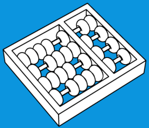
Buscas em grafos. Busca em largura

MC558 - Projeto e Análise de Algoritmos II

08/24

04



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

UNICAMP

“Um algoritmo precisa ser visto para ser acreditado.”

Donald Knuth

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?

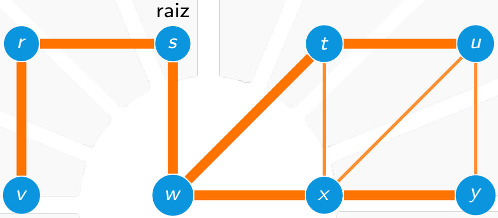
- A enraizamos em um **VÉRTICE DE ORIGEM** **s**.
- A representamos com um vetor π de pais.
- O pai de um vértice **v** é $\pi[\mathbf{v}]$.
- convencionamos que $\pi[\mathbf{s}] = \mathrm{NIL}$

Algumas propriedades:

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

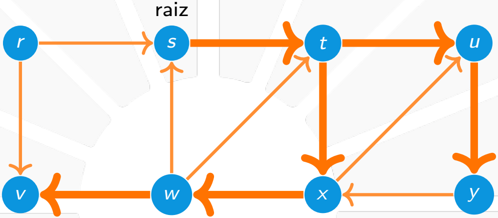
$$\mathbf{s} \to \cdots \to \pi[\pi[\pi[\mathbf{v}]]] \to \pi[\pi[\mathbf{v}]] \to \pi[\mathbf{v}] \to \mathbf{v}$$

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

```
se v = s
    imprima s
senão se π[v] = NIL
    imprima não existe caminho de s a v
senão
    PRINT-PATH(G, s, π[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 $\mathbf{v}$ a partir de $\mathbf{s}$ se há caminho de $\mathbf{s}$ a $\mathbf{v}$.
- Pode haver diversos caminhos entre $\mathbf{s}$ a $\mathbf{v}$.
- Queremos algum com o menor **COMPRIMENTO**.

A **DISTÂNCIA** de $\mathbf{s}$ a $\mathbf{v}$ é o comprimento de um caminho mais curto de $\mathbf{s}$ a $\mathbf{v}$:

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

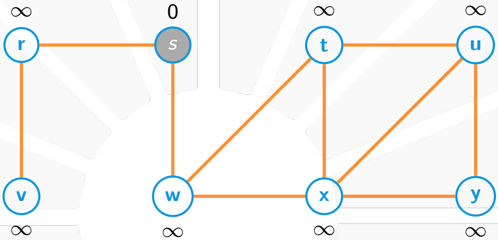
Vamos pintar o grafo durante a busca:

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:

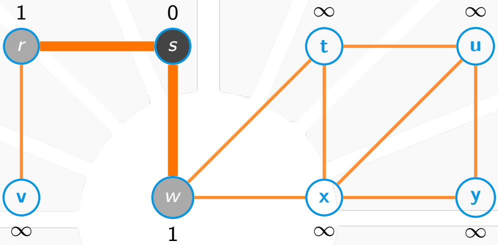
- Não é necessário em uma implementação.
- Facilita o entendimento do algoritmo.

Exemplo de busca em largura



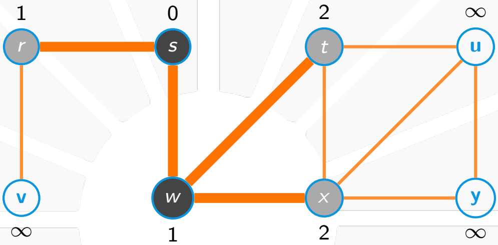
Q	s
distância	0

Exemplo de busca em largura



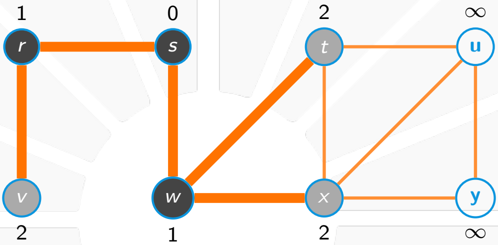
Q	**w**	**r**
distância	1	1

Exemplo de busca em largura



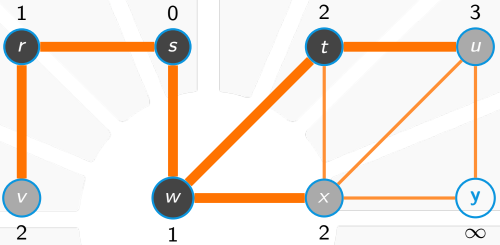
Q	**r**	**t**	**x**
distância	1	2	2

Exemplo de busca em largura



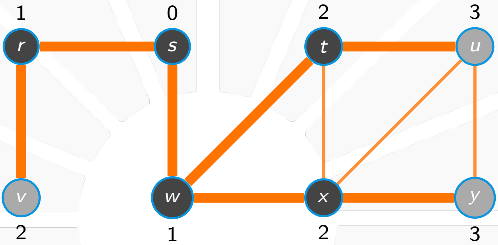
Q	t	x	v
distância	2	2	2

Exemplo de busca em largura



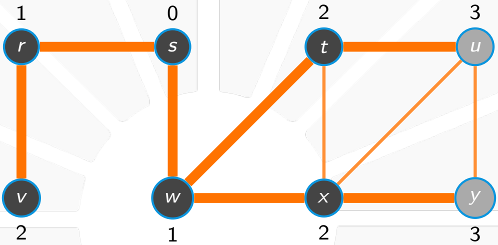
Q	**x**	**v**	**u**
distância	2	2	3

Exemplo de busca em largura



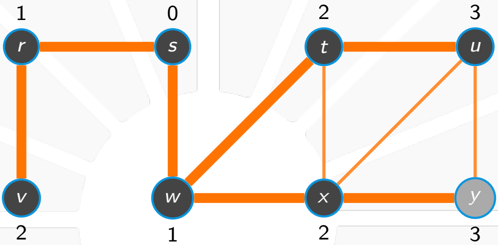
Q	**v**	**u**	**y**
distância	2	3	3

Exemplo de busca em largura



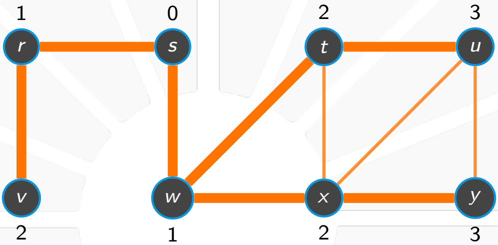
Q	**u**	**y**
distância	3	3

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

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

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

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:

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

- Portanto:

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

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 $\mathbf{v}$ 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 $\mathbf{u}$ com $\text{dist}(\mathbf{s}, \mathbf{u}) < k$:

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

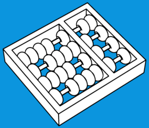
- Suponha (por contradição) que **v** seja preto:
  - Então **v** foi removido de Q antes de **u**.
  - Pelo Lema 2 temos que $d[\mathbf{v}] \leq d[\mathbf{u}] < k$.
  - Mas o Corolário 1 implica que $k = \text{dist}(\mathbf{s}, \mathbf{v}) \leq d[\mathbf{v}]$.
  - Logo, temos uma contradição e **v**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

Buscas em grafos. Busca em largura

MC558 - Projeto e Análise de Algoritmos II

04

08/24



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