C.E.S.A.R school

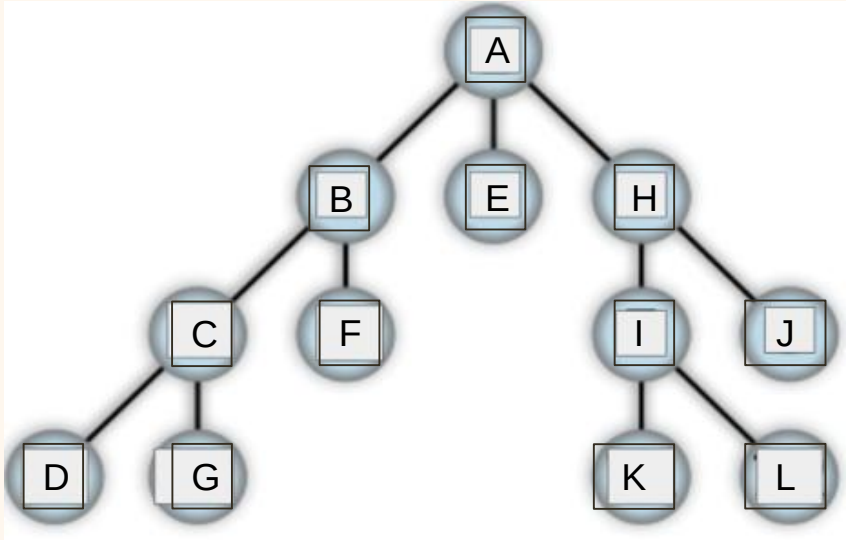
# TEORIA DOS GRAFOS

Profª Laura Pacifico

2025 | SETEMBRO

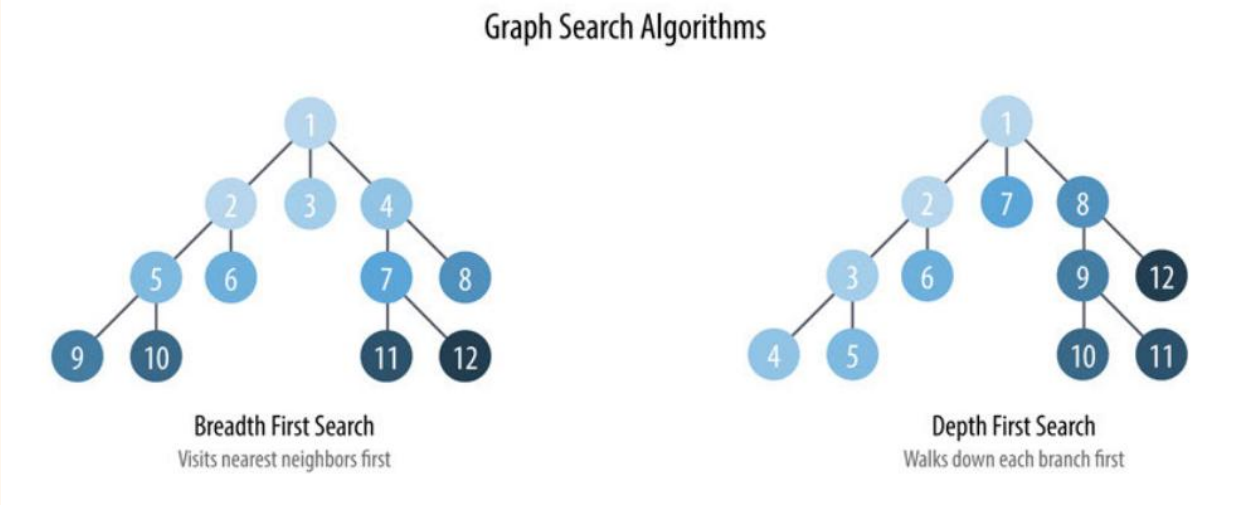
# Algoritmos de Busca em Grafos

- Como navegar por todos os vértices?

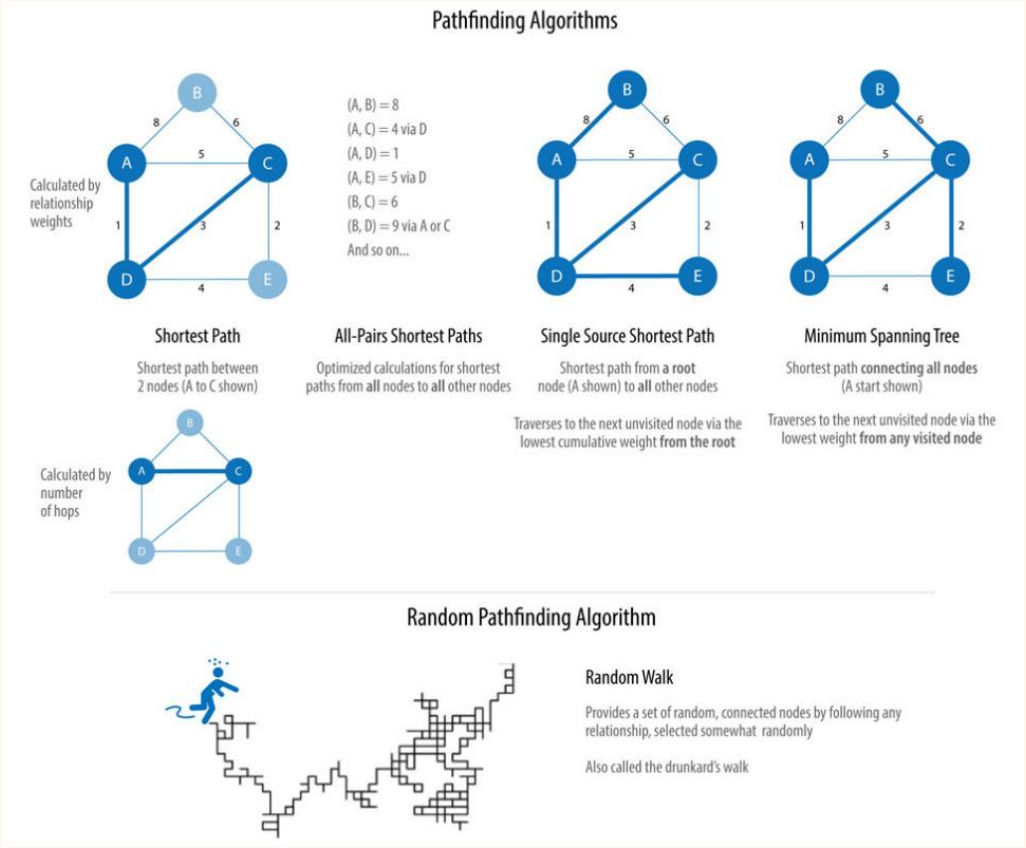


# Algoritmos de Busca em Grafos

- Algoritmos de busca em grafos exploram um grafo para busca explícita ou exploração
  - Alguns caminhos não são computacionalmente ótimos
- Busca em largura e busca em profundidade



# Algoritmos de Caminhos em Grafos



# Busca em Largura (Breadth First Search - BFS)

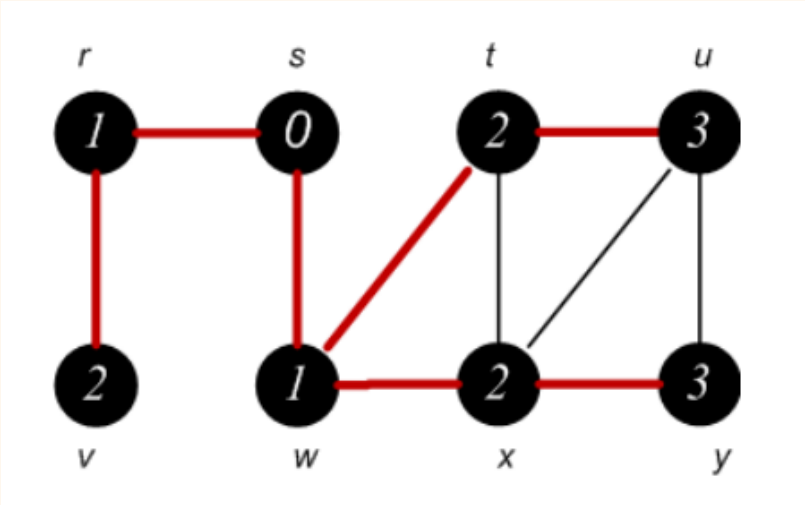
- Proposto em 1959 por Edward F. Moore e desenvolvido por C. Y. Lee em 1961

- Começa de um determinado nó e explora toda sua vizinhança imediata antes de visitar os demais nós

- A lógica BFS pode ser utilizada como base para algoritmos mais especializados
  - Por exemplo, menor caminho e componentes conectados

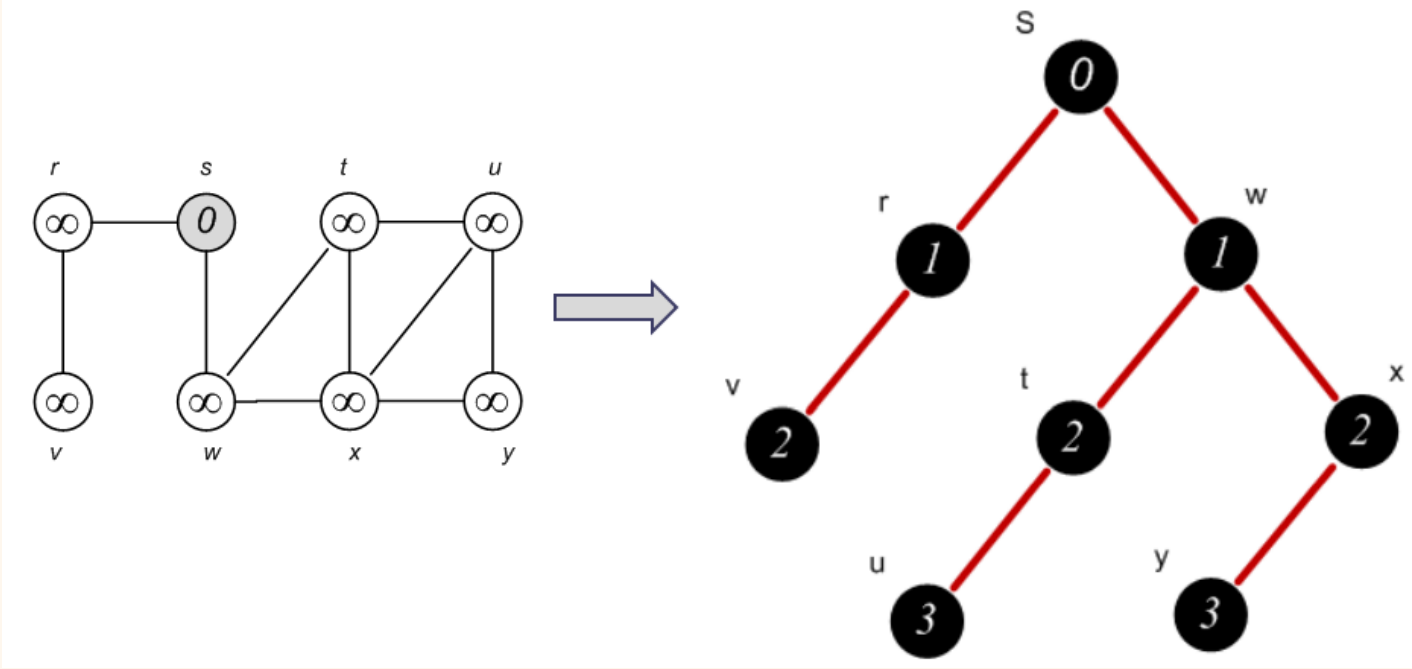
# Busca em Largura (Breadth First Search - BFS)

- O algoritmo da Busca em Largura calcula a distância (menor número de arestas) desde o vértice s (raiz) até todos os vértices acessíveis;
  - Não considera a distância como o somatório do peso de arestas;
  - Considera a quantidade de saltos necessários mínimos para alcançar outro vértice do grafo;

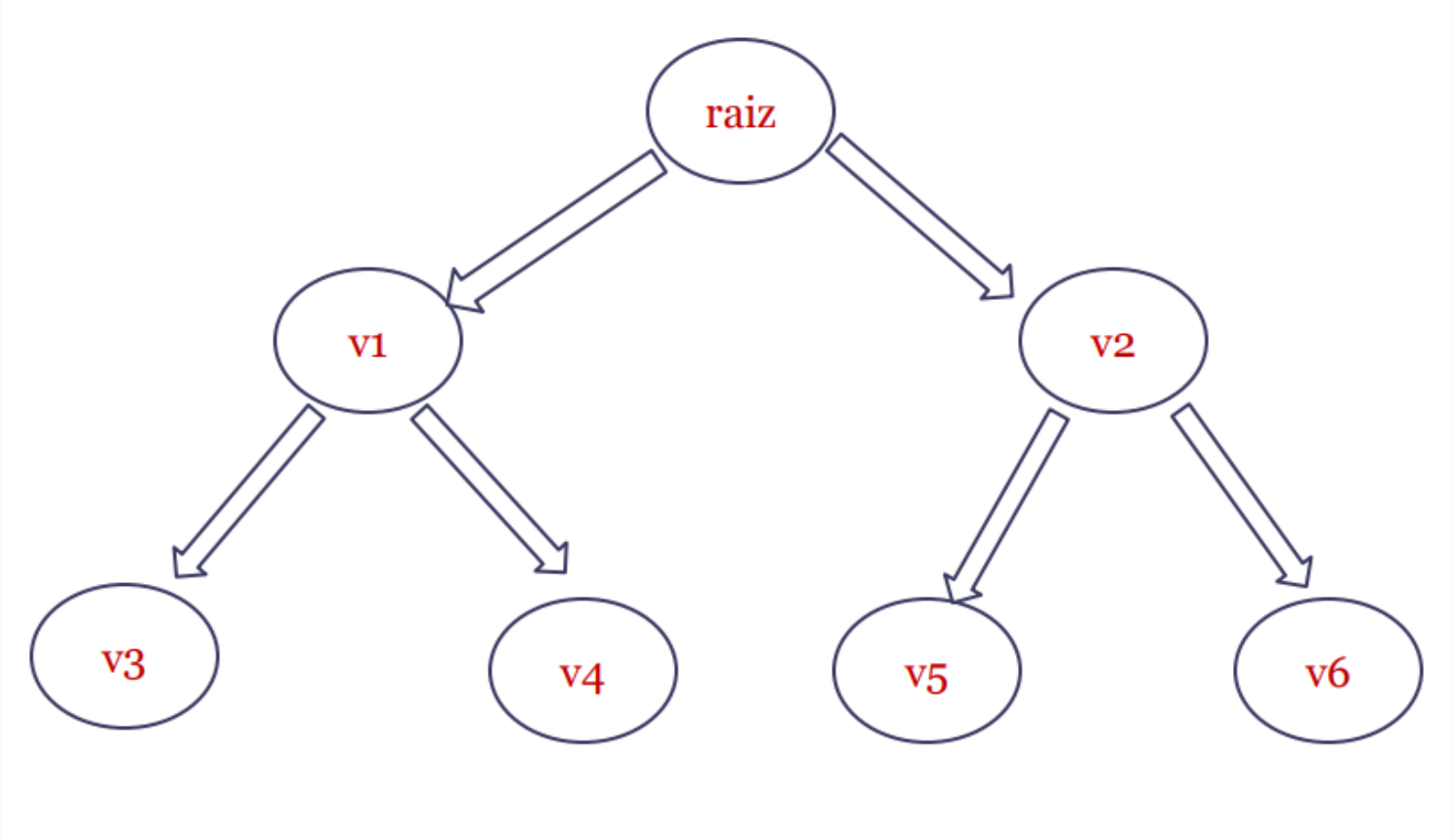


# Busca em Largura (Breadth First Search - BFS)

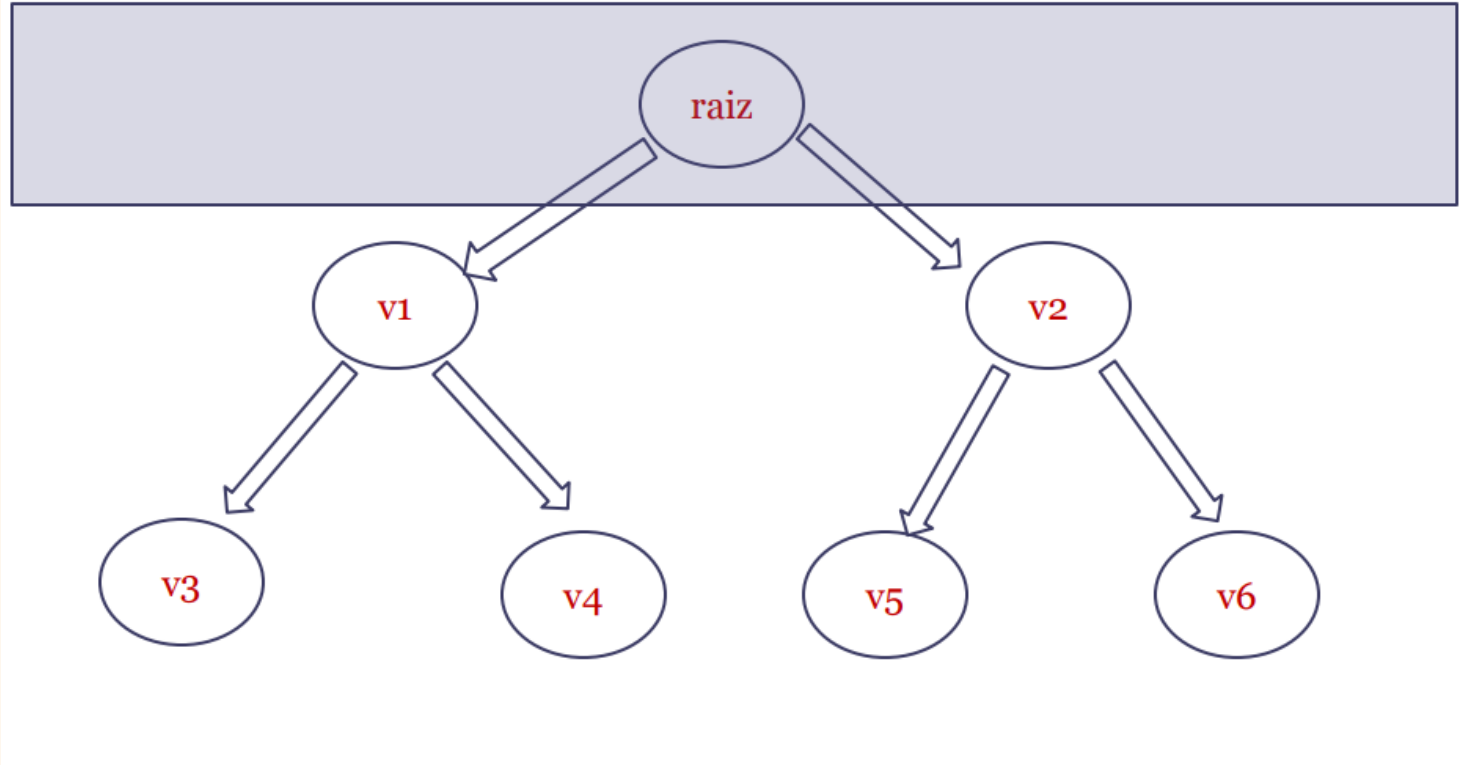
- Ele também produz uma “Árvore Primeiro na Extensão”, com raiz  no vértice de partida, que contém todos os vértices acessíveis;



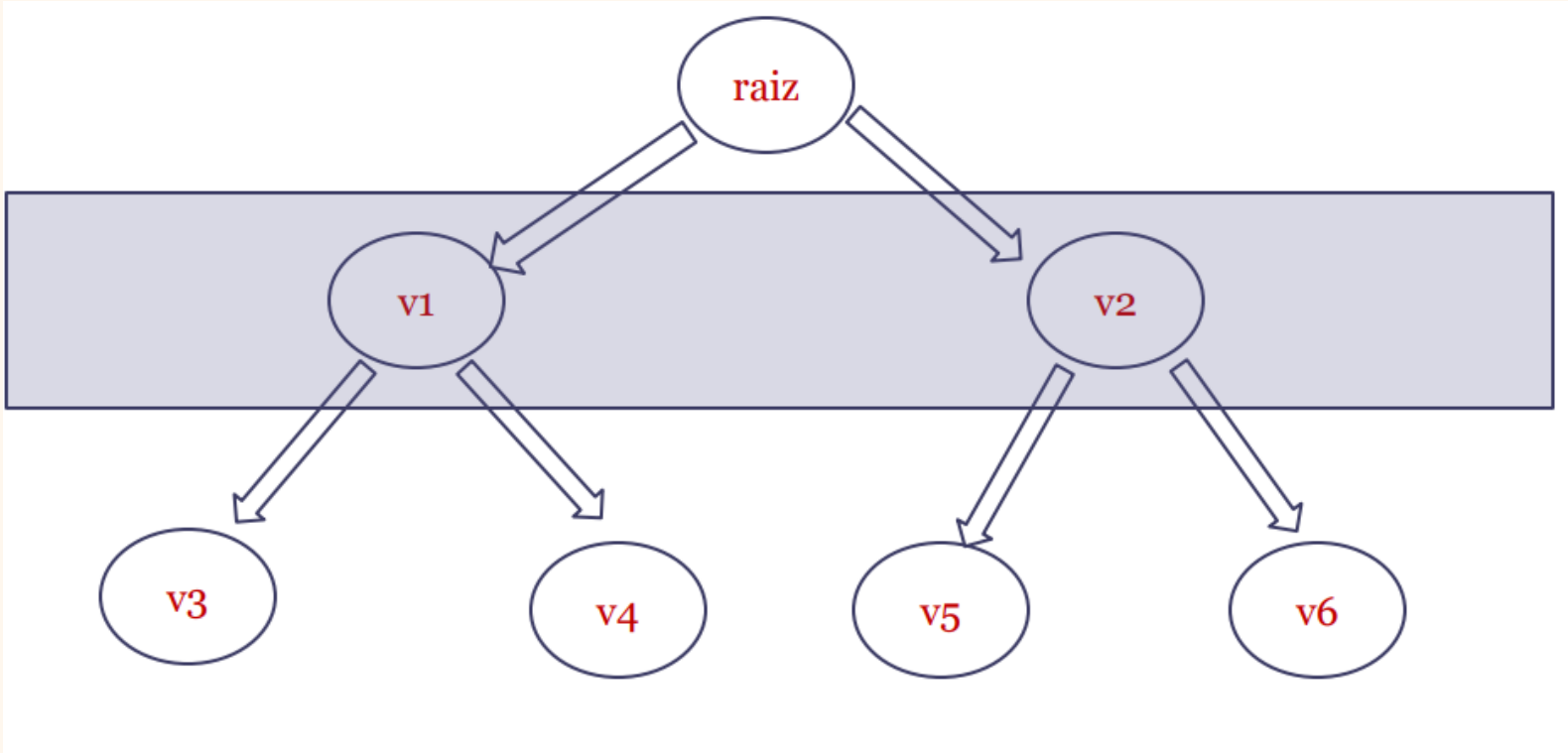
# Aplicando Busca em Largura em uma Árvore



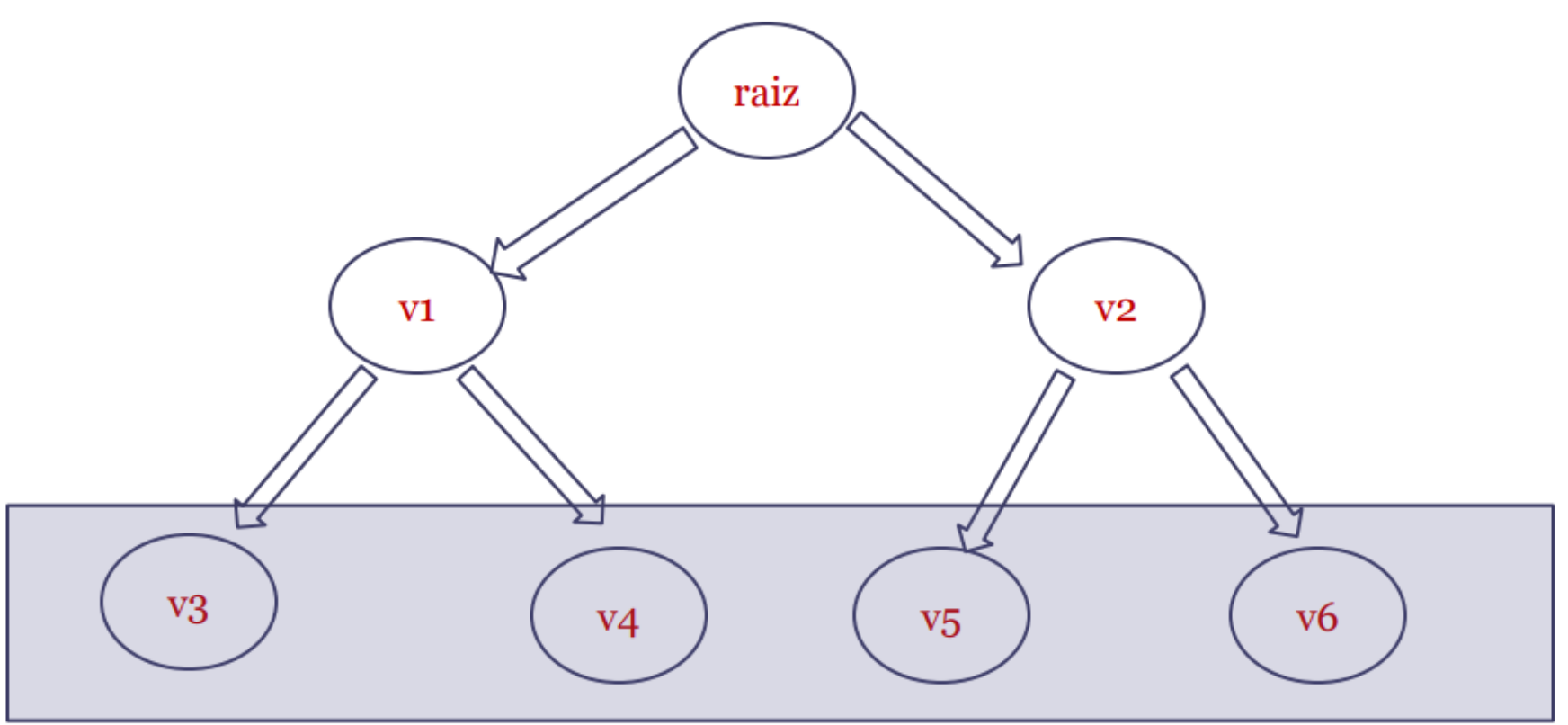
# Aplicando Busca em Largura em uma Árvore



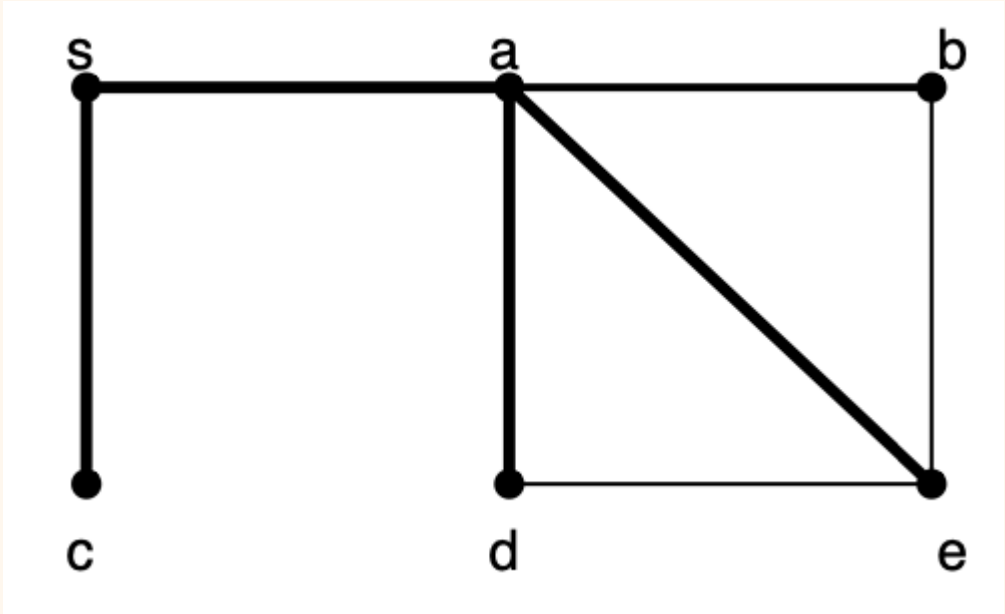
# Aplicando Busca em Largura em uma Árvore

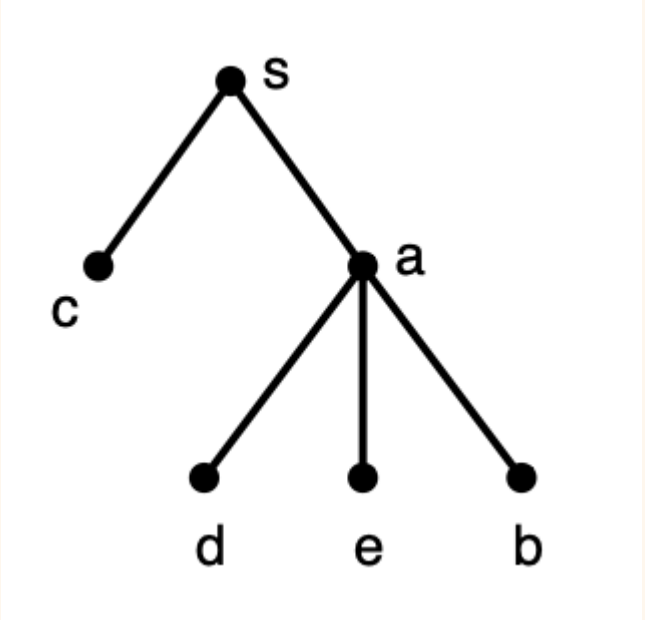


# Aplicando Busca em Largura em uma Árvore



# Busca em Largura



# Busca em Largura

A árvore de busca em largura é uma árvore *n-ária*, pois não sabemos a priori quantos filhos cada nó da árvore terá.

Nas implementações de árvores *n-árias*, geralmente cada nó possui uma lista encadeada de referências para os nós filhos.

No algoritmo, não sabemos quantos filhos um nó terá. Mas sabemos que todos os filhos possuem um único pai.

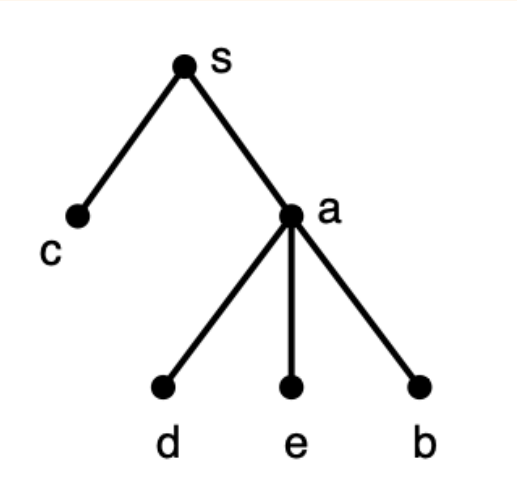
Ao invés de armazenar os filhos, armazenamos a referência do pai. Fica muito mais simples e podemos armazenar **em um vetor.**

# Busca em Largura

A este vetor, damos o nome de **vetor de roteamento ou vetor de predecessores** e o denotamos pelo símbolo *R(G)*.

Qual o vetor de roteamento do grafo?

| *v* | *s* | *a* | *b* | *c* | *d* | *e* |
|---|---|---|---|---|---|---|
| *p(v)* | *nil* | *s* | *a* | *s* | *a* | *a* |



# Busca em Largura

enquanto a fila não estiver vazia
retire um vértice *v* da fila
para cada vizinho *w* de *v*
se *w* não está numerado
então numere *w*
ponha *w* na fila

No começo da *primeira* iteração, a fila contém o vértice *s*, com número 0, e nada mais.

# Código: Busca em Largura

$v_0$ -> vértice inicial

***estado(v)*** -> VISITADO (busca encontrou-o, mas não visitou seus adjacentes),
NÃO-VISITADO (busca não o encontrou ainda),
ENCERRADO (busca o encontrou e também seus adjacentes)

***p(v)*** -> vértice pai do vetor

***F*** -> fila de vértices encontrados na busca

***d(v)*** *-> distância de v0 até v em quantidades de arestas percorridas*

***adj(v)*** -> lista com todos os vértices adjacentes de *v*

# Pseudocódigo: Busca em Largura *(BFS)*

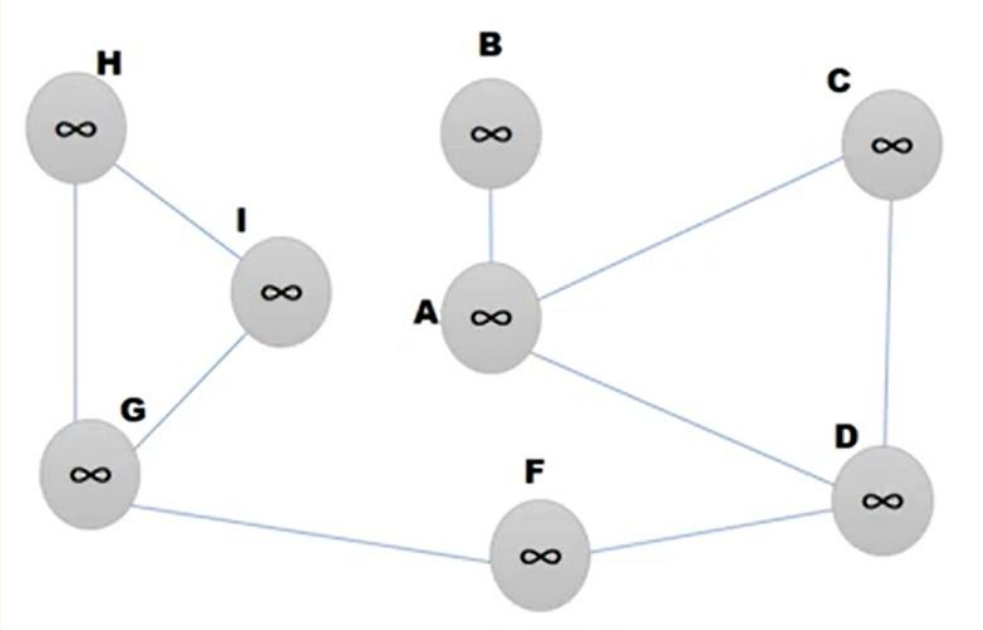
**Para cada vértice, inicialize**
estado(v) = NAO_VISITADO
p(v) = nil
d(v) = 0

**Escolha arbitrariamente um vértice $v_0$ e inicialize:**
$d(v_0) = 0$
$estado(v_0) = VISITADO$
F = 0
$F.insere(v_0)$

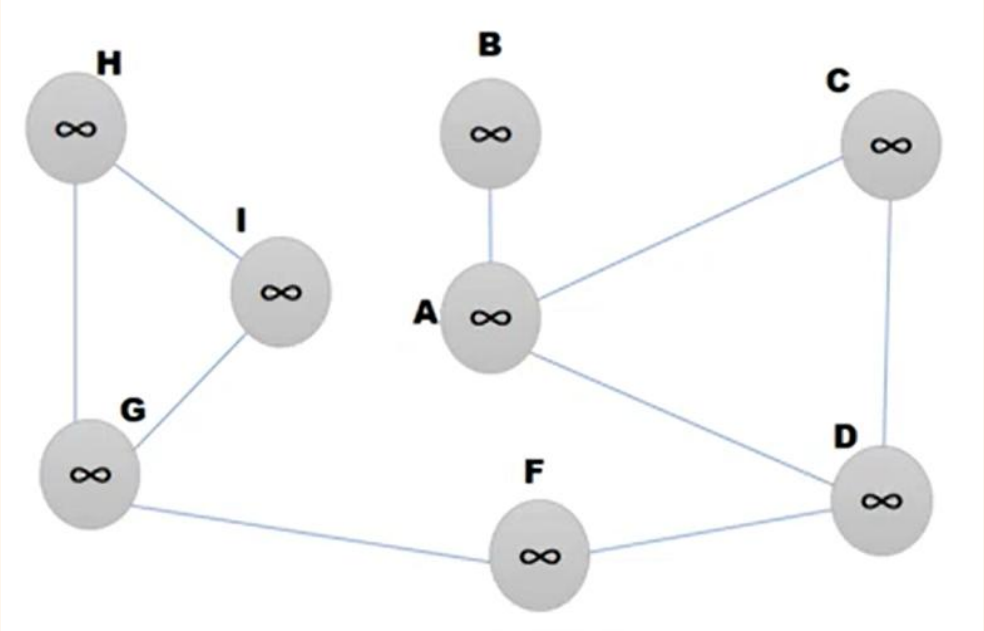
***$v_0$*** -> vértice inicial
***estado(v)*** -> VISITADO, NAO_VISITADO, ENCERRADO
***p(v)*** -> vértice pai do vetor
***F*** -> fila de vértices encontrados na busca
***d(v)*** -> *distância de v0 até v em qtd de arestas percorridas*
***adj(v)*** -> lista com todos os vértices adjacentes de *v*

**enquanto F != 0:**
    $v_i = F.remove()$
    **para cada $v_j$ adjacente a $v_i$ :**
        **se estado($v_j$) = NAO_VISITADO**
            $F.insere(v_j)$
            $estado(v) = VISITADO$
            $p(v_j) = v_i$
            $d(v_j) = d(v_i) + 1$
    $estado(v_i) = ENCERRADO$

# Busca em Largura

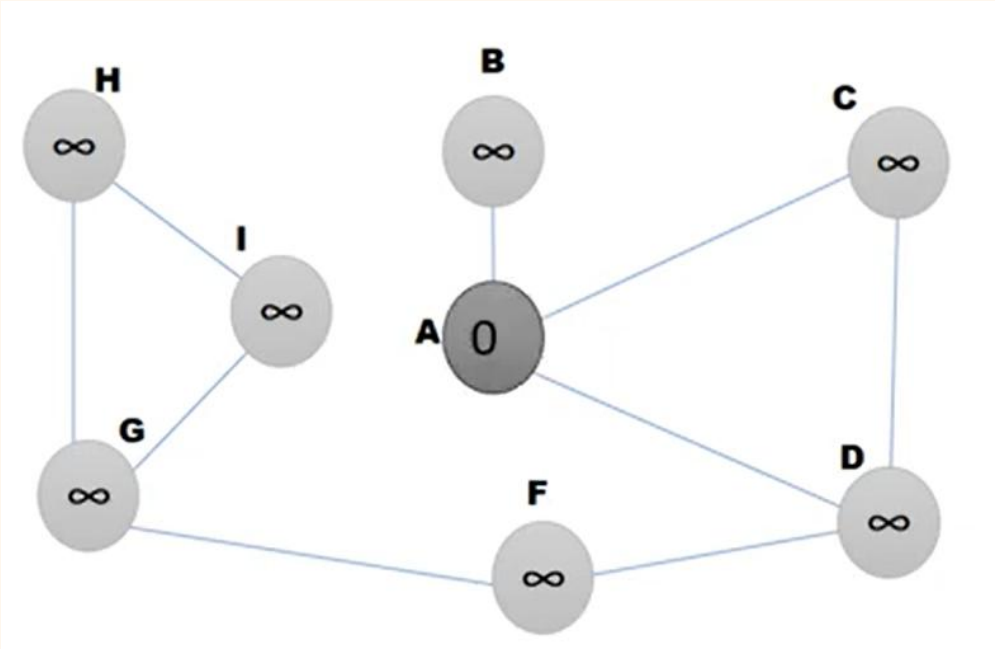


# Busca em Largura



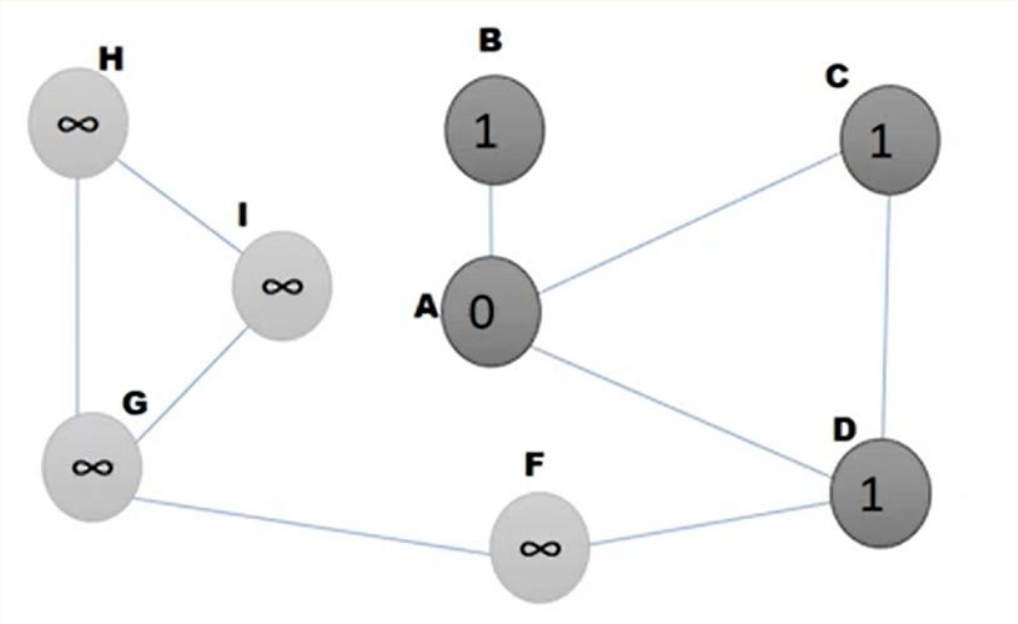

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| D | 0 | | | | | | | | |
| P | - | | | | | | | | |

# Busca em Largura




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | | | | | | | |
| P | - | A | | | | | | | |

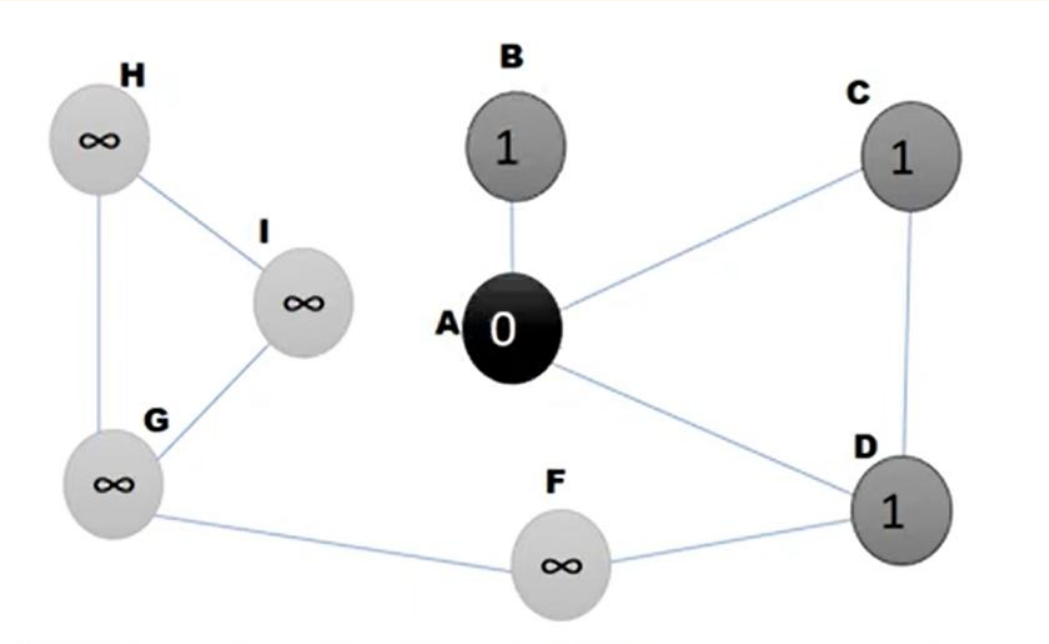
# Busca em Largura




**A B C D**

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | | | | | |
| P | - | A | A | A | | | | | |

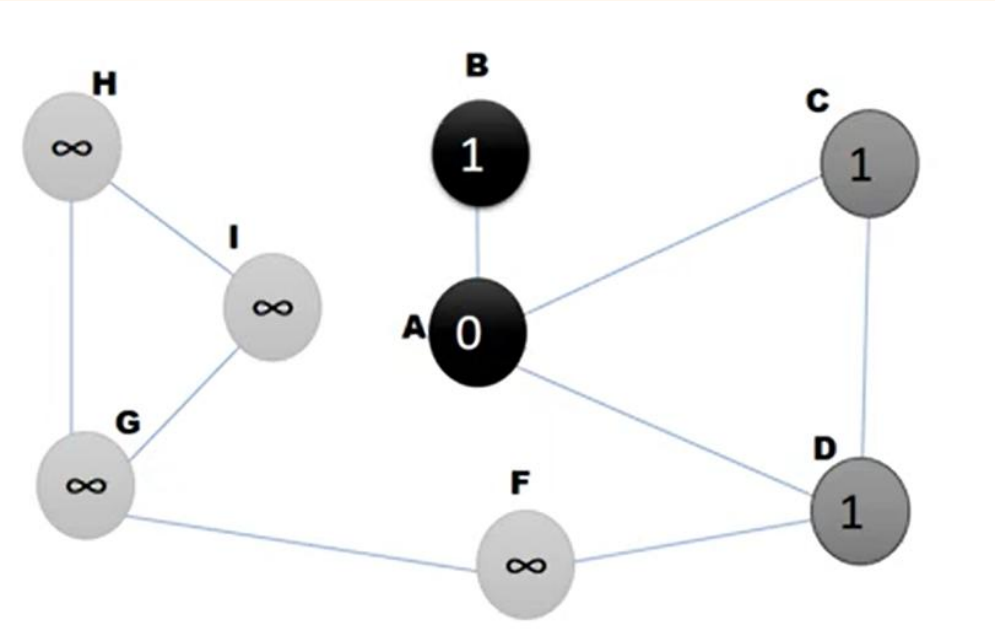
# Busca em Largura




B C D

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | | | | | |
| P | - | A | A | A | | | | | |

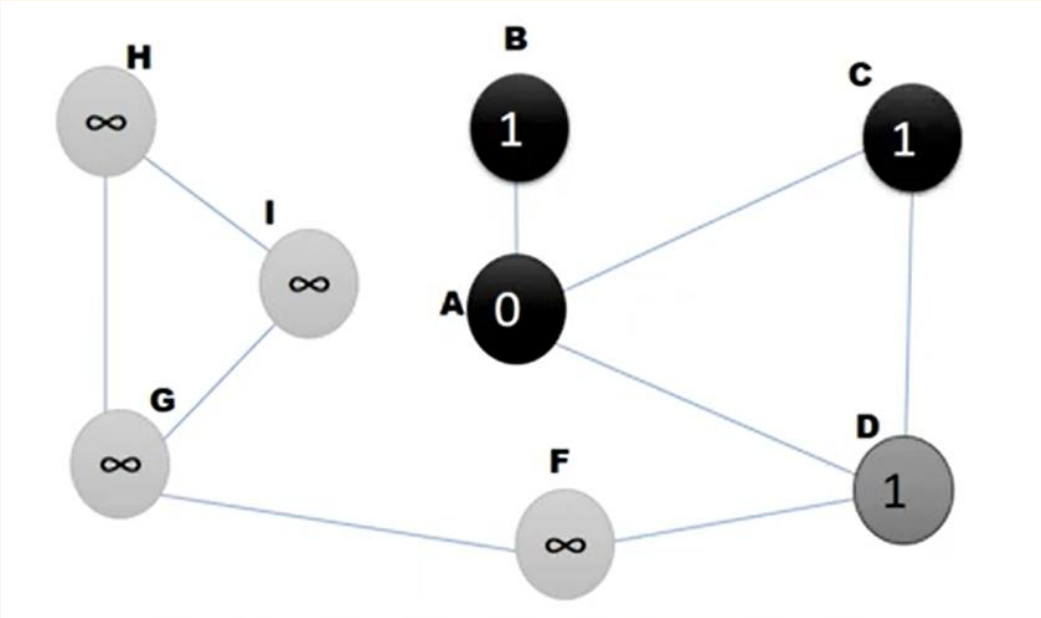
# Busca em Largura




C D

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | | | | | |
| P | - | A | A | A | | | | | |

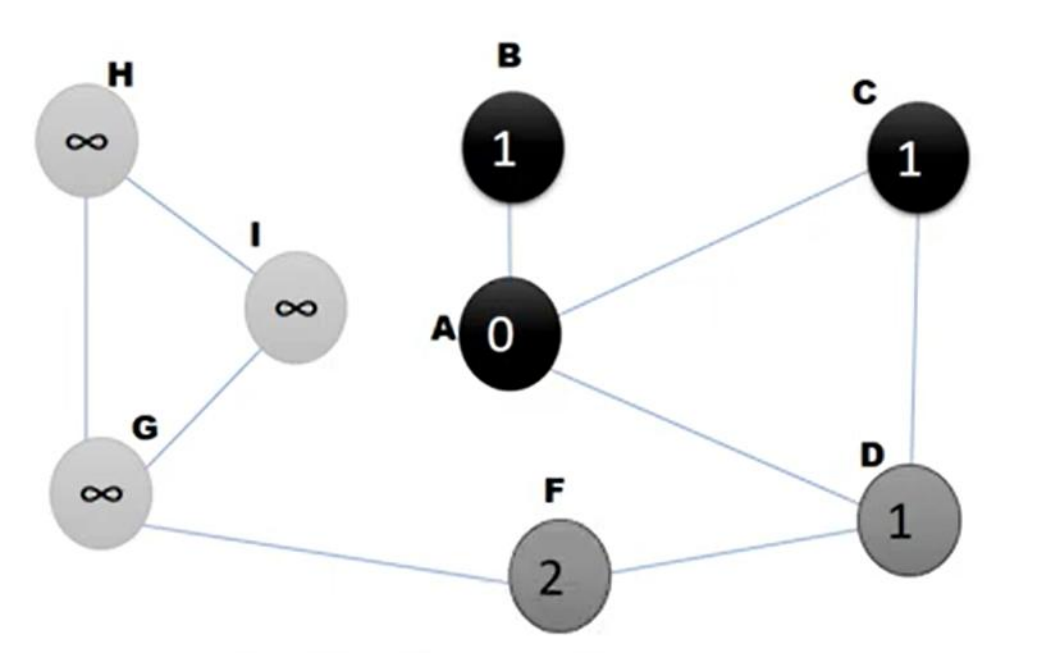
# Busca em Largura




D

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | | | | |
| P | - | A | A | A | | | | |

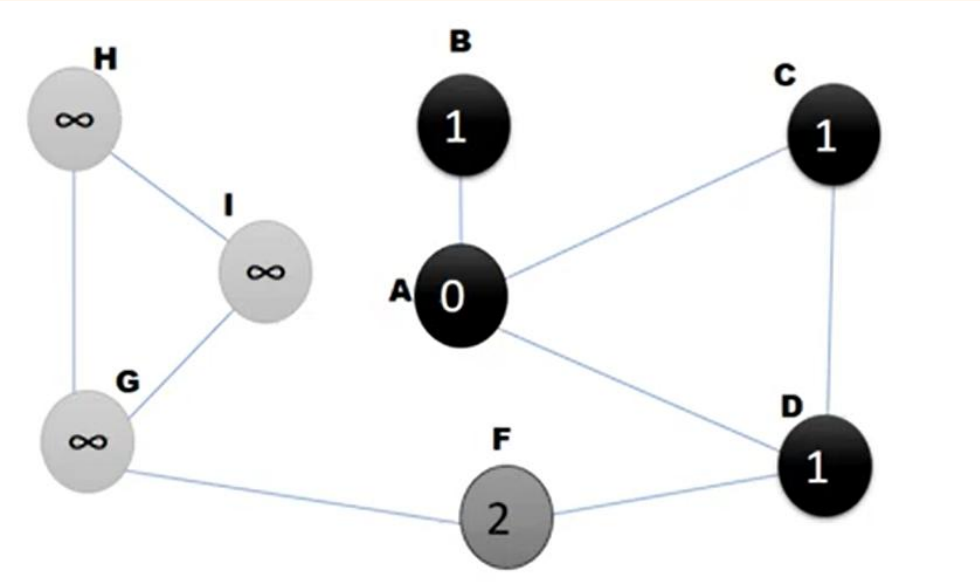
# Busca em Largura




DF

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | | | |
| P | - | A | A | A | D | | | |

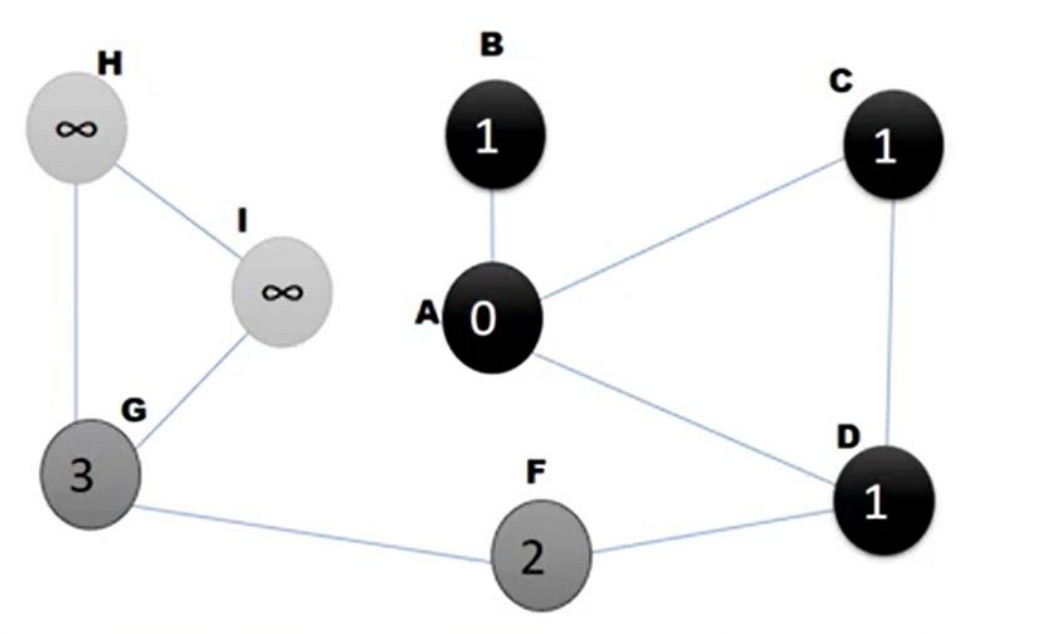
# Busca em Largura




F

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | | | |
| P | - | A | A | A | D | | | |

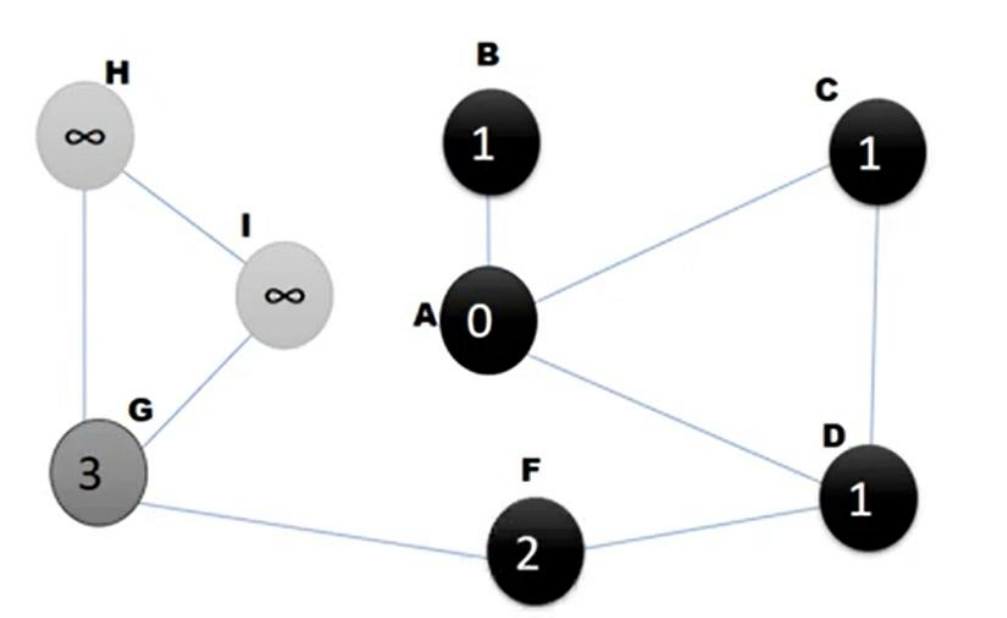
# Busca em Largura




FG

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | | |
| P | - | A | A | A | D | F | | |

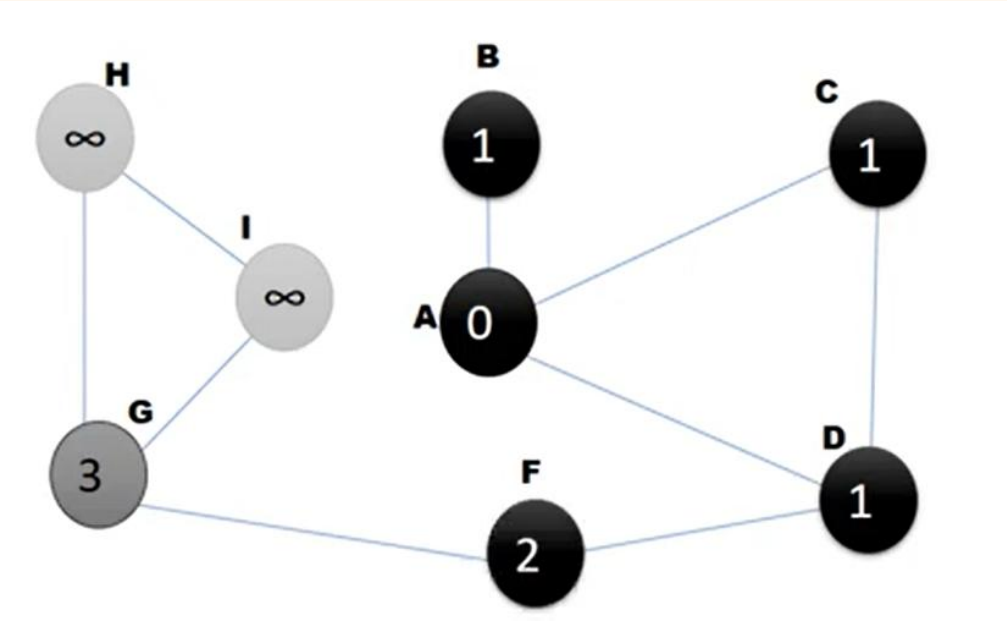
# Busca em Largura




G

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | | |
| P | - | A | A | A | D | F | | |

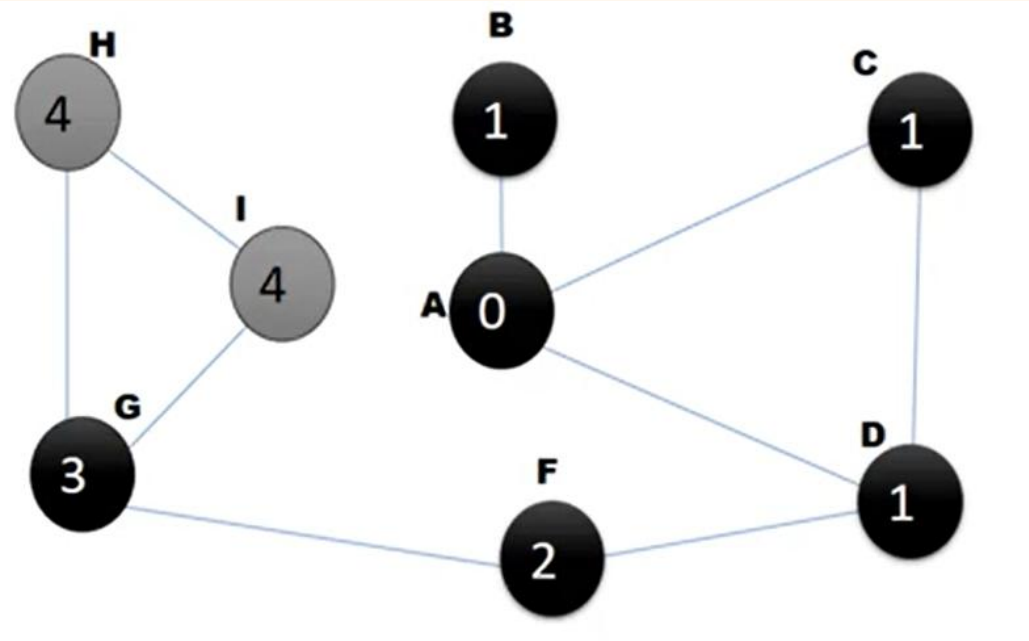
# Busca em Largura




G I H

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| P | - | A | A | A | D | F | G | G |

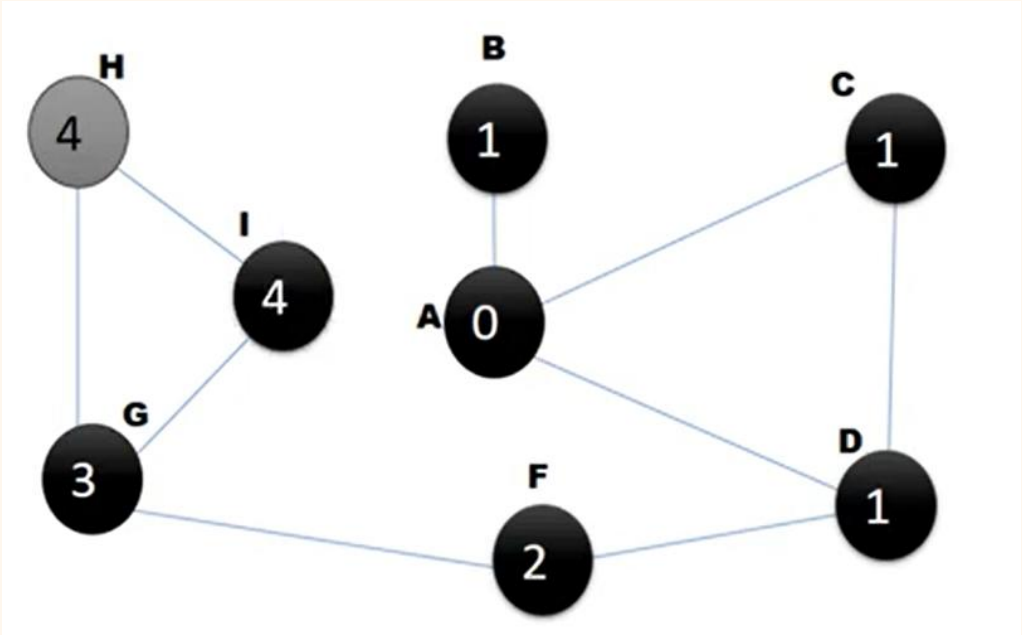
# Busca em Largura




I H

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| P | - | A | A | A | D | F | G | G |

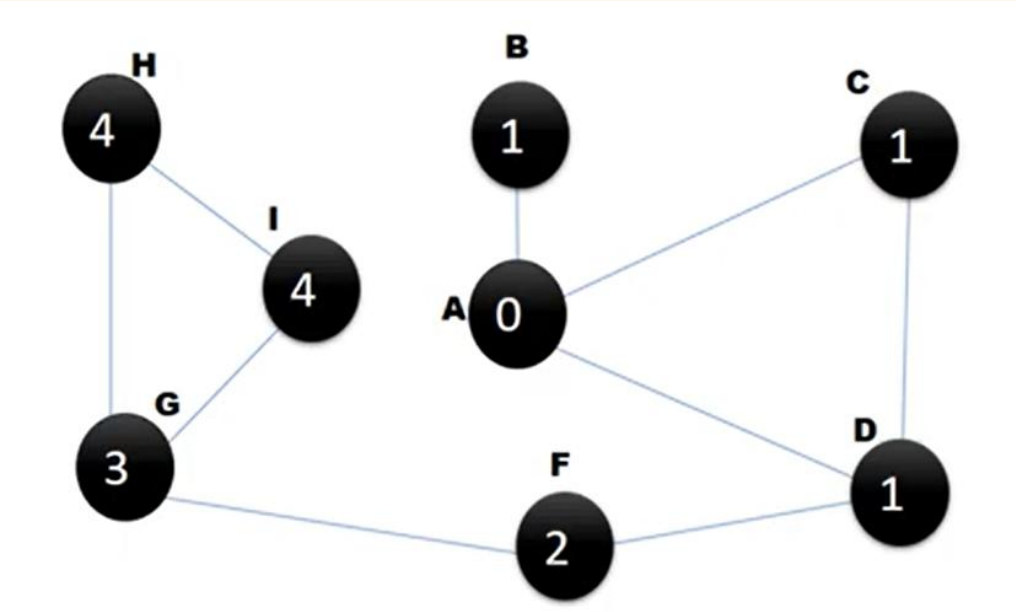
# Busca em Largura




H

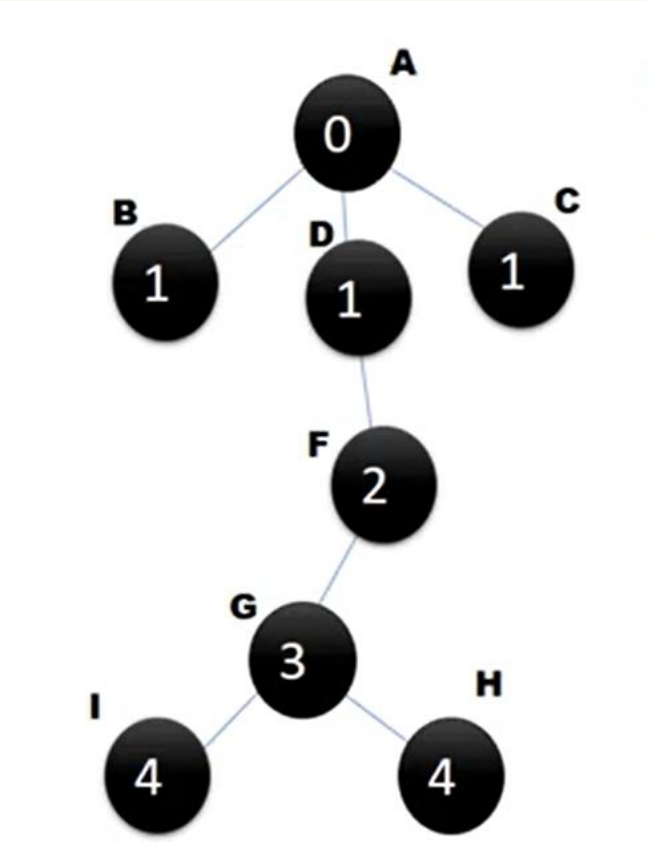
| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| P | - | A | A | A | D | F | G | G |

# Busca em Largura




| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| P | - | A | A | A | D | F | G | G |

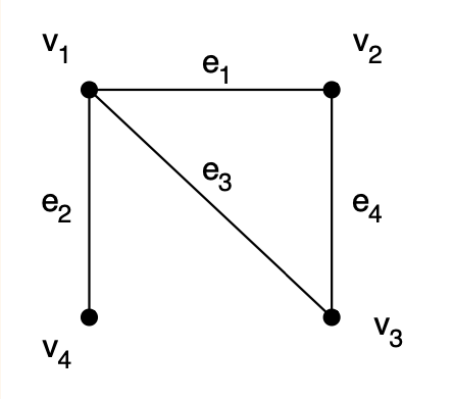
# Busca em Largura



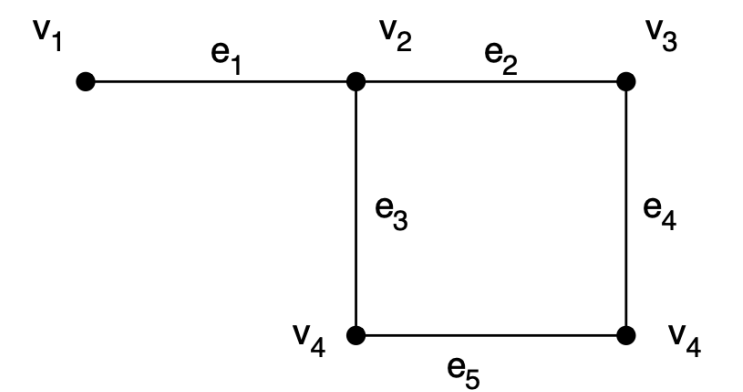

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| P | - | A | A | A | D | F | G | G |

# Busca em Largura

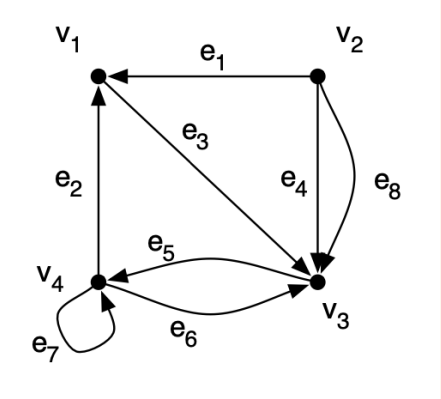
Calcule a distância de cada grafo abaixo usando o algoritmo de *BFS*. Simule os estados da Fila por iteração e informando a distância com cada vértice.




d(v3, v4)




d(v4, v1)




d(v4, v2)

# Dúvidas?

Laura Alves Pacifico
laps@cesar.school
Slack: Laura Pacifico