

Computabilidade e Complexidade de Algoritmos



Cruzeiro do Sul Virtual
Educação a distância

Material Teórico



Introdução a Computabilidade e Complexidade de Algoritmos

Responsável pelo Conteúdo:

Prof. Dr. Luciano Rossi

Revisão Textual:

Mateus Gonçalves

UNIDADE

Introdução a Computabilidade e Complexidade de Algoritmos



- Aspectos Introdutórios;
- Teoria da Complexidade;
- Teoria da Computabilidade;
- Modelos de Computação.



OBJETIVOS DE APRENDIZADO

- Fornecer ao aluno uma visão geral sobre a área e introduzir os conceitos de complexidade e computabilidade como formas de classificação de algoritmos;
- Apresentar, de forma introdutória, diferentes modelos computacionais que serão úteis para a compreensão dos conceitos seguintes.



Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

Aspectos Introdutórios

Na disciplina de Computabilidade e Complexidade de Algoritmos teremos a oportunidade de refletir sobre aspectos relevantes do projeto e análise de algoritmos eficientes. A área da análise de algoritmos é fundamentalmente teórica, porém com grande impacto prático no desenvolvimento de aplicações computacionais, como veremos no decorrer desse curso. Trata-se de um tema desafiador que nos convida a pensar sobre o processo de desenvolvimento de algoritmos, tanto sob um olhar de eficácia quanto, sobretudo, do ponto de vista da eficiência.

Iniciaremos nossos estudos a partir de diversas definições conceituais que visam o alinhamento do nosso entendimento comum sobre diferentes temas. Assim, nesta unidade introdutória, buscamos discutir sobre os objetivos e a importância da análise de algoritmos, de modo a motivá-lo em aprofundar-se nos temas propostos para esse curso. Além disso, estudaremos sobre as teorias da complexidade e computabilidade de algoritmos, avançando sobre a utilização de diferentes modelos computacionais que oferecem o suporte necessário para as nossas análises.

O termo algoritmo tem sua origem, segundo alguns historiadores, com o sobrenome do matemático persa Al-Khwarizmi, que viveu no século IX, o qual produziu a obra *Algorithmi de numero indorum*, que trata a respeito de algoritmos considerando o sistema de numeração decimal. No contexto da ciência da computação, Cormen (2009) define algoritmo como sendo um procedimento computacional no qual um valor, ou conjunto de valores, é recebido como entrada e, por meio de uma sequência de etapas, produz um valor, ou conjunto de valores, como resultado (saída). Em outras palavras, um algoritmo é uma sequência de passos computacionais bem definidos que transformam uma entrada em uma saída pretendida.

Podemos considerar uma infinidade de exemplos que ilustram a definição informal fornecida, anteriormente, para um algoritmo. Podemos considerar, por exemplo, um algoritmo de ordenação que recebe uma sequência de números como entrada e retorna uma permutação ordenada sobre a entrada como saída.

Veja que, considerando o exemplo anterior, podemos ter diferentes algoritmos com o mesmo objetivo, ou seja, procedimentos diferentes que recebem a mesma entrada e produzem a mesma saída. Assim, podemos concluir que existem diferentes algoritmos que são corretos. Considere um algoritmo correto como sendo aquele que produz a saída esperada a partir de qualquer instância de entrada.



O que diferencia dois algoritmos corretos que produzem o mesmo resultado, a partir de uma mesma instância do problema, por meio de passos diferentes?

O projeto de algoritmos é uma tarefa que deve considerar diferentes aspectos importantes; nesse contexto, o fato do algoritmo ser correto (corretude) é importante, porém existem outros aspectos que são tão importantes quanto esse. Considere dois

algoritmos corretos que produzem o mesmo resultado a partir de passos (procedimentos) diferentes, assim, não será incomum que os algoritmos apresentem, também, níveis diferentes de eficiência.

A eficiência computacional de um algoritmo está ligada ao tempo gasto por ele para produzir o resultado esperado, em outras palavras, a eficiência de um algoritmo é a sua velocidade. Quanto mais rápido um algoritmo produza o resultado correto esperado, tanto maior é a sua eficiência. No universo da computação, a área de análise de complexidade de algoritmos estuda a eficiência computacional dos algoritmos, ou seja, quão rápidos eles podem ser.

Quando consideramos os problemas que podem ser resolvidos computacionalmente, vamos nos deparar com problemas fáceis e difíceis de serem solucionados. Os problemas fáceis são resolvidos rapidamente, enquanto alguns problemas difíceis podem levar anos de processamento, mesmo considerando supercomputadores. Há, ainda, outra classe de problemas que não podem ser resolvidos computacionalmente.

A análise de complexidade de algoritmos fornece os meios para que possamos classificar os algoritmos de acordo com sua respectiva eficiência na resolução de problemas. Nesse contexto, a avaliação dos algoritmos não considera uma medida de tempo absoluta. Seria pouco útil saber que um algoritmo demora dez milissegundos para produzir o resultado esperado. Nesse tipo de avaliação, há uma série de questões que poderiam ser colocadas, mas a principal questão é saber o tamanho da instância de entrada. Por exemplo, um algoritmo de ordenação recebe um conjunto de mil números como entrada e retorna um arranjo ordenado sobre esse conjunto após um milissegundo. Assim, qual seria o tempo de resposta desse mesmo algoritmo considerando, como entrada, uma instância com cem mil números? Se fossemos capazes de responder a questão anterior, em que isso seria útil? Como classificar o algoritmo com esse tipo de análise?

Uma forma mais representativa para medirmos a eficiência de um algoritmo é a partir da utilização de funções. Nesse sentido, uma função nos mostra a relação existente entre o tamanho da instância de entrada e o tempo que o algoritmo levará para produzir o resultado. Assim, a informação mais importante que a função pode nos fornecer é como o tempo que o algoritmo gasta para produzir a saída crescerá à medida que o tamanho da instância de entrada aumenta.

Considere o exemplo de um algoritmo de ordenação, suponha que tenhamos um conjunto de valores de entrada com somente três elementos a serem ordenados e que, para cada elemento, o algoritmo gasta uma unidade de tempo, aqui não é importante se estamos nos referindo a um milissegundo ou a uma hora. Adicionalmente, sabemos que a função que representa a relação entre o tamanho da instância de entrada e o tempo é $f(n) = n$, na qual n é o tamanho da nossa instância, no caso $n = 3$, e $f(n)$, e representa o tempo que o algoritmo gasta para ordenar os valores. Denominamos esse tipo de função de linear, visto que o tempo gasto pelo algoritmo aumenta linearmente ao tamanho da entrada. Veja que para um conjunto de três elementos, o algoritmo gastaria três tempos, para 1.000 elementos levaria 1.000 tempos, e assim sucessivamente. Algoritmos que apresentam complexidade linear são muito rápidos.

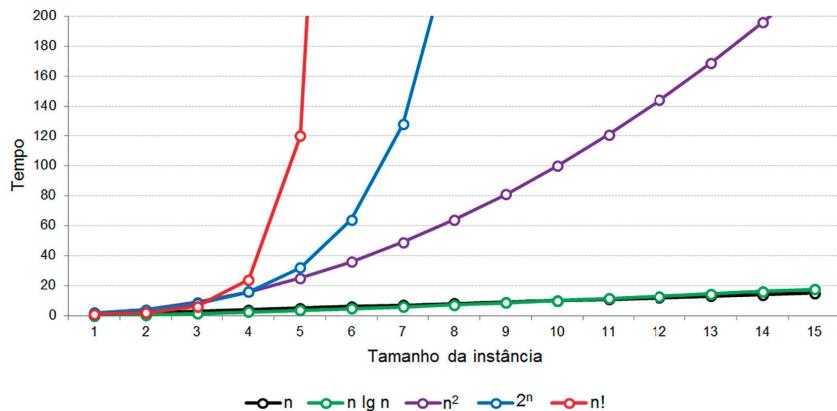


Figura 1 – Diagrama de correlação entre o tamanho das instâncias de entrada e os respectivos tempos de convergência dos algoritmos, de acordo com diferentes funções de complexidade

Fonte: Acervo do conteudista

A Figura 1 representa as correlações existentes entre o tamanho da instância de entrada de algoritmos e o respectivo tempo gasto para produzir o resultado esperado, considerando algumas funções de complexidade mais usuais. Veja que para o exemplo anterior ($f(n) = n$) a curva da função é representada na cor preta no diagrama, é fácil observar que o tempo cresce linearmente em função do tamanho da entrada.

Vamos analisar outro exemplo considerando um algoritmo cuja função de complexidade de tempo seja $f(n) = n^2$ e a respectiva curva da função é representada na cor roxa na Figura 1. Esse tipo de função é denominado função polinomial de grau 2 (ou 2º grau) em razão do valor do maior expoente da função. Veja que se compararmos o crescimento das funções linear e polinomial, essa última cresce muito mais rapidamente que a primeira. Considerando o diagrama da Figura 1, para uma instância de tamanho cinco o tempo correspondente é igual a 25 (5^2), para uma instância com tamanho igual a 10, o algoritmo gastaria 100 tempos, e assim por diante.

A análise de complexidade de tempo é, realmente, muito importante. Vamos verificar um exemplo, mais extremo, para tentarmos evidenciar a importância desse tema. A curva vermelha, representada no diagrama da Figura 1, é aquela que cresce mais rapidamente dentre todas as curvas consideradas. Nesse sentido, essa curva representa o comportamento de uma função factorial, isto é $f(n) = n!$. Como bem sabemos, o factorial de um número é definido da seguinte forma:

$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$$

Assim, se quisermos calcular o valor de cinco factorial teríamos:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Suponha que tenhamos um algoritmo cuja função de complexidade seja $f(n) = n!$, considere, ainda, que o tempo gasto pelo algoritmo para processar uma instância de tamanho um ($n = 1$) seja de um milissegundo ou 0,001 segundo. Quanto tempo esse mesmo algoritmo demoraria em processar uma instância de tamanho igual a 15 ($n = 15$)? A resposta para essa pergunta é, aproximadamente, 41 anos. Outra pergunta, você esperaria todo esse tempo por uma resposta de um algoritmo?

Nesta seção introdutória tivemos um contato inicial com a área de computabilidade e complexidade de algoritmos, por meio da definição de alguns conceitos e da verificação da importância desse tipo de análise para a ciência da computação. A seguir, aprofundaremos no assunto por meio de uma descrição mais específica das Teorias da Complexidade e da Computabilidade, além disso, vamos estudar alguns modelos computacionais que são importantes para a compreensão das teorias.

Teoria da Complexidade

Informalmente, um problema computacional é um conjunto de instâncias dos problemas para as quais há uma associação com suas respectivas soluções que, por sua vez, podem ser obtidas computacionalmente por meio de algoritmos. Nesse sentido, os problemas computacionais podem ser classificados de acordo com sua dificuldade computacional, ou seja, podem ser classificados como fáceis ou difíceis.

Sipser (2007) descreve algumas possibilidades quando tratamos de problemas computacionalmente difíceis:

- Compreender a origem da dificuldade e transformar o problema de modo que ele seja mais facilmente solucionável;
- Aceitar uma solução que não seja perfeita, assim poderíamos obtê-la mais facilmente;
- Identificar os problemas que são difíceis somente para o pior caso – definiremos esse conceito mais a frente –, de modo que a solução seja eficiente na maior parte das vezes; e
- Considerar formas alternativas de computação pode ser a chave para acelerar a obtenção do resultado.

A teoria da complexidade estuda os recursos que são demandados para a solução de problemas computacionais. Dentre esses recursos, poderíamos destacar, principalmente, os consumos de memória e de tempo. No contexto desta disciplina, daremos uma maior ênfase à questão do consumo de tempo como um importante critério para a classificação de algoritmos.

Por vezes, os problemas computacionais são solucionáveis, do ponto de vista conceitual, entretanto, na prática, a solução consome uma quantidade de tempo demasiada de modo a inviabilizar a busca pela solução. Assim, percebemos que a análise da complexidade de algoritmos é uma área importante para que possamos identificar os problemas possíveis de serem solucionados em um tempo adequado.

Veja que a teoria da complexidade trata dos problemas que são solucionáveis computacionalmente, porém nem todos os problemas apresentam essa característica. A seguir, veremos a teoria da computabilidade, que tem o objetivo de classificar os problemas em solúveis e não solúveis computacionalmente.

Teoria da Computabilidade

A teoria da computabilidade se origina na descoberta de que não são todos os problemas que podem ser resolvidos computacionalmente. Enquanto a teoria da complexidade ajuda a verificar quais são os problemas computacionais fáceis e difíceis, a teoria da computabilidade revela quais problemas são, de fato, computáveis ou solúveis.

Existem diversos exemplos que podem ser úteis para a compreensão das diferenças entre as duas teorias. Novamente, Sipser (2007) nos auxilia nesse entendimento descrevendo alguns exemplos. O problema de se ordenar uma lista de números é um problema fácil, portanto solucionável computacionalmente. Por outro lado, um problema de escalonamento de tarefas pode ser muito difícil, considerando que podemos ter muitas tarefas a serem escalonadas de forma que atendam a diferentes restrições. Mesmo com grande capacidade computacional, um problema de escalonamento pode demorar muito tempo para ser resolvido, mas, ainda assim, é um problema solucionável computacionalmente. Ambos os exemplos, descritos anteriormente, são objetos de estudo da teoria da complexidade.

Suponha que se queira determinar se um enunciado matemático é verdadeiro ou falso, de acordo com Sipser (2007), essa não é uma tarefa solucionável computacionalmente. Apesar de ser trivial para a maioria dos matemáticos, não há um algoritmo capaz de resolver essa questão. Trata-se, portanto, de um problema não computável e a teoria da computabilidade nos ajuda a identificar esse tipo de problema.

A teoria da computabilidade se baseia no estudo de modelos computacionais teóricos, por meio dos quais podemos realizar a classificação de problemas e, em um contexto histórico, representam os fundamentos para a construção de computadores com as características que conhecemos hoje.

Na próxima seção, discutiremos sobre alguns modelos computacionais que serão úteis para a aplicação das teorias anteriormente descritas. Teremos especial atenção aos autômatos e, consequentemente, às máquinas de *Turing*.

Modelos de Computação

Um modelo computacional é um computador idealizado de modo que nos permita manipulá-lo por meio de uma teoria matemática. Esses modelos são muito precisos, do ponto de vista de sua definição e, também, não descrevem os detalhes de um computador moderno, nos quais estamos habituados a trabalhar e reúnem grande complexidade.

Iniciaremos nosso estudo a partir de um modelo bastante simples que é denominado autômato finito. Esse tipo de modelo é muito útil e fundamental para a compreensão de outros modelos mais sofisticados.

Autômatos Finitos

Um autômato finito, também denominado máquina de estados finitos, é um modelo computacional que aceita ou rejeita cadeias de símbolos. Vamos iniciar nosso estudo sobre os autômatos finitos de forma abstrata, considerando a exploração de um diagrama de estados, representado na Figura 2, o qual descreve de forma gráfica um autômato finito.

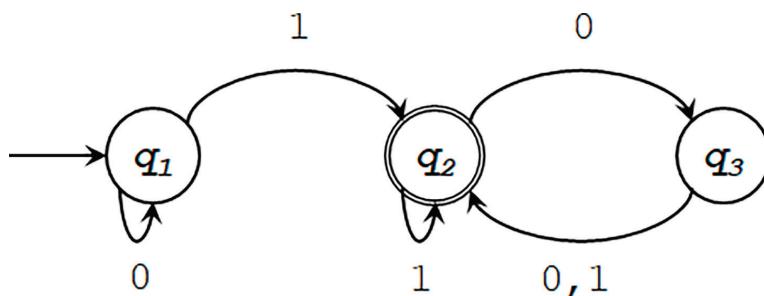


Figura 2 – Diagrama de estados que representa, conceitualmente, um autômato finito

Fonte: Adaptado de SIPSER, 2007

O diagrama de estados, representado na Figura 2, é composto por três estados: q_1 , q_2 e q_3 . Note que o estado q_1 apresenta uma seta incidente nele, de forma a indicar que ele é o estado inicial, a partir do qual se inicia o processo de leitura da cadeia de caracteres.

O estado de aceitação é aquele que determina que a cadeia de entrada será aceita se as transições entre os estados finalizarem nesse estado específico. No caso do nosso exemplo, o estado q_2 é o estado de aceitação, pois está representado por um círculo duplo.

As setas que interligam um estado a outro indicam as transições. Note que cada transição está rotulada com um símbolo pertinente ao alfabeto considerado, no nosso caso o alfabeto é $\Sigma = \{0,1\}$. Por exemplo, se o autômato estiver no estado q_2 e o símbolo lido for 0, ele fará a transição para o estado q_3 , essa transição é representada pela seta cujo rótulo é 0 e interliga os estados q_2 e q_3 , respectivamente. Por outro lado, se no estado q_2 for lido o símbolo 1, a transição correspondente levará ao próprio estado q_2 . Veja que deve existir uma transição para cada símbolo do alfabeto a partir de cada estado.



Importante!

Um alfabeto é um conjunto finito não vazio de símbolos sobre o qual uma cadeia é constituída. Sobre o alfabeto $\Sigma = \{0,1\}$ poderíamos constituir, por exemplo, a cadeia 00110101.

Considerando o diagrama de estados representado na Figura 2, o qual denominaremos daqui em diante como M_1 , vamos realizar uma simulação do funcionamento de M_1 tendo como entrada a cadeia 01100. Nesse caso, o processamento teria os seguintes passos:

- Inicia no estado q_1 ;
- Lê 0, faz a transição de q_1 para q_1 ;

- Lê 1, faz a transição de q_1 para q_2 ;
- Lê 1, faz a transição de q_2 para q_2 ;
- Lê 0, faz a transição de q_2 para q_3 ;
- Lê 0, faz a transição de q_3 para q_2 ;
- Finaliza com a aceitação da cadeia, pois M_1 está no estado de aceitação.

Veja que, na simulação anterior, M_1 aceita a cadeia de entrada, pois, após toda a cadeia ter sido lida, M_1 está no estado de aceitação q_2 . A realização de várias simulações, considerando cadeias de entrada diferentes, nos permitirá identificar um padrão nas cadeias que são aceitas. Assim, veremos que M_1 aceita cadeias que terminem com o símbolo 1 ou cadeias que terminem com um número par de 0's após o último 1. Qualquer outra cadeia fora desse padrão é rejeitada.



Importante!

Uma linguagem é um conjunto de cadeias. Suponha A como sendo o conjunto de cadeias que a máquina M_i reconhece, então utilizamos a notação $L(M_i)$ para dizer que A é a linguagem de M_i .

Sobre o autômato finito dizemos que ele é uma autômato finito determinístico, pois a saída produzida pela cadeia de entrada será sempre aceita ou rejeitada. Nesse contexto, toda a transição é realizada de maneira única a partir do estado anterior, assim, o próximo estado está determinado. Para esse tipo de comportamento damos o nome de computação determinística.

A utilização de um diagrama de estados para representar um autômato finito facilita o entendimento, porém necessitamos de uma maneira formal para definir esses modelos. Há duas justificativas para a utilização de uma definição formal de autômatos finitos: precisão e notação. A precisão elimina possíveis incertezas a respeito do modelo. Por sua vez, a notação fornece os meios para uma expressão mais clara do pensamento.

Considerando os componentes, descritos anteriormente, de um autômato finito, podemos dizer que ele é uma 5-upla (lê-se quíntupla) representada por $(Q, \Sigma, \delta, q_0, F)$, os quais são definidos da seguinte forma:

- Q é o conjunto de estados;
- Σ é o alfabeto;
- $\delta: Q \times \Sigma \rightarrow Q$ é a função de transição;
- $q_0 \in Q$ é o estado inicial; e
- $F \subseteq Q$ é o conjunto dos estados de aceitação.

A definição formal de um autômato finito é capaz de descrever precisamente o modelo em questão. Vamos considerar o autômato M_1 da Figura 2 e representá-lo formalmente. Nesse contexto, teremos o seguinte:

- $Q = \{q_1, q_2, q_3\}$;
- $\Sigma = \{0, 1\}$;

- δ :

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- q_1 é o estado inicial;
- $F = \{q_2\}$.

Vamos interpretar a definição formal de M_1 confrontando as definições e o respectivo diagrama de estados. O diagrama é composto por três estados, os quais são descritos na definição formal por $Q = \{q_1, q_2, q_3\}$, isto é, o conjunto Q contém os estados do nosso autômato. As transições no diagrama de estados são rotuladas com símbolos, os quais compõem o alfabeto considerado, ou seja, $\Sigma = \{0,1\}$.

A descrição da função de transição δ é um pouco mais complicada, vamos considerar a notação formal $\delta: Q \times \Sigma \rightarrow Q$ que pode ser lida como o produto cartesiano dos elementos de Q e Σ que mapeiam um elemento de Q . Se considerarmos o par ordenado $(q_1, 0)$ com $q_1 \in Q$ e $0 \in \Sigma$, a função de transição mapeará para o estado q_1 . Veja que a tabela que define a função de transição descreve todos os elementos de Q nas linhas e todos os elementos de Σ nas colunas. Quando tomamos qualquer par ordenado $(q, \sigma) \in Q \times \Sigma$, devemos procurar o estado que está descrito na interseção da linha e da coluna, indicadas pelo par ordenado (q, σ) respectivamente. Assim, se o estado atual é, por exemplo, q_2 (linha) e o símbolo lido é 0 (coluna) o estado mapeado pela função é q_3 (interseção).

O quarto item da representação formal é o estado inicial que, no caso do nosso exemplo M_1 , é o estado q_1 . Finalmente, F é o conjunto dos estados de aceitação, note que podemos ter mais de um estado de aceitação.

Ambas as representações, diagrama de estados e definição formal, referem-se ao mesmo modelo, no entanto a definição formal é mais precisa e utiliza notação, características que são importantes para esse contexto. Sabemos que M_1 reconhece algumas cadeias de símbolos específicas e, também sabemos, que um conjunto de cadeias é uma linguagem. Nesse sentido, a linguagem A que M_1 reconhece é definida da seguinte forma:

$A = \{w | w \text{ contém pelo menos um } 1 \text{ e um número par de } 0\text{s após o último } 1\}$, em que w é uma cadeia de símbolos.

Consideremos outro exemplo, suponha a seguinte definição formal de um autômato finito $M_2 = (\{s, q_1, q_2, r_1, r_2\}, \{a, b\}, \delta, s, \{q_1, r_1\})$ cuja função de transição δ é representada na tabela abaixo:

	a	b
s	q_1	r_1
q_1	q_1	q_2
q_2	q_1	q_2
r_1	r_2	r_1
r_2	r_2	r_1

Note que a definição formal, dada anteriormente, refere-se ao autômato finito representado pelo diagrama de estados da Figura 3.

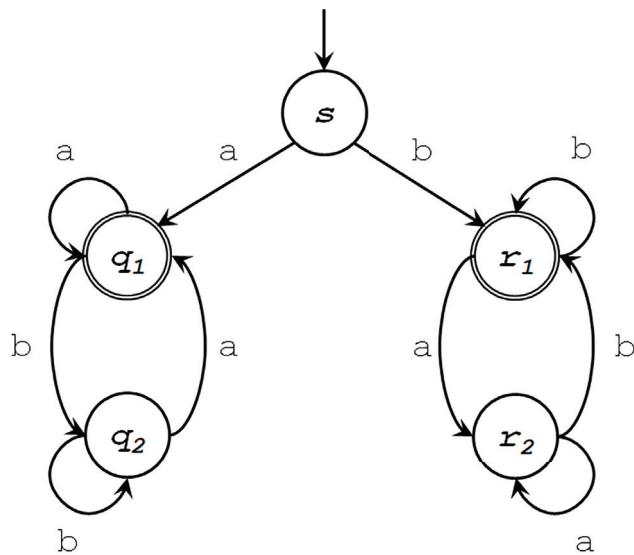


Figura 3 – Diagrama de estados que representa, conceitualmente, um autômato finito

Fonte: Adaptado de SIPSER, 2007

Após a leitura do primeiro símbolo da cadeia, M_2 faz a transição à direita ou à esquerda, não podendo retornar para o lado oposto em nenhuma das situações. A linguagem reconhecida pelo referido autômato é:

$$A = \{w \mid w \text{ começa e termina com o mesmo símbolo}\}$$

Ilustramos informalmente a computação de um autômato finito por meio da leitura dos símbolos que compõem uma cadeia e realizando a transição do estado original para o próximo estado, a depender do símbolo lido. Essa descrição para a computação é fácil de entender, porém apresenta as mesmas deficiências que a representação por meio de um diagrama de estados para um autômato finito, ou seja, falta precisão, podendo gerar ambiguidade.

Uma definição formal para a computação de um autômato finito pode ser feita, matematicamente, de forma mais precisa. A definição, descrita por Sipser (2007), considera um autômato $M = (Q, \Sigma, \delta, q_0, F)$ e uma cadeia de símbolos $w = w_1 w_2 \dots w_n$ na qual cada w_i é um membro do alfabeto Σ . Nesse contexto, M aceita w se existe uma sequência de estados r_0, r_1, \dots, r_n em Q que atenda três condições:

- $r_0 = q_0$,
- $\delta(r_i, w_{i+1}) = r_{i+1}$, para $i = 0, \dots, n - 1$, e
- $r_n \in F$.

A primeira condição define que a computação começa pelo estado inicial, ou seja, o primeiro estado da sequência (r_0) é igual ao estado inicial (q_0). A segunda condição ilustra o comportamento da função de transição, por exemplo, se o estado atual é r_5 e é lido o símbolo w_6 , é feita a transição para o estado r_6 . Finalmente, a terceira condição diz que se o estado final r_n entre os estados de aceitação, então M aceita a entrada.



Importante!

Chamamos de linguagem regular toda linguagem para a qual existe um autômato finito que a reconhece.

Os autômatos finitos que analisamos até o momento têm uma característica importante, sempre que há uma transição o próximo estado ele é sempre único e conhecido (determinado). Por conta dessa característica, vamos chamá-los, daqui em diante, de autômatos finitos determinísticos. Nesse sentido, vamos verificar outro tipo de autômato que não apresenta essa característica.

Considere o autômato finito da Figura 4, o qual denominaremos de M_3 , ele parece um pouco estranho se comparado com os autômatos vistos até agora. Podemos listar algumas diferenças importantes. Os estados nos autômatos finitos determinísticos sempre têm uma opção de transição para cada símbolo do alfabeto. Essa característica não é observada em M_3 , veja que o estado q_1 apresenta duas setas de transição para o símbolo 1. O estado q_3 não tem uma seta de transição para o símbolo 0. Note, também, que os autômatos finitos determinísticos têm suas transições de acordo com os símbolos do alfabeto. No entanto, em M_3 , a transição entre q_2 e q_3 é rotulada com o símbolo ϵ .

O autômato M_3 é denominado autômato finito não determinístico, ou seja, não há somente uma maneira de se realizar uma transição, pode-se transitar para mais de um estado após a leitura de um símbolo da cadeia. Vamos considerar um exemplo: suponha que estejamos no estado inicial de M_3 , o estado q_1 , e que lemos o símbolo 1, veja que há uma transição para o estado q_2 e outra que retorna para q_1 . Desse modo, haverá um desdobramento da computação, ou seja, teremos dois ramos de computação, um no estado q_1 e outro em q_2 . A partir daí, haverá duas máquinas M_3 rodando simultaneamente, a partir de cada estado alcançado.

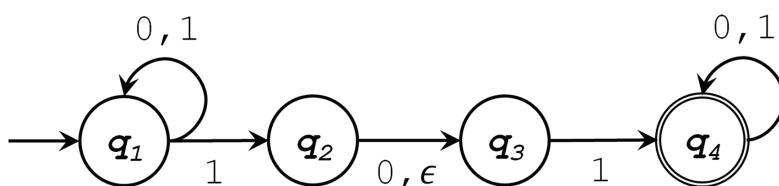


Figura 4 – Diagrama de estados que representa, conceitualmente, um autômato finito não determinístico

Fonte: Adaptado de SIPSER, 2007

O símbolo novo que pode aparecer nos autômatos finitos não determinísticos (ϵ – lê-se *épsilon*), indica uma transição espontânea. Nesse caso, sempre que se atinja um estado que é origem de uma transição rotulada com ϵ , haverá uma duplicação da máquina (criação de um novo ramo de computação) sem que se tenha lido algum símbolo. Assim, uma máquina estará no estado atual e outra no estado indicado pela transição espontânea. Considerando o autômato M_3 , da Figura 4, suponha que se chegue ao estado q_2 , desse modo sem que haja a leitura do próximo símbolo haverá a criação de um novo ramo de computação, a partir da transição espontânea para q_3 , mantendo o ramo original em q_2 .

A computação de um autômato finito não determinístico segue de forma similar ao determinístico, entretanto, poderemos ter diferentes ramos de computação ocorrendo paralelamente. Para ilustrar a computação de um autômato finito não determinístico, vamos realizar uma simulação considerando M_3 sobre a cadeia de entrada 010110, cada passo dessa simulação é representado na Figura 5.

O início da computação considera o estado q_1 , conforme descrição de M_3 , o primeiro símbolo da cadeia a ser lido é o 0. A transição correspondente ao símbolo lido indica que o estado ativo continua sendo q_1 . Veja na Figura 5 a representação desta transição, da esquerda para a direita. O primeiro estado é o q_1 e a transição a partir do símbolo 0 leva ao segundo estado que, também, é o q_1 . O segundo símbolo lido é o 1, note que temos duas transições possíveis a partir dessa entrada. A primeira transição leva, novamente ao estado q_1 , outra transição possível resulta em um novo ramo de computação em q_2 . Além disso, a partir de q_2 há uma transição espontânea para q_3 o que gerará mais um ramo de computação.

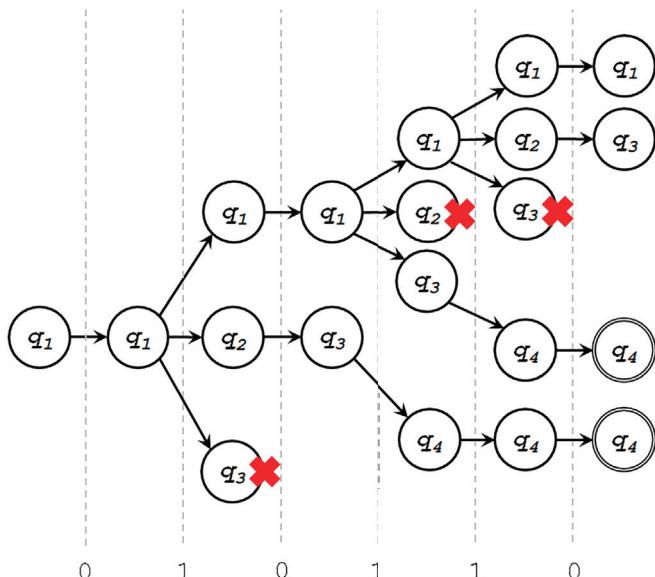


Figura 5 – Simulação da computação do autômato finito não determinístico sobre a cadeia

Fonte: Adaptado de SIPSER, 2007

A configuração atual da nossa simulação nos mostra que existem três ramos ativos de computação, os quais consideram os estados q_1 , q_2 e q_3 . Nesse sentido, a computação continuará, simultaneamente, a partir de cada um deles. O próximo símbolo da cadeia a ser lido é o 0, todos os ramos de computação considerarão estátua entrada. Veja que o estado q_3 não prevê uma transição para a entrada 0, assim o ramo de computação representado por esse estado “morre”, não havendo estado subsequente para a manutenção desse ramo. Para os dois estados ativos restantes, q_1 e q_2 , as transições após a leitura do símbolo 0 são para os estados q_1 e q_3 , respectivamente.

O próximo símbolo lido é 1, assim o estado ativo q_1 realiza as transições para os estados q_1 , q_2 e q_3 , ou seja, essas transições dão origem a três novos ramos de computação. Do mesmo modo, o estado q_3 realiza a transição para o estado q_4 , após o último símbolo lido. Novamente, o penúltimo símbolo lido é o 1. A partir do estado

q_1 , três novos ramos surgem pelas transições para q_1 , q_2 e q_3 . Veja que o estado ativo q_2 não prevê transição a partir do símbolo 1, assim esse ramo de computação não continuará. As outras transições realizadas são de q_3 para q_4 e de q_4 para q_4 .

Finalmente, após a leitura do último símbolo, 0, temos que o estado ativo q_1 faz a transição para o próprio q_1 , o estado q_2 transita para q_3 , o estado q_3 não prevê transição a partir do símbolo 0, então esse ramo é extinto, e os dois últimos estados ativos q_4 transitam para ele mesmo.

Após a finalização da computação do autômato M_3 sobre a cadeia de símbolo 010110, veja que temos quatro estados resultantes dos ramos de computação ativos, e dois destes estados coincidem com o estado de aceitação (q_4). Assim, podemos dizer que M_3 aceita a cadeia de entrada.

Qualquer autômato finito não determinístico pode ser convertido em um autômato finito determinístico equivalente. Essa propriedade é especialmente útil, pois, por vezes, é mais simples construir autômatos não determinísticos. Além disso, os autômatos não determinísticos podem ser muito menores e mais fáceis de entender que os seus equivalentes determinísticos.

O diagrama de estados, apresentado na Figura 4, é uma representação simples de se compreender, porém, da mesma forma que descrevemos para os autômatos finitos determinísticos, há a necessidade de uma definição formal, também, para os autômatos finitos não determinísticos.

A representação formal de um autômato finito não determinístico é muito similar àquela descrita para os determinísticos. Nesse sentido, esse novo modelo é representado por uma 5-upla $(Q, \Sigma, \delta, q_0, F)$, cujos componentes são descritos da seguinte forma:

- Q é um conjunto de estados;
- Σ é um alfabeto;
- $\delta: Q \times \Sigma \rightarrow P(Q)$ é a função de transição;
- $q_0 \in Q$ é o estado inicial; e
- $F \subseteq Q$ é o conjunto de estados de aceitação.

Considerando o autômato finito não determinístico representado por um diagrama de estados na Figura 4, sua representação formal seria a 5-upla $(Q, \Sigma, \delta, q_1, F)$, cuja descrição dos componentes seriam:

- $Q = \{q_1, q_2, q_3, q_4\}$;
- $\Sigma = \{0, 1\}$;
- δ é descrita pela tabela:

	0	1	ϵ
q_1	$\{q_1\}$	$\{q_1, q_2\}$	\emptyset
q_2	$\{q_3\}$	\emptyset	$\{q_3\}$
q_3	\emptyset	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$	\emptyset

- q_1 é o estado inicial; e
- $F = \{q_4\}$.

A representação formal de um autômato finito não determinístico segue de forma muito similar aos exemplos determinísticos que estudamos. Há algumas diferenças sutis que devemos nos atentar, principalmente no que diz respeito à computação. Note que quando uma transição é feita não há um único estado de destino, mas sim um conjunto de estados. Por exemplo, se estamos no estado q_1 e lemos o símbolo 1, a função de transição indica um conjunto de estados de destino e, assim, haverá novos ramos de computação. Outra diferença se refere à transição espontânea, todo estado de origem prevê um destino para uma transição espontânea. Caso não haja tal transição, a função indica como destino um conjunto vazio (\emptyset), esse tipo de destino indica o fim de um ramo de computação.

A definição formal de computação para um autômato finito não determinístico é definida por Sipser (2007) da seguinte forma. Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito não determinístico e w é uma cadeia sobre o alfabeto Σ . Dizemos que M aceita w se w pode ser escrita como membros de Σ_ϵ da seguinte forma $w = y_1 y_2 \dots y_n$ e existe uma sequência de estados $r_0, r_1, \dots, r_m \in Q$ que atende a três condições:

- $r_0 = q_0$;
- $r_{i+1} \in \delta(r_i, y_{i+1})$, para $i = 0, \dots, m - 1$; e
- $r_m \in F$.

Como vimos até aqui, os autômatos finitos são modelos muito úteis quando estamos interessados no reconhecimento de padrões em conjuntos de dados. Esses tipos de modelos podem ter aplicações práticas em áreas como processamento de voz e reconhecimento de caracteres ópticos. Entretanto, os autômatos finitos modelam computadores com uma quantidade muito pequena de memória.

Na próxima seção, vamos conhecer um modelo computacional mais poderoso que é semelhante a um autômato finito, porém possui uma memória ilimitada e irrestrita, são as máquinas de *Turing*.

Máquinas de Turing

As máquinas de *Turing* são modelos mais específicos de computadores de uso geral. Esse tipo de modelo foi proposto inicialmente pelo matemático britânico Alan *Turing* em 1936. *Turing* se notabilizou, dentre outras conquistas, pela concepção de uma máquina que foi capaz de quebrar os códigos utilizados pelos nazistas para ocultar suas mensagens durante a Segunda Grande Guerra Mundial. Estudiosos desse conflito especulam que a intervenção de *Turing*, e de seus colegas, foi determinante para a vitória dos Aliados e para o encurtamento do período de guerra, levando à preservação de milhares de vidas. Além disso, *Turing* é reconhecido como o pai da Ciência da Computação.

Uma máquina de *Turing* utiliza uma “fita” infinita como memória, além disso, ela conta com uma cabeça que permite a leitura e a escrita de símbolos na fita e se

movimenta por toda sua extensão. A definição formal de uma máquina de Turing é uma 7-upla $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{aceita}}, q_{\text{rejeita}})$, os componentes da definição são os seguintes:

- Q é o conjunto de estados;
- Σ é o alfabeto de entrada;
- Γ é o alfabeto da fita, no qual $\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$;
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{E, D\}$ é a função de transição;
- $q_0 \in Q$ é o estado inicial;
- $q_{\text{aceita}} \in Q$ é o estado de aceitação; e
- $q_{\text{rejeita}} \in Q$ é o estado de rejeição.

Suponha que queremos verificar se uma determina cadeia de símbolos pertence à linguagem $A = \{w\#w \mid w \in \{0,1\}^*\}$. São exemplos de cadeias pertencentes a essa linguagem: 11100#11100, 010101#010101 e 000#000. Note que para pertencer à linguagem basta que a cadeia parcial à esquerda de # seja igual à cadeia parcial à direita.

Para realizar essa verificação, considere que há somente a cadeia escrita na fita da máquina de Turing, seguida de espaços em branco. Inicialmente, a cabeça identificaria o primeiro símbolo e faria uma marcação sobre ele. Em seguida, a cabeça se moveria para a direita até encontrar o símbolo # e verificaria o próximo símbolo à direita, caso combine com o primeiro símbolo lido, a cabeça faz a marcação para identificar que aquele símbolo foi lido e retorna para o início da fita. A operação é repetida até que não haja mais caracteres a serem lidos.

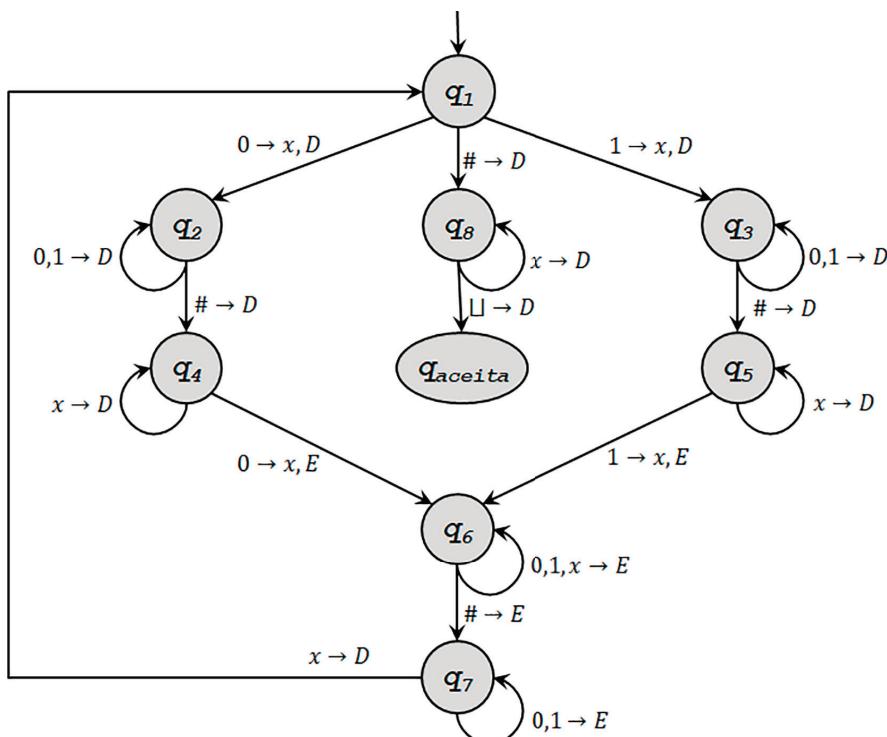


Figura 6 – Diagrama de estados que representa uma máquina de Turing que reconhece cadeias que pertençam à linguagem

Fonte: Adaptado de SIPSER, 2007

O diagrama de estados, representado na Figura 6, descreve a máquina de Turing que reconhece cadeias de caracteres pertencentes à linguagem A. O estado inicial é q_1 , a notação $1 \rightarrow x, D$, que rotula a transição à direita do estado inicial, indica que caso o símbolo 1 seja lido, deverá ser escrito o símbolo x na fita e realizar um deslocamento à direita (D) para a próxima posição na fita, além disso, há a transição para o estado q_3 . Caso o próximo símbolo lido seja 0 ou 1 a cabeça se move à direita na fita sem que haja nenhuma escrita, desse modo, a transição é feita para o próprio estado q_3 . Essa dinâmica de transições entre estados e leitura, movimentação e escrita na fita segue essa mesma lógica até que termine a leitura da cadeia.

Suponha a cadeia $10\#10 \in A$ acompanhe os passos de computação da máquina de Turing, da Figura 6, sobre essa cadeia, no quadro abaixo. A ordem das representações é de cima para baixo, da esquerda para a direita, o símbolo \sqcup representa um espaço vazio na fita.

Tabela 1

$q_1 10\#10 \squparrow$	$xq_1 0\#x0 \squparrow$	$xxq_1\#xx \squparrow$
$xq_1 0\#10 \squparrow$	$xxq_2\#x0 \squparrow$	$xx\#q_8 xx \squparrow$
$xxq_2\#10 \squparrow$	$xx\#q_4 x0 \squparrow$	$xx\#xq_8 x \squparrow$
$xx\#q_4 10 \squparrow$	$xx\#xq_4 0 \squparrow$	$xx\#xxq_8 \squparrow$
$xx\#q_8 0 \squparrow$	$xx\#q_8 xx \squparrow$	$xx\#xx \squparrow q_{aceita}$
$xx\#xx \squparrow$		
$q_8 \squparrow$		

Nesta unidade introdutória, tivemos nosso primeiro contato com a área da análise de algoritmos e conhecemos os modelos computacionais básicos para esse tipo de análise. Nas próximas unidades, vamos nos aprofundar no assunto, considerando uma notação particular para a classificação de algoritmos.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

▶ Vídeos

O que é um algoritmo, e por que você deve se importar com isso?

https://youtu.be/8WU_E9tNnEw

Complexidade de algoritmos

<https://youtu.be/KVIGx-9Cu04>

Autômatos Finitos Determinísticos e Não-Determinísticos

<https://youtu.be/mCwQoM8KaZk>

📄 Leitura

Alan Turing

<https://bit.ly/3eGg0TO>

Referências

CORMEN, T. H. *et al.* **Introduction to algorithms**. MIT press, 2009.

SIPSER, M. **Introdução à Teoria da Computação**. 2^a edição. Cengage Learning, 2007.

TOSCANI, L. V. **Complexidade de algoritmos**. v. 13: UFRGS. 3. Porto Alegre Bookman, 2012. (*e-book*)



Cruzeiro do Sul
Educacional