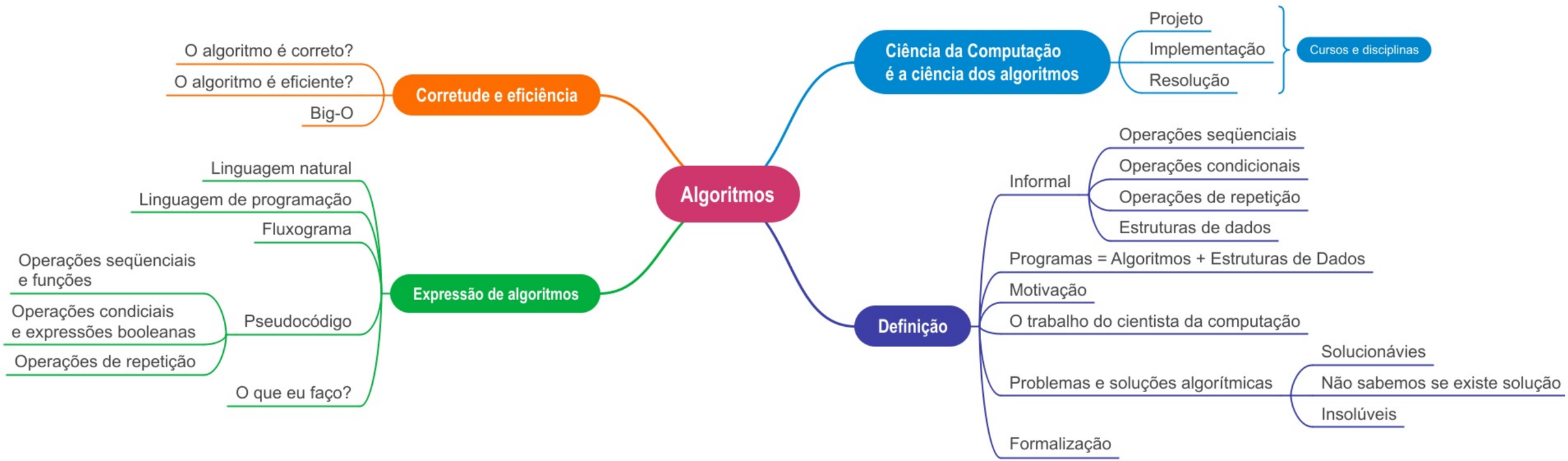
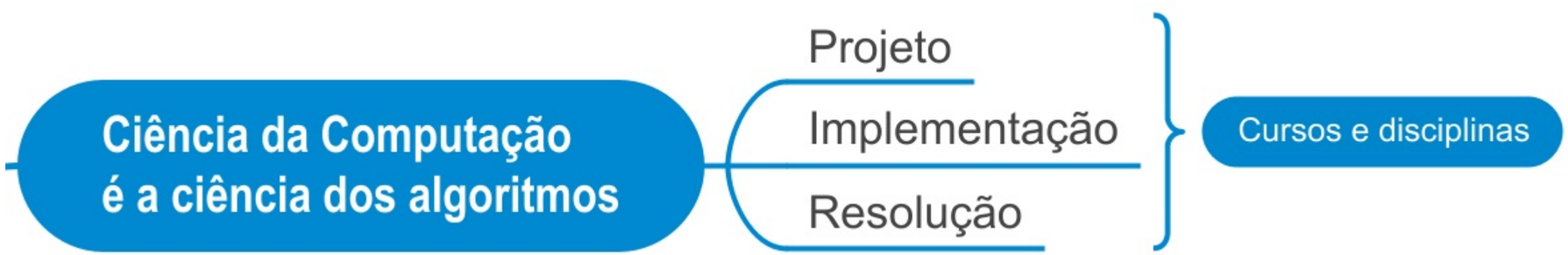
FUNDAMENTOS DA COMPUTAÇÃO

Algoritmos


Algoritmos
Corretude e eficiência
O algoritmo é correto?
O algoritmo é eficiente?
Big-O
Expressão de algoritmos
Linguagem natural
Linguagem de programação
Fluxograma
Pseudocódigo
Operações seqüenciais e funções
Operações condiciais e expressões booleanas
Operações de repetição
O que eu faço?
Ciência da Computação é a ciência dos algoritmos
Projeto
Implementação
Resolução
Cursos e disciplinas
Definição
Informal
Operações seqüenciais
Operações condicionais
Operações de repetição
Estruturas de dados
Programas = Algoritmos + Estruturas de Dados
Motivação
O trabalho do cientista da computação
Problemas e soluções algorítmicas
Solucionávies
Não sabemos se existe solução
Insolúveis
Formalização

Algoritmos


Ciência da Computação
é a ciência dos algoritmos
Projeto
Implementação
Resolução
Cursos e disciplinas

Ciência da Computação: a ciência dos algoritmos

Até agora vimos qua a Ciência da Computação:

> **É a ciência que projeta e implementa algoritmos para a resolução de problemas.**

O que realmente é esse "projeto"?

O que realmente é essa "implementação"?

Quais são realmente os "problemas a serem solucionados"?

O que realmente são os "algoritmos"?

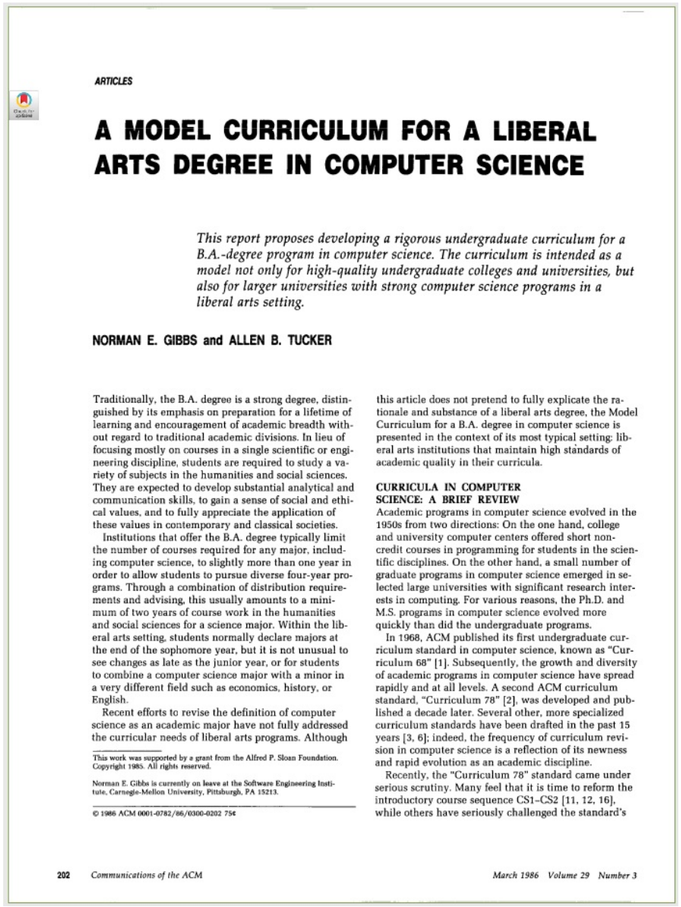
Ciência da Computação: a ciência dos algoritmos

É a ciência que projeta e implementa algoritmos para a resolução de problemas.

Na parte do "projeto" estamos interessados nas:
Propriedades formais e matemáticas dos algoritmos.

Na parte da "implementação" estamos interessados na:
Realização linguística e realização em hardware dos algoritmos (incluindo a máquina virtual com a qual o usuário interage).

Na parte de "resolução de problemas" estamos interessados nas:
Aplicações possíveis dos algoritmos.



ARTICLES

A MODEL CURRICULUM FOR A LIBERAL ARTS DEGREE IN COMPUTER SCIENCE

This report proposes developing a rigorous undergraduate curriculum for a B.A.-degree program in computer science. The curriculum is intended as a model not only for high-quality undergraduate colleges and universities, but also for larger universities with strong computer science programs in a liberal arts setting.

NORMAN E. GIBBS and ALLEN B. TUCKER

Traditionally, the B.A. degree is a strong degree, distinguished by its emphasis on preparation for a lifetime of learning and encouragement of academic breadth without regard to traditional academic divisions. In lieu of focusing mostly on courses in a single scientific or engineering discipline, students are required to study a variety of subjects in the humanities and social sciences. They are expected to develop substantial analytical and communication skills, to gain a sense of social and ethical values, and to fully appreciate the application of these values in contemporary and classical societies.

Institutions that offer the B.A. degree typically limit the number of courses required for any major, including computer science, to slightly more than one year in order to allow students to pursue diverse four-year programs. Through a combination of distribution requirements and advising, this usually amounts to a minimum of two years of course work in the humanities and social sciences for a science major. Within the liberal arts setting, students normally declare majors at the end of the sophomore year, but it is not unusual to see changes as late as the junior year, or for students to combine a computer science major with a minor in a very different field such as economics, history, or English.

Recent efforts to revise the definition of computer science as an academic major have not fully addressed the curricular needs of liberal arts programs. Although this article does not pretend to fully explicate the rationale and substance of a liberal arts degree, the Model Curriculum for a B.A. degree in computer science is presented in the context of its most typical setting: liberal arts institutions that maintain high standards of academic quality in their curricula.

This work was supported by a grant from the Alfred P. Sloan Foundation.

Norman E. Gibbs is currently on leave at the Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA 15213.

© 1986 ACM 0001-0782/86/0300-0202 75¢

CURRICULA IN COMPUTER SCIENCE: A BRIEF REVIEW

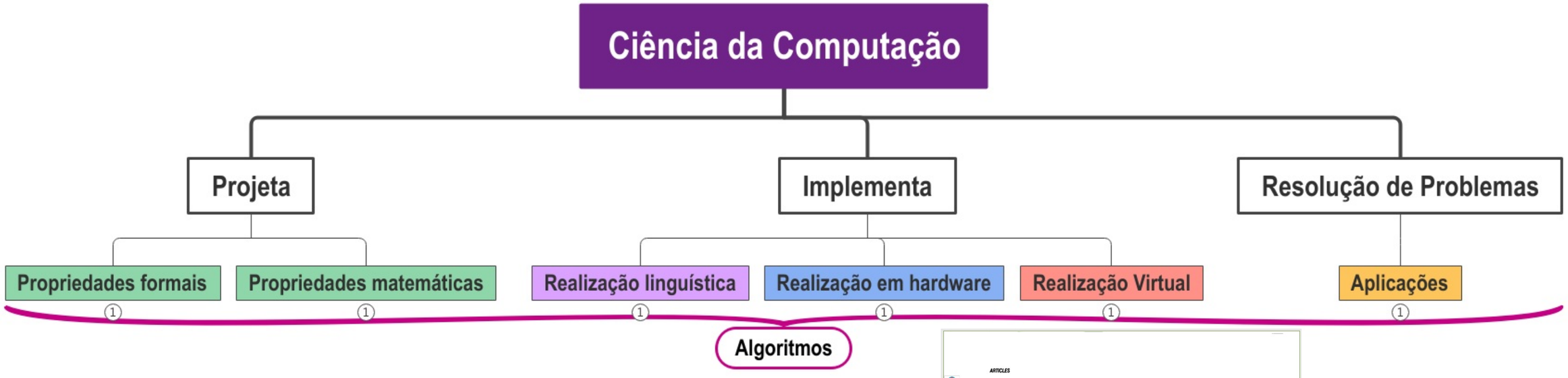
Academic programs in computer science evolved in the 1950s from two directions: On the one hand, college and university computer centers offered short noncredit courses in programming for students in the scientific disciplines. On the other hand, a small number of graduate programs in computer science emerged in selected large universities with significant research interests in computing. For various reasons, the Ph.D. and M.S. programs in computer science evolved more quickly than did the undergraduate programs.

In 1968, ACM published its first undergraduate curriculum standard in computer science, known as "Curriculum 68" [1]. Subsequently, the growth and diversity of academic programs in computer science have spread rapidly and at all levels. A second ACM curriculum standard, "Curriculum 78" [2], was developed and published a decade later. Several other, more specialized curriculum standards have been drafted in the past 15 years [3, 6]; indeed, the frequency of curriculum revision in computer science is a reflection of its newness and rapid evolution as an academic discipline.

Recently, the "Curriculum 78" standard came under serious scrutiny. Many feel that it is time to reform the introductory course sequence CS1–CS2 [11, 12, 16], while others have seriously challenged the standard's

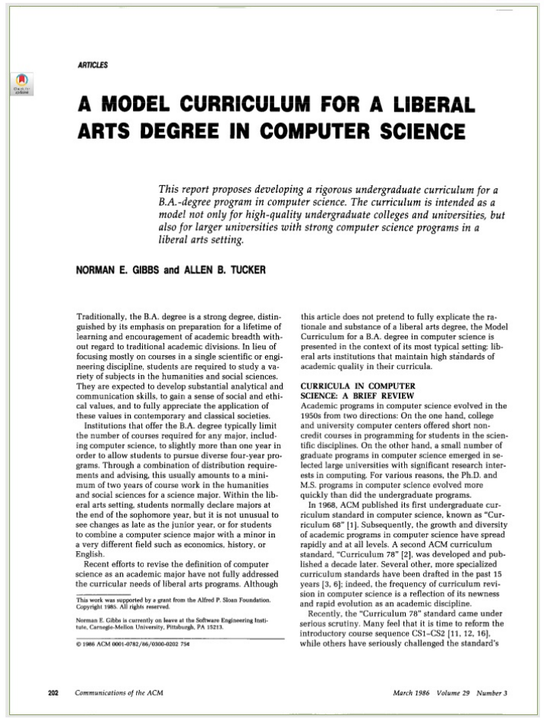
Gibbs & Tucker: Communications of the ACM, vol. 29, n.º 3, 1986.

Ciência da Computação: a ciência dos algoritmos



Dependendo do foco, temos os 4 "grandes cursos":

- **Ciência da computação**
- **Engenharia da computação**
- **Engenharia de software**
- **Sistemas de informação**



ARTICLES

A MODEL CURRICULUM FOR A LIBERAL ARTS DEGREE IN COMPUTER SCIENCE

This report proposes developing a rigorous undergraduate curriculum for a B.A.-degree program in computer science. The curriculum is intended as a model not only for high-quality undergraduate colleges and universities, but also for larger universities with strong computer science programs in a liberal arts setting.

NORMAN E. GIBBS and ALLEN B. TUCKER

Traditionally, the B.A. degree is a strong degree, distinguished by its emphasis on preparation for a lifetime of learning and encouragement of academic breadth without regard to traditional academic divisions. In lieu of focusing mostly on courses in a single scientific or engineering discipline, students are required to study a variety of subjects in the humanities and social sciences. They are expected to develop substantial analytical and communication skills, to gain a sense of social and ethical values, and to fully appreciate the application of these values in contemporary and classical societies.

Institutions that offer the B.A. degree typically limit the number of courses required for any major, including computer science, to slightly more than one year in order to allow students to pursue diverse four-year programs. Through a combination of distribution requirements and advising, this usually amounts to a minimum of two years of course work in the humanities and social sciences for a science major. Within the liberal arts setting, students normally declare majors at the end of the sophomore year, but it is not unusual to see changes as late as the junior year, or for students to combine a computer science major with a minor in a very different field such as economics, history, or English.

Recent efforts to revise the definition of computer science as an academic major have not fully addressed the curricular needs of liberal arts programs. Although this article does not pretend to fully explicate the rationale and substance of a liberal arts degree, the Model Curriculum for a B.A. degree in computer science is presented in the context of its most typical setting: liberal arts institutions that maintain high standards of academic quality in their curricula.

This work was supported by a grant from the Alfred P. Sloan Foundation.

Norman E. Gibbs is currently on leave at the Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA 15213.

© 1986 ACM 0001-0782/86/0300-0202 75¢

CURRICULA IN COMPUTER SCIENCE: A BRIEF REVIEW

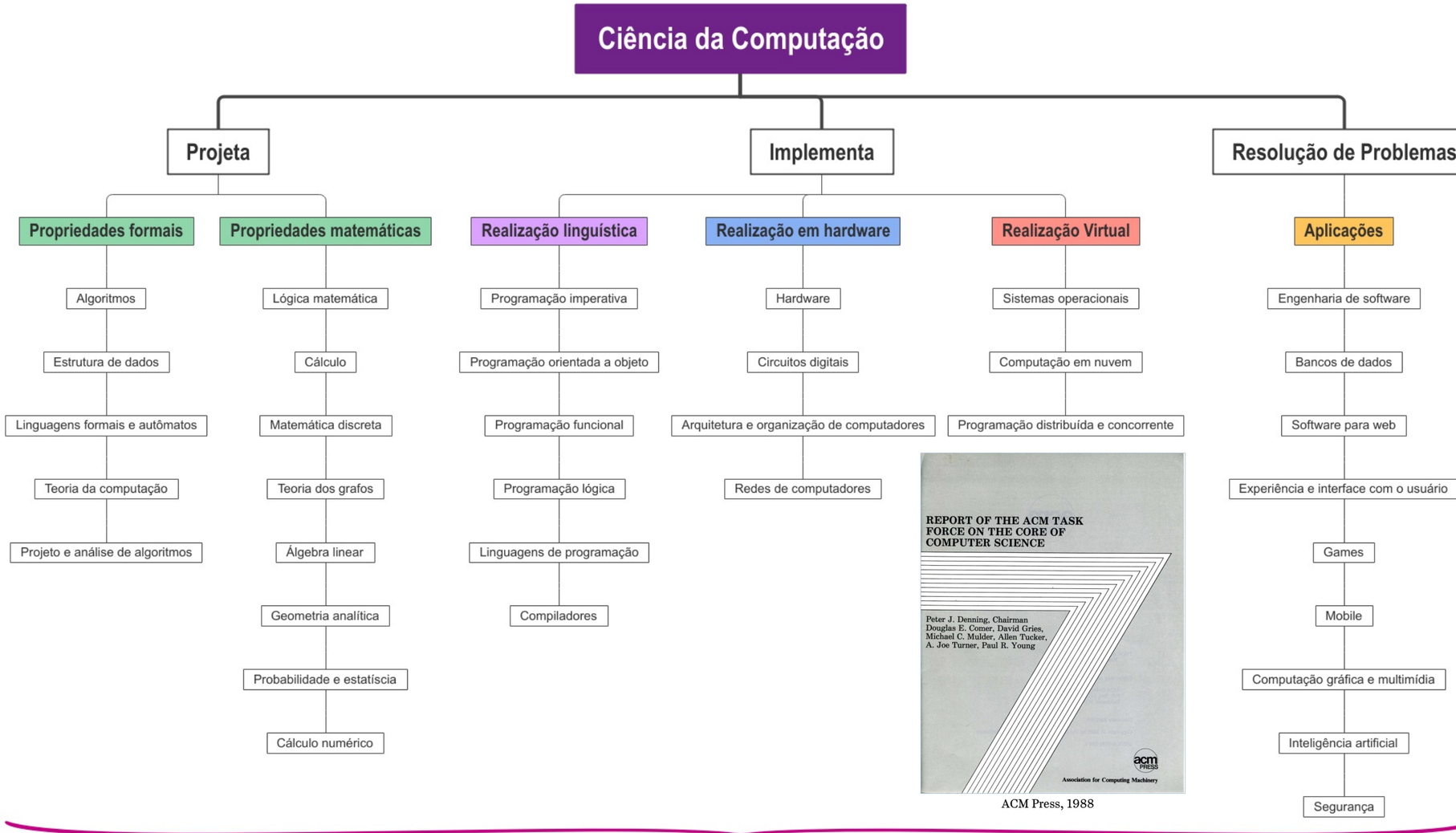
Academic programs in computer science evolved in the 1950s from two directions: On the one hand, college and university computer centers offered short non-credit courses in programming for students in the scientific disciplines. On the other hand, a small number of graduate programs in computer science emerged in selected large universities with significant research interests in computing. For various reasons, the Ph.D. and M.S. programs in computer science evolved more quickly than did the undergraduate programs.

In 1968, ACM published its first undergraduate curriculum standard in computer science, known as "Curriculum 68" [1]. Subsequently, the growth and diversity of academic programs in computer science have spread rapidly and at all levels. A second ACM curriculum standard, "Curriculum 78" [2], was developed and published a decade later. Several other, more specialized curriculum standards have been drafted in the past 15 years [3, 6]; indeed, the frequency of curriculum revision in computer science is a reflection of its newness and rapid evolution as an academic discipline.

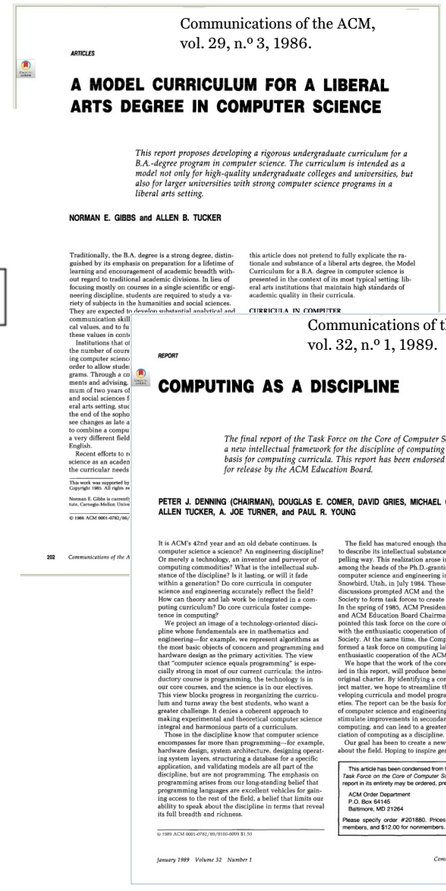
Recently, the "Curriculum 78" standard came under serious scrutiny. Many feel that it is time to reform the introductory course sequence CS1–CS2 [11, 12, 16], while others have seriously challenged the standard's

Gibbs & Tucker: Communications of the ACM, vol. 29, n.º 3, 1986.

Ciência da Computação: a ciência dos algoritmos



ACM Press, 1988



Communications of the ACM,
vol. 29, n.º 3, 1986.

ARTICLES

A MODEL CURRICULUM FOR A LIBERAL ARTS DEGREE IN COMPUTER SCIENCE

This report proposes developing a rigorous undergraduate curriculum for a B.A.-degree program in computer science. The curriculum is intended as a model not only for high-quality undergraduate colleges and universities, but also for larger universities with strong computer science programs in a liberal arts setting.

NORMAN E. GIBBS and ALLEN B. TUCKER

Communications of the ACM,
vol. 32, n.º 1, 1989.

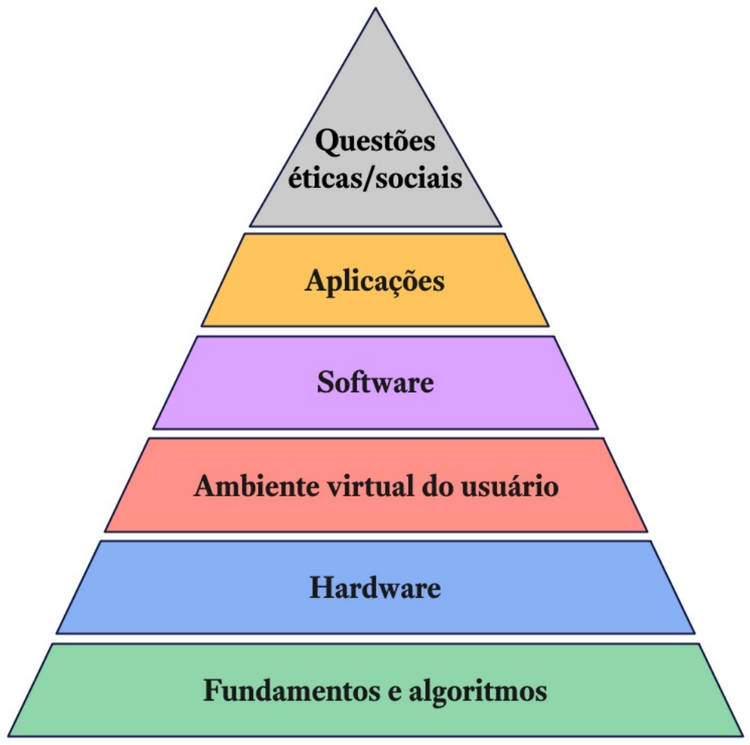
REPORT

COMPUTING AS A DISCIPLINE

The final report of the Task Force on the Core of Computer Science presents a new intellectual framework for the discipline of computing and a new basis for computing curricula. This report has been endorsed and approved for release by the ACM Education Board.

PETER J. DENNING (CHAIRMAN), DOUGLAS E. COMER, DAVID GRIES, MICHAEL C. MULDER, ALLEN TUCKER, A. JOE TURNER, and PAUL R. YOUNG

Ciência da Computação: a ciência dos algoritmos



Adaptado de: Schneider & Gersting. **Invitation to Computer Science**. 8ª ed. Cengage, 2019.

PROJETO DE ALGORITMOS

Propriedades Formais
- Algoritmos e estruturas de dados
- Linguagens formais e autômatos
- Teoria da computação
- Projeto e análise de algoritmos

Propriedades Matemáticas
- Lógica matemática
- Cálculo, eq. diferenciais
- Matemática discreta
- Teoria dos grafos
- Álgebra linear
- Geometria analítica
- Probabilidade/estatística
- Cálculo numérico

RESOLUÇÃO DE PROBLEMAS

Aplicações e questões éticas/sociais
- Engenharia de software
- Software para web
- Games
- Computação gráfica/multimídia
- Experiência e interface com usuário
- Bancos de dados
- Robótica
- Mobile
- Inteligência artificial
- Segurança

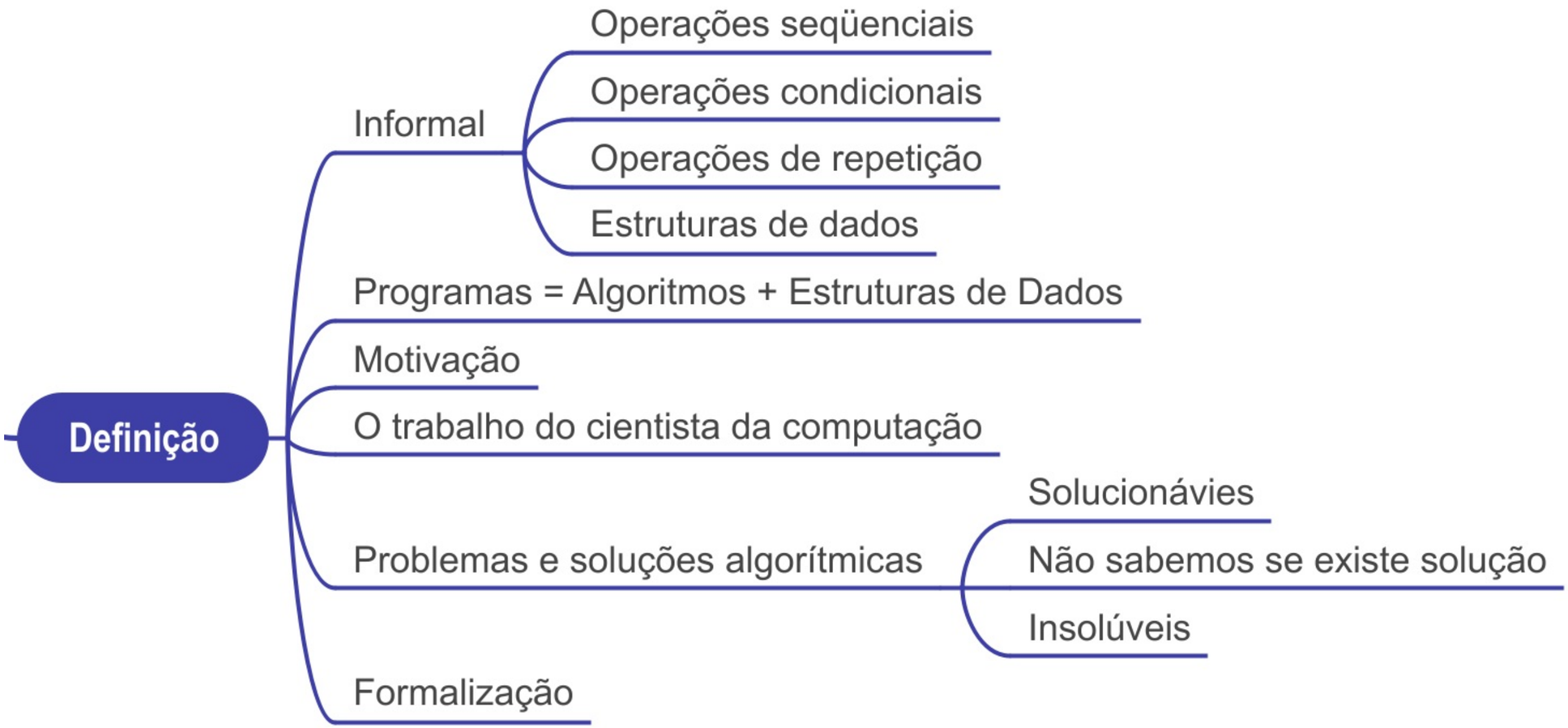
IMPLEMENTAÇÃO DE ALGORITMOS

Realização linguística
- Programação imperativa
- Programação orientada a objeto
- Programação funcional
- Programação lógica
- Linguagens de programação
- Compiladores

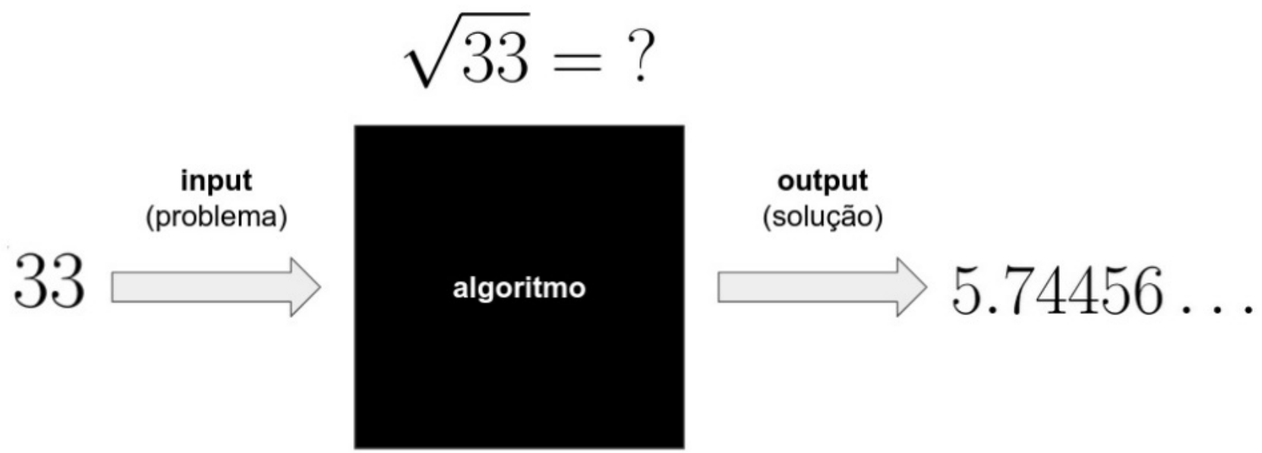
Realização em Hardware
- Hardware
- Circuitos digitais
- Arquitetura e organização de computadores
- Redes de computadores

Realização Virtual
- Sistemas operacionais
- Computação em nuvem
- Programação distribuída e concorrente

Algoritmos


Definição
Informal
Operações seqüenciais
Operações condicionais
Operações de repetição
Estruturas de dados
Programas = Algoritmos + Estruturas de Dados
Motivação
O trabalho do cientista da computação
Problemas e soluções algorítmicas
Solucionávies
Não sabemos se existe solução
Insolúveis
Formalização

Mas afinal, o que é um algoritmo?



Método de Newton:

1. Chute um valor para y;
2. Calcule o valor de y^2;
3. Se $y^2 = x$ (ou um valor próximo o suficiente), você achou a raiz. Termine o procedimento.
4. Se $y^2 \neq x$ (ou não está próximo o suficiente), melhore a estimativa de y fazendo o seguinte cálculo: novo $y = \dfrac{y + \dfrac{x}{y}}{2}$;
5. Retorne ao segundo passo até que você encontre a raiz ou uma aproximação suficiente.

Noção intuitiva, informal:

É uma receita passo a passo que nos indica como resolver um problema ou realizar uma tarefa.

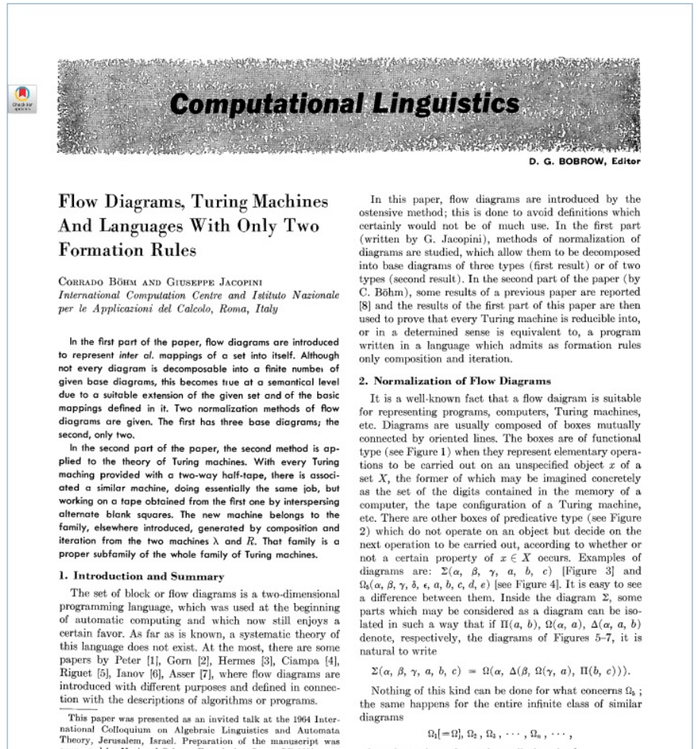
(conhecimento imperativo)

Mas afinal, o que é um algoritmo?

Uma das principais tarefas de um cientista da computação é criar e desenvolver algoritmos para resolver diversos problemas importantes para a humanidade.

Para criarmos um algoritmo são necessárias apenas 4 coisas:

- **Operações seqüenciais** (para indicar a ordem das tarefas)
- **Operações condicionais** (seleção, decisão)
- **Operações de repetição** (iteração ou recursão)
- **Estruturas de dados** (armazena dados em processamento)



Computational Linguistics

D. G. BOBROW, Editor

Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules

CORRADO BÖHM AND GIUSEPPE JACOPINI
International Computation Centre and Istituto Nazionale per le Applicazioni del Calcolo, Roma, Italy

In the first part of the paper, flow diagrams are introduced to represent *inter al.* mappings of a set into itself. Although not every diagram is decomposable into a finite number of given base diagrams, this becomes true at a semantical level due to a suitable extension of the given set and of the basic mappings defined in it. Two normalization methods of flow diagrams are given. The first has three base diagrams; the second, only two.

In the second part of the paper, the second method is applied to the theory of Turing machines. With every Turing machine provided with a two-way half-tape, there is associated a similar machine, doing essentially the same job, but working on a tape obtained from the first one by interspersing alternate blank squares. The new machine belongs to the family, elsewhere introduced, generated by composition and iteration from the two machines λ and R. That family is a proper subfamily of the whole family of Turing machines.

1. Introduction and Summary

The set of block or flow diagrams is a two-dimensional programming language, which was used at the beginning of automatic computing and which now still enjoys a certain favor. As far as is known, a systematic theory of this language does not exist. At the most, there are some papers by Peter [1], Gorn [2], Hermes [3], Ciampa [4], Riguet [5], Ianov [6], Asser [7], where flow diagrams are introduced with different purposes and defined in connection with the descriptions of algorithms or programs.

This paper was presented as an invited talk at the 1964 International Colloquium on Algebraic Linguistics and Automata Theory, Jerusalem, Israel. Preparation of the manuscript was supported by National Science Foundation Grant GP-2880.

This work was carried out at the Istituto Nazionale per le Applicazioni del Calcolo (INAC) in collaboration with the International Computation Centre (ICC), under the Italian Consiglio Nazionale delle Ricerche (CNR) Research Group No. 22 for 1963–64.

In this paper, flow diagrams are introduced by the ostensive method; this is done to avoid definitions which certainly would not be of much use. In the first part (written by G. Jacopini), methods of normalization of diagrams are studied, which allow them to be decomposed into base diagrams of three types (first result) or of two types (second result). In the second part of the paper (by C. Böhm), some results of a previous paper are reported [8] and the results of the first part of this paper are then used to prove that every Turing machine is reducible into, or in a determined sense is equivalent to, a program written in a language which admits as formation rules only composition and iteration.

2. Normalization of Flow Diagrams

It is a well-known fact that a flow daigram is suitable for representing programs, computers, Turing machines, etc. Diagrams are usually composed of boxes mutually connected by oriented lines. The boxes are of functional type (see Figure 1) when they represent elementary operations to be carried out on an unspecified object x of a set X, the former of which may be imagined concretely as the set of the digits contained in the memory of a computer, the tape configuration of a Turing machine, etc. There are other boxes of predicative type (see Figure 2) which do not operate on an object but decide on the next operation to be carried out, according to whether or not a certain property of $x \in X$ occurs. Examples of diagrams are: $\Sigma(\alpha, \beta, \gamma, a, b, c)$ [Figure 3] and $\Omega_5(\alpha, \beta, \gamma, \delta, \epsilon, a, b, c, d, e)$ [see Figure 4]. It is easy to see a difference between them. Inside the diagram Σ, some parts which may be considered as a diagram can be isolated in such a way that if $\Pi(a, b)$, $\Omega(\alpha, a)$, $\Delta(\alpha, a, b)$ denote, respectively, the diagrams of Figures 5–7, it is natural to write

$$\Sigma(\alpha, \beta, \gamma, a, b, c) = \Omega(\alpha, \Delta(\beta, \Omega(\gamma, a), \Pi(b, c))).$$

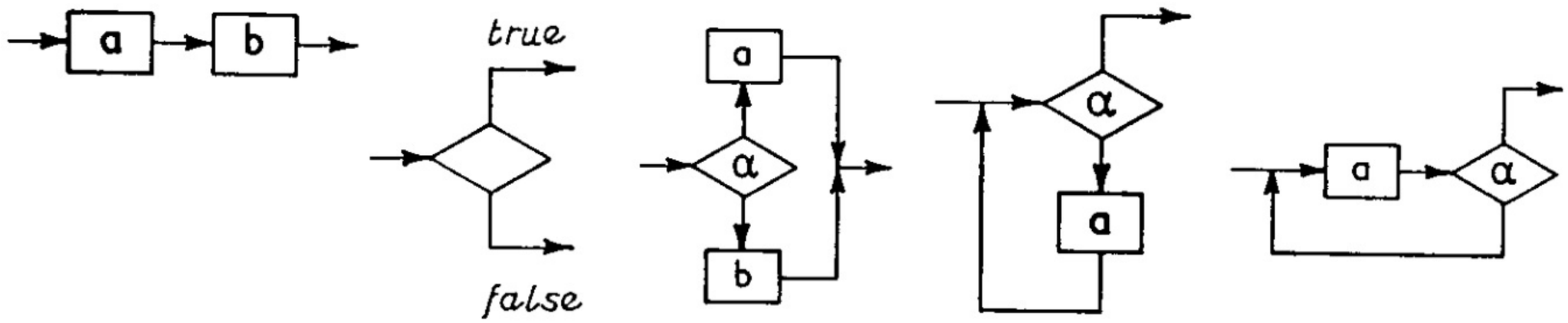
Nothing of this kind can be done for what concerns Ω_5; the same happens for the entire infinite class of similar diagrams

$$\Omega_1[=\Omega], \Omega_2, \Omega_3, \cdots, \Omega_n, \cdots,$$

whose formation rule can be easily imagined.

Let us say that while Σ is decomposable according to subdiagrams Π, Ω and Δ, the diagrams of the type Ω_n are not decomposable. From the last consideration, which should be obvious to anyone who tries to isolate with a

Böhm, G. & Jacopini, G. **Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules**. Communications of the ACM: volume 9, número 5, maio de 1966 (p. 366-371)



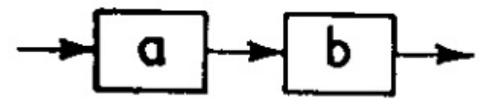
Mas afinal, o que é um algoritmo?

Operações seqüenciais:

O algoritmo tem início e fim, e as operações são realizadas através de sentenças (statements). Cada sentença determina uma ação específica a ser realizada.

Cada sentença deve corresponder a uma ação única, a uma ação simples, do algoritmo.

Uma sentença não pode, em geral, ter diversas ações, diversos comandos. Assim, uma setença deve ser atômica (ou primitiva), ou seja, não deve poder ser dividida em outras setenças menores individuais.



Mas afinal, o que é um algoritmo?

Operações condicionais (seleção, decisão):

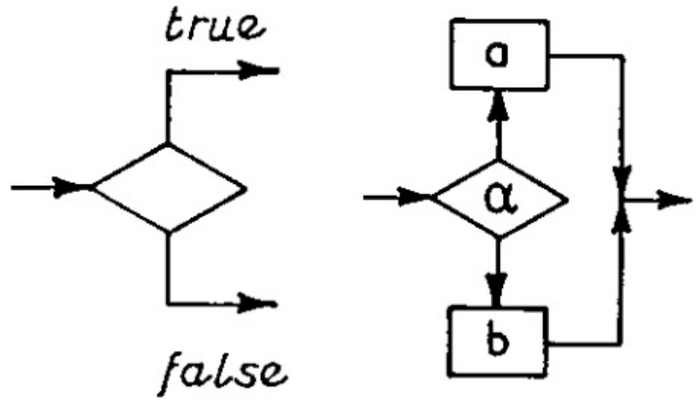
São estruturas que permitem escolher entre dois (ou mais) caminhos a serem seguidos, dependendo da avaliação de uma expressão booleana.

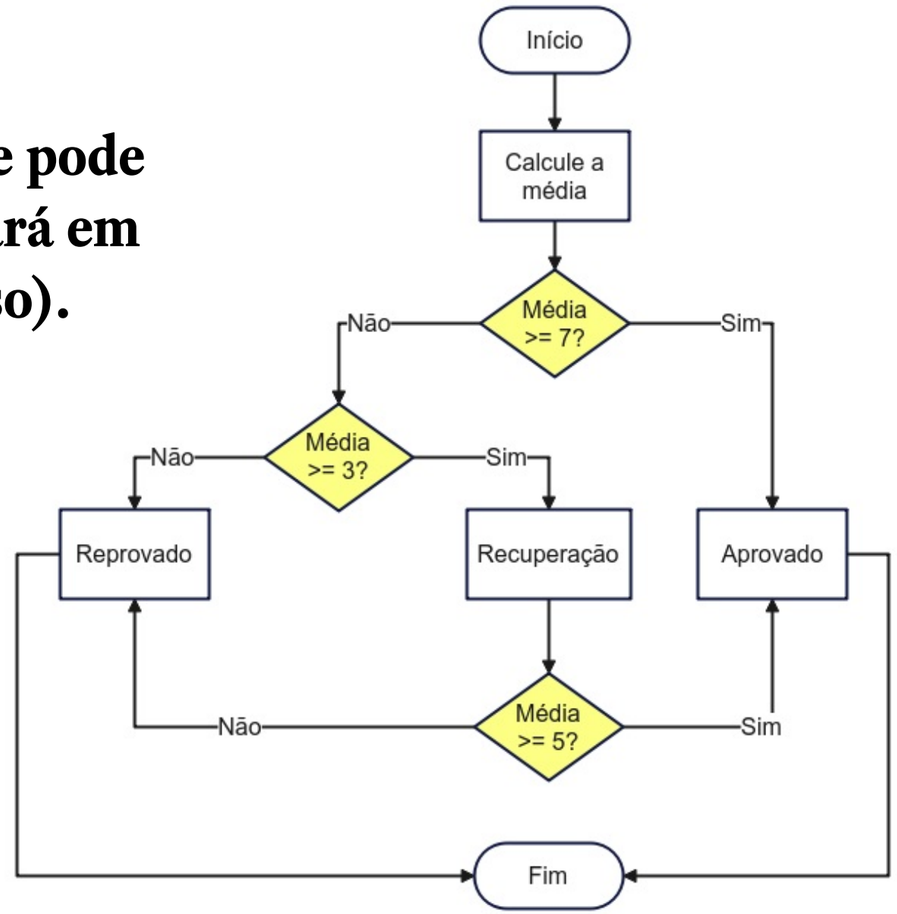
Uma expressão booleana é uma pergunta (que pode ser simples ou complexa), que sempre resultará em uma resposta SIM (verdadeiro) ou NÃO (falso).

O resultado de uma expressão booleana é:

- **sim, verdadeiro, true, 1**
- **não, falso, false, 0**

Existem diversas estruturas condicionais, que serão estudadas futuramente.



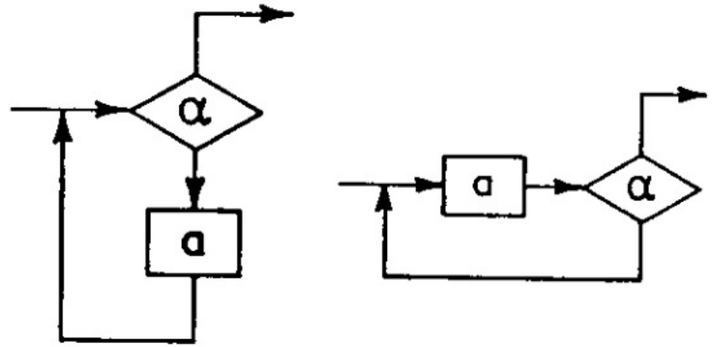


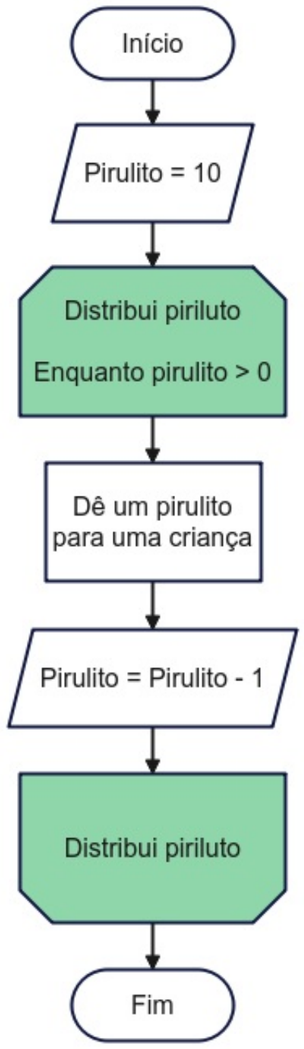
Mas afinal, o que é um algoritmo?

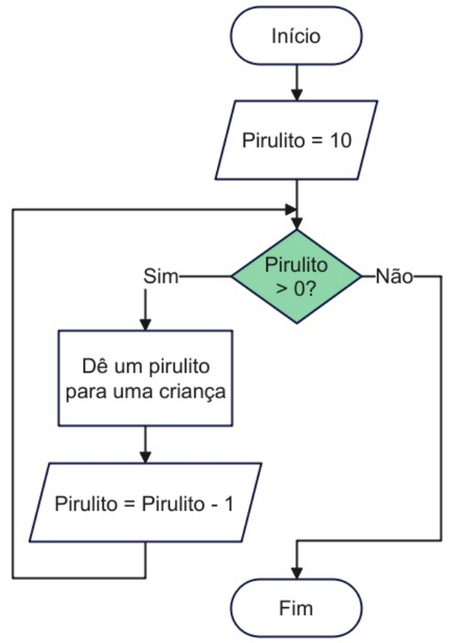
Operações de repetição (iteração ou recursão):

São instruções que nos permitem executar várias vezes a mesma sentença, ou o mesmo bloco (conjunto) de sentenças.

Existem diversas estruturas de repetição, que serão estudadas posteriormente.







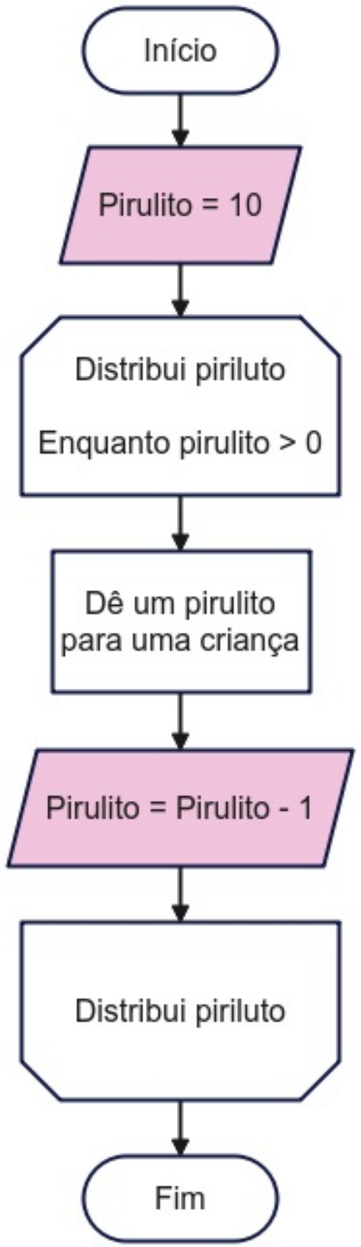
Mas afinal, o que é um algoritmo?

Estruturas de dados:

São estruturas que nos permitem armazenar um ou mais valores na memória do computador, para utilização em cálculos e outros tipos de processamento.

A estrutura de dado mais simples é uma variável, que armazena apenas um único valor.

Existem diversas estruturas de dados altamente complexas e especializadas que serão estudadas futuramente.



Parênteses: "algoritmos + estruturas de dados = programas"

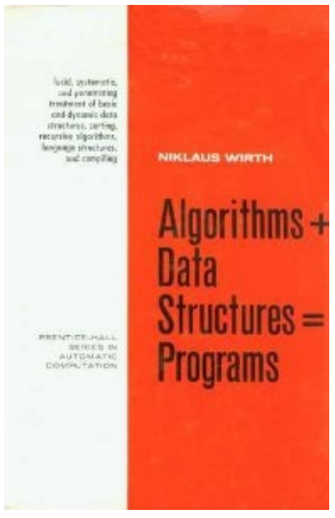
Adaptado de: Tyomitch, na Wikipedia
(https://en.wikipedia.org/wiki/File:Niklaus_Wirth,_UrGU.jpg)

Os algoritmos, em si mesmos, podem ser criados apenas com:

- **Operações seqüenciais**
- **Operações condicionais (seleção, decisão)**
- **Operações de repetição (iteração ou recursão)**

Entretanto, os algoritmos operam sobre dados que queremos manipular ou processar. Esses dados são armazenados em estruturas de dados.

Os conceitos de algoritmos e estruturas de dados estão tão intimamente relacionados que é praticamente impossível estudar um sem o outro. De fato, Niklaus Emil Wirth, em 1976, escreveu um livro importante sobre o assunto intitulado "Algorithms + Data Structures = Programs".



Ainda:

Euler (1965)	**PL360 (1966)**	**Algol W (1966)**
Pascal (1970)	**Modula (1975)**	**Modula-2 (1978)**
Oberon (1987)	**Oberon-2 (1991)**	**Oberon-07 (2007)**

Motivação: Um algoritmo do dia a dia: somar 2 números

- Adicione: 99,7 e 9,89

Motivação: Um algoritmo do dia a dia: somar 2 números

- Adicione: 99,7 e 9,89

Considere dois números positivos a e b com o formato "$p.f$", onde p é a quantidade de algarismos na parte inteira e f é a quantidade de algarismos na parte fracionária. Os algarismos individuais nos números serão indexados pela posição i, que começa em $p-1$ (algarismo mais significativo), passa por 0 (algarismo das unidades) e termina em $-f$ (algarismo menos significativo), de tal forma que:

$$a = a_{p-1}\, a_{p-2} \ldots a_1\, a_0 \,.\, a_{-1}\, a_{-2} \ldots a_{-f+1}\, a_{-f}$$
$$b = b_{p-1}\, b_{p-2} \ldots b_1\, b_0 \,.\, b_{-1}\, b_{-2} \ldots b_{-f+1}\, b_{-f}$$

Para encontrar a soma $c = a + b$, conforme abaixo, siga as instruções a seguir:

$$\begin{aligned} c_p\, c_{p-1}\, c_{p-2} \ldots c_1\, c_0 \,.\, c_{-1}\, c_{-2} \ldots c_{-f+1}\, c_{-f} = &(a_{p-1}\, a_{p-2} \ldots a_1\, a_0 \,.\, a_{-1}\, a_{-2} \ldots a_{-f+1}\, a_{-f}) \\ &+(b_{p-1}\, b_{p-2} \ldots b_1\, b_0 \,.\, b_{-1}\, b_{-2} \ldots b_{-f+1}\, b_{-f}) \end{aligned}$$

1. Verifique a quantidade p de algarismos na parte inteira dos dois números. Se a quantidade for diferente, adicione 0s à equerda no menor número até que p seja o mesmo em ambos os números.
2. Verifique a quantidade f de algarismos na parte fracionária dos dois números. Se a quantidade for diferente, adicione 0s à direita no número com a menor quantidade de algarismos fracionários até que f seja o mesmo em ambos os números.
3. Atribua o valor $-f$ à variável i.
4. Atribua o valor 0 à variavel v (que será o controle do "vai um").
5. Enquanto $i \leq p-1$, repita:
   (a) Adicione os algarismos a_i e b_i ao valor atual de v para obter c_i.
   (b) Se $c_i \geq 10$, então faça $c_i = c_i - 10$ e faça $v = 1$; caso contrário, faça $v = 0$.
   (c) Faça $i = i + 1$.
6. Faça $c_p = v$.
7. Imprima a resposta final, c, exibindo $c_p\, c_{p-1}\, c_{p-2} \ldots c_1\, c_0 \,.\, c_{-1}\, c_{-2} \ldots c_{-f+1}\, c_{-f}$, descartando eventuais zeros à esquerda na parte inteira e zeros à direita na parte fracionária.
8. Termine.

Motivação: Algoritmos: por que essa complicação toda?

Foto por Zachary Kadolph, no Unsplash (https://unsplash.com/photos/grayscale-photo-of-girl-in-polka-dot-long-sleeve-shirt-Hl_o1K6OPsA)

Motivação: Algoritmos: para automatização!

Foto por Kate Kozyrka, no Unsplash
(https://unsplash.com/photos/smiling-woman-B41fY4dhX18)

Se criarmos um algoritmo formal correto para resolver um problema, podemos automatizar sua execução!

- podemos construir uma máquina para executar o algoritmo e resolver vários problemas, rapidamente

- a máquina não precisa ter consciência do que está fazendo, basta seguir o algoritmo

- os resultados serão consistentes, reprodutíveis

- como o algoritmo é correto, resolve todos os problemas da mesma "classe"

- diminuir tarefas repetitivas

Algoritmos: é isso que o cientista da computação faz!

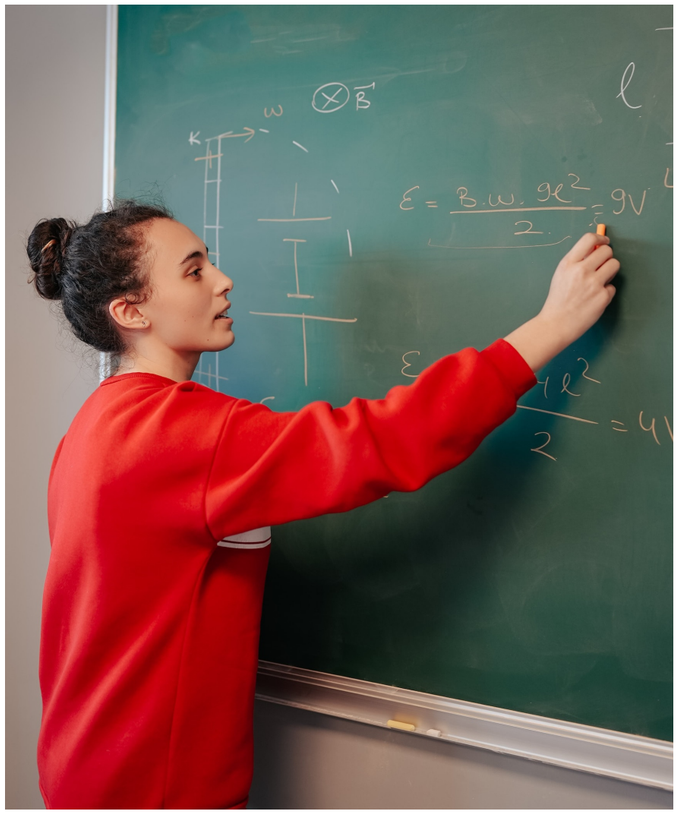


Foto: Yunus Tuğ, no Unsplash
(https://unsplash.com/photos/a-woman-writing-on-a-blackboard-with-chalk-DNhCPsVF8l4)

Trabalho do cientista da computação:

- **Pesquisar e projetar algoritmos corretos e eficientes para diversos problemas do dia a dia, estudando suas propriedades formais e matemáticas**

- **Projetar e criar linguagens de programação que possam expressar esses algoritmos**

- **Projetar e construir sistemas computacionais que possam executar os algoritmos de modo eficiente**

Algoritmos: Tudo é festa? Problemas e soluções algorítmicas

Arthur Chauvineau, no Unsplash (https://unsplash.com/photos/yellow-red-and-blue-fireworks-Dn7P1U26ZkE)

Com o avanço da computação, já temos algoritmos para muitos e muitos problemas. Cálculos, desenho, medicina, engenharia, inteligência artificial, artes, biologia, física, matemática...

Em uma avaliação simples, parece que todos os problemas existentes podem ser solucionáveis através de algoritmos.

Será?

Algoritmos: Kurt Gödel e os Teoremas da Incompletude

Em 1931, Kurt Friedrich Gödel, um matemático com 25 anos, publicou um artigo que foi um marco revolucionário na lógica e matemática.

Gödel provou que existem problemas para os quais nenhuma solução algorítmica pode existir.

Esses são problemas essencialmente insolúveis. Não importa o tempo e o esforço, nenhuma solução algorítmica jamais será encontrada.

"Sobre as Proposições Indecidíveis dos *Principia Mathematica* e Sistemas Correlatos"

Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I[1]).

Von **Kurt Gödel** in Wien.

1.

Die Entwicklung der Mathematik in der Richtung zu größerer Exaktheit hat bekanntlich dazu geführt, daß weite Gebiete von ihr formalisiert wurden, in der Art, daß das Beweisen nach einigen wenigen mechanischen Regeln vollzogen werden kann. Die umfassendsten derzeit aufgestellten formalen Systeme sind das System der Principia Mathematica (PM)[2]) einerseits, das Zermelo-Fraenkelsche (von J. v. Neumann weiter ausgebildete) Axiomensystem der Mengenlehre[3]) andererseits. Diese beiden Systeme sind so weit, daß alle heute in der Mathematik angewendeten Beweismethoden in ihnen formalisiert, d. h. auf einige wenige Axiome und Schlußregeln zurückgeführt sind. Es liegt daher die Vermutung nahe, daß diese Axiome und Schlußregeln dazu ausreichen, alle mathematischen Fragen, die sich in den betreffenden Systemen überhaupt formal ausdrücken lassen, auch zu entscheiden. Im folgenden wird gezeigt, daß dies nicht der Fall ist, sondern daß es in den beiden angeführten Systemen sogar relativ einfache Probleme aus der Theorie der gewöhnlichen ganzen Zahlen gibt[4]), die sich aus den Axiomen nicht

[1]) Vgl. die im Anzeiger der Akad. d. Wiss. in Wien (math.-naturw. Kl.) 1930, Nr. 19 erschienene Zusammenfassung der Resultate dieser Arbeit.

[2]) A. Whitehead und B. Russell, Principia Mathematica, 2. Aufl., Cambridge 1925. Zu den Axiomen des Systems PM rechnen wir insbesondere auch: Das Unendlichkeitsaxiom (in der Form: es gibt genau abzählbar viele Individuen), das Reduzibilitäts- und das Auswahlaxiom (für alle Typen).

[3]) Vgl. A. Fraenkel, Zehn Vorlesungen über die Grundlegung der Mengenlehre, Wissensch. u. Hyp. Bd. XXXI. J. v. Neumann, Die Axiomatisierung der Mengenlehre. Math. Zeitschr. 27, 1928. Journ. f. reine u. angew. Math. 154 (1925), 160 (1929). Wir bemerken, daß man zu den in der angeführten Literatur gegebenen mengentheoretischen Axiomen noch die Axiome und Schlußregeln des Logikkalküls hinzufügen muß, um die Formalisierung zu vollenden. — Die nachfolgenden Überlegungen gelten auch für die in den letzten Jahren von D. Hilbert und seinen Mitarbeitern aufgestellten formalen Systeme (soweit diese bisher vorliegen). Vgl. D. Hilbert, Math. Ann. 88, Abh. aus d. math. Sem. der Univ. Hamburg I (1922), VI (1928). P. Bernays, Math. Ann. 90. J. v. Neumann, Math. Zeitschr. 26 (1927). W. Ackermann, Math. Ann. 93.

[4]) D. h. genauer, es gibt unentscheidbare Sätze, in denen außer den logischen Konstanten: $\neg$ (nicht), $\vee$ (oder), (x) (für alle), $=$ (identisch mit) keine anderen Begriffe vorkommen als $+$ (Addition), . (Multiplikation), beide bezogen auf natürliche Zahlen, wobei auch die Präfixe (x) sich nur auf natürliche Zahlen beziehen dürfen.

Monatshefte für Mathematik und Physik, Leipzig, v. 38, n. 1, p. 173-198, 1931.

Autor desconhecido, na Wikimedia Commons (https://commons.wikimedia.org/wiki/File:1925_kurt_g%C3%B6del.png)

Algoritmos: alguns problemas podem ter, ou não, solução algorítmica

Existem problemas para os quais nós ainda não sabemos se existem ou se não existem soluções algorítmicas:

- **Ninguém nunca provou que uma solução existe**
- **Ninguém nunca provou que uma solução não existe**

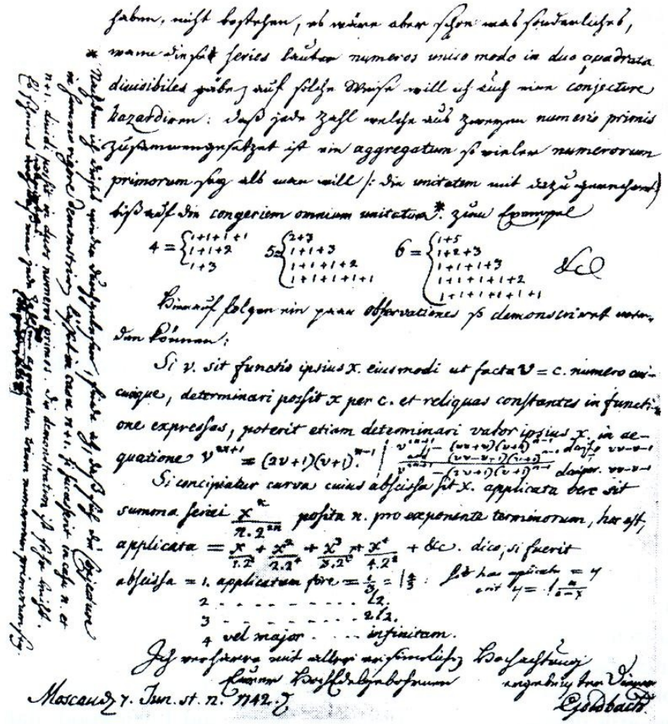
Ex.: Conjectura de Goldbach:

"Todo inteiro par positivo é a soma de dois números primos."

Em geral são problemas que exigem certo grau de inteligência, com raciocínio abstrato e compreensão sofisticada da linguagem natural, por exemplo.

- "Você está tratando esse papel ou a criança?"

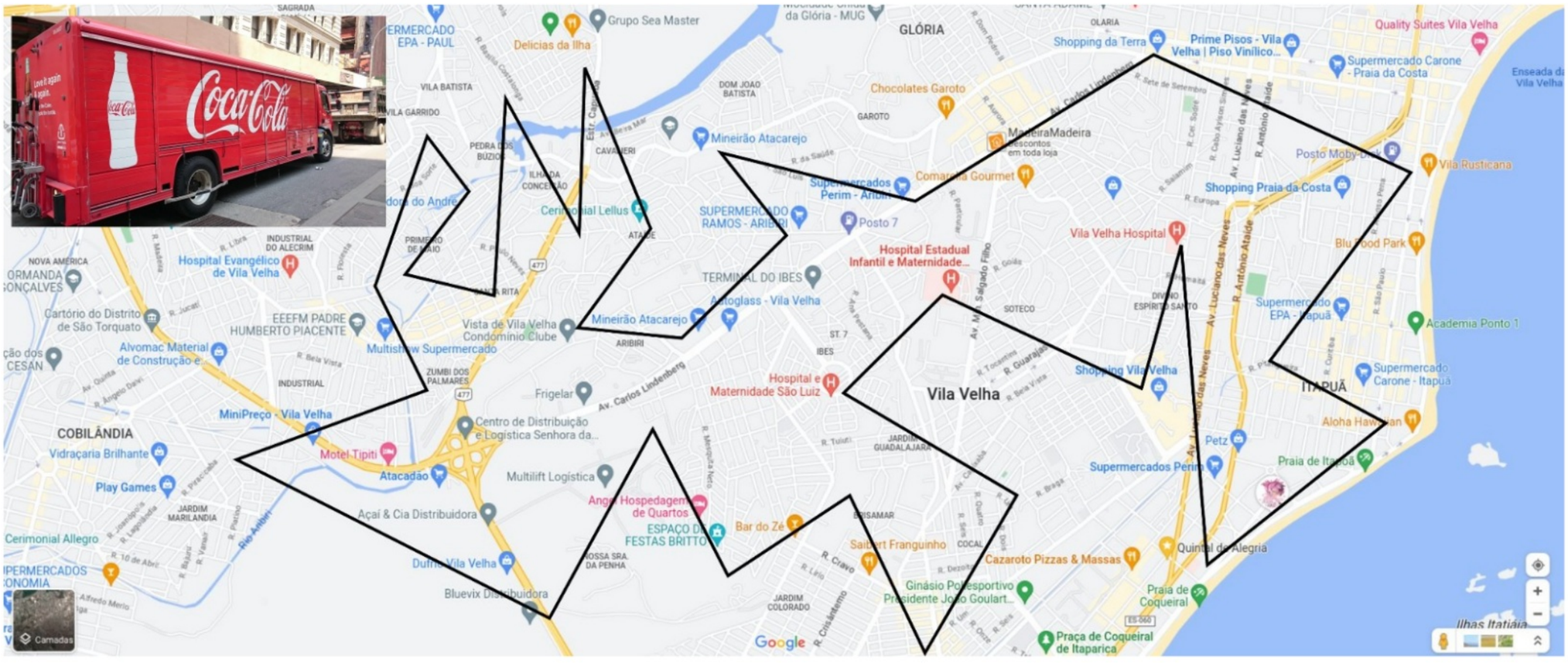
- "Esse jeitão é de apendicite"



Carta de Christian Goldbach para Leonhard Euler, 1742-06-07,
na Wikipedia (https://en.wikipedia.org/wiki/File:Letter_Goldbach-Euler.jpg)

Algoritmos: alguns problemas têm solução, mas ela é inútil para nós

Qual a melhor rota possível para o caminhão, a rota ótima, com 31 bares?



Algoritmos: alguns problemas têm solução, mas ela é inútil para nós

O Caixeiro Viajante

Bares	Intel Core i9-13900KS (6 GHz)	Supercomputador Frontier (1,1E18 FLOPS)
5	insignific.	insignific.
10	0,0005 seg	insignific.
15	3,4 min	insignific.
20	12 anos	2 seg
25	79 milhões de anos	5 meses
30	1,4 quatrilhões de anos	7,4 milhões de anos
35	53 sextilhões de anos	289 trilhões de anos

Algoritmos: alguns problemas têm solução, mas ela é inútil para nós

O que fazer se a solução for inútil?

- **Tentar achar outro algoritmo**
- **Usar uma heurística**

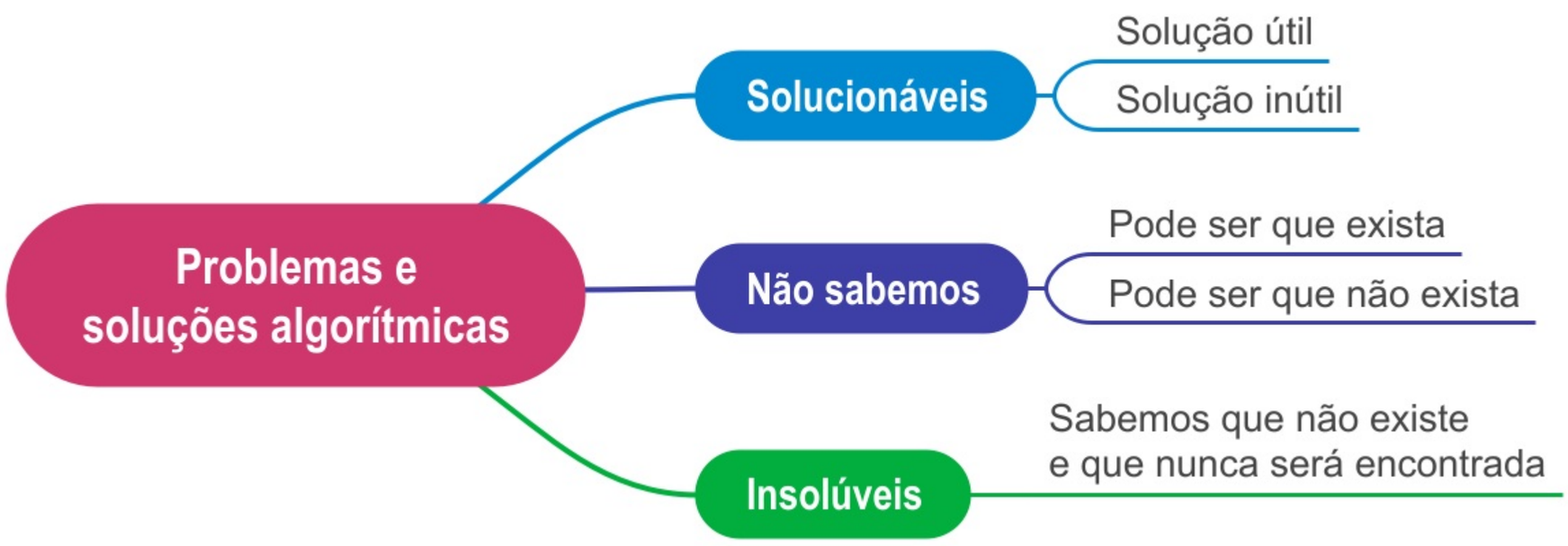
Algoritmo:

É a solução para uma classe de problemas, ótima, perfeita.

Heurística:

Uma resposta sem garantia de ser ótima, perfeita, mas que é boa o suficientemente para nossos propósitos.

Algoritmos: problemas e soluções



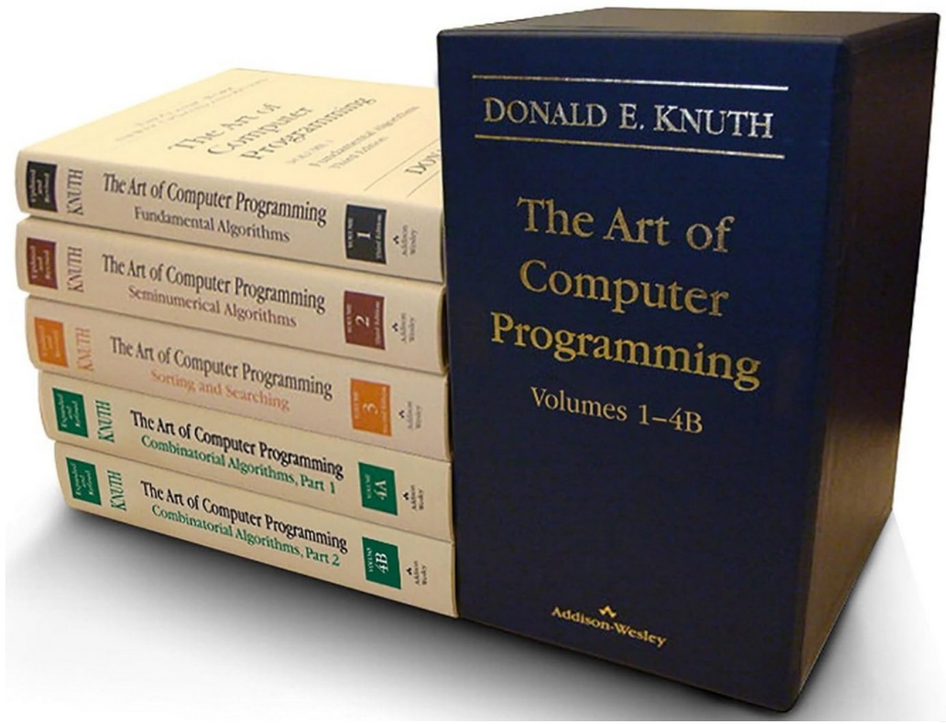
O cientista da computação tem que conhecer esses problemas para:

- **Não buscar a quadratura do círculo, o dobro do volume do cubo**
- **Não reinventar a roda**
- **Não perder tempo procurando uma solução ótima**
- **Criar algoritmos importantes para uma pessoa, organização, ou humanidade!**

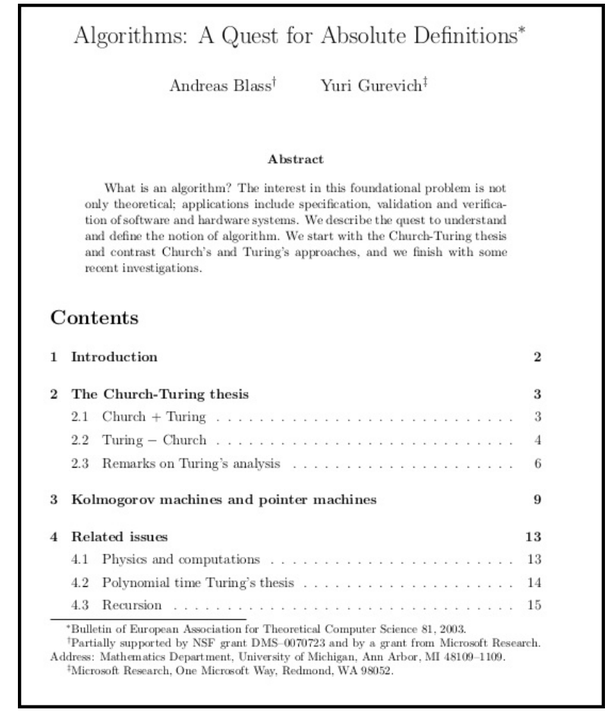
Algoritmo: definição formal

Nem Donald Ervin Knuth se atreveu! Então, aqui vai a minha...

Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.



amazon.com.br (https://www.amazon.com.br/dp/0137935102/)



Algorithms: A Quest for Absolute Definitions*

Andreas Blass† Yuri Gurevich‡

Abstract

What is an algorithm? The interest in this foundational problem is not only theoretical; applications include specification, validation and verification of software and hardware systems. We describe the quest to understand and define the notion of algorithm. We start with the Church-Turing thesis and contrast Church's and Turing's approaches, and we finish with some recent investigations.

Contents

*Bulletin of European Association for Theoretical Computer Science 81, 2003.
†Partially supported by NSF grant DMS–0070723 and by a grant from Microsoft Research.
Address: Mathematics Department, University of Michigan, Ann Arbor, MI 48109–1109.
‡Microsoft Research, One Microsoft Way, Redmond, WA 98052.

Blass & Gurevich, no site de Andreas Blass
(https://dept.math.lsa.umich.edu/~ablass/comp.html)

THE AUTHOR

Donald E. Knuth is Associate Professor of Mathematics at California Institute of Technology, where he received the Ph.D. degree in 1963. A specialist in the fields of computer science and combinatorial mathematics, Dr. Knuth is the author of numerous articles in the technical literature. His professional and honorary memberships include the Association for Computing Machinery, the Mathematical Association of America, the American Mathematical Society, Tau Beta Pi, Pi Delta Epsilon, and Sigma Xi.

Knuth, em 1968, lançamento da 1ª edição do Volume 1 (https://cs.stanford.edu/~knuth/taocp.html)

Algoritmo: definição formal

"coleção bem ordenada"

> Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.

Um algoritmo deve ter uma coleção de operações, ações, em uma ordem correta, ou seja, sabemos qual a ordem exata que as operações devem ser executadas (a ordem correta das sentenças).

Ordem incorreta:

1. Coloque o vinho no copo
2. Abra a garrafa (retire a rolha)
3. Feche a garrafa (coloque a rolha)
4. Beba
5. Peque a garrafa
6. Repita

Ordem correta:

1. Pegue a garrafa
2. Abra a garrafa (retire a rolha)
3. Coloque o vinho no copo
4. Beba
5. Repita
6. Feche a garrafa (coloque a rolha)

Algoritmo: definição formal

"operações efetivamente computáveis"

> **Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.**

As operações (ações que serão executadas pelo algoritmo) devem ser realizáveis (computáveis), ou seja, cada operação deve poder sem concluída com sucesso.

Operações não computáveis:

1. Liste todos os números primos
2. Calcule o valor exato de pi
3. Raiz quadrada de -39 em R
4. Valor de x em: "0x = 4"

Operações computáveis:

1. Liste os números primos < 100
2. Cacule o valor de pi até a 100ª casa
3. Raiz quadrada de 39 em R
4. Valor de x em: "4x = 0"

Algoritmo: definição formal

> **Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.**

"definidas e não ambíguas"

A definição das operações deve ser absolutamente precisa e detalhada, sem ambigüidade ou margem para interpretações duvidosas, e totalmente sem necessidade de simplificações ou explicações adicionais.

Operações definidas e não ambíguas são chamadas de operações primitivas.

Um algoritmo deve ser composto inteiramente de operações primitivas.

Oper. mal definidas/ambíguas:

1. Ache a raiz de 33
2. Repita
3. Abra a garrafa

Oper. bem definidas e não ambíguas:

1. Chame a função `sqrt` e passe como argumento o valor 33
2. Repita as 4 operações a seguir enquanto x > 0
3. Adicione x e y

Algoritmo: definição formal

> **Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.**

"sobre uma entrada"

Um algoritmo recebe zero ou mais entradas, quantidades que são dadas para ele antes que a execução se inicie ou depois, enquanto o algoritmo está sendo executado.

Exemplos de entradas:

1. **Ordenar o conjunto de números: {39, 12, 88, 0, 1, 54}**
2. **Ordenar o conjunto de números: { }**
3. **Verificar se "Maria" está na seguinte lista de nomes:**
   {Maria, Abrantes, Flávia, Tomás, Elis, Amora}
4. **Verificar se "Maria" está na seguinte lista de nomes: { }**
5. **Produzir um número aleatório, no Linux, com o comando:**
   `echo $((RANDOM))`

Algoritmo: definição formal

> **Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.**

"**produz uma saída**"

Um algoritmo produz uma ou mais saídas, quantidades que são relacionadas de algum modo com as entradas.

Exemplos de saídas:

1. Números ordenados: {0, 1, 12, 39, 54, 88}
2. Números ordenados: { }
3. Maria está na lista
4. Maria não está na lista
5. 26061
6. Erro: divisão por zero

Algoritmo: definição formal

"termina"

> **Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.**

Todo algoritmo deve terminar, ter um ponto de parada. Afinal: para que você iria utilizar um algoritmo que não termina nunca?

O término de um algoritmo tem relação com:

- **a quantidade de passos (operações) que ele executa**
- **a quantidade de tempo que ele demora para executar**

Algoritmo: definição formal

"quantidade finita de passos"

Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.

Todo algoritmo deve terminar após um número finito de passos (operações). Mais ainda: um algoritmo útil precisa não apenas de um númeor finito de passos, precisa de um número "muito finito" de passos, um número razoável de passos.

```c
int n = get_int("Insira um número inteiro: ");
if (n == 1)
    printf("ímpar\n");
else if (n == 2)
    printf("par\n");
else if (n == 3)
    printf("ímpar\n");
else if (n == 4)
    printf("par\n");
else if (n == 5)
    printf("ímpar\n");
else if (n == 6)
    printf("par\n");
else if (n == 7)
    printf("ímpar\n");
else if (n == 8)
    printf("par\n");
...
```

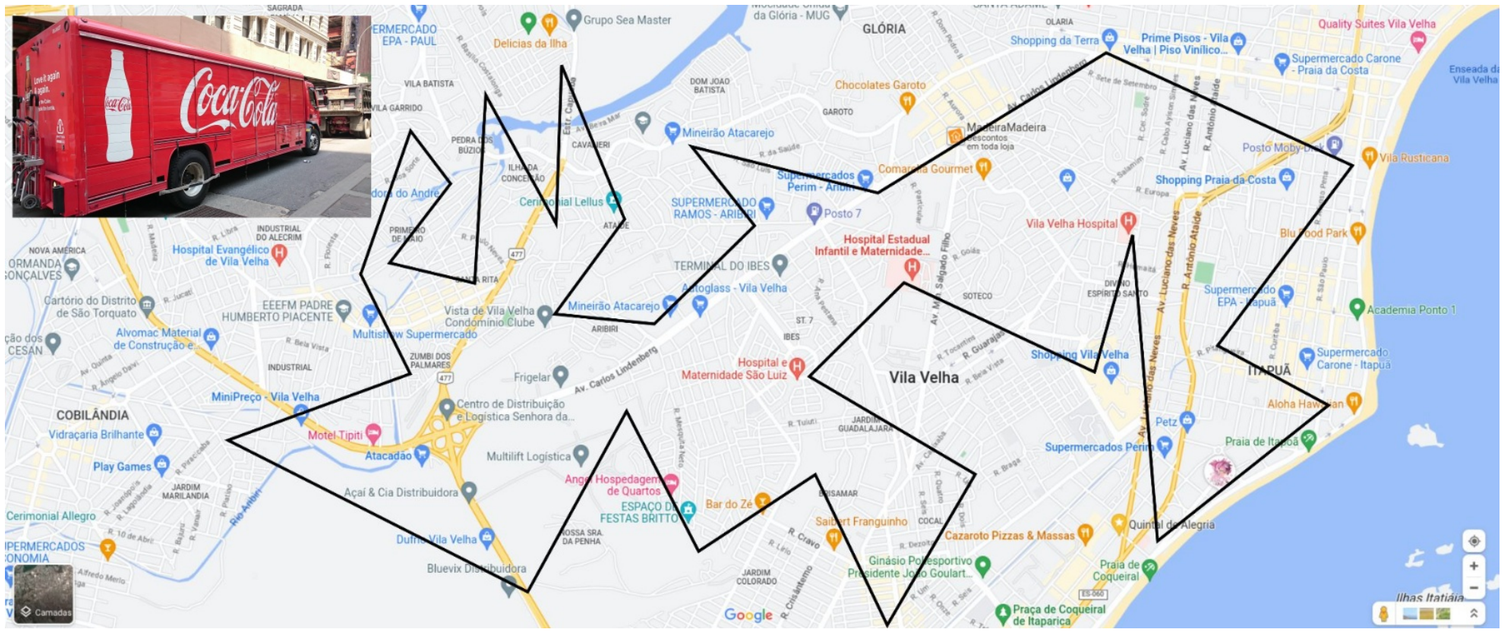
```c
int n = get_int("Insira um número inteiro: ");
if (n % 2 == 0)
    printf("par\n");
else
    printf("ímpar\n");
```

Algoritmo: definição formal

"quantidade finita de tempo"

Uma coleção bem ordenada de operações efetivamente computáveis, definidas e não ambíguas que, quando executada sobre uma entrada produz uma saída e termina em uma quantidade finita de passos e de tempo.

Todo algoritmo deve terminar após uma quantidade finita de tempo, uma quantidade "muito finita" de tempo, uma espera razoável de tempo.



Algoritmo: outras propriedades

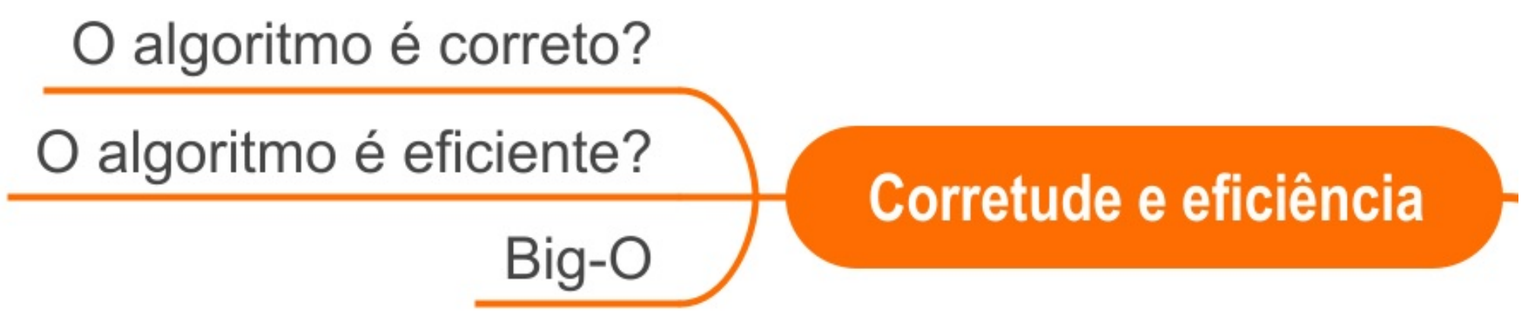
Propriedades mais "concretas":

- **Correto**
- **Bem projetado (design)**
- **Estilo correto**
- **Eficiente (tempo e espaço)**

Propriedades mais "abstratas":

- **Elegante**
- **Simples**
- **Bom em um sentido estético**

Algoritmos


O algoritmo é correto?
O algoritmo é eficiente?
Big-O
Corretude e eficiência

Algoritmos diferentes, mesma solução

Kristaps Ungurs, no Unsplash
(https://unsplash.com/photos/an-aerial-view-of-a-road-intersection-in-the-middle-of-a-forest-RuA4wAGmTl4)

Para muitos problemas existem vários algoritmos diferentes capazes de encontrar a resposta correta.

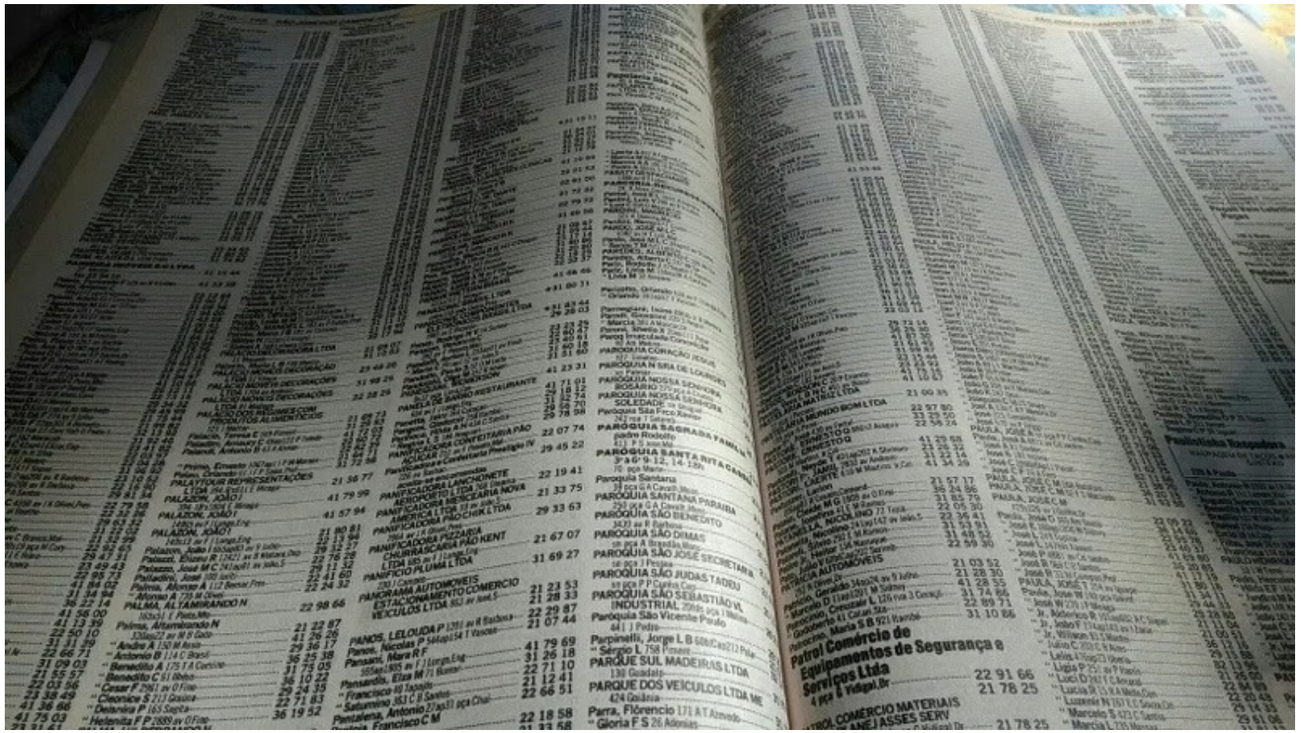
Quando temos vários algoritmos diferentes para o mesmo problema, temos que decidir qual o melhor ou, pelo menos, qual o melhor para cada situação.

Essa decisão avalia, principalmente, 2 coisas:

- **corretude**
- **eficiência**

Algoritmos diferentes, mesma solução: os algoritmos são corretos?

Problema: encontrar o telefone de uma pessoa em uma lista telefônica.

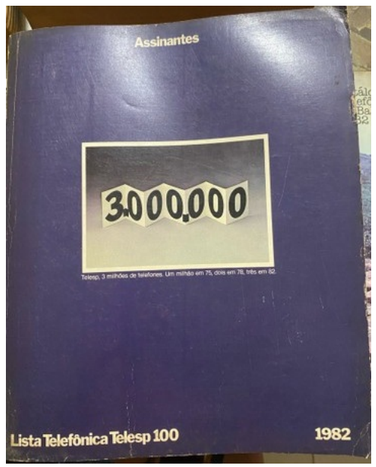


User7778, na Wikipedia (https://pt.wikipedia.org/wiki/Ficheiro:P%C3%A1gina_Lista_Telef%C3%B4nica_Telesp_138.png)

1º algoritmo:
Comece a pesquisar a partir da 1ª página, lendo todos os nomes, um por um, até encontrar.

2º algoritmo:
Comece a pesquisar a partir da 1ª página, lendo todos os nome, um por um, até encontrar, mas passe duas páginas de cada vez, voltando duas páginas caso ultrapasse o nome.

3º algoritmo:
Abra a lista no meio e verifique nas duas páginas se o nome está lá. Se não estiver, descarte a metada da lista onde o nome não está. Repita isso até encontrar.



User7778, na Wikimedia (https://commons.wikimedia.org/wiki/File:Capa_Lista_Telef%C3%B4nica_Telesp_100_-_1982.png)

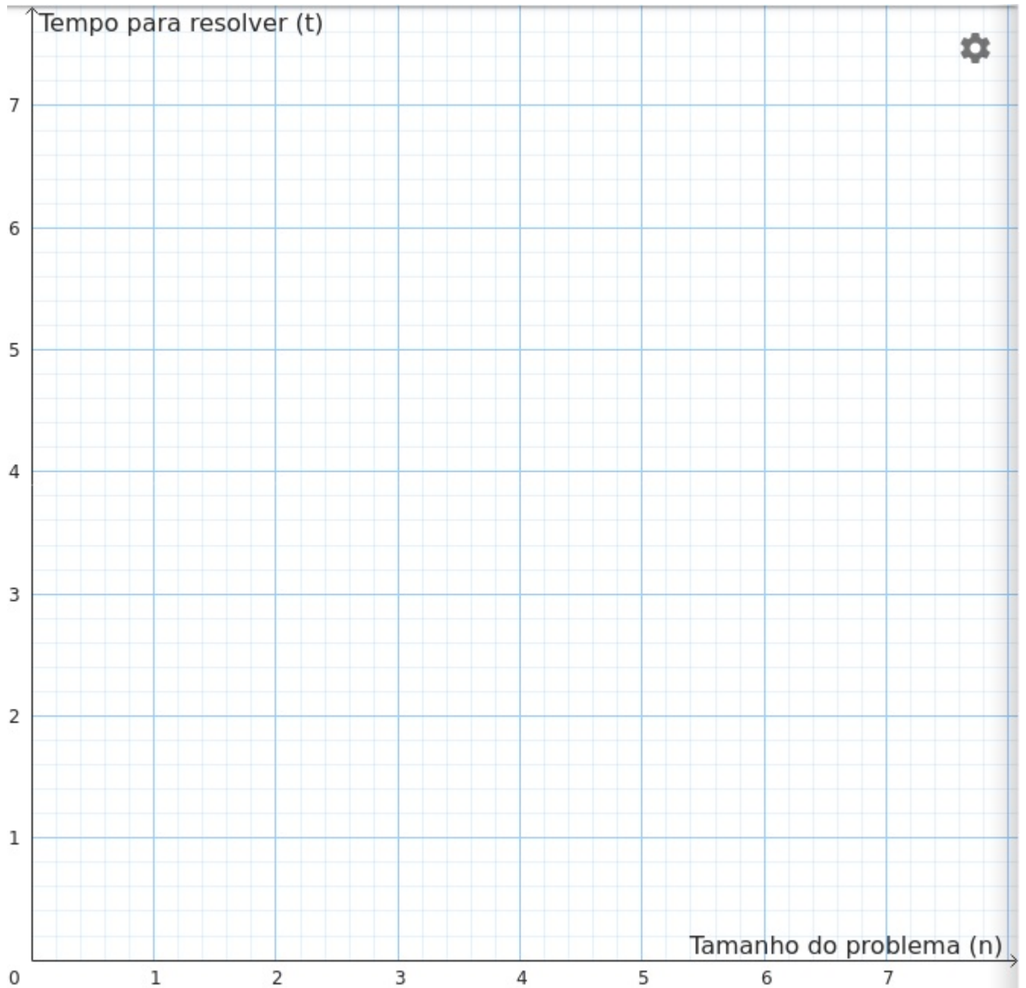
Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

Em geral a eficiência é medida em termos de recursos que o algoritmo consome para ser executado, em geral:

- **tempo de execução**
- **memória**

O tempo de execução é medido em número de operações que o algoritmo realiza, não exatamente o tempo de relógio.

Comparamos o tamanho do problema com o tempo de resolução, e estamos interessados em problemas de grande tamanho!



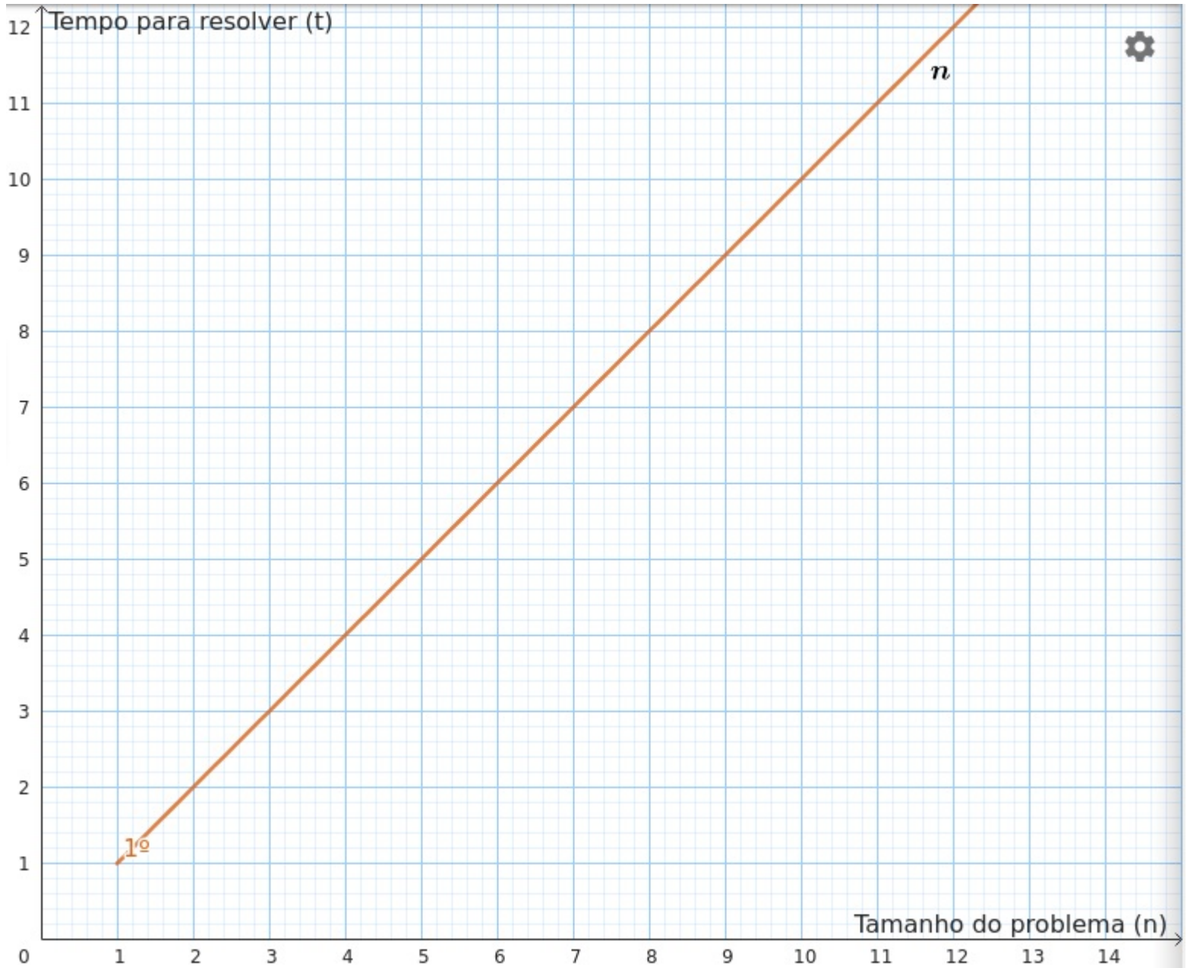
Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

O 1º algoritmo verifica todos os nomes, então para n nomes, executaremos n operações.

Este algoritmo executa em "tempo linear".

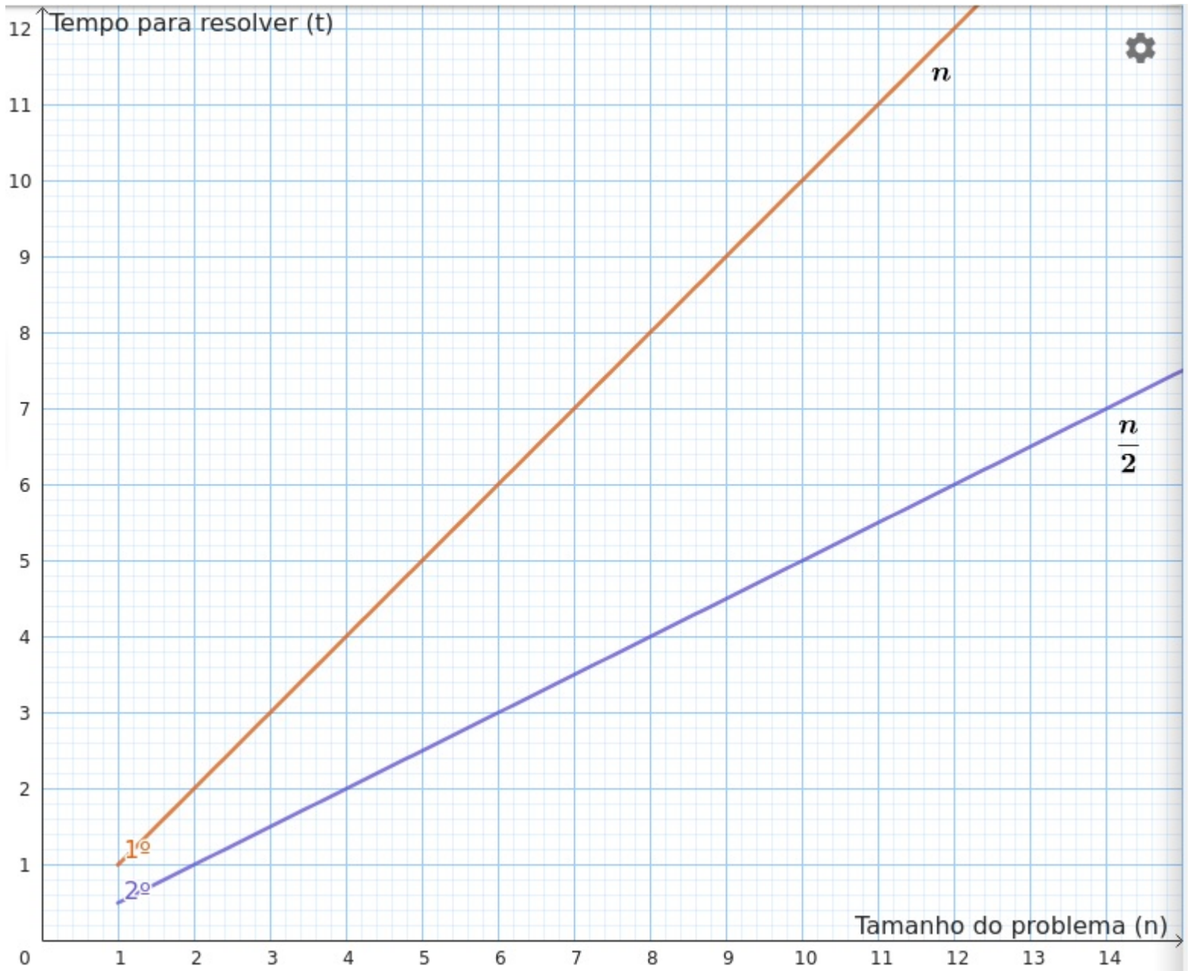
Os cientistas da computação utilziam uma notação especial para indicar essa "velocidade" de execução, a "Big-O":

O(n)



Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

O 2º algoritmo verifica aprox. a metade dos nomes, então para n nomes, executaremos cerca de n/2 operações.



Este algoritmo também executa em "tempo linear", mas é mais eficiente do que o 1º algoritmo! A notação Big-O para ele também será:

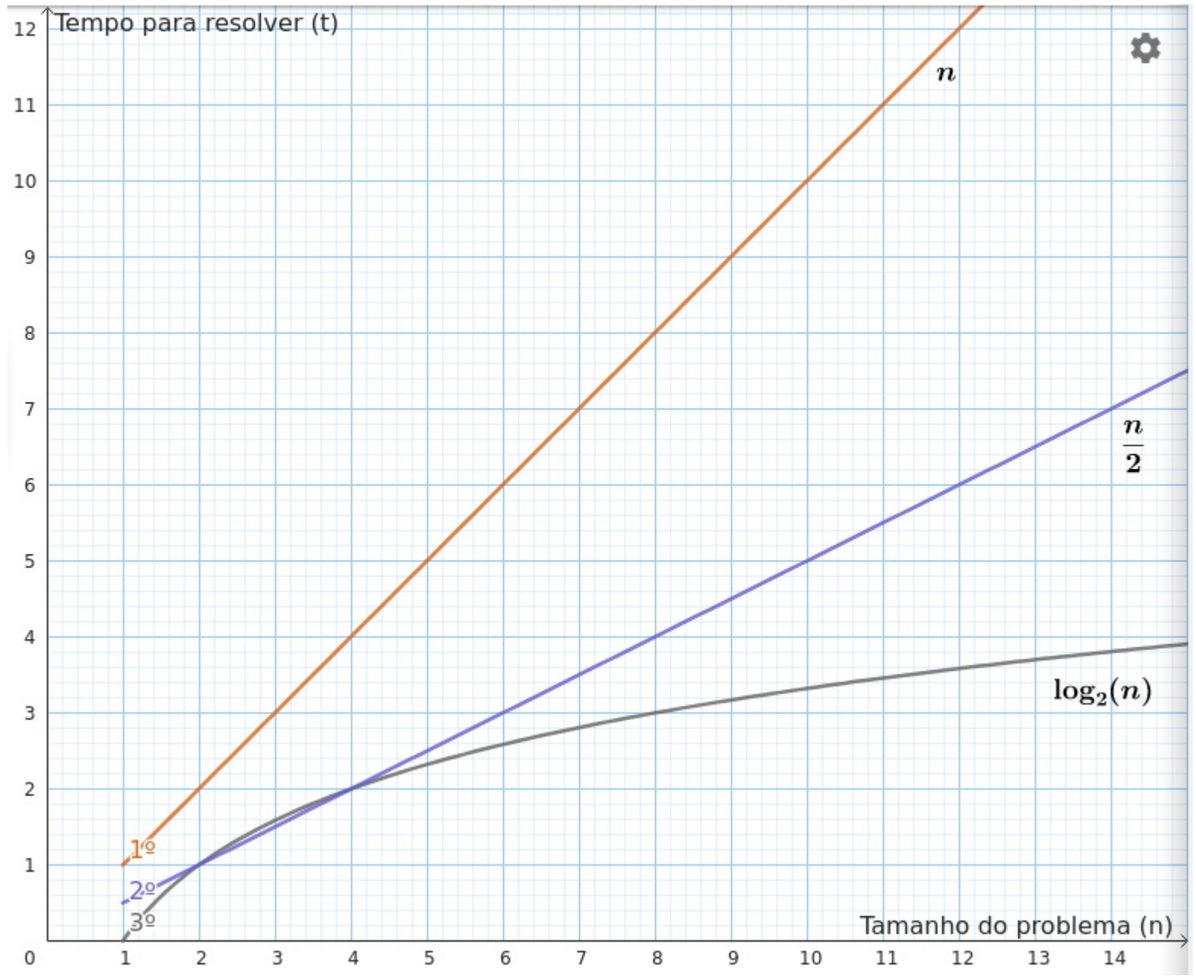
O(n)

Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

O 3º algoritmo é muito diferente. A cada operação executada, ele descarta a metade dos nomes, ou seja, executaremos $\log_2(n)$ operações.

Este algoritmo executa em "tempo logarítmico", e é o mais eficiente dos três. A notação Big-O para ele é:

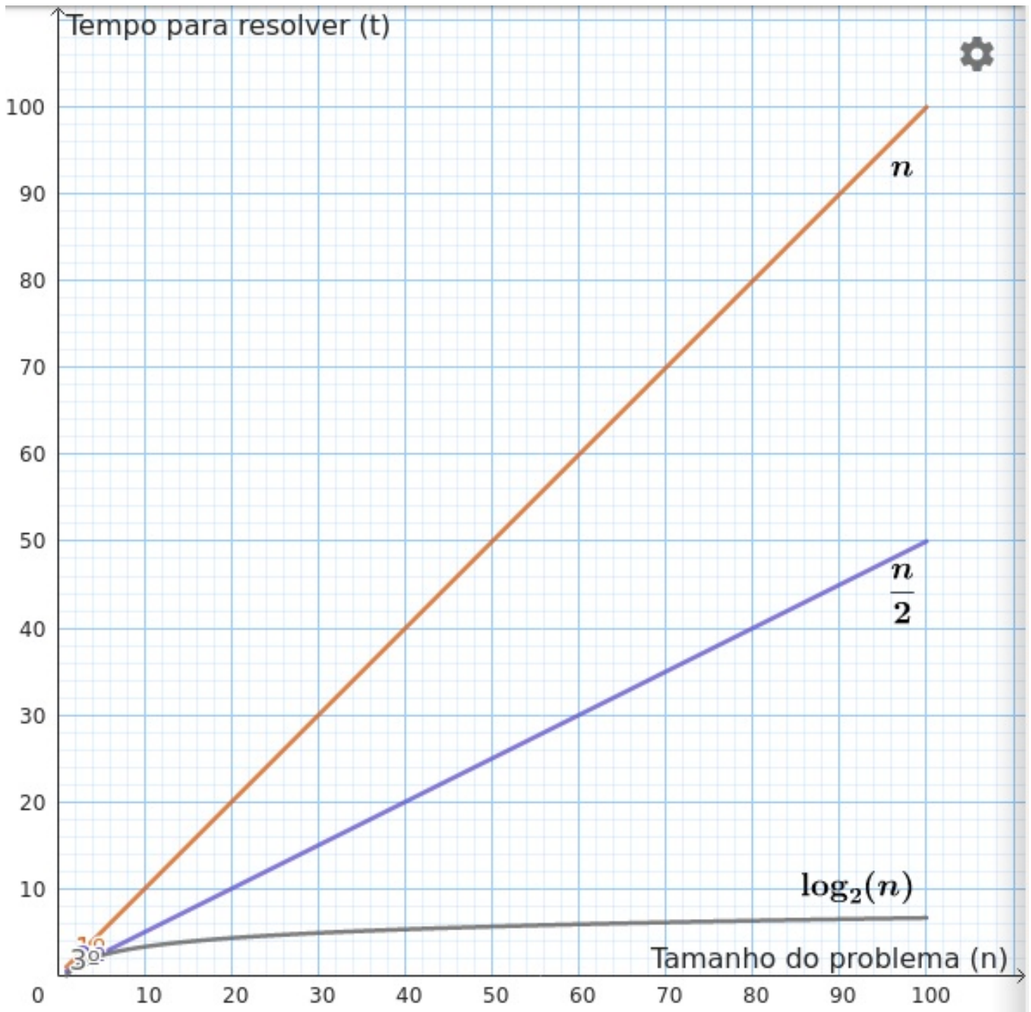
$O(\log_2 n)$



Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

Dependendo da quantidade de nomes a serem verificados, a diferença entre os três algoritmos será grande!

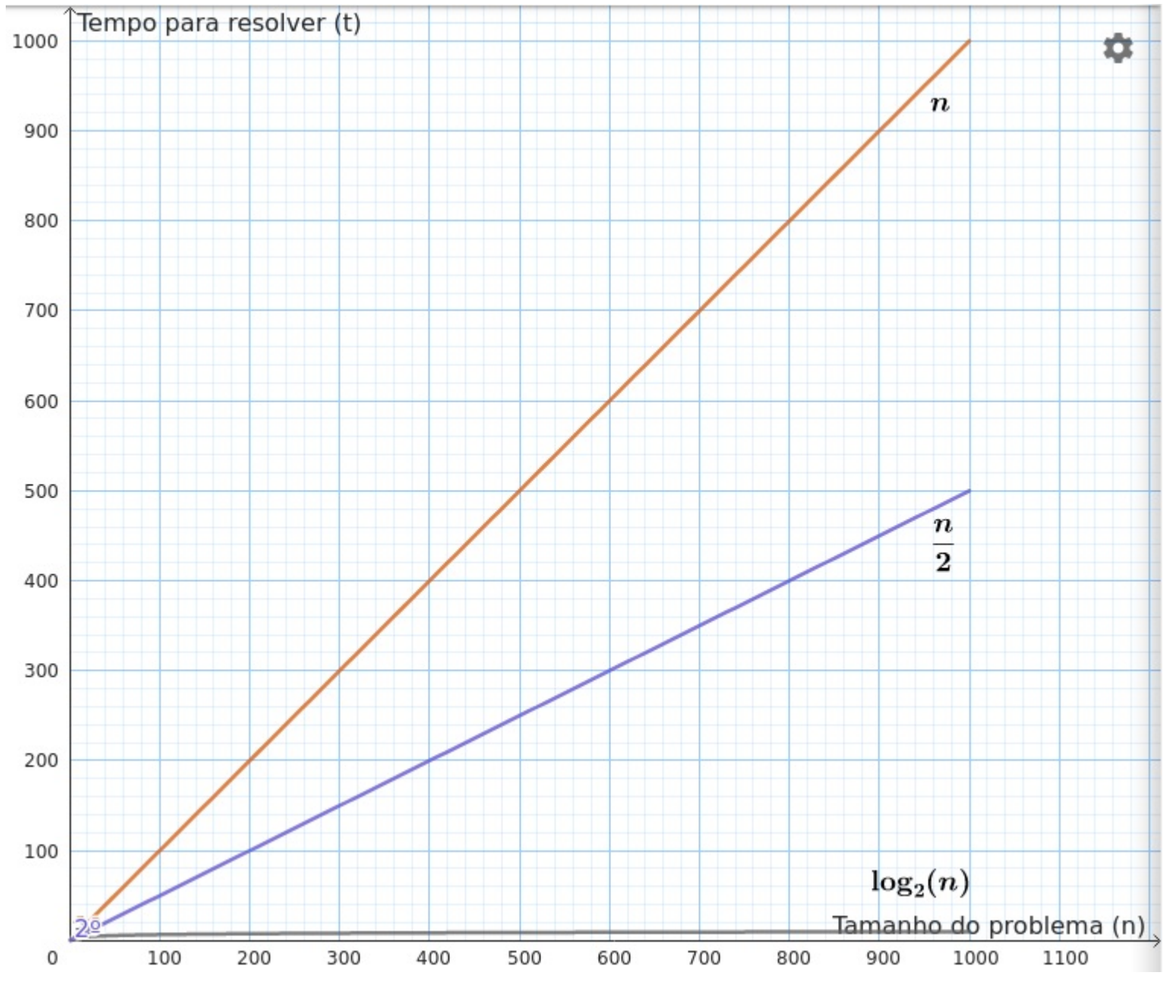
Com n = 100 temos:



Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

Dependendo da quantidade de nomes a serem verificados, a diferença entre os três algoritmos será grande!

Com n = 1000 temos:



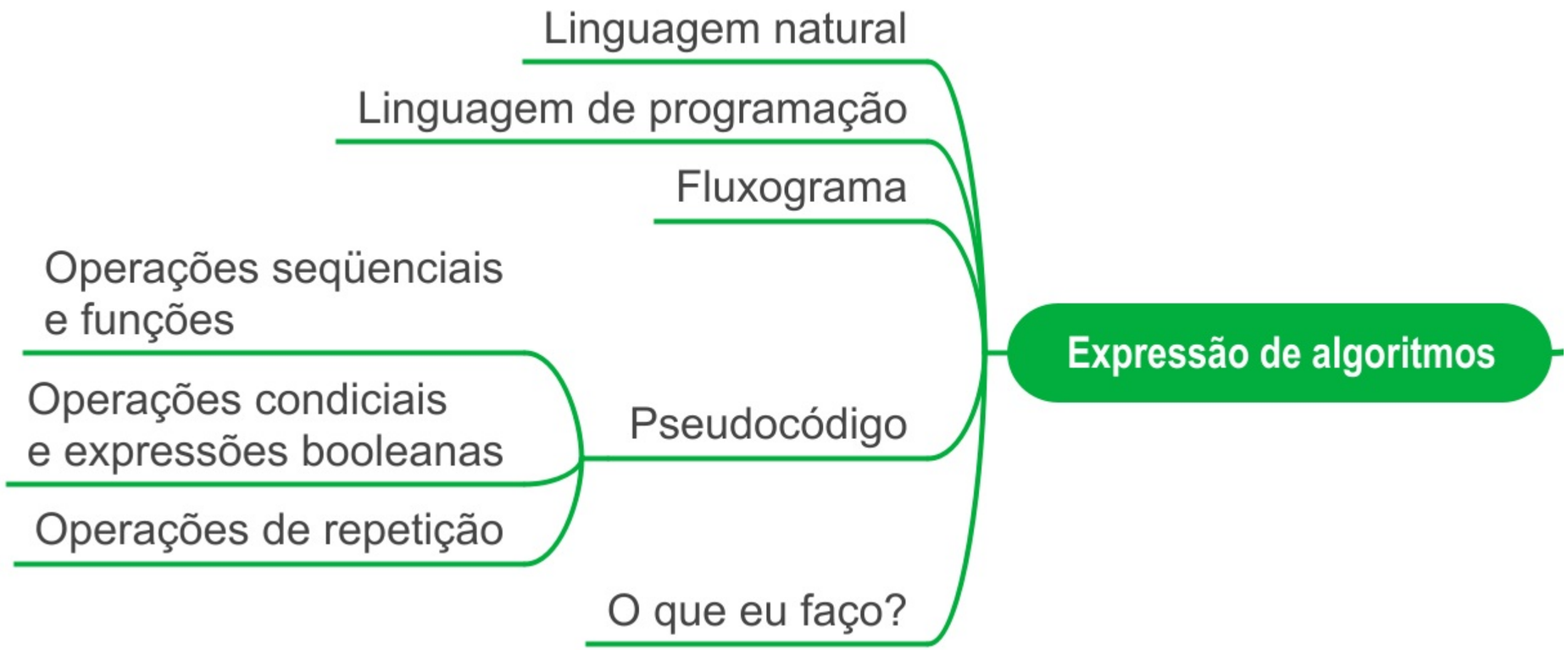
Algoritmos diferentes, mesma solução: os algoritmos são eficientes?

Diferentes algoritmos, mesmo que corretos, podem ter eficiência muito diferente!

	Tempo de execução (em operações)		
Nomes	**1º alg: n**	**2º alg: n/2**	**3º alg: $\log_2 n$**
512	512	256	9
1.024	1.024	512	10
2.048	2.048	1.024	11
4.096	4.096	2.048	12
8.192	8.192	4.096	13
16.384	16.384	8.192	14
32.768	32.768	16.384	15
65.536	65.536	32.768	16
131.072	131.072	65.536	17
262.144	262.144	131.072	18
524.288	524.288	262.144	19
1.048.576	1.048.576	524.288	20
2.097.152	2.097.152	1.048.576	21
4.194.304	4.194.304	2.097.152	22
8.388.608	8.388.608	4.194.304	23
16.777.216	16.777.216	8.388.608	24

$O(1)$ $O(\log_2 n)$ $O(n)$ $O(n \log_2 n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$ $O(n!)$

Algoritmos


Linguagem natural
Linguagem de programação
Fluxograma
Operações seqüenciais
e funções
Operações condiciais
e expressões booleanas
Operações de repetição
Pseudocódigo
Expressão de algoritmos
O que eu faço?

Como expressar um algoritmo?

- Linguagem natural

- Linguagem de programação

- Fluxograma

- Pseudocódigo

Como expressar um algoritmo?

Linguagem natural:
- **longa**
- **ambígua**
- **difícil de seguir**
- **depende muito de interpretação, contexto e experiência do leitor**

"Pegue uma lista telefônica, abra no meio e procure o nome. Se o nome estiver aí, ligue para a pessoa e termine. Caso contrário descarte a metade esquerda ou a metade direita da lista, onde o nome não estará por causa da ordem alfabética. Abra no meio e procure novamente o nome. Se o nome estiver aí, ligue para a pessoa. Caso contrário repita o procedimento de descarte e busca."

Como expressar um algoritmo?

Linguagem de programação:

- **pouco abstrata para uma primeira abordagem**
- **muitos detalhes específicos da linguagem**
- **preocupação com sintaxe**
- **detalhes técnicos atrapalham o raciocínio**

```c
#include <cs50.h>
#include <stdio.h>

#define TAMANHO 10

int main(void)
{
    int x[TAMANHO] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int n = get_int("Número a ser procurado: ");
    int i = 0;
    int f = TAMANHO - 1;
    int m;
    int a = 0;

    while (i <= f)
    {
        m = (i + f) / 2;
        if (n == x[m])
        {
            printf("O número está na lista.\n");
            a = 1;
            break;
        }
        else if (n > x[m])
            i = m + 1;
        else
            f = m - 1;
    }
    if (a == 0)
        printf("O número não está na lista.\n");
}
```

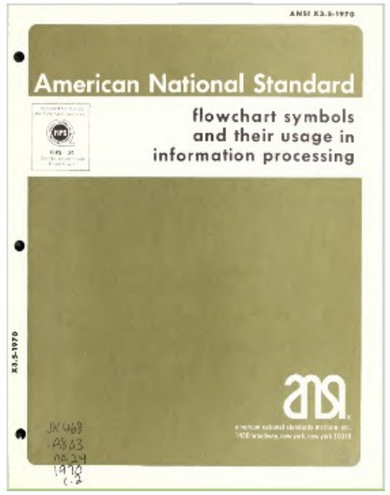
Como expressar um algoritmo?

Fluxograma:

- **boa indicação visual**
- **raciocínio abstrato**
- **pode ficar muito grande**


International Standard ISO 5807

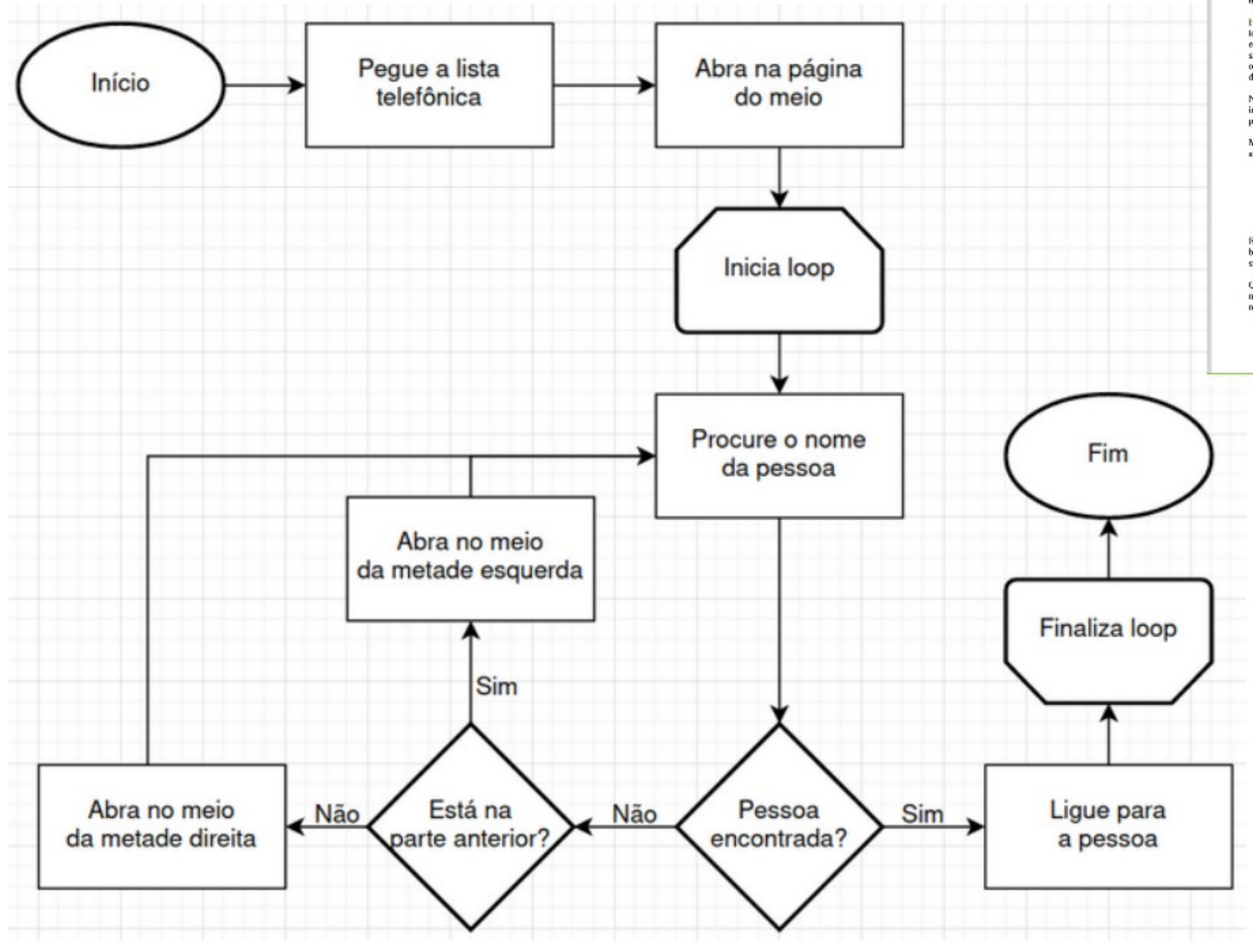
Information processing — Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts

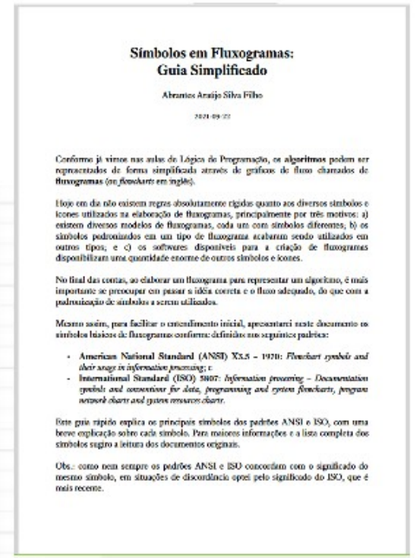

ANSI X3.5-1970

American National Standard

flowchart symbols and their usage in information processing

X3.5-1970




Símbolos em Fluxogramas: Guia Simplificado

Abramo Araújo Silva Filho

2021-09-22

Conforme já vimos nas aulas de Lógica de Programação, os **algoritmos** podem ser representados de forma simplificada através de gráficos de fluxo chamados de **fluxogramas** (*no flowcharts* em inglês).

Hoje em dia não existem regras absolutamente rígidas quanto aos diversos símbolos e ícones utilizados na elaboração de fluxogramas, principalmente por três motivos: a) existem diversos modelos de fluxogramas, cada um com símbolos diferentes; b) os símbolos padronizados em um tipo de fluxograma acabarem sendo utilizados em outros tipos; e c) os softwares disponíveis para a criação de fluxogramas disponibilizam uma quantidade enorme de outros símbolos e ícones.

No final das contas, ao elaborar um fluxograma para representar um algoritmo, é mais importante se preocupar em passar a idéia correta e o fluxo adequado, do que com a padronização de símbolos a serem utilizados.

Mesmo assim, para facilitar o entendimento inicial, apresentarei neste documento os símbolos básicos de fluxogramas conforme definidos nos seguintes padrões:

- **American National Standard (ANSI) X3.5 – 1970:** *Flowchart symbols and their usage in information processing*; e
- **International Standard (ISO) 5807:** *Information processing – Documentation symbols and conventions for data, programming and system flowcharts, program network charts and system resources charts*.

Este guia rápido explica os principais símbolos dos padrões ANSI e ISO, com uma breve explicação sobre cada símbolo. Para maiores informações e a lista completa dos símbolos sugiro a leitura dos documentos originais.

Obs.: como nem sempre os padrões ANSI e ISO concordam com o significado do mesmo símbolo, em situações de discordância optei pelo significado do ISO, que é mais recente.

Como expressar um algoritmo?

Pseudocódigo:

- escrever em linguagem natural, mas como se fosse de programação: com sentenças
- simples e de fácil leitura
- não tem regras gramaticais, permite pensamento abstrato e lógico
- sentenças tem estrutura bem definida e é fácil visualizar a organização do algoritmo
- relativamente fácil de transformar em linguagem de programação

```
 1   Pegue a lista telefônica
 2   Abra na página do meio
 3   Procure o nome na página
 4   Se a pessoa procurada está na página
 5      Ligue para a pessoa e vá para a linha 13
 6   Ou se a pessoa está na parte anterior da lista:
 7      Abra na página do meio da metade esquerda da lista
 8      Volte para a linha 3
 9   Ou se a pessoa está na parte posterior da lista:
10      Abra na página do meio da metade direita da lista
11      Volte para a linha 3
12   Caso contrário:
13      Saia
```

Como expressar um algoritmo?

Pseudocódigo:

- os verbos das sentenças indicam ações ou funções

```
1    Pegue a lista telefônica
2    Abra na página do meio
3    Procure o nome na página
4    Se a pessoa procurada está na página
5       Ligue para a pessoa e vá para a linha 13
6    Ou se a pessoa está na parte anterior da lista:
7       Abra na página do meio da metade esquerda da lista
8       Volte para a linha 3
9    Ou se a pessoa está na parte posterior da lista:
10      Abra na página do meio da metade direita da lista
11      Volte para a linha 3
12   Caso contrário:
13      Saia
```

Ações: FUNÇÕES

Como expressar um algoritmo?

Pseudocódigo:

- algumas sentenças permitem optar por caminhos diferentes de execução: são as operações condicionais (seleção, decisão)

```
1    Pegue a lista telefônica
2    Abra na página do meio
3    Procure o nome na página
4    Se a pessoa procurada está na página
5       Ligue para a pessoa e vá para a linha 13
6    Ou se a pessoa está na parte anterior da lista:
7       Abra na página do meio da metade esquerda da lista
8       Volte para a linha 3
9    Ou se a pessoa está na parte posterior da lista:
10      Abra na página do meio da metade direita da lista
11      Volte para a linha 3
12   Caso contrário:
13      Saia
```

Decisões: CONDIÇÕES

Como expressar um algoritmo?

Pseudocódigo:

- as operações condicionais se baseiam no resultado de expressões booleanas, que são perguntas cujas respostas são sempre SIM ou NÃO

```
1   Pegue a lista telefônica
2   Abra na página do meio
3   Procure o nome na página
4   Se a pessoa procurada está na página
5      Ligue para a pessoa e vá para a linha 13
6   Ou se a pessoa está na parte anterior da lista:
7      Abra na página do meio da metade esquerda da lista
8      Volte para a linha 3
9   Ou se a pessoa está na parte posterior da lista:
10     Abra na página do meio da metade direita da lista
11     Volte para a linha 3
12  Caso contrário:
13     Saia
```

Perguntas: EXPRESSÕES BOOLEANAS

Como expressar um algoritmo?

Pseudocódigo:

- algumas operações indicam repetições (loops)

```
1   Pegue a lista telefônica
2   Abra na página do meio
3   Procure o nome na página
4   Se a pessoa procurada está na página
5      Ligue para a pessoa e vá para a linha 13
6   Ou se a pessoa está na parte anterior da lista:
7      Abra na página do meio da metade esquerda da lista
8      Volte para a linha 3
9   Ou se a pessoa está na parte posterior da lista:
10     Abra na página do meio da metade direita da lista
11     Volte para a linha 3
12  Caso contrário:
13     Saia
```

Repetições: LOOPS

Como expressar um algoritmo?

O que eu faço:

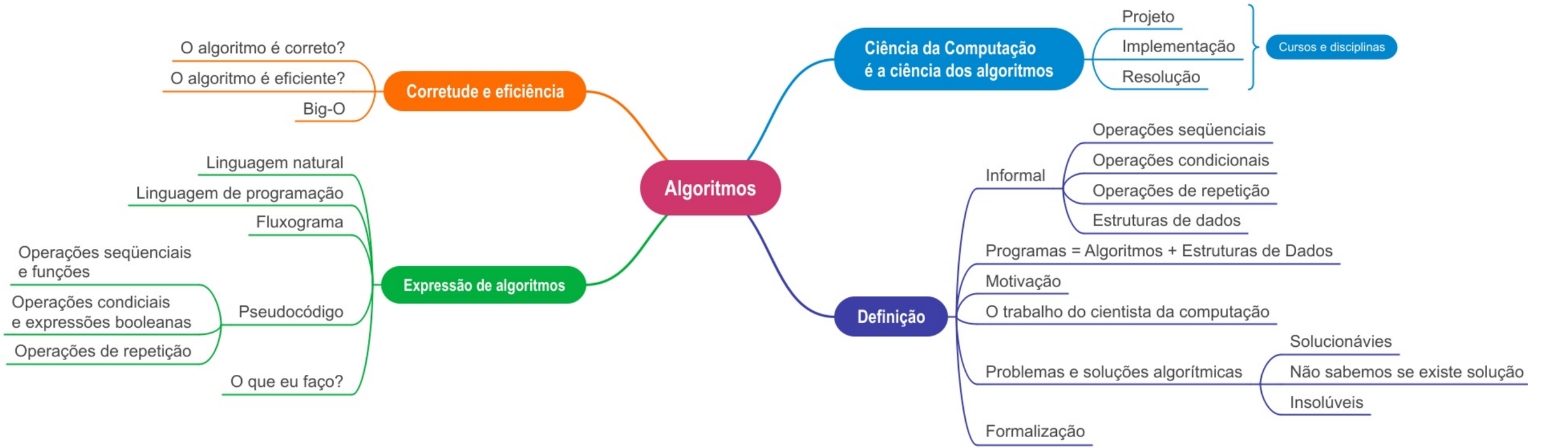
- **entender e pensar como resolver o problema (pensamento computacional) (versões mais simples ajudam!)**
- **pseudocódigo**
- **fluxograma**
- **desenhos, rabiscos e esquemas no caderno**
- **lápis de cores variadas**
- **borracha**
- **apontador**
- **papel de pão, rascunhos, blocos...**

O que eu não faço de jeito nenhum:

- **colocar a mão no teclado**

Adaptado de Charlie Harris, no Unsplash
(https://unsplash.com/photos/a-person-typing-on-a-keyboard-lXGoRVlV7u4)

Em resumo


Algoritmos
Corretude e eficiência
O algoritmo é correto?
O algoritmo é eficiente?
Big-O
Expressão de algoritmos
Linguagem natural
Linguagem de programação
Fluxograma
Pseudocódigo
Operações seqüenciais e funções
Operações condiciais e expressões booleanas
Operações de repetição
O que eu faço?
Ciência da Computação é a ciência dos algoritmos
Projeto
Implementação
Resolução
Cursos e disciplinas
Definição
Informal
Operações seqüenciais
Operações condicionais
Operações de repetição
Estruturas de dados
Programas = Algoritmos + Estruturas de Dados
Motivação
O trabalho do cientista da computação
Problemas e soluções algorítmicas
Solucionávies
Não sabemos se existe solução
Insolúveis
Formalização