Busca

em

Grafos

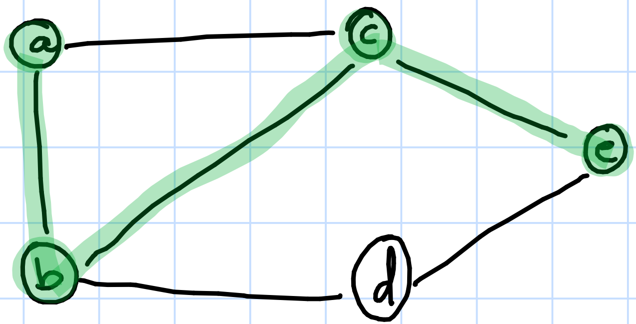
Busca em Grafos

- Usamos algoritmos de busca para obter mais informações sobre a estrutura de um grafo
  - Encontrar um caminho entre dois vértices.
  - testar se o grafo é conexo.
  - calcular a distância entre dois vértices.
  - verificar se o grafo possui ciclo.

- Servem de base para vários algoritmos importantes.

uv - caminho

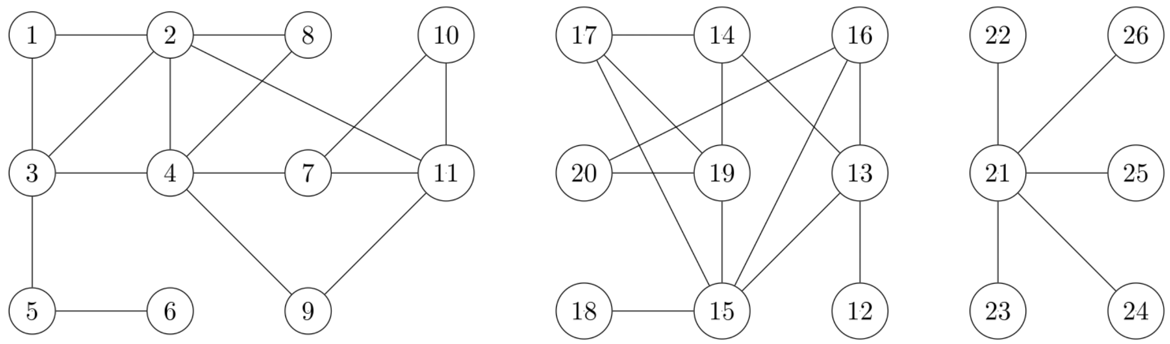
Dizemos que um caminho $P = w_0, w_1, w_2, \ldots, w_k$ é um uv-caminho se $w_0 = u$ e $w_k = v$



Exemplo de 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

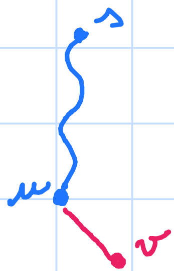
- Muitos dos problemas e propriedades que surgem em grafos dependem de sabermos se existe um caminho entre dois vértices

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

Busca em Grafos

- Vamos buscar os vértices alcançáveis a partir de um vértice s
  - s será a <u>raiz</u> da busca.

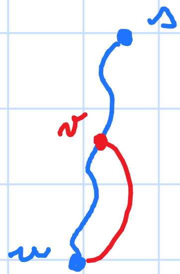
CASO 1



Proposição

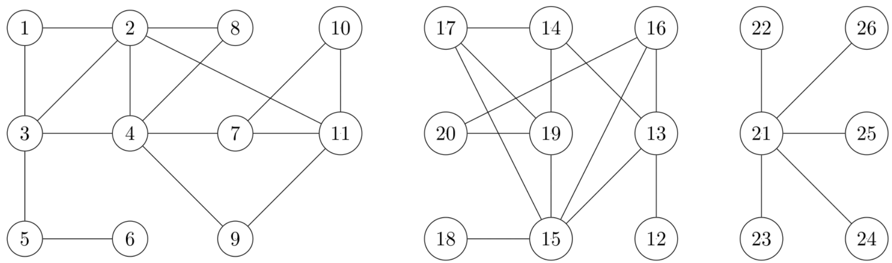
Se u é alcançável a partir de s e $uv \in E(G)$, então v é alcançável a partir de s.

CASO 2



Busca em Grafos

```
Função BUSCA(G, s)
  ▷ onde G é um grafo e s ∈ V(G)
  Seja T = ({s}, ∅)
  Enquanto E_G(V(T)) ≠ ∅ faça
      Seja uv ∈ E_G(V(T)), onde u ∈ V(T)
      V(T) = V(T) ∪ {v}
      E(T) = E(T) ∪ {uv}
  Devolva T
```



Lembrete: marcar a ordem em que os vértices estão entrando

Busca em Grafos

Obs Note que esse procedimento gera uma árvore

- Árvore geradora da componente H que contém s.

Lema Se T é uma árvore, então $T + uv$ é uma árvore, onde $u \in V(T)$ e v é um vértice novo.

Demonstração (Exercício)

Busca em Grafos

- 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>

  - Dizemos que essa árvore é <u>enraizada</u> em s

Busca em Grafos

A função Busca(G, s) possui várias opções de como estender a árvore.

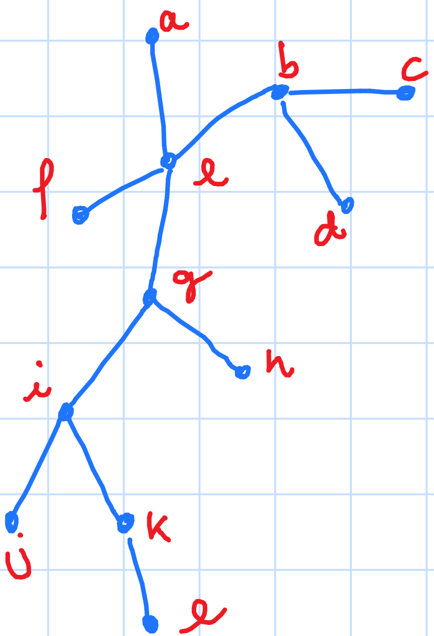
Dependendo do critério utilizado podemos ter informações adicionais.

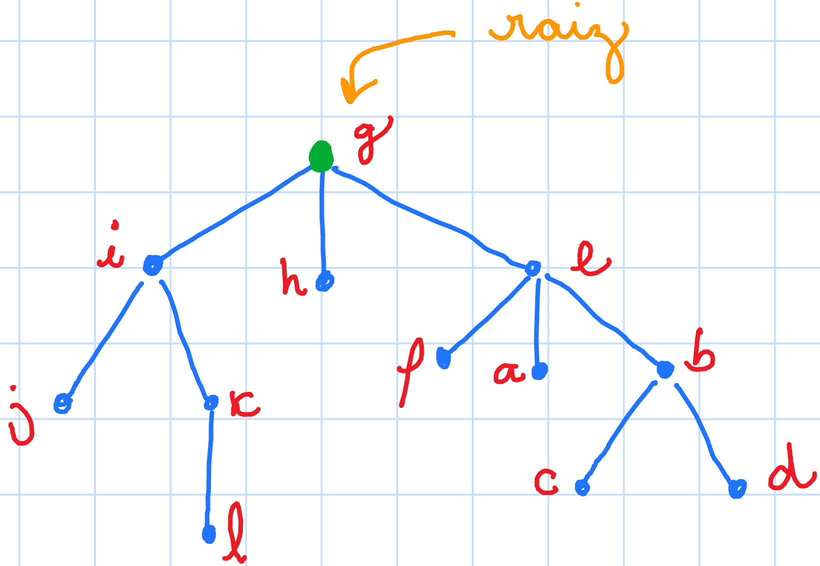
- Busca em largura (BFS - Breadth-first Search)
  - expande o vértice com o menor tempo de descoberta
  - primeiro a entrar, primeiro a sair

- Busca em profundidade (DFS - Depth-first Search)
  - expande o vértice com o maior tempo de descoberta
  - último a entrar, primeiro a sair

Árvore Enraizada

Uma árvore **enraizada** é uma árvore T e um vértice $u \in V(T)$ chamado de **raiz**

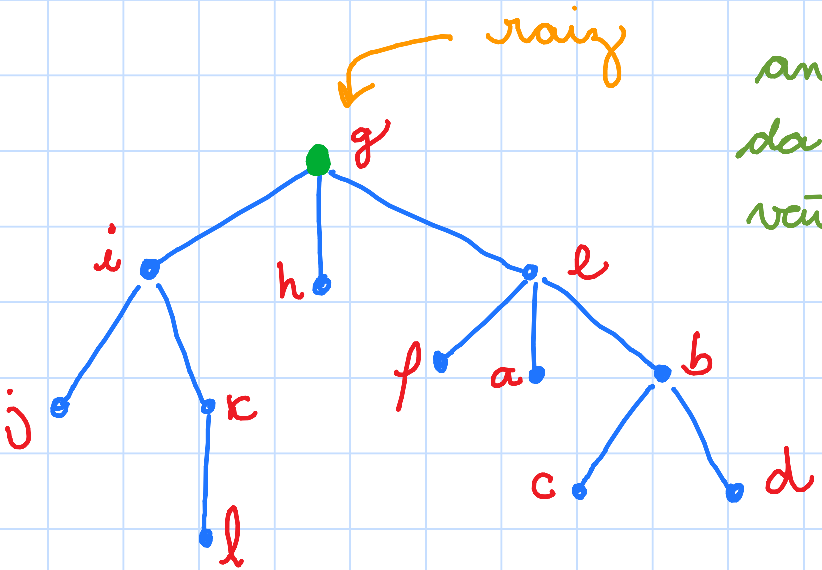




geralmente desenhamos uma árvore enraizada por "camadas"

Árvore Enraizada

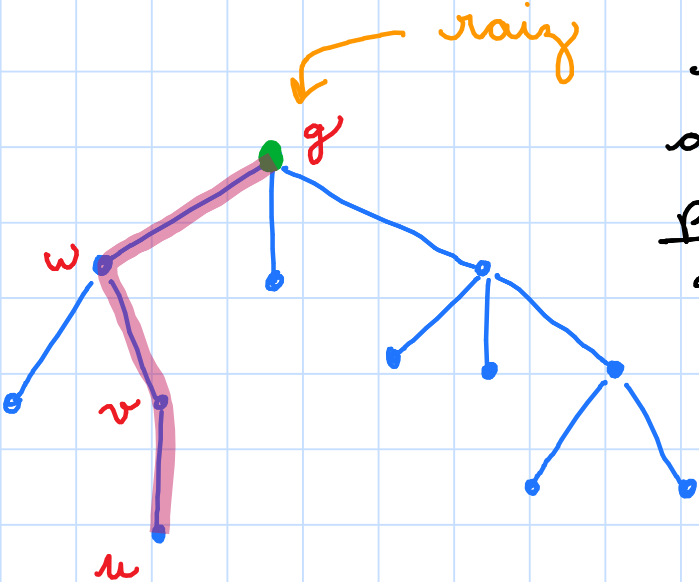
Uma árvore <u>enraizada</u> é uma árvore T e um vértice $u \in V(T)$ chamado de <u>raiz</u>



analisamos a árvore enraizada como caminhos que vão / saem da raiz.

Árvore Enraizada

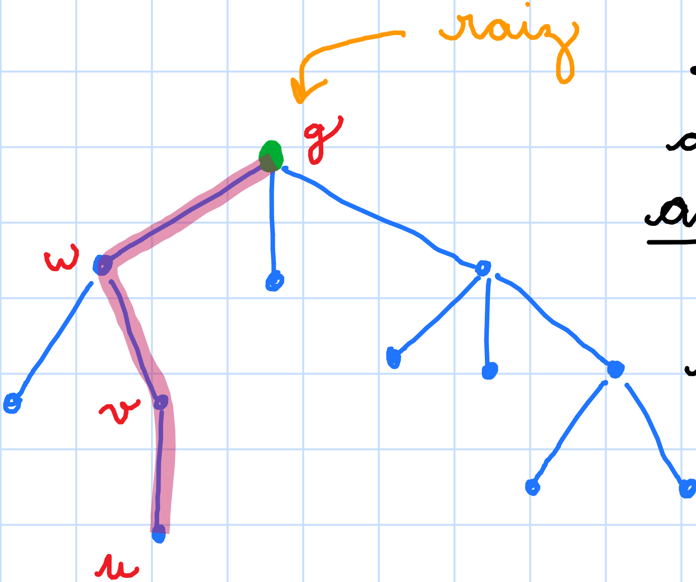
Uma árvore <u>enraizada</u> é uma árvore T e um vértice $u \in V(T)$ chamado de <u>raiz</u>



dado um vértice $u \in V(T)$, dizemos que $v \in V(T)$ é o <u>pai de u</u> em T se v precede imediatamente o vértice u no caminho da raiz s até u. Além disso, também dizemos que u é o <u>filho</u> de v.

Árvore Enraizada

Uma árvore enraizada é uma árvore T e um vértice $u \in V(T)$ chamado de raiz



dado um vértice $u \in V(T)$, dizemos que $w \in V(T)$ é um ancestral de u se w precede u no caminho da raiz s até u. Além disso, também dizemos que u é um descendente de w.

Árvore Enraizada

- Na prática, não construímos um grafo para representar uma árvore enraizada, usamos um vetor
  - pred[V(T)]

notação para dizer que o vetor é indexado pelos elementos do conjunto

- Para todo vértice u diferente da raiz

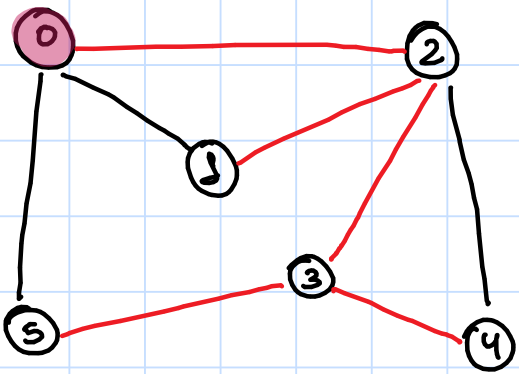
$$pred[u] = v \iff v \text{ é o pai de } u$$

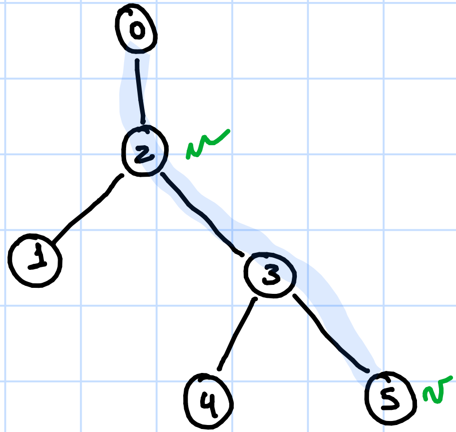
- Se u é a raiz, então

$$pred[u] = u$$

Árvore Enraizada

- Seja T uma árvore enraizada em 0





pred

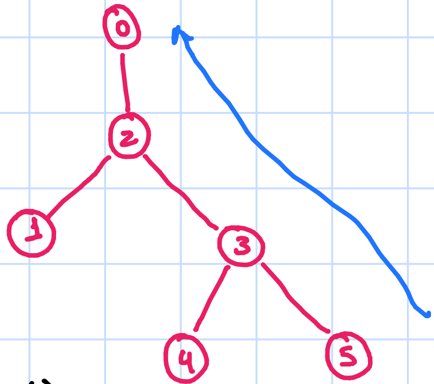
0	1	2	3	4	5
0	2	0	2	3	3

$$E(T) = \Big\{ \{u, \mathrm{pred}(u)\} : u \in V(T) \setminus s \Big\}$$

Árvore Enraizada: Recuperando o Caminho

Função Imprime-caminho (u, pred)

    Se pred[u] = NIL
        imprime u
    Senão
        Imprime-Caminho (pred[u], pred)
        Imprime u

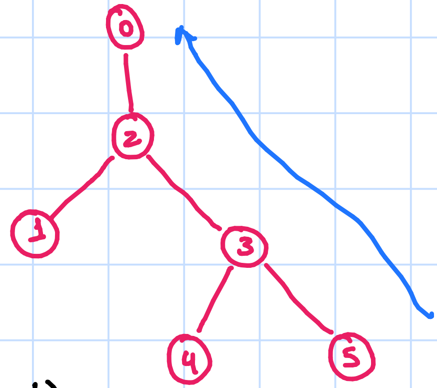


pred

0	1	2	3	4	5
0	2	0	2	3	3

Árvore Enraizada: Recuperando o Caminho

```
Função Imprime-caminho (u, pred)
    Se pred[u] = NIL
        imprime u
    Senão
        Imprime-Caminho (pred[u], pred)
        Imprime u
```



pred

0	1	2	3	4	5
0	2	0	2	3	3

Complexidade: $O(\text{comprimento do caminho}) = O(V)$

Busca em Grafos

Função Busca(G, s)

Para todo $v \in V(G)$ faça

$\quad vis[v] = F$

$\quad pred[v] = NULL$

$vis[s] = T$

$pred[s] = s$

Enquanto $\{uv \in E(G) : vis[u] = T$ e $vis[v] = F\} \neq \emptyset$ faça

$\quad$ Seja $xy \in E(G)$, onde $vis[x] = T$ e $vis[y] = F$

$\quad vis[y] = T$

$\quad pred[y] = x$

Devolve pred

Função Busca(G, s)

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

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

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

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

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

Devolve T

BFS

Breadth-First Search

Busca em Largura

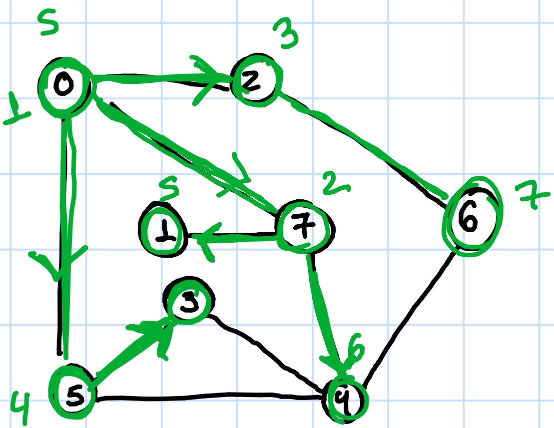
Busca em Largura (BFS)

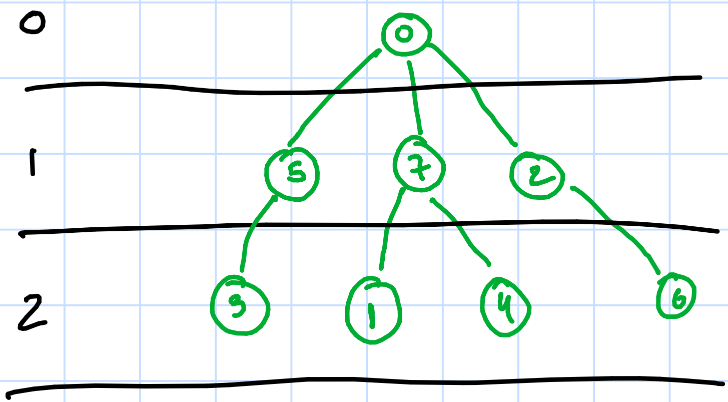
A ideia é expandir a árvore pela vizinhança do vértice que entra primeiro na árvore

"primeiro a entrar, primeiro a sair"

Processa os vértices por "camada"

↳ distância da raiz





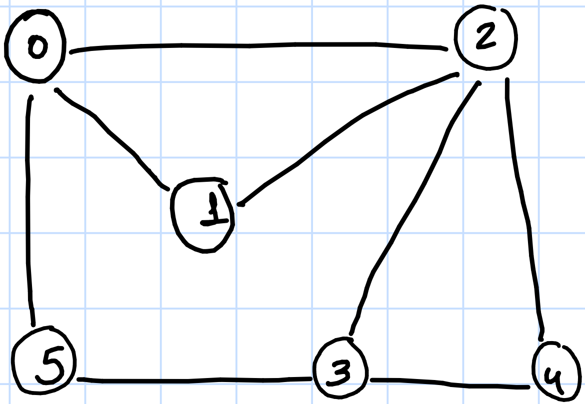
Busca em Largura (BFS)

Esse algoritmo pode ser usado para:

- encontrar componentes conexas
- calcular distância entre vértices
- encontrar caminhos entre vértices
- detectar ciclos
- verificar se um grafo é bipartido (e produzir uma em caso afirmativo)
- encontrar uma árvore geradora da componente contendo a raiz.

Execução da BFS

Fila



vis

0	1	2	3	4	5

pred

0	1	2	3	4	5

BFS

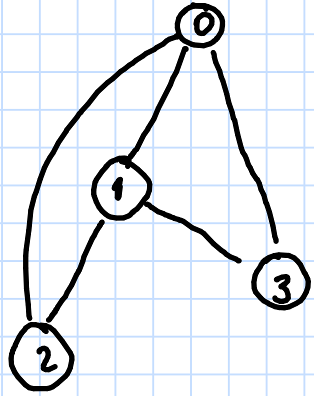
▶ G é um grafo e $s \in V(G)$

```
Função Busca-Largura (G, s)
    para todo u ∈ V(G)
    |  vis[u] = F
    |  pred[u] = NULL
    pred[s] = s
    vis[s] = T
    cria fila F
    Enfileira (F, s)

    Enquanto |F| > 0
    |  u = Desenfileira (F)
    |  para todo vértice v ∈ N(u)
    |  |  se vis[v] == F
    |  |  |  vis[v] = T
    |  |  |  pred[v] = u
    |  |  |  Enfileira (F, v)
    Devolve pred
```

BFS

```
Função Busca-Largura (G, s)
    para todo u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    vis[s] = T
    cria fila F
    Enfileira (F, s)
    Enquanto |F| > 0
        u = Desenfileira (F)
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                Enfileira (F, v)
    Devolva pred
```



	0	1	2	3
vis				
pred				

Fila:

BFS (Complexidade)

```
Função Busca-Largura (G, s)
    para todo u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    vis[s] = T
    cria fila F
    Enfileira (F, s)
    Enquanto |F| > 0
        u = Desenfileira (F)
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                Enfileira (F, v)
    Devolva pred
```

BFS (Complexidade)

```
Função Busca-Largura (G, s)
    Para todo u ∈ V(G)                              A = O(V)
    |  vis[u] = F
    |  pred[u] = NULL

    pred[s] = s                                     B = O(1)
    vis[s] = T
    cria fila F
    Enfileira (F, s)

    Enquanto |F| > 0                                C = O(V)
    |  u = Desenfileira (F)
    |  para todo vértice v ∈ N(u)                   D
    |  |  se vis[v] == F
    |  |  |  vis[v] = T
    |  |  |  pred[v] = u
    |  |  |  Enfileira (F, v)                  O(1)

    Devolva pred                                    E = O(1)
```

- G é um grafo e $s \in V(G)$

- Podemos implementar as operações de fila Enfileira, Desenfileira e Cria Fila em $O(1)$ usando uma lista ligada

- Um vértice visitado nunca deixa de ser visitado e apenas vértices que não foram visitados (vis[u] == F) são adicionados a fila. Portanto cada vértice é enfileirado no máximo uma vez e o laço Enquanto executa no máximo $O(V)$

D: $O(V)$ se matriz adj

$O(d(u))$ se lista adj

BFS (Complexidade)

Função Busca-Largura (G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
$A = O(V)$

    pred[s] = s
    vis[s] = T
    cria fila F
    Enfileira (F, s)
$B = O(1)$

    Enquanto |F| > 0
        u = Desenfileira (F)
$C = O(V)$
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                Enfileira (F, v)
$O(1)$
D
    Devolva pred
$E = O(1)$

- G é um grafo e $s \in V(G)$

- Vamos analisar o custo de D ao longo de todas as iterações do programa. Essa análise é chamada de análise agregada.

- Se G for implementada como matriz de adj. então

$$D = O(V^2)$$

- Se G for implementada como lista de adj

$$D = \sum_{u \in V(G)} d(u) = 2|E(G)| = O(E)$$

$O(V)$ se matriz adj
$O(d(u))$ se lista adj

BFS (Complexidade)

Função Busca-Largura (G, s)

- G é um grafo e $s \in V(G)$

```
Função Busca-Largura (G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    vis[s] = T
    cria fila F
    Enfileira (F, s)
    Enquanto |F| > 0
        u = Desenfileira (F)
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                Enfileira (F, v)
    Devolva pred
```

$A = O(V)$

$B = O(1)$

$C = O(V)$

D

$O(1)$

$E = O(1)$

$O(V)$ se matriz adj

$O(d(u))$ se lista adj

$T(G) = A + B + C + D + E$

$= O(V) + O(1) + O(V) + D + O(1)$

$= O(V + 1 + V + 1) + D$

$= O(V) + D$

BFS (Complexidade)

```
Função Busca-Largura (G, s)
    Para todo u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    vis[s] = T
    cria fila F
    Enfileira (F, s)
    Enquanto |F| > 0
        u = Desenfileira (F)
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                Enfileira (F, v)
    Devolva pred
```

$A = O(V)$

$B = O(1)$

$C = O(V)$

$O(1)$

D

$E = O(1)$

▶ G é um grafo e $s \in V(G)$

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

Se G foi implementado por matriz

$D = O(V^2)$

$T(G) = O(V) + O(V^2) = O(V^2)$

Se G foi implementado por lista

$D = O(E)$

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

$O(V)$ se matriz adj

$O(d(u))$ se lista adj

Observações

- A árvore resultante da BFS é chamada de <u>árvore de busca em largura.</u> ou <u>árvore da BFS</u>.

Distância em Grafos

BFS e 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, denotada por $dist(u,v)$, é o menor comprimento de um uv-caminho. Se não existe um caminho entre u e v, definimos $dist(u,v) = \infty$

$$dist(u,v) = \min\{e(P) : P \text{ é um } uv\text{-caminho}\}$$

- Ademais, podemos fazer a BFS exibir esse caminho mais curto

BFS e Distância

Problema Single Source Shortest Path (SSSP)

Entrada: Um grafo G e uma raiz $s \in V(G)$

Saída: a distância de s para todos os vértices de G (e um caminho mais curto de s para cada vértice de G)

BFS

- Vamos usar um vetor $d[u]$ para armazenar a distância de s até u

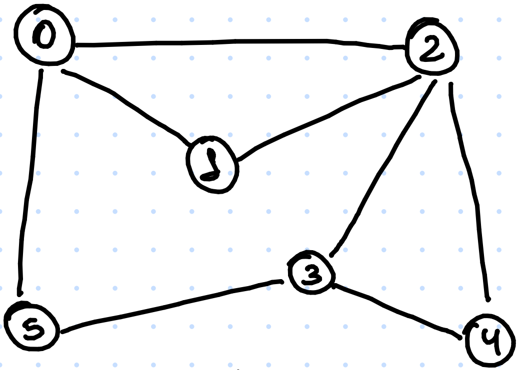
- Vamos usar a árvore de busca pred[] para armazenar os caminhos mais curtos

Busca Largura Distância

```
Função Busca-Largura (G, s)
    para toda u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
        d[u] = ∞
    pred[s] = s
    vis[s] = T
    d[s] = 0
    cria fila F
    Enfileira (F, s)
    Enquanto |F| > 0
        u = Desenfileira (F)
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                d[v] = d[u] + 1
                Enfileira (F, v)
    Devolva pred, d
```

Busca Largura Distância

Função Busca-Largura (G, s)
  para todo u ∈ V(G)
    vis[u] = F
    pred[u] = NULL
    d [u] = ∞
  pred[s] = s
  vis[s] = T
  d [s] = 0
  cria fila F
  Enfileira (F, s)
  Enquanto |F| > 0
    u = Desenfileira (F)
    para todo vértice v ∈ N(u)
      se vis[v] == F
        vis[v] = T
        pred [v] = u
        d [v] = d[u] + 1
        Enfileira (F, v)
  Devolva pred, d



FiLA :

	0	1	2	3	4	5
vis						
pred						
d						

Busca Largura Distância

Função Busca-Largura (G, s)

para todo $u \in V(G)$

- $vis[u] = F$
- $pred[u] = NULL$
- $d[u] = \infty$

$pred[s] = s$

$vis[s] = T$

$d[s] = 0$

cria fila F

Enfileira (F, s)

Enquanto $|F| > 0$

- u = Desenfileira (F)
- para todo vértice $v \in N(u)$
  - se $vis[v] == F$
    - $vis[v] = T$
    - $pred[v] = u$
    - $d[v] = d[u] + 1$
    - Enfileira (F, v)

Devolva $pred$, d

Complexidade

lista: $O(V+E)$

Matriz: $O(V^2)$

Objetivo

- Mostrar que $d[u] = \text{dist}_G(s, u)$ para todo $u \in V(G)$.
- Vetor pred[ ] define um árvore de Busca em Largura com raiz em s.

Função Busca-Largura (G, s)

```
Função Busca-Largura (G, s)
    para todo u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
        d[u] = ∞
    pred[s] = s
    vis[s] = T
    d[s] = 0
    cria fila F
    Enfileira (F, s)
    Enquanto |F| > 0
        u = Desenfileira (F)
        para todo vértice v ∈ N(u)
            se vis[v] == F
                vis[v] = T
                pred[v] = u
                d[v] = d[u] + 1
                Enfileira (F, v)
    Devolva pred, d
```

Obs: Note que d[u], pred[u], vis[u] nunca mudam após inserirmos u na fila

Alguns Lemas

Lema 1

Se $d[u] < \infty$, então u pertence à arvore T induzida por $pred[\,]$ e o caminho de s a u em T tem comprimento $d[u]$.

Corolário 2.

Durante a execução do algoritmo vale o seguinte invariante $d[v] \geq dist(s,v)$ para todo $v \in V(G)$.

$$d[v] = dist_T(s, v) \geq dist_G(s,v)$$

Lema 3 Sejam G um grafo e $s \in V(G)$. Na execução de Busca-Largura-Distância(G,s), se u e v são dois vértices que estão na fila e u entrou na fila antes de v, então

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

$$\begin{array}{c} \text{início} \\ \downarrow \end{array} \qquad\qquad \begin{array}{c} \text{Fim} \\ \downarrow \end{array}$$

$$F = \langle \underset{u}{x_1}, x_2, x_3, x_4, x_5, \underset{v}{x_6} \rangle$$

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

- Note que temos no máximo duas distâncias na fila.

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

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

Demonstração

- Pelo Corolário 2, $d[v] \geq dist_G(s,v)$ $\forall$ $v \in V(G)$

> **Corolário 2**
> Durante a execução do algoritmo vale o seguinte invariante
> $d[v] \geq dist(s,v)$ 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 para o qual $d[u] > dist_G(s, u)$

- Dentre todos os vértices u tais que $d[u] > dist(s,u)$, seja v um com a menor distância para s
$dist_G(s,v) = \min\{dist(s,u) : u \in V(G) \text{ e } d[u] > dist_G(s,u)\}$
- Seja $P = s, \ldots, u, v$ um caminho mais curto de s a v
    - $e(P) = dist_G(s,v)$
- Note que $dist(s,v) = dist(s,u) + 1$ (A)
- Pela escolha de v e por (A), temos que $d[u] \leq dist(s,u)$ e, consequentemente que $d[u] = dist_G(s,u)$
- Assim, $d[v] > dist(s,v) = d(s,u) + 1 = d[u] + 1$ (B)
- Considere o momento em que Busca-Largura-Distância(G,s) removeu u de F.

- Considere o momento em que Busca-Largura-Distância (G, s) removeu u de F.

① v havia sido visitado; ou

② v não havia sido visitado

- Caso 1: v havia sido visitado
  - um vizinho $w \neq u$ visitou v
    - $d[v] = d[w] + 1$
    - w saiu de F antes de u

Pelo Lema 3, $d[w] \leq d[u]$

Lema 3 Sejam G um grafo e $s \in V(G)$. Na execução de Busca-Largura-Distância (G, s), se u e v são dois vértices que estão na fila e u entrou na fila antes de v, então

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

Por Ⓑ

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

absurdo

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

- Caso 2: v não havia sido visitado
  - v é visitado
  - v é inserido em F
  - $d[v] = d[u] + 1$

  Por B, $d[v] > d[u] + 1$

$$d[v] > d[u] + 1 = d[v] \qquad \square$$

Lema A

Seja G um grafo (X, Y)-bipartido e seja $u_1 \in X$.
Se $P = u_1, u_2, u_3, \dots, u_\ell$, então

ⓐ $u_i \in X$ se i é ímpar

ⓑ $u_i \in Y$ se i é par

DFS

Busca em Profundidade

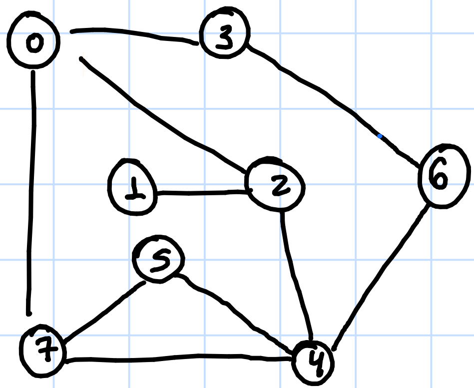
Depth-First Search

Busca em Profundidade (DFS)

- a ideia é expandir a árvore pela vizinhança do último vértice adicionado a árvore.

"último a entrar, primeiro a sair"

Exemplo

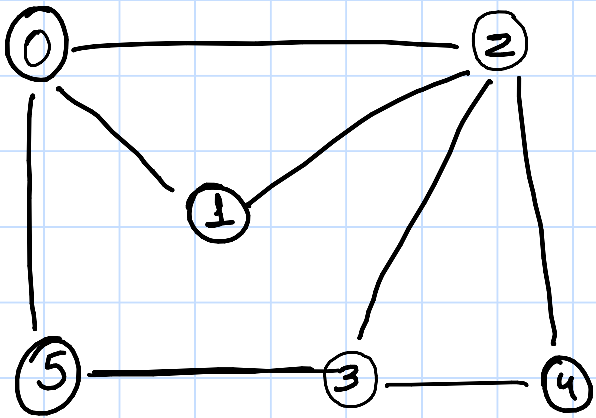


Busca em Profundidade (DFS)

Esse algoritmo pode ser usado para:

- encontrar componentes conexas
- ~~calcular distância entre vértices~~
- encontrar caminhos entre vértices
- detectar ciclos
- verificar se um grafo é bipartido (e produzir uma em caso afirmativo)
- encontrar uma árvore geradora da componente contendo a raiz.
- Encontrar aresta de corte
- Encontrar vértice de corte

Execução da DFS



Pilha

Vis

0	1	2	3	4	5

pred

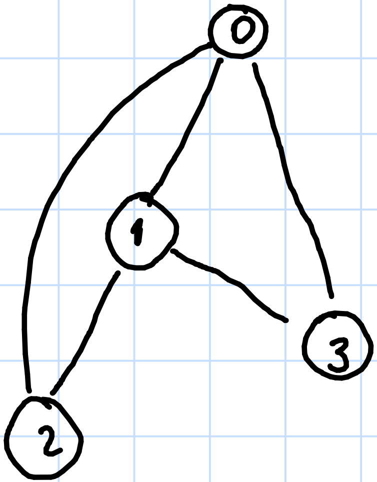
0	1	2	3	4	5

DFS

```
Função DFS(G, s)
    Para toda u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    DFS-SEARCH(G, s)
```

```
Função DFS-SEARCH(G, u)
    vis[u] = T
    Para todo v ∈ N(u)
        Se vis[v] == F
            pred[v] = u
            DFS-SEARCH(G, v)
```

Execução DFS



Função DFS(G, s)

Para toda $u \in V(G)$

vis[u] = F

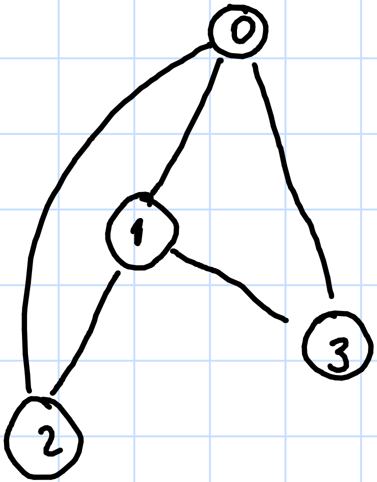
pred[u] = NULL

pred[s] = s

DFS - SEARCH(G, s)

	0	1	2	3
vis	F	F	F	F
pred	–	–	–	–

Execução DFS

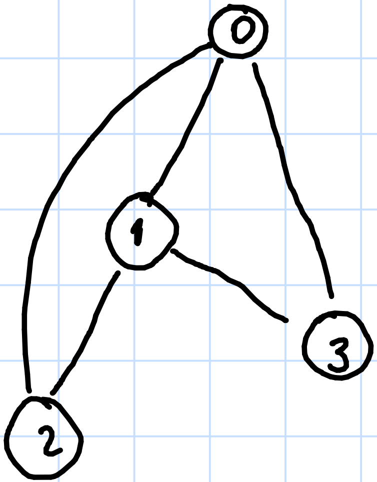



```
Função DFS(G, s)
    Para toda u ∈ V(G)
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    DFS-SEARCH(G, s)
```

	0	1	2	3
vis	F	F	F	F
pred	0	–	–	–

Vamos escolher 0 como raiz

Execução DFS



Função DFS(G, s)

  Para toda $u \in V(G)$

    vis[u] = F

    pred[u] = NULL

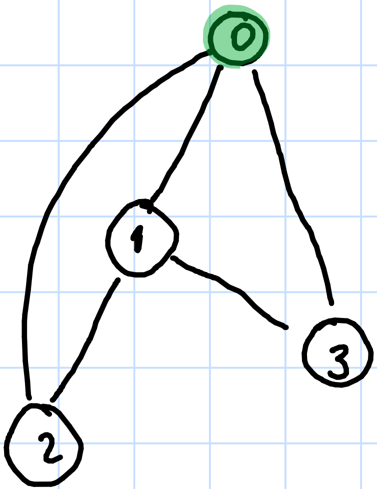
  pred[s] = s

  → DFS-SEARCH(G, s)

	0	1	2	3
vis	F	F	F	F
pred	0	–	–	–

DS(0)

Execução DFS



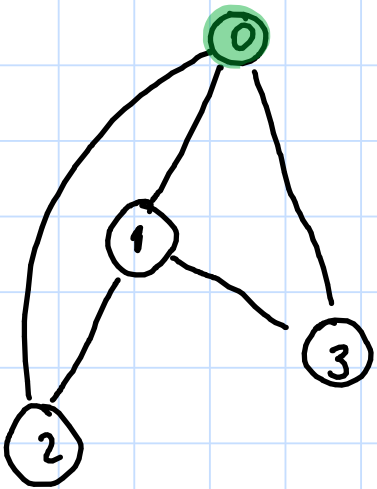
Função DFS-SEARCH(G, u)

```
→ vis[u] = T
  Para todo v ∈ N(u)
      Se vis[v] == F
          pred[v] = u
          DFS-SEARCH(G, v)
```

	0	1	2	3
vis	T	F	F	F
pred	0	–	–	–

DS(0)

Execução DFS



Função DFS-SEARCH(G, u)

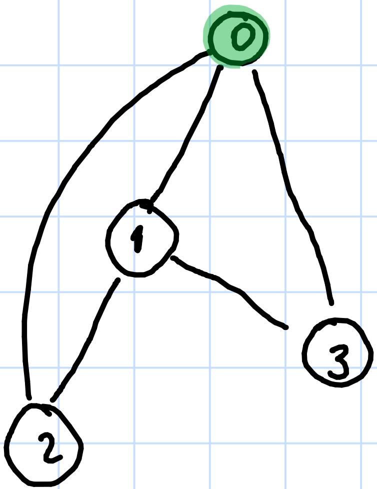
    vis[u] = T
    Para todo v ∈ N(u)
        Se vis[v] == F
            pred[v] = u
            DFS-SEARCH(G, v)

	0	1	2	3
vis	T	F	F	F
pred	0	–	–	–

DS(0)

$u = 0$ $v = 1$

Execução DFS



Função DFS-SEARCH(G, u)

vis[u] = T

Para todo v ∈ N(u)

Se vis[v] == F

pred[v] = u

→ DFS-SEARCH(G, v)

	0	1	2	3
vis	T	F	F	F
pred	0	0	–	–

DS(0)

DS(1)

$u = 0$ $v = 1$

DFS (Complexidade)

Função DFS(G, s)
    Para todo $u \in V(G)$
        vis[u] = F
        pred[u] = NULL
    pred[s] = s
    DFS-SEARCH(G, s)

A $\Theta(V)$

B $\Theta(1)$

Função DFS-SEARCH(G, u)
    vis[u] = T
    para todo $v \in N(u)$
        if vis[v] == F
            pred[v] = u
            DFS-SEARCH(G, v)

C $O(V)$

D.

$\Theta(1)$

D

	lista	matriz
custo 1 chamada	$\Theta(d(u))$	$\Theta(V)$
custo de todas as chamadas	$O(E)$	$O(V^2)$

Vamos chamar de H a comp. que contém a raiz

$$D = \sum_{u \in V(H)} d(u) = 2e(H) = O(e(H))$$
$$= O(e(G))$$

DFS (Complexidade)

Função DFS(G, s)
  Para toda $u \in V(G)$ — A — $\Theta(V)$
    vis[u] = F
    pred[u] = NULL
  pred[s] = s
  DFS-SEARCH(G, s) — B — $\Theta(1)$

Função DFS-SEARCH(G, u)
  vis[u] = T — C — $O(V)$
  para toda $v \in N(u)$ — D.
    if vis[v] == F — $\Theta(1)$
      pred[v] = u
      DFS-SEARCH(G, v)

$$T(G) = A + B + C + D$$
$$= O(V) + O(1) + O(V) + O(D)$$
$$= O(V + 1 + V + D) = O(V + D)$$

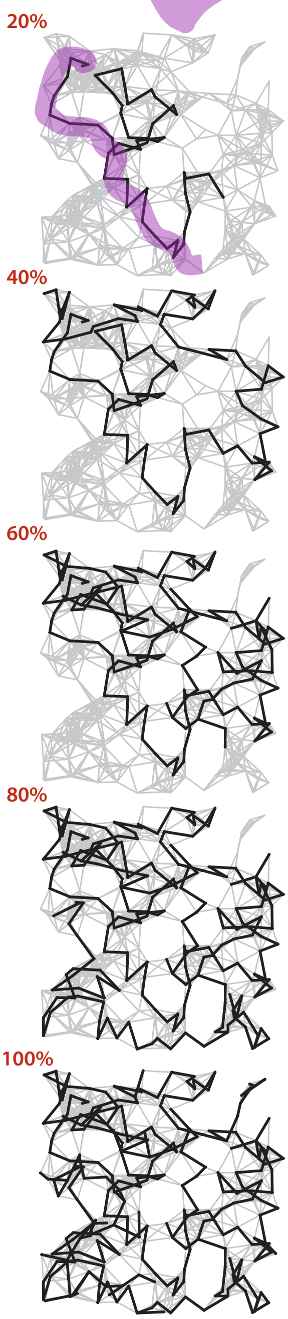
Lista → $O(V+E)$

MATRIZ → $O(V^2)$

Observações

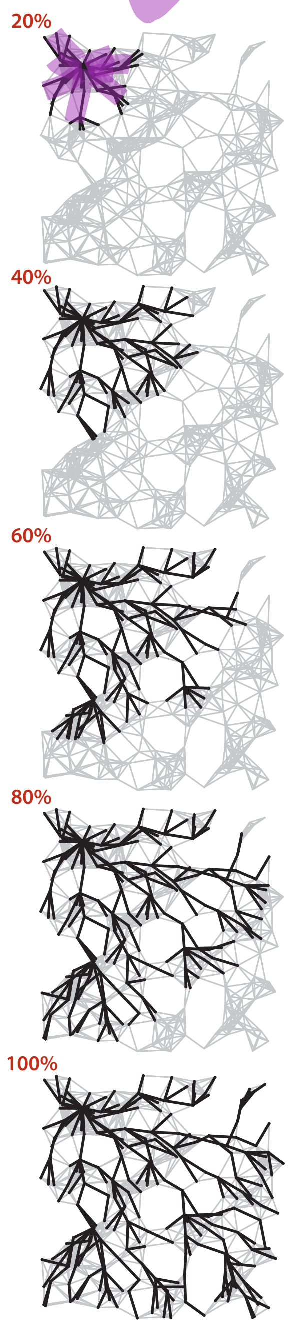
- Complexidade DFS
  - lista de adjacência : $O(V+E)$
  - matriz de adjacência : $O(V^2)$

- A árvore resultante da DFS é chamada de <u>árvore de busca em Profundidade</u>, ou <u>árvore da DFS</u>.



DFS for paths (250 vertices)

THE DIAGRAMS ON EITHER SIDE of this page, which show the progress of DFS and BFS for our sample graph `mediumG.txt`, make plain the differences between the paths that are discovered by the two approaches.DFS wends its way through the graph, storing on the stack the points where other paths branch off; BFS sweeps through the graph, using a queue to remember the frontier of visited places. DFS explores the graph by looking for new vertices far away from the start point, taking closer vertices only when dead ends are encountered; BFS completely covers the area close to the starting point, moving farther away only when everything nearby has been examined. DFS paths tend to be long and winding; BFS paths are short and direct. Depending upon the application, one property or the other may be desirable (or properties of paths may be immaterial). In SECTION 4.4, we will be considering other implementations of the `Paths` API that find paths having other specified properties.



BFS for shortest paths (250 vertices)