UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

HENRIQUE DE S. Q. DOS SANTOS

WITOR M. A. DE OLIVEIRA

Intratabilidade

São Carlos

2020

Henrique de S. Q. dos Santos, NUSP 10819029

Witor M. A. de Oliveira, NUSP 10692190

Intratabilidade

Trabalho apresentado à disciplina SCC0205 - Teoria da Computação e Linguagens Formais como requisito parcial para obtenção de nota no trabalho final da mesma disciplina.

Professor: Dr. Diego Raphael Amancio

São Carlos

2020

**SUMÁRIO**

[**1 INTRODUÇÃO**](#_1fob9te) **4**

[**2 DESENVOLVIMENTO**](#_cwb7k98y9prt) **4**

[2.1 Definições](#_bp3x0stpvp3y) 4

[2.1.1 Alfabeto](#_vks7rj28dyu7) 4

[2.1.2 Linguagem](#_4xhlsjlblwbc) 5

[2.1.3 Aceitação](#_zbi4ltwvplrt) 5

[2.1.3.1 Aceitação em tempo polinomial](#_83wz2cefg64m) 5

[2.1.4 Decisão](#_p0ek20co9phk) 5

[2.1.4.1 Decisão em tempo polinomial](#_8u2rchtyteja) 6

[2.1.5 Verificação](#_bf2by931cfdv) 6

[2.1.6 Redutibilidade em tempo polinomial](#_mzcssislemei) 6

[2.2 Complexidade computacional](#_hhpzu9qw2pkj) 6

[2.3 Classes P e NP](#_jqy6rdvpgww2) 7

[2.3.1 NP Completo](#_66dlfgzh37hr) 8

[2.3.1.1 Consequências da NP Completude](#_spdqh334nfcw) 9

[2.3.1.2 Exemplos de problemas NP Completos em grafos](#_gqrtusbnk5bk) 9

[2.3.1.2.1 O Problema do Caixeiro Viajante](#_565gmmu0vq2o) 9

[2.3.1.2.2 O Problema da Cobertura de Nós](#_dcsxnoju3csa) 10

[2.3.1.2.3 O Problema Clique](#_8i8uh7rww24e) 10

[2.3.2 NP Hard](#_g4j9durarsd2) 11

[2.4 Hipótese P ≠ NP](#_q3f5u5xog38y) 11

[**3 REFERÊNCIAS BIBLIOGRÁFICAS**](#_7vew6zf91eoz) **12**

# 1 INTRODUÇÃO

"Alguns problemas computacionais são resolvíveis a princípio, mas as soluções requerem tanto tempo ou espaço a mais que elas não são usadas na prática. Tais problemas são chamados intratáveis." (Sipser M., 1997)

(Harel, 1987) dá a qualidade de razoável ao algoritmo de tempo polinomial, ou seja, àquele a qual a sua ordem de magnitude de desempenho temporal é limitada superiormente por uma função polinomial , onde é o tamanho das entradas desse algoritmo. No caso contrário, o algoritmo é dito não razoável. Ainda, define que um problema que admite apenas soluções não razoáveis ou com tempo exponencial são intratáveis.

Em suma, intratabilidade, no contexto computacional, se refere aos problemas computacionais que são possíveis de se resolver na teoria, mas na prática, são inviáveis já que, ou levam muito tempo para realizar a computação necessária, ou gastam muito espaço para tal ou ambos, mesmo quando as entradas para o problema são pequenas. Neste trabalho, iremos discorrer sobre as bases dos problemas intratáveis.

# 2 DESENVOLVIMENTO

## 2.1 Definições

Abaixo, definições que serão usadas ao longo deste trabalho. Todas as definições foram retiradas de (Cormen, 2009). As definições retiradas de outras fontes possuem suas citações.

### 2.1.1 Alfabeto

Um “alfabeto é um conjunto finito de símbolos.”

### 2.1.2 Linguagem

Uma “linguagem sobre é qualquer conjunto de cadeias feitas de símbolos do alfabeto .”

### 2.1.3 Aceitação

Um “algoritmo aceita uma cadeia se, dada a entrada , a saída do algoritmo é 1. A linguagem aceita por é o conjunto de cadeias = { : = 1}, ou seja, o conjunto de cadeias que o algoritmo aceita. Se = 0, é dito que o algoritmo rejeita a cadeia ”;

#### 2.1.3.1 Aceitação em tempo polinomial

Uma “linguagem é aceita em tempo polinomial por um algoritmo se é aceita por e se, em adição, existe uma constante tal que para qualquer cadeia de tamanho como entrada, com , o algoritmo aceita em tempo .” Por hora, basta dizermos que significa que o tempo de execução do algoritmo é limitado superiormente pela função = . Essa definição será aprofundada na seção 2.2.

### 2.1.4 Decisão

Uma “linguagem é decidida por um algoritmo se qualquer cadeia binária em é aceita por e qualquer cadeia binária não pertencente à é rejeitada por .”

#### 2.1.4.1 Decisão em tempo polinomial

Uma “linguagem é **decidida** **em tempo polinomial** por um algoritmo se existir uma constante que, para qualquer cadeia de tamanho , o algoritmo decide corretamente se em tempo .”

### 

### 2.1.5 Verificação

Um “problema é verificado se pudermos nos certificar de que a solução está correta em tempo polinomial em relação ao tamanho da entrada do problema.”

### 2.1.6 Redutibilidade em tempo polinomial

Uma “linguagem é redutível em tempo polinomial para uma linguagem se existir uma função computável em tempo polinomial tal que para todo se, e somente se, . Essa função é chamada função de redução e um algoritmo em tempo polinomial que computa é chamado algoritmo de redução.”

## 2.2 Complexidade computacional

A complexidade computacional, ramo da Teoria da Computação, determina os requisitos de recursos para diferentes problemas, lidando com afirmações sobre qualquer algoritmo possível para um problema. Os recursos relacionados à memória computacional e tempo necessário até encontrar a solução são, geralmente, os pontos com maior foco em particular.

A complexidade temporal assintótica é o estudo sobre como o tempo de execução do algoritmo aumenta em relação ao tamanho da entrada permitida quando o tamanho dessa entrada aumenta sem restrições. Um algoritmo assintoticamente eficiente é sempre preferível em todos os casos, exceto quando a entrada é muito pequena (Cormen, 2009). Neste trabalho, sempre que nos referirmos à complexidade , onde é alguma função limitante superior, estamos dizendo que o problema tem uma complexidade temporal (complexidade de tempo) assintótica limitada superiormente por , com uma constante qualquer, para qualquer .

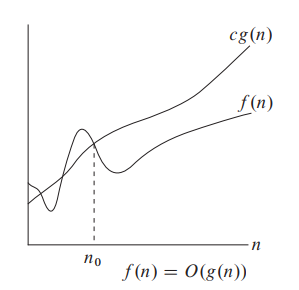


Imagem retirada de (Cormen, 2009)

## 2.3 Classes P e NP

A classe de complexidade P (polinomial determinística) engloba todos os problemas que podem ser resolvidos em tempo polinomial e foi apresentada, primeiramente, por (Cobham, 1964). Rigorosamente, definimos como pertencentes à classe P todos os problemas que podem ser resolvidos por uma Máquina de Turing Determinística em tempo polinomial. De acordo com (Cormen, 2009), mais especificamente, são problemas que podem ser resolvidos em para alguma constante , onde é o tamanho da entrada para o problema. Dada a definição de **decisão** e **aceitação** em tempo polinomial, uma outra definição, ainda por (Cormen, 2009), diz que, dada uma linguagem ,

P = { {0,1}\* : existe um algoritmo que decida em tempo polinomial}.

Ou seja, P é, também, a classe de linguagens que podem ser aceitas em tempo polinomial. A grande maioria dos problemas estudados estão em P, alguns famosos como o cálculo do maior divisor comum e determinar se um número é primo.

Já a classe de complexidade NP (polinomial não determinística) é o conjunto de problemas que podem ser **verificados** em tempo polinomial por uma Máquina de Turing Não Determinística e foi apresentada por (Edmond, 1965). Outra definição é que uma linguagem pertence à NP se, e somente se, existem duas entradas , um algoritmo em tempo polinomial, e , uma constante, tal que

: existe um certificado com tal que }.

Ou seja, um algoritmo verifica a linguagem em tempo polinomial. Na Teoria da Computação, é fato que toda Máquina de Turing determinística é, também, uma Máquina de Turing Não Determinística com, no máximo, um movimento e, por isso, P NP (Comen, 2009).

### 2.3.1 NP Completo

Informalmente, (Cormen, 2009) define como NP Completo um problema NP que é tão “difícil” quanto qualquer problema em NP. (Sipser, M. 1997) define como NP Completo uma linguagem que satisfaz duas condições:

1. está em NP;
2. Todo algoritmo em NP é redutível à em tempo polinomial.

O termo NP Completo foi apresentado pela primeira vez por (Cook, 1971). Em seu artigo, é mostrado que qualquer problema de reconhecimento que pode ser resolvido em tempo polinomial por uma Máquina de Turing Não Determinística pode ser “reduzido” ao problema de satisfação de uma sentença, que testa se uma fórmula *booleana* é satisfeita.

#### 2.3.1.1 Consequências da NP Completude

Como constata (Cormen, 2009), se qualquer problema NP Completo puder ser resolvido em tempo polinomial, então qualquer problema em NP tem um algoritmo em tempo polinomial, ou seja, . Equivalentemente, se algum problema NP não puder ser resolvido em tempo polinomial, então nenhum problema NP Completo pode ser resolvido em tempo polinomial.

#### 2.3.1.2 Exemplos de problemas NP Completos em grafos

Aqui são exemplificados, sem prova, alguns problemas famosos na literatura e que são conhecidos por serem problema de complexidade NP Completos. Todos os problemas e suas descrições foram retirados de (Cormen, 2009).

##### 2.3.1.2.1 O Problema do Caixeiro Viajante

Dado um grafo completo G com vértices, o objetivo é procurar o trajeto com menor custo possível, passando por todos os vértices apenas uma vez e retornando ao vértice de origem. O problema define um custo positivo ao visitar a aresta , com sendo vértices adjacentes no grafo. O custo total do trajeto é dado pela soma dos custos individuais de cada aresta visitada no trajeto. A definição formal é dada por:

“TSP = {⟨G, c, k⟩ : G = (V, E) é um grafo completo, é uma função de , e G tem um trajeto do tipo definido pelo problema do caixeiro viajante com custo mínimo de, no mínimo, k}.”

##### 2.3.1.2.2 O Problema da Cobertura de Nós

“Se G é um grafo não direcionado, uma cobertura de nós de G é um subconjunto de nós onde cada aresta de G possui um desses nós. O problema da Cobertura de Nós verifica se um dado grafo G contém uma cobertura de nós com tamanho específico. A definição formal é:

*VERTEX-COVER* = {⟨G, k⟩| G é um grafo não direcionado que possui uma cobertura de k-nós}.”

##### 2.3.1.2.3 O Problema Clique

“Um *clique* em um grafo não direcionado , onde é o conjunto de vértices e o conjunto de arestas, é o subconjunto de vértices, com cada par seu conectado por uma aresta em . Em outras palavras, um clique é um subgrafo completo de G. O tamanho de um clique é o número de vértices que ele contém. O problema *Clique* é o problema de otimização de encontrar um clique com tamanho máximo em um grafo. Como é um problema de decisão, a pergunta feita é se, dado um tamanho , existe um clique desse tamanho no grafo. A definição formal é:

*CLIQUE* = {⟨G, k⟩| G é um grafo contendo um clique de tamanho }.”

### 2.3.2 NP Hard

Há, ainda, outro subconjunto de problemas existentes na classe NP. Dada a definição de NP-Completo, se uma linguagem L satisfaz a condição 2, mas não necessariamente a condição 1, L é dita NP-Hard.

## 2.4 Hipótese P ≠ NP

A questão se é um dos maiores problemas ainda não resolvidos na ciência da computação teórica e matemática contemporânea (Sipser, M. 1997), além de ser um dos problemas selecionados pelo *Clay Mathematics Institute* (CMI) que valem um milhão de dólares (*Millennium Prize Problems*)[[1]](#footnote-0) caso a solução seja corretamente apresentada[[2]](#footnote-1). Segundo (Sipser, M. 1997), “se (uma linguagem) é NP-Completo e , então ”

Caso a igualdade seja confirmada, isso implicaria que qualquer problema **verificado** em tempo polinomial poderia ser **decidido** em tempo polinomial. Em outras palavras, se um problema pode ser verificado em tempo polinomial, então a sua solução pode ser encontrada também em tempo polinomial.

Embora ainda sem resposta, muitos teóricos da ciência da computação acreditam que , o que significaria que os problemas de complexidade NP-Completo são **intratáveis** (Sipser, M. 1997). (Cormen, 2009) diz que, talvez, a razão que leva os teóricos à acreditarem que vem da existência de problemas da classe NP-Completo e que, após anos de estudo, nenhum algoritmo em tempo polinomial foi descoberto para algum problema NP-Completo.

# 3 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Sipser, M. (1997). ***Introduction to the Theory of Computation*** (2a e 3a ed.). PWS;

[2] Harel, D. (1987). ***Algorithmics - The Spirit of Computing*** (3a ed.)**.** Addison-Wesley;

[3] Hopcroft, J.E., Motwani, R. and Ullman, J.D. (2007) ***Introduction to Automata Theory, Languages, and Computation***, Addison Wesley, Boston/San Francisco/New York;

[4] Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein. (2009) ***Introduction to Algorithms*** (3a ed.). The MIT Press;

[5] Stephen A. Cook. 1971. ***The complexity of theorem-proving procedures***. *In Proceedings of the third annual ACM symposium on Theory of computing* (*STOC '71*). Association for Computing Machinery, New York, NY, USA, 151–158;

[6] Michael R. Garey and David S. Johnson. 1979. ***Computers and Intractability; A Guide to the Theory of NP-Completeness***. W. H. Freeman &amp; Co., USA;

[7] Alan Cobham. **The intrinsic computational difficulty of functions**. *In Proceedings of the 1964 Congress for Logic, Methodology, and the Philosophy of Science*. NorthHolland, 1964.

[8] Jack Edmonds. **Paths, trees, and flowers**. Canadian Journal of Mathematics, 17:449–467, 1965.

1. https://www.claymath.org/millennium-problems/p-vs-np-problem [↑](#footnote-ref-0)
2. https://www.claymath.org/millennium-problems/rules-millennium-prizes [↑](#footnote-ref-1)