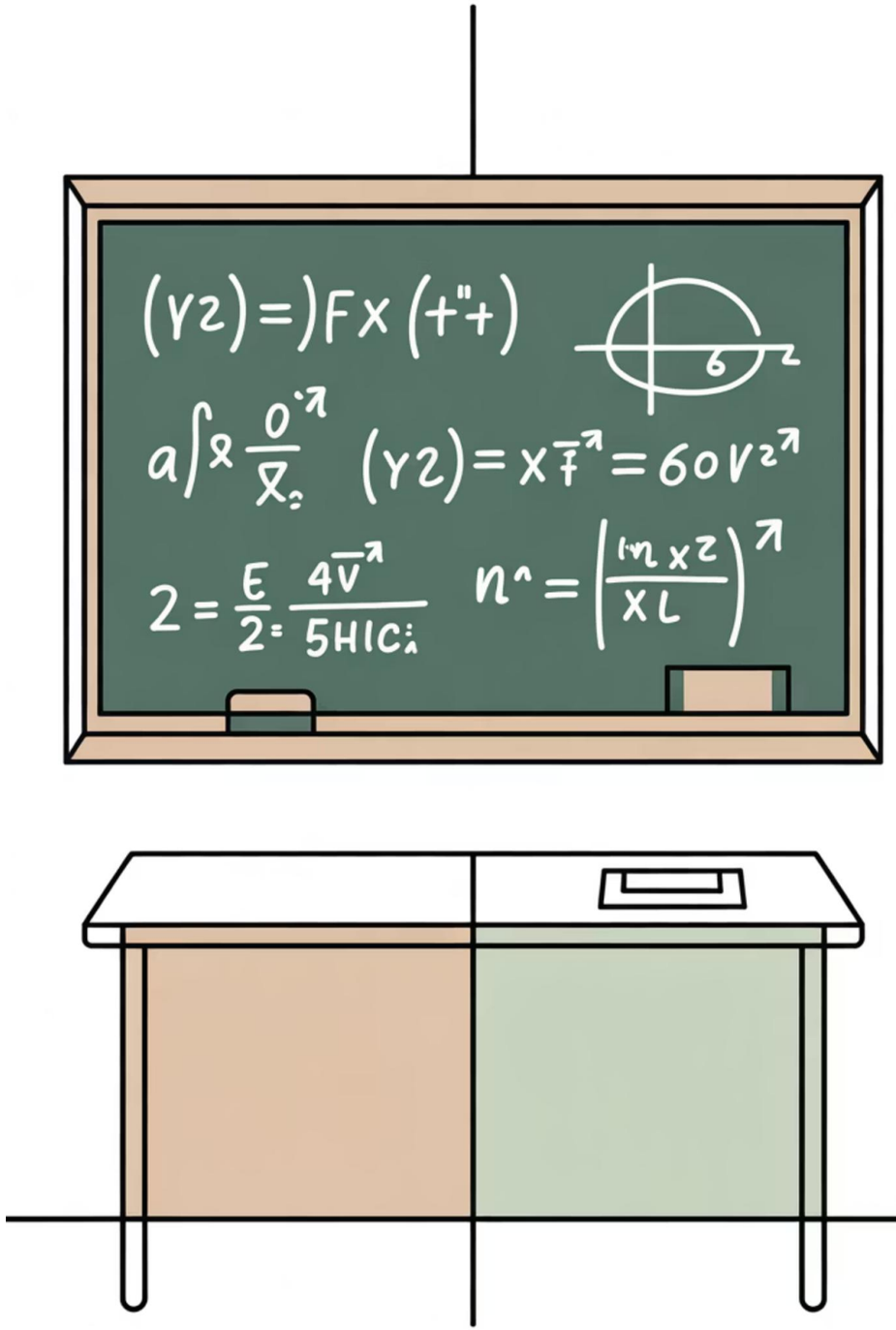
CEFET/RJ

Escrita de Formalização

A comunicação precisa de conceitos matemáticos e computacionais por meio de linguagem formal rigorosa e estruturada.



Eduardo Ogasawara
eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

Formalização Matemática e Computacional

A formalização é o processo fundamental de expressar conceitos matemáticos e computacionais usando linguagem precisa, notação padronizada e estrutura lógica rigorosa. Este processo permite a comunicação inequívoca de ideias complexas e a verificação sistemática de resultados.

Na matemática, a formalização transforma intuições em definições rigorosas e argumentos em provas verificáveis. Na ciência da computação, ela permite especificar algoritmos e sistemas com precisão absoluta.

A linguagem formal elimina ambiguidades presentes na linguagem natural, estabelecendo um vocabulário técnico compartilhado pela comunidade científica. Cada símbolo, cada operador e cada construção possui significado bem definido.

Dominar a escrita formal é essencial para estudantes e pesquisadores, pois representa a base para a comunicação científica, o desenvolvimento de teorias e a implementação de sistemas computacionais confiáveis.

Definições, Teoremas e Provas

Os conceitos fundamentais da matemática formal formam a espinha dorsal do raciocínio lógico e da construção do conhecimento matemático. Cada elemento possui um papel específico na arquitetura do pensamento formal:

Definição

Estabelece o significado matemático preciso de um conceito, eliminando ambiguidades através de linguagem formal rigorosa.

Lema e Proposição

Uma proposição é uma afirmação matemática verdadeira; um lema é um resultado auxiliar que facilita a demonstração de teoremas mais complexos.

Teorema

Representa uma proposição verdadeira de grande importância e impacto, geralmente requerendo demonstração elaborada.

Corolário

Um resultado que decorre diretamente de um teorema já demonstrado, geralmente com prova simples ou imediata.

Prova

Demonstração rigorosa e lógica da veracidade de uma proposição, usando axiomas, definições e resultados previamente estabelecidos.

Conjectura

Afirmação que se acredita verdadeira com base em evidências, mas ainda sem prova formal completa.

Axioma

Suposição fundamental aceita como verdadeira sem necessidade de demonstração, servindo de base para todo o sistema.

Referências: [1] S.G. Krantz, 2017, *A Primer of Mathematical Writing*. American Mathematical Society. [2] K. Houston, 2009, *How to Think Like a Mathematician: A Companion to Undergraduate Mathematics*. Cambridge University Press.

Elementos Essenciais na Escrita de Teoremas

A escrita eficaz de teoremas requer atenção cuidadosa a múltiplos elementos que garantem clareza, precisão e compreensibilidade. Cada componente desempenha papel crucial na comunicação matemática:

01

Enunciado Claro e Preciso

O teorema deve ser formulado de maneira inequívoca, estabelecendo claramente as hipóteses e a conclusão. A linguagem deve ser formal, mas acessível ao público-alvo.

02

Hipóteses Bem Definidas

Todas as condições necessárias devem ser explicitadas completamente, sem deixar pressupostos implícitos que possam gerar confusão ou interpretações errôneas.

03

Notação Consistente

Utilizar símbolos padronizados e manter consistência na notação ao longo de todo o texto, definindo claramente cada variável e operador introduzido.

04

Estrutura Lógica

Organizar o argumento de forma sequencial e coerente, apresentando cada passo de maneira clara e justificada, facilitando a verificação da validade.

05

Contextualização Adequada

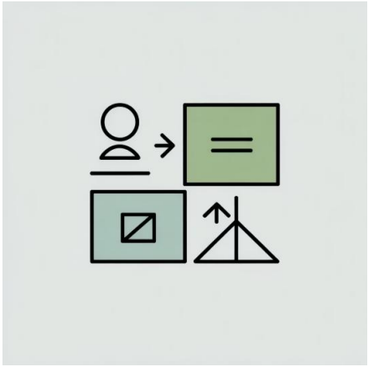
Fornecer motivação e contexto quando apropriado, explicando a relevância do resultado e suas conexões com outros teoremas estabelecidos.

Provas Matemáticas

Uma prova matemática é um argumento lógico rigoroso que estabelece a verdade de uma proposição de forma inquestionável. Ela constitui o coração da matemática, diferenciando-a de outras ciências através de seu padrão de certeza absoluta.

Existem diversos tipos de provas, cada uma adequada a diferentes situações:

- Prova direta: demonstra a conclusão seguindo logicamente das hipóteses
- Prova por contradição: assume a negação da conclusão e deriva uma contradição
- Prova por indução: estabelece casos base e passo indutivo
- Prova construtiva: fornece exemplo explícito do objeto cuja existência se afirma
- Prova por contrapositiva: demonstra que a negação da conclusão implica a negação da hipótese



Uma prova bem escrita deve ser completa, sem lacunas lógicas, mas também clara e compreensível. O equilíbrio entre rigor e legibilidade é essencial para comunicação efetiva.

Referências: [1] S.G. Krantz, 2017, *A Primer of Mathematical Writing*. American Mathematical Society. [2] K. Houston, 2009, *How to Think Like a Mathematician: A Companion to Undergraduate Mathematics*. Cambridge University Press.

Escrita Clara e Bem Estruturada de Formalizações

Clareza de Propósito

Defina objetivos explícitos no início, estabelecendo o que será demonstrado e por que é importante.

Estrutura Lógica

Organize o conteúdo em seções coerentes, progredindo das definições básicas aos resultados principais.

Precisão Técnica

Use terminologia rigorosa e notação padronizada, evitando ambiguidades e imprecisões.

Compreensibilidade

Balance rigor com explicações intuitivas, auxiliando o leitor na compreensão dos conceitos.

A escrita eficaz de formalizações matemáticas e computacionais exige domínio tanto do conteúdo técnico quanto das técnicas de comunicação. Textos bem escritos apresentam argumentos de forma linear e progressiva, introduzindo cada conceito no momento apropriado e justificando cada passo com clareza.

É fundamental estabelecer convenções notacionais logo no início do texto e mantê-las consistentemente. Definições devem preceder seu uso, e teoremas devem ser enunciados antes de suas provas. A divisão em seções temáticas facilita a navegação e compreensão, permitindo que leitores identifiquem rapidamente os elementos de interesse.

Referência: [2] J. Zobel, 2015, *Writing for Computer Science*. Springer.

Exemplo de Escrita Eficiente com Boa Formatação

> A formatação adequada transforma conceitos complexos em conteúdo acessível e profissional.

Hierarquia Visual Clara

Utilize títulos, subtítulos e numeração para estabelecer a estrutura hierárquica do documento. Destaque definições, teoremas e provas usando formatação distinta, como negrito ou itálico para termos técnicos importantes.

Espaçamento Apropriado

Mantenha espaçamento adequado entre parágrafos, equações e seções. Equações importantes devem ser centralizadas e numeradas para referência futura. O espaço em branco melhora legibilidade significativamente.

Referências e Citações

Inclua referências bibliográficas completas e consistentes. Cite adequadamente resultados conhecidos e trabalhos anteriores, demonstrando conhecimento do contexto histórico e acadêmico do tema.

Exemplos Ilustrativos

Forneça exemplos concretos após definições abstratas e teoremas gerais. Exemplos ajudam a consolidar compreensão e demonstram aplicabilidade prática dos conceitos apresentados.

Um documento bem formatado não apenas facilita a leitura, mas também demonstra profissionalismo e respeito pelo leitor. A consistência na formatação ao longo de todo o texto é essencial para manter credibilidade acadêmica.

Pseudocódigo

O pseudocódigo é uma ferramenta fundamental na ciência da computação, permitindo expressar algoritmos de forma independente de linguagens de programação específicas. Ele combina estruturas algorítmicas com linguagem natural, proporcionando clareza sem as restrições sintáticas de linguagens formais.

Características essenciais:

- Clareza e legibilidade superiores ao código real
- Independência de sintaxe específica de linguagens
- Foco na lógica e estrutura do algoritmo
- Organização hierárquica com indentação
- Ausência de ambiguidades na descrição

The **WeightedEdit** function computes the edit distance between two strings, assigning a higher penalty for errors closer to the front.

Input: $S1, S2$: strings to be compared.
Output: weighted edit distance
Variables: $L1, L2$: string lengths
$F[L1, L2]$: array of minimum distances
W: current weighting
M: maximum penalty
C: current penalty

WeightedEdit$(S1, S2)$:
1. $L1 = len(S1)$
2. $L2 = len(S2)$
3. $M = 2 \times (L1 + L2)$
4. $F[0,0] = 0$
5. **for** i **from** 1 **to** $L1$
6. $\quad F[i,0] = F[i-1,0] + M - i$
7. **for** j **from** 1 **to** $L2$
8. $\quad F[0,j] = F[0,j-1] + M - j$
9. **for** i **from** 1 **to** $L1$
10. $\quad C = M - i$
11. $\quad$ **for** j **from** 1 **to** $L2$
12. $\quad\quad C = C - 1$
13. $\quad\quad F[i,j] = min(F[i-1,j] + C,$
$\quad\quad\quad\quad\quad\quad F[i,j-1] + C,$
$\quad\quad\quad\quad\quad\quad F[i-1,j-1] + C \times isdiff(S1[i], S2[j]))$
14. **WeightedEdit** $= F[L1, L2]$

O pseudocódigo deve ser suficientemente detalhado para permitir implementação direta em qualquer linguagem de programação, mas abstrato o suficiente para não se prender a idiossincrasias específicas. Convenções de indentação e nomenclatura clara de variáveis são cruciais para manter legibilidade.

Referência: [2] J. Zobel, 2015, *Writing for Computer Science*. Springer.

Pseudocódigo: Encapsulamento em Funções

A divisão de algoritmos complexos em funções modulares e reutilizáveis representa uma prática fundamental de engenharia de software e escrita algorítmica. O encapsulamento melhora significativamente a qualidade e manutenibilidade do código.

Modularidade

Cada função executa uma tarefa específica e bem definida, facilitando compreensão e teste isolado de componentes individuais.

Reutilização

Funções podem ser chamadas múltiplas vezes em diferentes contextos, evitando duplicação de código e reduzindo erros.

Abstração

Detalhes de implementação são ocultados, permitindo que usuários da função se concentrem apenas em sua interface e propósito.

Manutenibilidade

Alterações em funcionalidades específicas podem ser realizadas localmente sem afetar o restante do sistema.

```
SCHEDULER(Workflow W, ExecutionProcessesSet execProc)
1.   FragmentSet fragSet = optimizeWorkflow(W);
2.   while (hasElements(fragSet))
3.      frag = getReadyFragment(fragSet);
4.      FAISet faiSet = generateActivations(frag);
5.      DispatchStrategy dispStrat = getDispatchStrategy(frag);
6.      for each FAI f in faiSet
7.        dispatch(f, dispStrat, execProc);
8.      fragSet = removeCompleted(fragSet, frag);
```

Figure 8. Scheduler Algorithm.

Ao projetar funções, é essencial escolher nomes descritivos, definir claramente parâmetros de entrada e saída, e documentar o comportamento esperado. Funções bem projetadas possuem responsabilidade única e interface simples.

Referência: [1] E. Ogasawara, D. De Oliveira, P. Valduriez, J. Dias, F. Porto, e M. Mattoso, "An Algebraic Approach for Data-Centric Scientific Workflows", *Proceedings of the VLDB Endowment*, vol. 4, no 12, p. 1328–1339, 2011.

Prosecode

O Prosecode representa uma abordagem híbrida entre pseudocódigo formal e explicação em linguagem natural. Esta técnica foca na compreensão intuitiva do algoritmo, tornando-o acessível a uma audiência mais ampla enquanto mantém estrutura clara.

Características Distintivas

- Mistura linguagem natural com estrutura algorítmica
- Prioriza clareza conceitual sobre precisão sintática
- Facilita compreensão inicial antes de implementação formal
- Ideal para documentação e comunicação de ideias
- Reduz barreira de entrada para não-programadores

O Prosecode é particularmente útil em contextos educacionais e documentação técnica, onde o objetivo principal é transmitir a lógica e estratégia do algoritmo, não necessariamente sua implementação precisa.

WeightedEdit(s,t) compares two strings s and t, of lengths k_s and k_t respectively, to determine their edit distance—the minimum cost in insertions, deletions, and replacements required to convert one into the other. These costs are weighted so that errors near the start of the strings attract a higher penalty than errors near the end.

We denote the ith character of string s by s_i. The principal internal data structure is a 2-dimensional array F in which the dimensions have ranges 0 to k_s and 0 to k_t, respectively. When the array is filled, $F_{i,j}$ is the minimum edit distance between the strings $s_1 \cdots s_i$ and $t_1 \cdots t_j$; and F_{k_s,k_t} is the minimum edit distance between s and t.

The value p is the maximum penalty, and the penalty for a discrepancy between positions i and j of s and t, respectively, is $p-i-j$, so that the minimum penalty is $p-k_s-k_t = p/2$ and the next-smallest penalty is $p/2+1$. Two errors, wherever they occur, will outweigh one.

1. (Set penalty.) Set $p \leftarrow 2 \times (k_s + k_t)$.
2. (Initialize data structure.) The boundaries of array F are initialized with the penalty for deletions at start of string; for example, $F_{i,0}$ is the penalty for deleting i characters from the start of s.
   (a) Set $F_{0,0} \leftarrow 0$.
   (b) For each position i in s, set $F_{i,0} \leftarrow F_{i-1,0} + p - i$.
   (c) For each position j in t, set $F_{0,j} \leftarrow F_{0,j-1} + p - j$.
3. (Compute edit distance.) For each position i in s and position j in t:
   (a) The penalty is $C = p - i - j$.
   (b) The cost of inserting a character into t (equivalently, deleting from s) is $I = F_{i-1,j} + C$.
   (c) The cost of deleting a character from t is $D = F_{i,j-1} + C$.
   (d) If s_i is identical to t_j, the replacement cost is $R = F_{i-1,j-1}$. Otherwise, the replacement cost is $R = F_{i-1,j-1} + C$.
   (e) Set $F_{i,j} \leftarrow \min(I, D, R)$.
4. (Return.) Return F_{k_s,k_t}.

Referência: [1] J. Zobel, 2015, *Writing for Computer Science*. Springer.

CEFET/RJ

Introdução à Metodologia Científica

Eduardo Ogasawara
eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara