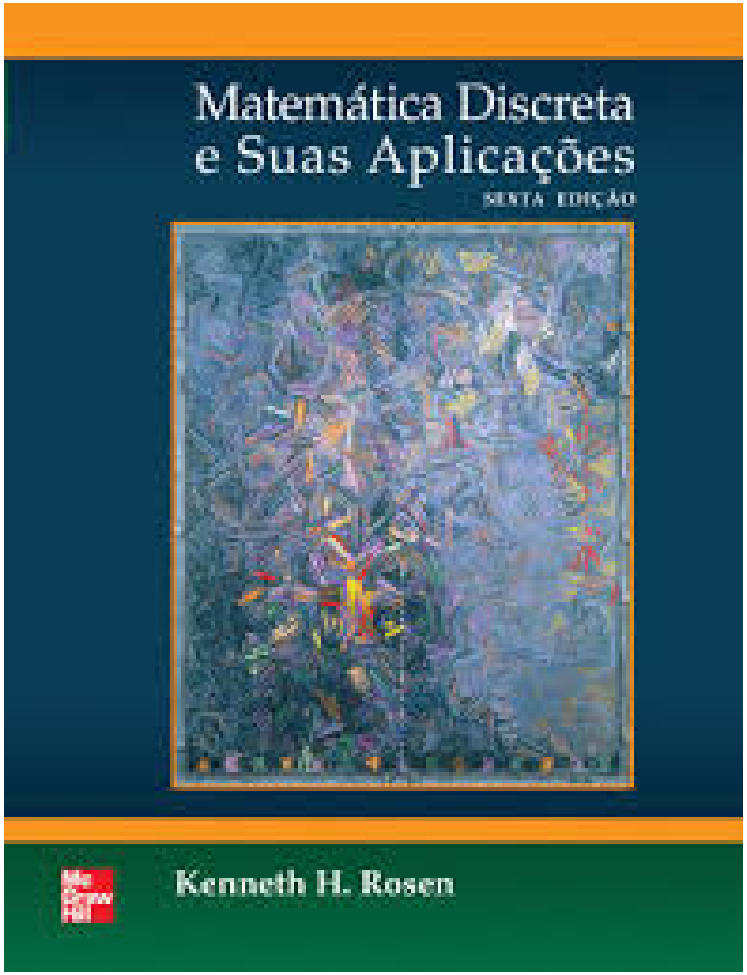
Matemática Discreta

Grafos – Caminhamento em grafos

Daniel Hasan Dalip
hasan@decom.cefetmg.br

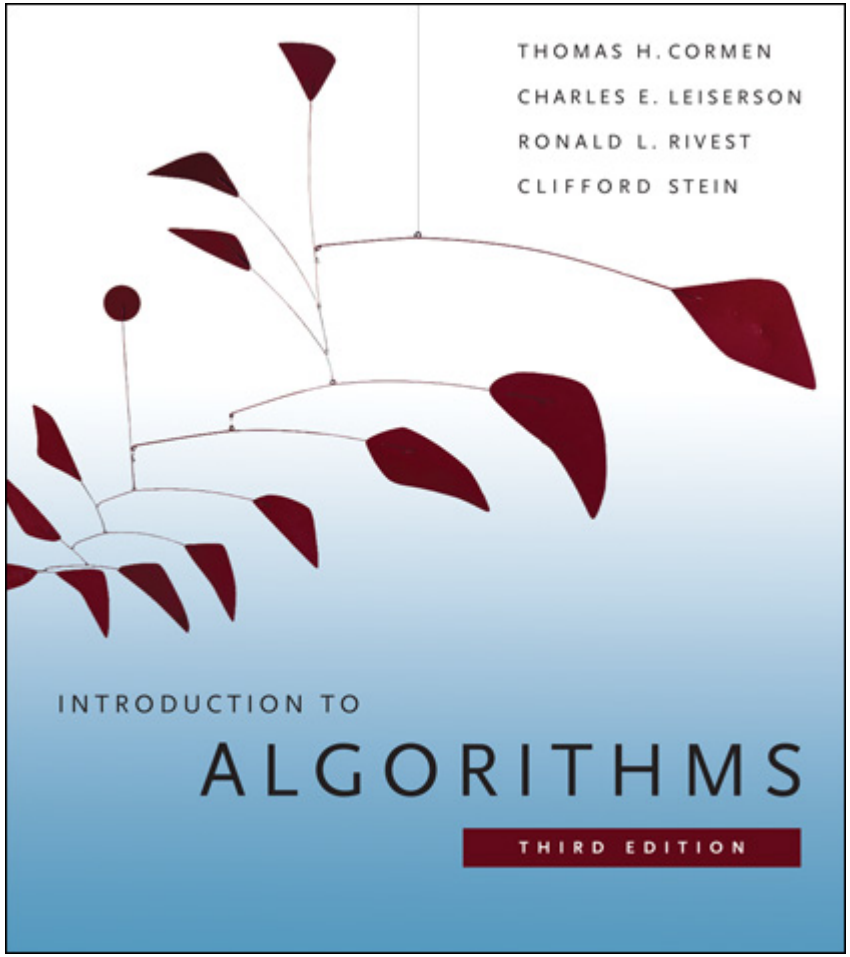
Bibliografia de Hoje
Principal

- Rosen, K. H., Matemática Discreta e Suas Aplicações.
  - Capítulo 10



Bibliografia de Hoje
Principal

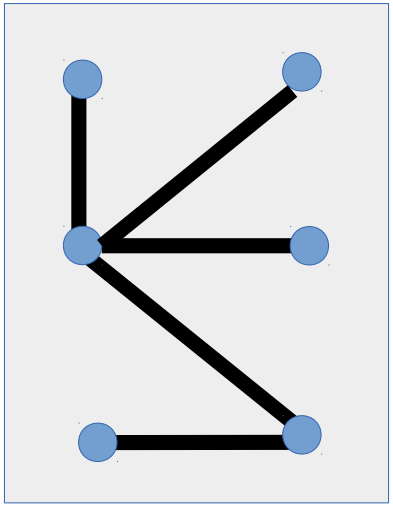
- Cormen T. et al., Introduction to Algorithms.
  - Capítulo 22



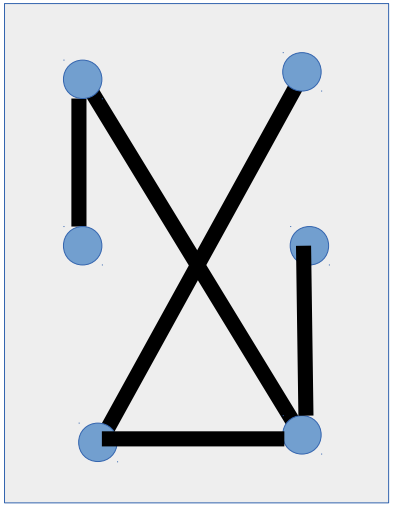
Árvores

- Árvore é um grafo:
  - acíclico
    - Não possui ciclos
  - conexo
    - Existe caminho entre cada par de vértices

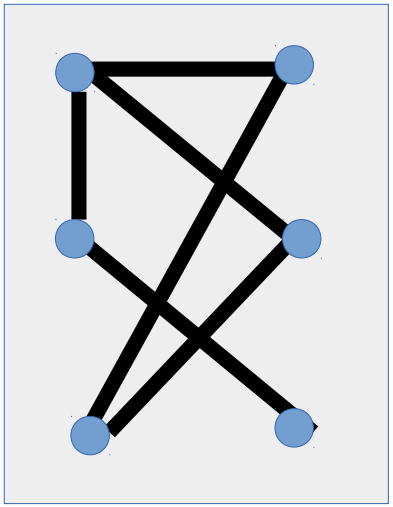
Quais grafos são árvore? [Rosen, 2009]



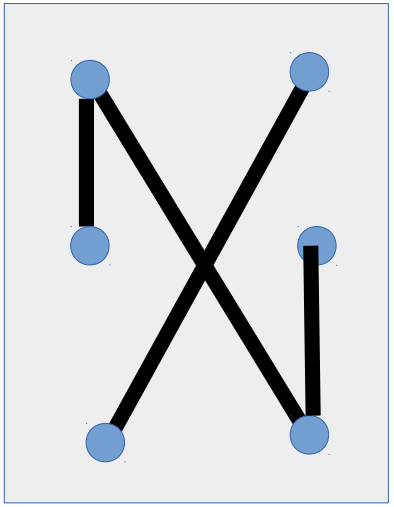
G_1



G_2

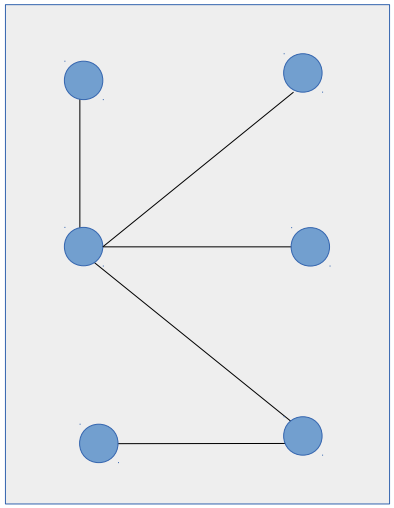


G_3

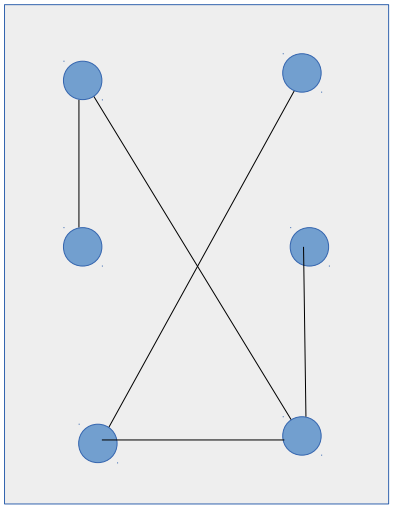


G_4

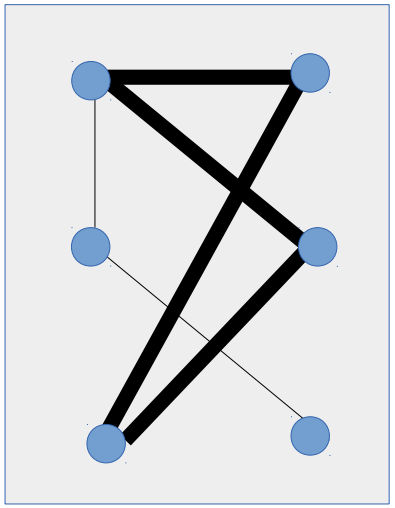
Quais grafos são árvore? [Rosen, 2009]



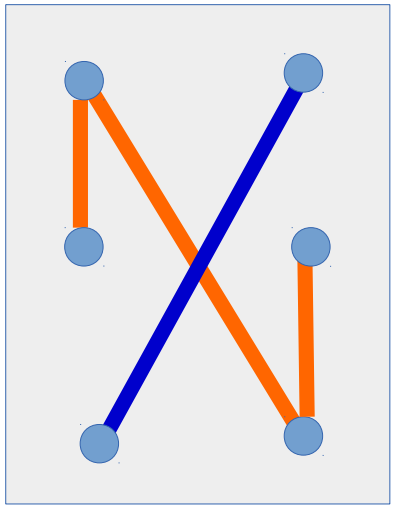
G_1



G_2



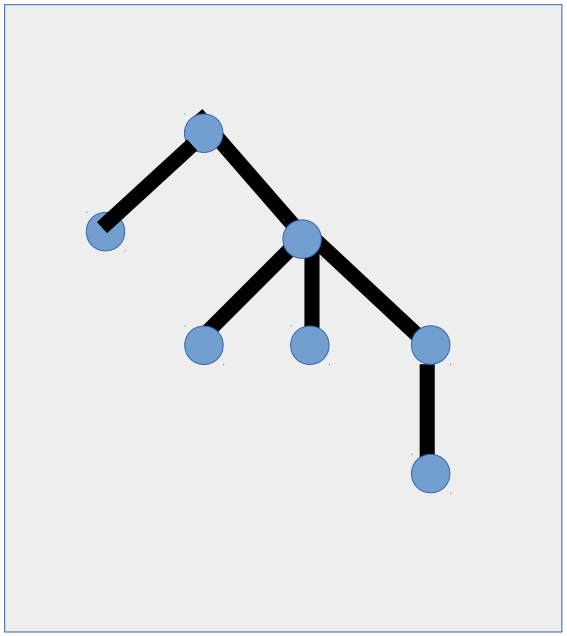
G_3



G_4

- São árvores os grafos: G_1 e G_2
- G_3: possui um ciclo
- G_4: não é conexo

Árvores Enraizadas



- Árvores em que elegemos um de seus vértices para ser sua raiz
- Podemos enraizar qualquer árvore
- Diferentes raízes, produzem diferentes árvores
- As arestas são orientadas no sentido de afastamento da raiz
  - Tal orientação pode ser omitida

Aplicação de árvores: arvore de decisão

Aprendizado de Máquina

O que é?

"Área de estudo que fornece aos computadores a habilidade de aprender sem ser explicitamente programado" (Samuel A., 1959)

- Ramo da Inteligência Artificial
- Alto grau de aplicação
- Cenário atual é favorável:
  - Facilidade de coleta e armazenamento
  - Disponibilidade de soluções

Aprendizado de Máquina

Definição mais formal (Mitchell, 1997):

- Estudo de algoritmos capazes de
  - melhorar o **desempenho (D)**
  - em uma certa **tarefa (T)**
  - através de **experiência (E)**

Aprendizado de Máquina

Definição mais formal (Mitchell, 1997):

- Estudo de algoritmos capazes de
  - melhorar o **desempenho (D)**
  - em uma certa **tarefa (T)**
  - através de **experiência (E)**

Cada tarefa utiliza uma métrica
para avaliar o quão bem é executada

Aprendizado de Máquina

Definição mais formal (Mitchell, 1997):

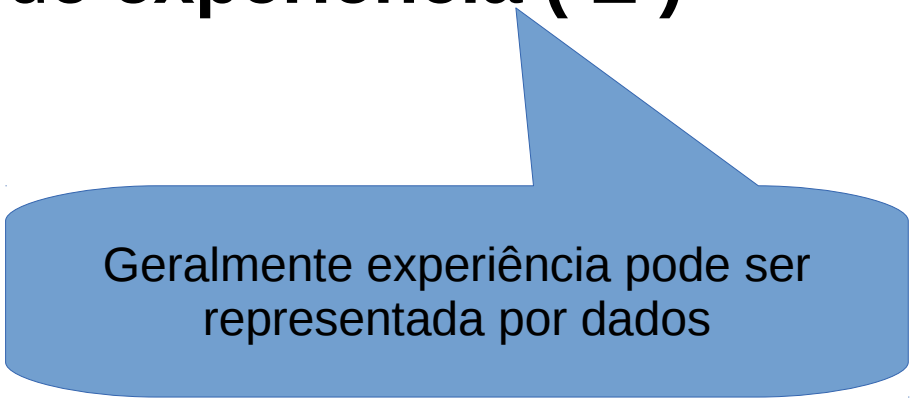
- Estudo de algoritmos capazes de
  - melhorar o **desempenho (D)**
  - em uma certa **tarefa (T)**
  - através de **experiência (E)**

Diversos tipos de tarefas que demandam diferentes tipos de algoritmo

Aprendizado de Máquina

Definição mais formal (Mitchell, 1997):

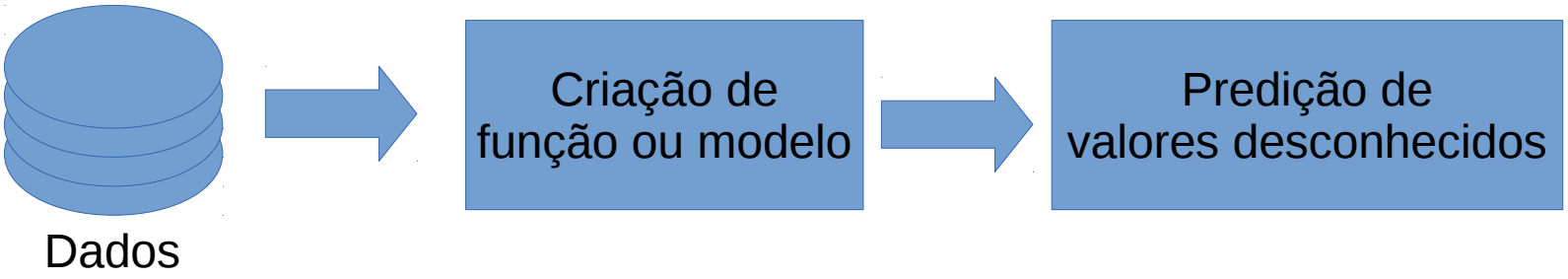
- Estudo de algoritmos capazes de
  - melhorar o **desempenho (D)**
  - em uma certa **tarefa (T)**
  - através de **experiência (E)**



Aprendizado Supervisionado

- Humanos aprendem com experiências
  - Experiência no que deu **certo** e **errado**

- Um computador não tem experiências
  - Mas pode aprender através de dados



Aprendizado Supervisionado

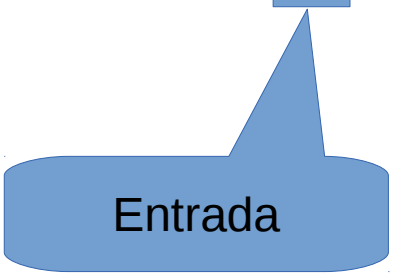
- Onde está a supervisão?

$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$

Aprendizado Supervisionado

- Onde está a supervisão?

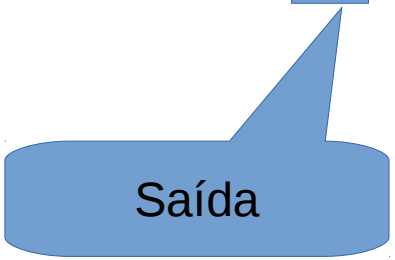
$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$



Aprendizado Supervisionado

- Onde está a supervisão?

$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$



Aprendizado Supervisionado

- Onde está a supervisão?

$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$

- O fato de saber a saída para algumas entradas torna o processo supervisionado

Aprendizado Supervisionado

- Onde está a supervisão?

$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$

- O fato de saber a saída para algumas entradas torna o processo supervisionado
  - y_i contínuo: Regressão
  - y_i categórico: Classificação

Aprendizado Supervisionado

- Onde está a supervisão?

$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$

Número real, escala ex: Nota, Lucro obtido, Tempo de execução de uma tarefa

- O fato de saber a saída para algumas entradas torna o processo supervisionado
  - y_i contínuo: Regressão
  - y_i categórico: Classificação

Aprendizado Supervisionado

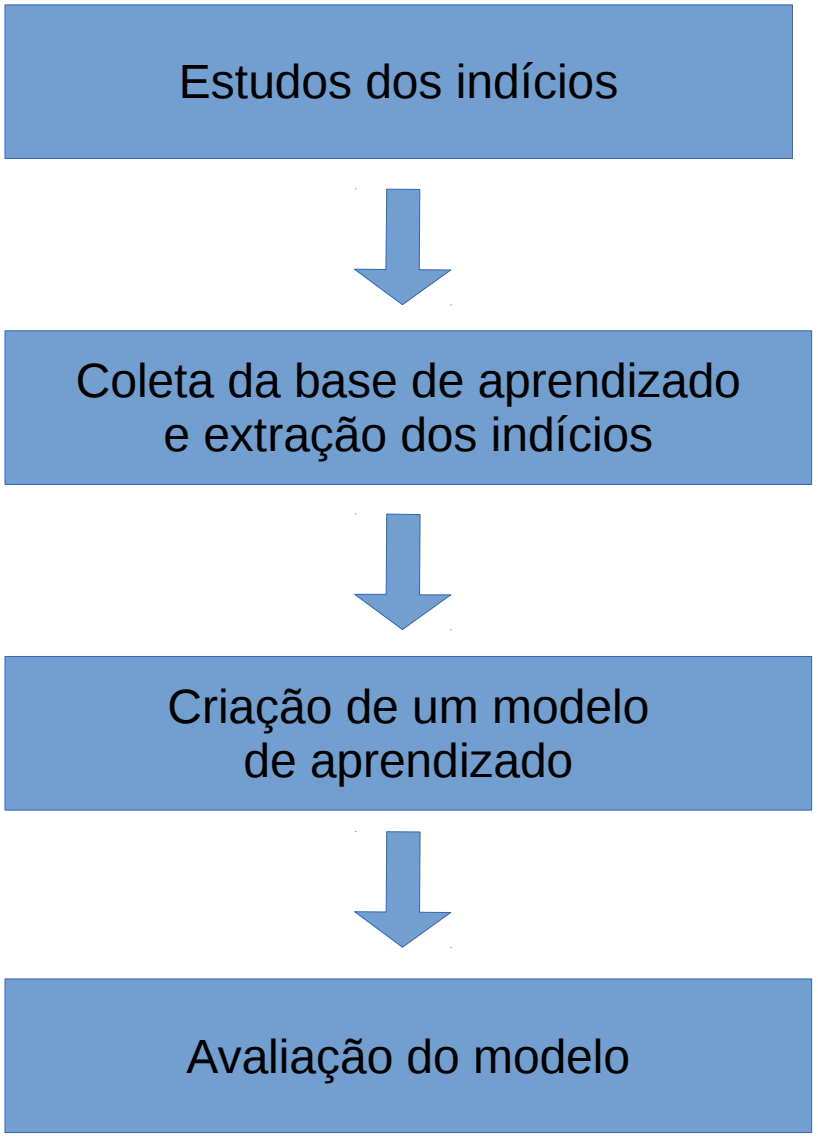
- Onde está a supervisão?

$$S = \{ (x_1,y_1), (x_2,y_2), \ldots (x_n,y_n)\}$$

- O fato de [y] ser conhecido para algumas entradas torna o problema supervisionado
  - y_i contínuo: Regressão
  - y_i categórico: Classificação

Valores finitos ex: "sim"/"nao", temas de um Texto (ciências, historia...)

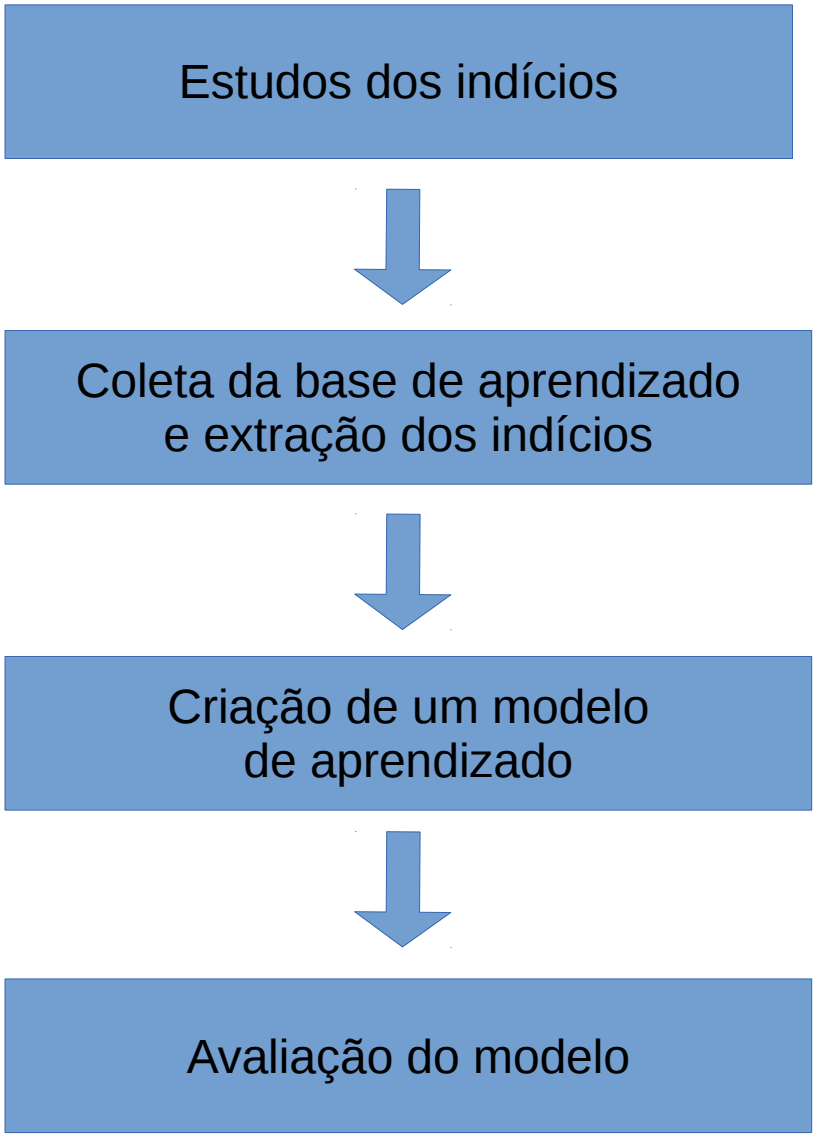
Processo de Aprendizado



Exemplo de Aplicação

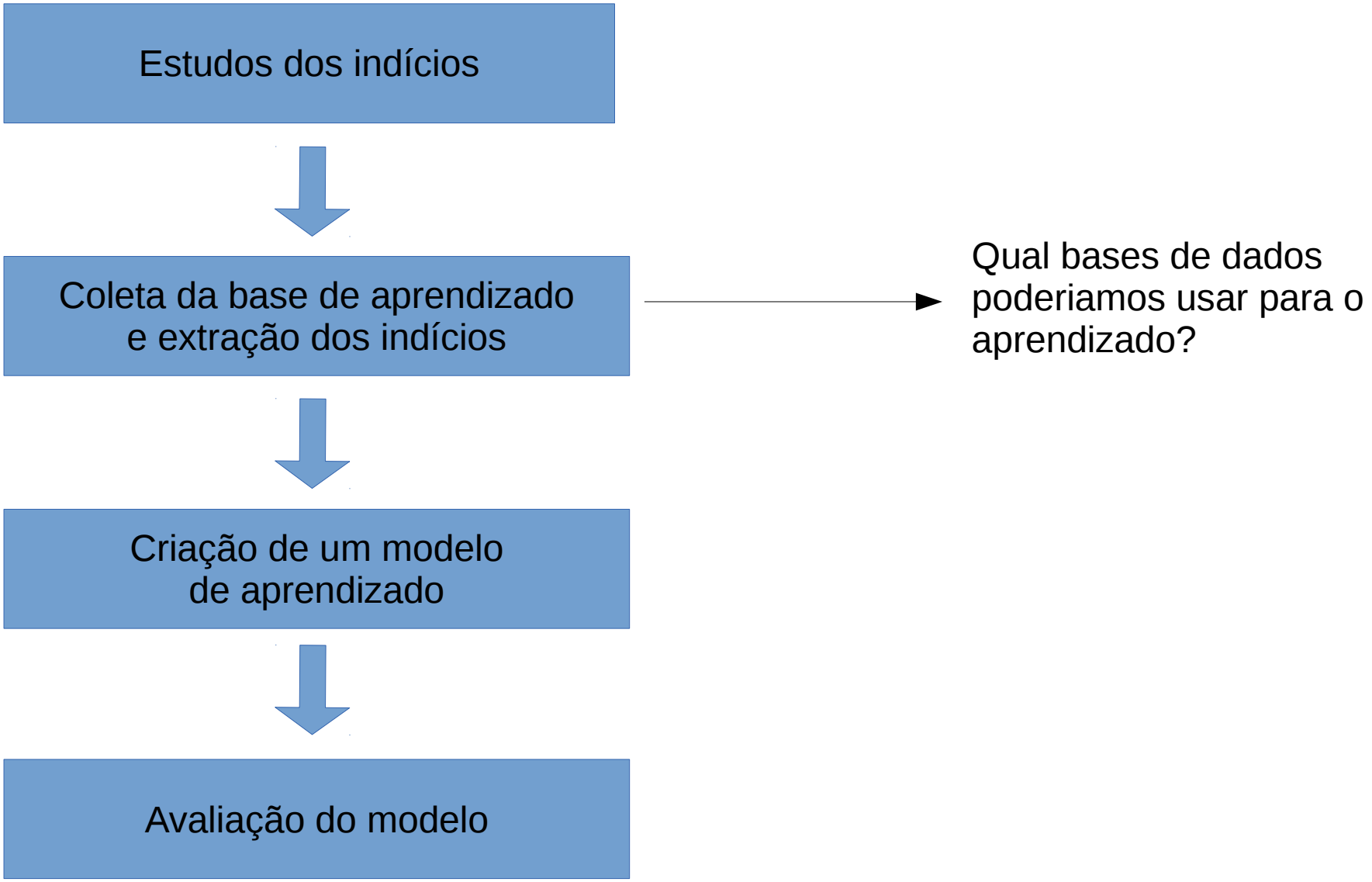
Prever resultados de jogos da Seleção Brasileira

Exemplo de Aplicação
Predição de Resultado de Jogos

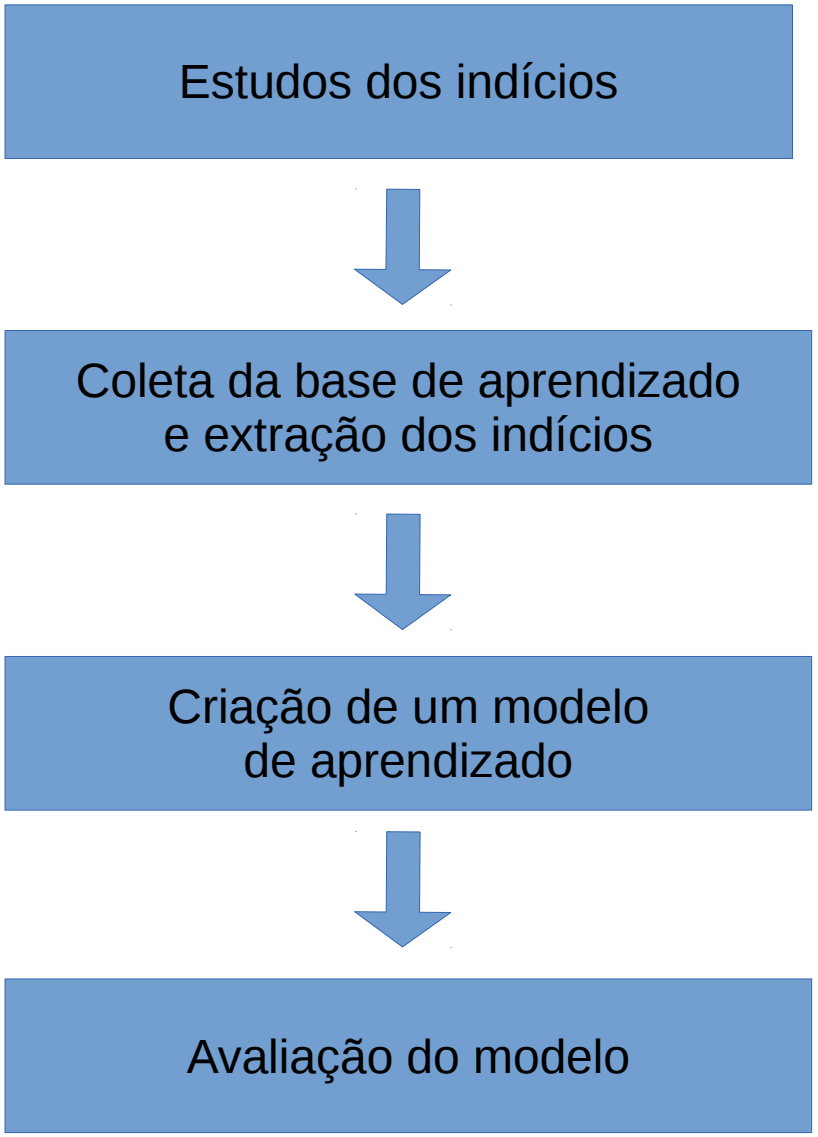


Dado uma partida do Brasil contra um time T, quais indícios seriam interessantes levar em consideração?

Exemplo de Aplicação
Predição de Resultado de Jogos



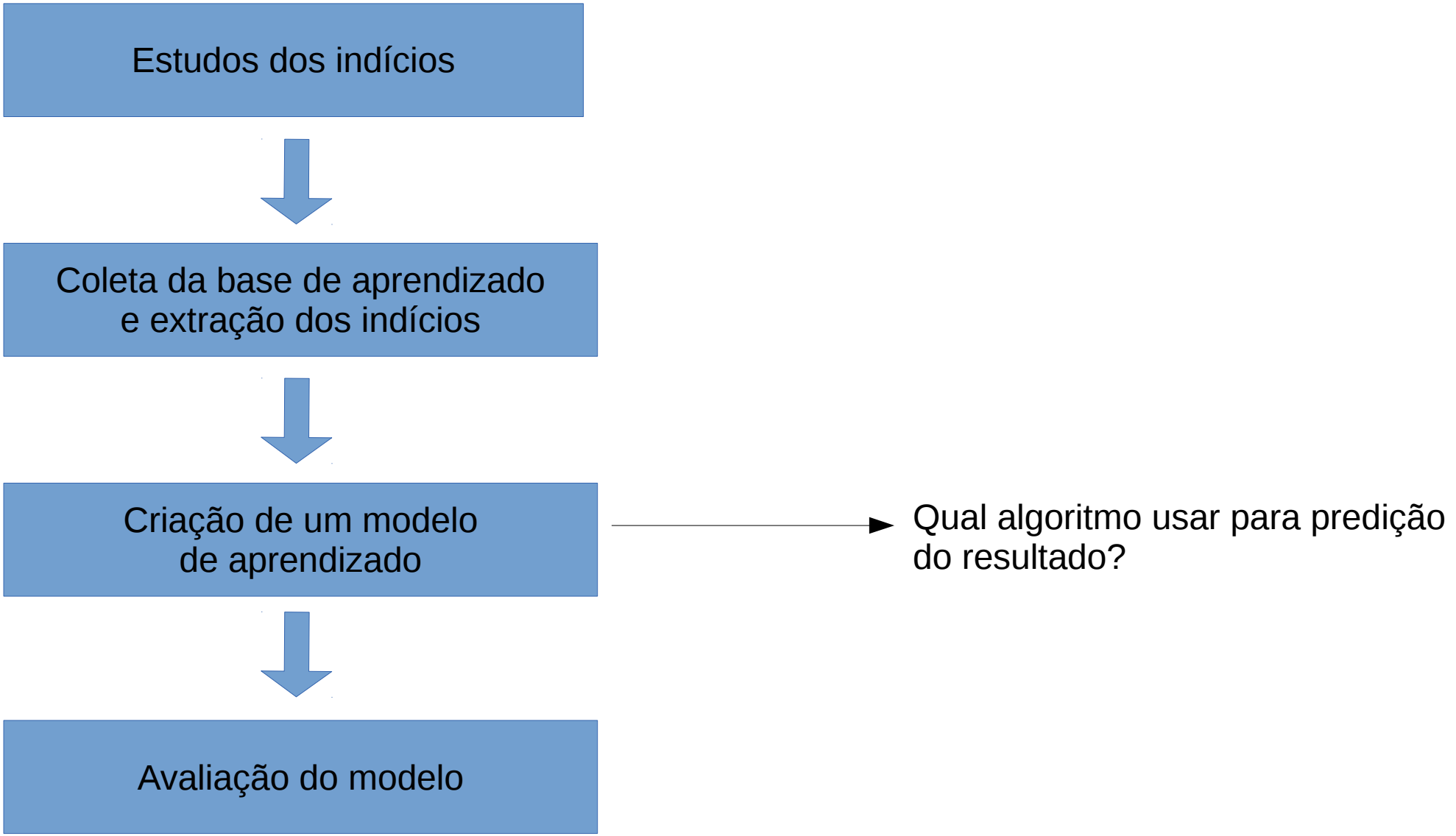
Exemplo de Aplicação
Predição de Resultado de Jogos



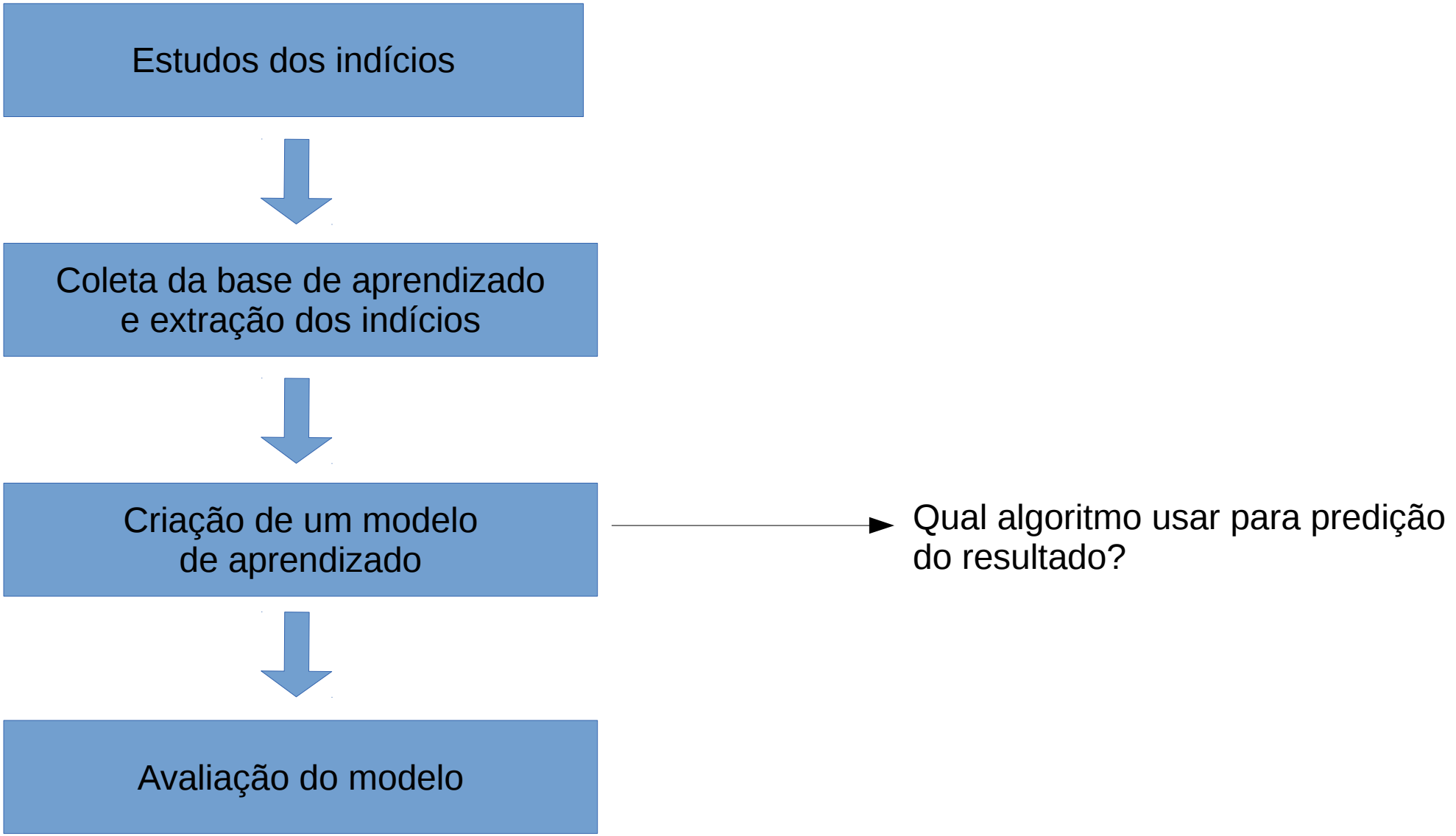
Qual bases de dados poderiamos usar para o aprendizado?

- Manual
- Wikipedia
- Extração de algumas páginas de resultados na Web:
  - Terra
  - Site da Fifa
  - http://www.futebol.com
  - http://ww.resultados.com

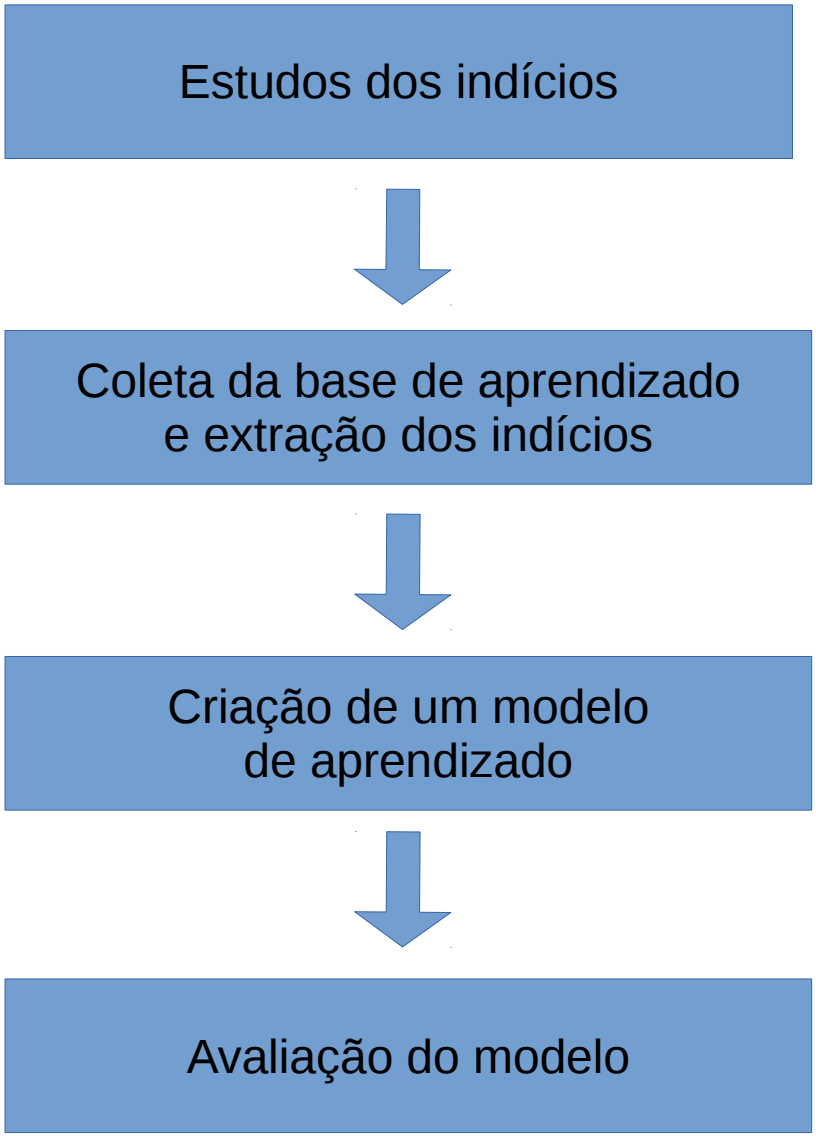
Exemplo de Aplicação
Predição de Resultado de Jogos



Exemplo de Aplicação
Predição de Resultado de Jogos



Exemplo de Aplicação
Predição de Resultado de Jogos



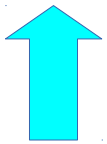
Exemplo de Aplicação
Predição de Resultado de Jogos

Árvore de Decisão:

Conjunto de treinamento (Experiência):

Partidas do Brasil contra outros adversários

Adversário	Adv. já foi Campeão?	Saldo de vitorias do Brasil	Reputação do Treinador Bras. atual	Resultado
Argentina	S	+	+	Vitoria
Chile	N	+	-	Derrota
Camarões	N	+	+	Vitória
Italia	S	-	+	Derrota



Cada adversário é representado por essas caracteristicas

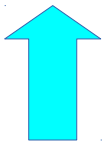
Exemplo de Aplicação
Predição de Resultado de Jogos

Árvore de Decisão:

Conjunto de treinamento (Experiência):

Partidas do Brasil contra outros adversários

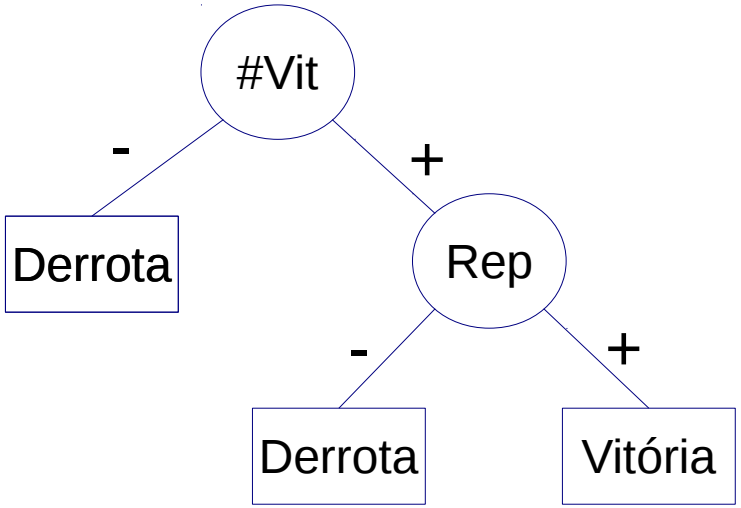
Adversário	Adv. já foi Campeão?	Saldo de vitorias do Brasil	Reputação do Treinador Bras. atual	Resultado
Argentina	S	+	+	Vitoria
Chile	N	+	-	Derrota
Camarões	N	+	+	Vitória
Italia	S	-	+	Derrota



Resultado da partida com cada adversário

Exemplo de Aplicação
Predição de Resultado de Jogos

Árvore de Decisão:



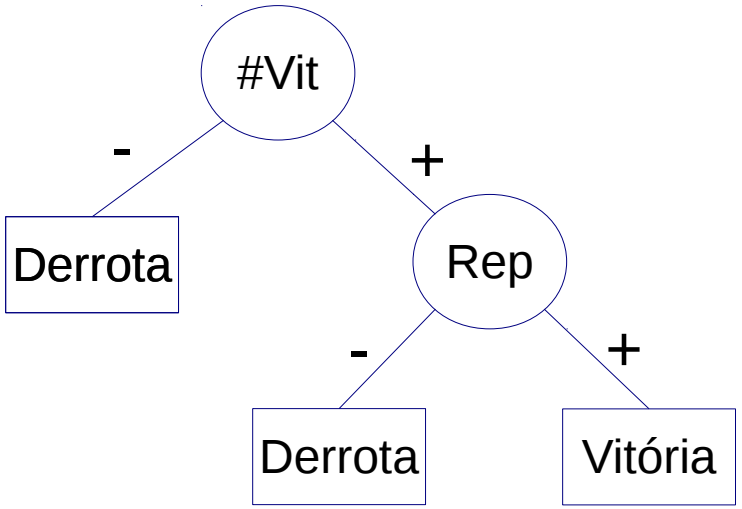
Adversário	Camp?	#Vit	Rep	Result
Argentina	S	+	+	V
Chile	N	+	-	D
Camarões	N	+	+	V
Italia	S	-	+	D

Exemplo de Aplicação
Predição de Resultado de Jogos

Árvore de Decisão:

Dado a seguinte entrada:

Adversário	Adv. já foi Campeão?	Saldo de vitorias do Brasil	Reputação do Treinador Bras. atual	Resultado
Alemanha	S	+	+	??

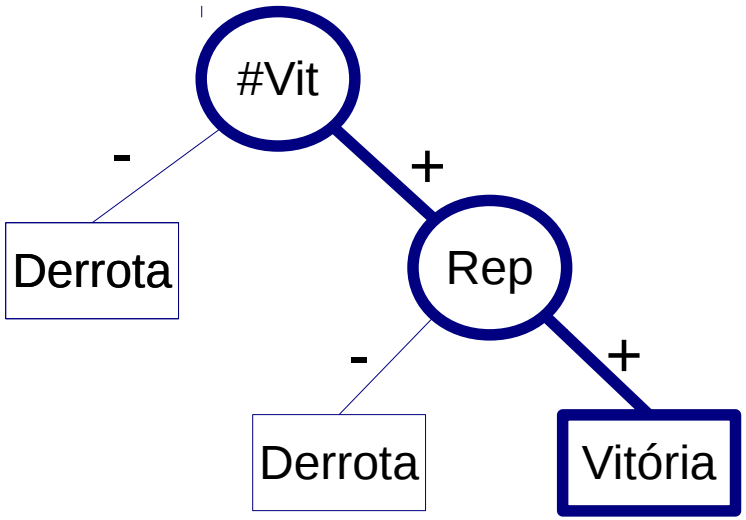


Exemplo de Aplicação
Predição de Resultado de Jogos

Árvore de Decisão:

Dado a seguinte entrada:

Adversário	Adv. já foi Campeão?	Saldo de vitorias do Brasil	Reputação do Treinador atual	Resultado
Alemanha	S	+	+	??



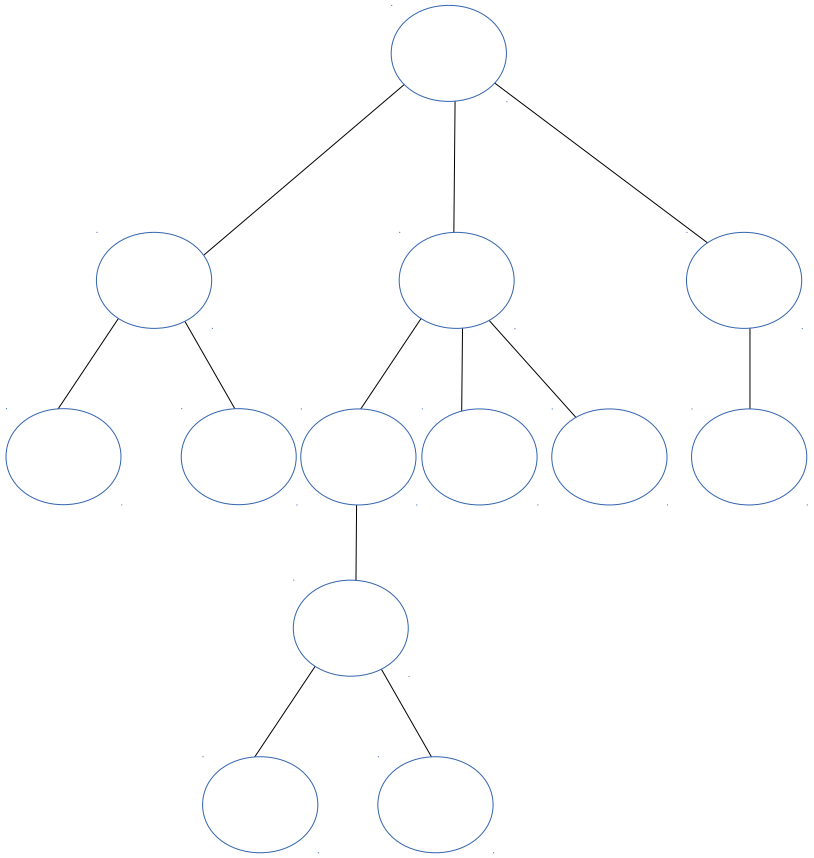
Caminhamento em Grafos

Caminhamento em Grafos

- Existem vários algoritmos em grafos
- Dependendo dele, necessitamos percorrer o grafo de forma diferente
- As principais formas de percorrer um grafo são:
  - busca em largura
  - busca em profundidade
- Dependendo do problema, será necessário usar um desses tipos de caminhamento

Busca em Largura – em uma Árvore Enraizada

A “fronteira” entre os vértices visitados e não visitados é expandida em largura (Cormen et. al., 2001):

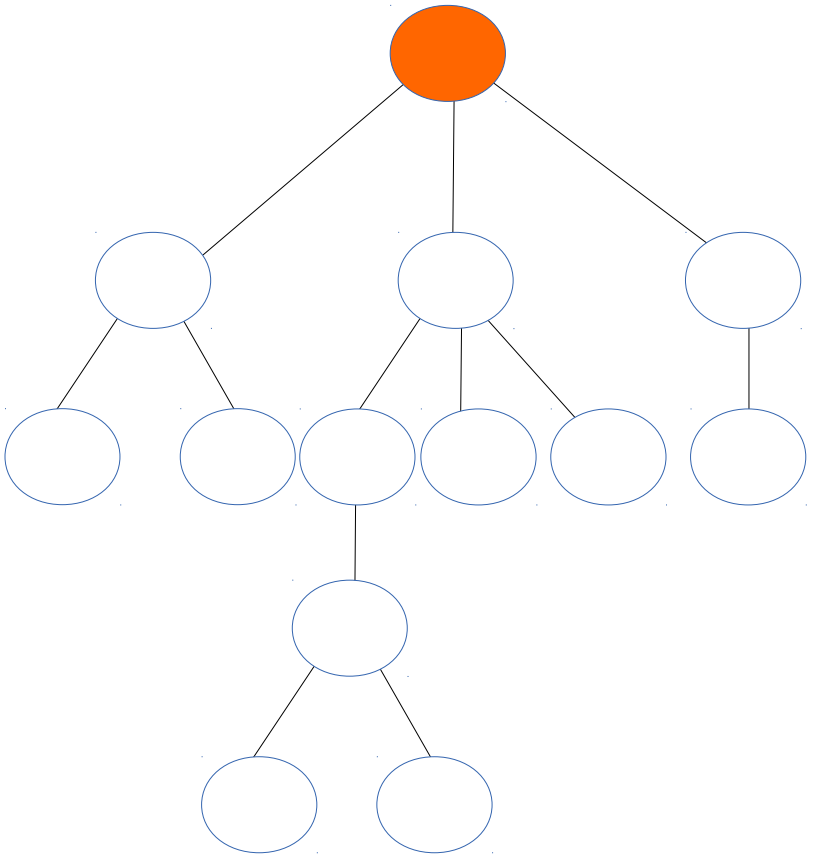


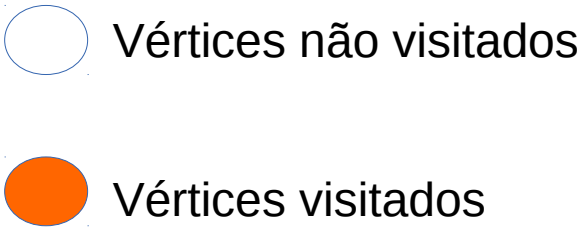
Vértices não visitados

Vértices visitados

Busca em Largura – em uma Árvore Enraizada

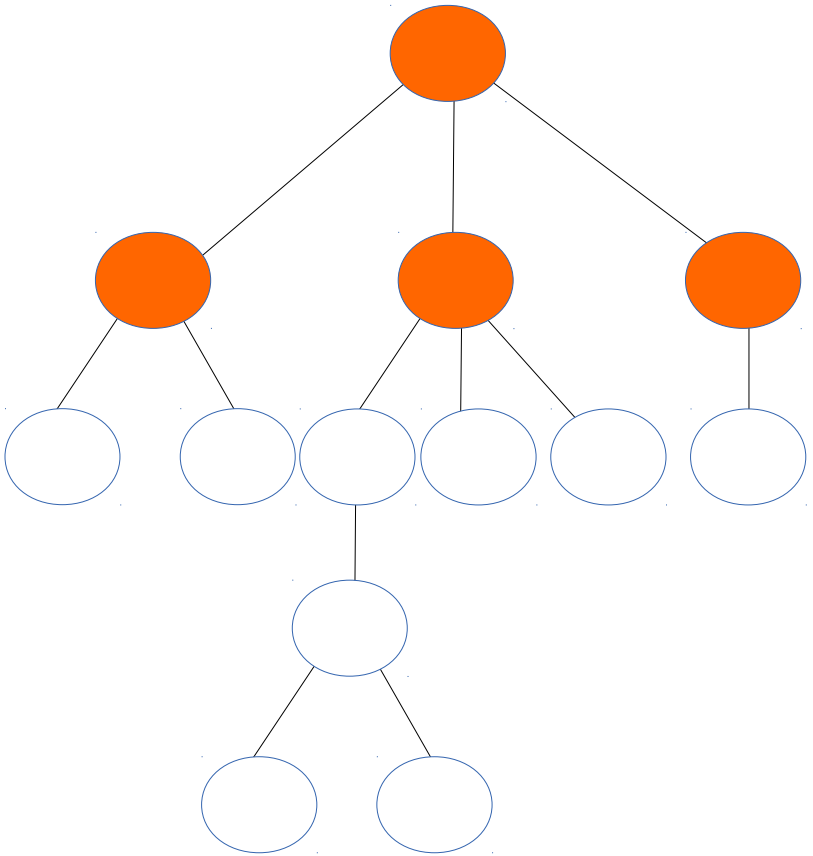
A “fronteira” entre os vértices visitados
e não visitados é expandida em largura (Cormen et. al., 2001):

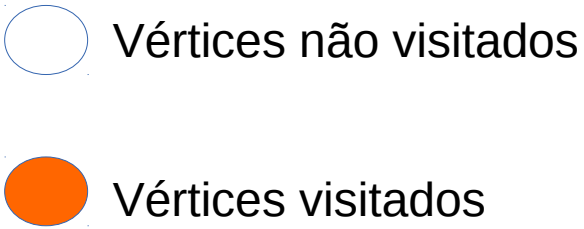




Busca em Largura – em uma Árvore Enraizada

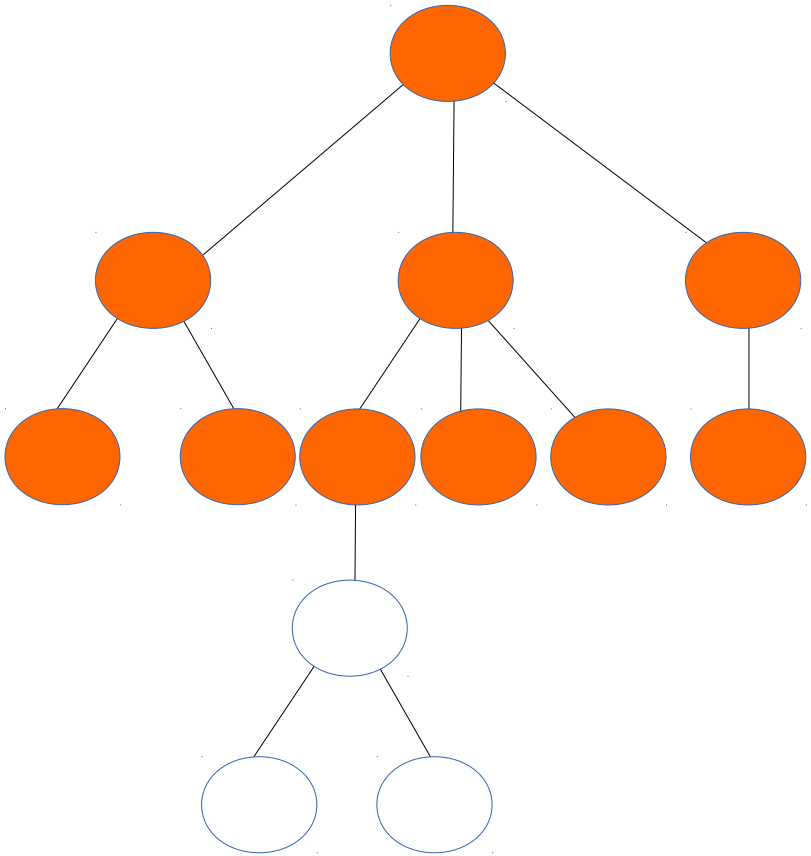
A “fronteira” entre os vértices visitados
e não visitados é expandida em largura (Cormen et. al., 2001):

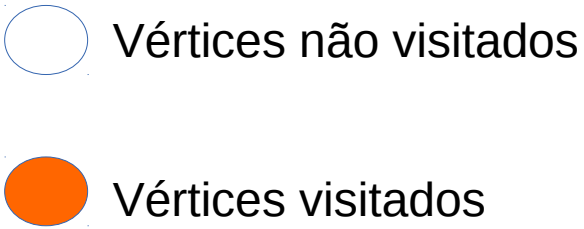




Busca em Largura – em uma Árvore Enraizada

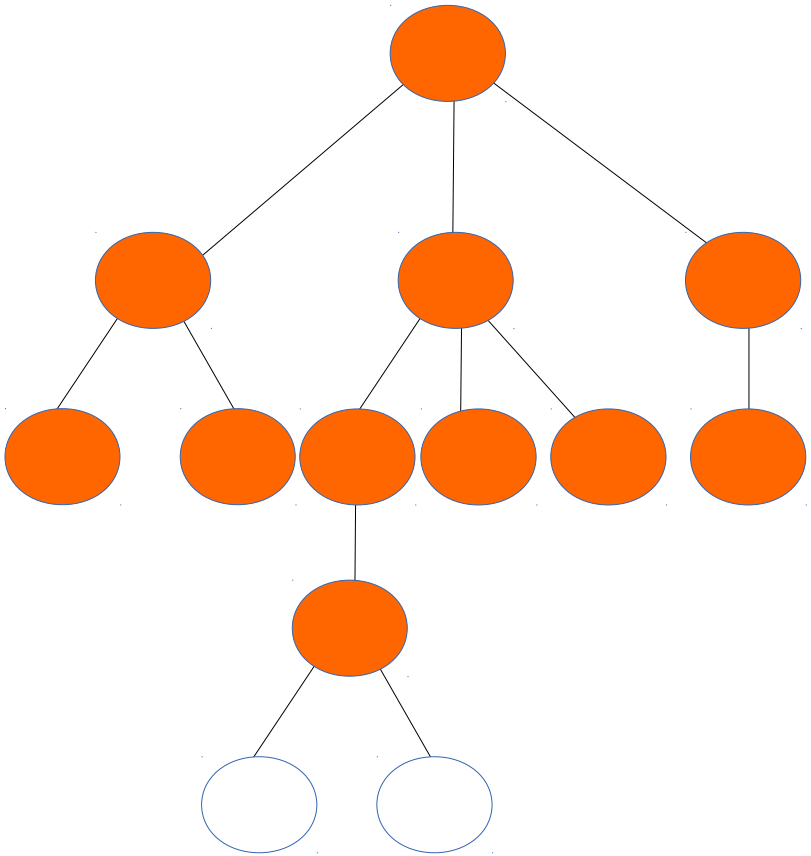
A “fronteira” entre os vértices visitados
e não visitados é expandida em largura (Cormen et. al., 2001):

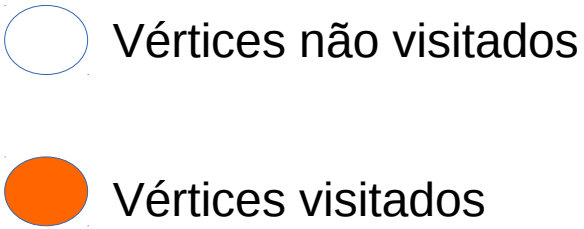




Busca em Largura – em uma Árvore Enraizada

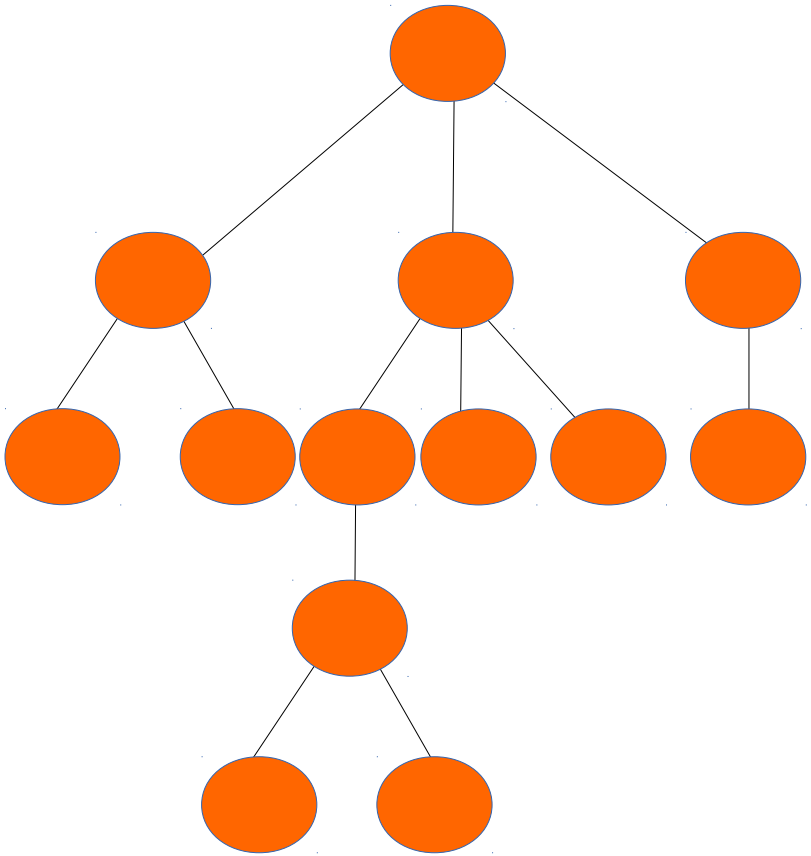
A “fronteira” entre os vértices visitados
e não visitados é expandida em largura (Cormen et. al., 2001):

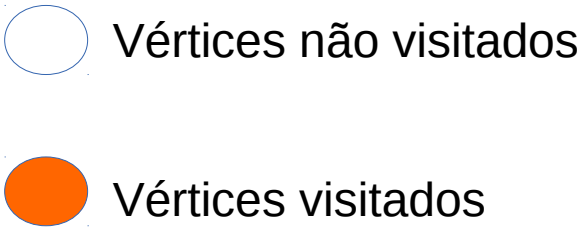




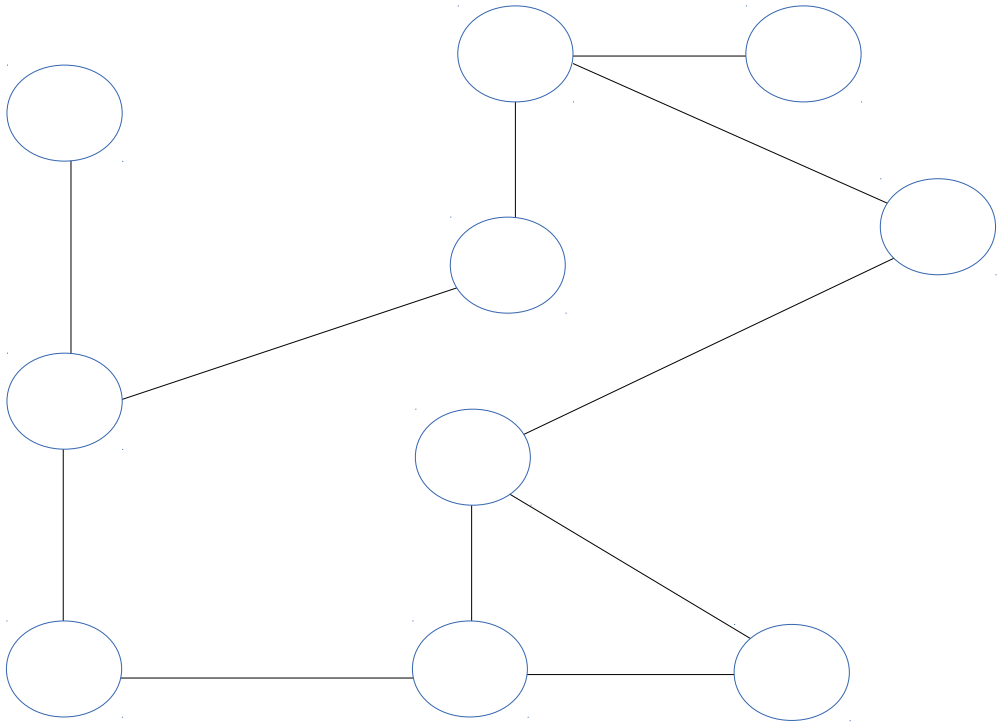
Busca em Largura – em uma Árvore Enraizada

A “fronteira” entre os vértices visitados
e não visitados é expandida em largura (Cormen et. al., 2001):





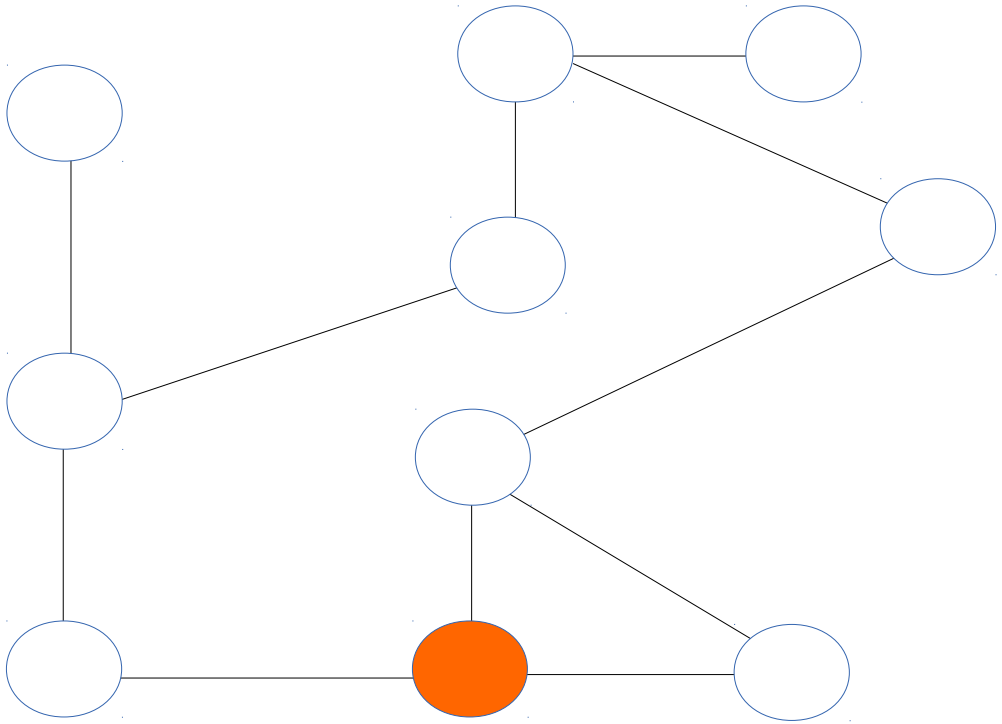
Busca em largura – Grafo qualquer



Vértices não visitados

Vértices visitados

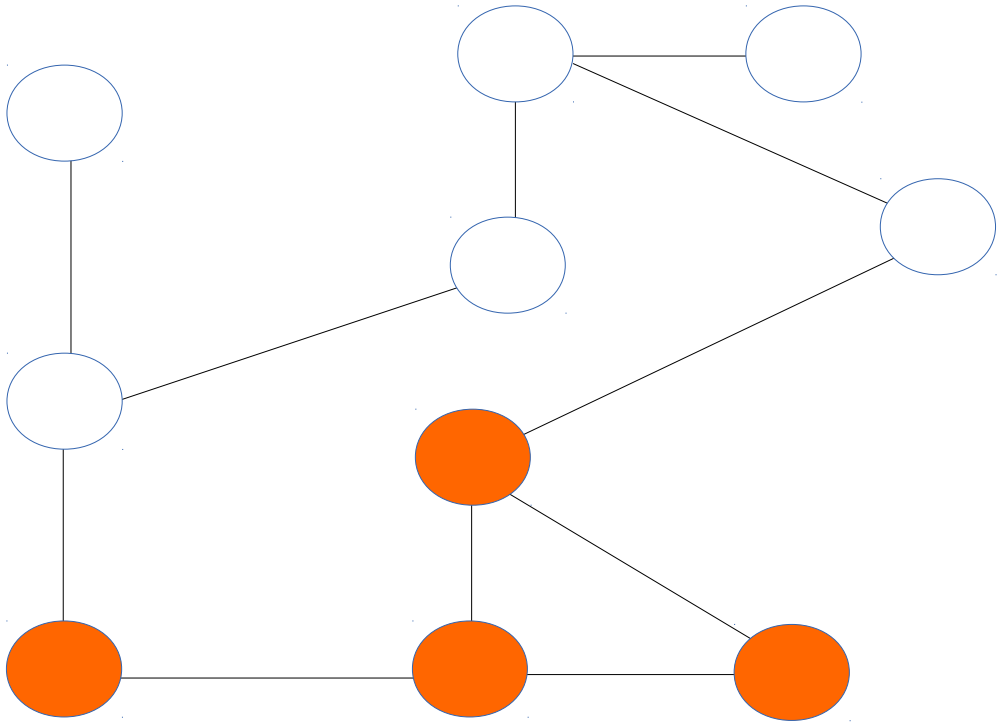
Busca em largura – Grafo qualquer



Vértices não visitados

Vértices visitados

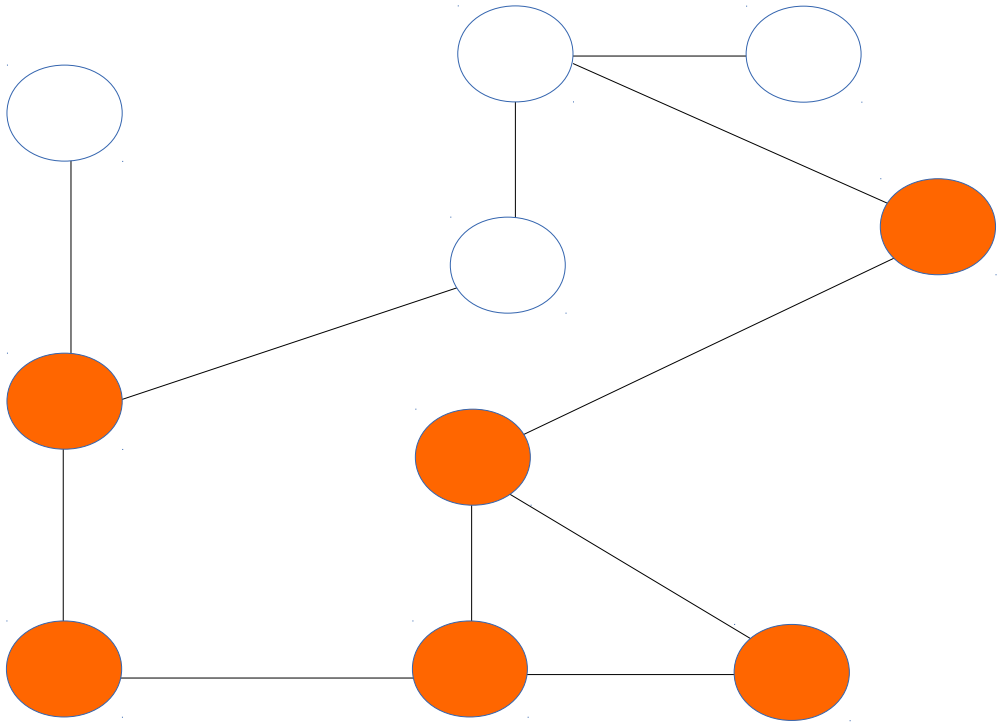
Busca em largura – Grafo qualquer



Vértices não visitados

Vértices visitados

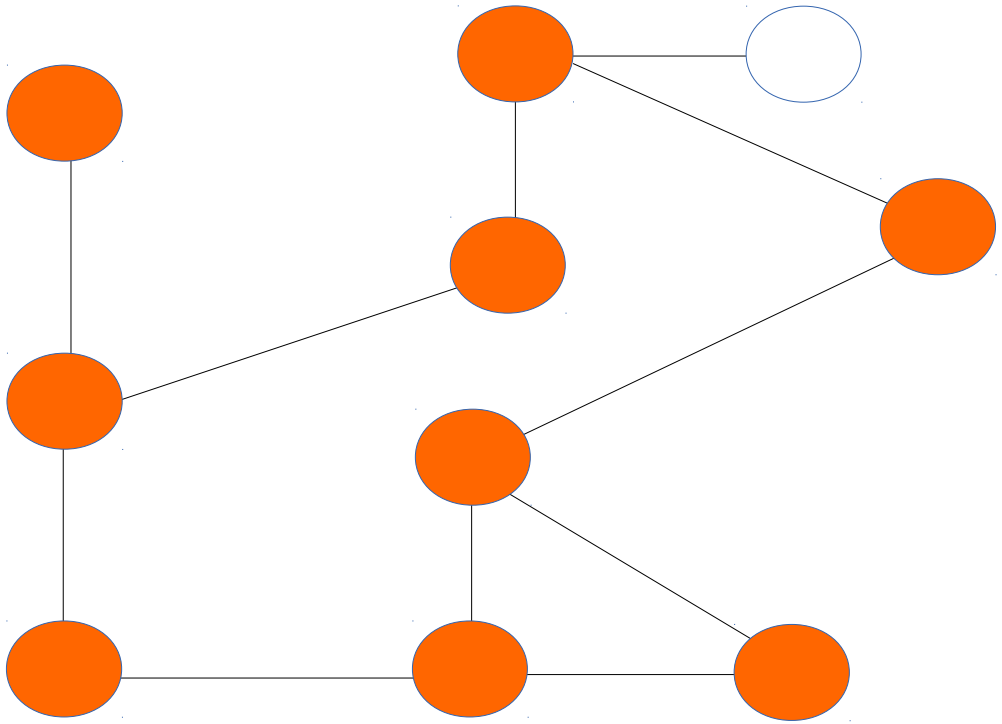
Busca em largura – Grafo qualquer



Vértices não visitados

Vértices visitados

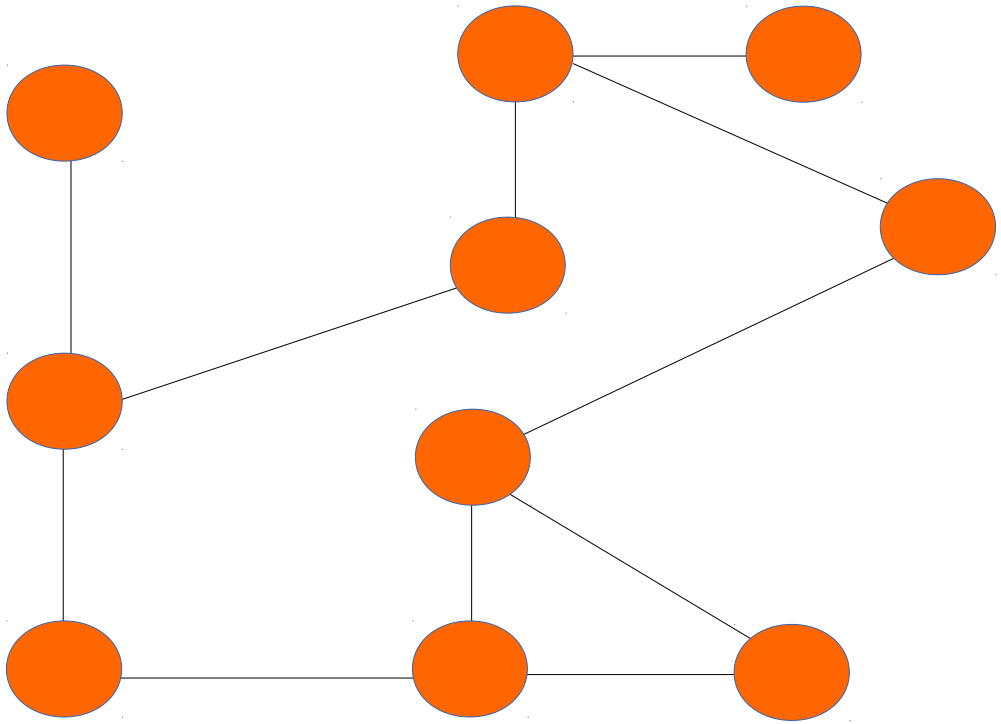
Busca em largura – Grafo qualquer



Vértices não visitados

Vértices visitados

Busca em largura – Grafo qualquer



Vértices não visitados

Vértices visitados

Estrutura de Dados
Pilhas e Filas

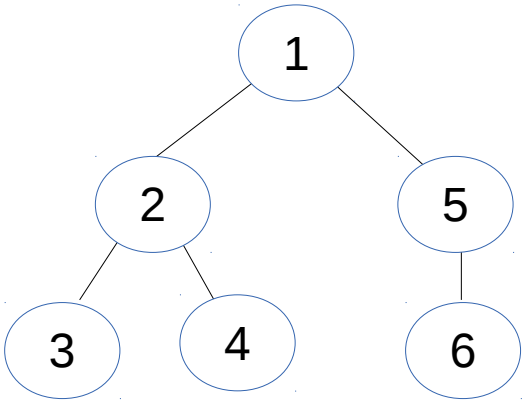
- Estrutura de dados em formato de lista que possui duas operações básicas:
  - push: adiciona um elemento
  - pop: retira elemento
- Pilha
  - Estrutura em que **último elemento a entrar**, é o **primeiro a sair** (*LIFO,* do inglês, *Last in, First out*)
  - Exemplo: caso tenha sido adicionado os elementos (nesta ordem): A, B, C, D, E. Os elementos serão retirados na seguinte ordem: E, D, C, B, A
  - As operações **push** e **pop** podem ser chamadas de **empilha** e **desempilha**, respectivamente.

Estrutura de Dados
Pilhas e Filas

- Estrutura de dados em formato de lista que possui duas operações básicas:
  - push: adiciona um elemento
  - pop: retira elemento
- Fila
  - Estrutura em que **primeiro elemento a entrar**, é o **primeiro a sair** (F*IFO, do inglês, First in, First out*)
  - Exemplo: caso tenha sido adicionado os elementos (nesta ordem): A, B, C, D, E. Os elementos serão retirados na seguinte ordem: A, B, C, D, E
  - As operações **push** e **pop** podem ser chamadas de **enfileira** e **desenfileira**, respectivamente.

Busca em Largura – Algoritmo

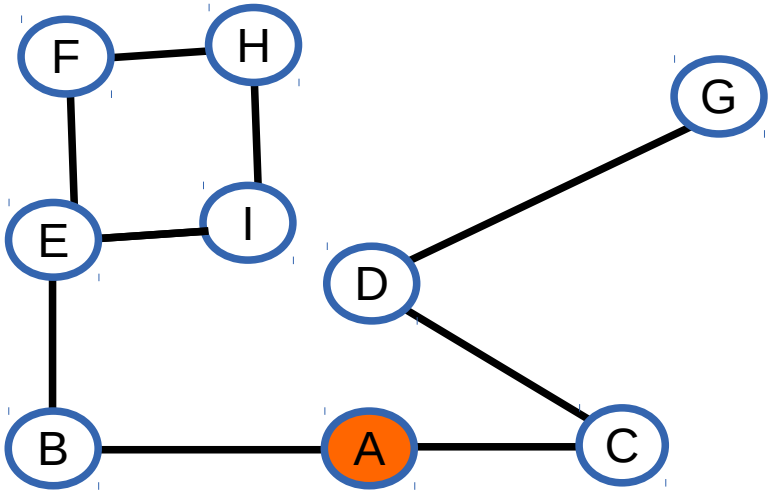
Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q



para cada u ∈ V[G] **faça**:
visit[u] ← false
visit[s] ← true
Q ←{s}
enquanto Q ≠Ø **faça:**
u ← desenfileira(Q)
para cada v ∈ adj[u] **faça:**
se visit[v] = false **então**:
visit[v] = true
enfileira(Q,v)

Busca em Largura – Algoritmo

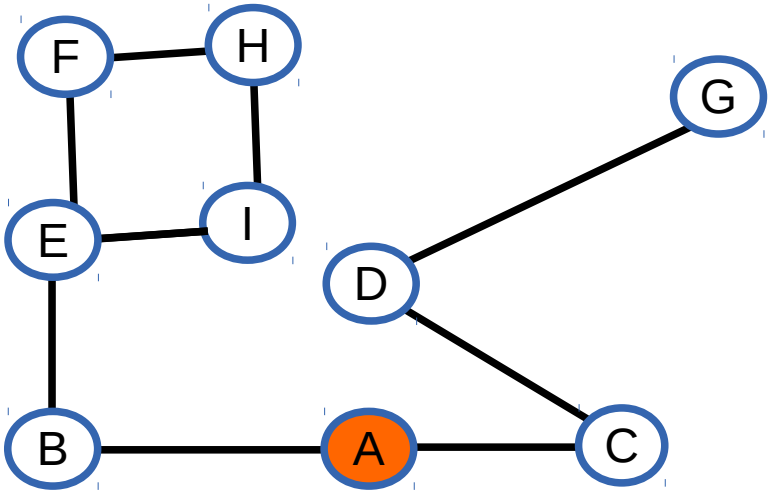
- Seja o grafo direcionado abaixo, em que suas arestas não são ponderadas. Qual é a distância do caminho mínimo entre o vértice A e os demais?



Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A

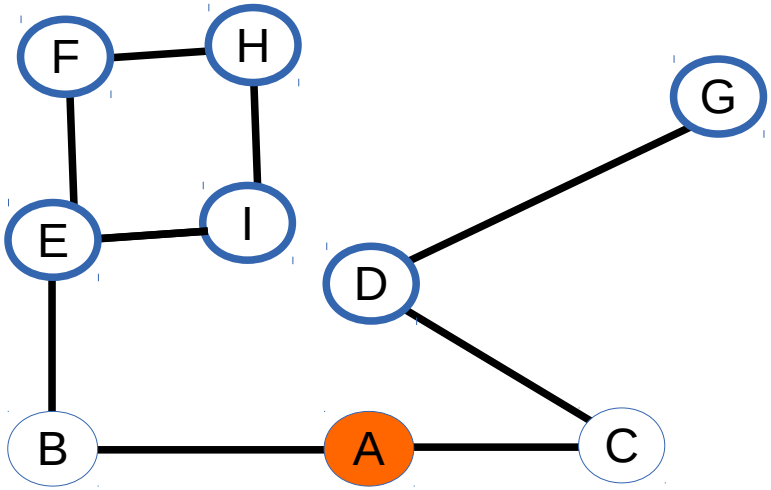
Fila Q

∞	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A

Fila Q

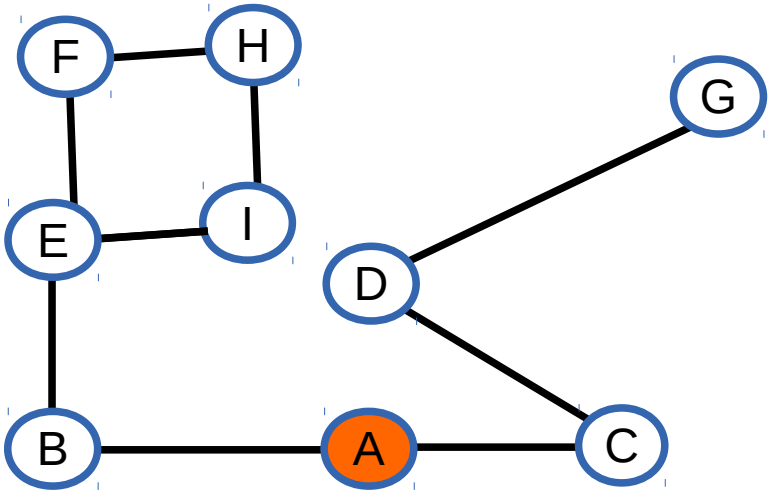
A

∞	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A

Fila Q

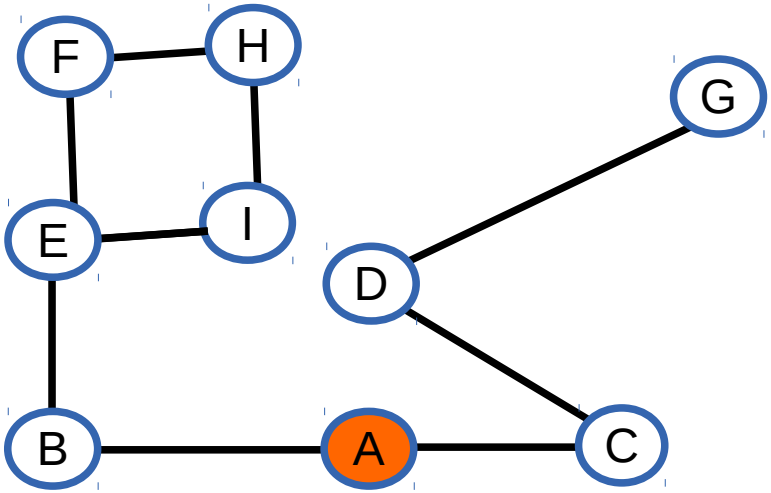
A

0	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A

Fila Q

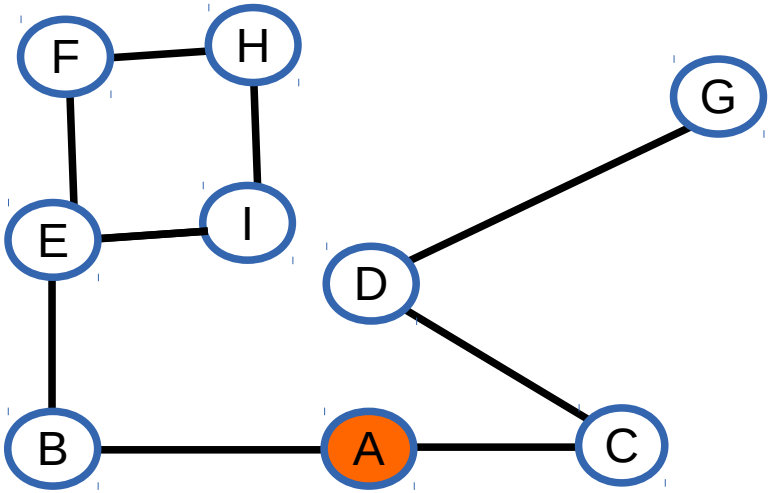
A

0	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
    u ← desenfileira(Q)
    para cada v ∈ adj[u] faça:
        se visit[v] = false então:
            dist[v] = dist[u] + 1
            visit[v] = true
            enfileira(Q,v)
```



s: A u: A

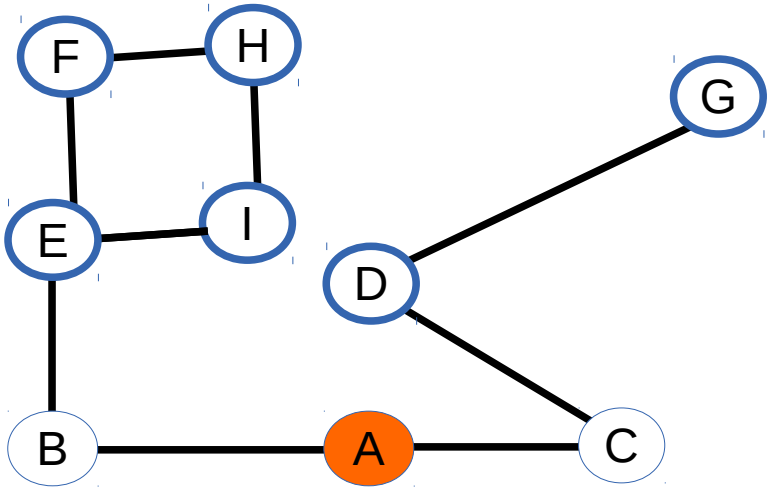
Fila Q

0	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: A v:

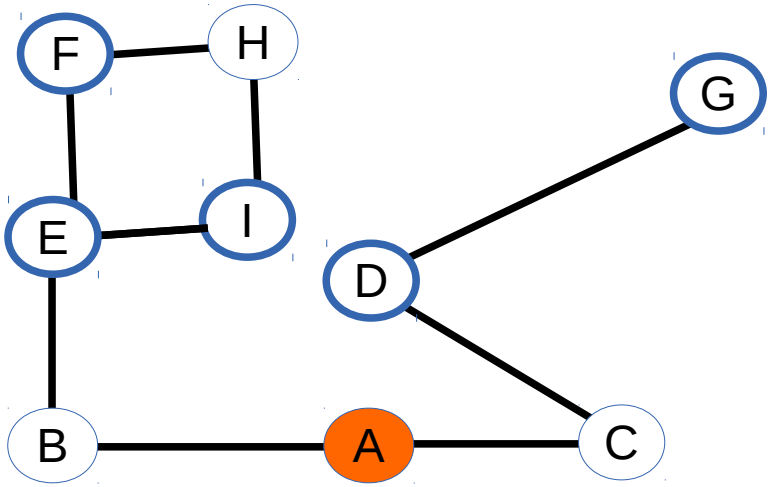
Fila Q

0	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: A v: B

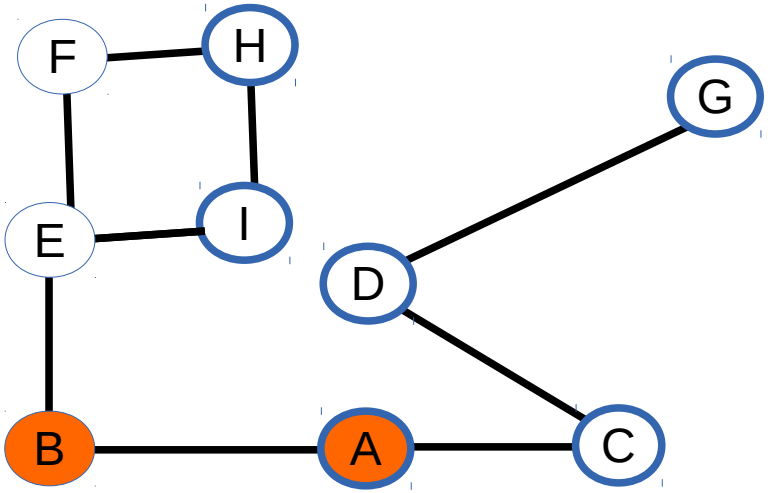
Fila Q

0	∞	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



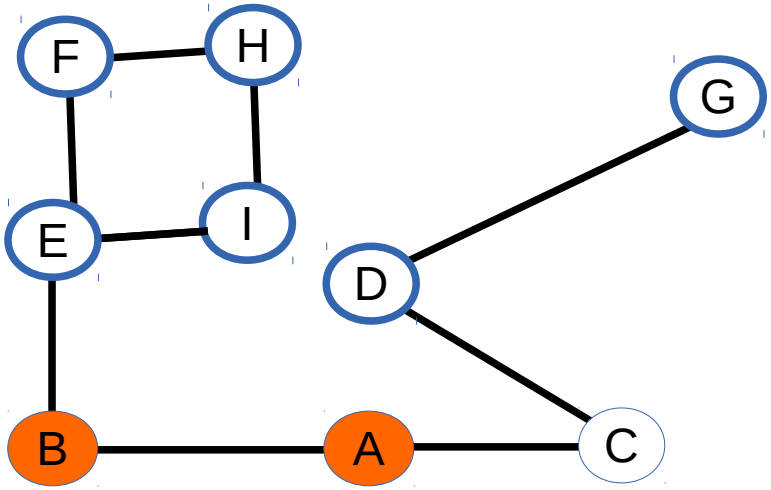
s: A u: A v: B

Fila Q

0	1	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: A v: B

Fila Q

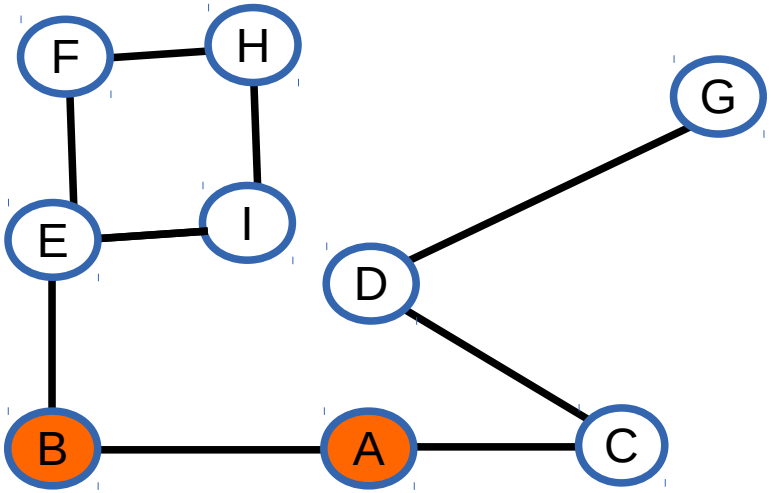

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: A v: C

Fila Q

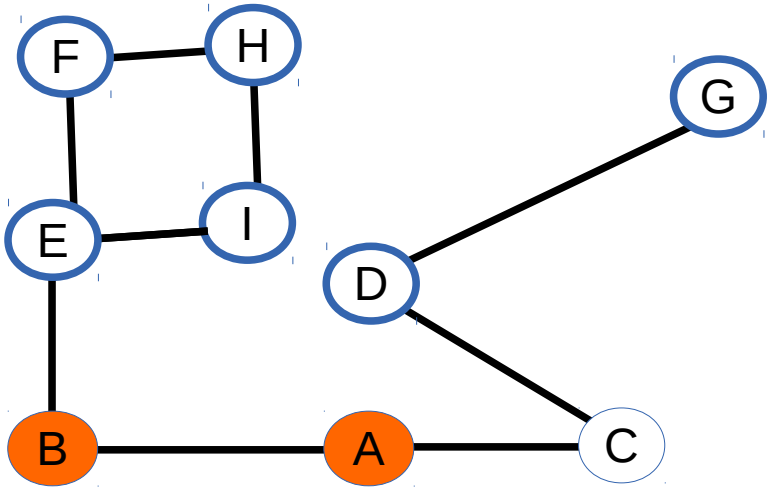
B

0	1	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: A v: C

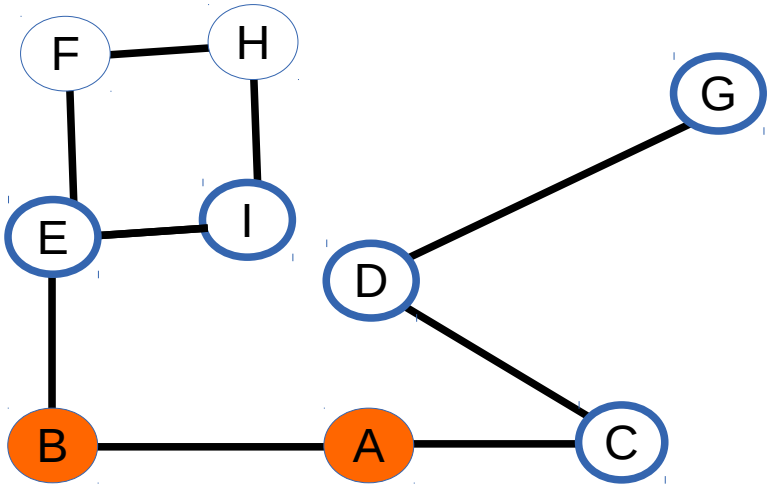
Fila Q

0	1	∞	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A u: A v: C

Fila Q

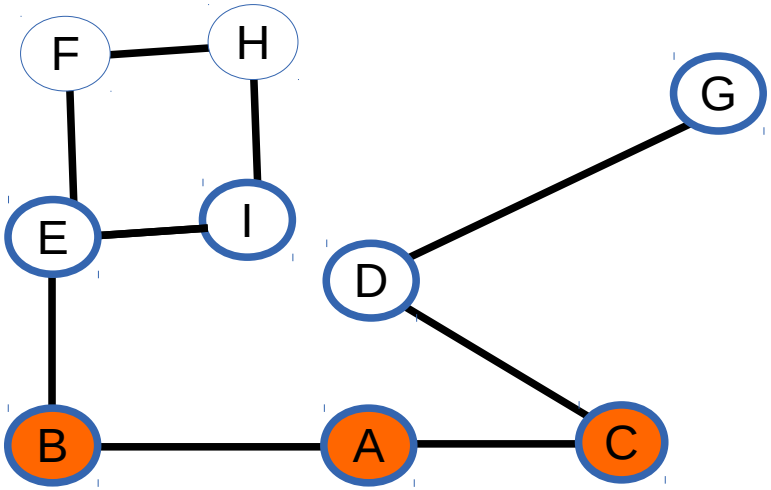
B

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: A v: C

Fila Q

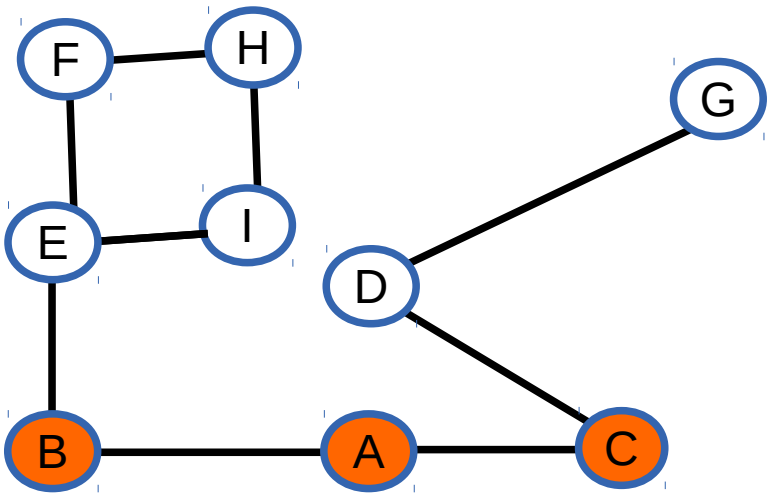
B

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A u: A v: C

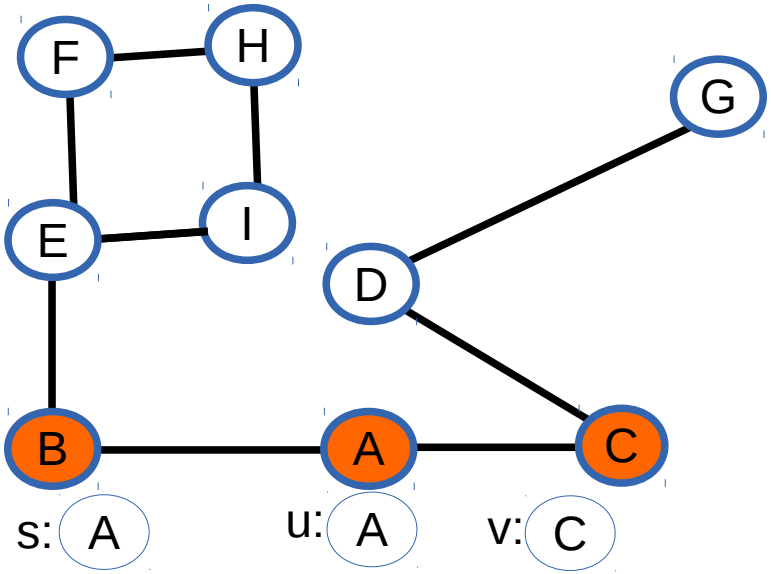
Fila Q

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



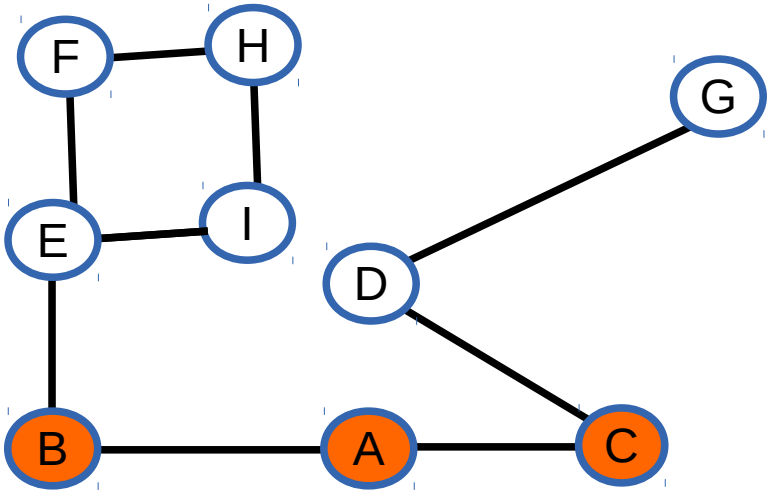
Fila Q

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: A v: D

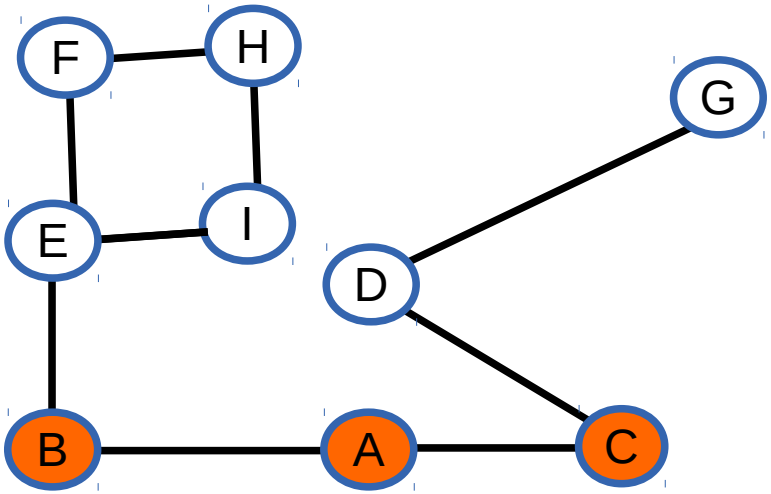
Fila Q

B C

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**




```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
    u ← desenfileira(Q)
    para cada v ∈ adj[u] faça:
        se visit[v] = false então:
            dist[v] = dist[u] + 1
            visit[v] = true
            enfileira(Q,v)
```

s: A u: B v:

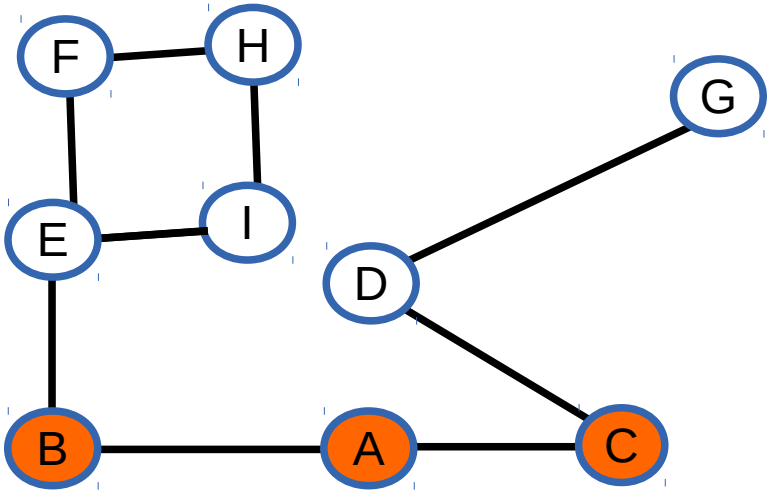
Fila Q

C

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**




```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```

s: A u: B v: E

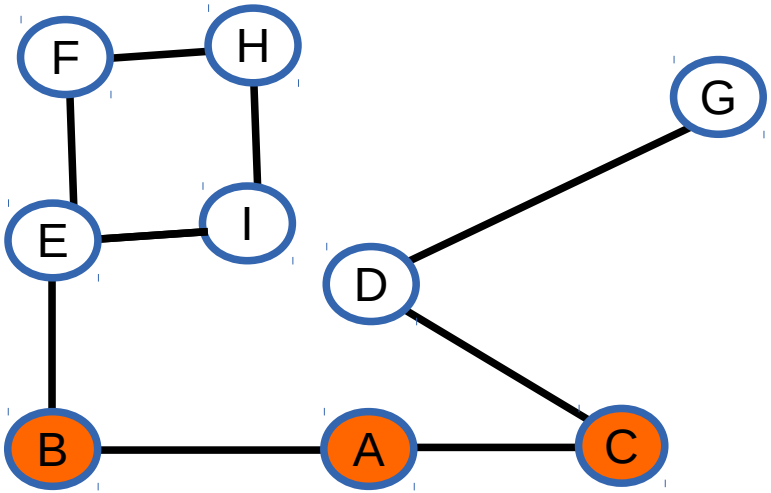
Fila Q

C

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: B v: E

Fila Q

C

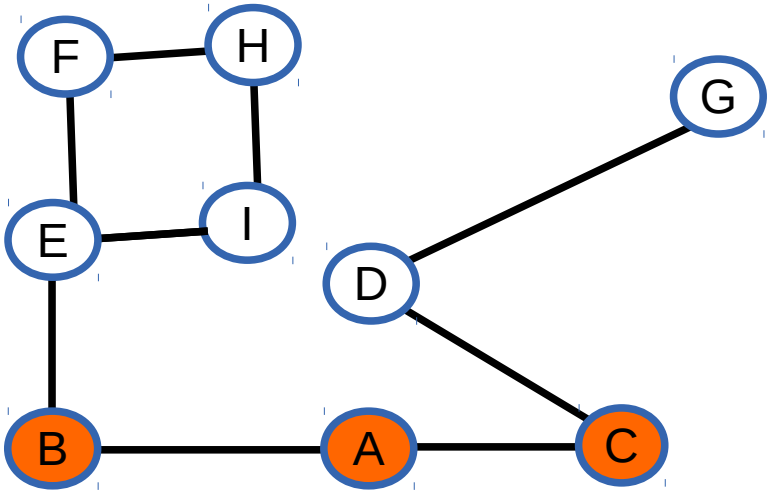
```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```

0	1	1	∞	∞	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A u: B v: E

Fila Q

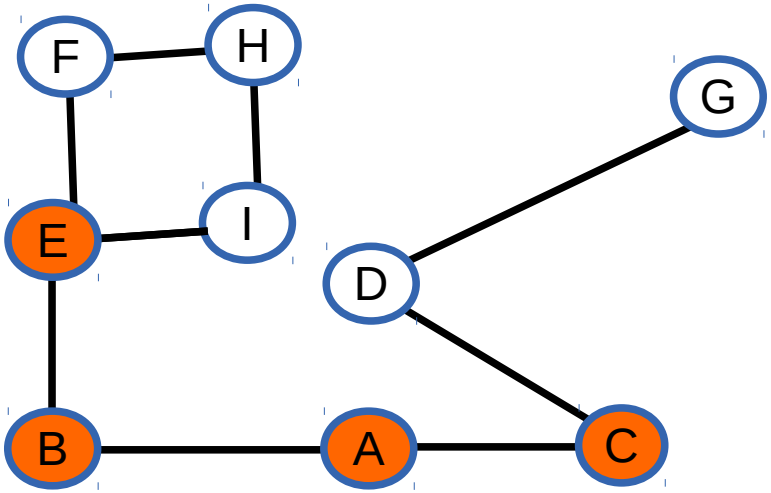
C

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: B v: E

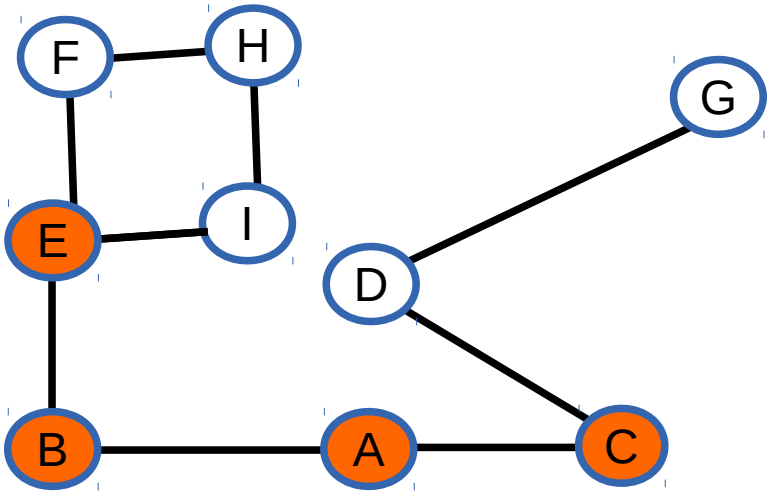
Fila Q

C

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: B v: E

Fila Q

C E

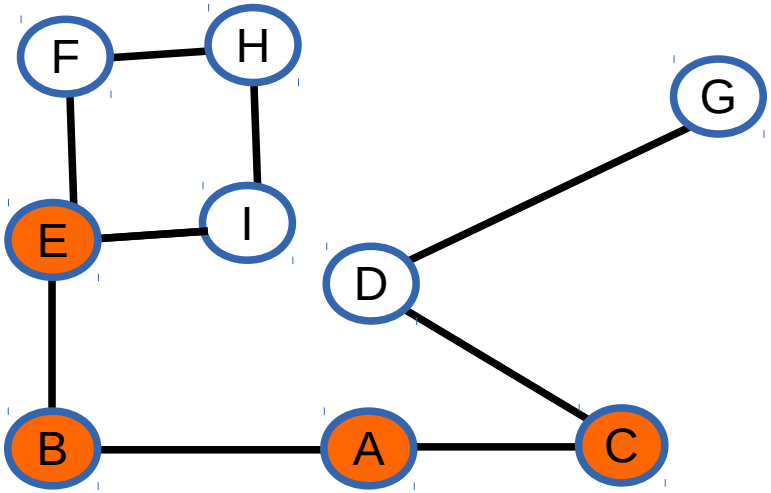
```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: B v: E

Fila Q

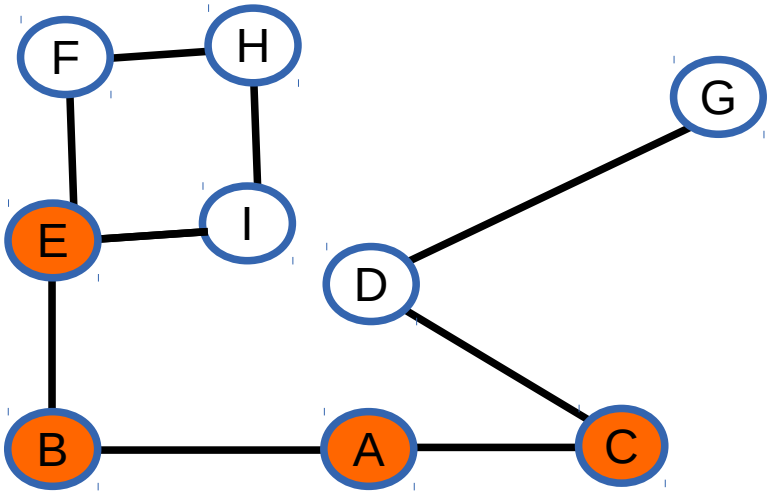
C E

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
    u ← desenfileira(Q)
    para cada v ∈ adj[u] faça:
        se visit[v] = false então:
            dist[v] = dist[u] + 1
            visit[v] = true
            enfileira(Q,v)
```



s: A u: C v:

Fila Q

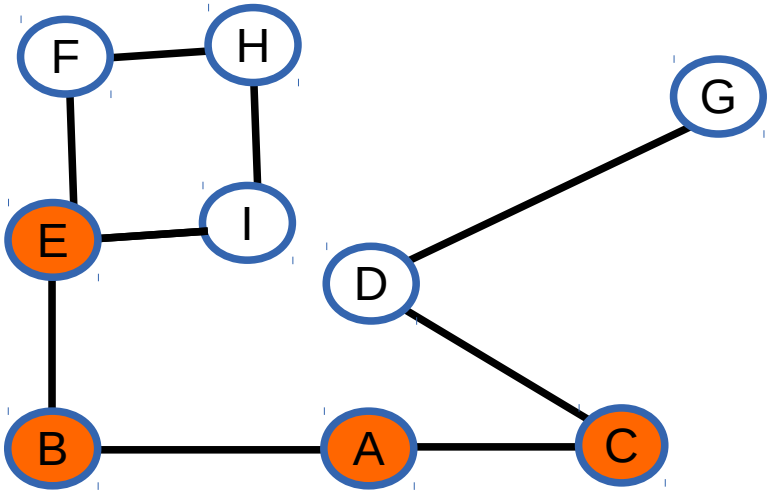
E

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: C v: D

Fila Q

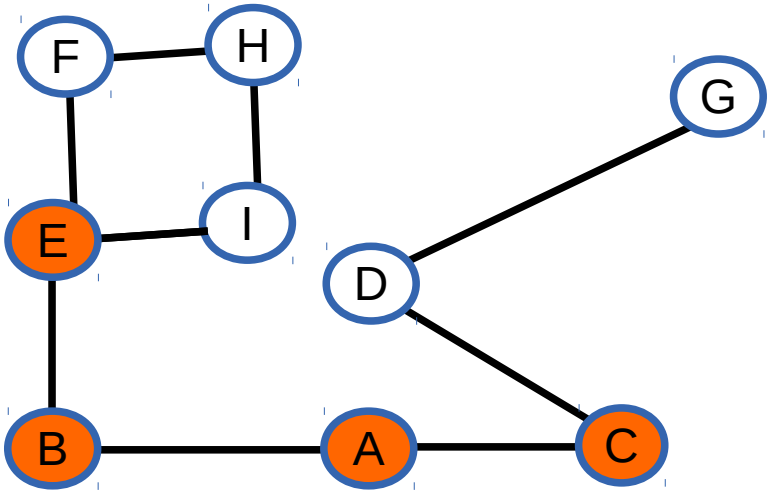
E

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: C v: D

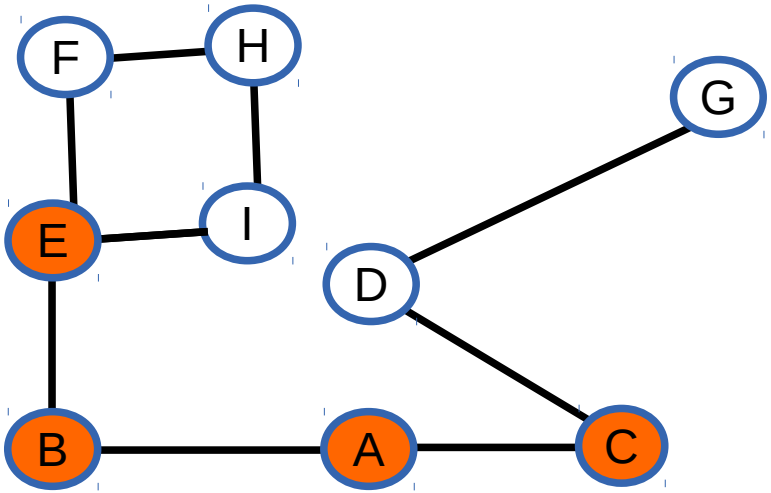
Fila Q

E

0	1	1	∞	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: C v: D

Fila Q

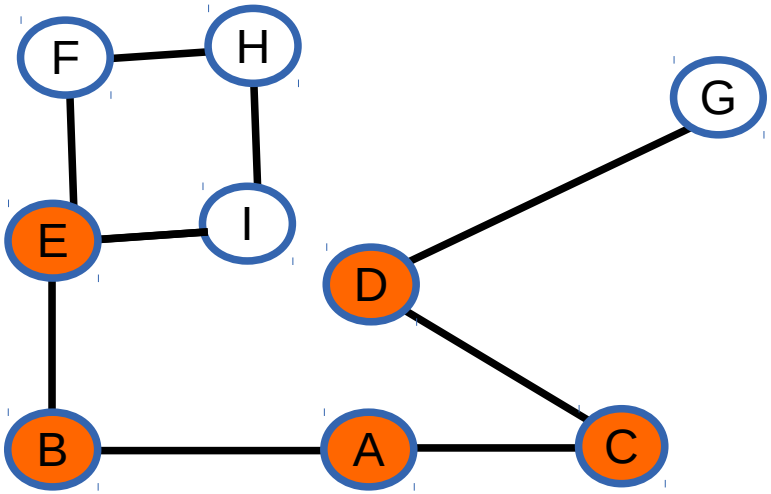
E

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: C v: D

Fila Q

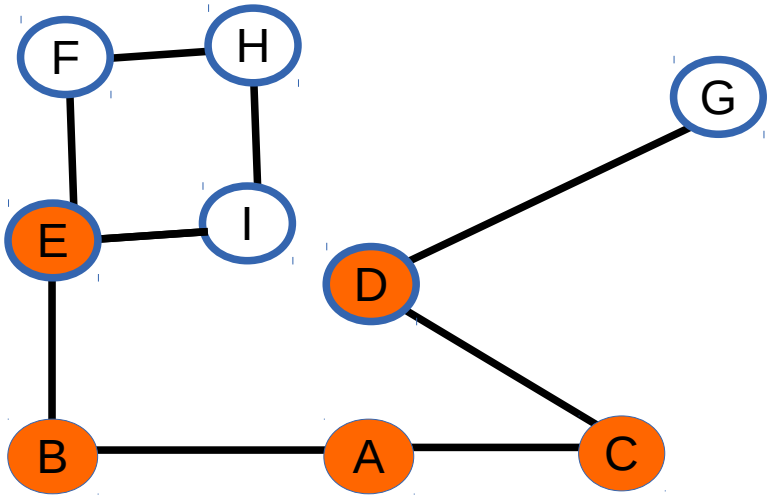
E

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: C v: D

Fila Q

E D

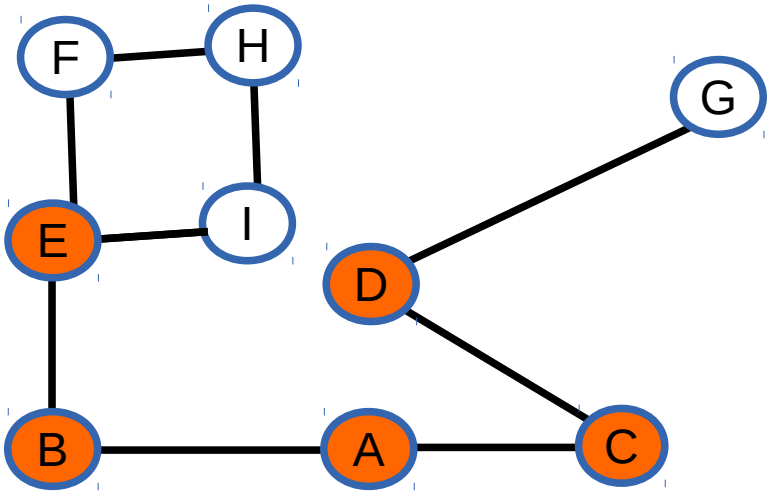
```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
    u ← desenfileira(Q)
    para cada v ∈ adj[u] faça:
        se visit[v] = false então:
            dist[v] = dist[u] + 1
            visit[v] = true
            enfileira(Q,v)
```



s: A u: E

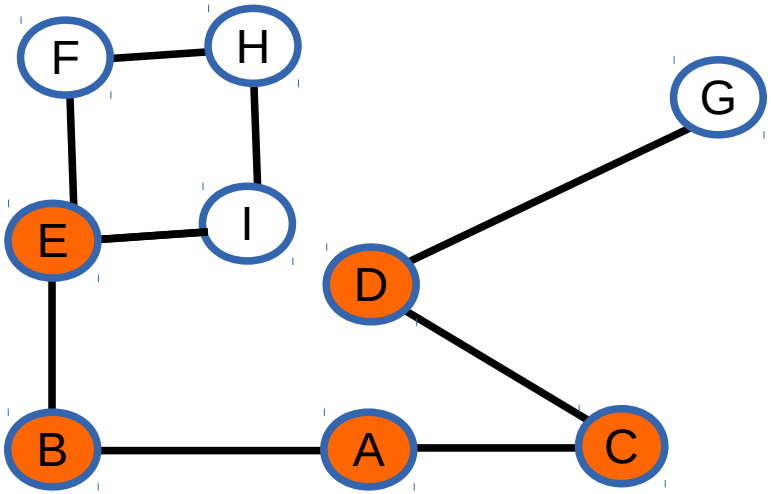
Fila Q

0	1	1	2	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: E v: I

Fila Q

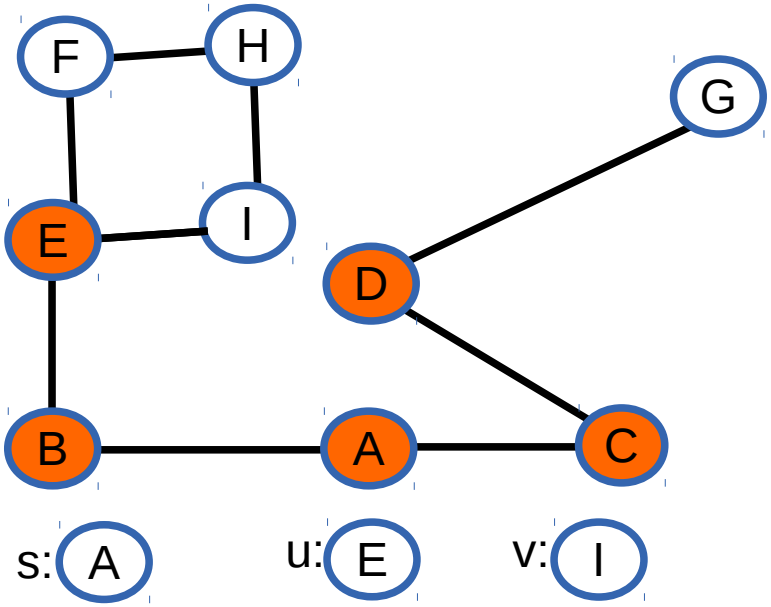
D

0	1	1	2	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```

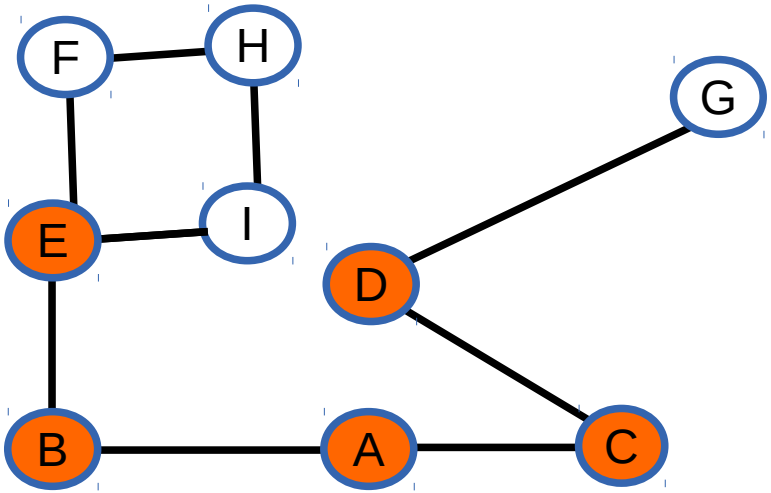


Fila Q

0	1	1	2	2	∞	∞	∞	∞
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: E v: I

Fila Q

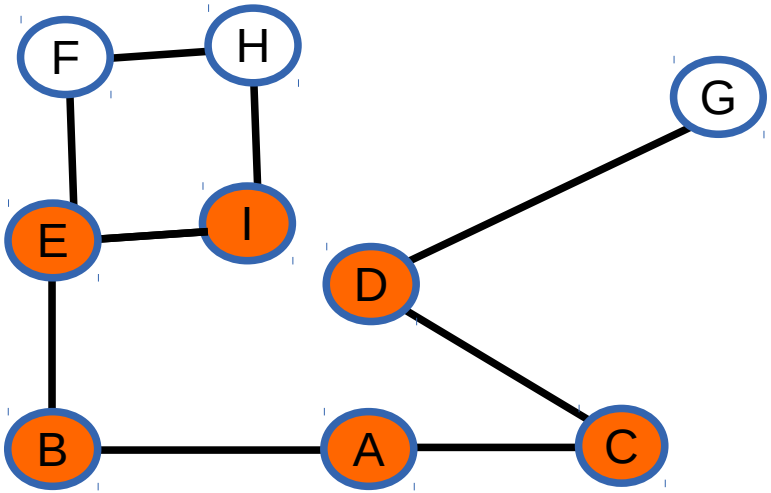
D

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	∞	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: E v: I

Fila Q

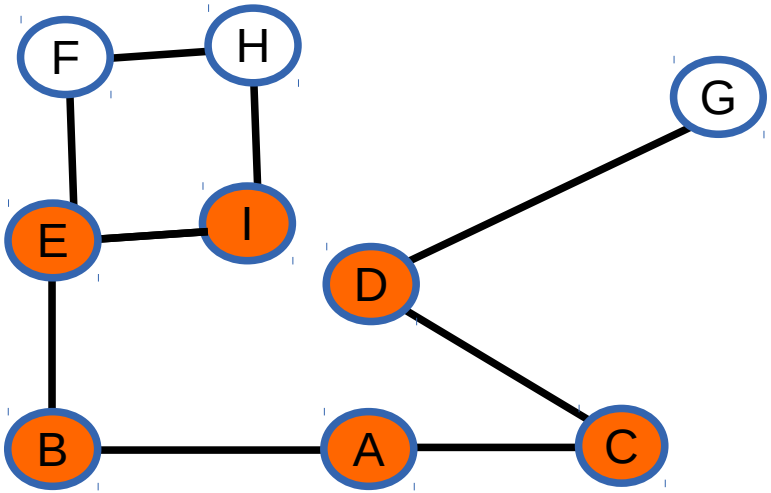
D

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	∞	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: E v: I

Fila Q

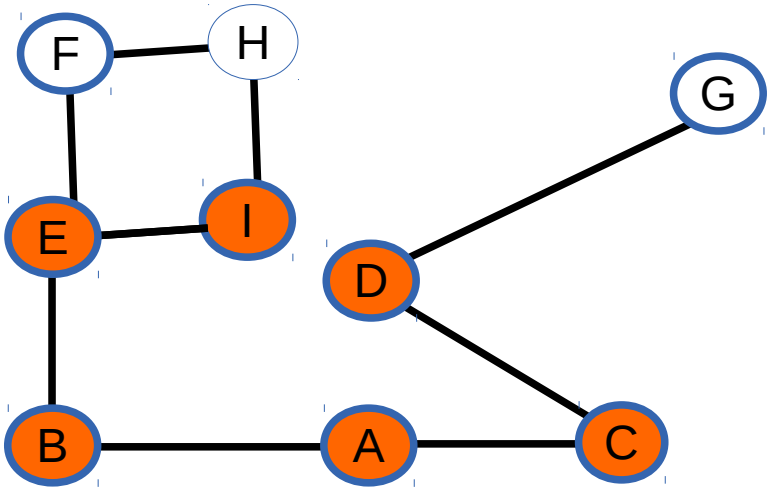

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	∞	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: E v: F

Fila Q

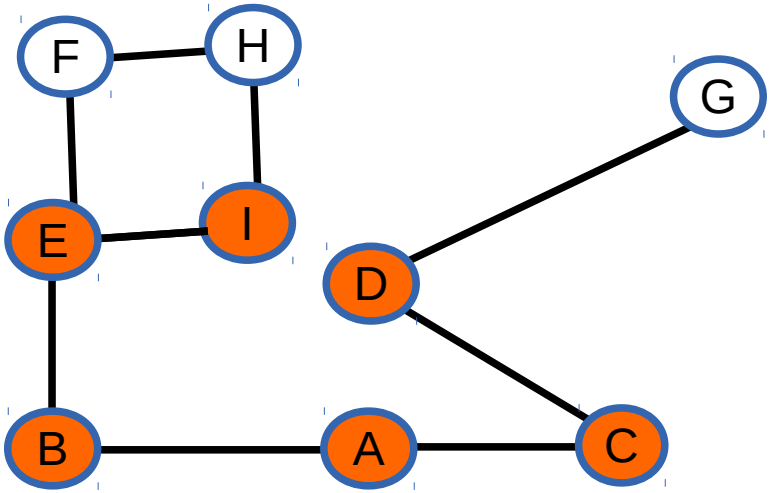
D I

0	1	1	2	2	∞	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: E v: F

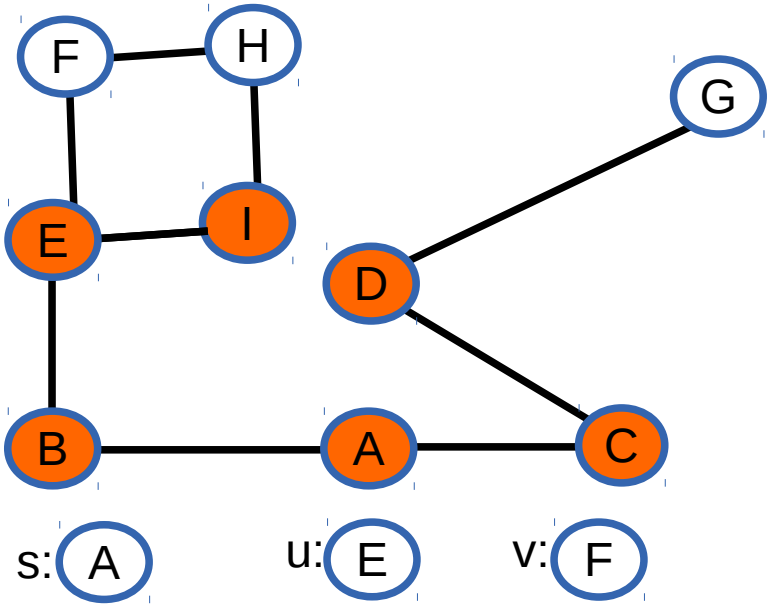
Fila Q

0	1	1	2	2	∞	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

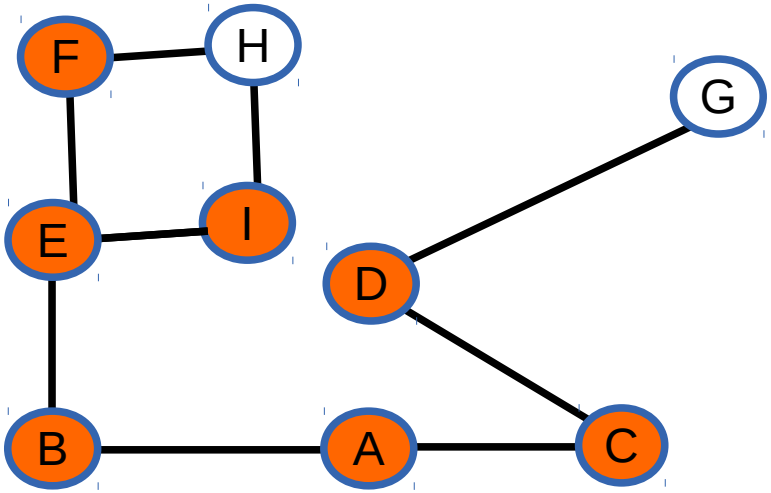


0	1	1	2	2	3	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: E v: F

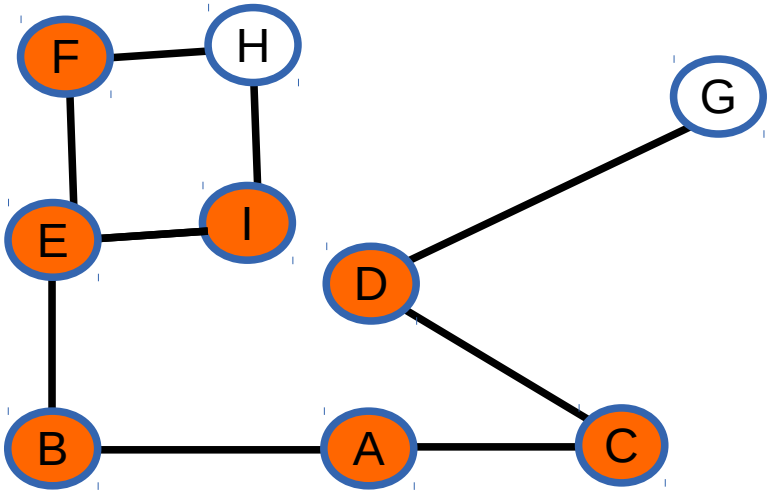
Fila Q

D I

0	1	1	2	2	3	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: E v: F

Fila Q

D I F

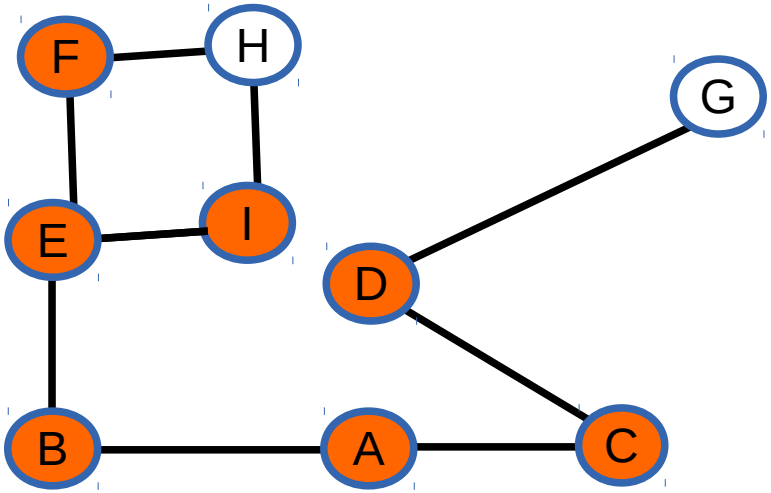
```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```

0	1	1	2	2	3	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
    u ← desenfileira(Q)
    para cada v ∈ adj[u] faça:
        se visit[v] = false então:
            dist[v] = dist[u] + 1
            visit[v] = true
            enfileira(Q,v)
```



s: A u: D v:

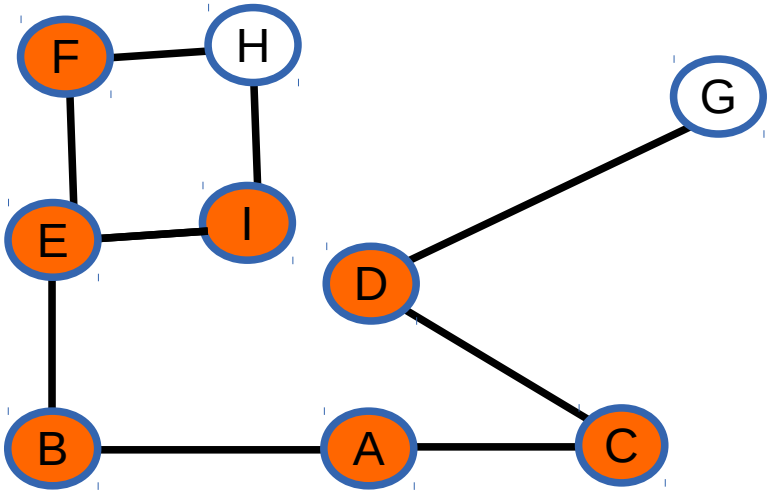
Fila Q

0	1	1	2	2	3	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: D v: G

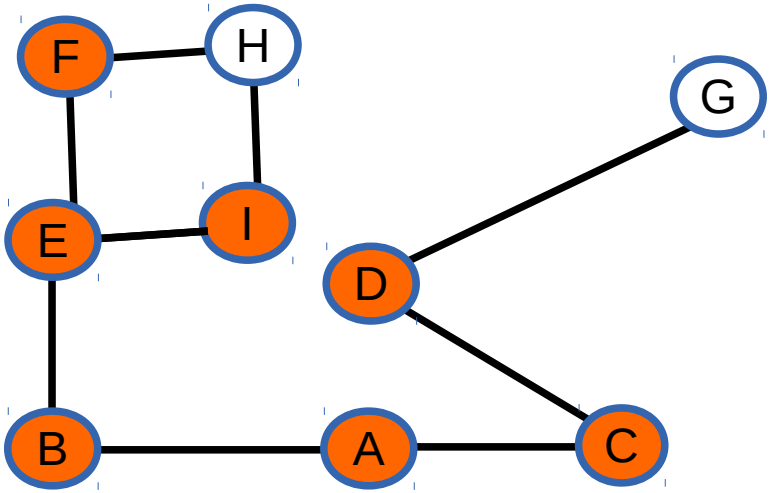
Fila Q

0	1	1	2	2	3	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] = dist[u] + 1
                  visit[v] = true
                  enfileira(Q,v)
```



s: A u: D v: G

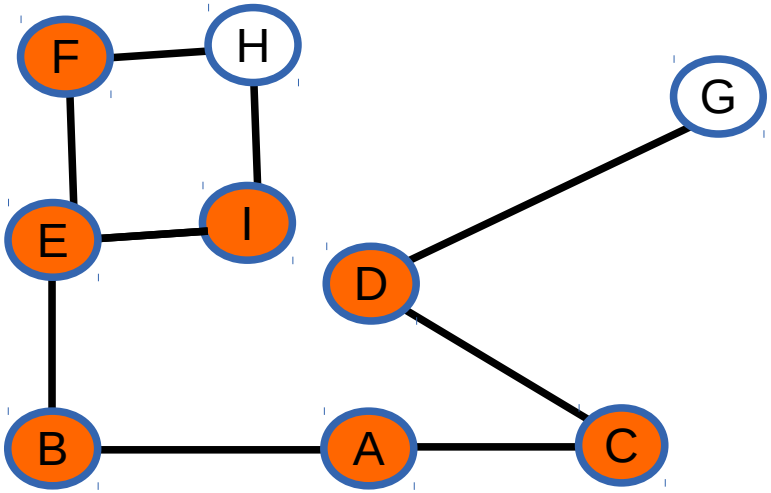
Fila Q

0	1	1	2	2	3	∞	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A u: D v: G

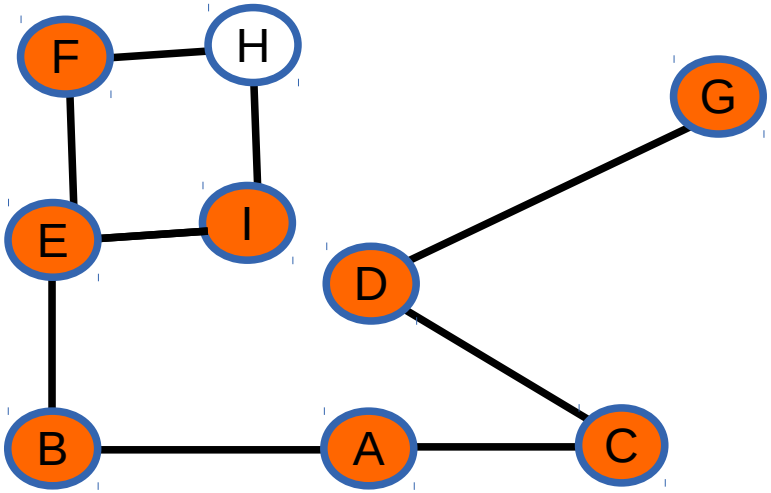
Fila Q

0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: D v: G

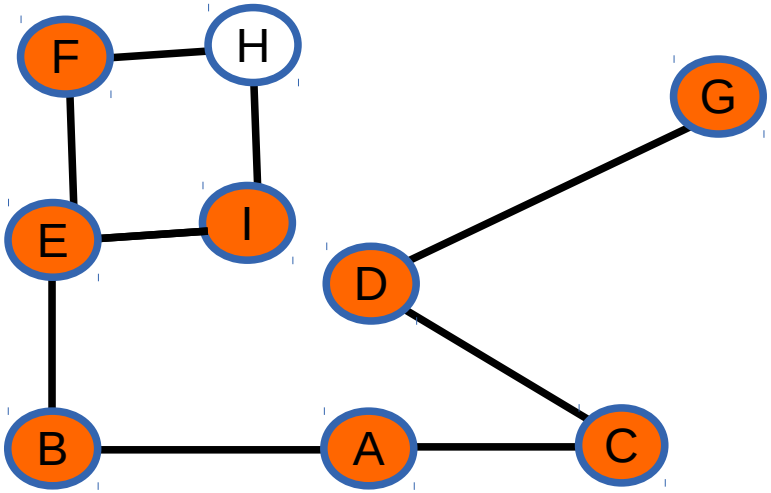
Fila Q

0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

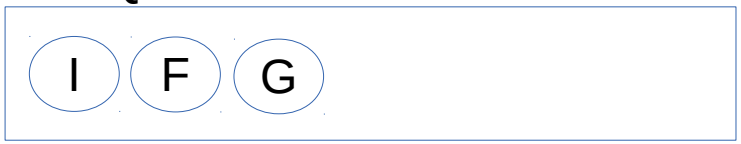
Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] = dist[u] + 1
              visit[v] = true
              enfileira(Q,v)
```



s: A u: D v: G

Fila Q

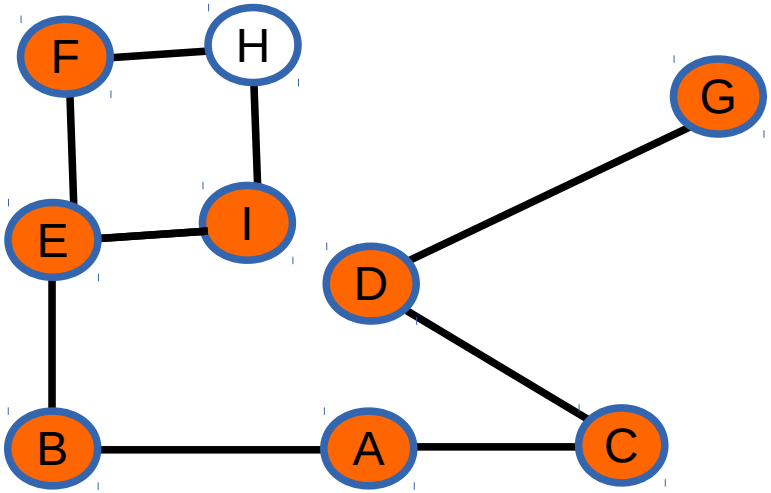


0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

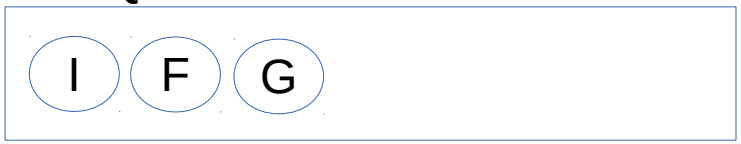
Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] = dist[u] + 1
                visit[v] = true
                enfileira(Q,v)
```



s: A u: D v: C

Fila Q

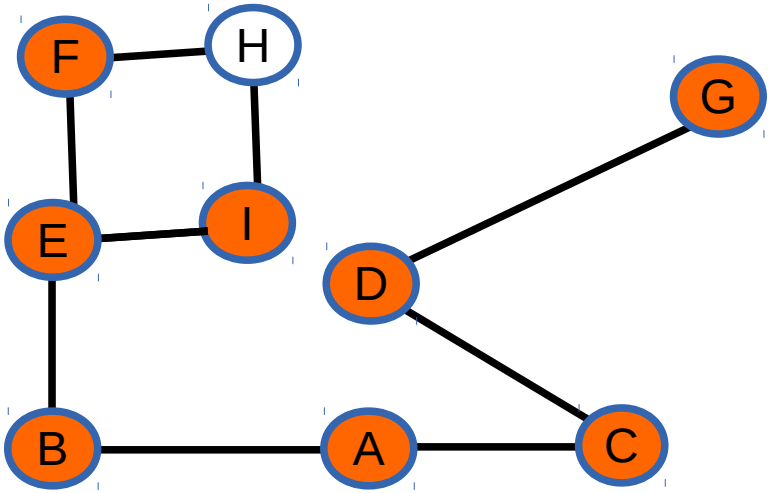


0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠∅ faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] ← dist[u] + 1
                  visit[v] ← true
                  enfileira(Q,v)
```



s: A u: D v: C

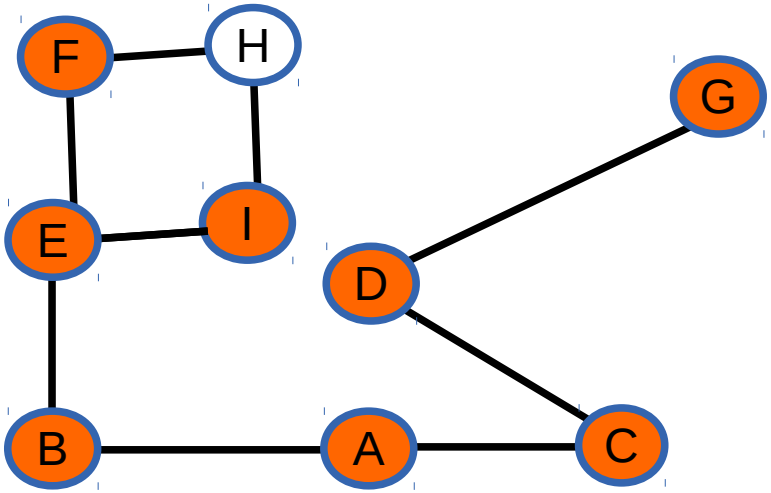
Fila Q

0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
    u ← desenfileira(Q)
    para cada v ∈ adj[u] faça:
        se visit[v] = false então:
            dist[v] ← dist[u] + 1
            visit[v] ← true
            enfileira(Q,v)
```



s: A u: I v:

Fila Q

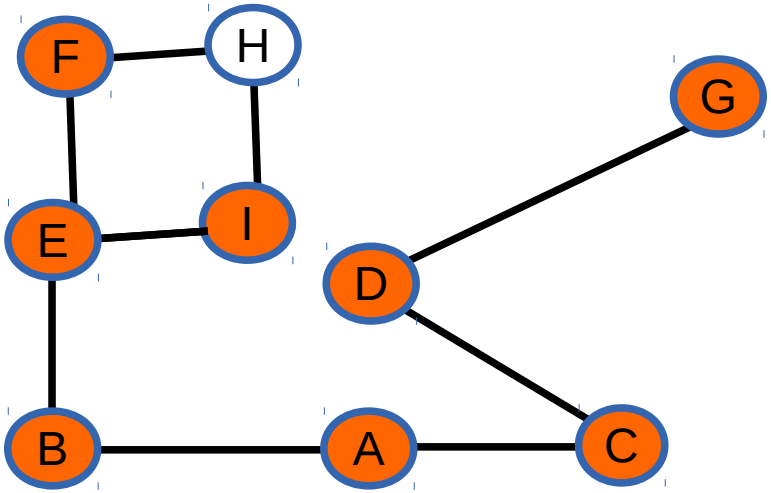
F G

0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
           se visit[v] = false então:
                dist[v] ← dist[u] + 1
                visit[v] ← true
                enfileira(Q,v)
```



s: A u: I v: H

Fila Q

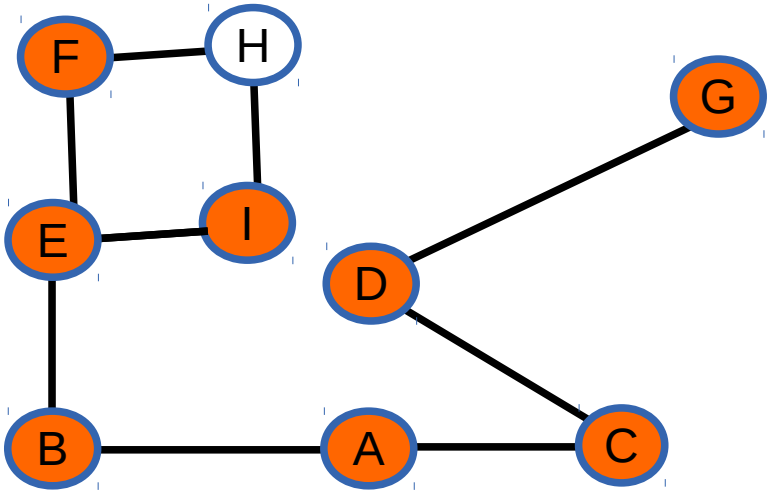
F G

0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
            se visit[v] = false então:
                  dist[v] ← dist[u] + 1
                  visit[v] ← true
                  enfileira(Q,v)
```



s: A u: I v: H

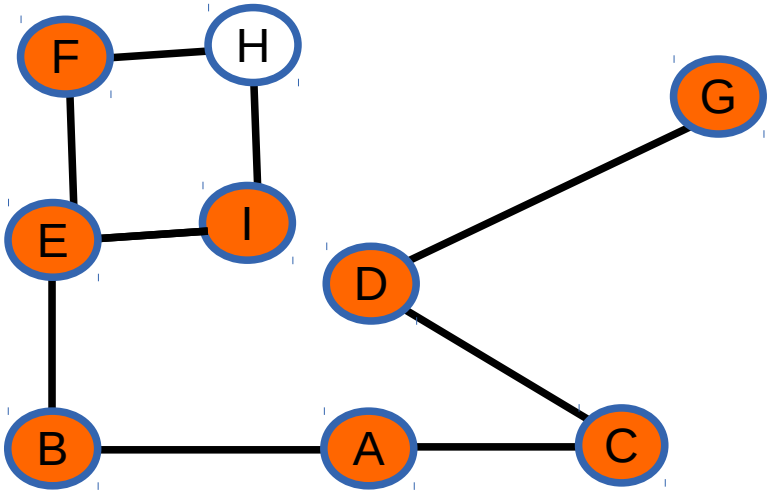
Fila Q

F G

0	1	1	2	2	3	3	∞	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**



s: A u: I v: H

Fila Q

F G

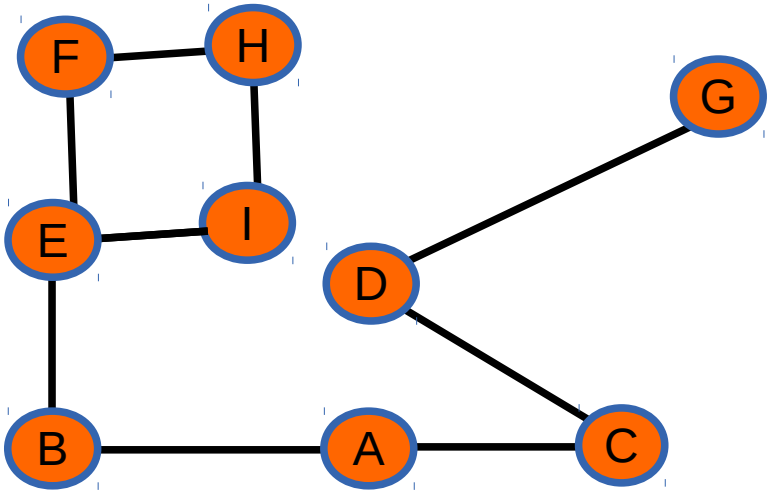
```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] ← dist[u] + 1
              visit[v] ← true
              enfileira(Q,v)
```

0	1	1	2	2	3	3	4	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] ← dist[u] + 1
              visit[v] ← true
              enfileira(Q,v)
```



s: A u: I v: H

Fila Q

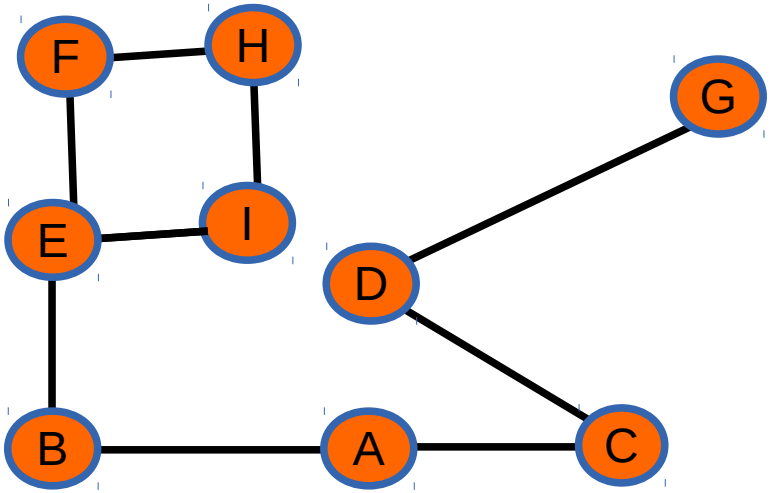
F G

0	1	1	2	2	3	3	4	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] ← dist[u] + 1
              visit[v] ← true
              enfileira(Q,v)
```



s: A u: I v: H

Fila Q

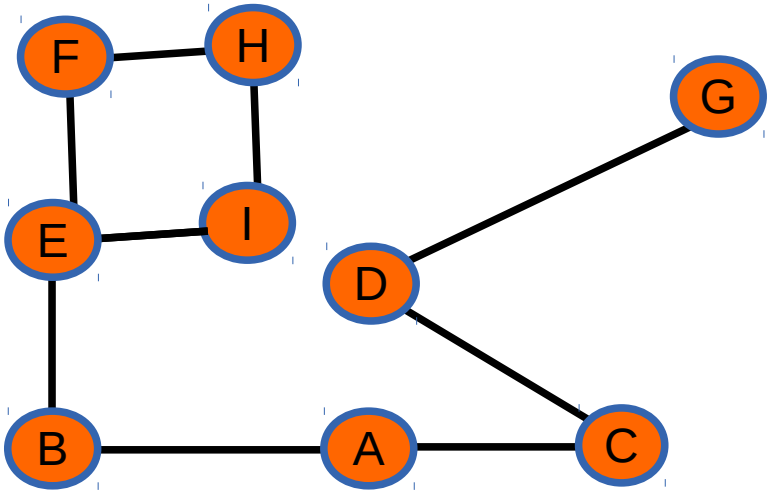
F G H

0	1	1	2	2	3	3	4	3
A	B	C	D	E	F	G	H	I

Busca em Largura – Algoritmo do cálculo da distância do caminho mínimo

Entrada:
→ Grafo **G**
→ vértice que inicial **s**
Considere:
→ conjunto de vértices do grafo G **V[G]**
→ vetor **visit** de vértices visitados:
se **visit**[u] = true, então o vértice u foi visitado
→ adj[u], que lista os vértices adjacentes
ao vértice u
→ Fila Q
Saída:
→ vetor dist[v] com a distância entre o vértice **s** e o vértice **v**

```
para cada u ∈ V[G]  faça:
        visit[u] ← false
        dist[u] ← ∞
visit[s] ← true
Q ←{s}
dist[s] ← 0
enquanto Q ≠Ø faça:
      u ← desenfileira(Q)
      para cada v ∈ adj[u] faça:
          se visit[v] = false então:
              dist[v] ← dist[u] + 1
              visit[v] ← true
              enfileira(Q,v)
```



s: A u: I v: H

Fila Q

F G H

0	1	1	2	2	3	3	4	3
A	B	C	D	E	F	G	H	I

Exercícios

1) Dentre os grafos abaixo, quais podem ser considerados uma árvore?

2) O algoritmo de busca em largura pode ser usado para descobrir a distancia do caminho mínimo apenas quando as arestas não possuem peso. Apresente (graficamente) um exemplo de grafo com arestas ponderadas em que o algoritmo de busca em largura não funcionaria.

3) Para cada problema abaixo: (a) defina se trata de uma classificação ou regressão; (b) defina qual seria a classe alvo; (c) defina dois atributos, exemplos de treino e uma possível árvore de decisão com tais exemplos.

- Predição de quantos gols um time irá marcar
- Você gostaria de saber se um determinado usuário é malicioso ou não
- Predição do valor da nota em uma prova antes da mesma ser aplicada

4) O algoritmo de busca em largura proposto não armazena o caminho mínimo obtido. Altere o algoritmo proposto para que seja armazenado o tal caminhamento. Para isso, crie um vertor **v_anterior[v]**. Que, para cada vértice **v**, armazena o vértice anterior com objetivo de ir do vértice origem **s** ao vértice **v** pelo menor caminho. Perceba que, desta forma, você armazenará a árvore criada pela busca em largura efetuada.