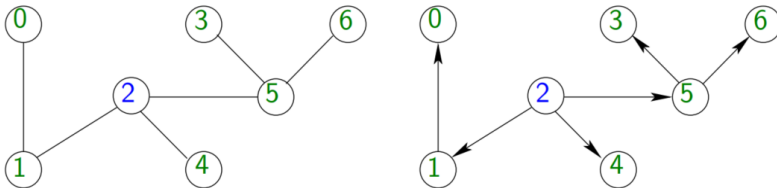
raiz **c** (arcos orientados da raiz até as folhas)

Representação de árvores

Vetor π ($\pi[v]$ é pai de v):

v	0	1	2	3	4	5	6
$\pi[v]$	1	2	<i>NIL</i>	5	2	2	5

NIL: símbolo usado para indicar não existência.



raiz 2 (arcos orientados da raiz até as folhas)

Representação de árvores

- Essa representação permite determinar facilmente o caminho que vai de um vértice v até a raiz s : $v, \pi[v], \pi[\pi[v]], \dots, s$.
- Podemos chegar de s a v usando o caminho inverso.

Representação de árvores

PrintPath (G, s, v)

```
1: se  $v == s$  então
2:   Print  $s$ 
3: senão se  $\pi[v] = \text{NIL}$  então
4:   Print Não existe caminho de  $s$  a  $v$ 
5: senão
6:   PrintPath ( $G, s, \pi[v]$ )
7:   Print  $v$ 
```

Representação de árvores

PrintPath (G, s, v)

```

1: se  $v == s$  então
2:     Print  $s$ 
3: senão se  $\pi[v] = \text{NIL}$  então
4:     Print Não existe caminho de  $s$  a  $v$ 
5: senão
6:     PrintPath ( $G, s, \pi[v]$ )
7:     Print  $v$ 

```

Imprime o caminho de **s** a **v** na árvore de raiz **s** em tempo **linear** no tamanho do caminho.

Resumo

- 1 Introdução e objetivo
- 2 Algoritmos de busca em grafos
- 3 Busca em largura, BFS
- 4 Síntese

Busca em largura: ideia do algoritmo

Ideia do algoritmo BFS:

- Começar pelo vértice inicial **s**;
- Depois os vizinhos de **s**;
- Depois os vizinhos dos vizinhos de **s**;
- ... [e assim por diante]

Busca em largura

Ideias para implementar o algoritmo (e obter a árvore geradora):

- A **árvore de busca** no início contém apenas o vértice inicial **s**.
- Cada vizinho **v** de **s** e a aresta **(s, v)** são acrescentadas à árvore;
- O processo é repetido para os vizinhos dos vizinhos, e assim até que todos os vértices alcançáveis por **s** sejam inseridos na árvore.
- Qual estrutura de dados permite “lembrar” qual é o próximo vértice a ser visitado?

Busca em largura

Ideias para implementar o algoritmo (e obter a árvore geradora):

- A **árvore de busca** no início contém apenas o vértice inicial **s**.
- Cada vizinho **v** de **s** e a aresta **(s, v)** são acrescentadas à árvore;
- O processo é repetido para os vizinhos dos vizinhos, e assim até que todos os vértices alcançáveis por **s** sejam inseridos na árvore.
- Qual estrutura de dados permite “lembrar” qual é o próximo vértice a ser visitado? – **Fila Q**, com as operações **Enqueue** e **Dequeue**.

Busca em largura

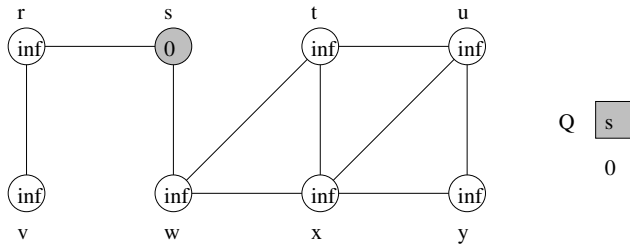
Como podemos saber **quais vértices já foram visitados?** – Usaremos **cores**.

Busca em largura

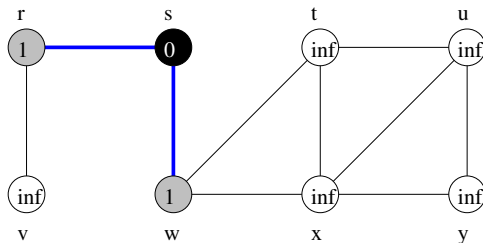
Como podemos saber **quais vértices já foram visitados**? – Usaremos **cores**.

- **Branco** = “ainda não visitado” (inicialmente todos os vértices são brancos).
- **Cinza** = “na fila, ainda não finalizado” (ainda não analisei seus vizinhos).
- **Preto** = “visitado e finalizado” (vizinhos já analisados).

Busca em largura: Exemplo



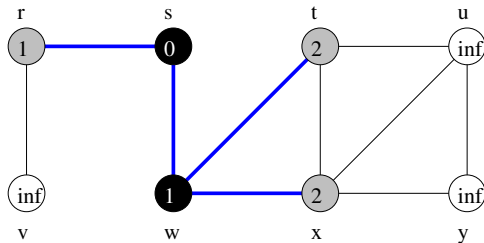
Busca em largura: Exemplo



Q

w	r
1	1

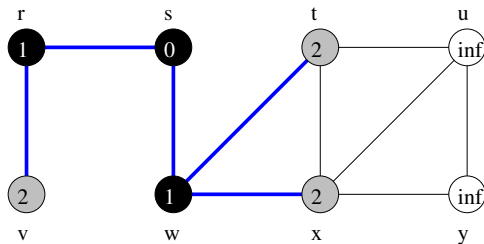
Busca em largura: Exemplo



Q

r	t	x
1	2	2

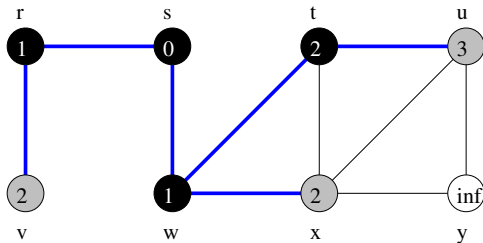
Busca em largura: Exemplo



Q

t	x	v
2	2	2

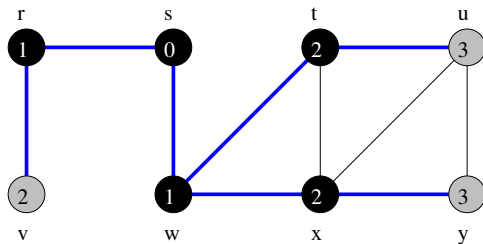
Busca em largura: Exemplo



Q

x	v	u
2	2	3

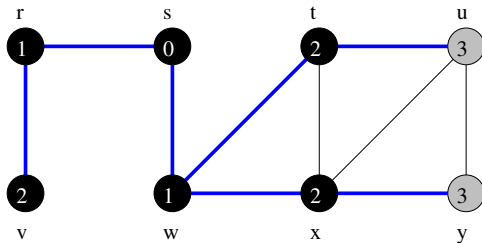
Busca em largura: Exemplo



Q

v	u	y
2	3	3

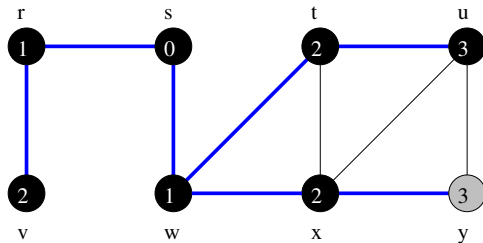
Busca em largura: Exemplo



Q

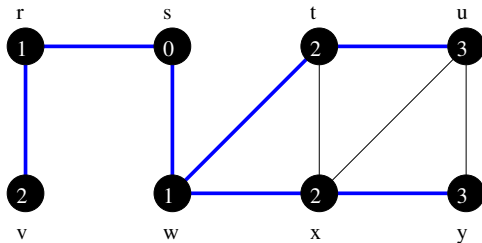
u	y
3	3

Busca em largura: Exemplo



Q y
3

Busca em largura: Exemplo



Busca em largura: O algoritmo

Qual **estrutura de dados** é melhor usar neste algoritmo: [matriz](#) ou [listas de adjacência](#)?

Busca em largura: O algoritmo

Qual **estrutura de dados** é melhor usar neste algoritmo: **matriz** ou **listas de adjacência**?

Por quê?

Busca em largura: O algoritmo

Qual **estrutura de dados** é melhor usar neste algoritmo: **matriz** ou **listas de adjacência**?

Por quê?

O algoritmo BFS recebe um **grafo** G , na forma de **listas de adjacência**, e um vértice inicial $s \in V$, e devolve uma **arvore de busca em largura**, e adicionalmente as **distâncias** dos vértices ao vértice inicial s .

Busca em largura: O algoritmo

BFS (G, s)

- 1: **para cada** $u \in V - \{s\}$ **faça**
 - 2: $\text{cor}[u] \leftarrow \text{branco}$
 - 3: $d[u] \leftarrow \infty$
 - 4: $\pi[u] \leftarrow \text{NIL}$
 - 5: $\text{cor}[s] \leftarrow \text{cinza}$
 - 6: $d[s] \leftarrow 0$
 - 7: $\pi[s] \leftarrow \text{NIL}$
 - 8: $Q \leftarrow \emptyset$
 - 9: Enqueue(Q, s)
-

(Inicialização)

Busca em largura: O algoritmo

BFS (G, s)

```
1: para cada  $u \in V - \{s\}$  faça
2:    $cor[u] \leftarrow \text{branco}$ 
3:    $d[u] \leftarrow \infty$ 
4:    $\pi[u] \leftarrow \text{NIL}$ 
5:  $cor[s] \leftarrow \text{cinza}$ 
6:  $d[s] \leftarrow 0$ 
7:  $\pi[s] \leftarrow \text{NIL}$ 
8:  $Q \leftarrow \emptyset$ 
9: Enqueue( $Q, s$ )
```

(Inicialização)

```
10: enquanto  $Q \neq \emptyset$  faça
11:    $u \leftarrow \text{Dequeue}(Q)$ 
12:   para cada  $v \in Adj[u]$  faça
13:     se  $cor[v] = \text{branco}$  então
14:        $cor[v] \leftarrow \text{cinza}$ 
15:        $d[v] \leftarrow d[u] + 1$ 
16:        $\pi[v] \leftarrow u$ 
17:       Enqueue( $Q, v$ )
18:  $cor[u] \leftarrow \text{preto}$ 
```

(Execução)

Busca em largura: complexidade de tempo

Método de análise **agregado**.

Busca em largura: complexidade de tempo

Método de análise **agregado**.

- A inicialização consome tempo $\Theta(V)$.

Busca em largura: complexidade de tempo

Método de análise **agregado**.

- A inicialização consome tempo $\Theta(V)$.
- Depois que um vértice deixa de ser branco, ele não volta a ser branco novamente. Assim, cada vértice é inserido na fila Q no máximo 1 vez. Cada operação sobre a fila consome tempo $\Theta(1)$ resultando em um total de $O(V)$.

Busca em largura: complexidade de tempo

Método de análise **agregado**.

- A inicialização consome tempo $\Theta(V)$.
- Depois que um vértice deixa de ser branco, ele não volta a ser branco novamente. Assim, cada vértice é inserido na fila Q no máximo 1 vez. Cada operação sobre a fila consome tempo $\Theta(1)$ resultando em um total de $O(V)$.
- Em cada lista de adjacência, os vértices são visitados apenas 1 vez (linhas 12-13). A soma dos comprimentos das listas de adjacência é $\Theta(E)$. Assim, o tempo gasto para percorrer as listas é $O(E)$.

Busca em largura: complexidade de tempo

Método de análise **agregado**.

- A inicialização consome tempo $\Theta(V)$.
- Depois que um vértice deixa de ser branco, ele não volta a ser branco novamente. Assim, cada vértice é inserido na fila Q no máximo 1 vez. Cada operação sobre a fila consome tempo $\Theta(1)$ resultando em um total de $O(V)$.
- Em cada lista de adjacência, os vértices são visitados apenas 1 vez (linhas 12-13). A soma dos comprimentos das listas de adjacência é $\Theta(E)$. Assim, o tempo gasto para percorrer as listas é $O(E)$.

Conclusão: A complexidade de tempo do algoritmo **BFS** é $O(V + E)$.

Busca em largura: correção do algoritmo

Vamos mostrar que **BFS** funciona. Para isso, precisamos mostrar duas coisas:

- 1 O vetor π define uma **árvore de busca** com raiz s .
- 2 $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Busca em largura: correção do algoritmo

Vamos mostrar que **BFS** funciona. Para isso, precisamos mostrar duas coisas:

- 1 O vetor π define uma **árvore de busca** com raiz s .
- 2 $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Teorema

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

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

Busca em largura: correção do algoritmo

Para provar o **Teorema**, 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 $d[v]$.

Busca em largura: 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 $d[v]$.

(*) As arestas de $T = (V_\pi, E_\pi)$ são $(\pi[v], v)$, para todo v , $\pi[v] \neq \text{NIL}$.

Busca em largura: 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 $d[v]$.

(*) As arestas de $T = (V_\pi, E_\pi)$ são $(\pi[v], v)$, para todo v , $\pi[v] \neq \text{NIL}$.

Prova. (por indução no número de operações **Enqueue** – “Enfileirar”)

Busca em largura: 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 $d[v]$.

(*) As arestas de $T = (V_\pi, E_\pi)$ são $(\pi[v], v)$, para todo v , $\pi[v] \neq \text{NIL}$.

Prova. (por indução no número de operações **Enqueue** – “Enfileirar”)

Base: Depois que executamos **Enqueue** pela primeira vez, T continha apenas s , com $d[s] = 0$. Nem $d[s]$ (nem nenhum $d[v]$) nunca muda após o vértice ser inserido na fila.

Busca em largura: Lema 1

Hipótese de indução: antes de enfileirar o vértice v , todo vértice u com $d[u] < \infty$ já está em T , e possui um caminho de s a u em T , com comprimento $d[u]$.

Busca em largura: Lema 1

Hipótese de indução: antes de enfileirar o vértice v , todo vértice u com $d[u] < \infty$ já está em T , e possui um caminho de s a u em T , com comprimento $d[u]$.

No instante em que enfileiramos v :

- v foi descoberto percorrendo os vizinhos de um vértice u ;
- mas u já havia sido enfileirado;
- pela **hipótese de indução**, existe um caminho de s a u com comprimento $d[u]$.

Busca em largura: Lema 1

Hipótese de indução: antes de enfileirar o vértice v , todo vértice u com $d[u] < \infty$ já está em T , e possui um caminho de s a u em T , com comprimento $d[u]$.

No instante em que enfileiramos v :

- v foi descoberto percorrendo os vizinhos de um vértice u ;
- mas u já havia sido enfileirado;
- pela **hipótese de indução**, existe um caminho de s a u com comprimento $d[u]$.

Portanto, v está em T (linha 16: $\pi[v] \leftarrow u$), e há caminho de s a v em T , pois $\pi[v] = u$, sendo $d[v] = d[u] + 1$.

→ Algoritmo

Busca em largura: Lema 1

Corolário 1

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

Busca em largura: Lema 2

Lema 2

Suponha que $\langle v_1, v_2, \dots, v_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então:

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

Ou seja, os vértices são inseridos na fila em ordem não-decrescente dos valores de $d[]$ e há no máximo dois valores de $d[]$ para vértices na fila.

Busca em largura: Lema 2

Lema 2

Suponha que $\langle v_1, v_2, \dots, v_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então:

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

Ou seja, os vértices são inseridos na fila em ordem não-decrescente dos valores de $d[]$ e há no máximo dois valores de $d[]$ para vértices na fila.

Prova. (por indução no número de iterações)

Busca em largura: Lema 2

Lema 2

Suponha que $\langle v_1, v_2, \dots, v_r \rangle$ seja a disposição da fila Q em alguma iteração do algoritmo. Então:

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

Ou seja, os vértices são inseridos na fila em ordem não-decrescente dos valores de $d[]$ e há no máximo dois valores de $d[]$ para vértices na fila.

Prova. (por indução no número de iterações)

Base: Antes da primeira iteração, $Q = \langle s \rangle$, e o lema vale.

Busca em largura: Lema 2

Hipótese: Na iteração i , a desigualdade $d[v_1] \leq \dots \leq d[v_r] \leq d[v_1] + 1$ vale.

Passo de indução: Considere a execução do laço.

- no início da iteração a fila era $\langle v_1, v_2, \dots, v_r \rangle$
- 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

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

- portanto

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

Busca em largura: correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo, e seja $s \in V$ 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 $v \in V$.

Busca em largura: correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo, e seja $s \in V$ 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 $v \in V$.

Prova. Primeiramente, o grafo $T = (V_\pi, E_\pi)$ cujas arestas são $(\pi[v], v)$, para todo v , $\pi[v] \neq \text{NIL}$, é uma árvore. Por quê?

Busca em largura: correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo, e seja $s \in V$ 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 $v \in V$.

Prova. Primeiramente, o grafo $T = (V_\pi, E_\pi)$ cujas arestas são $(\pi[v], v)$, para todo v , $\pi[v] \neq \text{NIL}$, é uma árvore. Por quê? – É conexa e $|E_\pi| = |V_\pi| - 1$.

Busca em largura: correção do algoritmo

Teorema

Seja $G = (V, E)$ um grafo, e seja $s \in V$ 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 $v \in V$.

Prova. Primeiramente, o grafo $T = (V_\pi, E_\pi)$ cujas arestas são $(\pi[v], v)$, para todo v , $\pi[v] \neq \text{NIL}$, é uma árvore. Por quê? – É conexa e $|E_\pi| = |V_\pi| - 1$.

Precisamos provar que $d[v] = \text{dist}(s, v)$ para todo $v \in V$.

Busca em largura: correção do algoritmo

$$d[v] = \text{dist}(s, v) \text{ para todo } v \in V$$

Se $\text{dist}(s, v) = \infty$, então $d[v] = \infty$ pelo Corolário 1.

Resta provar que se $\text{dist}(s, v) < \infty$, então $d[v] = \text{dist}(s, v)$.

Seja $v \in V$, e seja $\text{dist}(s, v) = k$. Vamos provar que $d[v] = k$ por **indução em k** .

Busca em largura: correção do algoritmo

$$d[v] = \text{dist}(s, v) \text{ para todo } v \in V$$

Se $\text{dist}(s, v) = \infty$, então $d[v] = \infty$ pelo Corolário 1.

Resta provar que se $\text{dist}(s, v) < \infty$, então $d[v] = \text{dist}(s, v)$.

Seja $v \in V$, e seja $\text{dist}(s, v) = k$. Vamos provar que $d[v] = k$ por **indução em k** .

Base: se $k = 0$, $v = s$, e o resultado vale.

Busca em largura: correção do algoritmo

$$d[v] = \text{dist}(s, v) \text{ para todo } v \in V$$

Se $\text{dist}(s, v) = \infty$, então $d[v] = \infty$ pelo Corolário 1.

Resta provar que se $\text{dist}(s, v) < \infty$, então $d[v] = \text{dist}(s, v)$.

Seja $v \in V$, e seja $\text{dist}(s, v) = k$. Vamos provar que $d[v] = k$ por **indução em k** .

Base: se $k = 0$, $v = s$, e o resultado vale.

Hipótese de indução: $d[u] = \text{dist}(s, u)$ para todo $u \in V$ com $\text{dist}(s, u) < k$.

Busca em largura: correção do algoritmo

$$d[v] = \text{dist}(s, v) \text{ para todo } v \in V$$

Se $\text{dist}(s, v) = \infty$, então $d[v] = \infty$ pelo Corolário 1.

Resta provar que se $\text{dist}(s, v) < \infty$, então $d[v] = \text{dist}(s, v)$.

Seja $v \in V$, e seja $\text{dist}(s, v) = k$. Vamos provar que $d[v] = k$ por **indução em k** .

Base: se $k = 0$, $v = s$, e o resultado vale.

Hipótese de indução: $d[u] = \text{dist}(s, u)$ para todo $u \in V$ com $\text{dist}(s, u) < k$.

Passo de indução: Seja v um vértice com $\text{dist}(s, v) = k$. Considere um caminho mínimo de s a v em G e seja u o vértice que antecede v neste caminho.

Resumo

- 1 Introdução e objetivo
- 2 Algoritmos de busca em grafos
- 3 Busca em largura, BFS
- 4 Síntese

Síntese

- Vimos as características gerais dos algoritmos de busca em grafos.
- Formulamos o algoritmo de busca em largura (**BFS**), e mostramos como esse algoritmo funciona.

Aplicações

A busca em largura é a **base** ou modelo de diversos algoritmos em grafos, entre eles alguns dos mais importantes:

- O algoritmo de Dijkstra.
- O algoritmo de Prim.

Dúvidas

Dúvidas?