

# SICO7A

# SISTEMAS INTELIGENTES 1

Aula 03 C - Estratégias  
de Buscas sem Informação

Prof. Rafael G. **Mantovani**

# Roteiro



- 1** Introdução
- 2** Busca em Largura (BFS)
- 3** Busca em Profundidade (DFS)
- 4** Exercícios
- 5** Referências

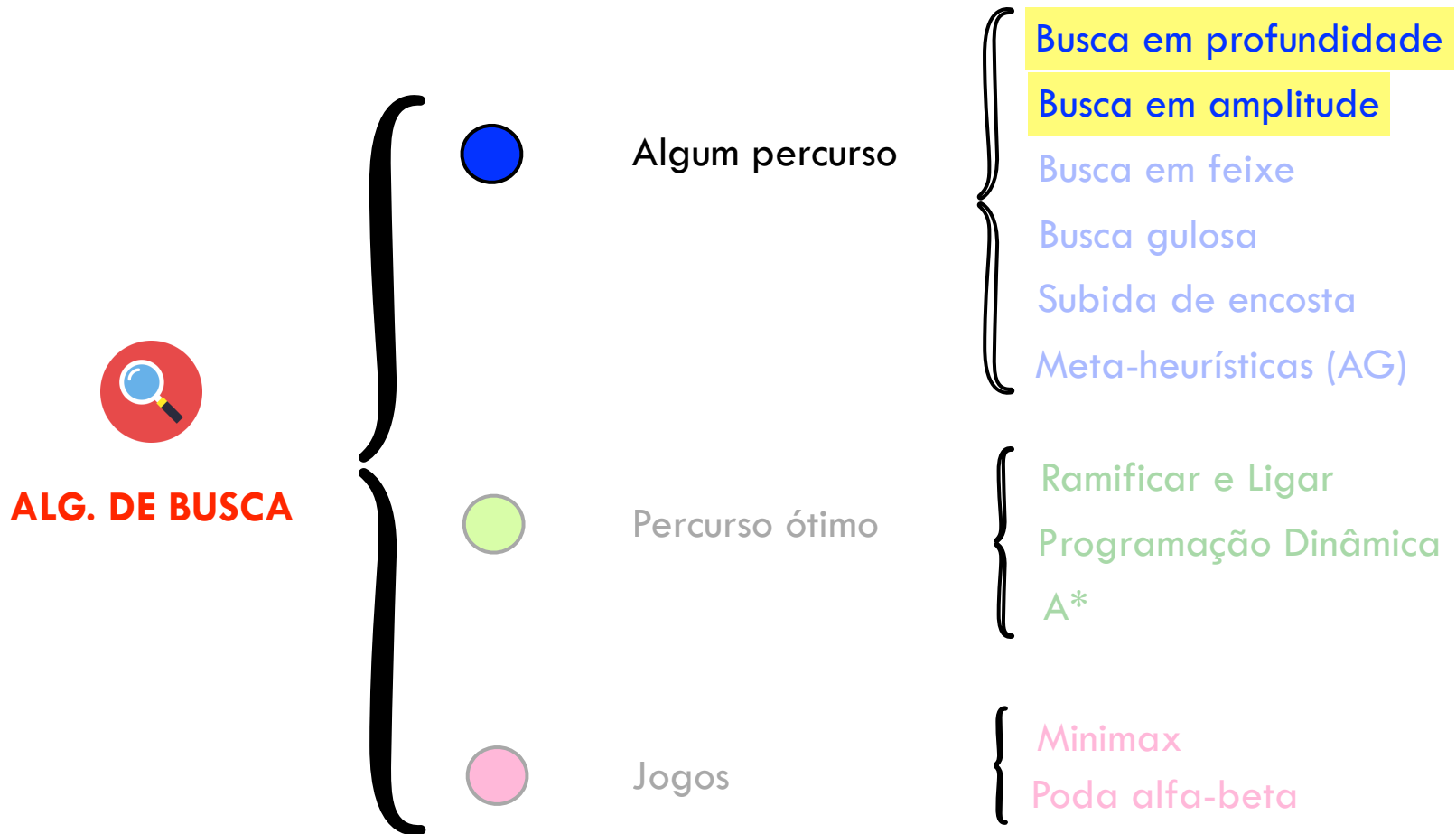
# Roteiro

- 1** Introdução
- 2** Busca em Largura (BFS)
- 3** Busca em Profundidade (DFS)
- 4** Exercícios
- 5** Referências

# Introdução



# Introdução



# Introdução

## Algoritmos de Busca:

- ... Especificam uma direção de busca, explorando o grafo de estados do problema
- ... Deve determinar a ordem em que os estados são examinados durante a busca

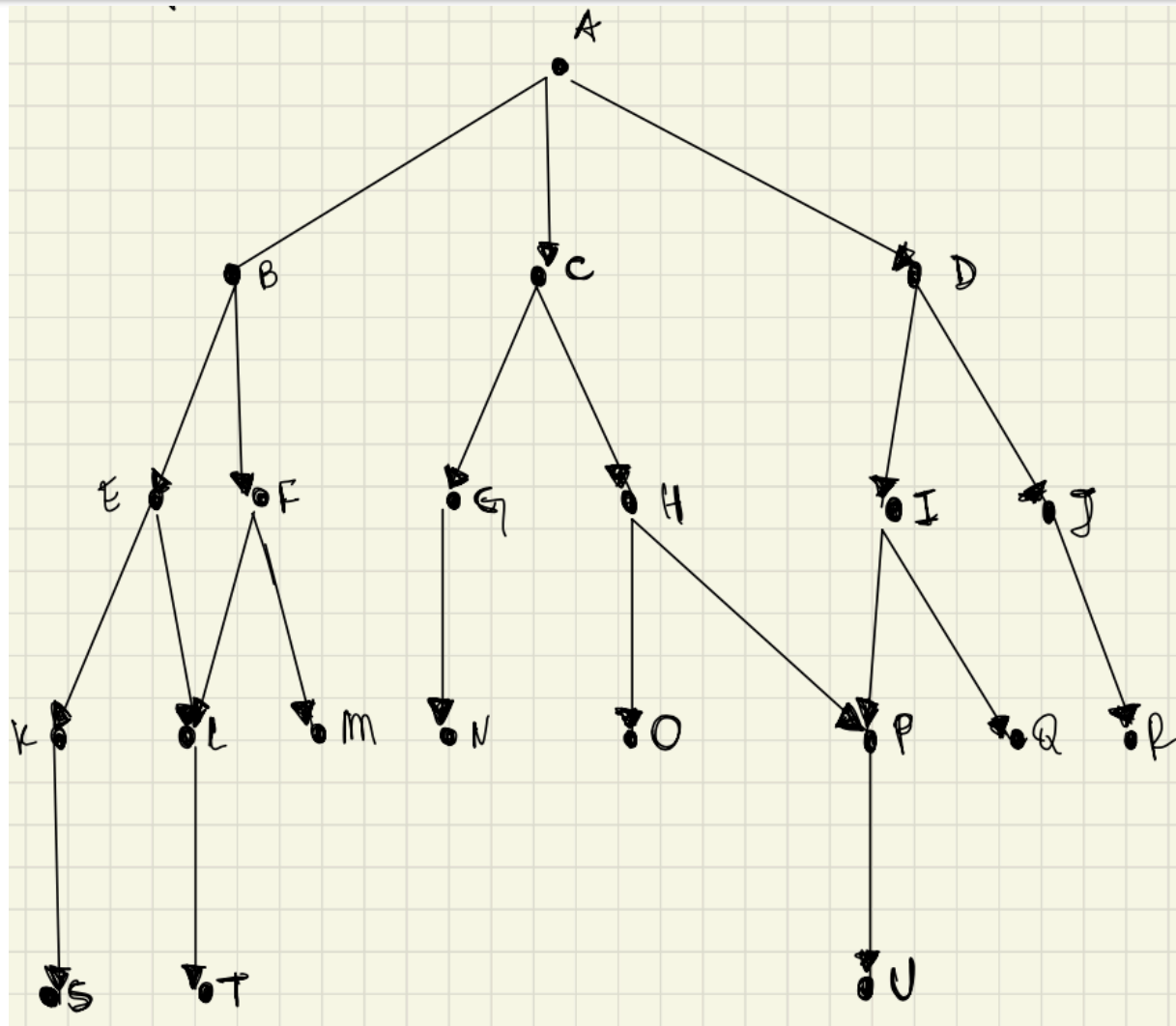
# Introdução

## Algoritmos de Busca:

- ... Especificam uma direção de busca, explorando o grafo de estados do problema
- ... Deve determinar a ordem em que os estados são examinados durante a busca
- ... Possibilidades:

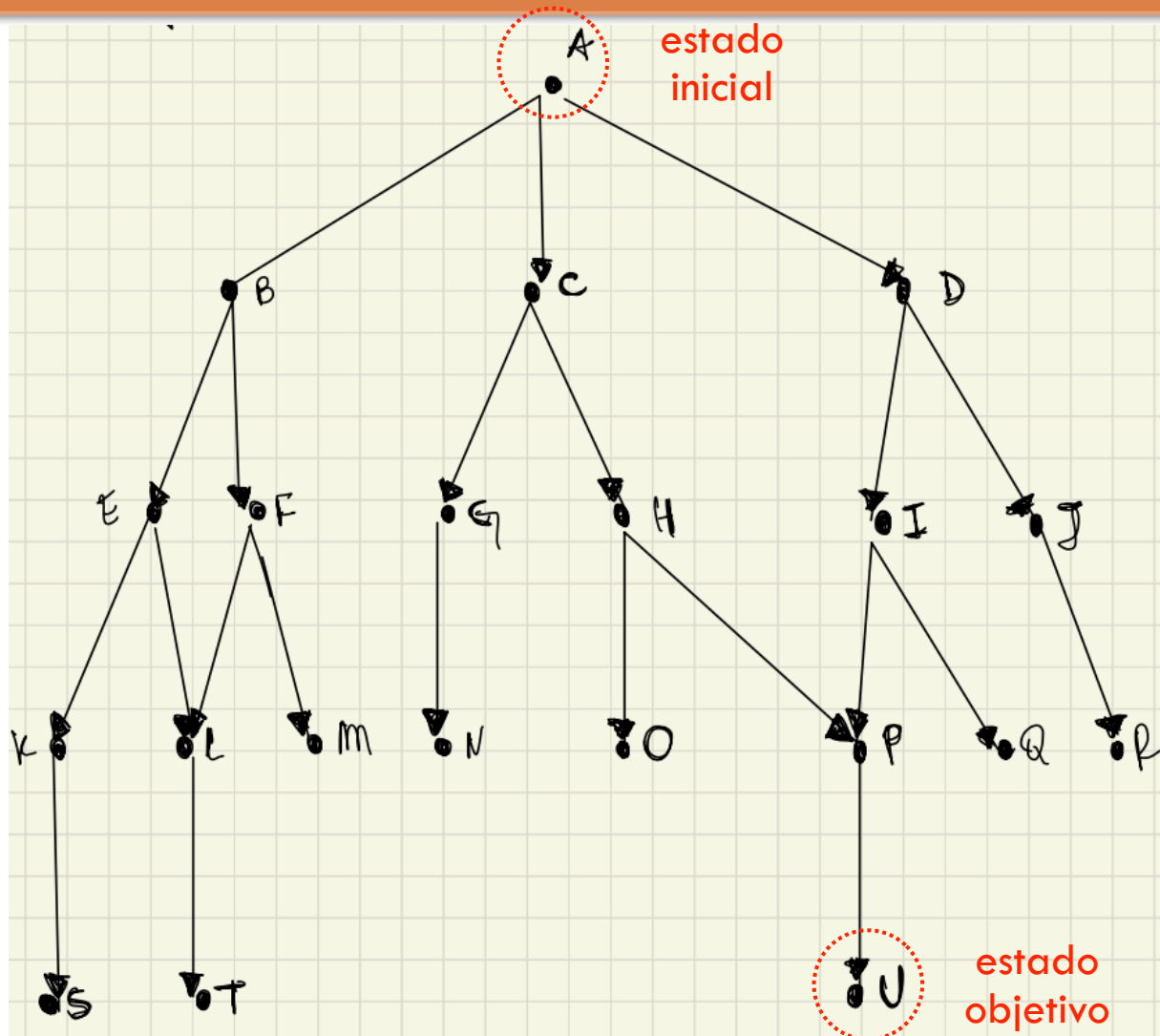
- **Busca em largura** (*Breadth First Search - BFS*) → **Fila**
- **Busca em Profundidade** (*Depth First Search - DFS*) → **Pilha**

# Introdução





# Introdução



# Roteiro



- 1 Introdução
- 2 Busca em Largura (BFS)
- 3 Busca em Profundidade (DFS)
- 4 Exercícios
- 5 Referências

# Busca em Largura (BFS)



\* **Busca em Amplitude/Largura** (*Breadth First Search* - BFS)

# Busca em Largura (BFS)

\* **Busca em Amplitude/Largura** (*Breadth First Search* - BFS)

- Explora o espaço nível por nível
- Apenas quando não houver mais estados a serem explorados em nível, é que a busca aprofunda/desce no grafo
- Usa uma estrutura de dados do tipo **Fila** para acessar os próximos estados gerados

# Busca em Largura (BFS)

\* **Busca em Amplitude/Largura** (*Breadth First Search* - BFS)

- ☉... Explora o espaço nível por nível
- ☉... Apenas quando não houver mais estados a serem explorados em nível, é que a busca aprofunda/desce no grafo
- ☉... Usa uma estrutura de dados do tipo **Fila** para acessar os próximos estados gerados

Acha o caminho mais **curto**!

# Busca em Largura (BFS)

**Sugestão de implementação:** Usar duas listas de estados para registrar o progresso através do espaço de estado

■ **ABERTOS:**

■ **FECHADOS:**

# Busca em Largura (BFS)

**Sugestão de implementação:** Usar duas listas de estados para registrar o progresso através do espaço de estado

- **ABERTOS:** lista todos os estados que foram gerados, mas cujos filhos ainda não foram examinados
- **FECHADOS:** registra os estados que já foram examinados e corresponde à união das listas BSS e LE do algoritmo de busca com *backtracking*

# Pseudocódigo: BFS

---

**BFS (Inicial):**





# Pseudocódigo: BFS

## BFS (Inicial):

1. *// Inicialização*  
**ABERTOS** = [Inicial] *// é uma Fila*
2. **FECHADOS** = [ ]

# Pseudocódigo: BFS

## BFS (Inicial):

```
// Inicialização
1.  ABERTOS = [Inicial] // é uma Fila
2.  FECHADOS = [ ]
3.  Enquanto ABERTOS != [ ] faça:
4.      Remova o estado mais à esquerda de ABERTOS, chame-o de X
5.      Se X for um objetivo, então retornar SUCESSO
```

# Pseudocódigo: BFS

## BFS (Inicial):

```
// Inicialização
1.  ABERTOS = [Inicial] // é uma Fila
2.  FECHADOS = [ ]
3.  Enquanto ABERTOS != [ ] faça:
4.      Remova o estado mais à esquerda de ABERTOS, chame-o de X
5.      Se X for um objetivo, então retornar SUCESSO
6.      Senão
7.          Gere filhos de X
8.          Coloque X em FECHADOS
9.          Descarte filhos de X que já estão em ABERTOS ou FECHADOS
// Evita ciclos ou loops
```

# Pseudocódigo: BFS

## BFS (Inicial):

```
10. | | | Coloque os filhos que restam no final à direita de ABERTOS
11. | | | fim se // enfileirar os estados na Fila
12. | | fim enquanto
13. | Retorna FALHA // não restam mais estados
14. | fim BFS
```

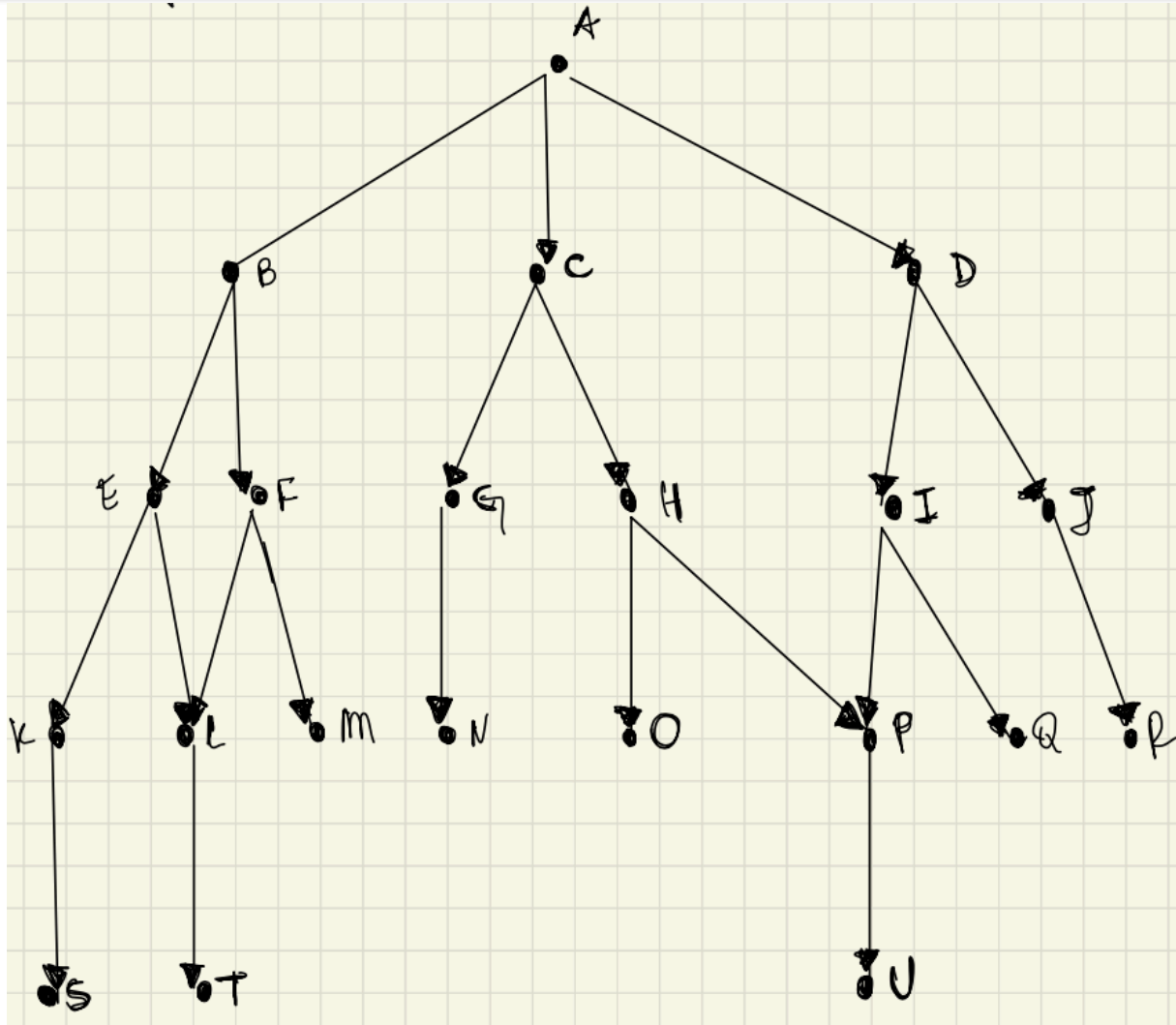
# Pseudocódigo: BFS

## BFS (Inicial):

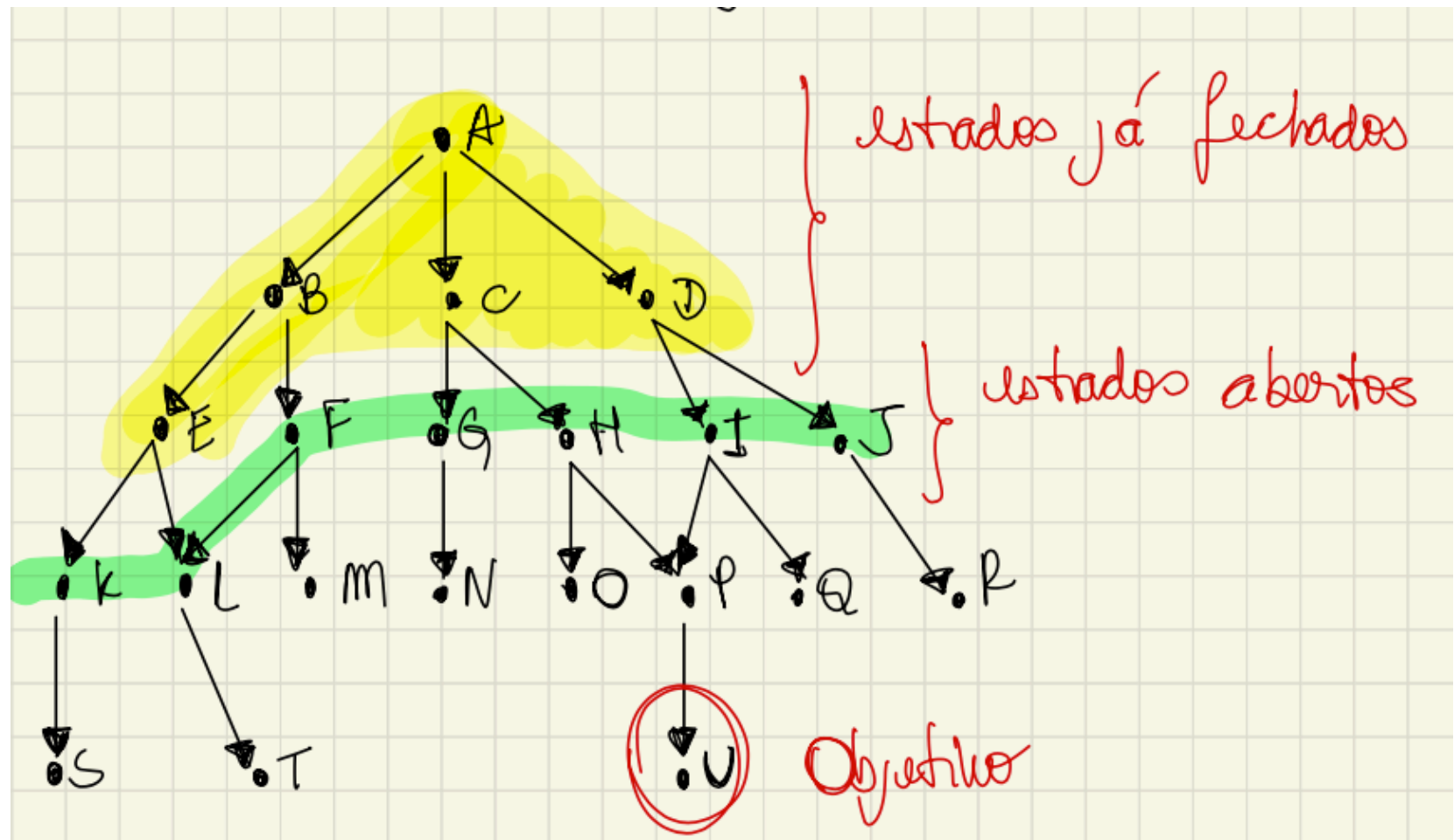
```
10.  Coloque os filhos que restam no final à direita de ABERTOS
11.  fim se                                     // enfileirar os estados na Fila
12.  fim enquanto
13.  Retorna FALHA                             // não restam mais estados
14.  fim BFS
```

- \* A lista **ABERTOS** é mantida como uma **Fila** (*FIFO*)
- \* Estados filhos que já foram descobertos são descascados
- \* Pode/Deve-se guardar outras informações além dos nomes dos estados
  - caminhos?

# Exemplo: BFS



# Exemplo: BFS



# Roteiro



- 1 Introdução
- 2 Busca em Largura (BFS)
- 3 Busca em Profundidade (DFS)
- 4 Exercícios
- 5 Referências



# Busca em Profundidade (DFS)



\* **Busca em Profundidade** (*Depth First Search* - DFS)

# Busca em Profundidade (DFS)

## \* Busca em Profundidade (*Depth First Search* - DFS)

- ... Quando um estado é examinado, todos os seus filhos os descendentes deles são examinados antes de qualquer um dos irmãos
- ... Avança a busca se aprofundando no espaço de estados sempre que possível
- ... Usa uma estrutura de dados do tipo **Pilha** para acessar os próximos estados gerados

# Busca em Profundidade (DFS)

## \* Busca em Profundidade (*Depth First Search* - DFS)

- ☉... Quando um estado é examinado, todos os seus filhos os descendentes deles são examinados antes de qualquer um dos irmãos
- ☉... Avança a busca se aprofundando no espaço de estados sempre que possível
- ☉... Usa uma estrutura de dados do tipo **Pilha** para acessar os próximos estados gerados

Não garante encontrar o caminho mais curto. Mas retorna a primeira solução encontrada a partir do aprofundamento.

# Pseudocódigo: DFS

**DFS (Inicial):**



# Pseudocódigo: DFS

## DFS (Inicial):

1. *// Inicialização*  
**ABERTOS** = [Inicial] *// é uma Pilha*
2. **FECHADOS** = [ ]

# Pseudocódigo: DFS

## DFS (Inicial):

```
// Inicialização
1.  ABERTOS = [Inicial] // é uma Pilha
2.  FECHADOS = [ ]
3.  Enquanto ABERTOS != [ ] faça:
4.      Remova o estado mais à esquerda de ABERTOS, chame-o de X
5.      Se X for um objetivo, então retornar SUCESSO
```

# Pseudocódigo: DFS

## DFS (Inicial):

```
// Inicialização
1. ABERTOS = [Inicial] // é uma Pilha
2. FECHADOS = [ ]
3. Enquanto ABERTOS != [ ] faça:
4.     Remova o estado mais à esquerda de ABERTOS, chame-o de X
5.     Se X for um objetivo, então retornar SUCESSO
6.     Senão
7.         Gere filhos de X
8.         Coloque X em FECHADOS
9.         Descarte filhos de X que já estão em ABERTOS ou FECHADOS
// Evita ciclos ou loops
```

# Pseudocódigo: DFS

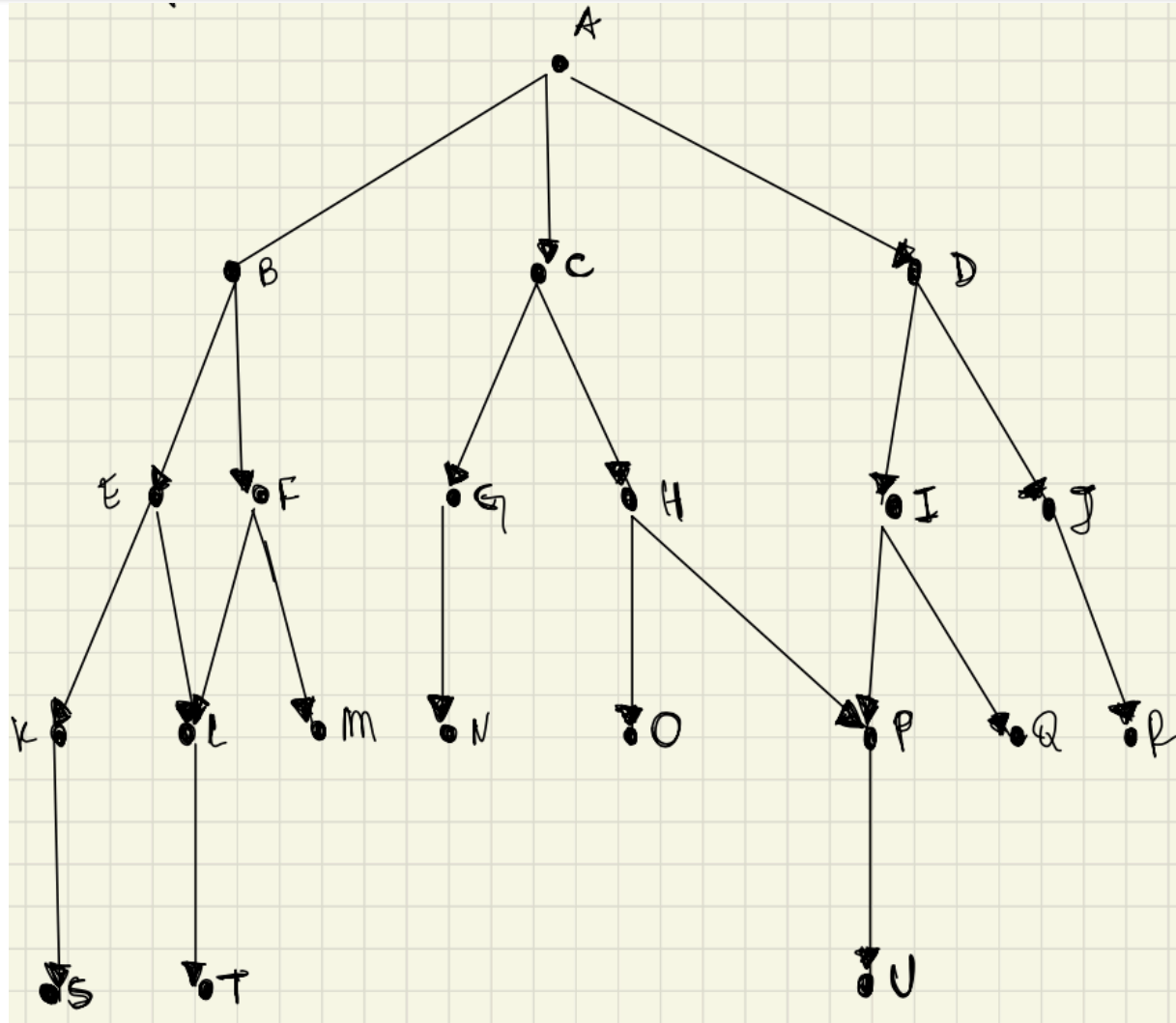
## DFS (Inicial):

```
10. | | | Coloque os filhos que restam no final à esquerda de ABERTOS
11. | | | fim se // empilhar os estados na Fila
12. | | fim enquanto
13. | Retorna FALHA // não restam mais estados
14. | fim DFS
```

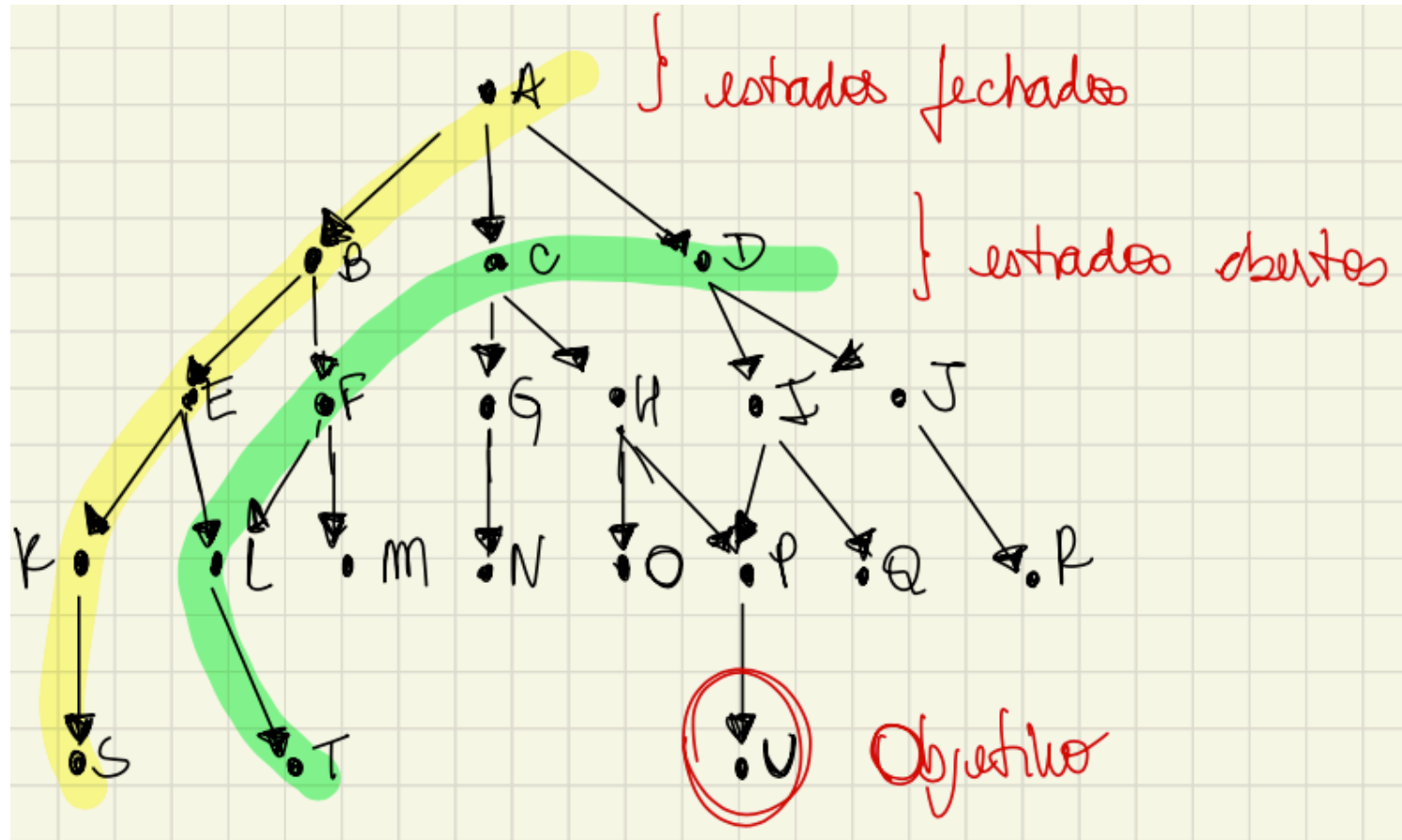
- \* A lista **ABERTOS** é mantida como uma **Pilha** (LIFO)
- \* Estados filhos que já foram descobertos são descascados
- \* Quase igual à BFS, muda apenas alguns comandos



# Exemplo: DFS



# Exemplo: DFS



# BFS x DFS

## BFS (Inicial):

```
10. | | Coloque os filhos que restam no final à direita de ABERTOS
11. | | fim se // enfileirar os estados na Fila
12. | fim enquanto
13. | Retorna FALHA // não restam mais estados
14. | fim BFS
```

## DFS (Inicial):

```
10. | | Coloque os filhos que restam no final à esquerda de ABERTOS
11. | | fim se // empilhar os estados na Pilha
12. | fim enquanto
13. | Retorna FALHA // não restam mais estados
14. | fim DFS
```

# BFS x DFS

## BFS (Inicial):

```
10.     Coloque os filhos que restam no final à direita de ABERTOS
11.     fim se                                     // enfileirar os estados na Fila
12.     fim enquanto
13.     Retorna FALHA                             // não restam mais estados
14.     fim BFS
```

## DFS (Inicial):

```
10.     Coloque os filhos que restam no final à esquerda de ABERTOS
11.     fim se                                     // empilhar os estados na Pilha
12.     fim enquanto
13.     Retorna FALHA                             // não restam mais estados
14.     fim DFS
```

# BFS x DFS

A escolha de qual algoritmo usar depende do problema em específico:

- ☛ **importância** de achar o **caminho mais curto** para o objetivo
- ☛ **o fator de ramificação** do espaço
- ☛ **o tempo** de computação e uso de **memória**
- ☛ **tamanho médio** dos **caminhos** para um nó de objetivos
- ☛ se queremos **todas as soluções** ou apenas a **primeira**

# BFS x DFS



# BFS x DFS

## Busca em Largura (BFS)

- sempre acha o caminho mais curto
- em problemas de solução simples, vai encontrar esta solução
- **desvantagem**: alto fator de ramificação
- impraticável para problemas, como jogo de xadrez

## Busca em Profundidade (DFS)

- chega rapidamente a um espaço de busca profundo
- não desperdiça tempo com níveis superficiais
- **desvantagem**: pode-se perder na profundidade
- mais eficiente para espaços de estados com muitas ramificações

# BFS x DFS

## Busca em Largura (BFS)

- sempre acha o caminho mais curto
- em problemas de solução simples, vai encontrar esta solução
- **desvantagem**: alto fator de ramificação
- impraticável para problemas, como jogo de xadrez

## Busca em Profundidade (DFS)

- chega rapidamente a um espaço de busca profundo
- não desperdiça tempo com níveis superficiais
- **desvantagem**: pode-se perder na profundidade
- mais eficiente para espaços de estados com muitas ramificações



# BFS x DFS

## Busca em Largura (BFS)

- sempre acha o caminho mais curto
- em problemas de solução simples, vai encontrar esta solução
- **desvantagem**: alto fator de ramificação
- impraticável para problemas, como jogo de xadrez

## Busca em Profundidade (DFS)

- chega rapidamente a um espaço de busca profundo
- não desperdiça tempo com níveis superficiais
- **desvantagem**: pode-se perder na profundidade
- mais eficiente para espaços de estados com muitas ramificações

# BFS x DFS

## Busca em Largura (BFS)

- sempre acha o caminho mais curto
- em problemas de solução simples, vai encontrar esta solução

### Busca em Feixe

- impraticável para problemas, como jogo de xadrez

## Busca em Profundidade (DFS)

- chega rapidamente a um espaço de busca profundo
- não desperdiça tempo com níveis superficiais

### Aprofundamento Iterativo

- mais eficiente para espaços de estados com muitas ramificações

# Roteiro



- 1 Introdução
- 2 Busca em Largura (BFS)
- 3 Busca em Profundidade (DFS)
- 4 Exercícios
- 5 Referências

# Exercícios



- 1) Reuna-se com seu grupo e implemente os algoritmos BFS e DFS em Python para o problema do quebra-cabeça de 8 peças.

# Exercícios

1) Reuna-se com seu grupo e implemente os algoritmos BFS e DFS em Python para o problema do quebra-cabeça de 8 peças.

2	8	3
1	6	4
7		5

estado  
inicial

1	2	3
8		4
7	6	5

estado  
objetivo

# Exercícios

1) Reuna-se com seu grupo e implemente os algoritmos BFS e DFS em Python para o problema do quebra-cabeça de 8 peças.

**Obs:** se tudo estiver certo, seu algoritmo de **busca em largura (BFS)** vai necessitar de **46** iterações para solucionar o problema.

**Obs2:** se tudo estiver certo, seu algoritmo de **busca em profundidade (DFS)** (com limitação de 5 níveis de profundidade) vai necessitar de **31** iterações para solucionar o problema.


# Exercícios



2) Teste seus algoritmos para outros estados iniciais do mesmo problema do quebra-cabeça das 8 peças.

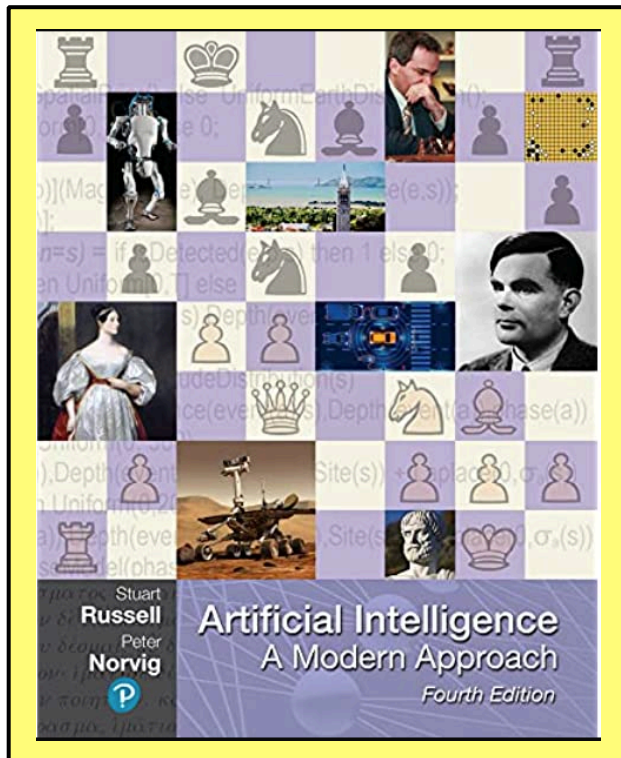
# Roteiro



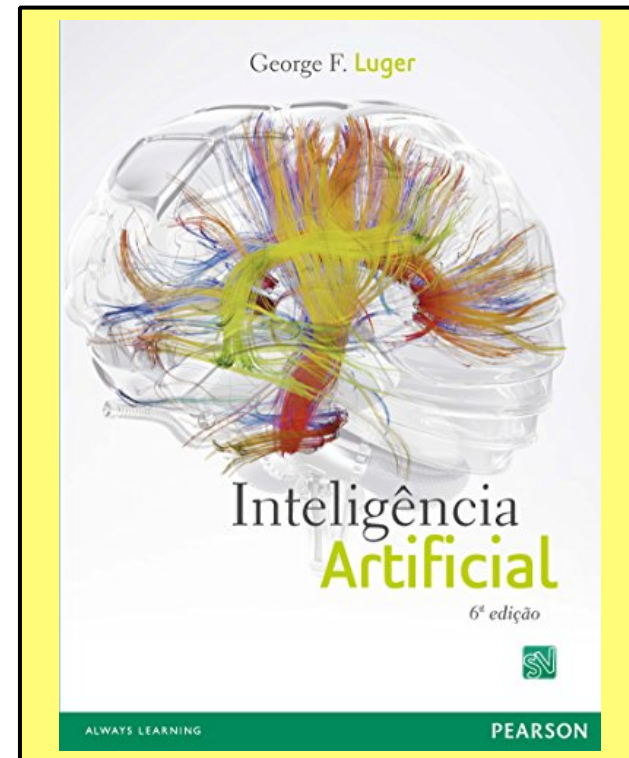
- 1 Introdução
  - 2 Busca em Largura (BFS)
  - 3 Busca em Profundidade (DFS)
  - 4 Exercícios
  - 5 Referências
- 



# Referências sugeridas



[Russel & Norvig, 2021]

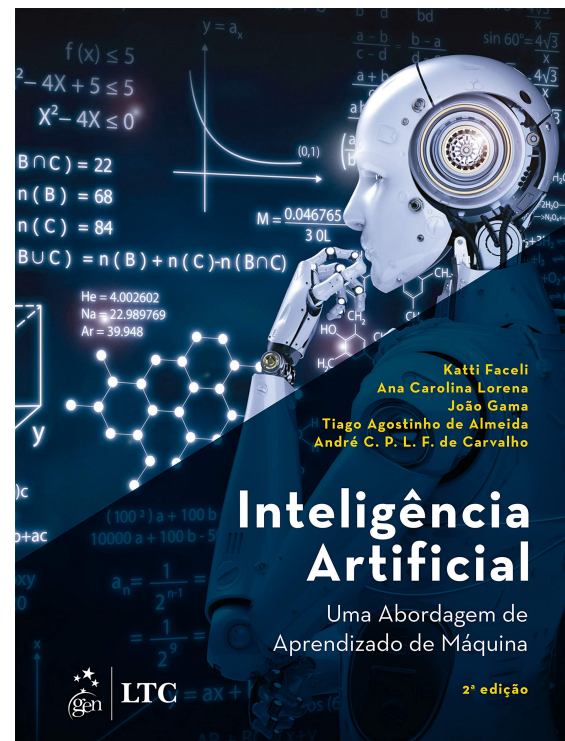


[Luger, 2013]

# Referências sugeridas



[Coppin, 2010]



[Faceli et al, 2021]

# Perguntas?

Prof. Rafael G. **Mantovani**

[rafaelmantovani@utfpr.edu.br](mailto:rafaelmantovani@utfpr.edu.br)