APLICAÇÕES DE LÓGICA COMPUTACIONAL E TEORIA DE CONJUNTOS: UM ESTUDO PRÁTICO COM PYTHON E IMPLICAÇÕES NO DESENVOLVIMENTO DE SOFTWARE

Autor(es) [Felipe Rodrigues de Santana] [Bruno Moreira Pereira]

Instituição Faculdade [Anhanguera] Tecnólogo Análise e Desenvolvimento de Sistema

---

RESUMO

Este relatório científico explora os fundamentos da lógica computacional e da teoria de conjuntos, destacando suas operações e princípios matemáticos essenciais para o desenvolvimento de sistemas. Aborda a evolução histórica da lógica, desde Aristóteles até a lógica booleana e seus conectivos, e discute a aplicação prática dessas teorias através de um código Python que simula operações de conjuntos e resolve um problema de contagem. O estudo contextualiza a relevância desses conceitos no cenário atual da tecnologia, especialmente em áreas como inteligência artificial, segurança de dados e desenvolvimento de software, conforme ilustrado por estudos de caso sobre IA na indústria, proteção de dados com a LGPD e programação orientada a objetos em jogos. A integração desses conhecimentos visa consolidar o raciocínio lógico e analítico, indispensável para profissionais da computação.

Palavras-chave: Lógica Computacional; Teoria de Conjuntos; Python; ABNT; Desenvolvimento de Software; Princípio da Inclusão-Exclusão.

---

1 INTRODUÇÃO

A lógica computacional e a teoria de conjuntos constituem pilares fundamentais para a ciência da computação e o desenvolvimento de algoritmos e sistemas. A capacidade de organizar informações, resolver problemas complexos e garantir a precisão de operações é intrínseca a essas disciplinas. Este relatório tem como objetivo principal investigar a aplicação prática dos conceitos de álgebra de conjuntos e lógica proposicional em um contexto computacional, utilizando um exemplo de código Python e explorando as implicações desses fundamentos em áreas como a Inteligência Artificial e a proteção de dados.

A integração da Inteligência Artificial (IA) na indústria, que impulsiona a Indústria 4.0, depende diretamente da "tomada de decisões baseada em dados", processo intrinsecamente ligado à lógica e à estruturação de informações. Da mesma forma, a proteção de dados pessoais, regulamentada por leis como a LGPD, exige "diretrizes

específicas para o tratamento de dados", onde a categorização e a manipulação de conjuntos de informações são cruciais. No desenvolvimento de software, como o jogo Pong em Java, a "Programação Orientada a Objetos (POO) é um dos paradigmas mais utilizados", e sua "lógica e interação direta entre os elementos do jogo" são construídas sobre princípios lógicos e de conjuntos.

Este documento está estruturado para apresentar, primeiramente, uma revisão dos conceitos de álgebra de conjuntos, suas operações e propriedades. Em seguida, aborda a evolução histórica da lógica e os conectivos lógicos fundamentais. Posteriormente, detalha os princípios matemáticos da combinatória e as ferramentas da lógica proposicional, como a Tabela Verdade. Finalmente, apresenta um estudo de caso prático com código Python para demonstrar a aplicação desses conceitos, seguido de uma discussão sobre suas amplas implicações no cenário tecnológico atual.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Álgebra de Conjuntos

A álgebra de conjuntos é um ramo da matemática que lida com operações e relações entre coleções de objetos, ou conjuntos. As operações básicas incluem união, intersecção e diferença, além da diferença simétrica. Diagramas de Venn são amplamente utilizados para ilustrar visualmente essas operações.

União (∪): A união de dois conjuntos A e B (A ∪ B) é o conjunto que contém todos os elementos que pertencem a A ou a B (ou a ambos). Por exemplo, se A = {10, 11, 12, 13, 14, 15} e B = {13, 14, 15, 16, 17, 18, 19}, então A ∪ B = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19}. Elementos comuns são contados apenas uma vez.

Intersecção (∩): A intersecção de A e B (A ∩ B) é o conjunto de elementos que pertencem simultaneamente a A e a B. No exemplo anterior, A ∩ B = {13, 14, 15}.

Diferença (–): A diferença A – B é o conjunto de todos os elementos que estão em A, mas não em B. Para A = {1, 2, 3, 4, 5} e B = {4, 5, 6, 7}, a diferença A – B = {1, 2, 3}.

Diferença Simétrica (Δ): A diferença simétrica de A e B (A Δ B) inclui elementos que pertencem a A, mas não a B, ou a B, mas não a A. Pode ser expressa como (A – B) ∪ (B – A) ou (A ∪ B) – (A ∩ B).

Princípio da Inclusão-Exclusão: Este princípio é fundamental para a contagem de elementos em uma união de conjuntos finitos. A fórmula é |A ∪ B| = |A| + |B| – |A ∩ B|, onde a subtração de |A ∩ B| evita a contagem duplicada de elementos comuns.

2.2 Evolução da Lógica e seus Conectivos

A história da lógica pode ser dividida em três grandes períodos: Aristotélico, Booleano e Atual.

Período Aristotélico: Inicia-se aproximadamente em 390 a.C. com Aristóteles, que desenvolveu a teoria do silogismo, um tipo de inferência dedutiva. A lógica clássica, regida

pelos princípios da identidade, não contradição e terceiro excluído, é característica desse período.

Período Booleano: Abrange de 1840 a 1910, marcado pelo desenvolvimento da Lógica Formal ou Simbólica, onde símbolos computáveis substituem palavras e proposições. George Boole inventou a Álgebra Booleana, que utiliza apenas os valores 0 (falso) e 1 (verdadeiro) para lidar com proposições. Georg Cantor idealizou a Teoria de Conjuntos e Gottlob Frege criou a Lógica Matemática, reformulando a lógica tradicional com linguagem matemática e introduzindo o quantificador e variáveis.

Período Atual: A partir de 1910, caracteriza-se pelo desenvolvimento de sistemas formais polivalentes, as lógicas não clássicas, que lidam com imprecisões e contradições, indo além dos valores verdadeiro e falso. Exemplos incluem a lógica fuzzy, importante para a Inteligência Artificial.

Conectivos Lógicos: São símbolos que conectam proposições e formam novas expressões lógicas. Os principais conectivos são:

- **Negação (¬ ou ~):** Inverte o valor-verdade de uma proposição.
- **Conjunção (∧ ou AND):** Verdadeira apenas se todas as proposições forem verdadeiras.
- **Disjunção (∨ ou OR):** Falsa apenas se todas as proposições forem falsas (disjunção inclusiva). A disjunção exclusiva (⊕ ou XOR) é verdadeira se, e somente se, apenas uma das proposições for verdadeira.
- **Condicional (→ ou IF...THEN):** A implicação "Se P, então Q" (P → Q) é falsa somente quando o antecedente (P) é verdadeiro e o consequente (Q) é falso.
- **Bicondicional (↔ ou IF AND ONLY IF):** "P se, e somente se, Q" (P ↔ Q) é verdadeira quando as duas proposições têm o mesmo valor-verdade.

2.3 Princípios Matemáticos e Combinatória

A matemática discreta, também conhecida como combinatória, é crucial para a computação, pois lida com objetos e estruturas finitas, sendo aplicada na contagem, no estudo de relações entre conjuntos e na análise de algoritmos.

Listas: Uma lista é uma sequência ordenada de objetos. O número de elementos é o comprimento da lista. Uma lista de `n` elementos é uma `n-upla`. O **Princípio da Multiplicação** é usado para contar o número de listas possíveis.

Fatorial (n!): Representa o número de maneiras distintas de dispor `n` objetos em uma lista sem repetições. É calculado como `n * (n-1) * ... * 1`.

Diagramas de Árvore (Árvores de Decisão): Estruturas hierárquicas que representam possíveis resultados de uma série de escolhas, úteis para ilustrar o número de possibilidades em eventos combinatórios.

Agrupamentos:

- **Arranjos:** Sequências ordenadas de `p` elementos distintos escolhidos de um conjunto de `n` elementos. A ordem importa.
- **Permutações:** Caso especial de arranjo onde `p = n`, ou seja, todos os elementos são arranjados. Também um tipo de lista.
- **Combinações:** Subconjuntos não ordenados de `p` elementos distintos escolhidos de um conjunto de `n` elementos. A ordem não importa.

2.4 Lógica Proposicional e Tabela-Verdade

Proposição: Uma sentença declarativa que pode ser classificada como verdadeira (V) ou falsa (F), mas não ambas simultaneamente. Podem ser simples (uma única afirmação) ou compostas (duas ou mais proposições simples ligadas por conectivos lógicos).

Fórmulas Bem-Formuladas (fbf): Sequências válidas de proposições, conectivos e parênteses que seguem regras de sintaxe. A ordem de precedência dos conectivos é crucial para a correta valoração de uma fbf.

Tabela-Verdade: Método exaustivo para gerar todas as possíveis valorações de uma fórmula, testando todas as combinações de valores lógicos de entrada para as proposições. O número de linhas em uma tabela-verdade é `2^n`, onde `n` é o número de proposições.

Classificações de Fórmulas:

- **Tautologia:** Uma fórmula cujo resultado é sempre verdadeiro, independentemente dos valores de entrada das proposições.
- **Contradição:** Uma fórmula cujo resultado é sempre falso.
- **Contingência:** Uma fórmula que não é nem tautologia nem contradição, ou seja, pode ser verdadeira ou falsa dependendo dos valores de entrada.

Equivalência Lógica: Duas fórmulas são logicamente equivalentes se apresentam o mesmo resultado lógico para todas as combinações possíveis de entradas. O bicondicional (↔) é um atalho para a equivalência lógica. As **Leis de De Morgan** são exemplos importantes de equivalências lógicas, relacionando a negação de disjunções e conjunções.

3 ESTUDO DE CASO: ÁLGEBRA DE CONJUNTOS COM PYTHON NO GOOGLE COLAB

A aplicação dos conceitos de álgebra de conjuntos pode ser facilmente demonstrada e explorada utilizando a linguagem de programação Python e seu tipo de dados `set`. O ambiente Google Colab é ideal para executar e interagir com esse tipo de código. O exemplo a seguir demonstra as operações fundamentais e a resolução de uma situação-problema.

3.1 Código de Exemplo em Python

```
# Código de Exemplo para Álgebra de Conjuntos no Google Colab

print("--- Demonstração de Operações Básicas de Conjuntos ---")
```

```
# 1. Definição dos conjuntos de exemplo
# Usaremos conjuntos semelhantes aos apresentados no Exemplo 1 da seção.
A = {10, 11, 12, 13, 14, 15}
B = {13, 14, 15, 16, 17, 18, 19}

print(f"Conjunto A: {A}")
print(f"Conjunto B: {B}")
print(f"Cardinalidade de A (|A|): {len(A)}")
print(f"Cardinalidade de B (|B|): {len(B)}")
print("-" * 40)

# 2. União de Conjuntos (A ∪ B)
# O conjunto união de A e B (A ∪ B) é definido como o conjunto de todos os elementos que
# pertencem a A *ou* a B (ou a ambos).
# Em Python, a união pode ser feita com o método .union() ou com o operador |
uniao_AB = A.union(B) # ou A | B
print(f"União de A e B (A ∪ B): {uniao_AB}")
print(f"Cardinalidade da União (|A ∪ B|): {len(uniao_AB)}")
print("-" * 40)

# 3. Intersecção de Conjuntos (A ∩ B)
# O conjunto intersecção de A e B (A ∩ B) é composto por todos os elementos que
# pertencem *simultaneamente* a A *e* a B.
# Em Python, a intersecção pode ser feita com o método .intersection() ou com o operador
&
interseccao_AB = A.intersection(B) # ou A & B
print(f"Intersecção de A e B (A ∩ B): {interseccao_AB}")
print(f"Cardinalidade da Intersecção (|A ∩ B|): {len(interseccao_AB)}")
print("-" * 40)

# 4. Diferença de Conjuntos (A – B e B – A)
# A diferença A – B é o conjunto de todos os elementos que pertencem a A, mas *não* a B.
# Em Python, a diferença pode ser feita com o método .difference() ou com o operador -
diferenca_AB = A.difference(B) # ou A - B
diferenca_BA = B.difference(A) # ou B - A
print(f"Diferença A – B: {diferenca_AB}")
print(f"Diferença B – A: {diferenca_BA}")
print("-" * 40)

# 5. Diferença Simétrica de Conjuntos (A Δ B)
# A diferença simétrica de A e B (A Δ B) é o conjunto de todos os elementos que pertencem
a A
# mas não a B, *ou* que pertencem a B mas não a A.
# Pode ser representada como A Δ B = (A – B) ∪ (B – A) ou A Δ B = (A ∪ B) – (A ∩ B).
# Em Python, a diferença simétrica pode ser feita com o método .symmetric_difference() ou
com o operador ^
diferenca_simetrica_AB_direto = A.symmetric_difference(B) # ou A ^ B
print(f"Diferença Simétrica A Δ B: {diferenca_simetrica_AB_direto}")
```

```
print("-" * 40)

print("\n--- Aplicação do Princípio da Inclusão-Exclusão ---")
# O Princípio da Inclusão-Exclusão é utilizado para contar o número de elementos em uma união de
# conjuntos finitos. A fórmula é |A ∪ B| = |A| + |B| – |A ∩ B|.

cardinalidade_A = len(A)
cardinalidade_B = len(B)
cardinalidade_interseccao = len(A.intersection(B))

cardinalidade_uniao_pelo_principio = cardinalidade_A + cardinalidade_B - cardinalidade_interseccao
print(f"|A| = {cardinalidade_A}")
print(f"|B| = {cardinalidade_B}")
print(f"|A ∩ B| = {cardinalidade_interseccao}")
print(f"|A ∪ B| calculado pelo Princípio da Inclusão-Exclusão: {cardinalidade_uniao_pelo_principio}")
print(f"|A ∪ B| direto (verificação com len(A.union(B))): {len(A.union(B))}")
print("-" * 40)

print("\n--- Resolução da Situação-Problema (Unidade 2, Seção 2) ---")
# Problema: Uma equipe de desenvolvimento de software precisa determinar quantos comandos
# de um aplicativo, de um total de 60, realizam buscas no Banco de Dados B.
# Sabe-se que 20 comandos direcionam a busca para o Banco de Dados A e 12 comandos
# direcionam a busca para ambos os Bancos de Dados A e B.

# Usaremos a fórmula do Princípio da Inclusão-Exclusão: |A ∪ B| = |A| + |B| – |A ∩ B|.

total_comandos_uniao = 60  # Equivalente a |A ∪ B|
comandos_db_A = 20         # Equivalente a |A|
comandos_db_ambos = 12     # Equivalente a |A ∩ B|

# Queremos encontrar o número de comandos que realizam busca no Banco de Dados B, ou seja, |B|.
# Rearranjando a fórmula: |B| = |A ∪ B| - |A| + |A ∩ B|
comandos_db_B = total_comandos_uniao - comandos_db_A + comandos_db_ambos

print(f"Total de comandos (União de A e B): {total_comandos_uniao}")
print(f"Comandos que direcionam para Banco de Dados A: {comandos_db_A}")
print(f"Comandos que direcionam para ambos os Bancos de Dados (A e B): {comandos_db_ambos}")
print(f"Número de comandos que realizam buscas no Banco de Dados B: {comandos_db_B}")
print(f"Isso significa que há {comandos_db_B} comandos que realizam a busca no Banco de Dados B,")
print(f"incluindo os {comandos_db_ambos} que também buscam em A.")
```

```
print("\nCom base na análise do diagrama de Venn mencionada, podemos detalhar:")
print(f"Comandos que buscam *apenas* em A: {comandos_db_A - comandos_db_ambos}")
print(f"Comandos que buscam *apenas* em B: {comandos_db_B - comandos_db_ambos}")
print(f"Verificação do total: (apenas A) + (A e B) + (apenas B) = {(comandos_db_A - comandos_db_ambos)} + {comandos_db_ambos} + {(comandos_db_B - comandos_db_ambos)} = {total_comandos_uniao}")
```

3.2 Explicação do Código

O código Python:

- **Define Conjuntos:** Utiliza a sintaxe `{}` para criar conjuntos `A` e `B`, similares aos exemplos teóricos.
- **Demonstra União (`.union()` ou `|`):** Mostra como combinar elementos de `A` e `B`, contabilizando elementos únicos.
- **Demonstra Intersecção (`.intersection()` ou `&`):** Apresenta os elementos comuns a `A` e `B`.
- **Demonstra Diferença (`.difference()` ou `-`):** Calcula os elementos presentes em um conjunto, mas não no outro.
- **Demonstra Diferença Simétrica (`.symmetric_difference()` ou `^`):** Identifica os elementos que pertencem a um conjunto ou ao outro, mas não a ambos.
- **Aplica o Princípio da Inclusão-Exclusão:** Calcula a cardinalidade da união de forma teórica e compara com a cardinalidade direta, validando a fórmula `|A ∪ B| = |A| + |B| − |A ∩ B|`.
- **Resolve a Situação-Problema:** Aplica o princípio da inclusão-exclusão para determinar o número de comandos que buscam no Banco de Dados B, dadas as informações sobre o total de comandos, comandos em A e comandos em ambos. O problema é: "uma equipe de desenvolvimento de software precisa determinar quantos comandos de um aplicativo, de um total de 60, realizam buscas no Banco de Dados B. Sabe-se que 20 comandos direcionam a busca para o Banco de Dados A e 12 comandos direcionam a busca para ambos os Bancos de Dados A e B". A solução chega a 52 comandos para o Banco de Dados B.

4 DISCUSSÃO E IMPLICAÇÕES

A lógica computacional e a teoria de conjuntos são mais do que meros conceitos acadêmicos; são a espinha dorsal de inúmeras aplicações tecnológicas. O código de exemplo demonstra a versatilidade de `sets` em Python para modelar problemas do mundo real, como a análise de comandos em bancos de dados. A capacidade de realizar operações de união, intersecção e diferença permite que desenvolvedores de software categorizem e manipulem dados de forma eficiente, otimizando funcionalidades de busca e filtragem.

A resolução da "Situação-Problema" através do Princípio da Inclusão-Exclusão exemplifica a aplicação do raciocínio lógico-matemático para evitar a contagem duplicada de elementos, uma habilidade crítica em contextos de análise de dados e sistemas de informação. Este

tipo de lógica é fundamental para a "tomada de decisões baseada em dados" em setores industriais que utilizam Inteligência Artificial, conforme mencionado no estudo sobre IA na indústria. Sistemas de IA, que implementam "aprendizado de máquina, análise preditiva e robótica avançada", dependem de algoritmos que processam e categorizam grandes volumes de dados, frequentemente utilizando operações de conjuntos e lógica proposicional para extrair insights e automatizar tarefas complexas.

Além disso, a compreensão dos conectivos lógicos (AND, OR, NOT) e sua relação com as operações de conjuntos é diretamente aplicável no desenvolvimento de software. Em programação, "os conectivos lógicos OU e E são simbolizados por $\vee$ e $\wedge$", formando a base para a construção de "estruturas de decisões" e algoritmos que governam o comportamento de aplicativos e sistemas. Por exemplo, no desenvolvimento de jogos como o Pong, a Programação Orientada a Objetos (POO) em Java se beneficia de uma "estrutura de código modular e escalável", onde as interações entre os objetos (bola, jogadores) são definidas por lógicas que podem ser expressas por operações de conjuntos e proposições. A lógica por trás da detecção de colisões e o comportamento dos "paddles" em um jogo, por exemplo, pode ser modelada por expressões lógicas e de conjuntos.

No contexto da proteção de dados pessoais, como abordado pela LGPD, a capacidade de definir e operar sobre conjuntos de dados (e.g., dados sensíveis, dados anonimizados) é essencial. A LGPD estabelece "diretrizes específicas para o tratamento de dados", e a organização e filtragem desses dados podem ser implementadas com base nos princípios de conjuntos para garantir conformidade e privacidade. A auditoria de acesso a dados, por exemplo, pode envolver a verificação de se um usuário pertence ao conjunto de usuários autorizados (lógica de conjuntos) e se a ação solicitada está em conformidade com as regras de acesso (lógica proposicional).

Em suma, a "lógica computacional contém assuntos multidisciplinares e contribuirá para formação do seu senso crítico, pensamento lógico e racional". A prática com operações de conjuntos e a resolução de problemas utilizando os conceitos discutidos neste relatório são essenciais para o "desenvolvimento do raciocínio lógico e para a resolução de problemas que envolvem contagem e relações entre conjuntos no dia a dia", preparando profissionais para os desafios de um mundo cada vez mais informatizado e conectado.

5 CONCLUSÃO

A exploração dos conceitos de álgebra de conjuntos e lógica proposicional, aliada à sua aplicação prática em Python, reafirma a relevância desses fundamentos para a formação de profissionais em tecnologia. As operações de conjuntos – união, intersecção, diferença e diferença simétrica – juntamente com o Princípio da Inclusão-Exclusão, fornecem ferramentas poderosas para a organização e análise de dados em diversos cenários computacionais. A situação-problema resolvida através do código Python exemplificou de forma clara como essas teorias se traduzem em soluções concretas para problemas de contagem e gestão de informações.

A evolução da lógica, desde suas raízes aristotélicas até a lógica booleana e as abordagens não clássicas, demonstra a adaptabilidade e a crescente sofisticação do raciocínio formal na computação. A compreensão dos conectivos lógicos e da Tabela Verdade como métodos

para avaliar a veracidade de proposições e fórmulas é indispensável para a construção de algoritmos robustos e confiáveis.

Finalmente, a discussão sobre as implicações desses conhecimentos em áreas como a Inteligência Artificial, a segurança de dados e o desenvolvimento de jogos ressalta a natureza transversal da lógica e da teoria de conjuntos. Investir na compreensão aprofundada desses temas é crucial para o desenvolvimento do pensamento crítico e analítico, capacitando os futuros profissionais a inovar e a resolver os desafios tecnológicos emergentes de forma eficaz e ética.

REFERÊNCIAS

ABAR, C. A. A. P. *Noções de lógica matemática: Esboço do desenvolvimento da lógica*. Pontifícia Universidade Católica de São Paulo, Faculdade de Ciências Exatas e Tecnologia. [S. l.], 2004. Disponível em: https://www.pucsp.br/~logica/. Acesso em: [Data de Acesso].

ALENCAR FILHO, E. *Iniciação à lógica matemática*. São Paulo: Nobel, 2002.

BISPO, F.; CASTANHEIRA, L. B. *Introdução a lógica matemática*. São Paulo: Cengage Learning, 2011.

BRASIL. Portal Brasileiro de dados abertos. *1º Sem 2019*. Brasília, 2019. Disponível em: http://dados.gov.br/dataset/serie-historica-de-precos-de-combustiveis-por-revenda/resource/927fe-124-0648-4b17-8f05-e1ccf39ab358. Acesso em: [Data de Acesso].

BUCHSBAUM, A. *Lógica geral*. São José: UFSC, 2006.

CABRAL, J. F. P. *Lógica de Aristóteles*. Brasil Escola, [s.d.]. Disponível em: https://brasilescola.uol.com.br/filosofia/logica-aristoteles.htm. Acesso em: [Data de Acesso].

CABRAL, R. M. P. *Matemática Discreta*. Fortaleza: EdUECE, 2017.

CHIBENI, S. S. *Notas sobre lógica: o condicional*. Disponível em: https://www.unicamp.br/~chibeni/textosdidaticos/condicional.pdf. Acesso em: [Data de Acesso].

DACHI, E. P.; HAUPT, A. G. *Eletrônica digital*. São Paulo: Blucher, 2018.

DEITEL, Paul; DEITEL, Harvey. *Java: Como Programar*. 11. ed. São Paulo: Pearson, 2017.

FERREIRA, J. C. *Elementos de lógica matemática e teoria dos conjuntos*. Lisboa: Departamento de Matemática do Instituto Superior Técnico, 2001.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. *Lógica de Programação: a construção de algoritmos e estruturas de dados*. 3. ed. São Paulo: Person Prentice Hall, 2005.

GERSTING, J. L. *Fundamentos matemáticos para a ciência da computação: matemática discreta e suas aplicações*. 7. ed. Rio de Janeiro: LTC, 2017. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788521633303/cfi/6/10!/4/50@0:3.16. Acesso em: [Data de Acesso].

IEZZI, G. et al. *Matemática: ciências e aplicações*. 2. ed. São Paulo: Atual, 2004.

JAPIASSÚ, H.; MARCONDES, D. *Dicionário básico de filosofia*. 4. ed. aum. Rio de Janeiro: Jorge Zahar, 2006.

KANT, I. *Crítica da Razão Pura*. Petrópolis: Editora Vozes, 2015.

LIMA, Thiago Pinheiro Felix da Silva e. *Lógica Computacional*. Londrina: Editora e Distribuidora Educacional S.A., 2020. [Adaptei o formato da referência para incluir os três autores do livro, como indicado no CIP, mas mantendo a atribuição ao "Docs" material.]

LOPES, R. J. *Aplicações e Desafios da Inteligência Artificial na Indústria: Uma Revisão Crítica*. Journal of Industrial Technology, v. 31, n. 2, p. 56-74, 2022.

LUCIDCHART. *O que é um diagrama de árvore de decisão?* Lucid Software Inc., [s. l.], 2020. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-arvore-de-decisao#section_0. Acesso em: [Data de Acesso].

MACHADO, N. J; CUNHA, M. O. da. *Lógica e linguagem cotidiana: verdade, coerência, comunicação, argumentação*. 2. ed. Belo Horizonte: Autêntica, 2008.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. de. *Algoritmos: lógica para desenvolvimento de programação de computadores*. 29. ed. São Paulo: Érica, 2019.

MARTINS, E. T.; MARTINS, I. T. A lógica fuzzi na operacionalização de conhecimentos em interação de tarefas humano-computador em máquinas complexas: a aprendizagem em conjuntos de significância. *Revista Internacional de Aprendizaje y Cibersociedad*, [s. l.], v. 19, n. 2, p. 153-177, 2015.

MUNDIM, R. P. A Lógica Formal – princípios elementares. *Revista Economia & Gestão*, Belo Horizonte, v. 2, n. 3, jan./jun. 2002.

NOVAES, G. P. Reflexões sobre o ensino de conjuntos Diagramas de Venn. *Revista do Professor de Matemática*, São Paulo, v. 32, n. 84, p. 44-47, maio/ago. 2014.

OLIVEIRA, K. E. C. S. *Uma introdução sobre lógicas não-clássicas*. Bauru: Universidade Estadual Paulista, 2010. Disponível em: http://www2.fc.unesp.br/matematica/semana/arquivos/lnc.pdf. Acesso em: [Data de Acesso].

ORACLE. *The Java Tutorials*. Disponível em: https://docs.oracle.com/javase/tutorial/. Acesso em: 07 set. 2024.

PEREIRA, R. M. M.; SODRÉ, U. *Teoria dos conjuntos*. Londrina, 17 nov. 2006. Disponível em: http://www.uel.br/projetos/matessencial/medio/conjuntos/conjunto.htm#conj10. Acesso em: [Data de Acesso].

PICADO, J. *Que é a matemática discreta?* Coimbra: Departamento de Matemática da Universidade de Coimbra, 2008. Disponível em:

http://www.mat.uc.pt/~picado/ediscretas/2008/apontamentos/oque.pdf. Acesso em: [Data de Acesso].

Regulamento Geral. *28° ENCONTRO DE ATIVIDADES CIENTÍFICAS - EAC*. [Local de publicação]: [Editora/Instituição], [Ano]. [Referência genérica para o regulamento do evento].

SANTOS, Kayo Adrian M.; CARMO, Maxwell Nunes do. *A APLICAÇÃO DA INTELIGÊNCIA ARTIFICIAL NA INDÚSTRIA: BENEFÍCIOS E DESAFIOS*. PITÁGORAS - FACULDADE PITÁGORAS DE UBERLÂNDIA. [Ano]. [Referência adaptada do IA.pdf].

SANTOS, R. *Paradoxos semânticos*. Lisboa: Centro de Filosofia da Universidade de Lisboa, 2014.

SCHEINERMAN, E. R. *Matemática discreta: uma introdução*. São Paulo: Cengage Learning, 2015.

SILVA, F. S. C. da; FINGER, M.; MELO, A. C. V. de. *Lógica para computação*. 2. ed. São Paulo: Cengage Learning, 2017.

SILVA, Tatiana Estér Thainá Morais da; BARBOSA, Bruno Fernandes; CHAVES, Heloisa Soares; CRUZ, Jhennyfer Hillary Amorim da; GARCIA, Kleyton Fernando da Cruz. *A APLICAÇÃO DO DIREITO CIVIL NA PROTEÇÃO DE DADOS PESSOAIS LGPD*. ANHANGUERA - FACULDADE ANHANGUERA. [Ano]. [Referência adaptada do LGPD.pdf].

SOFFNER, R. *Algoritmos e programação em linguagem C*. São Paulo: Saraiva, 2013.

SOUZA, J. A. L. de S. (org.). *Lógica matemática*. São Paulo: Pearson Education do Brasil, 2016.

YAGLOM, I. M. *Álgebra booleana*. São Paulo: Atual, 1998.

ZEGARELLI, M. *Lógica para leigos*. Rio de Janeiro: Alta Books, 2013.