Grafos

Busca em Grafos

- Algoritmos de busca em grafos são extremamente importantes para grafos.

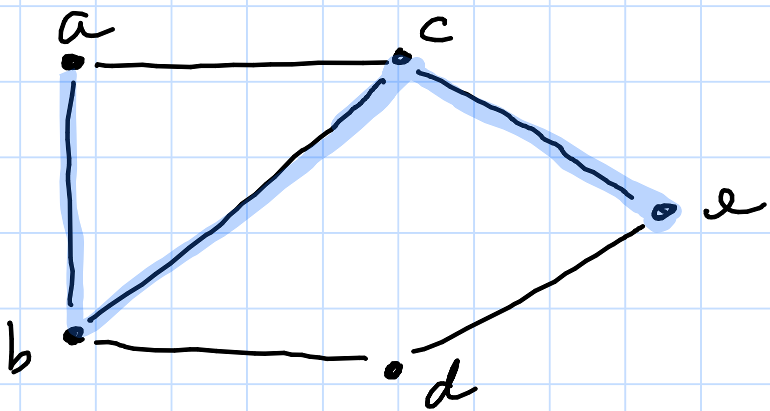
- Usamos algoritmos de busca para obter mais informações sobre a estrutura de um grafo:

  - Encontrar o caminho entre dois vértices
  - testar se o grafo é conexo
  - Calcular a distância entre dois vértices
  - Verificar se o grafo possui ciclos

- Servem de base para vários algoritmos importantes

uv- Caminho

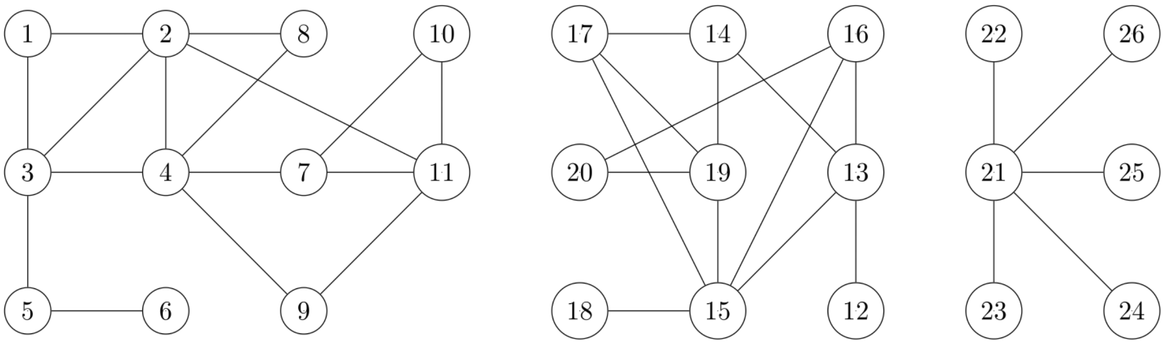
Dizemos que um caminho $P = w_0, w_1, w_2, \cdots, w_k$ é um <u>uv-Caminho</u> se $w_0 = u$ e $w_k = v$.



$P = a, b, c, e$ é um ae-caminho

Vértice Alcançável

Dizemos que um vértice v é <u>alcançável</u> a partir de um vértice u se existe um uv-caminho



Busca em Grafos

Muitos dos problemas e propriedades que surgem em grafos dependem de sabermos se existe um caminho entre dois vértices.

- Por isso é importante saber quais são os vértices alcançáveis a partir de um dado vértice.

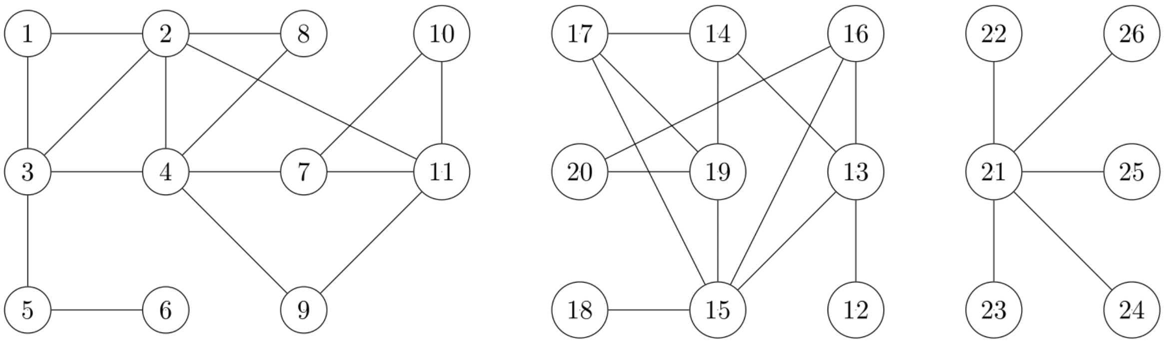
Busca em Grafos

- Vamos buscar os vértices alcançáveis a partir de s.

    - Chamaremos s de <u>raiz</u>

- IDEIA: Se u é alcançável a partir de s e $uv \in E(D)$, então v é alcançável a partir de s.

Exemplo



Obs: Note que esse procedimento gera uma árvore geradora da componente de s.

Lema Se T é uma árvore, então o grafo obtido a partir de T pela criação de um novo vértice u e pela adição de uma aresta uv, onde $v \in V(T)$, é uma árvore.

Busca em Grafos Genérica

Função Busca (G, s) // G é um grafo e $s \in V(G)$

Seja $T = (\{s\}, \emptyset)$

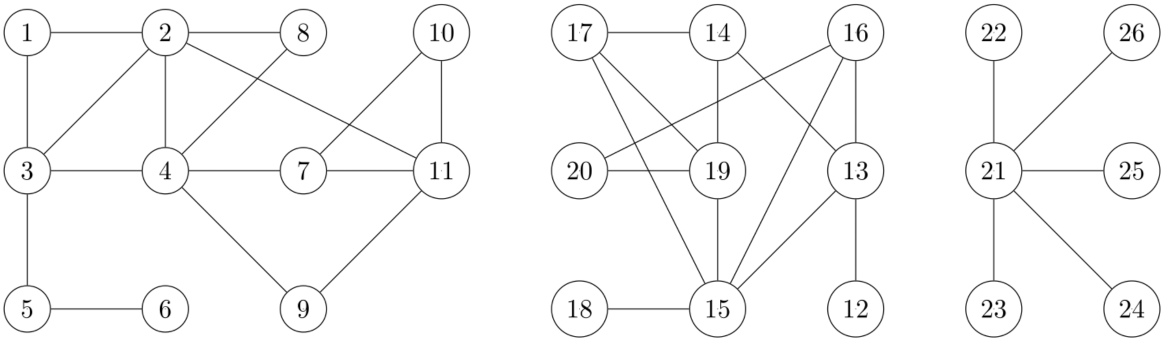
Enquanto $E_G(V(T)) \neq \emptyset$

Seja $uv \in E(V(T))$, onde $u \in V(T)$

$V(T) = V(T) \cup \{v\}$

$E(T) = E(T) \cup \{uv\}$

Devolva T



Busca(G, s) termina com uma árvore geradora da componente conexa do grafo que contém s

- Procedimentos assim costumam ser chamados de <u>busca</u> e a <u>árvore</u> resultante é chamada de <u>árvore de busca</u>

Busca em Grafos Genérica

A função busca tem várias formas de expandir a árvore

Função Busca (G, s) // G é um grafo e $s \in V(G)$

Seja $T = (\{s\}, \emptyset)$

Enquanto $E_G(V(T)) \neq \emptyset$

Seja $uv \in E(V(T))$, onde $u \in V(T)$

$V(T) = V(T) \cup \{v\}$

$E(T) = E(T) \cup \{uv\}$

Devolva T

Podemos ter políticas para escolher esse vértice

Busca em largura (BFS) Expande a árvore pela vizinhança dos vértices mais antigos em T

Busca em profundidade (DFS) Expande T pela vizinhança dos vértices mais novos em T

Representação de Árvores de Busca

- Vamos enraizá-la em um vértice de origem s
- Representar a árvore como um vetor pred[] de predecessores
- O pai, predecessor imediato, de um vértice v é pred[v],

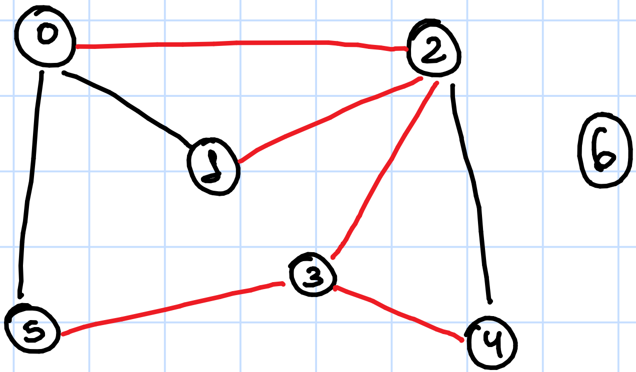
  se pred[v] $\neq -1$

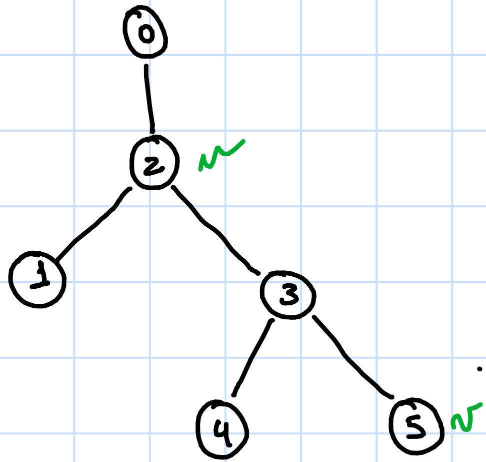
- Se pred[v] $= -1$, então v <u>não</u> está na árvore.
- Convencionamos que pred[s] $= s$
- A <u>árvore induzida</u> por pred[] é a árvore T tal

$$E(T) = \{ \{u, \text{pred}[u]\} : u \in V(T) \setminus s \ \text{ e } \ \text{pred}[u] \neq -1 \}$$

Exemplo

	0	1	2	3	4	5	6
pred	0	2	0	2	3	3	-1



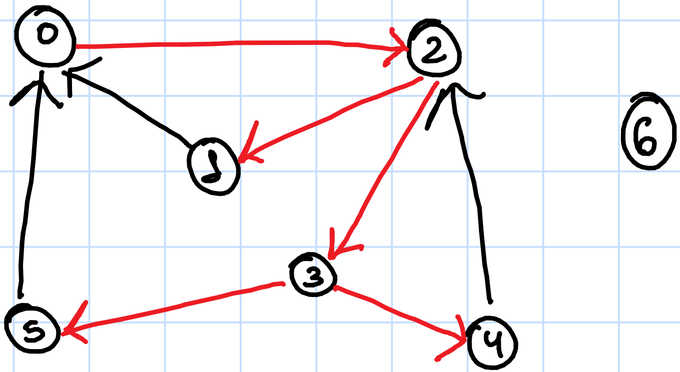


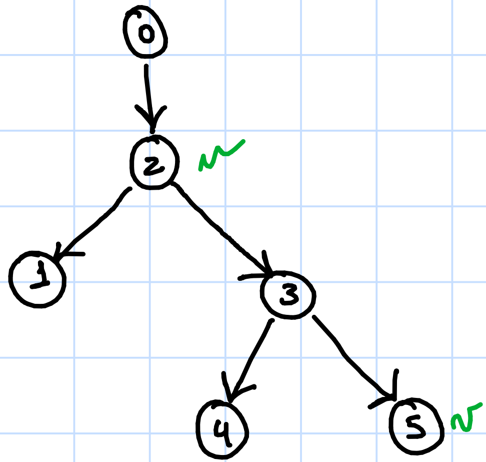
$$E(T) = \{\{u, \text{pred}[u]\} : u \in V(T) \setminus s \text{ e } \text{pred}[u] \neq -1\}$$

Dada uma árvore enraizada em s e um vértice v, dizemos que cada vértice no caminho de s a v é um ancestral de v

Exemplo

	0	1	2	3	4	5	6
pred	0	2	0	2	3	3	-1





$$E(T) = \{ (u, \text{pred}[u]) : u \in V(T) \setminus s \text{ e } \text{pred}[u] \neq -1 \}$$

Dada uma árvore enraizada em s e um vértice v, dizemos que cada vértice no caminho de s a v é um ancestral de v

Imprime-Caminho (pred , u)

    Se pred[u] = -1

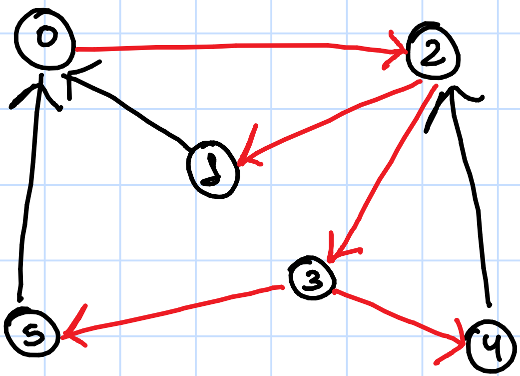
        Imprime "não há caminho"

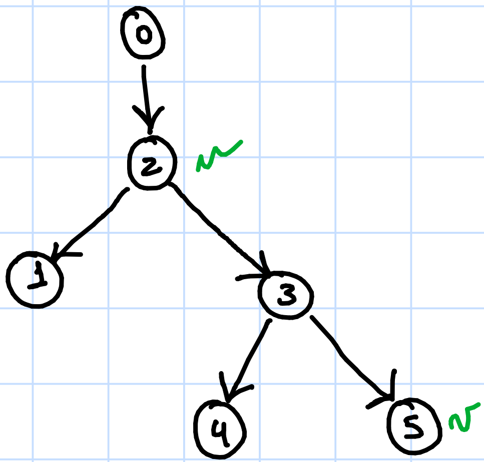
    Senão

        Se pred[u] ≠ u

            Imprime-Caminho (pred , pred[u])

        Imprime u





Complexidade: O(comp. do caminho) = O(n)

Busca em Grafos

Função Busca (G, s)

Sejam pred$[1..v(G)]$ e vis$[1..v(G)]$

Para todo $u \in V(G)$

pred$[u] \leftarrow -1$

vis$[u] \leftarrow$ False

pred$[s] \leftarrow s$

Vis$[s] \leftarrow$ True

Enquanto

BFS

Breadth-First Search

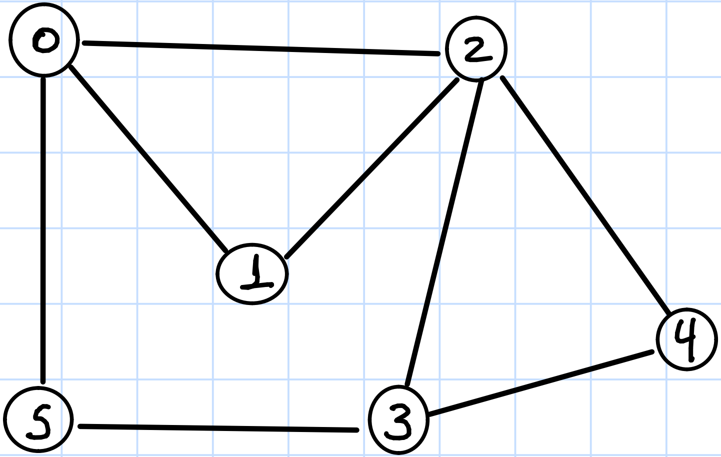
Busca em Largura

Busca em largura

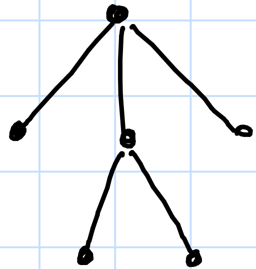
Ideia do Algoritmo

- Percorre os vértices usando uma <u>fila</u> Q
- Inicialmente a raiz é o único vértice na fila
- Em cada iteração
  - extraímos o primeiro elemento u da fila
  - adicionamos uma aresta $uv \in E_G(V(T))$ à árvore de busca T
  - Enfileiramos v
- Repetimos até a fila ficar vazia

Exemplo



pred



Vis

Fila


```
Função BFS(G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = -1
    vis[s] = T
    pred[s] = s
    Cria Fila F
    Enfileira(F, s)
    Enquanto |F| > 0
        u = Desenfileira(F)
        Para todo v ∈ N(u)
            Se vis[v] = F
                vis[v] = T
                pred[v] = u
                Enfileira(F, v)
```

Função BFS(G, s)

```
Função BFS(G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = -1
    vis[s] = T
    pred[s] = s
    Cria Fila F
    Enfileira(F, s)
    Enquanto |F| > 0
        u = Desenfileira(F)
        Para todo v ∈ N(u)
            Se vis[v] = F
                vis[v] = T
                pred[v] = u
                Enfileira(F, v)
```

Complexidade

Complexidade

(assumindo lista de prioridade)

Função BFS(G, s)

Para todo $u \in V(G)$

  $vis[u] = F$

  $pred[u] = -1$ $\Theta(V)$

$vis[s] = T$

$pred[s] = s$ $\Theta(1)$

Cria Fila F $\Theta(1)$

Enfileira(F, s) $\Theta(1)$

Enquanto $|F| > 0$ $\Theta(V)$

  u = Desenfileira(F) $\Theta(1)$ $\Theta(V)$

  Para todo $v \in N(u)$

    Se $vis[v] = F$

      $vis[v] = T$

      $pred[v] = u$

      Enfileira(F, v) $\Theta(1) \cdot \Theta(E)$

Tamanho de G = $|V(G)| + |E(G)|$

$T(G) = O(V+E)$

$\downarrow$

açúcar sintático p/ $O(|V(G)| + |E(G)|)$

Complexidade da BFS

- O tempo da inicialização (linhas 1-3) é $O(V)$

- Note que um vértice visitado continua visitado até o final da execução do algoritmo
  - um vértice é posto na fila somente qndo não foi visitado (linha 11). Na sequência ele é visitado e inserido na fila. Portanto cada vértice entra na fila no máximo uma vez
  - Cada operação da fila leva tempo $\Theta(1)$
  - O tempo gasto com a fila é $O(V)$

Complexidade da BFS

- Processamos cada vértice apenas uma vez (linha 9)
  - cada lista de adjacência é percorrida apenas uma vez
  - O tempo gasto com a lista (linha 10) é $O(E)$
    - O laço da linha 10, gasta $\Theta(1)$ para executar as linhas 11-13. Portanto essas linhas gastam $O(E)$.
      - Note que a linha 14 teve seu custo contabilizado quando contamos o tempo gasto com a fila
- Portanto, a BFS tem tempo $O(V+E)$.

Observações

- A complexidade da BFS é $O(\underline{V+E})$
  ↳ linear no tamanho da entrada

- A árvore induzida pelo vetor pred da BFS é chamada de <u>árvore de busca em largura</u>.

Busca em Largura

Esse algoritmo pode ser usado para:

- Encontrar componentes conexas
- Calcular a distância entre vértices
- Encontrar um caminho entre vértices
- Detectar ciclos
- Encontrar uma árvore geradora

BFS e Distância

BFS & Distância

- Podemos modificar a BFS de forma que ela compute a distância entre a raiz s e qualquer outro vértice no grafo

Dados dois vértices u e v em um grafo, a <u>distância</u> entre u e v, denotado por $dist(u,v)$, é o menor comprimento de um uv-caminho, se esse existir, ou ∞, caso contrário.

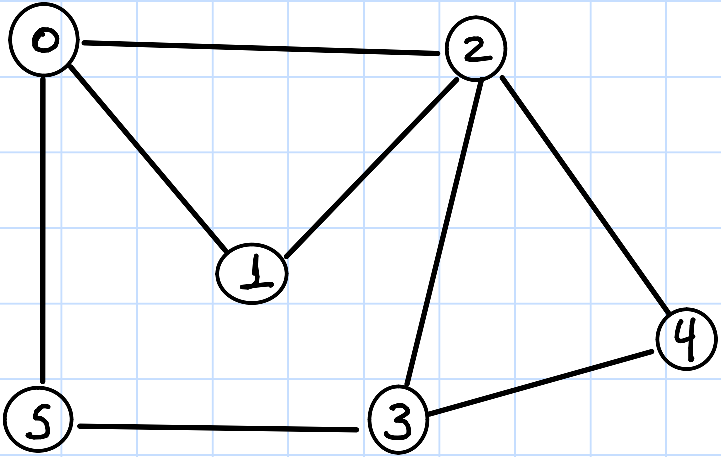
- Além de computar $dist(u,v)$, a BFS é capaz de encontrar esse caminho mais curto
- Single-Source-Shortest Path (SSSP) Problem
  - Problema do caminho mínimo de raiz única
  - computar o menor caminho de uma raiz s para todos os outros vértices.
- Adicionamos um vetor $d[1..v(G)]$ para armazenar $dist(s,u)$

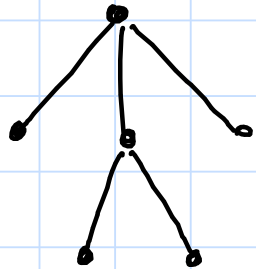
BFS para cálculo de Distância

```
Função BFS-dist(G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = -1
        d[u] = ∞
    vis[s] = T
    pred[s] = s
    d[s] = 0
    Cria Fila F
    Enfileira(F, s)
    Enquanto |F| > 0
        u = Desenfileira(F)
        Para todo v ∈ N(u)
            Se vis[v] = F
                vis[v] = T
                pred[v] = u
                d[v] = d[u] + 1
                Enfileira(F, v)
```

Exemplo

pred





Vis

d

Fila

Teo Seja $G=(V,E)$ um grafo e s um vértice de G. Então, depois de executar BFS-dist(G,s), temos

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

Antes de provarmos esse resultado, precisamos de dois lemas auxiliares

Correção da BFS-dist

Objetivo

Mostrar que:

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

2) o vetor pred[1..V] induz uma árvore T com raiz s para todo vértice alcançável por s

3) Se P é um su-caminho em T, então $e(P) = d[u]$

```
Função BFS-dist(G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = -1
        d[u] = ∞
    vis[s] = T
    pred[s] = s
    d[s] = 0
    Cria Fila F
    Enfileira(F, s)
    Enquanto |F| > 0
        u = Desenfileira(F)
        Para todo v ∈ N(u)
            Se vis[v] = F
                vis[v] = T
                Pred[v] = u
                d[v] = d[u] + 1
                Enfileira(F, v)
```

Obs: d[u] e pred[u] nunca mudam após u ser inserido na fila

Obs 2: Se u é enfileirado, então $d[u] < \infty$

Lema 1 Seja T a árvore induzida por pred[ ]. Se $d[u] < \infty$, então o caminho de s a u em T tem comprimento $d[u]$.

Demonstração

- A prova segue por indução no número l de operações Enfileira

- <u>Base $l = 1$</u>

  - $d[s] = 0$ e $d[u] = \infty$ $\forall u \in V(G) \neq s$

Passo $\ell > 1$

- Seja T' a árvore induzida pelo vetor pred[ ] qndo fizemos a $(\ell-1)$ésima operação Enfileira
- Por hipótese de indução

  Se $d[x] < \infty$, então $dist_{T'}(s,x) = d[x]$

- no momento em que fizemos a ℓ-ésima operação Enfileira, estávamos percorrendo a vizinhança de um vértice u e encontramos um vértice $v \in N_G(u)$ tal que $vis[v] =$ Falso

- Como u foi desenfileirado, temos $d[u] < \infty$
  - Portanto $d[u] = dist_{T'}(s,u)$
  - Seja P' o caminho em T' de s à u

- Na sequência o algoritmo faz $pred[v] = u$ e $d[v] = d[u] + 1$.
- Seja T a árvore induzida por esse novo vetor pred.
  - Claramente $T = T' + uv$.
  - note que $P = P' + uv$ é o caminho de s à v em T
  - Assim, $dist_T(s, v) = dist_{T'}(s, u) + 1$

$$
\begin{aligned}
&= d[u] + 1 \\
&= d[v]
\end{aligned}
$$

- Portanto, a prova segue. $\square$

Corolário Durante a execução de BFS vale que

$d[v] \geq dist(s,v)$ para todo $v \in V(G)$.

Lema Sejam G um grafo e $s \in V(G)$. Na execução de BFS-dist(G, s), se u e v são dois vértices que estão na fila e u entrou na fila antes de v, então

$$d[u] \leq d[v] \leq d[u] + 1$$

$$F = \langle \underset{\substack{\uparrow \\ u}}{\overset{\substack{\text{início} \\ \downarrow}}{x_1}}, x_2, x_3, x_4, x_5, \underset{\substack{\uparrow \\ v}}{\overset{\substack{\text{Fim} \\ \downarrow}}{x_6}} \rangle$$

$$\underset{\substack{\| \\ 8}}{d[x_1]} \leq d[x_2] \leq d[x_3] \leq \cdots \leq d[x_6] \leq \underset{9}{\underline{d[x_1] + 1}}$$

Note que temos no máximo duas distâncias na fila

Lema Sejam G um grafo e $s \in V(G)$. Na execução de BFS-dist (G, s), se u e v são dois vértices que estão na fila e u entrou na fila antes de v, então

$$d[u] \leq d[v] \leq d[u] + 1$$

Demonstração

- Por indução no número de iterações ℓ do laço enquanto
- Vamos provar que a propriedade é verdadeira no início da ℓ-ésima iteração.
- Se $\ell = 1$, então $F = \langle s \rangle$ e o resultado segue.

- Assuma que $\ell > 1$
- suponha que no início da $(\ell-1)$-ésima iteração do laço, temos que

$$F = \langle x_1, x_2, x_3, \dots, x_k \rangle \qquad (A)$$

e que, para qualquer par de elementos x_i e x_j em F, com $i < j$, vale que

x_i entrou antes

$$d[x_i] \leq d[x_j] \leq d[x_i] + 1 \qquad (C)$$

- Em particular

$$d[x_1] \leq d[x_j] \leq d[x_1] + 1 \quad \forall\, 2 \leq j \leq k \qquad (D)$$

- Vamos analisar o que acontece na $(\ell-1)$-ésima iteração

- O algoritmo
  - remove x_1 de F
  - adiciona os vizinhos não visitados $w_1, w_2, \ldots w_n$ de x_1 a F
  - Fazemos $d[w_i] = d[x_1] + 1 \quad \forall\, i = 1, \ldots, n \quad (E)$
- Portanto, no início da l-ésima iteração temos que $F = \langle x_2, x_3, \ldots, x_k, w_1, w_2, \ldots, w_n \rangle$
- Sejam a e b dois elementos de F
- Caso 1: $a = x_i$, $b = x_j$ e $i < j$

Por (C), $\quad d[x_i] \leq d[x_j] \leq d[x_i] + 1$

- Caso 2: $a = w_i$, $b = w_j$ e $i < j$

Por (E) $\quad d[w_i] = d[w_j] = d[x_1] + 1$

$$\boxed{\begin{aligned} d[w_i] &= d[x_1] + 1 \\ &= d[w_j] < d[x_1] + 2 \\ &= d[w_i] + 1 \end{aligned}}$$

- Caso 3: $a = x_i$ e $b = w_j$

Por (D) e (E), (C) $\quad d[x_i] \leq d[x_1] + 1 = d[w_j] = d[x_1] + 1 \leq d[x_i] + 1$

Teo Sejam G um grafo e $s \in V(G)$. Ao fim de BFS-dist(G, s), para todo $v \in V(G)$, vale que

$$d[v] = \mathrm{dist}_G(s, v)$$

Teo Sejam G um grafo e $s \in V(G)$. Ao fim de BFS-dist(G, s), para todo $v \in V(G)$, vale que
$$d[v] = dist_G(s, v).$$

Demonstração

- Pelo Corolário, $d[v] \geq dist(s,v)$ para todo $v \in V(G)$

> **Corolário** Durante a execução de BFS vale que $d[v] \geq dist(s,v)$ para todo $v \in V(G)$.

- Agora vamos mostrar que $d[v] \leq dist(s,v)$ $\forall v \in V(G)$

$$dist(s,v) \leq d[v] \leq dist(s,v)$$

- Suponha, para fins de contradição, que existe um vértice u tal que $d[u] > dist_G(s,u)$
- Dentre todos os vértices u tais que $d[u] > dist(s,u)$, seja v um com a menor distância p/ s.

$$dist(s,v) = \min \{ dist(s,u) : u \in V(G) \text{ e } d[u] > dist(s,u) \}$$

- Seja $P = s, \ldots, u, v$ um caminho mais curto de s à v em G

$$e(P) = dist_G(s,v)$$

- note que $dist(s,v) = dist(s,u) + 1$ (A)
- Pela escolha de v e por (A), temos que $d[u] = dist(s,u)$
- Assim, $d[v] > dist(s,v) = dist(s,u) + 1 = d[u] + 1$ (B)
- Considere o momento em que BFS-dist(G,s) remove o vértice u de F.
   - Dois casos: $vis[v] = $ Verdadeiro ou $vis[v] = $ Falso

- Caso vis[v] = Falso

- O algoritmo faz
  - vis[v] = Verdadeiro
  - $d[v] = d[u]+1$ (C)
  - v é inserido na fila

$$\boxed{d[v] > dist(s,v) = dist(s,u) + 1 = d[u]+1 \quad (B)}$$

- Por (B) e (C),

$$d[v] > d[u]+1 = d[v]$$

- Caso $vis[v]$ = Verdadeiro
  - Um vizinho $w \neq u$ "visitou" (qndo estávamos percorrendo $N(w)$, fizemos $vis[v] = T$) v
  - Portanto, $d[v] = d[w] + 1$ (D)

    e w saiu da fila antes de u
  - Pelo Lema, $d[w] \leqslant d[u]$ (E)

> Lema Sejam G um grafo e $s \in V(G)$. Na execução de BFS-dist (G, s), se u e v são dois vértices que estão na fila e u entrou na fila antes de v, então
> $$d[u] \leqslant d[v] \leqslant d[u] + 1$$

- Por (B), (D) e (E)

$$d[v] > d[u] + 1 \geqslant d[w] + 1 = d[v]$$

$\square$