Projeto e Análise de Algoritmos

Buscas em grafos: Busca em Largura

Atílio G. Luiz

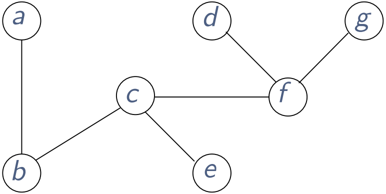
Primeiro Semestre de 2024

Definições

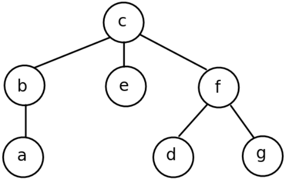
Definição: árvore enraizada

Def 1: Uma árvore é um grafo conexo e acíclico.

Def 2: Uma árvore enraizada é uma árvore com um vértice especial chamado raiz.



árvore T

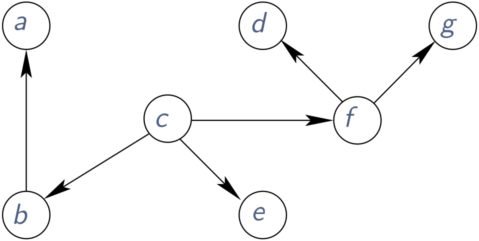


árvore T enraizada

Definição: árvore direcionada com raiz

Def.: Uma árvore direcionada com raiz r é um grafo direcionado acíclico $T = (V, E)$ tal que:

1. $d_{in}(r) = 0$,
2. $d_{in}(v) = 1$ para todo $v \in V \setminus \{r\}$.



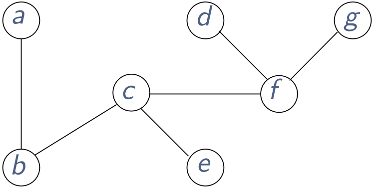
raiz c

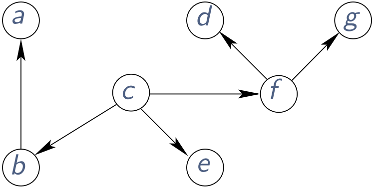
Representação de árvores

Vamos representar uma árvore enraizada com um vetor de predecessores π.

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

- usamos o símbolo NIL para indicar a ausência





raiz c

Buscas em grafo

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
- **um dos objetivos:** encontrar uma árvore geradora

Representação de árvores de busca

Como representar uma árvore de busca:

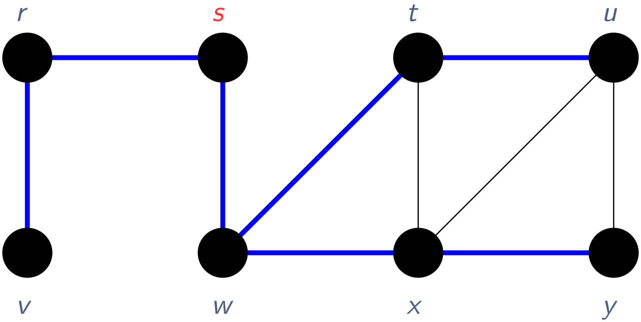
- vamos enraizá-la em um vértice de origem s
- representar a árvore com um vetor π de pais
- o pai de um vértice v é $\pi[v]$
- convencionamos que $\pi[s] = \text{NIL}$

Algumas propriedades

- existe aresta de $\pi[v]$ até v
- o caminho de s a v na árvore é

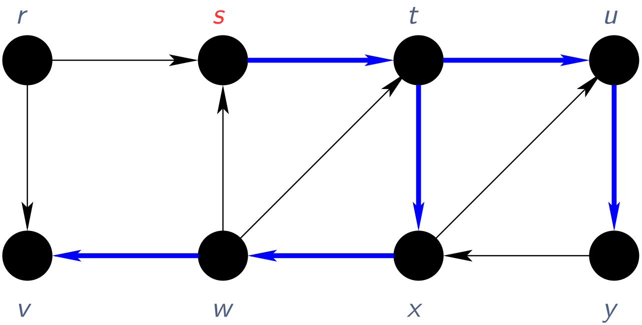
$$s \to \cdots \to \pi[\pi[\pi[v]]] \to \pi[\pi[v]] \to \pi[v] \to v$$

Exemplo com grafo não direcionado



v	r	s	t	u	v	w	x	y
$\pi[v]$	s	N	w	t	r	s	w	x

Exemplo com grafo direcionado



v	r	s	t	u	v	w	x	y
$\pi[v]$	N	N	s	t	w	x	t	u

Algoritmo: Caminho de s a v

Queremos um algoritmo que imprime o caminho de s a v (se existir) numa árvore de raiz s.

```
Print-Path(G, s, v)
    se v = s então
        imprima s
    senão se π[v] = NIL então
        imprima não existe caminho de s a v
    senão
        PRINT-PATH(G, s, π[v])
        imprima v
```

- gasta tempo linear no tamanho desse caminho

Busca Genérica

Busca Genérica — Algoritmo básico

- Seja G um grafo conexo em que todos os vértices se encontram desmarcados.
- No **passo inicial**, marca-se um vértice arbitrariamente escolhido.
- No **passo geral**, seleciona-se um vértice v já marcado e que seja incidente a alguma aresta (v, w) ainda não explorada.
  - A aresta (v, w) torna-se então explorada e, caso o vértice w não esteja marcado, marcamos w e fazemos $\pi[w] = v$.
- O processo termina quando todas as arestas de G tiverem sido exploradas.

Busca Genérica — Definições

- Quando a aresta (v, w) é selecionada a partir do vértice marcado v, diz-se que (v, w) foi **explorada**.
  - se o vértice w não estava marcado, dizemos que ele foi **alcançado** e ele é, então, marcado.
- Um vértice torna-se **finalizado** ou **explorado** quando todas as arestas incidentes ao mesmo tiverem sido exploradas.
- O vértice inicial é chamado **raiz** da busca.

Algoritmo de Busca Genérica

```
BuscaGenerica(G, s)
pré-condições: grafo G e vértice s ∈ V(G)
pós-condições: vetor de predecessores π
    seja π com π[v] = NIL para todo v ∈ V(G)
    marcar o vértice s
    Q = {s}
    enquanto Q ≠ ∅ faça
        seja u ∈ Q
        para cada v ∈ Adj[u] faça
            se v ainda não marcado faça
                marque v
                adicione v a Q
                π[v] = u
        remova u de Q
    devolva π
```

- Q : conjunto dos vértices alcançados e não finalizados.
- **invariante de laço:** no início de cada iteração do laço **enquanto** temos que: (i) todo vértice $u \in Q$ foi alcançado a partir de s e ainda não foi finalizado; e (ii) todo vértice $u \neq s$ com $\pi[u] \neq NIL$ foi alcançado a partir de s.

Tipos de Busca em Grafos

A busca em um grafo não é única. Durante o processo, há liberdade de escolha nas seguintes ocasiões:

- **No passo inicial:** seleção do vértice inicial da busca.
- **No passo geral:**
  - **vértice marcado:** seleção do vértice marcado v, a partir do qual se deseja explorar uma aresta (v, w) não explorada.
  - **aresta incidente:** seleção da aresta não explorada (v, w) incidente ao vértice marcado v.

Desta liberdade, surgem dois algoritmos principais:

1. Busca em largura (BFS) — do inglês breadth-first search
2. Busca em profundidade (DFS) — do inglês depth-first search

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 $\text{dist}(s, v)$
- se v não for alcançável, definimos $\text{dist}(s, v) = \infty$

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
- á árvore de busca fornece um caminho mais curto

Construindo uma árvore de busca

Ideia do algoritmo

1. percorremos os vértices usando uma fila Q
2. começamos com o vértice de origem s
3. para cada vizinho v do vértice atual u
   - se for a primeira vez que vemos v durante a busca, então adicionamos uma aresta (u, v) à árvore de busca, ou seja, fazemos $\pi[v] = u$
   - inserimos v na fila de processamento
4. repetimos o passo anterior com o primeiro vértice da fila

Cores dos vértices

Vamos pintar o grafo durante a busca

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

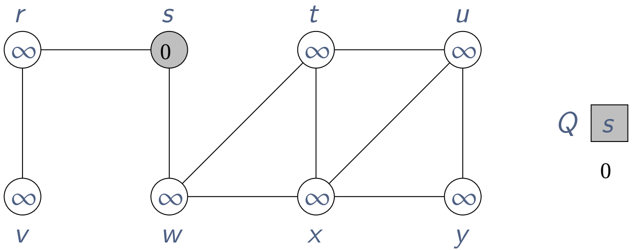
Observações

- Em uma implementação, podemos usar apenas duas cores: o branco e o cinza
- mas usamos três para facilitar o entendimento do algoritmo e das demonstrações

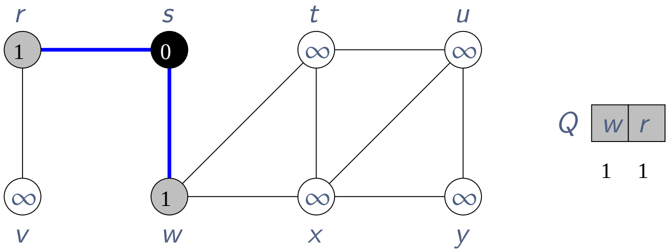
Cálculo de distâncias

- Será computado um vetor de distâncias d
- Para todo vértice $v \in V(G)$, a distância do vértice de origem s até v é dada por $d[u]$
- Por default, $d[s] = 0$
- A primeira vez que vemos um vértice $v \neq s$, ele é branco e foi descoberto na vizinhança de um vértice cinza u. Então fazemos $d[v] = d[u] + 1$.

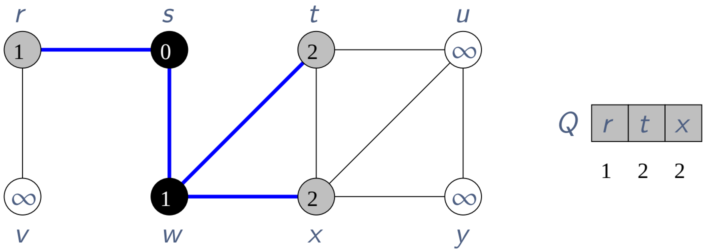
Exemplo de busca em largura



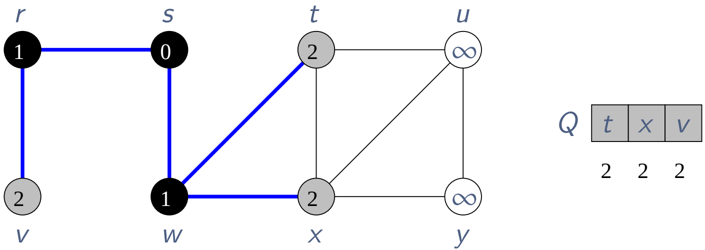
Exemplo de busca em largura



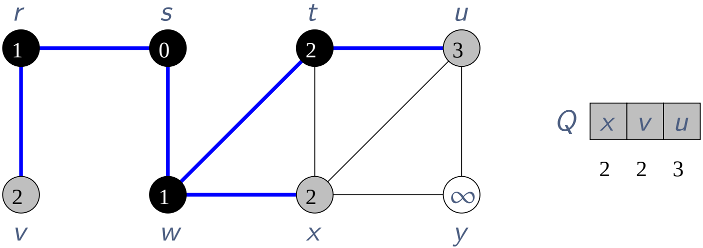
Exemplo de busca em largura



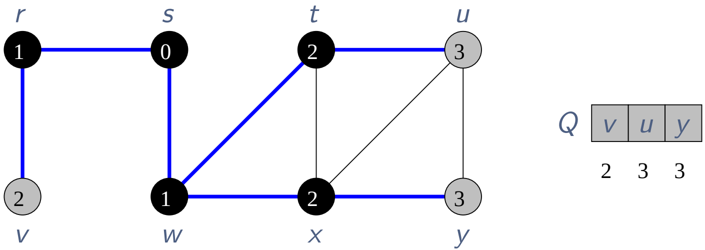
Exemplo de busca em largura



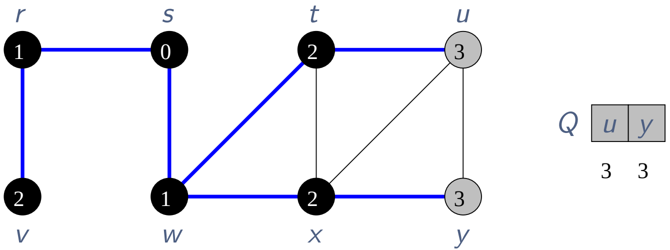
Exemplo de busca em largura



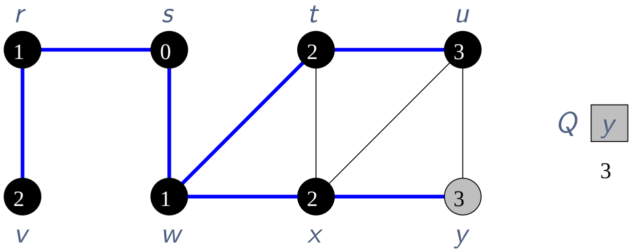
Exemplo de busca em largura



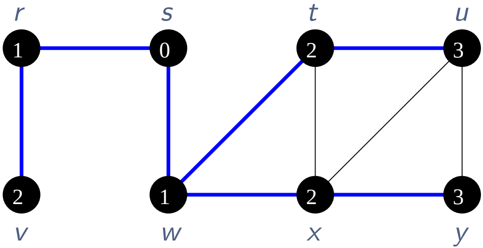
Exemplo de busca em largura



Exemplo de busca em largura



Exemplo de busca em largura



$Q \ \emptyset$

Algoritmo BFS

```
BFS(G, s)
    para cada u ∈ V(G) \ {s} faça
        cor[u] = branco
        d[u] = ∞
        π[u] = NIL
    cor[s] = cinza
    d[s] = 0
    π[s] = NIL
    Q = ∅
    ENQUEUE(Q, s)
    enquanto Q ≠ ∅ faça
        u = DEQUEUE(Q)
        para cada v ∈ Adj[u] faça
            se cor[v] = branco então
                cor[v] = cinza
                d[v] = d[u] + 1
                π[v] = u
                ENQUEUE(Q, v)
        cor[u] = preto
```

- representamos G com listas de adjacências

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 $v \in V(G)$.

Precisamos de três lemas

- Lema 0: π define uma árvore enraizada em s
- 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 0

Lema 0

Seja $G = (V, E)$ um grafo e s um vértice de G. Então, depois de executar BFS(G,s), temos que π define uma árvore T enraizada em s.

Demonstração:

- Devemos provar que T é conexo e acíclico.
- Todo vértice $v \in V - \{s\}$ que é alcançável a partir de s em G possui $\pi[v] \neq NIL$. Logo, o subgrafo T definido por π é conexo.
- Seja uma aresta (u, v) e suponha u alcançado antes de v. Como $(u, v) \in E(T)$ se e somente se v era branco quando foi alcançado, conclui-se que a adição de (u, v) a $E(T)$ se dá simultaneamente com a adição de v a T. Logo, T não contém ciclos. $\square$

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
- depois que executamos Enqueue pela primeira vez
  - T continha apenas s e valia $d[s] = 0$
  - como $d[s]$ nunca mais muda, isso completa a base

Prova do lema

Considere o instante em que enfileiramos v

- então v foi descoberto percorrendo os vizinhos de u
- mas u já havia sido enfileirado antes desse instante
- pela hipótese de indução
  1. existe um caminho de s a u em T
  2. esse caminho tem comprimento $d[u]$
- mais isso implica que
  1. há caminho de s a v em T, pois $\pi[v] = u$
  2. e esse caminho tem comprimento $d[v]$, pois $d[v] = d[u] + 1$
- e completamos a indução

Corolário 1

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

Lema 2

Lema 2

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

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

Demonstração:

- por indução no número de iterações do laço **enquanto**
- antes da primeira iteração, $Q = \langle s \rangle$ e o lema vale

Prova do lema

Considere a execução do laço

- no início da iteração a fila era $\langle v_1, v_2, \ldots, v_r \rangle$ e pela h.i. temos que $d[v_1] \leq d[v_2] \leq \cdots \leq d[v_r] \leq d[v_1] + 1$.
- na iteração, removemos v_1 e inserimos $v_{r+1}, \ldots, v_{r+t}$
- no final da iteração a fila será $\langle v_2, \ldots, v_r, v_{r+1}, \ldots, v_{r+t} \rangle$

Inserimos vizinhos de v_1

- se v_j é um vértice inserido, então $d[v_j] = d[v_1] + 1$
- pela hipótese de indução

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

- portanto

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

Prova 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, e
2. $d[v] = \text{dist}(s, v)$ para todo $v \in V(G)$.

Demonstração

- Pelo Lema 0, π define uma árvore enraizada em s e, pelo Lema 1, o caminho de s a v na árvore tem comprimento $d[v]$
- pelo Lema 1, $\text{dist}(s, v) = \infty$ se e somente se $d[v] = \infty$
- resta provar que se $\text{dist}(s, v) < \infty$, então $d[v] = \text{dist}(s, v)$

Prova do teorema (cont)

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

- iremos provar que $d[v] = k$ por indução em k

Caso Base:

- se $k = 0$, devemos ter $v = s$ e a afirmação vale.

Hipótese indutiva: Suponha que, para todo u com $\text{dist}(s, u) < k$, temos que $d[u] = \text{dist}(s, u)$

Passo indutivo: considere um caminho de s a v de comprimento k

- chame de u o vértice que antecede v nesse caminho
- daí $\text{dist}(s, u) = k - 1$ e portanto $d[u] = k - 1$

Prova do teorema (cont)

Considere o instante em que u foi removido da fila Q

- suponha por contradição que v seja preto
- daí v foi removido de Q antes de u
- então o Lema 2 implica que $d[v] \leq d[u] < k$
- mas o Corolário 1 implica que $k = \text{dist}(s, v) \leq d[v]$
- isso é uma contradição, então v não pode ser preto

Portanto, nesse instante, v era branco ou cinza

- se v era branco
  - v será inserido na fila nessa iteração
  - e teremos $d[v] = d[u] + 1 = k$
- se v era cinza
  - v já estava na fila nesse instante
  - então o Lema 2 implica $d[v] \leq d[u] + 1 = k$
  - como $k \leq d[v]$, temos $d[v] = k$
- em qualquer caso, concluímos a indução. $\blacksquare$

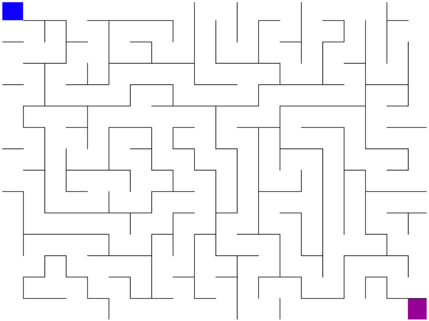
Aplicações de BFS

Algumas aplicações da BFS

- **Encontrar o caminho mais curto em um grafo não ponderado**
  - Em um grafo não ponderado, BFS pode ser usado para encontrar o caminho mais curto entre dois vértices. A distância é medida em termos de número de arestas.
- **Verificação de conectividade em um grafo**
  - BFS pode ser usado para verificar se um grafo é conexo. Se todos os vértices são alcançáveis a partir de um vértice inicial, o grafo é conexo.
- **Detecção de ciclos em grafos não direcionados**
  - BFS pode ajudar a detectar ciclos em grafos não direcionados. Se durante a busca encontrarmos um vértice já visitado que não é o pai do vértice atual, isso indica a presença de um ciclo.

Algumas aplicações da BFS

- **Resolução de quebra-cabeças de busca de caminho**
  - Muitos quebra-cabeças e jogos de tabuleiro que podem ser modelados como grafos podem ser resolvidos usando BFS para encontrar soluções, como o jogo de labirinto onde você precisa encontrar o caminho da entrada até a saída.



Algumas aplicações da BFS

- **Navegação e busca em redes sociais**
  - Em redes sociais modeladas como grafos, BFS pode ser usado para encontrar amigos de amigos ou pessoas a uma certa distância em termos de conexões.